Planning Algorithms for
Multi-Robot Active Perception

Graeme Best

A thesis submitted in fulfillment
of the requirements of the degree of
Doctor of Philosophy

Australian Centre for Field Robotics
School of Aerospace, Mechanical and Mechatronic Engineering
The University of Sydney

Submitted May 2018; revised January 2019
Declaration

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the award of any other degree or diploma of the University or other institute of higher learning, except where due acknowledgement has been made in the text.

Graeme Best

January 2019
Abstract

Graeme Best

Doctor of Philosophy

The University of Sydney

May 2018

Planning Algorithms for Multi-Robot Active Perception

A fundamental task of robotic systems is to use on-board sensors and perception algorithms to understand high-level semantic properties of an environment. These semantic properties may include a map of the environment, the presence, pose and class of objects, the behaviour of other agents, or the parameters of a dynamic field. Observations are highly viewpoint dependent and, thus, the performance of perception algorithms can be greatly improved by planning the motion of the robots to obtain high-value observations. This motivates the problem of active perception, where the goal is to plan the observation viewpoints for a team of robots while considering both the motion constraints and the perception objectives of the task at hand. This fundamental problem is central to many robotics applications, including environmental monitoring, search and rescue, planetary exploration, and precision agriculture.

The core contribution of this thesis is a suite of planning algorithms for multi-robot active perception. These planning algorithms are designed to improve the system-level performance of multi-robot systems on many fronts. We aim to address various challenges of multi-robot active perception that have not been adequately addressed in existing work: online and anytime planning, optimising over a long time horizon, decentralised coordination, being robust to unreliable communication, predicting plans of other agents, and exploiting characteristics of specific perception models.

We first propose the decentralised Monte Carlo tree search (Dec-MCTS) algorithm as a generally-applicable, decentralised algorithm for multi-robot active perception. Dec-MCTS is a novel, decentralised variant of the widely-used Monte Carlo tree search algorithm, and also leverages ideas from variational methods to plan over probability distributions of action sequences. Dec-MCTS is anytime, is robust to communication loss, balances the exploration-exploitation trade-off when expanding the decision tree, and converges towards a plan that minimises KL-divergence to the optimal joint plan.
The usefulness of the algorithm is demonstrated through experiments for a general active perception formulation and a coordinated active object recognition task.

We then formulate and solve a more specific active perception problem where the objective is for the robots to maximally observe a discrete set of points of interest. These points of interest are observed by visiting continuous viewpoint regions. We propose an efficient solution algorithm for this formulation that exploits geometric properties using a special type of neural network called a self-organising map (SOM). The SOM algorithm is centralised, efficiently plans over continuous space, and has guaranteed polynomial runtime. As an example, a 3D point-cloud processing method is presented for generating problem instances. Experimental results show the problem-specific SOM algorithm is more efficient than Dec-MCTS for this formulation. The behaviour and performance of the algorithm is demonstrated with a range of simulated experiments in a variety of offline and online scenarios.

Finally, we consider the problem of mission monitoring, where a team of agents observe and monitor the progress of a robotic mission. This problem arises in marine robotics scenarios where underwater robots are monitored by a team of surface vessels, as well as a variety of other applications. In current practice, the motion of the monitoring agents is typically ignored or planned naively, leading to unsuccessful or ineffective monitoring. We instead consider the motion of the monitoring agents by formulating mission monitoring as an active perception problem. First, an optimal, polynomial-time algorithm is proposed for single-tracker mission monitoring. This algorithm, spatiotemporal optimal stopping, constructs a directed acyclic graph then finds the path that maximises expected monitoring time. The generated path is optimal with respect to probabilistic models for trajectory prediction and communication. Then, we propose an extended, decentralised algorithm for multi-tracker mission monitoring that is inspired by the general approach and analytical results of Dec-MCTS. Simulated experiments are performed with realistic prediction and communication models for marine robotics and pedestrian monitoring scenarios.

The proposed suite of planning algorithms is a significant contribution towards enabling multi-robot teams to perform coordinated perception tasks in a principled and effective manner. Our analytical and experimental results demonstrate theoretically-interesting and practically-relevant properties that support the use of the proposed approaches in practice. Our hope is that, by presenting a suite of algorithms designed for a variety of generic and task-specific formulations, we gain and share a fundamental understanding of both the core multi-robot active perception problem and its application to important tasks in robotics.
Acknowledgements

This thesis on non-myopic multi-agent coordination was made possible by the forward thinking and combined efforts of multiple collaborating agents. First and foremost, thanks to my supervisor A/Prof. Robert Fitch for providing guidance, encouragement, leadership, amazing ideas, freedom to pursue my own ideas, and your incredible ability to make dry theory sound exciting. Also thanks to Prof. Stefan Williams for your support and encouragement. Thanks to my prior research supervisors—Peyman Moghadam (CSIRO), Navinda Kottege (CSIRO), Lindsay Kleeman (Monash), and Stuart Anstee (DSTG)—who introduced me to the world of robotics research and strongly encouraged me to pursue a PhD.

A big thank you to my paper co-authors—Jan Faigl (Czech Tech.), Oliver Cliff, Timothy Patten, Wolfram Martens, Ramgopal Mettu (Tulane), Michael Forrai, Robert Pěnička (Czech Tech.), Shoudong Huang (UTS), and of course Robert Fitch—for the great ideas, constructive arguments, long phone calls, annotated PDFs, late nights, slow weekends, and tolerating my incoherent whiteboard scribbles—this thesis has certainly benefited from your efforts. Thanks also to the thesis examiners and the numerous anonymous paper reviewers for your efforts.

Thanks to everyone at the Australian Centre for Field Robotics, especially Oliver, Tim, Wolfram, Akash, Will, Jack, Phil, Giovanni, Chanyeol, Brian, and many others, for helping me maintain my questionable sanity with procrastination walks to the coffee machine. Special thanks to all members of our “collaborative decision-making reading group” for the invaluable discussions, and the dedication you put in to the presentations. Thanks to my teaching supervisor Jason Chan for your professionalism, showing me how to do things the right way, and for affording me a massive handicap in our games of chess. Also many thanks to the countless people around the world I’ve met and had discussions with at conferences and seminars. And to anyone else I may have missed . . .

Thanks to my Mum, Dad, sister Fiona, Gran, Grandpa, Nan, Grandad, Aunties, Uncles, cousins, and friends from Melbourne, for your support over my lifetime of being a student. It has been a challenge living far away from home, but I guess we’ll have to get used to this!

Grandad, I know you would have loved to read this thesis—your enthusiasm for turning ideas into words, and words into ideas, inspired this thesis.
The whole is greater than the sum of its parts
only if the whole is carefully planned.
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Declaration</td>
<td>i</td>
</tr>
<tr>
<td>Abstract</td>
<td>ii</td>
</tr>
<tr>
<td>Acknowledgements</td>
<td>iv</td>
</tr>
<tr>
<td>Contents</td>
<td>vi</td>
</tr>
<tr>
<td>List of Figures</td>
<td>xiii</td>
</tr>
<tr>
<td>List of Tables</td>
<td>xv</td>
</tr>
<tr>
<td>List of Algorithms</td>
<td>xvi</td>
</tr>
<tr>
<td>List of Theorems</td>
<td>xvii</td>
</tr>
<tr>
<td>Nomenclature</td>
<td>xviii</td>
</tr>
<tr>
<td>1. Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Multi-robot active perception</td>
<td>4</td>
</tr>
<tr>
<td>1.1.1 Active perception in the natural world</td>
<td>4</td>
</tr>
<tr>
<td>1.1.2 Robotic active perception</td>
<td>4</td>
</tr>
<tr>
<td>1.1.3 Active perception as a system</td>
<td>5</td>
</tr>
<tr>
<td>1.1.4 Multi-robot systems</td>
<td>6</td>
</tr>
<tr>
<td>1.2 Applications of active perception</td>
<td>7</td>
</tr>
<tr>
<td>1.2.1 Information gathering as an objective</td>
<td>8</td>
</tr>
<tr>
<td>1.2.2 Information gathering as a sub-task</td>
<td>9</td>
</tr>
<tr>
<td>1.2.3 Relevance of this thesis</td>
<td>9</td>
</tr>
<tr>
<td>1.3 Thesis Scope</td>
<td>10</td>
</tr>
<tr>
<td>1.3.1 Active perception planning module</td>
<td>10</td>
</tr>
</tbody>
</table>
## 1.4 Principal contributions

1.5 Thesis structure

## 2 Related work

### 2.1 Prediction models and objective functions

- **2.1.1 Discrete sets of properties**
- **2.1.2 Moving targets**
- **2.1.3 Continuous fields**

### 2.2 Informative path planning

- **2.2.1 Simplified problem formulations**
- **2.2.2 Single-robot planning**
- **2.2.3 Centralised multi-robot planning**
- **2.2.4 Decentralised multi-robot planning**

### 2.3 Planning algorithms

- **2.3.1 Generic sequential decision problems**
- **2.3.2 Optimal stopping**
- **2.3.3 Branch and bound tree search**
- **2.3.4 Monte Carlo tree search**
- **2.3.5 Self-organising maps**
- **2.3.6 Planning as inference**

### 2.4 Summary and limitations

## 3 Decentralised Monte Carlo tree search

### 3.1 Overview

- **3.1.1 Chapter outline**

### 3.2 Problem statement

### 3.3 Dec-MCTS

- **3.3.1 Algorithm overview**
- **3.3.2 Local utility function**
- **3.3.3 Monte Carlo tree search with discounted-UCB**
- **3.3.4 Decentralised product distribution optimisation**
- **3.3.5 Communication**
- **3.3.6 Online replanning**
A.6 Conclusions ......................................................... 255
# List of Figures

1.1 Real-world information gathering systems .................................. 2
1.2 Example active perception system ........................................... 6
1.3 Multi-robot information gathering on a farm ............................ 7
1.4 Illustration of the motivating active perception problem ............... 15
1.5 The multi-tracker mission monitoring problem .......................... 16

2.1 Illustration of branch and bound tree search ............................... 48
2.2 Exploration versus exploitation in MCTS ................................. 50
2.3 Self-organising map examples .............................................. 53

3.1 Overview of decentralised Monte Carlo tree search (Dec-MCTS) ..... 63
3.2 Overview of standard MCTS ............................................... 66
3.3 The generalised team orienteering problem ................................ 79
3.4 Experimental results of Dec-MCTS for the generalised team orienteer-
ing problem ................................................................. 82
3.5 Experiment setup for the point cloud dataset ............................ 83
3.6 Experimental results for Dec-MCTS in the active object recognition scenario ......................................................... 86
3.7 Demonstration that communication is required for successful coordi-
nation ................................................................. 91

4.1 Illustration of the motivating active perception problem from above . 94
4.2 Overview of the proposed self-organising map algorithm ............ 101
4.3 Random problem instance ................................................. 111
4.4 Simulation results for random environments under various scenarios and comparison methods ......................................................... 113
4.5 Convergence of self-organising map (SOM) algorithm ............. 114
4.6 Picture of the mobile robot with Velodyne laser scanner ............ 116
4.7 Illustration of the example sensor model ................................ 117
List of Figures

4.8 Validation of example sensor model ................................ 119
4.9 Four example scenarios and solution paths ......................... 121
4.10 Comparison between the SOM algorithm and Dec-MCTS ........ 124
4.11 Example adaptive plan for the online experiments ............... 127
4.12 Comparison of SOM planners with different planning horizons .... 127
4.13 Example paths for different online scenarios ...................... 128
4.14 Comparison of planning performance for different densities of explora-
              tion rewards .................................................. 129
4.15 Comparison of efficient replanning ................................ 130

5.1 Geometric interpretation of spatiotemporal optimal stopping ........ 135
5.2 Graph generation: stopping locations .............................. 143
5.3 Graph generation: temporal dimension ............................ 147
5.4 Illustration of the edge categories ................................ 149
5.5 Illustration of the sweep-plane algorithm ........................ 153
5.6 Deterministic AUV mission plan from a sea trial ................... 158
5.7 Deterministic vs. probabilistic planning ........................... 160
5.8 Monte Carlo simulation results ..................................... 161
5.9 Example predicted AUV trajectory .................................. 163
5.10 Underwater acoustic communication ............................... 163
5.11 The simulated office environment ................................ 170

6.1 Comparing planning with and without communication ............. 191
6.2 Comparison of convergence with different distribution sizes ....... 193
6.3 Comparison to Dec-MCTS ........................................ 194

A.1 Illustrated example of the trajectory prediction model ............. 241
A.2 Graphical model of the agent’s trajectory .......................... 243
A.3 Illustration of the probabilistic dynamics model ................... 244
A.4 Illustrated example of an agent moving through the environment ... 251
A.5 Illustrated example of the predicted future position ............... 252
A.6 The pedestrian dataset ............................................ 254
A.7 With and without the intention estimate posterior ................. 254
List of Tables

3.1 Comparison of different communication scenarios ............... 92
5.1 Simulation results for deterministic target trajectories ........... 159
5.2 Simulation results for AUV monitoring case study ............... 168
5.3 Simulation results for pedestrian monitoring case study ........... 173
List of Algorithms

3.1 Overview of Dec-MCTS ........................................... 64
3.2 Grow the search tree using Monte Carlo tree search ............. 66
3.3 Probability distribution optimisation ............................ 73

4.1 Self-organising map algorithm ................................ 102
4.2 Adaptation step of the SOM algorithm .......................... 104

5.1 Overview of the trajectory planner for the tracker ............... 141
5.2 Generate a spatiotemporal search graph ........................ 142
5.3 Edge weight and time calculations .............................. 149
5.4 Vertex set adjustments for the start condition .................. 152
5.5 Sweep-plane graph search: forward pass ....................... 154
5.6 Backtracking ...................................................... 154

6.1 Decentralised planning algorithm ............................... 182

A.1 Trajectory prediction algorithm ................................. 247
List of Theorems

1.1 Problem (General multi-robot active perception planning problem) . 11
3.1 Problem (Decentralised planning problem) . . . . . . . . . . . . . . . 61
3.1 Theorem (Convergence rate of D-UCT) . . . . . . . . . . . . . . . . . 76
3.1 Remark (Convergence in practice) . . . . . . . . . . . . . . . . . . . 77
3.1 Proposition (Convergence of PC) . . . . . . . . . . . . . . . . . . . . 77
4.1 Problem (Generalised orienteering problem) . . . . . . . . . . . . . 99
4.1 Lemma (Runtime per epoch) . . . . . . . . . . . . . . . . . . . . . . . 108
4.2 Lemma (Convergence guarantee) . . . . . . . . . . . . . . . . . . . . 108
4.1 Theorem (Runtime of SOM) . . . . . . . . . . . . . . . . . . . . . . . 109
4.1 Remark (Runtime dependence on $R$) . . . . . . . . . . . . . . . . . 109
4.2 Remark (Early convergence) . . . . . . . . . . . . . . . . . . . . . . . 110
5.1 Problem (Mission monitoring) . . . . . . . . . . . . . . . . . . . . . . 140
5.1 Condition (Triangle inequality) . . . . . . . . . . . . . . . . . . . . . 144
5.2 Condition (Convex hull is feasible) . . . . . . . . . . . . . . . . . . . 144
5.3 Condition (Monotonically decreasing observation value) . . . . . . . 144
5.4 Condition (Monotonically increasing travel time) . . . . . . . . . . . 144
5.1 Remark (Unbounded distributions) . . . . . . . . . . . . . . . . . . . 144
5.1 Lemma (Stopping in effective monitoring region) . . . . . . . . . . . 144
5.2 Lemma (Stopping in convex hull) . . . . . . . . . . . . . . . . . . . . 145
5.2 Remark (Underestimates due to pass-through vertices) . . . . . . . . 149
5.3 Lemma (Optimal arrival time for deterministic cases) . . . . . . . . . 150
5.4 Lemma (Optimality of graph search algorithm) . . . . . . . . . . . . 155
5.1 Theorem (Optimality of spatiotemporal optimal stopping) . . . . . . 155
5.2 Theorem (Polynomial runtime complexity) . . . . . . . . . . . . . . . 156
6.1 Problem (Decentralised mission monitoring) . . . . . . . . . . . . . . 181
# Nomenclature

## Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACFR</td>
<td>Australian Centre for Field Robotics</td>
</tr>
<tr>
<td>AUV</td>
<td>autonomous underwater vehicle</td>
</tr>
<tr>
<td>BnB</td>
<td>branch and bound</td>
</tr>
<tr>
<td>CAMP</td>
<td>communication-aware motion planning</td>
</tr>
<tr>
<td>Dec-MCTS</td>
<td>decentralised Monte Carlo tree search</td>
</tr>
<tr>
<td>Dec-POMDP</td>
<td>decentralised partially observable Markov decision process</td>
</tr>
<tr>
<td>D-UCB</td>
<td>discounted-UCB</td>
</tr>
<tr>
<td>D-UCT</td>
<td>discounted-UCT</td>
</tr>
<tr>
<td>GP</td>
<td>Gaussian process</td>
</tr>
<tr>
<td>GPIS</td>
<td>Gaussian process implicit surface</td>
</tr>
<tr>
<td>GTSP</td>
<td>generalised travelling salesman problem</td>
</tr>
<tr>
<td>MAB</td>
<td>multi-armed bandit</td>
</tr>
<tr>
<td>MCTS</td>
<td>Monte Carlo tree search</td>
</tr>
<tr>
<td>MDP</td>
<td>Markov decision process</td>
</tr>
<tr>
<td>OP</td>
<td>orienteering problem</td>
</tr>
<tr>
<td>OPN</td>
<td>orienteering problem with neighbourhoods</td>
</tr>
<tr>
<td>PC</td>
<td>probability collectives</td>
</tr>
<tr>
<td>POMDP</td>
<td>partially observable Markov decision process</td>
</tr>
<tr>
<td>PRM</td>
<td>probabilistic roadmap</td>
</tr>
<tr>
<td>ROS</td>
<td>robot operating system</td>
</tr>
<tr>
<td>RRT</td>
<td>rapidly-exploring random tree</td>
</tr>
<tr>
<td>SLAM</td>
<td>simultaneous localisation and mapping</td>
</tr>
<tr>
<td>SOM</td>
<td>self-organising map</td>
</tr>
<tr>
<td>TSP</td>
<td>travelling salesman problem</td>
</tr>
<tr>
<td>UCB</td>
<td>upper confidence bound</td>
</tr>
<tr>
<td>UCT</td>
<td>UCB applied to trees</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Information gathering is an important family of problems in robotics that plays a primary role in a wide variety of tasks. During these tasks, mobile robots use their sensors and perception algorithms to understand their surrounding environment. The properties of the environment being estimated can vary greatly from application to application. Consider the three example real-world robotic systems illustrated in Figure 1.1. On the left, a robot drives through an orchard to measure properties of fruit such as their quantity and ripeness (Hung et al., 2015). In the middle, an autonomous underwater vehicle (AUV) is deployed in the ocean to create a map of the health of a coral reef (Steinberg et al., 2011). On the right, an unmanned aerial vehicle senses electromagnetic pulses to track the movements of radio-tagged wildlife (Cliff et al., 2018). These estimated properties represent information of immediate practical interest to the end users for timely decision making: farmers can coordinate harvesting schedules, environmentalists can target restoration efforts, and ecologists can respond to threats to endangered species. Beyond these applications, robotic information gathering systems have the potential to improve operations in a diverse range of fields including defence, security, emergency response, mining, infrastructure maintenance, transport, health care, domestic automation, and planetary exploration.

The motion of the robots plays a key role in information gathering as observations taken at different viewpoints or times offer complementary information. However, the importance of motion is often overlooked, as most robots are tasked to follow naive, preplanned or teleoperated paths that do not directly take into account the mission
Robots moving through and sensing an environment

Information gathered about each environment

Figure 1.1 – Example real-world information gathering robotic systems. (left) Modelling fruit trees (Hung et al., 2015). (middle) Mapping the health of a coral reef (Steinberg et al., 2011). (right) Tracking the movements of wildlife (Cliff et al., 2018). Figures courtesy of ACFR.

objectives. This approach can result in inefficient motion and a failure to collect the desired information. Instead, the concept of active perception jointly considers motion and perception as a unified task. The idea is to plan the motion of the robots in an online manner with the aim of collecting better data and, ultimately, improving the system performance at the perception task.

While this concept of active perception has been studied for over three decades (Bajcsy, 1988; Bajcsy et al., 2018), there are many unsolved challenges that have so far prevented the wide-spread use of this methodology in practice. Many of these challenges are algorithmic—we are yet to determine the best way to design computer programs for processing sensor data and planning intelligent decisions. On the other hand, challenges relating to the physical hardware of robotic systems have held back our ability to deploy these ideas in the real world. Part of the challenge is also societal, though many industries are now beginning to recognise and invite the important role robotics can play in improving their operations. With the ever-increasing access to faster onboard computation, cheaper sensors and reliable robotics hardware,
we feel the timing is now right to develop and refine sophisticated active perception algorithms designed for deployment on robotic information gathering systems.

One way to improve the capabilities of robotic systems is through the use of multi-robot systems. The deployment of thousands of robots in Amazon warehouses (D’Andrea, 2012) is an excellent realisation of the potential of multi-robot systems. In the context of active perception, teams of robots can efficiently achieve an improved set of viewpoints over a single robot, where concurrency allows for scaling up the number of observations in time and space. Using teams of robots also improves the robustness and reliability of the system. However, scaling up to multiple robots also introduces additional challenges, particular the need to coordinate the motions of the robots. While the Amazon warehouse example is an incredibly successful demonstration of multi-robot coordination, their highly-controlled environment does not exhibit many of the difficulties experienced in other contexts, such as planning with an unknown environment, uncertain objectives, unpredictable locomotion, noisy localisation, energy constraints, and unreliable communication.

The objective of this thesis is to develop principled and practical planning algorithms that are designed to improve our understanding of and address challenges of multi-robot active perception systems. The algorithms we propose have several desirable properties: online and anytime planning, efficiently search over a long time-horizon, perform multi-robot coordination, are robust to unreliable communication, predict and respond to plans of other agents, and are generally applicable while also allow exploiting characteristics of specific perception tasks. Our analytical and experimental results present compelling evidence that our proposed algorithms are a viable solution for multi-robot active perception.

This chapter motivates and introduces the concepts presented in the remainder of this thesis. First we provide a broad definition of multi-robot active perception and highlight the key components. Then we highlight real-world systems where this methodology is applicable. Then we define the main planning problem and subproblems addressed by this thesis. Finally, we provide an overview of the main algorithms proposed as the contributions of this thesis.
1.1 Multi-robot active perception

1.1.1 Active perception in the natural world

Animals, including humans, learn essential information about the world around us. We do this not by simply processing input stimuli in a passive manner, but by actively selecting what, when and how to perceive (Gibson, 1966). This is achieved through actions, such as looking, touching and listening, in order to find the information we are seeking. For example, seals track fish by moving along hydrodynamic trails sensed by their whiskers (Dehnhardt et al., 2001). Birds control their body posture to adjust their visual attention when simultaneously foraging for food and scanning for predators (Fernández-Juricic et al., 2004). Bats and dolphins control transmitted sound pulses and reshape their ears to perform echolocation (Thomas et al., 2004).

Many species perform this information gathering in teams, which allows them to achieve outcomes well beyond what individual animals could do alone. For example, ants forage for food cooperatively by spreading their search over different areas of the environment (Traniello, 1989). Communication is essential to the success of this coordination as it is used to exchange information about food location and recruit additional foragers to promising areas.

While in this thesis we do not attempt to directly replicate animal cognitive processes, these examples found in the natural world serve as motivation for applying this general methodology to robotics.

1.1.2 Robotic active perception

Robots are agents that are embodied in the physical world, and are typically required to learn information about their environment, either as a sub-task for other problems or as an objective in its own right. Robotic active perception is the task of controlling actuators in order to improve the performance at these information gathering tasks, while making efficient use of available resources.

The type of actions being executed varies depending on the type of actuators available; for example, robots that have locomotion capabilities can move to new viewpoints,
robots that have arms can physically manipulate the scene, and robots that have adjustable cameras can perceive different parts of the scene by changing the focus of the lens. In this thesis we primarily consider the first type of actions, but the methods we present could be adapted for other types of actions.

Similarly, the type of information gathering task varies depending on the application. Typically, a task is described as a variant of a standard problem, such as coverage, exploration, mapping, search, tracking, or classification. Specific definitions of these problems vary, but they each define a type of information measure for the value of actions that describes the objectives of the perception task at hand.

The actions must be selected while also considering the costs of actions that represent usage of a resource. The resource may be energy, which is likely to impose a hard constraint on the sum of action costs. The resource may also be time, which may be a hard or soft constraint. Ultimately, there exists a trade-off between the use of resources and the value of information collected, which must be addressed when selecting actions.

### 1.1.3 Active perception as a system

No matter what type of robot actions or task is being considered, the active perception methodology should be thought of as a system of several essential components that must work together. A standard single-robot system developed by Patten et al. (2016) is illustrated in Figure 1.2, which contains the following components: a planning module uses the current belief of the world to select the next observation locations; a navigation module is responsible for driving the robot to the next chosen locations; and observation, processing and update modules process sensor data to update the belief of the world. This new belief then feeds back in to the planning module to replan the observation locations, and this process continues in a cycle. The training data and initial belief represent some form of prior information about the world that is used to inform the planning and observation processing.

The planning module is a vital component of these systems and this module is the primary focus of this thesis. However, we stress that we do not consider planning in isolation; the proposed planners are specifically designed to optimise with respect to
1.1 Multi-robot active perception

Figure 1.2 – An example active perception system. In this thesis we focus on the planning module while considering the overall system performance. Diagram courtesy of Patten et al. (2016).

predictions of the perception value of actions. There are several desirable properties a planner should exhibit. A planner should consider the problem as a sequential decision process and efficiently search over a long time horizon; conversely, greedy decision-making can significantly compromise the ability to make informative observations later. A planner should be suitable for real-time applications, meaning that it should be online, anytime, and there should exist a trade-off between computation time and performance. In this thesis we propose several planning algorithms with these desirable properties.

1.1.4 Multi-robot systems

Scaling the system up for multi-robot systems presents additional opportunities and challenges. Having more robots provides an opportunity to make complementary observations from more viewpoints and with a wider spatial distribution. This enables collecting more information in a shorter time frame. Having more robots also improves the robustness of the system as the system should still perform reasonably well if some of the robots fail.

One of the main difficulties, though, is that the perception objective function is defined for the entire team, and therefore the actions for each robot need to be selected while considered the actions of the entire team. Single-robot planning algorithms cannot be directly applied, and so specialised multi-robot algorithms must be used instead. The
main challenge is to develop multi-robot algorithms that perform efficiently as more robots are added to the team since the size of the search space scales exponentially in the number of robots.

Depending on the application, the multi-robot algorithms may be centralised or decentralised, and we propose examples of both of these in this thesis. Decentralised planners in particular should use communication to improve coordination by exchanging information such as plans between robots, but should also be robust to unpredictable communication limits and failures.

### 1.2 Applications of active perception

We envision teams of robots, such as that in Figure 1.3, being deployed to adaptively use available resources (e.g., energy, time, communication, and prior knowledge) to successfully collect information for fulfilling the requirements of applications. This objective of gathering information may be the end goal for the robotic system, or it may be a necessary subtask of a broader problem. In this section we highlight a large variety of example applications for these tasks, in addition to the examples in Figure 1.1.
1.2.1 Information gathering as an objective

In many contexts, human decision-makers require timely information about their environment of interest. Historically, this information has usually been collected manually by humans, which in many cases can be labour intensive and require specialised knowledge. The first steps of automating this process involved static sensor networks (Akyildiz et al., 2002). Sensor networks provide a persistent stream of data about fixed locations in an environment, but typically require very large networks to achieve the desired spatial coverage. Robots can overcome this issue as small teams of mobile robots with embedded sensors can achieve the same spatial coverage.

We are beginning to see scientists perform environmental monitoring tasks with robots in the natural world (Dunbabin and Marques, 2012) such as for monitoring algae blooms (Das et al., 2015; Hitz et al., 2017), carp (Tokekar et al., 2010), clouds (Reymann et al., 2018), caves (Tabib et al., 2016), river geometry (Nuske et al., 2015), invasive weeds (Clements et al., 2014), and fires (Merino et al., 2012). In many cases, robots provide information to scientists in contexts that are not otherwise possible, e.g., on Mars (Bajracharya et al., 2008), in volcanoes (Astuti et al., 2008), and at the bottom of our oceans (Whitcomb et al., 2010). This information improves scientists’ knowledge of the world and their influence on decision making (Harding, 1998).

Information is also key in many industrial contexts. In agriculture, information allows farmers to make predictions and plan when to harvest, apply fertiliser, control greenhouses, schedule refinery usage, and plant crops in future years (Wang et al., 2006). Robots have been used for the inspection/monitoring of nuclear power plants (Nagatani et al., 2013), ship hulls (Hollinger et al., 2013), work-sites such as farms (Maturana et al., 2017), heat conduction of energy sources (Cunningham-Nelson et al., 2015), sensor networks (Lambrou and Panayiotou, 2013), and audio condition-monitoring of machinery (Even et al., 2017). In defence scenarios, robots are used to gather information about unexploded ordnances (Acar et al., 2003) and potential threats (Robin and Lacroix, 2015) to ensure safety of military personnel and assets.
1.2 Applications of active perception

1.2.2 Information gathering as a sub-task

In many applications, the gathered information is not only of interest to humans, but is also essential for the robot to complete other tasks. For navigation, information gathered is used for collision avoidance (Bonin-Font et al., 2008; Nuske et al., 2015; MacDonald and Smith, 2018), safe manoeuvring (Peynot et al., 2014), and landing unmanned aerial vehicles (Scherer et al., 2012). Efficient gliding requires simultaneously learning the wind field (Lawrance and Sukkarieh, 2011). Learning models of objects in the world is important for grasping and manipulation tasks (Schwarz et al., 2018; Kahn et al., 2015). Robots are used in coral reefs to identify and track invasive starfish, then respond by injecting biological agents (Dayoub et al., 2015). In search and rescue scenarios, once a target is found a robot can then deploy rescue equipment (Roberts et al., 2016). In asset guarding missions, potential threats are identified and tracked ready for possible intervention (Wolf et al., 2017). For agriculture, information about crops is used for spot-spraying of fertiliser and pesticides (Slaughter et al., 2008). Learnt maps of infrastructure are used by maintenance robots, such as for bridge maintenance (Paul et al., 2011).

Robots will soon be prevalent in human-centric environments, where accurate information is crucial to both task completion and for safe interaction with humans. Robotic vacuum cleaners are being used to clean our houses; early generations of these robots executed naive paths, while the latest models learn maps of the environment to improve cleaning efficiency (Moloney and Suarez, 2015). Robots could be used for providing physical and healthcare assistance to the elderly (Robinson et al., 2014). Robots will also need to navigate through human crowds, such as for surveillance, crowd control and transport (Trautman et al., 2015). Information gathering is also a necessary component of surgical robotics (Guru et al., 2015).

1.2.3 Relevance of this thesis

This thesis does not target any specific one of these applications; instead, we are interested in studying, understanding, and offering solutions to the fundamental planning task that underpins all of these applications. We stress that most of the applications
listed above have so far only demonstrated success in academic settings; practitioners are mostly still using passive perception methods, significant human input, or no robotics at all. We hope that the research contributions presented in this thesis are a step towards realising these applications in the wider community.

1.3 Thesis Scope

1.3.1 Active perception planning module

We are interested in improving the performance of an entire active perception system, such as the system illustrated in Figure 1.2. While all modules contribute to system performance, in this thesis we particularly focus on achieving this performance improvement by developing new planning modules. We emphasise that these planning modules are designed while considering all modules of the system. Specifically, the planners must consider the actions available to be chosen by the robots, the navigation travel-distance budget, the current belief of the world, predicted observations, and how these observations coincide with the perception objectives of the mission.

1.3.2 General problem statement

We propose several algorithms that are suitable for the planning module under different assumptions about the problem formulation. These algorithms also provide different algorithmic properties and output characteristics that are suitable for different applications. However, broadly, the algorithms are developed to solve similar active perception problems. We state a general problem formulation that encompasses the main problems and general assumptions considered in this thesis as follows.

We consider a team of $R$ robots $\{1, 2, \ldots, R\}$, where each robot $r$ performs a sequence of actions $x_r = (x_{r1}, x_{r2}, \ldots)$. Each action $x_{rj}$ has an associated cost $c_{rj}$ and each robot has a cost budget $B_r$ such that the sum of the costs must be less than the budget, i.e., $\sum_{x_{rj} \in x_r} c_{rj} \leq B_r$. This cost budget may be an energy or time constraint defined by the application, or it may be used to enforce a planning horizon. The feasible set of actions and associated costs at each step $j$ are a function of the previous actions
$(x_1^r, x_2^r, \ldots, x_{j-1}^r)$. Thus, there is a predefined set of feasible action sequences for each robot $X^r$. Further, we denote $x$ as the set of action sequences for all robots $x := \{x^1, x^2, \ldots, x^R\}$ and $x^{(r)}$ as the set of action sequences for all robots except robot $r$, i.e., $x^{(r)} := x \setminus x^r$.

The action sequences, while stated broadly, typically represent the robots moving around the workspace. As an example, we often define each action $x_j^r$ as the traversal of an edge in a probabilistic roadmap (PRM) (Kavraki et al., 1996) representation of the environment, and an action sequence $x^r$ is a feasible path through the PRM. Alternatively, the formulation in Chapter 4 does not require an underlying discrete action set, but instead the approach searches for sequences of discrete observation locations over a continuous space.

There also exists a global objective function $g(x)$ that encodes the objectives of the perception task at hand. The function $g$ is a function of all of the robots’ actions $x$, and may be a probabilistic or deterministic function. We broadly assume that all robots know and agree on this function, which would typically require some form of decentralised data fusion.

Given this formulation for the robot actions and perception objective, the general problem solved in this thesis is stated as follows.

**Problem 1.1** (General multi-robot active perception planning problem). Plan the action sequences $x$ for the team of robots such that the cost-budget constraints are satisfied and the global objective function $g(x)$ is maximised.

This problem formulates a trade-off between the resource usage of actions $\sum c_j^r$ and the perception objective $g$. There are several possible ways of formulating this multi-objective optimisation trade-off; we believe the most natural and useful way is to formulate the resource usage as hard constraints and the perception objective as the optimisation objective to be maximised, as presented in Problem 1.1.

Here, $g$ is stated as a general function of the robot actions, which can broadly define any coordination problem. In active perception problems, $g$ encodes some form of information gain relevant to the perception task at hand. This encoding can be formulated in many different ways depending on the perception task and what assumptions
are willing to be made. Throughout this thesis, each considered sub-problem makes different assumptions about $g(x)$, which is exploited by their respective algorithms. The problem considered in Chapter 3 makes no further assumptions about $g$ and solves this problem in the general case for an online, decentralised setting. Chapters 4, 5 and 6 consider more specific problem formulations, which are outlined below in Section 1.3.3, and propose efficient problem-specific solutions. We also discuss several typical formulations for $g$ found in the literature in Section 2.1.

### 1.3.3 Problem settings

In this section we introduce and motivate the sub-problems of Problem 1.1 considered in this thesis and key characteristics of the problem settings.

**Online planning and adaptivity**

*Online* algorithms produce sequences of decisions based on historical data while also considering the impact of these decision on the *final* quality of overall performance (Karp, 1992; Borodin and El-Yaniv, 1998). This type of decision making is relevant to settings where there is uncertainty as to what will happen in the future, such as in memory caching and network routing. Robotics generally, including the problems considered in this thesis, also falls into this category as planning decisions need to be made with respect to uncertain estimates of the environment, observations, team behaviour, etc.

One way to address online problem settings is to develop *adaptive* algorithms, such that each planned action is a function of the most recent observation (Hollinger et al., 2013). Solutions of this form are represented as a policy tree that branches on observations. This formulation is a general representation of sequential decision processes and, if solved optimally, guarantees the best possible performance. However, these benefits come with the cost of requiring significant computational resources, which may not be available onboard robots. Additionally it requires specifying the set of all possible observations in advance, which is often unknown or prohibitively large. Dec-POMDP solutions typically take this approach (Oliehoek and Amato, 2016).
In contrast, non-adaptive algorithms plan a fixed sequence of actions that are intended to be executed no matter which observations are received. This simplified solution typically enables using more efficient solution algorithms. It is common to then replan online whenever new observations are received that result in changes to the belief of the world (i.e., $g$ changes). The performance gap between an optimal adaptive algorithm and an optimal non-adaptive algorithm is referred to as the adaptivity gap; in certain classes of information gathering problems, this adaptivity gap is bounded (Hollinger et al., 2013).

In this thesis we propose non-adaptive algorithms with replanning when new observations are received. We feel this is the most suitable approach for these problem settings since our intention is for the algorithms to be computed onboard the robots; however, we acknowledge this debate of adaptive versus non-adaptive algorithms is a contentious issue in the planning community.

**Decentralised and centralised coordination**

In Chapters 3 and 6, we address formulations of Problem 1.1 in decentralised settings. We broadly define these decentralised settings as follows. We assume each robot $r$ knows the global objective function $g(x)$, but does not know the actions $x^{(r)}$ selected by the other robots. We assume that robots can communicate during planning-time to improve coordination. The communication channel may be unpredictable and intermittent, and all communication is asynchronous. Therefore, each robot will plan based on the information it has available locally. Bandwidth may be constrained and therefore message sizes should remain small, even as the plans grow. Although we do not consider explicitly planning to maintain communication connectivity, this may be encoded in the objective function $g(x)$ if a reliable communication model is available.

In Chapter 4, we address a formulation of Problem 1.1 in a centralised setting. This setting assumes there is a single server that plans on behalf of all robots, which is a reasonable assumption in some contexts. The server has the advantage of having full control over what plan $x^r$ all robots will execute, meaning it can plan over the space of joint action sequences $x$ without requiring communication.

In Chapter 5, we address a two-robot problem where one of the robots (the tracker)
plans with respect to the other robot (the \textit{target}). We assume that the target has already created its plan, and then present a single-robot algorithm for the tracker that considers probabilistic predictions of how the target may execute its plan. This type of planning is referred to as \textit{decoupled} planning, and is often useful in applications where there is a hierarchy of importance for individual tasks of the robots, as in Chapter 5. In Chapter 6, this particular problem is generalised to a multi-tracker setting and solved in a decentralised manner.

\textbf{Informative viewpoint regions}

The problem we formulate and address in Chapter 4 aims to capture the viewpoint-dependency of observation rewards in an efficient manner. Most approaches for active perception, including one of the examples provided in Chapter 3, typically estimate the value of visiting candidate viewpoints by simulating predicted observations (van Hoof et al., 2014; Wu et al., 2015; Patten et al., 2016). For complex sensor models, these predictions can be computationally expensive, which therefore restricts the capabilities of planning algorithms.

Instead, in Chapter 4 we formulate perception tasks by extracting informative features of the scene to be observed. This is defined using an inverse sensor model that generates a discrete set of overlapping continuous viewpoint regions, with associated rewards, where each feature can be observed. This problem can be thought of as a new generalisation of the orienteering problem (Vansteenwegen et al., 2011; Gunawan et al., 2016). Figure 1.4 illustrates an example outdoor scene that has been decomposed into a problem of this form. One advantage of this formulation is that it allows us to develop efficient non-myopic planners that exploit characteristics of this formulation to efficiently plan over continuous space.

\textbf{Mission monitoring}

In Chapters 5 and 6 we define and address the mission monitoring problem. Mission monitoring is a supervisory problem where one or more robots or manually driven vehicles track the progress of an autonomous mobile robot or other agent in performing a pre-planned task. There are many examples of such tasks that require monitoring,
1.3 Thesis Scope

Figure 1.4 – Chapter 4 active perception problem formulation. Illustration of the motivating active perception problem. Each object segment (point clouds) is observed by visiting the viewpoint regions (circle segments). Grey cylinders represent positions of two robots. The currently visited viewpoint regions are drawn in bold. Black lines represent the path plans. The aim is to collectively maximise the weighted sum of viewpoint regions visited by the robots.

including undersea surveys, environmental monitoring, autonomous farming and planetary exploration. Monitoring allows for rapid response to failures and to important information that the robot may discover during the progress of its mission (German et al., 2012; Hagen et al., 2008; Yilmaz et al., 2008; Khatib et al., 2016). Additionally, the monitoring vehicle may augment mission capabilities by providing observations from external viewpoints, such as for accurate localisation and navigation (Fallon et al., 2010; Heppner et al., 2013; Klotz et al., 2015; Saska et al., 2014; Kottege and Zimmer, 2011) or online sensor calibration (Bongiorno et al., 2013). The motion of the robot is typically represented by a mission plan, which may be defined probabilistically to take into account uncertain vehicle dynamics, environment models and mission objectives (Karydis et al., 2015; Chiang et al., 2014; Aoude et al., 2013).

We consider the case where the monitor vehicles must remain stationary in order to observe or communicate with the robot, which is motivated by marine robotics practices where communication equipment is most efficient while stationary. This problem is important because it is an essential part of employing outdoor robots for certain real-world tasks, such as various underwater missions (German et al., 2012), that
depend on timely transmission of sensor observations or system faults. It is also interesting in broader contexts because it applies to systems that must stop periodically to conserve energy (Brockers et al., 2011), to provide imagery taken from stationary viewpoints (Naseer et al., 2013), and for acoustically covert surveillance (Dunbabin and Tews, 2012).

A geometric interpretation of mission monitoring for the case where there is multiple monitoring vehicles problem is shown in Figure 1.5. The optimisation problem is for the monitor vehicles to decide where to stop (centre of cylinders), and when to move to the next observation location (height of cylinders), in order to best observe the probabilistic prediction model (blue lines). In Chapter 5 we formulate and address the case where there is one monitoring vehicle, and in Chapter 6 we generalise this problem for cases where there are multiple monitoring vehicles that coordinate. We propose algorithms that exploit geometric characteristics of these problems.
1.4 Principal contributions

The main contribution of this thesis is a suite of planning algorithms suitable for multi-robot active perception. Each of the proposed algorithms solves a particular active perception formulation: (Chapter 3) a generic decentralised planning problem, (Chapter 4) a generalisation of the orienteering problem, (Chapter 5) single-tracker mission monitoring, and (Chapter 6) multi-tracker mission monitoring. We enumerate specific contributions as follows:

1. Several new problem formulations for the planning component of multi-robot active perception systems. These include both generic and problem-specific formulations, which are designed to both reflect the real world and be in a suitable form for our proposed solutions.

2. A novel generic planning algorithm, decentralised Monte Carlo tree search (Dec-MCTS), for decentralised multi-robot planning. This algorithm is a powerful new method of decentralised coordination for any objective function defined over the robot action sequences. The algorithm has several useful properties, such as being anytime, online, non-myopic, balances exploration and exploitation of the search space, robust to unreliable communication, and allows incorporating prior knowledge. This is the first decentralised variant of Monte Carlo tree search.

3. A self-organising map (SOM) algorithm designed for an active perception problem formulated as a generalisation of the orienteering problem. This is a centralised algorithm that exploits geometric properties of the environment using a special type of neural network. It is a heuristic algorithm that efficiently searches over continuous space and a long time-horizon, and has guaranteed polynomial runtime.

4. The spatiotemporal optimal stopping algorithm for single-tracker mission monitoring. This algorithm features a novel spatiotemporal graph construction and a longest-path graph search. The algorithm plans with respect to probabilistic motion prediction and communication models. It has guaranteed optimality and polynomial runtime. This algorithm significantly outperforms
1.4 Principal contributions

the only other known solution for this problem, and provides new theoretical guarantees.

5. A decentralised algorithm for **multi-tracker mission monitoring**. This algorithm combines and extends elements of Dec-MCTS and spatiotemporal optimal stopping to form a novel decentralised planner. It inherits most of the useful properties of Dec-MCTS with stronger convergence properties. This is the first solution to the multi-tracker mission monitoring problem.

6. **Analytical results** are presented for all proposed algorithms. For Dec-MCTS, we summarise our previous results from Best et al. (2018a) that show convergence properties for the key algorithmic components, and discuss the implications of these results. For SOM, we show the algorithm has polynomial runtime and converges. For single-tracker mission monitoring, we show the algorithm has polynomial runtime and is optimal. For multi-tracker mission monitoring, we discuss runtime and convergence properties by extending the results from Dec-MCTS and spatiotemporal optimal stopping. These results collectively describe a new understanding of active perception and associated algorithms.

7. **Empirical results** for simulated experiments with all proposed algorithms. These results validate the theoretical claims, and evaluate the behaviour of the algorithms under various scenarios, such as object classification and marine robotics operations. The experiments reflect the real world by using several real-world outdoor datasets. Several perception and prediction models are formulated and/or implemented for these experiments.

The work presented in this thesis has partially appeared in our previous publications. More specifically, most of Chapter 3 is based on Best et al. (2018a) (which extends Best et al. (2016a)) and Section 3.7 is based on Best et al. (2018c); Chapter 4 is based on Best et al. (2018b) (which extends Best et al. (2016b), with related formulations in Best and Fitch (2016); Faigl et al. (2016)); Chapter 5 is based on Best et al. (2017) (preliminary results in Best et al. (2015); Best and Anstee (2014)) with experiments that use a model proposed in Best and Fitch (2015) (provided in Appendix A); and Chapter 6 is based on Best et al. (2018d). Graeme Best is the primary contribut-
ing author in all of these publications (except Faigl et al. (2016)). All references mentioned here have been published.

### 1.5 Thesis structure

The remainder of this thesis is organised as follows.

**Chapter 2** surveys related work in the fields of active perception, planning algorithms and multi-robot coordination.

**Chapter 3** presents decentralised Monte Carlo tree search (Dec-MCTS) as a solution algorithm for solving Problem 1.1 in a principled, decentralised and online manner.

**Chapter 4** presents a new active perception formulation as a generalisation of the orienteering problem, and a self-organising map (SOM) solution algorithm as an efficient, centralised heuristic for this formulation.

**Chapter 5** presents an active perception formulation of the mission monitoring problem and proposes the spatiotemporal optimal stopping algorithm that provides strong performance guarantees.

**Chapter 6** generalises the mission monitoring problem for larger teams of robots and proposes a decentralised solution algorithm motivated by spatiotemporal optimal stopping and Dec-MCTS.

**Chapter 7** concludes the thesis and discusses important avenues for future work.

**Appendix A** presents a new intention inference model for trajectory prediction that is used within the experiments of Chapter 5.
Chapter 2

Related work

The general methodology of active perception has been studied since the 1980s, beginning with the seminal work of Bajcsy (1988). The research field has continued to grow due to increasing interest from industry, and development in complementary research fields such as computer vision and hardware design. Several important survey papers have been presented over the years that highlight the growing interest in this field (Bajcsy, 1988; Chen et al., 2011; Bajcsy et al., 2018).

As argued in Section 1.1, active perception is best thought of as a system of several interconnected components. However, most attention in the field has focussed solely on the modules relating to sensing and perception, with only relatively simple planners being used. Similarly, in the robot planning community, most attention has focussed on lower-level tasks, such as navigating to a goal (LaValle and Kuffner, 2001), with less attention given to developing planners that require complex interaction with rich perception models. This trend is even more apparent in the multi-robot planning community. Outside of the robotics community, new algorithms are being developed for planning in many other contexts, especially in game theory, where there is broad interest in decision making for complex tasks.

This thesis borrows and extends ideas from these fields—active perception, path planning, and general planning algorithms—to develop new algorithms for the planning module of multi-robot active perception systems. This chapter reviews related literature in these three broad fields and draws relationships between them in order to set
the context for our new contributions. In Section 2.1, we begin by reviewing common perception prediction models and objective functions in the context of active perception; these models represent many of the applications surveyed earlier in Section 1.2. In Section 2.2, we review algorithms for informative path planning in both single- and multi-robot settings. In Section 2.3, we review generic planning algorithms that are particularly relevant to the ideas presented in this thesis. Finally, in Section 2.4 we summarise the chapter and emphasise the contributions of this thesis in the context of the literature.

2.1 Prediction models and objective functions

Passive perception methods address the problem of processing input sensor data to estimate properties of the world; many textbooks have been written about this broad field, e.g., Szeliski (2010). The perception modules of active perception methods borrow many of the same ideas, but also require the ability to predict and evaluate expected observations; this section reviews these concepts. We categorise methods for these concepts as either continuous models (such as Gaussian processes (GPs)), discrete models (such as semantic maps), or trajectories (such as moving targets).

The planning module critically relies on these perception predictions since optimisation is performed with respect to these models. The algorithms presented in this thesis could potentially be used to optimise paths with respect to any of the objective functions discussed here; we formalise and demonstrate several examples in our experimental sections.

2.1.1 Discrete sets of properties

In many applications, it is appropriate to represent the world as a discrete set of properties. This is common in scenarios where it is desired to estimate properties of a set of objects, a map, or targets. In most cases, this involves maintaining a joint probability distribution over a set of random variables; each random variable may be represented by different types of distributions and may or may not be correlated. Viewpoint evaluation is typically performed by predicting observations and evaluating
the effect this observation has on the belief of the world using an information-theoretic measure. We describe several examples as follows, separating them into different problem scenarios.

**Exploration**

Exploration problems involve observing all areas of an unknown environment as fast as possible. One way to formulate this problem is to describe the world as a grid of cells, and label each cell as open, unknown or occupied (Elfes, 1989; Yamauchi, 1998; Zlot et al., 2002; Vincent et al., 2008; Nieto-Granda et al., 2014); these labels may be deterministic or probabilistic. Observations from a viewpoint can be predicted using ray-tracing and evaluated by counting the decrease in unknown cells. A typical approach is to guide the robot towards “frontiers” of unknown cells (Kahn et al., 2015). It is also possible to incorporate knowledge about the type of maps that the environment may consists of, which enables more-accurate online predictions and therefore improved planning performance (Choudhury et al., 2017; Smith and Hollinger, 2018; Caley et al., 2016).

**Active SLAM**

Simultaneous localisation and mapping (SLAM) (Durrant-Whyte and Bailey, 2006) is the problem of jointly constructing a map of the environment and estimating the robot’s location within it. While there is a plethora of work in developing estimation techniques for SLAM, there has been undeservedly less attention devoted to the problem of path planning to improve estimation accuracy, i.e., active SLAM. The problem of active SLAM is similar to exploration problems in that the aim is to discover a map of an environment; however, the key difference is that active SLAM explicitly aims to reduce uncertainty in the map and the robot localisation. Observations can be predicted in a similar way as for exploration problems, but the key difficulty is to design objective functions that consider these sources of uncertainty, while also encouraging exploration. For landmark-based SLAM approaches that use Kalman filter estimation techniques, objective functions can be formulated using information-theoretic measures, such as Fisher information (Feder et al., 1999), Shannon entropy (Bour-
gault et al., 2002), the trace of the error covariance matrix (Huang et al., 2005) and differential entropy (Atanasov et al., 2015). Similar objective functions have been formulated for particle filter estimation techniques, such as Kullback-Leibler divergence (Carlone et al., 2010). Heuristics can be employed in the objective function to explicitly reward loop-closures (Kim and Eustice, 2015). A brief survey of active SLAM approaches is presented in Cadena et al. (2016).

Active object classification

Another common active perception problem is object classification. This problem is also about learning a map of the environment, but in this case the map also encodes higher-level semantic information. Objects in the scene are simultaneously detected, localised, and classified as an instance of a predefined class. For example, Huber et al. (2012); Atanasov et al. (2014); Wu et al. (2015); Patten et al. (2016); Becerra et al. (2016) classify indoor objects such as food and tools, Patten et al. (2018) classify objects found in a farm yard such as cars and bins, Ramon Soria et al. (2018) classify apples and leaves in a tree, Arora et al. (2017) classify rocks based on their geological properties, and Hollinger et al. (2011a); Köhntopp et al. (2015) classify objects on the seabed. Most formulations represent the world probabilistically in some form of joint distribution over all objects classes and poses, such as using Bayesian networks (Arora et al., 2017), particle filters (Patten et al., 2018), and Gaussian processes (Ramon Soria et al., 2018).

Predicting and evaluating observations is particularly difficult in this context due to the complex perception models and large state spaces. A typical approach is to predict point cloud observations using ray tracing of the environment from candidate viewpoints (Patten et al., 2018; Wu et al., 2015). These predictions may also hallucinate predicted objects based on their object class and a library of object models (Kriegel et al., 2013; Patten et al., 2016). Some formulations also jointly reason over the segmentation of objects (Huber et al., 2012; Pajarinen and Kyrki, 2015; Ramon Soria et al., 2018). Most approaches use information-theoretic objective functions defined for this joint probability distribution, such as mutual information (Hollinger et al., 2011a; Huber et al., 2012; Atanasov et al., 2014; van Hoof et al., 2014; Patten et al., 2016, 2018; Arora et al., 2017), typically computed using some form of approximation.
Our experiments in Section 3.6 are for a scenario that uses an object classification model based on these ideas.

Most of the above approaches assume that the locations of objects are already known or the objects are discovered opportunistically (Atanasov et al., 2014; Hollinger et al., 2011a; Wu et al., 2015; Patten et al., 2015, 2016). However, in larger environments, it is often necessary to explicitly encourage exploration of the environment to discover new objects. The objective function should balance the trade-off between exploration objectives and the primary perception task objectives, which may by achieved using a weighted sum of the objectives (Bourgault et al., 2002; Kriegel et al., 2013; Patten et al., 2018) or multi-criteria decision making (Quattrini Li et al., 2016). In Chapter 4 we present a new way of balancing this trade-off by defining new reward polygons in unseen areas.

2.1.2 Moving targets

Many active perception problems involve perceiving moving targets, such as in tracking, search, and pursuit-evasion (Chung et al., 2011; Robin and Lacroix, 2015), as well as the mission monitoring problem introduced in Chapter 5. The targets may be, e.g., robots, humans, or animals. Observation prediction models in this context require predicting the dynamic states of the targets, or trajectory prediction. In this subsection, we review trajectory prediction models, and then later in Section 5.8 and Appendix A we propose new example models used as prediction models for evaluating our planning algorithms.

Prediction models

Approaches to the trajectory prediction problem largely depend on the underlying assumptions about the motion of the agent or object of interest. The simplest assumption is that the agent is stationary, which may be appropriate for some applications (Xu et al., 2013; Hönig and Ayanian, 2016), but would require constantly replanning as the scene changes.

Another simple assumption is that the agent will continue moving with constant or near-constant velocity or acceleration (Chiang et al., 2014; Reece and Roberts, 2010).
These assumptions allow for efficient computation and may be a sufficient prediction model to improve performance in tracking applications. However these simple models are often insufficient for dynamic collision avoidance applications where performance is highly dependent on the accuracy of the predictions rather than the accuracy of the current position estimate, particularly when extrapolating relatively far into the future.

Similar models can be extended to model multi-agent collision avoidance in crowds, which is a highly-active area of research (Lerner et al., 2007; Pellegrini et al., 2009; Yamaguchi et al., 2011; Trautman et al., 2013; Kim et al., 2015). Due to the compounded uncertainty of crowds these predictions are usually only reliable in the very near future.

Another common assumption is that the agent will reliably follow one of possibly many predefined paths, which may come from training data based on previous agent trajectories (Bruce and Gordon, 2004; Aoude et al., 2013) or known mission plans (Section 5.8.1). For the case where there are multiple possible predefined paths, the predictions are characterised by a multi-modal distribution. Furthermore, the predictions can be improved by estimating which single paths out of all possible paths is the agent more likely to be following (Bruce and Gordon, 2004; Aoude et al., 2013; Ahmad et al., 2016). The benefit of this approach is that any underlying assumptions about the agent’s motion, such as probabilistic dynamics and velocity constraints, can be modelled implicitly within the predefined paths. However, most realistic scenarios are less predictable and therefore it is impractical to find a small discrete set of paths that accurately model the possible paths of the agent.

In the trajectory prediction model we propose in Appendix A, we reason over a potentially infinite number of paths that the agent could possibly take. However we group together all paths that have a common end position and then predictions are performed by first updating a belief for the end position of the agent’s path. If the end position is known then this information can be used to improve the predictions (Pellegrini et al., 2009). However, in most cases the end position is not known and therefore it is advantageous to instead maintain a belief over all possible end positions based on observations or training data.
2.1 Prediction models and objective functions

Intention inference

An agent is often guided by an underlying intention to move to another specific region of the environment. In this sense, each movement taken by the agent can be thought of as an action leading towards achieving the underlying intention to move to a goal region of the environment. This falls into the scope of plan recognition (Charniak and Goldman, 1993; Goldman et al., 1999). General approaches to plan recognition are formulated around the idea that every observed action gives information about higher-level objectives, while reasoning over the higher-level objectives in turn gives information to predict future actions. Example applications of plan recognition in robotics includes robot table tennis, interactive humanoid robots (Wang et al., 2013) and inferring the plan of wheelchair operators (Huntemann et al., 2013).

The plan recognition concept has been used to formulate solutions to trajectory prediction problems. Some interesting proposed methods use a general definition of the agent’s intention and therefore allow the use of more general frameworks such as POMDPs (Bandyopadhyay et al., 2012). It can also be beneficial, and computationally efficient, to consider a more narrow definition of intention. Kim et al. (2015) define the agent’s intention to maintain a velocity close to an unknown preferred velocity which may change slowly over time as the agent moves through a crowd, using Kalman filters and a maximum-likelihood estimate of the model parameters. Schreier et al. (2014) define intentions as typical manoeuvres while driving on structured roads (e.g., changing lanes), where inference is aided by observed properties of the road (e.g., the existence of lanes). An alternative interpretation of these concepts with a fundamentally similar formulation is presented by Nishimura and Schwager (2018), where one agent aims to convey one of several possible messages to another agent through its motion. Similar concepts have also been proposed in multi-player games, known as stochastic Bayesian games, where the inferred behaviours of other agents are described as one of several possible “types” (Albrecht et al., 2016; Barrett et al., 2011).
Objectives

Formulating a perception objective function suitable for planning requires these models to be reliable over a reasonably long time period. Ideally, the model should be probabilistic so planning can be performed with respect to all possible outcomes. The objective function can vary greatly from task to task, such as minimising the uncertainty of the tracking estimation (Xu et al., 2013), uncertainty of the intention inference (Nishimura and Schwager, 2018), or the distance to the agent (Švec et al., 2014). In Chapter 5 we formulate expected observation time as an objective, and we propose planners that optimise with respect to long-term probabilistic trajectory predictions.

2.1.3 Continuous fields

In many applications, particularly environmental monitoring, the properties of the environment being estimated constitute some form of continuous process that has spatial and temporal correlations. Observations of this continuous process involve taking in-situ point measurements. There are many examples of phenomena that can be represented in this way, such as temperature (Garg and Ayanian, 2014), bioacoustics activity (McCammon and Hollinger, 2018), algae blooms (Das et al., 2015), precipitation (Garg and Ayanian, 2014), clouds (Reymann et al., 2018), and communication bandwidth (Penumarthi et al., 2017). GPs have also been used for modelling and learning the preferences of human operators during information gathering missions (Somers and Hollinger, 2016).

GPs (Rasmussen and Williams, 2006) are the most common representation of these phenomena for the context of active perception. A GP defines a multivariate normal distribution where the variables are an infinite collection of random variables defined over a continuous domain. A covariance function describes the relationship between the variables. One of the most used covariance functions is the squared exponential, which is a decreasing function of distance between two points in the domain. Machine learning algorithms are used to fit the parameters of the covariance function.

What is particularly useful about GPs is the ability to make predictions about the value of the phenomena for parts of the domain that have not yet been directly
measured. The value at a test point is predicted by considering the correlation of this point to all measured datapoints. These predictions come in the form of a normal distribution, i.e., expectation and variance. For active perception, these predictions are useful because they can be used to estimate the value of visiting a viewpoint. Common measures of information gain are mutual information (Hollinger et al., 2013; Patten et al., 2013; Garg and Ayanian, 2014; Hitz et al., 2017), variance reduction (Binney and Sukhatme, 2012; Hollinger et al., 2013) or an upper confidence bound (UCB) (Marchant et al., 2014; Das et al., 2015).

GPs provide the advantages of being widely applicable and relatively efficient to compute. GPs also have associated objective functions that are monotone submodular (described later in Section 2.2.2), which allows deriving performance guarantees for myopic planners. However, they are only applicable to tasks where the observations are point measurements, the uncertainty is Gaussian noise, and the correlations between observations can be described by a suitable kernel function (Rasmussen and Williams, 2006); GPs would not be suitable for most of the perception models described in the previous subsection.

Continuous fields can also be used to model surfaces of objects. For example, the Gaussian process implicit surface (GPIS) is an extension to GPs that can be used in a similar way to a standard GP to provide measures of uncertainty of object shapes (Martens et al., 2017; Ramon Soria et al., 2018; Hollinger et al., 2013). Hilbert maps (Ramos and Ott, 2016) is a related concept with similar properties.

### 2.2 Informative path planning

In this section we review algorithms for informative path planning and related problems. The planning module for active perception systems can be considered to be solving a type of informative path planning problem; though “active perception” typically implies the planner is interacting with a more complex perception model than what is usually considered in informative path planning (Patten, 2017). We begin this section by considering planners for simplified formulations, particularly the travelling salesman problem (TSP). Then we discuss single-robot planners for active
2.2 Informative path planning

Finally we consider multi-robot planners, both centralised and decentralised.

2.2.1 Simplified problem formulations

The TSP is a canonical path planning problem, which is often used to describe informative path planning formulations. This subsection introduces the TSP and variants relevant to active perception. These variants are relevant to several formulations considered in this thesis, particularly Section 4.2 and Section 3.5.

Travelling salesman problem

The TSP is the problem of finding the shortest path that visits all given cities and returns to the origin. More specifically, a TSP instance can be described as a graph with vertices corresponding to cities and edges corresponding to travel distances between pairs of cities. The goal is to find a Hamiltonian cycle, i.e. a closed path that visits each vertex exactly once, that has minimum weight.

The TSP is studied in a wide range of fields, particularly operations research (Toth and Vigo, 2001). It is an NP-hard problem and significant attention has been devoted to finding efficient heuristic algorithms for the problem.

In its purest form, the TSP can be used to describe simple robotic coverage problems (Galceran and Carreras, 2013). However, there are several variants and generalisations of the TSP that more closely relate to complex robotics tasks; we describe several of these below. In some cases, the TSP is used to help setup and motivate a robotics problem before developing a custom solution algorithm, while in other cases TSP solution algorithms are used to solve sub-problems of a more general robotics problem.

m-TSP

The m-TSP generalises the TSP to multiple agents, which requires assigning nodes to agents and finding a path for each agent. There are several variations of the m-TSP with different objective functions such as minimising the maximum-cost path,
or minimising the sum of path costs (Bektas, 2006; Lagoudakis et al., 2005). Many different approach have been proposed, such as exact algorithms, heuristics based on the standard TSP, neural networks and genetic algorithms. The focus in the literature is solely on centralised and offline algorithms.

**Generalised TSP**

In robotic coverage problems it is often desired to observe a set of points using range sensors. This does not require the robot to visit the points, but rather just requires the robot to be within observation range of the points. This problem naturally maps to the generalised travelling salesman problem (GTSP) (Noon and Bean, 1989) and related variants, where it is required to visit one city from every set of cities, for a collection of city sets.

The GTSP is the case where these city sets are discrete and finite. A well known solution to the GTSP transforms the problem into a standard TSP and then any TSP solver can be applied (Noon and Bean, 1989). Recently, specialised solvers have been proposed that are typically more efficient (Smith and Imeson, 2017). The GTSP can be thought of as a generalisation of the set cover problem (Vazirani, 2001; Hochbaum, 1997) where path constraints are imposed on the set-selection costs; however, counterexamples show that GTSP is fundamentally harder than set cover and greedy solutions can perform arbitrarily poorly (Best and Fitch, 2016).

Several variants describe the city-sets as continuous spatial regions. In some ways this problem is harder than the discrete-set case due to having to deal with the infinite search space; however, efficient approximation algorithms have been developed that exploit the spatial-structure of the problem. This problem is often referred to as the TSP with neighbourhoods for the case of circular regions (Dumitrescu and Mitchell, 2003), and has occasionally been extended for the case with polygonal regions (Faigl et al., 2013), which is closely related to the watchman route problem (Faigl, 2010).

The GTSP has been formulated for robotics applications. For example, Mathew et al. (2013) formulate a mobile refuelling problem as a GTSP where the sets describe possible refuel points in time and space. In Chapter 4 we consider an active perception
formulation with continuous regions, but additionally has budget constraints, which more closely maps to the OP, described below.

**Orienteering problem**

The standard TSP requires finding a path that visits all vertices in shortest time. In many applications, particularly in informative path planning, a full coverage is not desirable, or even possible. Instead, the problem is to visit the maximum number of vertices in a given time. This time or distance constraint represents some form of budget that cannot be exceeded, such as due to fuel constraints, or specifications given by an operator. In contrast to the TSP, these problems not only require determining the order to visit vertices, but also the selection of which vertices to visit. Also, typically some vertices may be more important than others and therefore a common objective is to maximise a weighted sum of the visited vertices.

The prize-collecting TSP (Balas, 1989) requires finding a selection of vertices to visit, where rewards are gained by visiting vertices and deducted for omitting vertices. This presents a trade-off between the reward-value of selecting a vertex versus the travel cost required to visit the vertex. This formulation is useful in some scenarios, however it does not enforce budget constraints, and it may be difficult to define the rewards correctly to reflect a desired trade-off (Faigl and Hollinger, 2018).

The orienteering problem (OP) (sometimes known as the selective TSP) is a distance-constrained variant of the TSP that appears in a wide range of contexts (Laporte and Martello, 1990; Vansteenwegen et al., 2011; Gunawan et al., 2016). This problem is also known to be NP-hard as there exists a transformation from the Hamiltonian circuit problem (Laporte and Martello, 1990). Similar to the standard TSP, there are many relevant variants to the OP; the most relevant variants to informative path planning and this thesis include the team-OP that extends the problem for multi-agent systems, generalised-OP (Geem et al., 2005) that defines the objectives as a function of discrete sets, and the orienteering problem with neighbourhoods (OPN) where rewards are collected by visiting continuous regions.

Chapter 4 considers a new OP variant that includes continuous polygonal goal regions (similar to the GTSP variants), and multiple agents (similar to the team-OP). While
there are existing techniques for the team-OP (Dang et al., 2013a; Archetti et al., 2007; Dang et al., 2013b), none of these address continuous polygonal goal regions. Also, while OP formulations have been applied to many problems (Gunawan et al., 2016; Vansteenwegen et al., 2011), the focus has mostly been on offline planning rather than online settings where goals are discovered over time, which is more applicable to robotics.

2.2.2 Single-robot planning

In this section we review methods for single-robot informative path planning. We begin by highlighting the importance of viewpoint dependencies, and how the above TSP formulations do not adequately consider this. We then discuss myopic methods and the concept of submodularity, followed by non-myopic methods.

Dependencies between viewpoints

The standard TSP fails to capture one of the most important aspects of informative path planning reward functions: dependencies between viewpoints. More specifically, in the TSP, the reward for visiting a vertex is considered to be independent of whether or not other vertices have been visited. In information gathering, this is typically not the case; the value of making an observation usually depends on what other observations have been made since observations may provide overlapping information.

The GTSP, as described above, can model some types of observation dependencies; the value of a vertex is reduced to zero if the vertex is the element of a set that has already been visited. This can provide a useful approximation to more general definitions of dependencies while enabling relatively computationally efficient solutions. Permitting the vertex sets to be non-mutually exclusive allows richer descriptions of dependencies since vertex rewards after an observation can be reduced but still greater than zero. Adding more and more overlapping sets could potentially describe any types of dependencies, however the effectiveness of associated solution algorithms would likely diminish. Our self-organising map algorithm in Chapter 4 aims to solve this general class of problems for the case where the sets are continuous polygonal
regions; the algorithm exploits the geometry of the problem to efficiently find solutions. However, there are limitations to this definition of dependencies, and thus a more principled approach is often more appropriate.

An important implication of the above discussion is that informative path planning is a generalisation of the TSP, and thus is NP-hard. More importantly, TSP formulations and associated algorithms are typically not sufficient for solving informative path planning problems. We review existing methods for informative path planning in the following subsections, and propose new approaches in this thesis.

**Exploiting submodularity with greedy approaches**

Submodularity is a property of set functions that describes “diminishing returns”; i.e., if a set function is submodular then adding an element earlier is worth more than adding the same element later. More formally, a set function $f$ defined over the power set of a finite set $\Omega$ is such that $\forall X, Y : X \subseteq Y \subseteq \Omega$ and $\forall x \in \Omega \setminus Y$ the inequality

$$f(X \cup \{x\}) - f(X) \geq f(Y \cup \{x\}) - f(Y)$$

holds (Nemhauser et al., 1978). A set function is monotone submodular if it additionally has the monotone property:

$$f(X) \leq f(Y), \forall X, Y : X \subseteq Y \subseteq \Omega,$$

i.e., $f(X)$ does not decrease if more elements are added to the set $X$.

The significance of having an NP-hard optimisation problem with a monotone submodular objective function is that it often enables the use of efficient greedy algorithms with performance guarantees. In the simplest case, these are problems where the aim is to find a set $X^*$, defined as

$$X^* = \arg \max_{X \subseteq \Omega, |X| \leq k} [f(X)],$$

where $k$ is a cardinality constraint on the size of the set. The greedy policy is simply to add elements $x_i \in \Omega$ to the solution set $X_{\text{greedy}}$ until the cardinality constraint is
2.2 Informative path planning

met, where

\[ x_i = \arg \max_x [f(X_{\text{greedy}} \cup \{x\})]. \]  (2.4)

This greedy selection policy clearly has polynomial runtime and, more importantly, guarantees a constant-factor approximation (Nemhauser et al., 1978), such that

\[ f(X_{\text{greedy}}) \geq \left( 1 - \frac{1}{e} \right) f(X^*) \approx 0.63 f(X^*), \]  (2.5)

and typically achieves much better than these bounds in practice. Common extensions that provide similar guarantees include adding non-uniform element costs (Krause and Guestrin, 2011), and problems with adaptive policies (Golovin and Krause, 2011).

There are many examples of real-world optimisation problems with monotone submodular objective functions, such as sensor placement, activity recognition and data mining (Krause and Guestrin, 2011). Many robotic active perception problems are often formulated with monotone submodular objective functions, particularly the information-theoretic measures that appear in many of the examples discussed in Section 2.1, such as mutual information. In these cases, the set \( \Omega \) represents candidate viewpoints for making observations and \( f \) is the objective function that describes the perception task. Many robotics approaches directly apply the greedy algorithm described by (2.4), such as touch-based localisation (Javdani et al., 2013), object recognition (Wu et al., 2015; Patten et al., 2016), target tracking (Dames et al., 2017), and thermal mapping (Cunningham-Nelson et al., 2015).

**Non-myopic approaches**

Despite the wide-spread use of greedy algorithms for informative path planning, there are several limitations with these methods. Most significantly, incorporating path costs and path constraints into problem (2.3) breaks the assumptions, resulting in arbitrarily poor performance (Singh et al., 2009a; Best and Fitch, 2016). Also, even in cases where the guarantees in (2.5) hold, this analysis describes the \textit{worst-case} performance; more sophisticated algorithms are likely to achieve better \textit{actual} performance for many problem instances. Finally, many important problems do not have the monotone submodularity property, such as when optimising for decisions that depend on the gathered information (Chen et al., 2017). Non-myopic approaches
Several approaches for problems with submodular objective functions extend the basic greedy approach to non-myopic algorithms. Commonly, these approaches plan over a discretised topological representation (i.e., graph or “roadmap” (Kavraki et al., 1996)) representation of the workspace. The recursive greedy algorithm (Chekuri and Pal, 2005) recursively divides the problem into sub-problems to give a quasi-polynomial runtime with performance guarantees. Closely related approaches offer efficiency improvements by exploiting assumed spatial correlations (Singh et al., 2007, 2009b). Hollinger et al. (2013) employ the submodular greedy algorithm to select informative sensing locations then plan a TSP tour through these points. Approaches have also been proposed that plan directly over a continuous workspace by extending RRT approaches (LaValle and Kuffner, 2001) to information gathering problems; these include RIG (Hollinger and Sukhatme, 2014) which adaptively prunes less-informative paths from the search tree, IRRT (Levine et al., 2010) is based on similar ideas but is specifically for target tracking problems, and RRC (Lan and Schwager, 2016) finds persistent monitoring cycles. ReASC (Hollinger, 2015) extends RIG for adaptive problems by making “optimistic” approximations of expected rewards. Hitz et al. (2017) plan over a continuous workspace using an evolutionary algorithm to optimise parameters of splines that represent trajectories.

There has also been several approaches that do not explicitly require submodularity. Binney and Sukhatme (2012) propose the use of branch and bound tree search techniques (discussed further in Section 2.3.3) to efficiently and exhaustively explore the search space. Monte Carlo tree search (discussed further in Section 2.3.4) has recently become popular for its ability to heuristically expand a search tree with convergence-rate guarantees (Marchant et al., 2014; Nguyen et al., 2015; Hefferan et al., 2016; Patten et al., 2018; Arora et al., 2017). The RAId algorithm (Lim et al., 2016) solves the group Steiner problem as a sub-problem to find solutions in polynomial-time and with optimality guarantees. McCammon and Hollinger (2018) propose a hierarchical planner that first extracts the most informative regions of an information field then plan weighted coverage paths over these regions. In exploration scenarios, imitation learning can be used for planning by learning from non-myopic example plans that
have assumed full knowledge of the world (Choudhury et al., 2017).

Persistent monitoring is an important class of information gathering problems that typically require non-myopic planning. This class of problems generally requires finding cyclic paths that repeatedly observe parts of the environment over an infinite time horizon. This long-term periodic constraint, similar to TSP constraints, mean that myopic solvers are inadequate. Several non-myopic algorithms have been proposed, including rapidly exploring random cycles (Lan and Schwager, 2016), MCTS variants for surveillance and patrolling (Kartal et al., 2015; Hefferan et al., 2016), mixed integer programming (Yu et al., 2016), and bounded approximation algorithms for finding min–max latency paths (Alamdari et al., 2014). Although we do not explicitly consider the infinite horizon persistent monitoring problems in this thesis, our proposed self-organising algorithm can easily be setup to provide similar cyclic solution paths.

In the majority of the work referenced above, the objective function that the approaches have been tested on are relatively easy to compute. However, when considering more complex perception models (such as those described in Section 2.1), this is not the case, which severely limits how far the non-myopic planners can look ahead. Work that successfully apply non-myopic planners for these problems (e.g., Atanasov et al. (2014); Becerra et al. (2016); Patten et al. (2018)) usually involve careful consideration of this issue when developing perception models, such as making maximum likelihood approximations or using fast, specialised data structures.

The ideas in this thesis are particularly motivated by the non-myopic planners discussed above. Our main contribution relative to the above work is the generalisation of these formulations to multi-robot scenarios. Although this is not our focus, our proposed methods could also be applied to single-robot problems. The generic algorithm proposed in Chapter 3 does not require submodularity, but it can exploit this property when appropriate by incorporating greedy heuristics. Most of the proposed methods plan over a topological representation of the workspace, while the algorithm proposed in Chapter 4 directly plans over continuous space. Several of the example objective functions we use throughout to test our methods are richer perception tasks, such as object classification, while others are formulated as efficient approximations of more complex tasks to allow expanding deeper into the search space.
2.2.3 Centralised multi-robot planning

Generalising informative path planning for multi-robot systems is inherently a more difficult problem. Most of the considerations described above for single-robot planning, such as viewpoint dependencies, submodularity, and benefits of non-myopic reasoning, are also relevant to multi-robot planning. Approaches are characterised as being either centralised, i.e., one computer decides the actions of all robots, or decentralised, i.e., each robot decides its own actions while considering the team’s objectives. In this section we discuss centralised planning methods for informative path planning, and defer the discussion of decentralised planning to Section 2.2.4. We propose both centralised and decentralised planners in this thesis.

Centralised multi-robot planning shares many similarities to single-robot planning as the aim in both cases is for a single computing node to find a set of viewpoints. Therefore, many of the algorithms discussed above (Section 2.2.2) for the single-robot case have been adapted for the centralised multi-robot case. The main difference is that the search space generally scales exponentially in the number of robots. Thus, the focus when developing centralised algorithms is to improve the runtime scalability in the number of robots. For example, the recursive greedy algorithm has been generalised for the multi-robot case with a straight-forward extension to the original algorithm followed by heuristics and a branch and bound algorithm to improve runtime (Singh et al., 2007). Lan and Schwager (2016) propose an RRT-like algorithm for persistent monitoring then extend the algorithm by planning in the joint space of the robots. Coverage-type problems (Cao et al., 1988), which are a relatively simple case of informative path planning, have often been addressed with centralised planners: Hassan and Liu (2017) employ Voronoi partitioning of 3D structures followed by single-robot coverage algorithms, Dornhege et al. (2016) sample and rank candidate sensing locations then solve a multi-agent TSP, Hönig and Ayanian (2016) perform dynamic coverage by sampling from visibility polygons then assigning locations to robots.

Dec-POMDP formulations (Oliehoek and Amato, 2016) are typically solved using centralised planning to compute policies that are executed in a decentralised manner. While informative path planning is typically not explicitly formulated as a Dec-POMDP, it can be thought of as a special case. We discuss the Dec-POMDP in
more detail in Section 2.3.1.

Centralised planning is more common in robotic planning problems other than information gathering, particularly for application where it is common to have permanent infrastructure to support centralised systems, such as warehouses. The classical path planning problem, where the aim is to find collision free trajectories to goal locations, has received a lot of attention (Schwartz and Sharir, 1983; Solovey et al., 2016; Yu and LaValle, 2016); however the objectives here are fundamentally different to informative path planning.

2.2.4 Decentralised multi-robot planning

Decentralised planning offers many advantages over centralised planning approaches, particularly in outdoor environments where there is less permanent and reliable infrastructure. Most importantly, decentralised planning avoids having a single point of failure, and the robots should continue to behave reasonably even if communication is temporarily interrupted. Also, distributing the computing efforts over multiple nodes can increase the computational resources available to the team. Often, faster decisions can be made since the relevant computing is performed primarily onboard the robot that executes each decision.

However, there are many challenges to performing decentralised planning. The algorithms are inherently parallel algorithms, typically without time synchronisation, which can make the behaviour unpredictable. Each processing node (robot) has less information about the world compared to centralised systems; this includes knowledge about the state of the environment, the state of the other robots, and the intentions of the other robots. Ideally, the robots should still make reasonable decisions even if communication is disrupted. We review existing approaches to addressing these challenges as follows, beginning with relatively simple planning problems, then leading to active perception problems.

Swarm robotics

The swarm robotics literature (Brambilla et al., 2013) considers systems of very large teams of robots where each robot has extremely limited knowledge of the world, such
as only knowing the proximity to neighbouring robots. Additionally, communication is typically very limited, such as no explicit communication at all, or only being able to send small packets between local neighbours. Due to this limited information, the decision making performed by each robot is relatively simple. The focus is therefore on designing simple local decisions that result in emergent collective behaviours. This allows the team to complete tasks such as controlling local densities of robots (Demir et al., 2015), perimeter following (Caccavale and Schwager, 2017) and manipulation (Culbertson and Schwager, 2018). Unfortunately, this simple decision making and limited sensing is not enough to solve richer perception tasks. We believe it is much more appropriate to solve most active perception problems with smaller teams of robots that have increased onboard sensing, computation and communication capabilities.

**Distributed task assignment**

Task assignment problems (Munkres, 1957) involve assigning a set of tasks to a set of agents, such that each task is assigned to one agent, and each agent is assigned one task. Each task-agent pair has an associated assignment cost, and the aim is to minimise the sum of these costs. More formally, this problem involves finding a minimum-weight matching in a bipartite graph. The Hungarian algorithm solves this problem in a centralised manner in polynomial time (Munkres, 1957). Several decentralised algorithms have been proposed such as the distributed Hungarian algorithm (Chopra et al., 2017) and local task swaps (Liu et al., 2015). Auction-based methods consider generalisations where each agent can be assigned more than one task, or each task can be assigned to more than one agent (Dias et al., 2006).

The main benefit of this formulation and approaches is that they are relatively easy to compute. However, this formulation is typically not expressive enough for active perception tasks since rewards and/or costs are not additive. Also these approaches, particularly in the one-to-one case, are myopic planning. However, they have been used as a sub-routine of more sophisticated methods, e.g., for target tracking problems (Xu et al., 2013).
Non-myopic planning

Decentralised myopic methods with performance guarantees have been proposed for monotone submodular problems (Hollinger et al., 2009; Patten et al., 2013; Garg and Ayanian, 2014; Kenna et al., 2017). However, the benefits of myopic planning are equally applicable to the multi-robot case, as they are to the single-robot case (discussed in Section 2.2.2). Existing decentralised planning algorithms for multi-robot informative path planning typically involve exploiting problem-specific characteristics. The auction-based methods mention above (Dias et al., 2006) involve each robot negotiating over which tasks it will perform, and are more appropriate for coverage and exploration problems (Zlot et al., 2002). Stranders et al. (2009) combine max-sum message passing (discussed further in Section 2.3.6) with branch and bound pruning (discussed further in Section 2.3.3) to find sequences of viewpoints that minimise the entropy of a Gaussian process. Hollinger et al. (2009) propose solving a finite-horizon POMDP (discussed in Section 2.3.1) for target search problems. Corah and Michael (2017) propose a distributed sequential greedy assignment algorithm for multi-robot exploration, and provide performance guarantees by exploiting a submodularity assumption. Gan et al. (2014) solve search problems with inter-agent collision avoidance by solving a constraint optimisation problem and refining trajectories to avoid collisions. Atanasov et al. (2015) propose a decentralised algorithm for tracking targets that have linear Gaussian dynamics, such as for active SLAM (discussed in Section 2.1.1).

Our proposed algorithm in Chapter 3 is applicable to a general class of problems, which includes all problems mentioned above, since it does not rely on specific assumptions about the problem; however, our approach can readily incorporate problem-specific approximate solutions, such as those above, as heuristics to guide the search. The algorithm proposed in Chapter 6 is also a decentralised, non-myopic planning algorithm; however, in this case we make several assumptions about the problem in order to formulate an efficient solution for this specific case.
Communication considerations

The majority of decentralised coordination algorithms involve communicating each robot’s plan to other robots. This communicated information is used to ensure the team’s objectives are being met. One advantage of decentralised planning is that reasonable behaviour should still be exhibited if the communication breaks down. The role of communication has been considered in various ways, which we discuss as follows.

Several planners have been demonstrated to have a graceful degradation of performance as communication becomes less reliable. The max-sum algorithm (discussed further in Section 2.3.6) demonstrates this property, which is explained as being due to message redundancy (Farinelli et al., 2008). Otte and Correll (2013) demonstrate this property for a distributed RRT algorithm that solves coordinated path planning with collision avoidance. Otte et al. (2017) compare the performance of distributed auction algorithms for task allocation in harsh communication environments. The Dec-MCTS algorithm we propose in Chapter 3 is also demonstrated to have this useful property.

It is also possible to take an active approach to exploit this communication redundancy by explicitly selecting which messages to transmit. In most cases, “communication planning” has been performed where the messages are observations. The value of these messages can be measured by considering their effect on data fusion accuracy (Williamson et al., 2008; Kassir et al., 2015). Planning to communicate plans, rather than observations, is less common; Unhelkar and Shah (2016) address this problem for Dec-POMDP formulations by defining communication value as the reduction in reward caused by not communicating. We recently proposed a new approach to this problem that maintains a probabilistic belief over the future plans, and then measures the information value as uncertainty of the reward distributions; we summarise our approach in the context of Dec-MCTS in Section 3.7.

An alternative, and complementary, approach to improve communication is to actively position the robots in order to improve the communication channel. This problem is known as communication-aware motion planning (CAMP), where communication objectives, such as maximising network throughput, is formulated as secondary objec-
tives when planning the motion of the robots. It is not surprising that robot motion can be exploited to improve communication since communication quality is spatially varying, due to, e.g., the well studied effects of path attenuation, or the more complex issue of multi-path fading (Lindhé, 2012). Examples of CAMP problems include selecting robot paths to perform connectivity maintenance (Sabattini et al., 2013), periodic connectivity (Hollinger and Singh, 2012), or communicate with a fixed base station (Ghaffarkhah and Mostofi, 2011; Lindhé and Johansson, 2013). In Chapter 5 and Chapter 6 we consider a new problem of this type where robots choose to stop and communicate at times and locations that have a high prediction probability of communication success.

A difficulty in applying CAMP approaches in practice is that they assume a known model that maps pairs of spatial locations to communication quality. In general, learning this model with sufficient accuracy is an unsolvable challenge. For specific tasks, sufficient models may be learnt in indoor environments (Banfi et al., 2017), or by using local measurements of multi-path fading in complex environments (Lindhé and Johansson, 2013). We propose new models suitable for mission monitoring in Chapter 5, which combines communication models with probabilistic trajectory prediction models (introduced in Section 2.1.2).

2.3 Planning algorithms

In this section we introduce several planning algorithms relevant to this thesis that have been developed primarily outside of robotics. We begin by broadly discussing sequential decision problems, then looking more closely at optimal stopping, branch and bound tree search, Monte Carlo tree search, self-organising maps, and inference-inspired decentralised algorithms. These algorithms are mostly presented in the context of the contributions of this thesis.
2.3 Planning algorithms

2.3.1 Generic sequential decision problems

MDPs

Sequential decision problems are problems in which an agent’s utility depends on a sequence of decisions. One of the most basic forms is the Markov decision process (MDP) (Bellman, 1954). An MDP consists of a set of states \( s \) (including an initial state \( s_0 \)), a set \( \mathcal{A}(s) \) of actions \( a \) in each state (which is an empty set if \( s \) is a terminal state), a transition model \( P(s'|s, a) \) which defines the probability of moving from state \( s \) to state \( s' \) when action \( a \) is executed, and a reward function \( R(s) \). The optimisation problem is to find a policy \( \pi \) that defines an action \( \pi(s) \) to take when in a state \( s \). An optimal policy is \( \pi^* \) is the policy that yields the highest expected utility, defined as the sum of rewards \( R(s) \) for a sequence of visited states. Techniques such as value iteration and policy iteration are optimal algorithms commonly used for solving this formulation of sequential decision problems (Russell and Norvig, 2010, Chapter 17).

If \( \mathcal{A}(s) \) is a finite set \( \forall s \), the search space can be represented as a tree of states, where every path through the tree from the root node to a leaf node represents a valid sequence of state transitions with an associated utility. The action \( a = \pi(s) \) and the probabilistic transition model \( P(s'|s, a) \) defines which branches of the tree are taken when the agent executes the policy \( \pi \). In this way, the problem of optimising \( \pi \) can be thought of as a tree search; we discuss related tree-search algorithms in the following subsections.

POMDPs

A common extension of MDPs is the partially observable Markov decision process (POMDP), which extends the formulation for settings where the agent has a probabilistic belief \( b(s) \) of the current state \( s \). Policies need to be represented by a function of the belief, i.e., \( \pi(b) \), rather than a function of the state. In principle this is similar to an MDP but the key difficulty is that the space of all \( b \) is a continuous set. In the worst case, finding optimal policies for POMDPs is PSPACE-hard (Papadimitriou and Tsitsiklis, 1987).
2.3 Planning algorithms

Dec-POMDPs

Centralised multi-agent problems can be formulated as an MDP/POMDP where $s$ represents a joint state of the agents and $a$ represents a joint action of the agents; similar algorithms can be applied to these problems. A more interesting case is where these agents’ act in a decentralised manner. In these problems, each agent may have a different belief $b$ of the state $s$ and yet still needs to select actions that coordinate with other agent’s actions. This problem is known as a decentralised partially observable Markov decision process (Dec-POMDP) (Oliehoek and Amato, 2016). In the general case, Dec-POMDPs are formally harder to solve optimally than POMDPs, as it is NEXP-hard (Bernstein et al., 2002).

Approaches

Most solutions to problems formulated as any of the above formulations are offline algorithms, i.e., prior to executing a task, a policy $\pi$ is optimised for all possible states/beliefs. In contrast, online planners compute the relevant parts of $\pi$ as required while executing the task. The online setting simplifies the search procedure since it can focus on optimising the next action $\pi(s_0)$ from the current state $s_0$. However, online settings typically have much tighter time constraints that dictate the need to make fast decisions rather than optimal decisions. POMCP (Silver and Veness, 2010) and DESPOT (Somani et al., 2013) are common online algorithms for POMDPs. The time constraints of online settings also often necessitate using non-adaptive algorithms (Hollinger et al., 2013), as discussed earlier in Section 1.3.3.

The vast majority of Dec-POMDP solutions typically involve centralised, offline policy optimisation, followed by decentralised execution (Oliehoek and Amato, 2016; Amato, 2015; Kumar et al., 2015; Omidshafiei et al., 2017). The approach of Spaan et al. (2006) is a notable exception that performs planning and execution in an online, decentralised manner; our proposed algorithms in Chapter 3 and Chapter 6 also take this general approach.
Sequential decision formulations in this thesis

We do not explicitly formulate the sequential decision problems in this thesis as Dec-POMDPs or related formulations. The problems presented could be considered as specific instances of these formulations. However, we feel the problems presented and associated algorithms are better described using problem-specific formulations and notation.

2.3.2 Optimal stopping

One of the simplest forms of sequential decision problems are optimal stopping problems (Chow et al., 1971). These problems involve a binary choice at each time instant; at each time instant, the decision at hand is simply whether to stop or continue.

Secretary problem

The secretary problem (Ferguson, 1989) is one of the most extensively studied optimal stopping problems. The problem is often presented as a scenario where an administrator wants to hire the best secretary out of $n$ applicants for the position. Each applicant is interviewed one-by-one by the administrator. After each interview the administrator must immediately hire the applicant and perform no more interviews, or reject the applicant and continue interviewing the next applicants. An applicant cannot be recalled after being rejected. The administrator can rank the current applicant against all previous applicants, but is now aware of how they compare to future applicants. If the first $n - 1$ applicants are rejected, then the $n$th applicant must be hired.

An elegant solution to the secretary problem is the following strategy: reject the first $n/e$ applicants (where $e \approx 2.72$ is Euler’s number), then hire the first applicant who is better than every applicant interviewed so far. The first phase is essentially learning about the distribution of quality of the applicants, and the second phase is acting on this information. This strategy has a $1/e \approx 37\%$ probability of hiring the best applicant (Bruss, 2000).
Several variants have been considered, such as if $n$ is unknown (Presman and Sonin, 1972), or $k > 1$ secretaries are to be hired (Girdhar and Dudek, 2009). Similarly elegant solutions have been proposed for these variants.

**Repeated binary decisions**

If the binary choice at each timestep can be repeated, the problem can be considered to be one-dimensional in the sense that it involves a choice of nonoverlapping intervals along a single dimension representing time. There are several examples of this type of problem in robotics, which we discuss as follows.

Lindhé and Johansson (2013) studied an optimal stopping problem for a robot that communicates with a base station while traversing a predefined path. The robot must choose stopping points that maximise communication quality while also making progress along its path.

Das et al. (2015) applied optimal stopping theory to the problem of selecting sampling locations for persistent collection of water samples by an AUV. Similar to Lindhé and Johansson (2013), the stopping locations are restricted to points along a predefined path. Sampling is instantaneous, rather than pausing motion for a planned time interval such as in Lindhé and Johansson (2013).

The *beachcombers’ problem* (Czyzowicz et al., 2015a,b) is a related theoretical problem where a team of robots perform coverage of a one-dimensional interval. Each robot can switch between two behaviours: searching slowly while observing, or walking quickly but blindly. The decision of switching between the two behaviours is similar to optimal stopping.

**Extending to multiple dimensions**

The *spatiotemporal optimal stopping* problem formulated in Chapter 5 extends the optimal stopping robotics formulations discussed above in several ways. Most importantly, the decisions are no longer binary, but instead stopping intervals are described by both temporal and spatial components. Additionally we formulate motion and observation models suitable for the *mission monitoring* application. Our solution
algorithm in Chapter 5 has similarities to solutions to the above problems, in that the algorithm is a type of dynamic programming (Cormen et al., 2001).

However, instead, we describe our algorithm as a type of sweep-plane algorithm. Sweep-plane algorithms are often used for computational geometry problems such as Voronoi decomposition, intersections between line segments and unions of rectangles (Preparata and Shamos, 1985; De Berg et al., 2000). An $\mathbb{R}^{n-1}$ hyperplane is swept monotonically through an $\mathbb{R}^n$ space, and calculations are performed at event points. Classical robot motion planning problems can often be formulated geometrically and solved with sweep-plane solutions (Latombe, 1991; LaValle, 2006). Our approach features a sweep-plane moving through time, where the event calculations represent optimal sub-problems and lead to an optimal global solution.

Each sweep-plane event can be thought of as a vertex in a spatiotemporal search graph with edges linking back to previous events. This construction forms a directed acyclic graph and therefore a longest path can be computed in polynomial time (Lawler, 1976; Cormen et al., 2001). Bopardikar et al. (2014) employ this approach for dynamic vehicle routing, where an agent maximises the number of space-time demands visited. Our problem again is similar, however our agent seeks to occupy a region defined probabilistically over time. The novelty of our approach in comparison lies in our proposed graph construction algorithm to maintain optimality for a complex constraint space and objective function.

### 2.3.3 Branch and bound tree search

Branch and bound (BnB) is a popular class of algorithms for combinatorial optimisation problems that can be described as an optimal tree search (Clausen, 1999). As the search tree is expanded, subtrees that are guaranteed to not contain an optimal solution are successively pruned from the search space. This pruning enables an efficient exhaustive search to be performed since pruned subtrees do not need to be enumerated. The pruning crucially relies on being able to efficiently compute tight bounds for each subtree; loose bounds results in less pruning, while slow-to-compute bounds increases the iteration runtime.

An example expanded BnB search tree is illustrated in Figure 2.1, which shows how a
2.3 Planning algorithms

Figure 2.1 – Illustration of branch and bound tree search with two different expansion policies. Gold path is best solution found so far; green is current search tree; purple is pruned search space; grey has not yet been explored or pruned. Figure from Best and Fitch (2016).

search tree is expanded while pruning away suboptimal subtrees (purple). In (a), the tree is expanded using a naive depth-first policy and has pruned 42% of the search space after only visiting 25% of the nodes. In (b), the tree is expanded by visiting the subtrees that have the highest upper bounds (for a maximisation problem), which prunes 64% of the search space. In each case, the search returns the best solution found so far, along with computed bounds on the utility of the optimal solution.

There are several examples of BnB being applied to robotic path planning. Singh et al. (2009a) apply BnB to general environmental monitoring problems that use GPs and a mutual information objective. Binney and Sukhatme (2012) apply BnB to a similar scenario for marine robotics and use a variance reduction objective. In (Best and Fitch, 2016), we apply BnB to a type of informative path planning problem that generalises the set cover problem (Vazirani, 2001). D’Urso et al. (2018) apply BnB to a refilling problem in agricultural robotics.

While we do not propose BnB algorithms in this thesis, they serve as a useful introduction for the MCTS algorithms described in the following subsection. BnB and MCTS are similar except BnB requires problem-specific bounds with hard guarantees, whereas standard MCTS (UCT) relies on generally-applicable probabilistic-bounds derived from the Chernoff-Hoeffding inequality. Subtrees that are estimated to be suboptimal by MCTS are implicitly pruned since they are not visited again, however there always remains a possibility of eventually revisiting this subtree as the estimates change. Additionally, MCTS offers principled policies for the order in which the tree should be expanded, whereas for BnB effective expansion policies are harder
2.3 Planning algorithms

2.3.4 Monte Carlo tree search

Monte Carlo tree search (MCTS) (Browne et al., 2012) is a biased random sampling approach to tree search problems. Although it is a relatively new technique, with the main seminal work published in 2006 (Kocsis and Szepesvári, 2006), it has quickly found success in a diverse range of fields. Its most notable success to date was in the game of Go where a variant of MCTS beat the human world champion (Silver et al., 2016, 2017). It has recently gained popularity in robotics for online planning as a general approach for efficiently searching over a long planning horizon, and forms the basis of our Dec-MCTS algorithm proposed in Chapter 3.

Upper-confidence bounds for trees (UCT)

MCTS has been proposed in many different forms (Browne et al., 2012) but by far the most commonly used is the UCB applied to trees (UCT) algorithm (Kocsis and Szepesvári, 2006; Kocsis et al., 2006). The UCT algorithm performs an asymmetric expansion of a search tree using a best-first policy that generalises the UCB1 policy for multi-armed bandit (MAB) problems (Auer et al., 2002). We provide a detailed description of UCT later in Section 3.3.3 in the context of Dec-MCTS.

The UCT expansion policy provides theoretical guarantees for a polynomial bound on regret, and therefore is said to balance between exploration (search unvisited subtrees) and exploitation (revisit promising subtrees). Figure 2.2 is an illustrative example of the behaviour of UCT with a varying balance between exploration and exploitation. On the left, the expansion policy selects exploration only, which essentially results in breadth first search. On the right, the expansion policy selects exploitation only, which results in the search focusing on one suboptimal path. The theoretically-sound choice of parameter results in the tree in the middle, which successfully balances between focusing on promising paths while not leaving any subtrees too far behind in the expansion.
2.3 Planning algorithms

Figure 2.2 – A comparison between exploration and exploitation in Monte Carlo tree search with an example objective function. Green branches have higher empirical average rewards, whereas black have lower. The optimal first two actions are the middle subtree followed by the left subsubtree. Results shown after 800 expansions. Search space contains 177,147 possible paths.

UCT variants

Several other MCTS variants have been proposed, such as for exploiting smoothness of the reward function (Coquelin and Munos, 2007), for problems with partial-observability (Silver and Veness, 2010; Somani et al., 2013), and when using an alternative definition of regret (Feldman and Domshlak, 2014). A key component of our proposed Dec-MCTS algorithm is a novel UCT variant, D-UCT, that is suitable for decentralised planning. D-UCT accounts for a changing reward distribution (e.g., due to other robots changing their plans) by using a new expansion policy that generalises the D-UCB policy for switching bandit problems (Garivier and Moulines, 2011).

Parallelisation

MCTS is parallelisable (Chaslot et al., 2008), and various techniques have been proposed that split the search tree across multiple processors and combine their results. In the multi-robot case, the joint search tree interleaves actions of individual robots and it remains a challenge to effectively partition this tree. Auger (2011) addresses the related case of multi-player games, where a separate tree is maintained for each player; however, a single simulation traverses all of the trees and therefore this ap-
proach would be difficult to decentralise. Dec-MCTS is a similar approach to Auger (2011), except that each robot performs independent simulations. Each simulation is performed by sampling a locally stored probability distribution that represents the plans of other robots.

MCTS in robotics

MCTS has been recently been applied to a wide variety of single-robot tasks, including: active object recognition (Patten et al., 2018; Lauri et al., 2015), patrolling environments with adversarial agents (Hefferan et al., 2016; Kartal et al., 2015), information gathering by a glider in thermal wind-fields (Nguyen et al., 2015), environment exploration (Lauri and Ritala, 2016; Corah and Michael, 2017), autonomous science by planetary rovers (Arora et al., 2017), active parameter estimation for manipulation (Slade et al., 2017), and monitoring of a spatiotemporal process (Marchant et al., 2014). In most of these studies, the standard UCT algorithm is applied with problem-specific heuristics.

Our proposed decentralised MCTS algorithm is suitable for multi-robot generalisations of all of the above problems. So far, MCTS has been less studied in multi-robot scenarios, though promising ideas have been presented by Kartal et al. (2015) for centralised planning of a team of patrolling robots, and by Corah and Michael (2017) as a single-robot planner within a distributed multi-robot assignment algorithm for the context of exploration and mapping.

2.3.5 Self-organising maps

A self-organising map (SOM) (Kohonen, 1998, 1982) is a type of neural network that is trained to give a lower-dimensional representation of an input space, while preserving a given topological graph-based structure of the representation. SOMs have been used for a large range of modelling and inference applications, such as meteorology (Liu and Weisberg, 2011), water resources (Kalteh et al., 2008), and computer vision for surveillance (Maddalena and Petrosino, 2008). Most importantly for this thesis, SOMs have also been applied to vehicle routing problems, which we discuss further below.
Learning algorithm

While SOM networks have similarities to other neural networks, the main difference is the associated unsupervised learning algorithm designed for these networks. The main idea of the learning algorithm is to present example datapoints of the input space one at a time. For each presented datapoint, an output neuron (vertex of a network) that best matches the datapoint is selected as the “winner”. The winner neuron, along with neighbouring neurons, is adapted towards the datapoint by some fraction that is a decreasing function of the topological distance from the winner. These adaptations have the effect of reshaping the network while maintaining the topology. Eventually the network converges to a shape that best fits the presented data of the input space.

Figure 2.3 illustrates several example learnt networks using a simple implementation of the SOM unsupervised learning algorithm. The examples illustrate the result of using different given network topologies. In all cases, the behaviour of the algorithm naturally selects locations of the vertices that approximately matches the density of the sampled datapoints, while also keeping each vertex close to its topological neighbours.

SOM for path planning

SOM algorithms have also been adapted for the TSP and its variants (see Section 2.2.1 for a discussion of the TSP). The main insight is to let the input space be the specified locations that need to be visited, and the network represent the solution to the problem. For the standard TSP, the solution is a closed path, which can be represented by the cycle topology in Figure 2.3b, or as a longer cycle in Figure 2.3c. If there are multiple vehicles, as in the m-TSP, then the solution can be represented by a set of cycles, as in Figure 2.3d. If an open path is required, i.e., does not need to return to the start location, then the line topology can be used, as in Figure 2.3a.

The learning algorithm requires several modifications to be suitable for TSP problems. For example, it is necessary to visit all specified locations. Additionally the paths need to visit the exact locations rather than just getting close. For multi-vehicle cases, care needs to be taken to share the workload evenly between the vehicles. For
2.3 Planning algorithms

Figure 2.3 – Examples of self-organising maps with different network topologies. Cyan dots represent 50,000 samples of an input space. The learnt network (red vertices and black edges) is reshaped to best fit the input space. Each network took less than 1 s to compute.

orienteering problems, the travel budgets need to be enforced, and it is unknown in advance how many locations should be visited. In some cases, fixed start and/or end locations are required. Algorithms have been proposed for solving these various issues since the 1980s (Angéniol et al., 1988; Somhom et al., 1997); however, none of the proposed SOM algorithms compete with the best known combinatorial heuristics for the conventional TSP. The advantage of SOM algorithms can be seen in TSP variants where it is required to select observation locations. This is important in problems such as the prize-collecting TSP with neighbourhoods (Faigl and Hollinger, 2014) and orienteering problems (Faigl et al., 2016), where the algorithm implicitly selects sensing locations within continuous regions.

In Chapter 4 we present a new self-organising map algorithm for an active perception
formulation that is a generalisation of the orienteering problem. We borrow techniques from related solutions, but provide new modifications for the issues raised above, such as dealing with budget constraints and sharing the workload between robots. Additionally, in our experiments we provide the first demonstration of how this class of algorithms can be used for robotics tasks with complex perception models.

2.3.6 Planning as inference

Inference and planning are both optimisation problems, and several planning algorithms have been inspired by related techniques for inference. The SOM algorithms described above in Section 2.3.5 are an example of this. In this section we highlight two examples of decentralised planning algorithms inspired by inference techniques, particularly the probability collectives algorithm as it is directly relevant to Chapter 3 and Chapter 6.

Belief propagation

A common way to perform inference for graphical models is to use belief propagation techniques (Koller and Friedman, 2009). These methods aim to efficiently learn the value of unobserved variables in the model by passing “messages” between variables that could directly influence one another (i.e., neighbours in a graphical model). These methods achieve exact inference on simple model topologies, such as trees; for general graphs the methods are not exact, but are very commonly used as an efficient heuristic.

There is a body of work on adapting the max-sum belief propagation variant for decentralised planning, where the value of a node in the graph represents the decision of an individual agent and edges represent pairs of agents with communication links (Stranders et al., 2009; Farinelli et al., 2014). Messages are communicated between agents that indicate the likely decisions of individual agents after taking into account messages received from neighbouring agents. These communication messages have the same format as those used by belief propagation inference methods. These techniques have been applied to multi-robot task allocation problems, such as aerial imagery collection (Delle Fave et al., 2012).
Variational methods

Variational methods are another common class of methods for performing approximate inference of graphical models (Jordan et al., 1999). Variational methods seek to approximate the underlying global likelihood with a collection of structurally simpler distributions that can be evaluated efficiently and independently. These methods often characterise convergence based on the choice of product distribution, and work best when it is possible to strike a balance between the convergence properties of the product distribution and the KL divergence between the product and joint distributions.

As discussed in the body of work on probability collectives (PC) (Wolpert and Bieniawski, 2004; Wolpert et al., 2006, 2013), such variational methods can also be viewed under a game theoretic interpretation. These methods approximate the joint multi-agent action space as the product of independent individual robot action spaces. This approximation is made to facilitate efficient and decentralised planning. Optimisation is performed by defining and updating a probability distribution over the product distribution space. Guarantees are provided that state the resulting product distribution best approximates the optimal joint distribution. At every iteration, each agent communicates a probability distribution that describes its own action. The entropy of the distribution is decreased slowly to avoid becoming trapped in local optima.

PC has been applied to robotics problems, but so far has been limited to selecting a single action from a small action space (Waldock and Nicholson, 2007). However, in robotics we are typically interested in planning sequences of actions, which exponentially increases the size of the search space. A multi-action extension of PC is proposed by Kulkarni and Tai (2010), who combine PC with TSP heuristics to solve the multiple-TSP in a decentralised manner. In Chapter 3, we propose a similar approach to Kulkarni and Tai (2010), but instead we leverage the long-horizon planning of MCTS to dynamically select an effective and compact sample space of action sequences. We provide a full description of our adapted PC algorithm in Section 3.3.4 as a component of our Dec-MCTS algorithm.
2.4 Summary and limitations

This chapter surveyed the literature related to multi-robot active perception and generic planning algorithms for sequential decision making. We began by discussing perception models and objective functions for typical active perception problems. We then discussed methods for informative path planning in single- and multi-robot scenarios. Finally, we introduced several generic planning algorithms to provide background for the algorithms presented in this thesis.

This thesis presents planning algorithms that optimise paths with respect to perception objectives. The literature reviewed in Section 2.1 demonstrates there is a variety of existing perception models suitable as objective functions. In our experimental sections we use several standard perception models to demonstrate the performance of our algorithms. The mission monitoring scenario is a new problem and thus we develop new prediction models and objective functions for this case.

Section 2.2 reviewed existing approaches to informative path planning. The TSP is insufficient at modelling active perception problems, but several variants of the TSP, particularly the GTSP and the OP formulations, can be useful approximations of more complex perception models that admit efficient planning algorithms; we propose a new algorithm for this class of problems in Chapter 4.

While submodularity guarantees can lead to efficient greedy solutions, there is still significant benefits in performing long horizon planning, particularly when path costs are considered. There has been work in developing non-myopic solutions, although few have been demonstrated to work well with rich perception models, and even fewer are applicable to the decentralised multi-robot planning setting. In decentralised settings in particular, it is important to consider the role of communication, either in the objective function or within the actual planning algorithm. Chapter 3 proposes a new generic decentralised planning algorithm with analytical convergence guarantees and addresses these various communication considerations. Chapter 5 and Chapter 6 formulate new communication-aware motion planning problems that are motivated by the important role of communication in monitoring AUV missions. The decentralised planning literature has focussed on developing efficient problem-specific solutions; similarly, our algorithms in Chapter 5 and Chapter 6 are designed to exploit geometric
properties of the mission monitoring problem to efficiently find good solutions.

The new algorithms we propose for robotics problems are strongly motivated by algorithms found in the wider artificial intelligence community for general sequential decision problems; we introduced and reviewed relevant algorithms in Section 2.3. Our algorithm in Chapter 3 particularly builds on MCTS and PC. MCTS has recently become popular in robotics, but we present here the first decentralised generalisation of MCTS. PC has been used for decentralised decision making but has so far been limited to myopic planning. Our algorithm in Chapter 4 is a new variant of SOM that is generalised for centralised multi-robot planning. The SOM literature shows that these algorithms do not perform well at the standard TSP, but are particularly advantageous when it is required to plan directly over continuous space. Our extensive simulated experiments show the applicability of a generalised OP formulation and a new SOM solution algorithm for robotics scenarios. Our algorithm in Chapter 5 solves a new active perception formulation that is a generalisation of the optimal stopping problem to multiple dimensions. Our solution algorithm and associated analysis borrows ideas from computational geometry to efficiently prune and search over the space of trajectories. Finally, Chapter 6 combines ideas from Chapter 5 and Chapter 3 to efficiently solve a new decentralised active perception problem.

Overall, this chapter has introduced relevant background material and identified gaps in the literature in order to set the context of the new multi-robot active perception planning algorithms proposed in this thesis.
Chapter 3

Decentralised Monte Carlo tree search

In this chapter we propose the decentralised Monte Carlo tree search (Dec-MCTS) algorithm as a general decentralised coordination algorithm suitable for any objective function defined over the action sequences of the robots. This chapter addresses a decentralised formulation of the general multi-robot active perception planning problem stated in Problem 1.1, proposes a solution algorithm with strong analytical properties, presents empirical results for several example active perception formulations, and presents an extended algorithm that also considers communication limitations.

3.1 Overview

Dec-MCTS is essentially a novel decentralised variant of Monte Carlo tree search (MCTS). At a high level, the Dec-MCTS algorithm alternates between exploring each robot’s individual action space and optimising a probability distribution over the joint-action space. In any particular iteration of the algorithm, we first use a new variant of MCTS to find locally favourable sequences of actions for each robot. These favourable actions sequences are selected with respect to probabilistic estimates of other robots’ actions that evolve during planning-time. The main novelty is our new tree expansion policy, motivated by discounted-UCB (Garivier and Moulines, 2011), that accounts in general for changing reward distributions.
Next, during each planning iteration, the robots periodically attempt to asynchronously communicate a highly compressed version of their local search trees which, together, correspond to a product distribution approximation of the joint plan. These communicated distributions are used to estimate the underlying joint distribution for the teams’ plan. The estimates are probabilistic, unlike the deterministic representation of joint actions typically used in multi-robot coordination algorithms. Optimising a product distribution is similar in spirit to the mean-field approximation from variational inference, and also has a natural game-theoretic interpretation (Rezek et al., 2008; Wolpert and Bieniawski, 2004).

Dec-MCTS is a powerful new method of decentralised coordination for any objective function defined over the robot action sequences. Notably, this implies that Dec-MCTS is suitable for complex perception tasks that are highly viewpoint-dependent, which are the motivation for this thesis. Further, communication is assumed to be intermittent, and the amount of data sent over the network is small in comparison to the raw data generated by typical range sensors and cameras. Our method also inherits important properties from MCTS, such as the ability to compute anytime solutions and to incorporate prior knowledge about the environment. Moreover, our method is suitable for online replanning to adapt to changes in the objective function or team behaviour.

We provide an extensive theoretical analysis of the algorithm that leverages results from probability theory and game theory. Our main analytical result is to show convergence rates for the expected payoff at the root of the search tree towards the optimal payoff sequence. Thus, the proposed MCTS tree expansion policy balances exploration and exploitation while the reward distributions are changing. We prove this result in Best et al. (2018a) by extending the MCTS analysis of Kocsis et al. (2006) for the context of switching bandit problems (Garivier and Moulines, 2011). Our second result leverages Wolpert et al. (2006) to show that the product distribution optimisation phase locally minimises the KL divergence to the optimal joint probability distribution. While, given the difficulty of the problem, these results do not directly yield guarantees for global optimality, the analysis provides strong motivation for the use of these components in our algorithm for decentralised, long-horizon planning with general objective function definitions.
We empirically evaluate our algorithm in two scenarios: generalised team orienteering and online active object recognition. These experiments are run in simulation, where the robots traverse a PRM (Kavraki et al., 1996) with a Dubins motion model (Dubins, 1957), and the second scenario uses range sensor data collected a priori by real robots. We show that our decentralised approach performs as well as or better than centralised MCTS even with a significant rate of communication message loss. We also show the benefits of our algorithm in performing long-horizon and online planning.

Further empirical analyses for Dec-MCTS are also presented in later chapters: the results in Chapter 4 compare Dec-MCTS to our proposed SOM algorithm, and the results in Chapter 6 compare Dec-MCTS to the proposed decentralised mission monitoring algorithm that is motivated by Dec-MCTS.

Communication is fundamental to the coordinated behaviour that emerges from Dec-MCTS and other similar decentralised planning algorithms (Farinelli et al., 2008), as robots need to develop decision strategies that take into account the actions of other robots. However, communication is typically considered to be an infinite resource, but in practice communication is often limited, unreliable, or susceptible to interference (Hollinger et al., 2011b; Williamson et al., 2008; Fitch et al., 2017). In the experiments of Section 3.5 we show that Dec-MCTS is robust to communication loss such that reasonable task performance is achieved even if communication packets are lost. In Section 3.7, we take this one step further by explicitly planning how to effectively use communication resources. We present a communication scheduling algorithm as an extension of Dec-MCTS that aims to mitigate these various communication issues, while also maintaining task performance.

### 3.1.1 Chapter outline

The remainder of this chapter is organised as follows. Section 3.2 formally defines the decentralised planning problem considered in this chapter. Section 3.3 presents our proposed Dec-MCTS algorithm. Section 3.4 provides a theoretical analysis of Dec-MCTS, based on our extended results and proofs presented in Best et al. (2018a). Sections 3.5 and 3.6 present an empirical analysis of our algorithm for two example
active perception problems. Section 3.7 presents an algorithm for scheduling communication that can be used within Dec-MCTS for scenarios where communication bandwidth is severely limited. Finally, Section 3.8 summarises the chapter.

3.2 Problem statement

We consider a team of $R$ robots $\{1, 2, \ldots, R\}$, where each robot $r$ plans its own sequence of future actions $x_r = (x^r_1, x^r_2, \ldots)$. Each action $x^r_j$ has an associated cost $c^r_j$ and each robot has a cost budget $B^r$ such that the sum of the costs must be less than the budget, i.e., $\sum_{x^r_j \in x_r} c^r_j \leq B^r$. This cost budget may be an energy or time constraint defined by the application, or it may be used to enforce a planning horizon. The feasible set of actions and associated costs at each step $j$ are a function of the previous actions $(x^r_1, x^r_2, \ldots, x^r_{j-1})$. Thus, there is a predefined set $X^r$ of feasible action sequences $x^r$ for each robot $r$. We denote $x$ as the set of action sequences for all robots $x := \{x^1, x^2, \ldots, x^R\}$ and $x^{(r)}$ as the set of action sequences for all robots except robot $r$, i.e., $x^{(r)} := x \setminus x^r$. We denote $X$ as the set of all feasible $x$ and $X^{(r)}$ as the set of all feasible $x^{(r)}$.

The aim is to maximise a global objective function $g(x)$ that is a function of the set of action sequences $x$ for the robots. This function $g$ encodes the objectives of the perception task at hand; we later provide example definitions for generalised coverage tasks (Section 3.5) and active object recognition (Section 3.6). The problem of planning the action sequences of all robots must be solved in a decentralised setting.

We assume each robot $r$ knows the global objective function $g$, but does not know the action sequences $x^{(r)}$ selected by the other robots.

Thus, the problem to be solved by each robot $r$ is stated as follows.

**Problem 3.1** (Decentralised planning problem). Plan the action sequence $x^r$ such that a global objective function $g(x^r \cup x^{(r)})$ is maximised. This problem should be solved while considering the unknown action sequences $x^{(r)}$ of the other robots.

This problem is a decentralised formulation of the general active perception planning problem stated earlier in Problem 1.1.
We assume that robots can asynchronously communicate during planning-time to improve coordination. In particular, communication may be used to learn the action sequences $\mathbf{x}^{(r)}$ planned by other robots. However, the communication channel may be unpredictable and intermittent, and therefore should not be relied upon. Thus, each robot will plan based on the information it has available locally. Bandwidth may be limited and therefore message sizes should remain small, even as the plans grow. We address communication limitations further in Section 3.7. Although we do not consider explicitly planning to maintain communication connectivity, this may be encoded in the objective function $g(\mathbf{x})$ if a reliable communication model is available.

When presenting our proposed approach, we assume $g$ is deterministic given a known set of action sequences $\mathbf{x}$; in Section 3.3.7 we discuss potential extensions for probabilistic objective functions.

### 3.3 Dec-MCTS

In this section, we present our Dec-MCTS algorithm as a decentralised solution to the general multi-robot planning problem. We first provide an overview of the algorithm and relevant notation, followed by a detailed explanation of all components.

#### 3.3.1 Algorithm overview

**Planning cycle**

Dec-MCTS runs simultaneously and asynchronously on all robots; we present the algorithm from the perspective of robot $r$. The algorithm cycles between the three phases illustrated in Figure 3.1: (1) incrementally grow a search tree using MCTS while taking into account information about the other robots’ plans, (2) update the probability distribution over possible action sequences, and (3) communicate probability distributions with the other robots. These three phases continue regardless of whether or not the communication was successful, until a computation budget is met.
Probabilistic plan representation

A key idea of Dec-MCTS is to represent and reason over plans in a probabilistic manner. In particular, robot $r$’s current plan is represented by a probability distribution over action sequences. We define a probability mass function $q_r^n$, such that $q_r^n(x^r)$ defines the probability that robot $r$ will select the action sequence $x^r$. In general, the domain of the distribution $q_r^n$ is the set of all possible action sequences $\mathcal{X}^r$. However, to enable tractable computation and realistic communication, we restrict the domain of $q_r^n$ to a dynamically selected subset $\hat{\mathcal{X}}^r_n \subset \mathcal{X}^r$, i.e., $q_r^n(x^r) = 0, \forall x^r \notin \hat{\mathcal{X}}^r_n$. As the Dec-MCTS algorithm progresses, both the domain $\hat{\mathcal{X}}^r_n$ and the probability distribution $q_r^n$ are optimised. Note the subscript $n$ for $q_r^n$ and $\hat{\mathcal{X}}^r_n$ is used to denote the $n$th iteration of the main loop of our algorithm.

Key components

An illustration of the main loop is shown in Figure 3.1 and pseudocode for the algorithm is provided in Algorithm 3.1. During the MCTS phase, a search tree $T^r$ is grown over the space $\mathcal{X}^r$ of robot $r$’s action sequences using a new variant of the UCT algorithm. This tree growth is performed while considering the probability distributions over the other robots plans, denoted $\hat{\mathcal{X}}^{(r)}_n, q_n^{(r)}$. Periodically, the domain $\hat{\mathcal{X}}^r_n$ for robot $r$’s distribution is updated by selecting the most promising action sequences
Algorithm 3.1 Overview of Dec-MCTS for robot \( r \).

\begin{itemize}
  \item \textbf{input:} global objective function \( g \), budget \( B^r \), feasible action sequences and costs \\
  \item \textbf{output:} sequence of actions \( x^r \) for robot \( r \)
\end{itemize}

1: \( \mathcal{T}^r \leftarrow \text{initialise MCTS tree} \)
2: \textbf{while} computation budget not met, at iteration \( n \) \textbf{do}
3: \( \hat{X}^r_n \leftarrow \text{SelectSetOfSequences}(\mathcal{T}^r) \) \hspace{3em} \( \triangleright \) See Section 3.3.4
4: \textbf{for} \( \tau_n \) iterations \textbf{do}
5: \( \mathcal{T}^r \leftarrow \text{GrowTree}(\mathcal{T}^r, \hat{X}^r_n, q^r_n, B^r) \) \hspace{3em} \( \triangleright \) See Algorithm 3.2 and Section 3.3.3
6: \( q^r_n \leftarrow \text{UpdateDistribution}(\hat{X}^r_n, q^r_n, \hat{X}^r_n, q^r_n, \beta) \) \hspace{3em} \( \triangleright \) See Section 3.3.4
7: \( \text{CommunicationTransmit}(\hat{X}^r_n, q^r_n) \) \hspace{3em} \( \triangleright \) See Section 3.3.5
8: \( (\hat{X}^r_n, q^r_n) \leftarrow \text{CommunicationReceive} \) \hspace{3em} \( \triangleright \) See Section 3.3.5
9: \( \beta \leftarrow \text{cool}(\beta) \) \hspace{3em} \( \triangleright \) See Section 3.3.4
10: \textbf{return} \( x^r \leftarrow \arg \max_{x^r \in \hat{X}^r_n} [q^r_n(x^r)] \)

identified by the tree search.

In the probability distribution optimisation phase, the probabilities assigned to action sequences \( q^r_n(x^r) \) are optimised using a decentralised gradient descent algorithm while considering the distributions of the other robots. In the communication phase, robot \( r \) communicates its domain \( \hat{X}^r_n \) and probability distribution \( q^r_n \) to the other robots. If robot \( r \) receives a new distribution from any of the other robots, then in the next iteration \( \hat{X}^r_n \) and \( q^r_n \) are optimised while considering this new information. During this optimisation process, it is possible that \( q^r_n \) will change such that a previously optimal leaf of the tree \( \mathcal{T}^r \) becomes suboptimal; we refer to the times at which this happens as breakpoints.

Termination

When the computation budget is met, the algorithm returns the action sequence \( x^r \) that has the highest probability \( q^r_n(x^r) \). In online settings, the robot would then typically execute the first action \( x^r_1 \) in the action sequence, and then perform replanning to take into account new information received by observations. If the changes to the objective function are minor, then replanning may be performed more efficiently by adapting the previous search tree.
3.3 Dec-MCTS

3.3.2 Local utility function

The global objective function $g$ is optimised by each robot $r$ using a local utility function $f^r$. We define $f^r$ as the difference in global utility between robot $r$ performing action sequence $x^r$ and a default “no reward” sequence $x^r_\emptyset$, assuming fixed action sequences $x^{(r)}$ for the other robots, i.e.,

$$f^r(x) := g(x^r \cup x^{(r)}) - g(x^r_\emptyset \cup x^{(r)}). \quad (3.1)$$

The default sequence $x^r_\emptyset$ is chosen to be suitable for the application and would typically be an empty action sequence. In practice, optimising with respect to $f^r$ rather than $g$ improves the performance since $f^r$ is more sensitive to robot $r$’s plan and the variance of $f^r$ is less affected by the uncertainty of the other robots’ plans (Wolpert et al., 2013).

We chose this local utility function since it is generally applicable, although further performance improvements could be achieved with problem-specific heuristics (Rahmattalabi et al., 2016). We also note that this formulation assumes that all robots know the global utility function $g$. However, if instead each robot only has access to a local estimate of $g$ then our proposed algorithm will optimise the action sequences with respect to this inconsistent information.

3.3.3 Monte Carlo tree search with discounted-UCB

The first phase of the algorithm incrementally grows a search tree using a new variant of MCTS, as outlined in Algorithm 3.2. A single search tree $\mathcal{T}^r$ is maintained by robot $r$ which only contains the actions of robot $r$. The tree $\mathcal{T}^r$ is defined such that each edge in the tree represents an action by robot $r$, and a path from the root node $i_0$ to another node $i_d$ at depth $d$ represents a valid sequence of actions by robot $r$. The MCTS algorithm incrementally grows $\mathcal{T}^r$ from the root node using a best-first expansion policy (Section 2.3.4 presents an illustrated discussion of this best-first expansion). During the MCTS phase, coordination with other robots occurs implicitly by considering the plans of the other robots when performing the rollout policy and evaluation of the global objective function. This information about the other robots’
Algorithm 3.2 Grow the search tree for robot $r$ using Monte Carlo tree search. The four phases are illustrated in Figure 3.2 and described in the Section 3.3.3.

1: `function GrowTree($T^r, \hat{\lambda}_n^{(r)}, q_n^{(r)}, B^r$)
   `input: partial tree $T^r$, distributions for other robots ($\hat{\lambda}_n^{(r)}, q_n^{(r)}$), budget $B^r$
   `output: updated partial tree $T^r$
2: `for` fixed number of samples `do`
3:   `▷ Select node to expand using D-UCT policy (Section 3.3.3)`
4:   $i_{d-1} \leftarrow \text{NodeSelectionD-UCT} (T^r)$
5:   `▷ Add new child to node $i_{d-1}$`
6:   $i_d \leftarrow \text{ExpandTree} (i_{d-1})$
7:   `▷ Evaluate node`
8:   $x^{(r)} \leftarrow \text{Sample} (\hat{\lambda}_n^{(r)}, q_n^{(r)})$ `▷ Sample action sequences of other robots`
9:   $x^{(r)} \leftarrow \text{PerformRolloutPolicy} (i_d, x^{(r)}, B^r)$ `▷ Default policy`
10:  $F_t \leftarrow f' (x^{(r)} \cup x^{(r)})$ `▷ Local utility function (Section 3.3.2)`
11: `▷ Update statistics in tree`
12: $T^r \leftarrow \text{Backpropagation} (T^r, i_d, F_t)$
13: `return $T^r$`

Figure 3.2 – Overview of the four phases of standard MCTS. Our new MCTS variant follows this same general procedure. Diagram adapted from Hefferan et al. (2016) and Patten et al. (2018).

plans comes from the second phase of the algorithm, detailed later in Section 3.3.4. In this subsection, we detail our proposed MCTS algorithm which features a novel bandit-based node selection policy designed for our planning scenario.
Standard MCTS incrementally grows a tree by iterating through four phases, as depicted in Figure 3.2: selection, expansion, simulation and backpropagation (Browne et al., 2012). During each iteration $t$, a new leaf node is added, where each node represents a sequence of actions and contains statistics about the expected reward of all action sequences that begin with this sequence.

The selection phase (Algorithm 3.2, line 4) selects an expandable node in the tree, where an expandable node is defined as a node that has at least one child that has not yet been visited during the search. In order to find an expandable node, the algorithm begins at the root node $i_0$ of the tree and recursively selects child nodes until an expandable node $i_{d-1}$ is reached. For selecting the next child at each level of the tree, we propose an extension of the UCT policy (Kocsis and Szepesvári, 2006), detailed later in Section 3.3.3, to balance exploration and exploitation of the search space in this modified problem setting. In the expansion phase (Algorithm 3.2, line 6), a new child node $i_d$ is added to the selected expandable node $i_{d-1}$, which extends the parent’s action sequence with an additional action.

In the simulation phase (Algorithm 3.2, lines 8–10), the expected utility $E[g]$ of the expanded node $i_d$ is estimated by performing and evaluating a rollout policy that extends the action sequence represented by the node until a terminal state is reached. This rollout policy could be a random policy or a heuristic for the problem (James et al., 2017). The objective is evaluated for this sequence of actions and this result is saved.

For our problem, the objective is a function of the action sequence $x^r$ as well as the unknown plans of the other robots $x^{(r)}$, and thus we require an extension of the standard simulation procedure. To compute the rollout score, we first sample $x^{(r)}$ from a probability distribution $q_n^{(r)}$ over the plans of the other robots (as defined in Section 3.3.1). A heuristic rollout policy extended from $i_d$ defines $x^r$, which should be a function of $x^{(r)}$ to simulate coordination between the robots. Additionally, we optimise $x^r$ using the local utility $f^r$ (as defined in (3.1)) rather than $g$. The rollout score is computed as the utility of this joint sample $f^r(x^r \cup x^{(r)})$, which is an estimate for $E_{q_n}[f^r \mid x^r]$. We denote $F_t$ as the rollout evaluation at sample round $t$.

In the backpropagation phase (Algorithm 3.2, line 12), the rollout evaluation is added to the statistics of all nodes along the path from the expanded node back to the root of
the tree. Typically, these statistics are unbiased estimators of the rollout evaluations; however, as we discuss in the following section, it is more suitable to use a weighted average in the context of Algorithm 3.1.

**D-UCB node selection policy**

The main difference between our MCTS variant and the well-known UCT (Kocsis and Szepesvári, 2006) algorithm is that our tree search allows for changing reward distributions, which makes it suitable for our decentralised problem setting. This is achieved using a novel node selection policy described as follows. We recover similar analytical results to the UCT algorithm for this generalised case, which we detail later in Section 3.4.

The node selection policy is used in Algorithm 3.2, line 4, and dictates the order in which the tree $T_v$ is expanded. Consider an arbitrary node $i_d$ at depth $d$ in the tree which has an associated set of child nodes $C(i_d)$. For every sample round $t$ where node $i_d$ is visited, the problem is to select a child $I_{i_d,t} \in C(i_d)$ that balances both visiting promising subtrees and exploring uncertain ones.

An established approach for node selection is based on maintaining an upper confidence bound (UCB) on the value of each node. Under this paradigm, at each sample round $t$, a UCB $U_{j,t_{i_d},t_j}$ is computed for all children $j \in C(i_d)$ of the parent node $i_d$. Here, $t_{i_d}$ is the number of times the parent node $i_d$ has been visited and $t_j$ is the number of times child node $j$ has been visited. The algorithm then selects the node that maximises this quantity, i.e.,

$$I_{i_d,t} = \arg \max_{j \in C(i_d)} U_{j,t_{i_d},t_j}. \quad (3.2)$$

This continues recursively until an expandable node is reached.

The de facto UCB $U_{j,t_{i_d},t_j}$ is a combination of the empirical mean of rewards received at node $j$ and a confidence interval derived from the Chernoff-Hoeffding inequality (Browne et al., 2012). This bound was originally used in the context of the MAB problem and called UCB1 (Auer et al., 2002); when used for tree search, it is labelled UCT (Kocsis and Szepesvári, 2006). UCT was shown to yield polynomial
regret when the reward distributions at the leaf nodes are stationary (Kocsis and Szepesvári, 2006). However, Algorithm 3.1 alternates between growing the tree for a number of rollouts $\tau_n$ and updating the probability distributions for other robots. As mentioned in Section 3.3.1, this introduces breakpoints as instants where the reward distribution and optimal action can change abruptly. We denote the number of breakpoints up until time $t$ as $T_t$. Due to these breakpoints, the most recent rollouts are more relevant since they are obtained by sampling the most recent distributions. It was shown by Garivier and Moulines (2011) that UCB1 is inefficient in the bandit setting when breakpoints are expected. In this scenario a discounted variant, termed D-UCB, yields tighter bounds on regret. Due to the expected breakpoints caused by updating the distributions, we extend the approach of Garivier and Moulines (2011) for tree search, and propose a discounted variant of UCT for node selection, which we term D-UCT, described as follows.

Given some discount factor $\gamma \in (1/2,1)$ and exploration constant $C_p > 1/\sqrt{8}$, the D-UCT bound is defined as:

$$U_{j,t_d,t_j}(\gamma) := \bar{F}_{j,t_j}(\gamma) + c_{t_d,t_j}(\gamma),$$  \hspace{1cm} (3.3)

where $t_j$ is the number of times node $j$ has been visited, $\bar{F}_{j,t_j}(\gamma)$ is the discounted empirical reward, and $c_{t_d,t_j}(\gamma)$ is a discounted exploration bonus. A lower discount factor $\gamma$ enforces only the most recent rollouts to contribute towards the UCB, whereas at the upper limit $\gamma \to 1$ D-UCT becomes equivalent to UCT. These quantities are computed as follows. First, recall that the indicator function $1_{\{I_{id,t}=j\}}$ returns 1 if node $j$ was selected at round $t$, and 0 otherwise. Then, denote the discounted number of times the child node $j$ has been visited as:

$$t_j(\gamma) := \sum_{u=1}^{t} \gamma^{t-u} 1_{\{I_{id,u}=j\}},$$  \hspace{1cm} (3.4)

and the discounted number of times the parent node has been visited as:

$$t_{i_d}(\gamma) := \sum_{j \in C(i_d)} t_j(\gamma).$$  \hspace{1cm} (3.5)

Recall that $F_t$ is the rollout score received at sample $t$. Then, the discounted empirical
3.3 Dec-MCTS

The average is given by:

\[ \tilde{F}_{j,t}(\gamma) := \frac{1}{t_j(\gamma)} \sum_{u=1}^{t} \gamma^{t-u} F_u 1\{t_d,u=j\}, \quad (3.6) \]

and the discounted exploration bonus is defined as:

\[ c_{t_d,t_j}(\gamma) := 2C_p \sqrt{\log \frac{t_d(\gamma)}{t_j(\gamma)}}. \quad (3.7) \]

In Remark 2 of Best et al. (2018a) we offer an example definition of \( \gamma \) as a function of the expected number of breakpoints, which results in interesting analytical convergence properties. However, having \( \gamma \) change dynamically makes it difficult to efficiently recompute \( \tilde{F}_{j,t} \) and \( c_{t_d,t_j} \) as \( t \) grows large, since the sums cannot be computed incrementally. Therefore, a more practical definition for \( \gamma \) is to set it as a fixed constant; this is unlikely to detrimentally affect performance in practice.

### 3.3.4 Decentralised product distribution optimisation

The second phase of Dec-MCTS updates a probability distribution \( q_r^n \) over the set of possible action sequences for robot \( r \). The distribution \( q_r^n \) serves as a way of predicting the likelihood of an action sequence being selected as the search tree continues to grow. These distributions are communicated between robots and used when performing rollouts during future iterations of MCTS.

To define and optimise these distributions in a decentralised manner for improving global utility, we adapt a type of variational method originally proposed by Wolpert and Bieniawski (2004). This formulation can be viewed as a game between independent robots, where each robot selects its action sequence by sampling from a distribution. The approach to solving this formulation is essentially a decentralised gradient descent method over the space of product distributions. We describe the approach in detail as follows.
Sample space selection

One challenge is that the set of possible action sequences $X^r$ typically has a cardinality that is exponential in the time horizon. We obtain a sparse representation by periodically selecting the sample space $\hat{X}_n^r \subset X^r$ as the most promising action sequences \{\(x_1^r, x_2^r, \ldots\)} found by MCTS so far (Algorithm 3.1, line 3). We select a fixed number of nodes in the search tree $T^r$ that currently have the highest discounted empirical average $\bar{F}$. The set $\hat{X}_n^r$ is chosen as the action sequences used during the initial rollouts when the selected nodes were first expanded.

We chose this method for selecting $X^r$ since it provides a reasonable way predicting what the best path may be at future iterations of the planning algorithm. This allows future iterations of MCTS to plan with respect to more relevant information about the robots. Predicting future best paths accurately is as difficult as the entire planning problem, and thus we argue this sample space selection method is a reasonable approximation. Other possible methods could also be appropriate; for example, in Chapter 6 we present a different but closely related decentralised algorithm, where instead we use a sliding history of best paths for the purpose of sample space selection in a different context (see Section 6.4.3).

As mentioned in Section 3.3.1, when the sample spaces $\hat{X}_n^{(r)}$ are updated, this can introduce breakpoints in the reward distribution of robot $r$. Thus, we expect the maximum number of breakpoints $\mathbb{E}[\Upsilon_t]$ to be given by the number of changes to the sample spaces. To ensure convergence of the utility, each period $\tau_n$ should be governed by a function of $t$ such that $\{\mathbb{E}[\Upsilon_t]\}_t$ is bounded from above (Best et al., 2018a, Remark 2).

Probability collectives

The set $\hat{X}_n^r$ has an associated probability distribution $q_n^r$ such that $q_n^r(x^r)$ defines the probability that robot $r$ will select $x^r \in \hat{X}_n^r$. The distributions for different robots are independent and therefore they collectively define a product distribution $q_n$, such that the probability $p_n$ of a joint action sequence selection $x$ is

$$p_n(x) = q_n(x) := \prod_{r \in \{1, \ldots, R\}} q_n^r(x^r).$$ (3.8)
The advantage of defining \( p_n \) as a product distribution is so that each robot selects its action sequence independently, and therefore allows decentralised execution.

Consider the general class of joint probability distributions \( p_n \) that are not restricted to product distributions. Define the expected global objective function for a joint distribution \( p_n \) as \( E_{p_n}[g] \), and let \( \Gamma \) be a desired value for \( E_{p_n}[g] \). According to the maximum entropy principle from information theory, the most likely \( p_n \) that satisfies \( E[g] = \Gamma \) is the \( p_n \) that maximises entropy. The most likely \( p_n \) can be found by minimising the maxent Lagrangian, defined as

\[
L(p_n) := \lambda(\Gamma - E_{p_n}[g]) - H(p_n),
\]

where

\[
H(p_n) := -\sum_{x \in X} p_n(x) \ln(p_n(x))
\]

is the Shannon entropy and \( \lambda \) is a Lagrange multiplier. The intuition is to iteratively increase \( \Gamma \) and optimise \( p_n \). A descent scheme for \( p_n \) can be formulated with Newton’s method.

For decentralised planning and execution, we are interested in optimising the product distribution \( q_n \) rather than a more general joint distribution \( p_n \). We can approximate \( q_n \) by finding the \( q_n \) with the minimum \( pq \) KL divergence, where the \( pq \) KL divergence is defined as

\[
D_{KL}(p_n \parallel q_n) := \sum_{x \in X} p_n(x) \ln \left( \frac{p_n(x)}{q_n(x)} \right).
\]

By making a second-order approximation, a descent scheme can be formulated (Wolpert and Bieniawski, 2004) with the update policy for \( q_n \) shown in Algorithm 3.3, line 9. Here we use \( f^r \) (as defined in (3.1)) rather than \( g \), and the expectations \( E_{q_n} \) are defined with respect to the product distribution \( q_n \). Intuitively, this update rule increases the probability that robot \( r \) selects \( x^r \) if this results in an improved local utility, while also ensuring the entropy of \( q_n^r \) does not decrease too rapidly. The former behaviour is controlled by the \( (E_{q_n}[f^r] - E_{q_n}[f^r | x^r]) / \beta \) term in the update rule, while the latter behaviour is controlled by \( H(q_n^r) + \ln(q_n^r(x^r)) \). Parameter \( \beta \) specifies the balance between these two behaviours.
Algorithm 3.3 Probability distribution optimisation for robot \( r \)

1: function UpdateDistribution(\( \hat{X}^r_n, q^r_n, \hat{X}^r(R), q^r(R), \beta \))

   input: action sequence set for each robot \( \hat{X}^r_n := \{ \hat{X}^r_1, \hat{X}^r_2, \ldots, \hat{X}^r_R \} \)
   with associated probability distributions \( \{ q^r_1, q^r_2, \ldots, q^r_R \} \),
   update parameter \( \beta \)

   output: updated probability distribution \( q^r_n \) for robot \( r \)

2:     \( \triangleright \) Consider each action sequence \( x^r \) that robot \( r \) may select
3:     for each \( x^r \in \hat{X}^r_n \) do
4:         \( \triangleright \) Compute expected global utility
5:         \( E_{q^r_n}[f^r] \leftarrow \sum_{x \in \hat{X}_n} f^r(x) \prod_{r' \in \{1, \ldots, R\}} q^r_n(x_{r'}) \)
6:         \( \triangleright \) Compute expected global utility given robot \( r \) selects \( x^r \)
7:         \( E_{q^r_n}[f^r \mid x^r] \leftarrow \sum_{x \in \hat{X}_n} f^r(x_{r'} \cup x^r) \prod_{r' \in \{1, \ldots, R\} \setminus r} q^r_n(x_{r'}) \)
8:         \( \triangleright \) Update the probability of selecting \( x^r \)
9:         \( q^r_n(x^r) \leftarrow q^r_n(x^r) - \alpha q^r_n(x^r) \left[ \frac{E_{q^r_n}[f^r] - E_{q^r_n}[f^r \mid x^r]}{\beta} + H(q^r_n) + \ln(q^r_n(x^r)) \right] \)
10:    \( \triangleright \) Re-normalise the probability distribution
11:    \( q^r_n \leftarrow \text{Normalise}(q^r_n) \)
12:    return \( q^r_n \)

Implementation issues

Pseudocode for this approach is in Algorithm 3.3. Each iteration of the loop beginning at line 3 updates the probability \( q^r_n(x^r) \) of performing an action sequence \( x^r \). We require computing two expectations (lines 5 and 7) to evaluate the update equation (line 9). In general, to compute these expectations exactly it is necessary to sum over the enumeration of all \( x \in \hat{X}_n \). It is infeasible to perform this enumeration at every iteration, and therefore these expectations should instead be approximated using random sampling of \( \hat{X}_n \). For certain problem definitions, it may be possible to efficiently compute these expectations exactly by exploiting the structure of the problem, such as in our Section 3.5 experiments.

As the Dec-MCTS algorithm progresses, the parameter \( \beta \) should slowly decrease in order to slowly decrease the entropy of the probability distributions. The cooling schedule for \( \beta \) could be a fixed rate of descent or a more elaborate schedule (Wolpert et al., 2006). The parameter \( \alpha \) is a fixed step-size. When the sample space \( \hat{X}_n \) changes (Algorithm 3.1, line 3), theoretically it is possible to keep and update the previous
distribution, i.e., \( q_n^r = q_{n-1}^r \), by maintaining \( q_n^r \) over the entire space \( \mathcal{X}^r \). However, in practice, this is likely to become inefficient as the number of action sequences that have ever appeared in a sample space grows, particularly when calculating the expectations and normalising, as well as when communicating these distributions. Instead, we suggest resetting \( q_n^r \) to a uniform distribution and \( \beta \) to its initial value whenever \( \hat{X}_n^r \) changes.

### 3.3.5 Communication

At each iteration of the inner-loop of Algorithm 3.1, robot \( r \) communicates its current probability distribution \((\hat{X}_n^r, q_n^r)\) to the other robots. If robot \( r \) receives an updated distribution \((\hat{X}_{n'}^r, q_{n'}^r)\) from another robot \( r' \), then \((\hat{X}_{n'}^r, q_{n'}^r)\) replaces the locally stored distribution for \( r' \). The updated distribution is used during the next iteration, such that both the tree \( T^r \) and probability distribution \((\hat{X}_n^r, q_n^r)\) are updated based on the new \((\hat{X}_{n'}^r, q_{n'}^r)\). If no new messages are received from a robot, then robot \( r \) continues to plan based on the most recent distribution. If robot \( r \) is yet to receive any messages then it may assume a default policy.

Communicating plans between robots at every iteration will usually be feasible since message sizes are much smaller than other data typically communicated around robotic networks, such as high-bandwidth sensor data. Additionally, later in Section 3.5 we show that the approach is robust to random packet loss. However, in severely constrained communication scenarios it may be beneficial or necessary to explicitly plan when to communicate, rather than broadcasting messages naively at every iterations; we propose an extended algorithm for these scenarios later in Section 3.7.

### 3.3.6 Online replanning

The best action is selected as the first action in the highest probability action sequence in \( \hat{X}_n^r \) (Algorithm 3.1, line 10). The search tree may then be pruned by removing all children of the root except the selected action. Planning may then continue while using the sub-tree’s previous results. If the objective function changes, e.g., as a result
of a new observation, then the tree should be restarted. In practice, if the change is minor then it may be appropriate to continue planning with the current tree, and the discounting in D-UCT will help to quickly correct the reward estimates.

### 3.3.7 Probabilistic objective functions

So far, we have assumed the objective function $g$ is deterministic for a given set of action sequences $x$. This is reasonable in many scenarios since, for example, it is usually sufficient to plan based on the expectation of the reward, which is a deterministic quantity. However, sometimes it may be necessary to directly model other sources of uncertainty, such as the state of the environment, in addition to the uncertain plans of the robots. For these problems, we can define the objective function as $g(x, \Psi)$, where $\Psi$ is a random variable representing other sources of uncertainty. Our algorithm can readily be extended to this case by computing all expectations with respect to both $q_n$ and $\Psi$. In some cases these expectations could be computed exactly (this was feasible for our Section 3.6 experiments), but in general the expectations can be efficiently approximated by sampling for $\Psi$, as in POMCP (Silver and Veness, 2010; Patten et al., 2018). Our theoretical analysis (see Section 3.4) is valid for these cases since the standard UCT algorithm assumes the rewards obtained at leaf nodes are probabilistic (Kocsis and Szepesvári, 2006), and the standard probability collectives algorithm is applicable if there is noise in the system (Wolpert et al., 2006).

### 3.4 Analysis

In this section, we provide a summary of our theoretical analysis of Dec-MCTS that was presented in Best et al. (2018a). The algorithm is an anytime and decentralised approach to multi-robot coordination with two key algorithmic components: (1) the tree search (Section 3.3.3) is designed to perform long-horizon planning for single-robot action sequences while considering the changing plans of the other robots, and (2) the product distribution optimisation (Section 3.3.4) is designed to directly optimise the joint multi-robot plan while being restricted to a small subset of possible action sequences. While it is difficult to make any strong claims of global optimality
3.4 Analysis

in the context of decentralised, long-horizon planning with general objective functions, we focus our analysis on characterising the convergence properties of these two algorithmic components, then discuss the implications of these results in the context Dec-MCTS.

3.4.1 D-UCB applied to trees

Our main analytical result for Dec-MCTS is that the D-UCT algorithm (Algorithm 3.2) maintains an exploration-exploitation trade-off for child selection while the distributions $q_n^r$ are changing and converging. The main insight of this result is to relate D-UCT to D-UCB (Garivier and Moulines, 2011) in the context of a specific type of non-stationary, switching bandit problem. We refer the interested reader to Best et al. (2018a) for full details of this result.

The node selection problem at each node in the tree is equivalent to a bandit problem with special assumptions on the payoff received. From the perspective of node $i_d$, after selecting node $I_{i_d,t} = j$, the tree search further down the tree (e.g., $I_{j,t}$) and subsequent MCTS rollout yield a stochastic payoff $F_{j,t} = F_t \in [0, 1]$. As nodes are expanded in the tree search, the expected reward at any node higher up the tree slowly drifts until all nodes are explored in the subtree (Best et al., 2018a, Assumption 3).

The sequence of payoffs received generates the stochastic process $\{F_{j,t}\}_t$, $\forall j \in C(i_d)$ and $t \geq 1$. We make the simplifying assumption of a constant branching factor $K$ in the search tree $\mathcal{T}$, i.e., $C(i_d) = \{1, \ldots, K\}$, $\forall i_d$.

Recall that $\bar{F}_{i_d,t_{i_d}}$ is the empirical mean; it follows that $\bar{F}_{i_0,t_0}$ is the mean at the root node. Further, let $\mu_{i_0}^*$ denote the optimal expected payoff at the root node and note that $t_0 = t$.

**Theorem 3.1** (Convergence rate of D-UCT). (Best et al., 2018a, Theorem 1) Consider the D-UCT algorithm running on a tree $\mathcal{T}$ of depth $D$ and branching factor $K$. The payoff distributions of the leaf nodes are independently distributed and can change at breakpoints. The sequence that gives the expected bound of breakpoints $\{\mathbb{E}[T_{i_d}]\}$ follows Assumption 2 of Best et al. (2018a) and $\gamma_{t_j} = 1 - \sqrt{\mathbb{E}[T_{t_j}]/16t_j}$ for all nodes $j$. Then, after some time $T_0$, the bias of the payoff at the root node
\[ |\tilde{F}_{t_0 \epsilon t_0} - \mu^*_{t_0}| = \mathcal{O} \left( K D \log(t) \sqrt{E[\gamma_t]/t} \right). \] Further, the probability of selecting a suboptimal arm at the root node becomes zero as \( t \) grows large.

This result is proven in Best et al. (2018a) by relating the payoff sequences at the root node to a bandit problem, and performing induction on \( D \).

Remark 3.1 (Convergence in practice). The results of Theorem 3.1 are mainly concerned with the convergence of the bias after some transitory period. For the standard UCT case, Kocsis et al. (2006) assumed the number of iterations for the transitory period was \( \mathcal{O}(K^D) \). However, it was recently shown that this transitory period using the UCT algorithm on a binary tree (\( K = 2 \)) of depth \( D \) can be \( \Omega(\exp(\exp(\ldots\exp(1)\ldots))) \) (\( D - 1 \) nested exponentials) in a worst-case instance (Coquelin and Munos, 2007). Gelly et al. (2012) suggest instead that the UCT (and thus D-UCT) strategy will be most successful when the leaves of large subtrees share similar rewards, i.e., a “smoothness” assumption on the reward distributions. Active perception scenarios typically exhibit some degree of “smoothness”, such that similar sequences of actions yield similar rewards and thus there is a correlation amongst subtree leaves.

3.4.2 Variational methods by importance sampling

We now consider the effect of contracting the sample space \( \hat{X}_n \subset X \) on the convergence of Algorithm 3.3. Recall that the \( pq \) KL divergence is the divergence from a product distribution \( q_n \) to the optimal joint distribution \( p_n \). We then have the following proposition:

Proposition 3.1 (Convergence of PC). Algorithm 3.3 asymptotically converges to a distribution that locally minimises the \( pq \) KL divergence, given an appropriate subset \( \hat{X}_n \subset X \).

We justify Proposition 3.1 as follows.

Consider an alternative algorithm where, at each iteration \( n \), we randomly choose a subset \( \hat{X}_n^r \subset X^r \) for each robot. This approach is equivalent to Monte Carlo sampling
3.4 Analysis

of the expected utility and thus the biased estimator is consistent (asymptotically converges to $E[f^r]$).

However, for tractable computation and faster convergence, in our algorithm we modify the random selection by choosing a sparse set of strategies $\hat{X}_n$ with the highest expected utility (Section 3.3.4). Although this does not ensure we sample the entire domain $X$ asymptotically, in practice $q_n(\hat{X}_n)$ is a reasonably accurate representation of $q_n(X)$, and therefore this gives us an approximation to importance sampling (Wolpert et al., 2006). Variants of Algorithm 3.3 have been shown to converge to a distribution that locally minimises the $pq$ KL divergence under reasonable assumptions, such as an appropriate cooling schedule for $\beta$ (Wolpert and Bieniawski, 2004).

3.4.3 Analysis of Dec-MCTS

Now we consider the implications of the above results in the context of the overall Dec-MCTS algorithm (Algorithm 3.1). The analyses above show separately that the tree search of Algorithm 3.2 balances exploration and exploitation and that, under reasonable assumptions, Algorithm 3.3 converges to the product distribution that best optimises the joint action sequence.

These results provide strong motivation for the use of these components in the algorithm. However, they do not immediately yield a characterisation of optimality for Algorithm 3.1. To prove convergence rates and global optimality, we would need to characterise the co-dependence between the evolution of the reward distributions $E_{q_n}[f^r | x^r]$ used when sampling the tree and the contraction of the sample space $\hat{X}_n$ used for optimising $q_n$. This co-dependence is complex due to the cyclic nature of the algorithm and communication of information between robots, and thus it is unlikely that any strong claims for global optimality can be made. However, this is generally not achievable in the context of decentralised, long-horizon planning with general objective functions, as addressed in this chapter. Despite this, the following experiments show that the Dec-MCTS algorithm converges rapidly to high-quality solutions in multi-robot active perception scenarios.
3.5 Experiments: Generalised team orienteering

In this section, we evaluate the performance of Dec-MCTS in an abstract multi-robot information gathering problem. An illustration of the problem and an example solution is shown in Figure 3.3. We empirically show convergence, robustness to intermittent communication and a comparison to a centralised variant of MCTS. Further experiments for a different information gathering problem is presented later in Section 3.6.

3.5.1 Problem statement

The problem is motivated by tasks where a team of Dubins robots maximally observes a set of features of interest in an environment, given a travel budget. Each feature can be viewed from multiple viewpoints and each viewpoint may be within observation range of multiple features. This formulation generalises the orienteering problem (Vansteenwegen et al., 2011; Gunawan et al., 2016) by combining the set structure of the generalised travelling salesman problem (Noon and Bean, 1989) with the budget constraints of the orienteering problem with neighbourhoods (Faigl et al., 2017).
Robots navigate within a graph representation of an environment with vertices \( v_i \in \mathcal{V} \), edges \( e_{ij} := \{v_i, v_j\} \in \mathcal{E} \) and edge traversal costs \( c_{ij} \). Each vertex \( v_i \) represents a location and orientation \((x, y, \theta)\) within a square workspace with randomly placed obstacles. The action sequences of each robot are defined as paths of connected edges through the graph beginning at a start vertex unique to each robot. The edge costs are defined as the distance length of the minimum-distance Dubins path between the pair of configurations. All edges that have cost less than a fixed distance are connected and the larger edges are discarded. The connected edges represent feasible actions the robots may select.

For the objective function, we have a collection of sets \( \mathcal{S} = (S_1, S_2, \ldots) \), where each \( S_k \subseteq \mathcal{V} \). These sets may represent a set of features of interest, where a vertex is an element of a set only if the associated feature can be observed from the vertex location. We assume each set is a disc, however the formulation could extend to more complex models (see Chapter 4). The vertices \( v_j \in \mathcal{V} \) are randomly placed (drawn from a uniform distribution) within the sets. A set \( S_k \) is visited if \( \exists v_j \in \mathcal{X}, v_j \in S_k \) and each visited set yields an associated reward \( w_k \). There is no additional reward for revisiting a set. The objective is defined as the sum of the rewards of all visited sets.

### 3.5.2 Calculating expectations

Dec-MCTS requires computing several expectations that, in general, should be approximated using sampling. However, for this problem definition it is possible to exploit the structure of the objective function to efficiently compute exact expectations. We compute expectations as:

\[
\mathbb{E}_{q_n}[g] = \sum_{S_k \in \mathcal{S}} w_k \times P_{q_n}(\exists v_j \in \mathcal{X}, v_j \in S_k) \\
= \sum_{S_k \in \mathcal{S}} w_k \left[ 1 - \prod_{v_j \in S_k} \prod_{\mathcal{X} \in \mathcal{X}_n} \left( 1 - q^*_n(\mathcal{X}^r)1_{\{v_j \in \mathcal{X}^r\}} \right) \right] 
\]
3.5 Experiments: Generalised team orienteering

where \( P_{q_n}(\exists v_j \in x, v_j \in S_k) \) is the probability that at least one \( v_j \in S_k \) is visited by at least one robot. This can be computed much more efficiently than the general equation (linear rather than exponential time in the number of robots) since it only requires iterating over the possible paths for individual robots \((x^r \in \hat{X}_n^r, \forall r)\) rather than iterating over all joint action sequences \((x \in \hat{X}_n)\).

### 3.5.3 Experiment setup

We compare our algorithm (Dec-MCTS) to a centralised MCTS (Cen-MCTS), which consists of a single tree where robot \( r \)'s actions appear at tree depths \((r, r + R, r + 2R, \ldots)\). Intermittent communication is modelled by randomly dropping messages. Messages are broadcast by each robot at 4Hz and a message has a probability of being received by each individual robot.

Experiments were performed with 8 simulated robots running in separate robot operating system (ROS) nodes on a 4-core computer with hyperthreading (8 virtual cores). Each random problem instance (Figure 3.3) consisted of 200 discs with rewards between 1 and 10, 5 obstacles, 4000 graph vertices and random start vertices for each robot. Each iteration of Algorithm 3.1 performs 10 MCTS rollouts and 1 communication broadcast. The set \( \hat{X}_n^r \) consists of 10 paths that are resampled every 10 iterations. The MCTS rollout policy recursively selects the next edge that does not exceed the travel budget and maximises the ratio of the increase of the weighted set cover to the edge cost.

### 3.5.4 Results

**Comparison to centralised MCTS**

The first experiments (Figure 3.4a) show that Dec-MCTS achieved a median 7% reward improvement over Cen-MCTS after 120s, and a higher reward in 91% of the environments. Dec-MCTS typically converged after \(~60s\). A paired single-tailed \( t \)-test supports the hypothesis \((p < 0.01)\) that Dec-MCTS achieves a higher reward than Cen-MCTS for time > 7s. Cen-MCTS performs well initially since it performs
3.5 Experiments: Generalised team orienteering

Figure 3.4 – (a) Comparison of Dec-MCTS with varying computation time to Cen-MCTS (120s). (b) Performance of Dec-MCTS with intermittent communication (60s computation time). (a,b) Vertical axes show percentage additional reward achieved by Dec-MCTS compared to Cen-MCTS. Error bars show 0, 25, 50, 75 and 100 percentiles (excluding outliers) of 100 random problem instances.

a centralised greedy rollout that finds reasonable solutions quickly. Dec-MCTS soon reaches deeper levels of the search trees, though, which allows it to outperform Cen-MCTS. Dec-MCTS uses a collection of search trees with smaller branching factors than Cen-MCTS, but still successfully optimises over the joint-action space.

We note that in this implementation Dec-MCTS is performing parallel computation while Cen-MCTS is mostly sequential. While it is difficult to measure the difference in computation resources used (due to the use of virtual cores, less than 100% processor utilisation, and overheads of using ROS message passing), the results indicate that Dec-MCTS would outperform Cen-MCTS after adjusting for this difference in computation resources.

**Robustness to communication loss**

The second experiments analysed the effect of communication degradation. When the robots did not communicate, the algorithm achieved a median 31% worse than Cen-MCTS, but with full communication achieves 7% better than centralised, which shows the robots can successfully cooperate by using our proposed communication algorithm. Figure 3.4b shows the results for partial communication degradation. When half of the packets are lost, there is no significant degradation of performance.
Figure 3.5 – Experiment setup for the point cloud dataset. (a) Environment with labelled locations, (b) picnic table (PT), (c) barbecue (BQ), (d) wheelie bin (WB), (e) motorbike (MB), (f) street light (ST), (g) tree (TR), (h) palm tree (PT).

When 97% of packets are lost the performance is degraded but the algorithm still performs significantly better than with no communication.

These experiments demonstrate that Dec-MCTS achieves reasonable performance even when the communication becomes less reliable. While these results show a robustness to communication loss, they also indicate that some of the communication messages are not entirely necessary. Later in Section 3.7 we exploit this property to develop a communication-scheduling algorithm that selects when to communicate and who to communicate during each iteration of Dec-MCTS. This extended formulation is particularly useful in scenarios where communication resources are limited, and the scheduling enables using this communication resource only when it is predicted to be beneficial for the planning performance.

3.6 Experiments: Active object recognition

This section describes experiments for online active object recognition, using point cloud data collected from an outdoor mobile robot in an urban scene, illustrated in Figure 3.5. We first outline the problem and experiment setup, and then present results that analyse the value of online replanning and compare Dec-MCTS to a greedy planner.
3.6 Experiments: Active object recognition

3.6.1 Problem statement

A team of robots aim to determine the identity of a set of static objects in an unknown environment. Each robot asynchronously executes the following cycle: (1) plan a path that is expected to improve the perception quality, (2) execute the first planned action, (3) make a point cloud observation using onboard sensors, and then (4) update the belief of the object identities and their poses. Each robot asynchronously performs this cycle until their travel budget is exhausted. The robots have the same motion model as in Section 3.5. Each graph edge has an additional constant cost that represents the time required to process an observation and perform replanning. Thus, each robot’s budget is a constraint on the sum of its travel distance and processing time.

We use a perception model for object recognition similar to that proposed in Chapter 7.2 of Patten (2017). The robots maintain a belief of the identity of each observed object, represented as the probability that each object is an instance of a particular class of objects. The set of object classes are defined in a given database. The aim is to improve this belief, which is achieved by maximising the mutual information objective proposed by Patten et al. (2015). The posterior probability distribution for each object after a set of observations is computed recursively using Bayes’ rule. The observation likelihood is calculated by measuring the similarity between the shape of the point cloud with each model instance in the database. Similarity is computed by first aligning the point clouds of a pair of objects using the Iterative Closest Point (ICP) algorithm (Besl and McKay, 1992) and then calculating the symmetric residual error (Douillard et al., 2012). Objects may merge or split after each observation if the segmentation changes. Observations are fused using decentralised data fusion or a central processor and shared between all robots, and thus all robots are assumed to have the same belief of the environment. While planning, the value of future observations are estimated by simulating observations of objects in the database for all possible object identities, weighted by the belief probabilities, and using maximum likelihood estimates for poses.
3.6.2 Experiment setup

The experiments use a point cloud dataset (Patten et al., 2015) of Velodyne scans of outdoor objects in a $30 \times 30 \text{m}^2$ park shown in Figure 3.5(a). The environment consisted of 13 objects from 7 different model types as shown in Figures 3.5(b)–(h). The dataset consists of single scans from 50 locations and each scan was split into 8 overlapping observations with different orientations. Each observation had a 180° field of view and 8 m range. These locations and orientations form the roadmap vertices with associated observations. Each object was analysed from separate data to generate the model database. The robots are given a long-range observation from the start location to create an initial belief of most object locations. The team consists of 3 robots, who share a fixed start location with different orientations.

The experiments simulate an online mission where each robot asynchronously alternates between planning and perceiving. Three planners were trialled: our Dec-MCTS algorithm with 120 s replanning after each action, Dec-MCTS without replanning, and a decentralised greedy planner that selects the next action that maximises the mutual information divided by the edge cost. The recognition score of an executed path was calculated as the belief probability that each object matched the ground-truth object type, averaged over all objects. The planners cannot directly optimise the paths with respect to the recognition score since the ground-truth is not known in advance; however, planning with respect to the mutual information objective function is intended to indirectly optimise the recognition score.

3.6.3 Results

Overall, the results validate the coordination performance of Dec-MCTS. Figure 3.6a shows the recognition score (task performance) over the duration of the mission for 10 trials with 3 robots. The maximum possible recognition score subject to the perception algorithm and dataset was 0.62, which was achieved by visiting every location in the dataset. Dec-MCTS outperformed greedy halfway through the missions since some early greedy decisions and poor coordination reduced the possibility of making subsequent valuable observations. By the end of the missions some greedy plans
3.6 Experiments: Active object recognition

![Graph](image)

**Figure 3.6** – (a) Task performance over mission duration for 10 trials (maximum possible score is 0.62). (b) Overlay of 2 example missions with 3 robots. Blue paths denote online Dec-MCTS (score 0.53). Orange paths denote greedy policy (score 0.42). Objects are green point clouds where shading indicates height. Robots observe at black dots in direction of travel. Start location top right.

Successfully made valuable observations, but less often than Dec-MCTS. The no-replanning scenario achieved a similar score as the online planner in the first half, showing that the initial plans are robust to changes in the belief. For the second half, replanning improved the recognition score since the belief had changed considerably since the start. This shows that while the generated plans are reasonable for many steps into the future, there is also value in replanning as new information becomes available.

Figure 3.6b shows two example missions using online Dec-MCTS (blue) and greedy (orange) planners, and their score over the mission duration. Greedy stayed closer to the start location to improve the recognition of nearby objects, and consequently observed objects on the left less often; reaching this part of the environment would require making high cost/low immediate value actions. On the other hand, Dec-MCTS achieved a higher score since the longer planning horizon enabled finding the high value observations on the left, and was better able to coordinate to jointly observe most of the environment.
3.7 Extension: Communication scheduling

The Dec-MCTS algorithm presented so far naively communicates at every iteration. However, in practice this may not be possible, for example due to exceeding bandwidth limits. In this section, we propose a communication scheduling algorithm that can be used to select which communication messages should be sent by reasoning over the information value of the messages.

More specifically, in this section we are interested in addressing the problem of deciding when to communicate, and to whom, while the robots are performing decentralised planning. We aim to find a balance between minimising the use of limited communication resources, and satisfying the planning algorithm objectives. This problem is challenging because it is difficult to efficiently predict how communicating the current plan will impact the coordination performance in the long term (Becker et al., 2009).

We propose a novel planning algorithm that reasons over the value of communication messages to decide when and to whom each robot should communicate. We present this algorithm for the context of Dec-MCTS, although the methods presented here could be adapted for other online coordination algorithms. The aim is to minimise communication while maintaining bounds on the uncertainty of the reward distribution. Our approach predicts the value of future communication messages, then uses these predictions to plan a sequence of communication requests. The predictions are performed using a particle filter, and the optimal communication schedule is found using dynamic programming. Notably, the approach collapses the decision-tree into a directed acyclic graph, enabling polynomial runtime.

Overall, the approach trades drastically reduced communication for a modest overhead in computation time. We have evaluated our approach in a multi-robot information gathering scenario similar to that in the experiments of Section 3.5. Our results show large reductions in channel utilisation with little impact on task performance. This demonstrates our approach is suitable for communication planning in real-world multi-robot scenarios.

Our approach here is analogous to the belief-space planner of Ondruska et al. (2015), which was developed for scheduling localisation hardware usage to conserve energy
3.7 Extension: Communication scheduling during path-following scenarios. The general approach also has similarities to the dynamic programming algorithm proposed later in Chapter 5.

3.7.1 Summary of approach

We present a brief summary of our communication scheduling formulation and algorithm as follows. Full details may be found in Best et al. (2018c).

Formulation and algorithm

We propose a decentralised algorithm for deciding when to communicate, and to whom, while the robots are performing Dec-MCTS. During each iteration \( n \), robot \( i \) updates its plan, schedules sequences of communication requests, then performs the selected requests. This continues until the robots execute their plans.

In the decentralised planning phase, robot \( i \) updates its plan using Dec-MCTS while considering the most recently communicated plans of the other robots. As described in the preceding sections, this update involves selecting a subset of possible action sequences \( \hat{\mathcal{X}}_i^n \subset \mathcal{X}_i \) using Monte Carlo tree search, and then optimising a probability distribution \( q^n_t \) over the selected subset \( \hat{\mathcal{X}}_i^n \).

Then, robot \( i \) decides whether or not to request communication from each robot \( j \) by considering how much its own plan \( q^n_i \) depends on the plan of robot \( j \). This dependency is measured as the uncertainty \( \sigma^n_i \) of the expected local utility \( f^i \) for robot \( i \) that is caused by not knowing the plan \( q^n_j \) of robot \( j \). This uncertainty is described as

\[
\sigma^n_i = \text{STDEV}_{\mathcal{B}^i_n(q^n_i)} \left( \mathbb{E}_{q^n_j \cup q^n_i(j)} \left[ f^i \left( \mathbf{x}^i \cup \mathbf{x}^{(j)} \right) \right] \right),
\]

(3.14)

where this standard deviation is measured with respect to the uncertain belief \( \mathcal{B}^i_n \) of the distribution \( q^n_i(\mathbf{x}^i) \) of robot \( j \). The decision is made by first evolving the belief \( \mathcal{B}^i_n \) over a finite time-horizon, then finding the optimal communication schedule using dynamic programming, summarised as follows.

The belief evolution is implemented using a particle filter that predicts the distribution \( q^n_i \) at future timesteps. If robot \( i \) chooses to request communication at a future
timestep, then the set of particles at that timestep has uncertainty $\sigma_n^l = 0$. Typically, $\sigma_n^l$ then increases at each iteration until the next communication request. When $\sigma_n^l$ increases beyond a threshold $\theta$ the robots must communicate. This belief evolution manifests as a prediction graph that describes valid request sequences.

Dynamic programming is then used to find the optimal sequence of communication decisions for $T$ future iterations. This sequence minimises the number of requests while satisfying the uncertainty $\sigma_n^l \leq \theta$ constraints.

At the end of each iteration, robot $i$ requests the plans $\hat{X}_n^l, q_n^l$ of selected robots according to the first decision in the communication schedule. The received information is used to improve coordination in future planning iterations.

**Analysis**

The schedule is optimal with respect to the belief and is guaranteed to satisfy the constraints. The algorithm has polynomial runtime, with complexity $O(BT^2RE)$, where $B$ is the number of particles at each decision node, $T$ is the number of steps in the planning horizon, $R$ is the number of robots, and $E$ is time taken to compute expected utility. We note that due to our construction, the total number of particles generated during each round of communication scheduling is $O(BT^2R)$, which is polynomial in the time horizon (rather than exponential, as would occur in a typical decision tree). Typically, we expect that $E$ is large for non-trivial problems; therefore, $B$ or $T$ should be selected to strike a balance between runtime and desired accuracy of the predictions.

**3.7.2 Experiments**

We analyse the performance and behaviour of our proposed communication scheduling algorithm in the context of Dec-MCTS and the information gathering scenario from Section 3.5. Overall, the results show that the performance of Dec-MCTS can be maintained, even with significantly reduced communication rates, by judiciously selecting which communication messages to transmit.
3.7 Extension: Communication scheduling

Although it is difficult to identify alternative algorithms that can be directly compared to ours, we do provide comparisons to a suite of communication reduction approaches. One of these is full (effectively all-to-all) communication. This scheme is not feasible in practice for our systems of interest, which are field robots with significant channel contention that arises from sources such as RTK GPS corrections, e-stop heartbeat messages, and telemetry. However, the full communication case provides a quality benchmark that allows us to measure relative coordination (task) performance.

Comparison scenarios

In the following experimental results we compare 5 different scenarios. The All-to-all scenario makes the unrealistic assumption of perfect communication and the robots communicate their intentions at every iteration. Random represents a scenario where only 20% of the packets are successfully received due to uniform-random message loss (e.g., to model excessive contention on the communication channel). We compare two version of our approach: Horizon 4 plans with a planning horizon of $T = 4$, while Greedy only looks one time-step ahead. In the Horizon 4 and Greedy scenarios, $\theta$ is selected such that they have a 20% average communication rate. As a baseline comparison, the None scenario assumes all communication fails and no messages are successfully received.

Results

For a baseline comparison we observe that communication is important for coordination. In Figure 3.7a, the robots take advantage of the perfect setting of having full communication (All-to-all) to coordinate their plans effectively. In Figure 3.7b, there is no communication (None) and therefore there is no coordination, resulting in multiple visits to the same regions. Table 3.1 compares the planning performance for different communication scenarios. All-to-all naturally resulted in the highest reward, but the partial communication scenarios performed well despite having 80% less communication. As expected, None resulted in the poorest performance. Planning with a horizon of $T = 4$ achieved higher rewards than Greedy, showing the advantage of
3.8 Summary

We have presented a new algorithm for decentralised coordination that is suitable for a general class of problems. Our results demonstrate that the performance (i.e., solution quality) of our approach is as good as or better than its centralised counterpart in planning over a time horizon. Both of these scenarios outperformed Random, which highlights the practical benefit of performing informative communication planning.

The planned communication scenarios achieved better results than Random since the proposed approach chose to communicate more frequently for pairs of robots that have a larger coupling between their local utilities. For the $T = 4$ scenarios, the highest communication rate (62%) is between the blue and pink robots in the bottom left of Figure 3.7a. We expect this pair to communicate more since their reachable regions significantly overlap. The yellow robot in the bottom right received the least requests (11%) since it is relatively isolated. The algorithm also selects when to communicate, which tended to be more during earlier iterations when successful coordination is most important.
Table 3.1 – Reward collected (as a percentage) in the information gathering problem with the different communication scenarios. Iterations are the sum of planning iterations performed by the 8 robots. Rewards averaged over 50 trials in the environment illustrated in Figure 3.7. Average standard deviation of reward is 7%.

<table>
<thead>
<tr>
<th>Iterations</th>
<th>All-to-all</th>
<th>Random</th>
<th>Horizon 4</th>
<th>Greedy</th>
<th>None</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>23.2</td>
<td>17.4</td>
<td>23.5</td>
<td>23.1</td>
<td>15.9</td>
</tr>
<tr>
<td>100</td>
<td>65.0</td>
<td>54.0</td>
<td>62.3</td>
<td>60.4</td>
<td>32.6</td>
</tr>
<tr>
<td>150</td>
<td>71.2</td>
<td>64.5</td>
<td>66.3</td>
<td>67.3</td>
<td>32.6</td>
</tr>
<tr>
<td>200</td>
<td>76.1</td>
<td>68.7</td>
<td>69.5</td>
<td>69.6</td>
<td>33.9</td>
</tr>
<tr>
<td>250</td>
<td>74.1</td>
<td>66.8</td>
<td>71.3</td>
<td>69.9</td>
<td>31.8</td>
</tr>
<tr>
<td>300</td>
<td>73.5</td>
<td>66.3</td>
<td>71.6</td>
<td>69.6</td>
<td>32.8</td>
</tr>
<tr>
<td>350</td>
<td>74.5</td>
<td>65.5</td>
<td>73.4</td>
<td>71.7</td>
<td>35.1</td>
</tr>
<tr>
<td>400</td>
<td>76.7</td>
<td>68.9</td>
<td>76.3</td>
<td>74.5</td>
<td>33.0</td>
</tr>
</tbody>
</table>

real-world applications, and that it effectively optimises over sequences in the joint-action space even with intermittent communication. A key conceptual feature of our approach is its generality in representing joint action sequences probabilistically rather than deterministically. Dec-MCTS has the ability to efficiently plan over long planning horizons, computes anytime solutions, allows incorporating prior knowledge, and provides convergence rate guarantees.
Chapter 4

Self-organising maps for generalised orienteering

In this chapter we propose a new self-organising map (SOM) planning algorithm suitable for active perception formulations. We formulate a new generalisation of the well-known orienteering problem that has polygonal goal regions and multiple robots. We demonstrate how this formulation is useful as an active perception formulation by exploiting an inverse sensor model. We propose an efficient solution algorithm for this problem based on the SOM adaptive learning procedure.

4.1 Overview

The performance of an active perception mission, such as a classification, exploration, coverage, or persistent monitoring task, is largely dependent on an appropriate choice of viewpoints. Current approaches for active perception typically estimate the value of visiting candidate viewpoints by simulating predicted observations (van Hoof et al., 2014; Wu et al., 2015; Patten et al., 2016). For complex sensor models, these predictions can be computationally expensive, which therefore restricts the capabilities of planning algorithms.

Instead, in this chapter, we focus on planning paths for perception tasks where informative parts of the objects in the environment have been extracted. Therefore,
Figure 4.1 – An example environment, set of object parts, viewpoint regions and solution paths for two robots (same as Figure 1.4). The 3D point cloud was generated by a real robot moving around an environment consisting of trees, tables, chairs, bins and a motorbike. The underlying grid has 5 m spacing. Almost all object parts are observed along the planned paths, with some skipped due to the travel budget constraints. In this scenario, all objects are known to the offline planner. In online scenarios, additional goals are placed in unexplored regions, and the goals and plans adapt as observations are made.

we use an inverse sensor model to define a discrete set of overlapping continuous viewpoint regions, with associated rewards, where each part can be observed. Correlations between viewpoints can be naturally modelled in our formulation as the overlap between viewpoint regions. Figures 1.4 (earlier) and 4.1 illustrate an example outdoor environment with a collection of objects observed by a 3D laser scanner. The path planning problem is to optimise the rewards gained by visiting these desirable viewpoint regions. This new formulation for active perception enables the planner to consider a continuous space of candidate viewpoints, long-horizon planning, multi-robot scenarios and efficient online replanning.

This active perception formulation describes a multi-goal path planning problem with similarities to the orienteering problem (OP) (Gunawan et al., 2016; Vansteenwegen et al., 2011) and the travelling salesman problem (TSP) (Toth and Vigo, 2001). The prize-collecting TSP with neighbourhoods (PC-TSPN) is a closely related TSP variant that has recently been solved and applied to data collection in sensor network applications (Faigl and Hollinger, 2014). In the PC-TSPN, the objective is to plan the path of a robot that maximally selects and visits a set of disks, where the ob-
Objective function is defined as the sum of the path length and the rewards for visiting each disk. This objective function has convenient algorithmic properties; however, it is unclear how to balance the trade-off between path lengths and rewards when applied to real problems. Instead, we develop a new formulation that generalises the OP; we directly optimise the observation rewards while the path length is limited by a maximum travel budget. These budget constraints can be selected to meet the requirements of the application, such as fuel, time and other resource constraints, or a planning horizon.

The considered problem is NP-hard, which can be shown by a reduction from the orienteering problem, and therefore we turn to heuristic solutions. In particular, we consider an extension of the self-organising map (SOM) approaches for the TSP. SOM is a two-layered neural network accompanied by an unsupervised learning procedure that preserves topological properties of an input space. SOM has been applied to the traditional TSP by several authors, e.g., Angéniol et al. (1988); Somhom et al. (1997). Although SOM for the TSP does not compete with the best known combinatorial heuristics for the conventional TSP (Helsgaun, 2000), it provides a significant advantage in problems that require selecting observation locations. This is particularly important in the TSPN (Faigl and Hollinger, 2014) and the OP with neighbourhoods (Faigl et al., 2016) where the algorithm implicitly selects sensing locations within continuous neighbourhoods.

Jointly optimising the selection and sequence of nodes to observe, along with finding favourable viewpoints within sensing regions, can greatly reduce the path distance by avoiding unnecessary travel. Therefore, we consider the original idea of the SOM-based data collection planning introduced in Faigl and Hollinger (2014) for our constrained problem with limited travel budgets. Our new approach ensures these hard budget constraints are satisfied by the planning algorithm.

Moreover, we also generalise the approach in Faigl and Hollinger (2014) to planning for multi-robot teams. This requires addressing additional challenges, including coordinating the robots to select mutually beneficial observation locations, effectively using the available resources of each robot, and overcoming the compounded computational complexity to quickly find good solutions. Our algorithm jointly plans for multiple robots simultaneously by optimising the allocation of nodes to robots.
Therefore, our approach does not require predefined or explicit partitioning of the environment. The algorithm has polynomial bounds on runtime complexity, and scales well as the number of robots increases.

A primary contribution of this chapter is to demonstrate that the algorithmic approach is suitable for online scenarios. In particular, we show that the formulation can naturally incorporate exploration objectives to discover new information, and the planner efficiently performs replanning as new information becomes available. Exploration objectives can naturally be encoded as viewpoint regions, so that the planner balances between making high-quality observations of known objects and visiting unexplored space to discover new objects. This formulation is motivated by scenarios where there are two complementary sensing modalities. For example, a long-range laser sensor (Bargoti et al., 2015) detects the presence and locations of trees on a farm, while a close-range high-resolution RGB-D sensor (Martens et al., 2017; Peng et al., 2016) performs the primary task of characterising the fruit in the trees. The planner needs to balance the use of these two modalities in order for the robots to discover as many objects as possible while also making sufficient close-range observations. The proposed SOM algorithm enables efficient online replanning as new information becomes available since it is able to effectively reuse and adapt previous solutions. This is a vitally important requirement for real robots performing onboard computations while executing a mission (Likhachev et al., 2005).

In addition to theoretical analysis, we also perform simulations of several random environments and active perception tasks using a 3D point cloud dataset (Patten et al., 2015) and a realistic observation model using ensemble of shape functions descriptors (Wohlkinger and Vincze, 2011). The results highlight advantages of the algorithm in an offline setting for addressing the multi-robot, non-uniform reward, constrained budget and polygonal region characteristics of the problem. We also show the advantages of planning over continuous rather than discrete space by showing our approach outperforms the Dec-MCTS algorithm presented in Chapter 3. Additionally we empirically evaluate the performance of the planner when incorporating exploration objectives and adapting to new information when replanning. We highlight the advantages of long-horizon planning over greedy approaches, even when limited information is available. The active perception experiments show the feasibility in
practice of online long-horizon planning for multi-robot active perception tasks.

4.1.1 Chapter outline

The remainder of this chapter is organised as follows. Section 4.2 introduces the active perception problem formulation considered in this chapter. In Section 4.3, we propose a self-organising map solution algorithm for this problem. The algorithm is analysed theoretically and empirically in Section 4.4. Sections 4.5 and 4.6 describe how this formulation can be applied to object recognition type problems in offline and online scenarios, and show results for simulated experiments with a 3D point-cloud dataset, including a comparison to Dec-MCTS. Finally, Section 4.7 concludes the chapter.

4.2 Problem formulation

This section formally defines the active perception formulation considered in this chapter, as a subproblem of Problem 1.1. The objective is to plan the paths for a team of robots such that they maximally observe a set of nodes in the environment with varying rewards. Each robot has an associated travel speed and maximum travel budget. Each node may be observed by visiting any point in its associated viewpoint region, represented as a polygon. These nodes, viewpoint regions and rewards may be defined to meet the objectives of the application; in Sections 4.5 and 4.6 we formulate example problem instances for perceiving 3D point-cloud objects and incorporating exploration objectives. In this section we define the problem by considering the objectives that are currently known at a given time instance. Though we are interested in solving this problem in an online setting such that the plans adapt as new information is discovered, and a formulation for these scenarios is developed further in Section 4.6.
4.2 Problem formulation

4.2.1 Multi-robot team

A team of $R$ robots is denoted $\mathcal{R} = \{r^1, r^2, ..., r^R\}$. The trajectory of each robot $r^i$ is defined as a sequence of waypoints $X^i = (x^i_1, x^i_2, x^i_3, ...)$, where each waypoint is a position within a free space environment $x^i_j \in \mathbb{R}^2$. Each robot $r^i$ moves along a straight line between waypoints at a constant speed $s^i$, which may be different for each robot. The cost of each robot’s path $c^i \geq 0$ is the time taken to travel through the sequence of waypoints $X^i$. Each robot has a cost budget $b^i > 0$, and a set of robot paths $\{X^i\}$ is deemed to be feasible if every robot meets its budget constraint, i.e., $c^i \leq b^i, \forall r^i \in \mathcal{R}$. We address several possible conditions for the start and end positions of the robots: (1) the start and end positions are unconstrained and free to be selected by the planner, (2) the start positions are unconstrained but the robots must end at their start position, and (3) the start and/or end position is fixed.

4.2.2 Viewpoint regions and rewards

The robots aim to observe a set of $N$ nodes $\mathcal{N} = \{n^1, n^2, ..., n^N\}$ at different locations within the environment. Every node has a weight $w^k > 0$ that defines the reward for observing the node. Each node $n^k$ has a continuous set of viewpoints $Z^k$ defined as all points on and within a simple polygon. The robot observes a node if any waypoint of the robot’s path is within the viewpoint region, i.e., $\exists x^i_j \in X^i : x^i_j \in Z^k$. The binary indicator variable $o^k \in \{0, 1\}$ for each node $n^k$ is 1 if the node is observed by one or more robots and 0 otherwise. All robots sense continuously along their paths, which can be taken into account in the above definition by adding additional waypoints along a path at no extra cost. Although we assume the regions are the same for each robot, the algorithm can easily be extended to robot-dependent observations.

The presented solution is applicable for any set of viewpoint regions and rewards. In active perception formulations, these viewpoint regions can be used to represent locations where key points of interest may be observed. The associated reward may encode the importance of observing the point of interest. Later in Section 4.5.1 we present an example definition motivated by object recognition scenarios, but we emphasise that the presented solution algorithm is not specific to this example definition.
We are also particularly interested in formulations for online tasks, where the robots should aim to observe known objectives as well as discover currently unknown objectives. This balance can be achieved using our formulation by introducing new nodes to the set $\mathcal{N}$ that represent regions where the robots may be expected to discover new objectives. We formulate this concept further in Section 4.6.

4.2.3 Problem statement

The optimisation problem is to plan the locations of waypoints for each robot and the sequence the waypoints are visited $X_i$, such that all budget constraints are met and the sum of the observation rewards for the nodes is maximised. More formally, we state the problem addressed in this chapter as follows.

\textbf{Problem 4.1} (Generalised orienteering problem). Find the set of paths $\{X_i\}$ that maximises $\sum_{n^k \in \mathcal{N}} d^k w^k$ and satisfies the constraints $c^i \leq b^i, \forall r^i \in \mathcal{R}$.

We are interested in solving Problem 4.1 by replanning in an online setting. The robots initially plan based on the information available offline. After each action is executed, an observation may result in a change of the nodes, viewpoint regions or rewards. We assume these changes are small and therefore the planner should efficiently adapt its previous solution to address the new objectives.

4.2.4 NP-hardness

The problem is NP-hard and a reduction from the orienteering problem (Vansteenweghen et al., 2011) with Euclidean costs can readily be shown by setting the number of robots $R$ to 1 and the viewpoint sets $\{Z^k\}$ as singleton. This result motivates the development of a heuristic algorithm to approximately solve the problem in polynomial time.
4.3 Self-organising map algorithm

Self-organising map algorithms aim to give a lower-dimensional representation of an input space, while preserving a given topological graph-based structure of the representation. A detailed description of SOMs is provided earlier in Section 2.3.5, with illustrated examples in Figure 2.3.

For our problem the input space is the set of viewpoint regions in the environment, and the algorithm aims to find a set of sequences of waypoints (representing robot paths) that best fits this input space. The learning procedure is competitive in that each viewpoint region is presented one at a time, and each waypoint competes to be the winner for representing that region. A winner waypoint moves towards that region, and neighbours of the winner in the graph topology will also move towards the region by a decreasing distance. This process is repeated for a fixed number of learning epochs, when convergence of the paths to a stable state is guaranteed.

This section details the proposed SOM learning procedure for our problem formulation, which includes addressing non-uniform observation rewards, node selection satisfying budget constraints, multi-robot task allocation to nodes and can efficiently perform online replanning by adapting previous solutions. We first provide an overview of the algorithm followed by a detailed explanation of all components.

4.3.1 Algorithm overview

An overview of the proposed self-organising map algorithm is presented in Figure 4.2 and Algorithm 4.1. The algorithm consists of two nested loops and in each iteration, the solution paths for the team of robots is adapted towards the final solution.

During each iteration of the outer loop (Algorithm 4.1 line 5), called an epoch, all viewpoint regions are addressed one at a time (line 7) by adapting the path of one of the robots towards the considered viewpoint region (line 12). The path adaptations are performed using an extension of the standard self-organising map adaptation process for TSP problems, in combination with a greedy robot-node allocation policy (line 11). This allocation policy divides the workload between robots while satisfying budget constraints. Regions with low reward are considered once per epoch while
high-reward regions are considered multiple times. At the end of each epoch, the paths are regenerated to remove non-informative waypoints (line 13), and an adaptation parameter is cooled (line 16).

The algorithm continues for a fixed number of epochs until convergence is guaranteed. During early epochs the paths typically make large sporadic jumps around the environment, while small local refinements are made in later epochs. In our analysis (later in Section 4.4) we show that the runtime complexity of the algorithm is $O(N'^2)$, where $N'$ is the number of viewpoint regions after duplication to take into account the rewards \( \{w^k\} \). In the remainder of this section, we provide a detailed explanation of all components of the algorithm.

### 4.3.2 Graph topology

The graph topology for the SOM is a set of $R$ sequences of waypoints that directly represent the robot paths. Each of these paths will transform over time according to the following learning procedure. For problems where each robot must return to its start position, the topology of each path is a closed loop. If this is not required,
4.3 Self-organising map algorithm

Algorithm 4.1 Self-organising map algorithm.

Input: robot speeds \{s^i\} and budgets \{b^i\},
      a set of nodes \{n^k\} with associated
      viewpoint regions \{Z^k\} and rewards \{w^k\},
      adaptation parameters \sigma_0 and \delta

Output: planned path for each robot \{X^i\}^* 

1: \(X^i \leftarrow \text{circle around arbitrary node } n^i, \forall r^i \in \mathcal{R}\)
2: \(N' \leftarrow \text{duplicate } n^k \in N \text{ by factor } \frac{w^k}{\text{gcd}\{w^k\}}\)
3: \(N' \leftarrow N' \cup \{\text{virtual node for each fixed waypoint}\}\)
4: \(\sigma \leftarrow \sigma_0; i \leftarrow 1\)  \hfill \triangleright \text{Adaptation parameter}
5: \textbf{while} not converged \textbf{do}
6: \(\text{perm} \leftarrow \text{random permutation of } \{n^k\}\)
7: \textbf{for each } \(n^k \in N', \text{ in order } \text{perm} \textbf{do}\)
8: \(X^u \leftarrow \text{ADAPTATION}(X^i, Z^k, \sigma)\) \hfill \triangleright \text{See Algorithm 4.2}
9: \(c^u \leftarrow \text{travel time of path } X^u \text{ at speed } s^i\)
10: \(r^i \leftarrow \arg\min_{r^i \in \mathcal{R}, c^u \leq b^i} \left(\frac{c^u}{b^i}\right)\) \hfill \triangleright \text{Robot selection}
11: \(X^i \leftarrow X^u\) \hfill \triangleright \text{Update selected robot}
12: \{X^i\} \leftarrow \text{regeneration of } \{X^i\}\)
13: \(F \leftarrow \sum_{n^k \in N} \sigma^k w^k\) \hfill \triangleright \text{Evaluate objective}
14: \textbf{if } F > F^* \textbf{ then } \{X^i\}^* \leftarrow \{X^i\}\)
15: \(\sigma \leftarrow (1 - i\delta)\sigma; i \leftarrow i + 1\) \hfill \triangleright \text{Save best plan}

then the topology is a set of open paths. When performing online replanning, the
paths may be initialised as the previously computed solution. If no previous solution is
available, then we initialise each path as a small circle (consisting of \(\lfloor N/R \rfloor\) waypoints)
around the centre of a unique arbitrary node (Algorithm 4.1 line 1). This initialisation
is reasonable since the paths will quickly spread over the input space and adjust their
number of nodes during the first learning epochs.

4.3.3 Viewpoint rewards

Each node has an associated reward for being visited. To ensure that the learning
procedure favours visiting the higher reward viewpoint regions, each node is duplic-
ated according to its reward. The node \(n^k\) is duplicated by a factor of \(w^k\) divided by
the greatest common divisor of the set of rewards \(\text{gcd}(\{w^k\})\). This is performed in
Algorithm 4.1 line 2. The computation time complexity is dependent on the number
of duplications. Therefore it may be beneficial to reduce the number of duplications by rounding the rewards to the nearest multiple of a number greater than $\text{gcd}(\{w^k\})$.

The motivation for this approach is that high-reward regions will be trialled more often in each learning epoch. This increases the likelihood of a robot path transforming towards the higher weighted nodes, decreases the likelihood of the node not being selected due to budget constraints, and decreases the likelihood of waypoints in high-reward regions being removed during the regeneration step.

### 4.3.4 Learning epochs

In each learning epoch (iteration of Algorithm 4.1 line 5 loop), each node is considered one at a time, and one robot is selected to transform its path towards each viewpoint region, if it meets its budget constraint. At the end of each learning epoch, any unnecessary waypoints are removed before starting the next learning epoch. We describe these steps in more detail as follows.

**Permute the nodes**

At the start of each epoch, the nodes are permuted in a random order which will determine the order that they are considered (Algorithm 4.1 lines 6–7). This ensures the algorithm is less sensitive to the ordering and the initial conditions, and more likely to escape from local optima.

**Winner selection and adaptation**

The key steps in the SOM algorithm are the winner waypoint selection and the adaptation of the position sequences, as detailed in Algorithm 4.2. For every node, this is performed for each robot, but then in the following step a robot allocation policy ensures only one of the robots gets updated for each node. The winner waypoint selection is performed by considering all waypoints and edges in the current robot path $X^i$. The existing waypoint or a point along one of the existing edges that is closest to any point within the viewpoint region $Z^k$ is considered as the winner (Algorithm 4.2
Algorithm 4.2 Adaptation step of the SOM algorithm.

1: function \textsc{adaptation}
2: \hspace{1em} \textbf{Input:} path $X^i$ of robot $r^i$, viewpoint region $Z^k$ of node $n^k$, adaptation parameter $\sigma$
3: \hspace{1em} \textbf{Output:} an adapted path $X^i'$ for robot $r^i$
4: 1: $x_w \leftarrow$ closest waypoint in $X^i$ to $Z^k$
5: 2: $z_w \leftarrow$ closest point in $Z^k$ to $x_w$
6: 3: $d_w \leftarrow \|z_w - x_w\|$
7: 4: $x_e \leftarrow$ closest point on edges of $X^i$ to $Z^k$
8: 5: $z_e \leftarrow$ closest point in $Z^k$ to $x_e$
9: 6: $d_e \leftarrow \|z_e - x_e\|$
10: \hspace{1em} \Comment{Winner selection}
11: if $x_w \in Z^k \lor d_w \leq d_e$ then
12: \hspace{1em} \Comment{Select waypoint as winner}
13: \hspace{1em} if $x_w$ is fixed then
14: \hspace{1em} \hspace{1em} $x_w \leftarrow$ copy of $x_w$
15: \hspace{2em} $X^i \leftarrow$ insert $x_w$ into $X^i$ next to fixed copy
16: \hspace{3em} $x^* \leftarrow x_w; z^* \leftarrow z_w$
17: \hspace{1em} else
18: \hspace{1em} \Comment{Select edge as winner}
19: \hspace{2em} $X^i \leftarrow$ insert $x_e$ into $X^i$ along edge
20: \hspace{3em} $x^* \leftarrow x_e; z^* \leftarrow z_e$
21: \hspace{1em} for each $x_j \in X^i$ do
22: \hspace{2em} \Comment{Adapt waypoints in neighbourhood of $x^*$}
23: \hspace{3em} $l \leftarrow$ cardinal distance from $x^*$ to $x_j$
24: \hspace{4em} $x_j \leftarrow$ move $x_j$ towards $z^*$ by factor $f(\sigma, l)$ in Eqn. (4.1)

line 11 or 14). If the winner is a point along an edge, then a new waypoint is inserted into the path at this point (line 13).

The winner waypoint $x^*$ is then moved to the closest point $z^*$ in $Z^k$. If $z^*$ is on the edge of the polygon it is moved slightly towards the centre to avoid numerical issues and to reduce the chances of the waypoint being immediately removed in the next path regeneration phase. The cardinal distance (number of hops) in the path/loop from $x^*$ to every other existing waypoint is denoted $l$. Each waypoint in $X^i$ is moved some fraction towards $z^*$ (line 19), such that waypoints with low cardinal distance to $x^*$ will move further towards $z^*$ than other waypoints. This fraction is determined by
the neighbourhood function

\[
f(\sigma, l) = \begin{cases} 
\mu e^{-\frac{l^2}{\sigma^2}} & \text{for } i < \frac{1}{\delta} \\
0 & \text{otherwise}
\end{cases},
\]

where \(i\) is the current learning epoch and the gain decreasing rate \(\delta\) is a small constant parameter, e.g., \(\delta = 0.002\). The value of \(\sigma\) is decreased at the end of each learning epoch as \(\sigma \leftarrow (1-i\delta)\sigma\) (Algorithm 4.1 line 16), which causes the neighbours to adapt less as the algorithm progresses. We define the learning rate as a constant \(\mu = 1\); alternatively, a cooling schedule could be defined for \(\mu\) (Zhang et al., 2006).

Forcing \(f(\sigma, l)\) to 0 after \(i = 1/\delta\) is a natural restriction of the neighbourhood function since \(\sigma = 0\) when \(i \geq 1/\delta\), which would cause an undesirable division by zero. According to definition (4.1), there is a maximum number of epochs \(i_{\text{max}} = 1/\delta\) before the adaptations stop and therefore the network converges. For example, \(\delta = 0.002\) provides \(i_{\text{max}} = 500\). We discuss the convergence properties further in Section 4.4.1.

**Robot-node allocation**

After adapting each path towards the viewpoint region \(Z^k\), the algorithm then only allocates one (or none) of the robots to the node and only this robot keeps their adapted path. The selection is performed by greedily selecting the robot that has used the least fraction of its budget after performing the adaptation (Algorithm 4.1 line 11). If no robot meets their travel budget then no paths are adapted. It is important to note that this allocation of robots to nodes is greedy just for the currently presented node and the particular learning epoch; these allocations are often modified in later learning epochs if a better allocation is found, and thus the overall SOM algorithm is not a greedy algorithm.

This allocation approach is motivated by the observation that in most cases an optimal solution should have each robot using approximately all of its travel budget. This is similar to what is typically seen in minimax problems. We wish to divide the work evenly between the robots as the learning progresses towards the final solution, such that a natural partitioning is found between the robots. Conversely, unbalanced path growth is likely to result in poor partitioning, such as when planning for the robots...
4.3 Self-organising map algorithm

sequentially (as seen in the experiments in Section 4.4.2).

Other possible allocation policies could also be appropriate here, such as the Hungarian algorithm. However, we believe it is better to have a fast and simple allocation policy, such as the greedy policy. This is because the actual reward or cost of each allocation is difficult to measure due to the flow-on effect of optimising *sequences* of viewpoints. Heuristic approaches are therefore appropriate, and suboptimal allocations can be quickly modified again in later epochs.

**Path regeneration**

At the end of each epoch (Algorithm 4.1 line 13), waypoints that are no longer useful are removed. A waypoint is useful if it is within at least one of the viewpoint regions. If multiple waypoints are within a viewpoint region then only one waypoint is randomly selected to remain, since there is no additional reward for multiple observations of a node. This also ensures the cardinal length of the paths do not grow beyond $N$ and therefore the computation time complexity at each iteration is bounded (see Section 4.4).

If all waypoints for a robot are removed, then the robot’s path is reinitialised following the procedure described in Section 4.3.2 with a new unique arbitrary node. This will not occur regularly in non-trivial problems since the robot will typically be allocated at least to its arbitrary starting node.

**Start and end conditions**

If the problem formulation specifies fixed start and/or end positions for the robots, then fixed waypoints are added to each path at these positions. If any of the fixed waypoints are selected as a winner, then the waypoint is duplicated (Algorithm 4.2 lines 9–10) and the new waypoint is adapted instead of the fixed waypoint (line 11). Fixed waypoints are not moved during the neighbourhood adaptations (line 19). Additionally, during each epoch, an adaptation is performed towards each fixed node—equivalent to if there was a singleton viewpoint region at each fixed waypoint (Algorithm 4.1 line 3). This ensures the waypoints with low cardinal distance to the fixed nodes maintain a minimal Euclidean distance to neighbouring fixed nodes.
4.4 Analysis

Adaptation parameter

The attraction between neighbouring waypoints during each adaptation is dependent on the $\sigma$ parameter of the neighbourhood function (Algorithm 4.2 line 19). When $\sigma$ is large then several waypoints will typically move a large distance during each adaptation and therefore large global adjustments are made to the solution paths. Conversely, when $\sigma$ is small then only waypoints within a small neighbourhood of the winner will move and therefore only small local refinements are made to the solution paths. Convergence of the algorithm is controlled by initialising $\sigma$ to an input parameter $\sigma_0$ and then cooling $\sigma$ after each epoch at a rate determined by an input parameter $\delta$ (Algorithm 4.1 line 16). Therefore, the number of epochs before the solutions reach a steady state are determined by $\sigma_0$ and $\delta$.

In online scenarios, new observations will typically cause minor adjustments to the objective function and therefore only minor local refinements to the previous solution are required. Online replanning can therefore be performed more efficiently by using the previous solution paths as the initial paths and initialising $\sigma$ to a lower $\sigma_0$ value. We demonstrate suitability for online replanning in the Section 4.6 experiments.

4.4 Analysis

This section provides a theoretical analysis of the algorithm’s runtime complexity and convergence, and then empirical analysis of the behaviour of the algorithm for various random environments. Further experiments are shown later in Sections 4.5 and 4.6 that focus on active perception of 3D point clouds in offline and online scenarios.

4.4.1 Theoretical analysis

Runtime complexity

The runtime complexity of the algorithm is polynomial in the number of nodes to be observed and the magnitude of the relative weighting of rewards. We formally state and prove this result as follows. Lemma 4.1 states the runtime complexity for
each epoch. Lemma 4.2 states the maximum number of epochs is constant, assuming
a given cooling schedule. These results are combined in Theorem 4.1 to state the
runtime complexity of the algorithm. We then remark on implications of this result.

**Lemma 4.1** (Runtime per epoch). The runtime complexity for each epoch is upper
bounded by

\[ O\left( \left( \sum_{i=1}^{R} |X^i| \right) N' \right) \leq O\left( N'^2 \left( \frac{\text{MAX}\{w^k\}}{\text{GCD}\{w^k\}} \right)^2 \right), \]

where \(|X^i|\) is the number of waypoints in the path for robot \(i\), \(N\) is the number of
viewpoint regions and \(N'\) is the number of viewpoint regions after duplication to take
into account the rewards \(\{w^k\}\).

**Proof.** The adaptation function (Algorithm 4.2) has runtime \(O(|X^i|)\), where \(|X^i|\)
is the number of waypoints in the current path for robot \(i\). In the inner-loop of
Algorithm 4.1 (lines 8–10), adaptation is called once per robot, and thus the runtime
for lines 8–10 is \(O(\sum_{i=1}^{R} |X^i|)\). Since only one waypoint is allocated to a node during
each epoch, and the regeneration step removes all waypoints not allocated to a node
at the end of each epoch (Algorithm 4.1 line 13), it holds that \(\sum_{i=1}^{R} |X^i| \leq N\) at
the end of each epoch. At most \(N'\) new waypoints are added during each epoch (if
all winners are edges), and thus \(\sum_{i=1}^{R} |X^i|\) is upper bounded by \(N + N'\). Thus, the
runtime for the line 8 loop is bounded by \(O(N + N') = O(N')\).

In each epoch, this is repeated for each duplicated node \((N')\) and any fixed start
or end nodes (up to \(2R\), if applicable). Thus, the runtime for each epoch is bounded
by \(O((\sum_{i=1}^{R} |X^i|)(N' + R)) \leq O(N'(N' + R))\). The \(R\) term only exists for problem
instances that specify fixed waypoints for start and end conditions. Furthermore,
\(R \ll N \leq N'\) for non-trivial problems; therefore, the \(R\) term is negligible. Thus, the
runtime for each epoch is bounded by \(O(N'^2)\).

Each viewpoint region is duplicated up to \(\frac{\text{MAX}\{w^k\}}{\text{GCD}\{w^k\}}\) times and thus
\(N' \leq N \frac{\text{MAX}\{w^k\}}{\text{GCD}\{w^k\}}\). Therefore, \(O(N'^2) = O(N^2(\frac{\text{MAX}\{w^k\}}{\text{GCD}\{w^k\}})^2).\)

**Lemma 4.2** (Convergence guarantee). The algorithm is guaranteed to converge
within \(i_{\text{max}} = 1/\delta\) epochs, where the gain decreasing rate \(\delta\) is a fixed parameter
of the algorithm.
4.4 Analysis

Proof. The neighbourhood function \( f(\sigma, l) \), as defined in (4.1), will become 0 for all \( l \) when the number of learning epochs \( i \geq 1/\delta \). When this occurs, all of the waypoints will remain at their current positions and therefore the network will not evolve any further. □

Theorem 4.1 (Runtime of SOM). The runtime complexity of Algorithm 4.1 is upper bounded by

\[
\mathcal{O}\left(\left(\sum_{i=1}^{R}|X^i|\right)N'\right) \leq \mathcal{O}\left(N'^2\left(\frac{\text{MAX}\{\{w^k\}\}}{\text{GCD}\{\{w^k\}\}}\right)^2\right),
\]

where \( |X^i| \) is the number of waypoints in the path for robot \( i \), \( N \) is the number of viewpoint regions and \( N' \) is the number of viewpoint regions after duplication to take into account the rewards \( \{w^k\} \).

Proof. Lemma 4.2 states the maximum number of epochs is constant, and thus the runtime complexity is a constant multiple of the epoch runtime given in Lemma 4.1. □

Remark 4.1 (Runtime dependence on \( R \)). Interestingly, the derived upper bound on runtime \( \mathcal{O}(N'^2) \) does not directly depend on the number of robots \( R \), and is instead dominated by properties of the environment. The key to the derivation of this bound is that each viewpoint region is allocated to a maximum of one robot during each epoch, and therefore the maximum total number of waypoints is independent of \( R \). This results in the line 8 loop having a runtime bounded by \( \mathcal{O}(N') \), as described in the proof of Lemma 4.1, which does not directly depend on \( R \). If, in an alternative algorithm, more than one robot could be allocated to a node, the line 8 loop runtime bound would increase to \( \mathcal{O}(RN') \), which is instead linear in \( R \).

However, it is important to note that the tighter bound \( \mathcal{O}((\sum_{i=1}^{R}|X^i|)N') \) is linear in \( \sum_{i=1}^{R}|X^i| \). Thus, if the team plans to observe a larger number of nodes, then the runtime will increase. There are several contributing factors that affect the number of observed nodes, including the number of robots \( R \), the travel budgets, the fixed start and end positions, and the distribution of nodes and rewards in the environment. Importantly, the number of observed nodes, and therefore the runtime, will typically be sublinear in \( R \), which we confirm empirically in Section 4.4.2. △
**Remark 4.2 (Early convergence).** Lemma 4.2 defines an upper bound on the number of epochs; though, in practice, convergence will typically occur much sooner than $i_{\text{max}}$ epochs. Early convergence occurs for a number of reasons, which we summarise here, and elaborate on further in the Appendix of Best et al. (2018b). Empirical evidence of convergence is provided in Section 4.4.2. Related discussions of convergence may be found in Cochrane and Beasley (2003); Faigl and Hollinger (2018); Tucci and Raugi (2010).

Most importantly, for the neighbours of the winner (i.e., $l > 0$), the neighbourhood function $f(\sigma, l)$ pragmatically becomes zero much sooner than epoch $i_{\text{max}}$. For example, when using IEEE 754 arithmetic, with $\sigma_0 = 4$ and $\delta = 0.002$ (therefore $i_{\text{max}} = 500$), the neighbourhood function becomes zero for $l > 0$ at epoch $i = 68$. When this point is reached, the winners $x^*$ are adapted with $f(\sigma, 0) = 1$, but the neighbours are never adapted. It is possible for the winners $x^*$ to continue adapting until epoch $i_{\text{max}}$, however this is unlikely to occur due to the travel budgets being exhausted.

Furthermore, our SOM algorithm maintains the best solution $\{X^i\}^*$ at the end of each epoch (Algorithm 4.1 line 15), which is likely to converge before the network $\{X^i\}$ converges. This is because the network may oscillate between different nodes due to the random permutation of $n^k$ (Algorithm 4.1 line 6), while the best found solution remains constant.\[\triangle\]

**Optimality**

Self-organising map algorithms, including ours, are stochastic learning procedures that can guarantee convergence in polynomial time, but unfortunately cannot guarantee optimality in finite time. These algorithms therefore are heuristic algorithms for giving approximate solutions to NP-hard problems in polynomial time. The algorithm does however have the advantage of being anytime, i.e., the algorithm can be halted early, since all intermediate solutions are feasible solutions. The parameters $\sigma_0$ and $\delta$ can also be tuned to strike a balance between optimality and computation time, and we exploit this property in the Section 4.6 formulation for online scenarios. The computation time can also be reduced, potentially at the cost of solution quality,
by reducing $\frac{\max(w^k)}{\gcd(w^k)}$ by rounding the rewards to multiples of a divisor greater than $\gcd(w^k)$.

### 4.4.2 Empirical analysis

Simulated experiments were performed to analyse the behaviour of the algorithm under various conditions. Since the problem is new, we do not have algorithms for direct comparison. Therefore, we compare to restricted versions of our algorithm with some components removed to analyse how the various algorithmic components contribute to generating high-quality solutions. We compare (1) planning using the joint multi-robot optimisation compared to sequential optimisation, (2) planning with and without the viewpoint rewards, and (3) planning with the viewpoint polygons compared to singular points. We also demonstrate the convergence and anytime properties. The algorithm plans paths through 100 random environments consisting of random sets of polygons. An example environment is illustrated in Figure 4.3.

The parameters are as follows, except where varied for specific experiments. The
environments are a continuous $1000 \times 1000$ space. There are 80 polygons with random centre points and from 3 to 6 vertices spaced at equal angles around the centre. The distance from the centre to each vertex is random between 40 and 120. Rewards are exponentially distributed between 1 and 4 and rounded to the nearest integer, such that few regions have high rewards. There are 3 robots with budgets 800, speeds 1 and a closed-loop path topology with free start locations. In all cases, convergence was reached in 70 epochs. The same sample environments are used for each pair of methods and a single-tailed paired $t$-test was performed for each comparison. For these experiments we use $\sigma_0 = 1$ and $\delta = 0.001$.

Multiple robots

Figure 4.4a shows the rewards collected by planning using the proposed method, which jointly optimises multiple robots, compared to planning for the robots sequentially. The sequential method performs the SOM algorithm for a single robot at a time, with each robot ignoring the nodes selected by previous robots. The two methods were compared for 2 to 6 robots, where the budgets were uniform and summed to 2400. The simulations show the proposed approach has the best performance in all cases, and these results were statistically significant ($p < 0.01$) in all cases except $R = 4$. The largest improvements were for planning for smaller teams, because in these cases the performance is greatly influenced by effective partitioning of the workspace between the robots, which can be more effectively optimised when planning for all robots jointly.

Observation rewards

Figure 4.4b shows the simulation results for planning using the proposed duplication approach compared to assuming uniform rewards. The rewards are exponentially distributed between 1 and $\bar{w}$ with lower rewards more likely, and $\bar{w}$ varied from 2 to 32. For this comparison method, the non-uniform rewards are not known to the planner, but the resulting solution paths are evaluated with respect to the non-uniform reward model. These experiments were performed with a budget of 600 and an average polygon size of 40. In all cases, planning with the proposed approach
4.4 Analysis

(a) Jointly planning for all robots following the proposed method compared to sequentially planning each robot.

(b) Planning while considering the actual observation rewards compared to planning assuming uniform rewards. The shown rewards were evaluated for the actual observation rewards.

(c) Planning while considering the viewpoint regions compared to planning while considering only the centroid of the polygons. The shown rewards were evaluated for the full polygons.

(d) Planning while considering the viewpoint regions compared to planning while considering only the centroid of the polygons. The shown rewards were evaluated for the full polygons.

Figure 4.4 – Simulation results for random environments under various scenarios and comparison methods. Vertical axes shows performance as the ratio of the achieved weighted sum of nodes visited to the weighted sum of all nodes in the environment. Box plots show lower bound, lower quartile, median, upper quartile, and upper bound for 100 sample environments.

improved the performance, and these results were statistically significant ($p < 0.01$). Greater improvements were achieved when the maximum reward was large since the proposed approach is more likely to select nodes with large rewards.

Viewpoint regions

We analyse the value of the proposed planning with continuous polygonal viewpoint regions compared to planning with single points at the region centres. Figure 4.4c compares these two methods with a varying number of nodes and Figure 4.4d has a varying average polygon size. The proposed planner outperformed the single point planner for all number of nodes and when the polygon size $\geq 20$, and these results
were statistically significant ($p < 0.01$). When the polygon size was very small (10) it was sufficient to plan by approximating the polygons as single points. The proposed approach achieved greater improvements when the number of nodes and the size of the polygons were large. In these cases, the algorithm can more effectively take advantage of being able to optimise the waypoint locations.

Convergence

In Figure 4.5 we illustrate the convergence of the algorithm for repeated trials of a single random problem instance. In all trials, the intermediate solutions made incremental improvements and converged towards the final solution, which was reached before 45 epochs. This convergence demonstrates that the algorithm is anytime since each intermediate solution is a feasible solution. This is an important property in practical applications where the computation budget is not known in advance, and therefore the algorithm may need to be halted early and return the best solution found so far. If the computation budget is known in advance, then the parameters may be tuned to meet this requirement; we discuss this idea further in Section 4.6.2.

Computation time

The SOM algorithm was implemented in MATLAB and the simulations were performed on a standard desktop computer with an Intel i7 processor on a single core. The runtime varied from 0.5 s to 30 s depending on the scenario. The trends agreed
with the theoretical analysis such that runtime increased with the number of nodes and maximum weight. Runtime increased with the number of robots, however this increase was sublinear. As discussed in Remark 4.1, this small runtime dependence on the number of robots is likely due to the multi-robot planning achieving greater performance and therefore a larger number of nodes are visited. Runtime was dominated (≈ 70%) by the winner selection and the waypoint usefulness evaluation, since these geometric computations are relatively expensive. Our implementation has not been thoroughly optimised since our primary focus was on validating the feasibility of the approach. Therefore, the runtime can be significantly improved by the implementation, as well as by using approximations, such as decreasing the number of polygon vertices or approximating polygons as discs.

4.5 Active perception of 3D point-cloud objects

Our primary motivation for the proposed problem formulation and SOM algorithm is active perception tasks that aim to observe a set of object parts in a large environment. These problems rely on prior observations or a predefined belief of the environment, which may have come from a coarse scan with noisy sensors. The aim is now to perform a more informative or complete scan of the environment, and this process may be repeated. In this section, we demonstrate how the algorithm can be applied to this class of active perception tasks. For these experiments we assume the observation regions and rewards are known in advance by an offline planner, while in Section 4.6 we extend the formulation for closed-loop scenarios where this information is discovered online.

We consider example scenarios using three variations of an outdoor scene from a real 3D point-cloud dataset first presented in Patten et al. (2015), which was also used earlier in Section 3.6. The data was recorded with a Velodyne laser scanner mounted on a robot pictured in Figure 4.6. Observations were made from several locations and fused together. The three scenes consist of 12, 15 and 18 objects spread around a 40 m × 40 m environment, including trees, tables, chairs, bins and a motorbike. The dataset has been used previously for testing object classification algorithms (Patten et al., 2015).
The environment is represented by a set of parts in a 3D point cloud with associated viewpoints and rewards. Examples of the segmentation and viewpoint regions are shown earlier in Figures 1.4 and 4.1. The point cloud processing method is detailed below in Section 4.5.1 and summarised here as follows: (1) oversegment the environment into parts, (2) estimate self-occlusion free viewpoint regions for each part, and (3) define the rewards as the discriminability between parts.

Our general objective function formulation provides a convenient way of expressing the viewpoint sensitivity of perception algorithms. The perception model defined here is an example instantiation of the viewpoint regions and rewards, and is intended to be generic for the purpose of evaluating the performance of our proposed planning algorithm. We emphasise that the proposed SOM algorithm is not limited to this perception model, but rather the model can be adapted to suit the requirements of a perception task.

### 4.5.1 Observation model for 3D point-cloud objects

The point cloud of the environment is segmented into parts by removing the ground plane and then segmenting into objects using region growing. Each object is oversegmented into 5 parts using \(k\)-means clustering on the set of 3D points associated with the object.
A viewpoint region is defined for each part by considering the sensing range, as well as occlusions caused by other parts of the object. These viewpoint regions could be computed in many different ways, but we describe our implementation for these experiments as follows. An illustration is provided in Figure 4.7 for computing the viewpoint region (purple shape on right) associated with an object part (purple point cloud on left). First, we compute the set of vectors (black lines), which represent occlusions. These vectors are from all points within the object part to all points in other parts of the same object (grey point cloud). Any of these vectors that have a vertical angle outside the range of $-\pi/8$ to $\pi/8$ are removed since they are unlikely to represent an occlusion. Next, the horizontal angles of all the remaining vectors are considered to represent occluded angles. Then, we find the largest window of angles that contains less than 10% of the occluded angles. The viewing angle range (between dashed red lines) is defined as the middle third of this window. The useful sensing range is defined as 1 to 4 m. The viewpoint region (purple shape) is defined as the intersection of the horizontal viewing angle range and the sensing range, measured relative to the part’s centroid. For efficiency, this region is approximated by a polygon with 6 to 8 vertices.

We define the rewards as the discriminability of each part in a feature space. Parts with a higher discriminability contain more unique features and therefore are more
likely to provide useful information to an object classifier. To measure discriminability, we perform feature extraction for each part, calculate the distance to all other parts in feature space, and normalise for each object. We compare each part to all other parts in the environment; alternatively each part could be compared to an object library. For the feature extraction, we use the ensemble of shape functions (ESF) global feature descriptor (Wohlkinger and Vincze, 2011), which is commonly used for object classification tasks (Patten et al., 2016; Wohlkinger et al., 2012). Discriminability is measured as the exponential of the sum of Mahalanobis distances in the feature space between each part and every other part. Each object is considered to be equally important, and therefore the sum of rewards for each object is normalised to 10. Each reward is rounded to the nearest integer. The rewards for the datasets ranged from 1 to 10 with 1 or 2 more likely.

Model validation

Here, we provide a short validation of this example model instantiation by illustrating how it maps to an existing perception technique. In particular, we show how it maps to an existing object recognition perception model (Patten et al., 2016), which is an instance of the general framework in Wohlkinger et al. (2012).

We implement a simplified version of the model by Patten et al. (2016) as follows. First, we build an offline database of object models, and then an observed object is probabilistically classified as an instance of an object in the database. The database is built by making several point cloud observations of each object from different angles. For each observation, the ESF global feature descriptor is stored. To classify an observed object, the ESF descriptor of the observed point cloud is computed and the Mahalanobis distance is measured to each database object and viewpoint. For each database object, the distance to the viewpoint with the closest distance is stored. A probability distribution is defined over the set of objects by computing a negative exponential of the closest distances and normalising. Multiple observations of the same object from different viewpoints are fused using Bayes’ rule. The objective function is the total entropy, defined as the object classification entropy summed over all objects.
4.5 Active perception of 3D point-cloud objects

Figure 4.8 – Comparison of path utility between the example perception model defined in Section 4.5.1 suitable for the SOM formulation (horizontal axis) and the total entropy when using an existing object recognition model described in Section 4.5.1 (vertical axis). Evaluated for 100 random paths generated with SOM algorithm. Linear trendline shown in red.

For this comparison we use the high-clutter dataset and generate a set of 100 single-robot paths with varying reward. The random paths were generated by running the SOM algorithm with randomly varying parameters. The utility of each path was evaluated using the example objective function in Section 4.5.1. For the comparison, the recognition performance is estimated using the point cloud observations in the dataset at viewpoints that are closest to each path.

The results are shown in Figure 4.8. There is a clear correlation between the rewards computed using the proposed formulation and the comparison model (linear trendline has $r^2 = 0.67$). The correlation is strong enough to indicate there is a reasonable mapping between the two models, and that the viewpoint region definition is a suitable model for evaluating the performance of our planning algorithm.

We note that predictions of the total entropy cannot be directly applied as an objective function for the SOM algorithm. This is due to the requirement of the SOM to have the reward function encoded via a set of viewpoint region weights. This formulation represents dependencies between viewpoints in a more restricted manner that does not directly allow representing the dependencies that the total entropy allows in general. The advantage, though, is that the viewpoint region rewards are a reasonable surrogate for the total entropy that is faster to compute and is in a form suitable for
specialised algorithms such as SOMs.

4.5.2 Results

We analyse four example scenarios illustrated in Figure 4.9, for three environments with varying clutter. In the scenarios, we plan for: (1) a single robot in the low clutter environment, (2) two robots in medium clutter, where one robot has double the budget, (3) three robots in high clutter, where the robots have speeds 2, 1.5 and 1, and (4) five robots in high clutter, where the robots have equal speeds and budgets. In these scenarios, the start positions of the robots are unconstrained but the robots must end at their start position; scenarios with fixed start positions are trialled later in Section 4.6. Planning was repeated 100 times each to measure the planning consistency.

Scenario 1: Low clutter, single robot

In the first scenario, the robot observed a weighted sum of 132 nodes, averaged over 100 trials, out of the maximum possible 151 nodes. The performance was consistent over the 100 trials, with a standard deviation of 2.13 weighted nodes. The worst plan had 124 and the best had 135. The average runtime was 6 s with standard deviation 0.1 s. An example solution is shown in Figure 4.9a. Each object has at least one of its parts observed. The parts not selected were in the bottom left and top left, which is expected since the time to travel to these regions is relatively high. The waypoints within the selected regions naturally found locations near the edges of the regions and closer to the other regions, which implicitly minimises the travel time. All of the parts in the top right were selected, even though they are further from the other objects, since there is a significant reward to be gained by visiting two objects in close proximity.

Scenario 2: Medium clutter, two robots

The second scenario was planned for two robots with different budgets. Figure 4.9b shows that the algorithm finds a natural partitioning between the robots in the same
4.5 Active perception of 3D point-cloud objects

(a) Low clutter (12 object) environment with 1 robot.

(b) Medium clutter (15 object) environment with 2 robots. Right robot has 2× budget.

(c) High clutter (18 object) environment with 3 robots. Left robot has 2× speed; bottom right has 1.5× speed.

(d) High clutter (18 object) environment with 5 robots. Robots have equal budget and speed.

Figure 4.9 – Example scenarios and solution paths (blue) for teams of robots. Object parts are shown in the coloured point clouds. Viewpoint regions are coloured black (low reward), orange (medium) and yellow (high).

The ratio of the travel budgets. The implicit partitioning naturally shared some of the objects between the two robots where the object parts were closer to a different robot. The planner typically avoided the object in the bottom left since there is a significant travel cost to reach those regions. The 100 trials had a weighted sum of 174 nodes on average, with a standard deviation of 3.4, out of the maximum 189. The worst plan had 156 while the best had 177, showing the distribution of plans was skewed.
towards the best performing plans. The average runtime was 11.7 s with the standard deviation 0.2 s.

**Scenario 3: High clutter, three robots**

A similar partitioning was achieved in the third scenario, shown in Figure 4.9c, for three robots with varying speeds in the most cluttered environment. The size of the implicit partitions are proportional to the speeds of the robots. The centre was well covered since several parts are observed at once from these locations and therefore have high reward. The average sum of weighted nodes was 199.2 out of the maximum 221, with standard deviation 9.1, worst case 175 and best case 211. The average runtime was 17.6 s with a standard deviation of 0.6 s. The performance was almost as consistent in this more complex scenario, and the solution paths have credible partitioning between multiple robots, selected lower cost locations within regions and favoured high-reward locations with overlapping viewpoint regions.

**Scenario 4: High clutter, five robots**

The fourth scenario trialled five robots in the high clutter environment, shown in Figure 4.9d. The robots were given equal budgets such that it was just enough to be possible to collect 100% of the rewards. In the Figure 4.9d example trial, the robots have successfully shared the workload to find 5 approximately equal length paths that collectively visit all of the goal regions. Over the 100 trials, the average sum of weighted nodes was 218.8 out of the maximum 221, with standard deviation 3.6, worst case 206 and best case 221. Full coverage was achieved by 70 of the trials. For the trials that achieved suboptimal results, the viewpoint regions in the bottom left of the environment were more often missed since there is less incentive to visit that area. The average runtime was 22.1 s with a standard deviation of 0.6 s.

### 4.5.3 Comparison to Dec-MCTS (Chapter 3)

In these experiments, we investigate the benefits of planning over continuous space by comparing the proposed SOM planner to the Dec-MCTS algorithm of Chapter 3.
There are several important differences between Dec-MCTS and our SOM approach. In particular, Dec-MCTS is decentralised, applicable to general objective functions and motion models, provides theoretical guarantees, and requires discretising the action space. While the SOM approach is centralised, is an efficient solution for a particular problem formulation, and effectively plans over continuous space. While these differences make it difficult to demonstrate a fair performance comparison, we show experiments here that highlight the benefit of planning over continuous space for our problem formulation (as in the SOM algorithm) rather than requiring a discretisation of the environment (as in Dec-MCTS).

These experiments were performed with 3 robots using the high-clutter dataset shown previously in Figure 4.9(c,d). In these experiments, the problem is discretised for Dec-MCTS using a probabilistic roadmap (PRM) with vertices $V$ randomly placed in the viewpoint regions. Also, since Dec-MCTS requires a fixed start location, these experiments were performed using fixed start positions spread out near the centre of the environment, and the end positions are variables to be optimised by the planner. The experiments were performed with varying number of PRM vertices $V$ for Dec-MCTS. Each scenario was repeated for 100 trials with this single problem instance. For each trial, a new set of PRM vertices $V$ was randomly generated. Dec-MCTS was run until convergence was observed, which was between several seconds and several minutes depending on the size of $V$. The SOM trials took 15 s each.

The results are shown in Figure 4.10. The rewards collected by Dec-MCTS clearly improves when using a finer discretisation. This is because having more roadmap vertices $V$ increases the probability of vertices being placed at valuable positions, e.g., positions that intersect multiple viewpoint regions and have relatively low travel-cost to other valuable vertices. On the other hand, the proposed SOM approach searches over the continuous space to adaptively find valuable positions for the path waypoints. This allowed the SOM approach to significantly outperform Dec-MCTS in all cases. Theoretically, Dec-MCTS would achieve the performance of SOM given a sufficient discretisation, but the computation and memory requirements would be intractable.
4.6 Online exploration and active perception

In this section, we generalise the active perception scenario in the previous section to online scenarios for a team of robots. The robots make long-range 3D point-cloud observations to learn viewpoint regions and move to selected viewpoint regions to collect these rewards by observing the object parts at close range. The robots must plan to balance their workload between visiting the currently known viewpoint regions and making observations to discover viewpoint goal regions. This is achieved by introducing exploration rewards as new viewpoint regions in unexplored areas of the environment. First, we formalise this observation model and planning scenario, then present results that illustrate the behaviour of the algorithm in online settings and highlight advantages of this formulation in comparison to short-horizon planning.

4.6.1 Online planning scenario

For these experiments, each robot has two 3D point-cloud sensing modalities such that high-quality observations are made at close range and coarse observations are made at long range. The close-range sensor is used to fulfil the primary perception task and has the same observation model as in Section 4.5.1. The long-range sensor
is used to discover new objects and associated viewpoint regions and rewards. These two modalities could be provided by two separate sensors or by a single sensor where a close range is required to achieve a desired resolution. The viewpoint regions for the primary perception task are generated as described in Section 4.5.1 based on the point clouds that have been observed by the long-range sensor. Exploration is encouraged by introducing new viewpoint regions and rewards. This is achieved by placing a uniform grid of goals in the unexplored areas. The density of goals and their rewards can be selected to achieve a desired balance between exploration and exploitation. Each exploration goal has an associated circular viewpoint region with radius equal to the close sensing range and the rewards are uniformly set to 1. We use the close range rather than the long range for the exploration nodes since this results in a more accurate prediction of the travel distance required to observe discovered objects at close range. In these experiments, we use the above definition for exploration goals since it avoids making strong assumptions about the environment. However, if more prior knowledge were available, such as a belief of non-uniform density of objects, then more elaborate formulations could be used instead.

The simulations cycle between four phases: (1) compute the viewpoint regions for both the primary perception task and exploration, (2) plan the paths for the team of robots with the SOM algorithm, (3) drive the robots a fixed distance along the planned paths, and (4) make new point-cloud observations with the long-range and short-range modalities. A team of five robots move through the 140×50 m environment shown in Figure 4.11, which consists of the medium-, low- and high-clutter 3D point-cloud datasets (from Section 4.5) placed side-by-side from left to right in an enlarged environment. The long-range sensing range is 15 m and the close range is 4 m. These observations are simulated using the dataset by truncating the Velodyne measurements. All robots have an initial travel budget of 100 m travel distance, make observations at 1 m intervals along the path and replan after every 15 m. After each replanning phase, the remaining travel budget is reduced by 15 m. Each robot has a fixed start position and known current position for each replanning phase.

The planner uses an open-path graph topology with fixed start positions (as defined in Section 4.3.2). For the first planning round we set \( \sigma_0 = 4 \) and use arbitrary initial plans around the start positions. Online replanning is performed more effectively by
adapting the previously planned paths and using $\sigma_0 = 2$. We use $\delta = 0.002$ for most experiments, and analyse the effect of these parameters in Section 4.6.2.

### 4.6.2 Results

The following results demonstrate: (1) the algorithm achieves better performance when using a long planning horizon compared to a short horizon, (2) the effect of exploration reward density on performance, (3) the algorithm achieves comparable performance when planning online with partial information to when planning offline with full information, and (4) the algorithm efficiently adapts previous solutions when replanning.

Figure 4.11 illustrates an example of the behaviour of the algorithm when replanning. (a) Initially, only a single object has been observed from the start positions. Robots 1 and 2 cooperate by planning to observe both sides of this object at close range before proceeding to explore the top left of the environment. Robots 3, 4 and 5 evenly spread out to explore the right side of the environment. (b) After 15 m has been travelled by each robot, more objects are discovered by the long range sensor. Robots 1, 2 and 3 make minor refinements to their plans to make close-range observations of the new objects. (c) After 45 m, robot 3 discovers several more objects in the middle. Rather than robot 3 visiting these discovered goals itself, it instead decides to continue exploring to the right since robot 2 plans to visit these goals later. (d) A large number of objects are discovered on the right and robots 3, 4 and 5 cooperate to share these goals. (e) Once the budgets are exhausted, the robots have explored nearly all of the environment while also visiting 344 out of 420 close-range weighted goals. The robots successfully cooperated by rarely crossing paths or making duplicated observations.

#### Planning horizon

Figure 4.12 compares online planning with the entire mission (100 m) as the planning horizon to when planning with shorter horizons. The longer planning horizons result in a significantly improved performance over the shorter horizons. The reason for this is illustrated in Figure 4.13. For the shortest horizon (a), the objects on the left are
4.6 Online exploration and active perception

(a) Initial plans. Plan observes 254/330 weighted nodes.

(b) 15m travelled. Plan observes 274/362 weighted nodes.

(c) 45m travelled. Plan observes 202/294 weighted nodes.

(d) 75m travelled. Plan observes 183/262 weighted nodes.

(e) Final executed paths (100m). The robots observed 309/330 exploration goals and 344/420 primary nodes.

Figure 4.11 – Example adaptive plan for the online experiments. Dotted lines are executed paths and solid lines are planned paths. Yellow regions have been explored. Point-clouds are observed at close-range (brown) and long-range (pink).

Figure 4.12 – Comparison of planners with different planning horizons. The ‘budget’ horizon optimises the entire remaining budget of each robot. Each scenario was performed 10 times.
4.6 Online exploration and active perception

(a) 15m planning horizon. Medium exploration density. The robots observed 267/420 primary nodes.

(b) 45m planning horizon. Medium exploration density. The robots observed 303/420 primary nodes.

(c) Full planning horizon. Medium exploration density. The robots observed 332/420 primary nodes.

(d) Full planning horizon. Low exploration density. The robots observed 255/420 primary nodes.

(e) Full point-cloud is available offline. Full planning horizon. The robots observed 376/420 primary nodes.

Figure 4.13 – Example paths executed for (a-c) different planning horizons, (c-d) exploration reward densities and (e) offline full-observability.

discovered first and since these discovered goals cannot be satisfied by a single robot with a 15m budget, the other robots also decided to visit these objects. As a result, the right side of the environment is never explored. As the mission progressed, the robots were left with no goals reachable within their budgets since they were already visited by other robots. Conversely, the longer planning horizons (b,c) enabled the robots to cooperate to explore the rest of the environment and visit the discovered objects. The longest planning horizon (c) achieved the best results since two robots managed to reach the dense group of objects on the right.
4.6 Online exploration and active perception

Figure 4.14 – Comparison of planning performance for different densities of exploration rewards. The full information scenario shows the performance of an offline planner with all primary viewpoint regions and rewards known in advance. Each scenario was performed 10 times.

Exploration reward density

The algorithm generates paths that naturally balance between exploring the environment to discover new objects and visiting the objects at close-range to make high-quality observations. This balance can be influenced by selecting the density of exploration goals. If the density is low, as in Figure 4.13d, then the robots have little incentive to visit unexplored regions and will instead focus on observing discovered objects. The robots in (d) did not perform well since they only just reached the objects on the right. However, if there were no objects in the right of the environment, then (d) would have outperformed the higher-density planners since it achieved better coverage of the discovered objects. This trend is also illustrated in Figure 4.14: the higher exploration density scenarios outperformed the lower density scenarios in most trials for this environment. We note that a similar behaviour of balancing between exploring and exploiting occurs if the exploration goal rewards are varied rather than the density.

Partial information

For the scenario in Figure 4.11e and the yellow column of Figure 4.14, the planner had full knowledge of all of the goals offline and the algorithm is able to exploit this information to outperform the other scenarios. In Figure 4.11e we see the robots do not need to spend their budget exploring empty space and instead take the most
direct routes to their selected viewpoint regions. However, the partial-information scenarios still achieved reasonable results, despite not having access to this valuable information upfront. The high exploration density scenario collected on average 95% of the reward collected by the full-information scenario.

**Adaptive replanning**

We now analyse the benefits of adapting the previous solution when replanning compared to restarting the algorithm from the beginning. The results are shown in Figure 4.15 for various combinations of parameter values. The parameter $\delta$ has the largest effect on computation time since this parameter directly influences the number of iterations before $\sigma$ reaches the termination threshold; the $\delta = 0.002$ scenarios had an average runtime of 12s during each replanning step and the $\delta = 0.004$ scenarios performed replanning more efficiently with a runtime of 3s. The $\sigma_0$ parameter directly affects the ratio of the time spent making large global changes (when $\sigma$ is large) to the time spent making smaller local refinements (when $\sigma$ is small).

In all of the scenarios, reusing the previous solutions helped the algorithm perform replanning and made a statistically significant improvement to the collected rewards ($t$-test $p < 0.001$). There was no significant difference between the rewards collected for the different combinations of parameters when adapting the previous solution.
even for cases with much fewer epochs ($\delta = 0.004$). When restarting the solution, the performance was poorer when the number of epochs was reduced, since it requires many iterations to adapt from the initial solution to a reasonable solution. There was a significant improvement ($p < 0.001$) for the ($\sigma_0 = 2, \delta = 0.004$) case over the ($\sigma_0 = 1, \delta = 0.004$) case when restarting the solution since the larger $\sigma$ values result in more global adaptations for reaching an initial reasonable solution. Overall, these results highlight that the algorithm can effectively adapt previous solutions so that replanning can be performed more efficiently. This is particularly advantageous in online scenarios where the plans need to adapt to small changes in the objectives as the robots make observations.

4.7 Summary

We have proposed a new formulation and approach for multi-robot active perception problems. The objectives are defined as a set of continuous viewpoint regions, and the robots coordinate to maximise coverage of these regions. Self-organising maps is a fitting choice for developing solution algorithms; they can select favourable observation locations within continuous regions, while simultaneously optimising the full paths of the robots. Optimising the full paths, i.e., planning over a long time horizon, results in significant performance improvements over greedy and short-horizon planning. Our new SOM formulation addresses scenarios with non-uniform observation rewards, budget constraints, polygonal observation regions and multiple robots. The algorithm has polynomial time-complexity, converges towards a final solution, and is anytime. Additionally, we demonstrated that the formulation is suitable for online scenarios where the objectives change over time and the planner needs to efficiently adapt the plans to meet the new requirements. We also showed how the planner can be used to balance between exploring the environment to obtain new information and making high-quality observations of known objects. Our implementation was unoptimised but still achieved reasonable clock time performance of milliseconds to seconds. Overall, our experimental results show that the proposed method enables multi-robot planning for budgeted active perception tasks with continuous sets of candidate viewpoints and multi-step planning horizons.
Chapter 5

Spatiotemporal optimal stopping for mission monitoring

In this chapter we formulate and solve the mission monitoring problem. In this problem, a monitor vehicle must remain in close proximity to an autonomous robot that stochastically follows a predicted trajectory. This problem arises in a diverse range of scenarios, such as autonomous underwater vehicles supervised by surface vessels, pedestrians monitored by aerial vehicles, and animals monitored by agricultural robots. The key problem characteristics we consider are that the monitor must remain stationary while observing the robot, robot motion is modelled in general as a stochastic process, and observations are modelled as a spatial probability distribution. We formulate this problem as a spatiotemporal generalisation of the well-known optimal stopping problems. We propose an optimal algorithm for this problem that has polynomial runtime.

5.1 Overview

Mission monitoring is a supervisory problem where a robot or a manually driven vehicle tracks the progress of an autonomous mobile robot or other agent in performing a pre-planned task. There are many examples of such tasks, including undersea surveys (Williams et al., 2012, 2015), environmental monitoring (Dunbabin and Marques,
autonomous farming (Oksanen and Visala, 2009; Ball et al., 2013) and planetary exploration (Peynot et al., 2014). Monitoring allows for rapid response to failures and to important information that the robot may discover during the progress of its mission (German et al., 2012; Hagen et al., 2008; Yilmaz et al., 2008; Khatib et al., 2016; Best and Moghadam, 2014). Additionally, the monitoring vehicle may augment mission capabilities by providing observations from external viewpoints, such as for accurate localisation and navigation (Fallon et al., 2010; Heppner et al., 2013; Klodt et al., 2015; Saska et al., 2014; Kottege and Zimmer, 2011) or online sensor calibration (Bongiorno et al., 2013). The motion of the robot is typically represented by a mission plan, which may be defined probabilistically to take into account uncertain vehicle dynamics, environment models and mission objectives (Karydis et al., 2015; Chiang et al., 2014; Aoude et al., 2013). In some cases, the monitor vehicle must remain stationary in order to observe or communicate with the robot. The monitor vehicle must decide where to stop, and when to move to the next observation location.

Classical optimal stopping problems (Chow et al., 1971), such as the well-known secretary problem, involve a binary choice; at each time point, the decision at hand is simply whether to stop or continue. If this choice can be repeated, the problem can be considered to be one-dimensional in the sense that it involves a choice of nonoverlapping intervals along a single dimension representing time. However, mission monitoring also involves spatial dimensions. We refer to this case as spatiotemporal optimal stopping. The goal of the work in this chapter is to develop complete algorithms for a spatiotemporal optimal stopping problem where the motion of the target robot and the observations in general are stochastic.

Our formulation is motivated by a variety of real-world scenarios, as discussed earlier in Section 1.3.3. Of particular interest is autonomous underwater vehicle (AUV) operations. Most AUVs in practice are supervised by powered surface vessels. The AUV navigates autonomously, often following a pre-planned trajectory with reasonable accuracy, but failures can occur that require human intervention. The AUV may also discover information of immediate value. Therefore, effective monitoring is relevant even if the robot is autonomous; monitoring allows operators to respond to failures and relevant information quickly.

Acoustic systems used for communication with the AUV have limited range and are
unreliable, and some operators must stop and deploy this communication equipment, with engines powered down, for maximum efficiency (Best and Anstee, 2014). In the simplest case, communication may be modelled deterministically with a fixed-range. However, this simple approach does not consider unpredictable hardware and environments; in practice, probabilistic models are typically required to account for physically realistic conditions (Hollinger et al., 2011b). An optimal stopping solution maximises the time spent communicating or observing effectively; this is achieved by choosing valuable observation locations and times, and reducing time spent unnecessarily travelling and stopping and starting the surface vessel.

A key focus of this chapter is the case where a mission is defined probabilistically. A probabilistic mission definition can take into account uncertainties such as unknown mission objectives, stochastic vehicle dynamics and imprecise environment models. Deterministic missions can often be generated directly from human-defined or automated plans, such as for AUV missions (Witt and Dunbabin, 2008; Rao and Williams, 2009; Das et al., 2010; Fang and Anstee, 2010; Faigl and Hollinger, 2014; Jones and Hollinger, 2017) or autonomous farming (Oksanen and Visala, 2009; Ball et al., 2013). In some applications, stochastic models of vehicle motion may be formulated as direct extensions of deterministic models, such as by adding model- or data-driven uncertainty to the trajectory (Karydis et al., 2015; Chiang et al., 2014; Peynot et al., 2014). In other cases, particular parameters of a mission are unknown, such as mission objectives or reactions to unforeseen events, leading to multi-modal predictions that are not direct extensions of a deterministic model (Aoude et al., 2013).

We propose a polynomial-time resolution-complete algorithm (complete with respect to the discretisation resolution) for the stochastic mission monitoring problem. Our algorithm generates an optimal nonoverlapping set of observation “cylinders” (in the simplest deterministic case) in the 3D configuration space consisting of two spatial dimensions and one time dimension, as illustrated in Figure 5.1. These cylinders represent a stationary observation range and time, and are linked by a path that respects motion constraints of the monitor platform. The objective is effective monitoring, defined as maximising the expected overlap time between the probabilistic observation regions and the stochastic mission trajectory. Hardware-setup time penalties are naturally modelled geometrically by modifying the cylinder heights when evaluating
5.1 Overview

Figure 5.1 – Geometric interpretation of the spatiotemporal optimal stopping problem. A sample robot trajectory (an AUV mission) is shown in blue and also projected onto a plane in the two spatial dimensions. An example monitor trajectory solution is overlaid. Cylinders represent effective monitoring range (in this case a fixed-range communication model) at stopping locations. Green stars represent parts of the mission that are not monitored.

trajectory overlap. Time and space are discretised, but fine resolution is feasible in practice. The algorithm first reduces the problem to a longest-path graph search, then passes a sweep-plane through the temporal dimension to compute a resolution-complete solution in polynomial time. We present an algorithm for the general probabilistic case as well as an elegant variant tuned for the deterministic case that runs more efficiently in practice.

In addition to analytical evaluation, we provide extensive simulation results for several example scenarios and realistic applications to demonstrate the relevance and applicability to real-world scenarios. In particular, we give implementation details and empirical analysis for applying the mission monitoring algorithm to two application case studies: (1) an AUV monitoring application using a probabilistic trajectory model with stochastic kinematics, localisation uncertainty, a closed-loop controller (de Wit et al., 2012) and a realistic underwater communication model (Hollinger et al., 2011b); and (2) a pedestrian monitoring application using a realistic trajectory prediction model (Appendix A) and observations with occlusions in a cluttered environment. We present results with data drawn from actual AUV missions (Best and Anstee, 2014),
5.1 Overview

A real pedestrian trajectory dataset (Lerner et al., 2007) and Monte Carlo simulations of example trajectory models. The simulations illustrate the behaviour and performance of the algorithm when planning with various modelling assumptions. Overall, the experiments highlight advantages of the probabilistic formulation, demonstrate that the algorithm admits a broad range of probabilistic trajectory prediction and probabilistic observation models for practical scenarios, as well as clock-time performance that shows the solution is viable for practical use in mission monitoring.

5.1.1 Mission monitoring variants

The mission monitoring problem was first posed in Best and Anstee (2014) along with preliminary algorithms and field trials that demonstrate its practical value in the case of AUV missions. Here, we present a spatiotemporal optimal stopping formulation of the mission monitoring problem that generalises the formulation in Best and Anstee (2014) to admit stochastic prediction models for the target and probabilistic communication models. The solution we present here has guaranteed optimality and polynomial runtime. Later in Chapter 6, we generalise the problem formulation for a team of trackers, and we leverage and extend the core mission monitoring algorithm presented in this chapter for decentralised multi-tracker planning.

5.1.2 Chapter outline

The remainder of this chapter is organised as follows. Section 5.2 formally defines the spatiotemporal optimal stopping formulation and mission monitoring objective as an active perception problem. Section 5.3 gives an overview of the algorithm, which is divided into two phases: Section 5.4 details the spatiotemporal graph generation phase, while Section 5.5 details the sweep-plane algorithm and analysis. Section 5.7 presents simulated experiments for analysing the behaviour of the algorithm and the probabilistic formulation. Section 5.8 presents two application case studies, with implementation details, extensive simulation results and empirical analysis. Finally, Section 5.9 summarises the chapter.
5.2 Problem formulation

In this section we formulate the mission monitoring problem. The problem involves two mobile agents: (1) a target agent which follows a probabilistic trajectory defined by a mission plan, and (2) a tracker agent that seeks to effectively monitor the target throughout the mission. To monitor effectively, the tracker must be within observation/communication range of the target and must be stationary. The trajectory of the tracker can therefore be characterised as a sequence of stopping waypoints in time and space. This scenario presents an optimisation problem with the target’s trajectory as the independent variable, while the tracker’s trajectory is optimised. In this section, we formally define the characteristics of the target and tracker trajectories, the general definition for probabilistic observation models, and the idea of effective monitoring as an optimisation objective.

5.2.1 Target trajectory (independent variable)

The trajectory of the target is described as its position as a function of time \( x(t) : [0, T] \rightarrow \mathcal{X} \), where \( \mathcal{X} \) is the space of all possible target locations. Time \( T \) is a planning horizon (in our experiments, we set \( T \) to the full duration of the mission). The mission is discretised into \( N \) time steps \( t_i := (i - 1)\Delta_t \in \mathcal{T} \), with \( t_1 = 0 \) and \( t_N = T \). The trajectory is not known precisely ahead of time, and therefore the predicted position of the target at time \( t_i \) is represented as a random variable \( X_i \) with a known distribution \( X_i \sim D_i \) and associated probability density function \( \rho_i(x) \). Therefore, the predicted trajectory of the target is described as the sequence of random variables \( X := (X_1, X_2, ..., X_N) \).

5.2.2 Tracker trajectory (dependent variable)

The trajectory of the tracker is described as its position as a function of time \( y(t) : [0, T] \rightarrow \mathcal{Y} \), where \( \mathcal{Y} \) is the space of all feasible positions of the tracker. The trajectory of the tracker is characterised as alternating between two states \( \{\text{STOPPED, MOVING}\} := \mathcal{S} \), which is described by a function of time \( s(t) : [0, T] \rightarrow \mathcal{S} \).
The functions \( y(t) \) and \( s(t) \) are sampled at time steps \( t_i \in \mathcal{T} \), resulting in the sequences of positions \( Y = (y_1, y_2, \ldots, y_N) \) and states \( S = (s_1, s_2, \ldots, s_N) \).

**Stationary waypoints**

The trajectory of the tracker is also described by the tuple \( U = [\hat{Y}, T^a, T^d] \), where \( \hat{Y} := (\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_M) \) is a sequence of waypoint positions with sequences of associated arrival times \( T^a := (t^a_1, t^a_2, \ldots, t^a_M) \) and departure times \( T^d := (t^d_1, t^d_2, \ldots, t^d_M) \).

During the time interval \([t^a_i, t^d_i)\), the tracker is in the STOPPED state and is stationary at the waypoint position \( \hat{y}_i \in \hat{Y} \subseteq Y \), where \( \hat{Y} \) is the set of positions where the tracker may stop. During the time interval \([t^d_i, t^a_{i+1})\), the tracker is in the MOVING state and is travelling between consecutive waypoints \( \hat{y}_i, \hat{y}_{i+1} \). The sequences of arrival and departure times satisfy the constraints: \( t^a_i \geq 0 \), \( t^a_M \leq T \), and \( t^a_i < t^d_i < t^a_{i+1} \), \( \forall i \).

The required travel time \( \delta(\hat{y}_i, \hat{y}_j) := t^a_j - t^d_i \) between two waypoints is defined by a function \( \delta(\hat{y}_i, \hat{y}_j) : \hat{Y} \times \hat{Y} \to \mathbb{R}_{\geq 0} \). The proposed algorithm does not depend on the exact trajectory taken to achieve this travel time. We require \( \delta(\hat{y}_i, \hat{y}_j) = 0 \) iff \( \hat{y}_i = \hat{y}_j \).

By this definition, the position as a function of time has

\[
y(t) = \hat{y}_i, \quad \forall t \in [t^a_i, t^d_i), \quad \forall i \in \{1, 2, \ldots, M\}, \tag{5.1}
\]

and the state as a function of time is

\[
s(t) = \begin{cases} 
    \text{STOPPED} & \text{if } t \in \bigcup_{i=1}^{M} [t^a_i, t^d_i) \\
    \text{MOVING} & \text{otherwise.} 
\end{cases} \tag{5.2}
\]

**Start and end conditions**

The start position \( \hat{y}_1 \) and end position \( \hat{y}_M \) of the tracker are assumed to be elements of given sets \( \hat{Y}_{\text{start}} \) and \( \hat{Y}_{\text{end}} \), respectively. These positions may be fixed, for example when the tracking vehicle is used for deploying/retrieving the target vehicle at fixed locations. Alternatively, if the sets are non-singleton then \( \hat{y}_1 \) and \( \hat{y}_M \) are to be optimised by the proposed planner.
5.2.3 Monitoring effectiveness (objective function)

The goal of the tracker is to effectively monitor the target. At time \( t_i \), the monitoring effectiveness is described by a function \( f(X_i, y_i, s_i) : \mathcal{X} \times \mathcal{Y} \times \mathcal{S} \rightarrow [0, 1] \), with output ranging from 0 (not monitoring) to 1 (effectively monitoring). This function is defined as

\[
f(X_i, y_i, s_i) := \begin{cases} 
\tilde{f}(\|X_i - y_i\|) & \text{if } s_i = \text{STOPPED} \\
0 & \text{if } s_i = \text{MOVING},
\end{cases}
\]  
(5.3)

where \( \tilde{f}(r_i) : \mathbb{R}_{\geq 0} \rightarrow [0, 1] \) is the observation model (interchangeable with communication model) and describes the observation value of monitoring the target from a distance of \( r_i \). Without loss of generality, the observation value is assumed to be scaled between 0 and 1. For example, \( \tilde{f}(r_i) \) may describe the probability of the tracker successfully communicating with or detecting the target, or the expected communication bandwidth. This function may be a simple binary \( r \)-disk model (Section 5.2.4) or a more realistic observation model (Section 5.8). For clarity, we define the observation model as translation-, orientation- and time-invariant, however the algorithm can readily be extended for more general models. (Section 5.8.2 demonstrates a translation-dependent model.)

The monitoring effectiveness objective function \( F(X, Y, S) \) is defined as the expected monitoring effectiveness over the duration of the mission:

\[
F(X, Y, S) := \mathbb{E} \left[ \Delta t \sum_{i=1}^{N} f(X_i, y_i, s_i) \right] = \Delta t \sum_{i=1}^{N} \mathbb{E} [f(X_i, y_i, s_i)],
\]  
(5.4)

which can be interpreted as the expected weighted sum of time that the tracker is STOPPED, weighted by the observation values. \( F(X, Y, S) \) can be evaluated using the expected values

\[
\mathbb{E} [f(X_i, y_i, \text{STOPPED})] = \mathbb{E} \left[ \tilde{f}(\|X_i - y_i\|) \right] = \int_{\mathcal{X}} \rho_i(x) \tilde{f}(\|x - y_i\|) dx,
\]  
(5.6)

\[
\mathbb{E} [f(X_i, y_i, \text{MOVING})] = 0.
\]
5.2 Problem formulation

For convenience, we also introduce notation for the monitoring effectiveness evaluated at the set of discrete timesteps $\subseteq \mathcal{T}$ that fall within a (continuous) time period of interest $\mathcal{T}$:

$$F_T := \Delta t \sum_{\eta \in \mathcal{N}} \mathbb{E}[f(X_{\eta}, y_{\eta}, s_{\eta})], \mathcal{N} = \{\eta : t_{\eta} \in \mathcal{T} \cap \mathcal{T}\}.$$ \hfill (5.7)

5.2.4 Deterministic problem instances

As a special case, we address deterministic scenarios where both: (1) the target’s trajectory $X$ is deterministic (i.e., each $\rho_i(x)$ is defined as a Dirac delta function), and (2) the monitoring effectiveness function $\tilde{f}(r_i)$ is defined as the binary $r$-disk model with monitoring range $r$, i.e.,

$$\tilde{f}(r_i) := \begin{cases} 1 & \text{if } r_i \leq r \\ 0 & \text{otherwise.} \end{cases}$$ \hfill (5.8)

In these deterministic problem instances, the expected value

$$\mathbb{E}[f(X_i, y_i, s_i)] = f(X_i, y_i, s_i)$$ \hfill (5.9)

and evaluates to 0 or 1 only.

For this special case, we also assume that the average speed of the tracker between waypoints is not less than the maximum instantaneous speed of the target $\|\dot{x}\|_{\text{max}}$, i.e.,

$$\frac{\|\hat{y}_j - \hat{y}_i\|}{\delta(\hat{y}_i, \hat{y}_j) \geq \|\dot{x}\|_{\text{max}}, \forall \hat{y}_i, \hat{y}_j.}$$ \hfill (5.10)

5.2.5 Problem statement

The optimisation problem to be solved is stated as follows.

**Problem 5.1** (Mission monitoring). For a given probabilistic model of the predicted target trajectory $X$, a set of possible waypoint locations $\hat{Y}$, and the start and/or end locations sets $\hat{Y}_{\text{start}}, \hat{Y}_{\text{end}}$, find the set of stopping waypoints $U$ with positions $\hat{Y}$, arrival times $T^a$ and departure times $T^d$, such that the travel time constraints
t^{a}_{i+1} - t^{d}_{i} = \delta (\hat{y}_{i}, \hat{y}_{i+1}), \forall r \in \{1, ..., R\} \text{ are satisfied, and the expected monitoring} \\
\text{effectiveness } F(X, Y, S), \text{ as defined in (5.2.3), is maximised over the mission duration.}

We address Problem 5.1 for the general case, as well as for the special deterministic 
\text{case defined in Section 5.2.4. The proposed algorithm has improved efficiency if} 
\text{certain reasonable assumptions hold, which are defined later in Section 5.4.1.}

5.3 Algorithm overview

The proposed algorithm is divided into a graph generation phase and then a longest-
\text{path graph search using a sweep-plane. Pseudocode is listed in Algorithm 5.1.} 

The first phase generates a search graph such that paths through this graph describe 
\text{feasible solution trajectories for the tracker. The generated graph is directed acyclic,} 
\text{which enables the optimal solution to be found efficiently. The graph generation} 
\text{exploits geometric properties of the problem so that, under certain conditions, the} 
\text{efficiency of the algorithm is improved and optimality is maintained. This graph} 
\text{generation procedure is presented in Section 5.4.} 

In the second phase, the optimal solution is found by performing a longest-path search 
\text{through the spatiotemporal graph. We describe this algorithm geometrically as a} 
\text{sweep-plane moving through time. For general graphs, a longest-path search is NP-} 
\text{hard. However, since the generated graph is directed acyclic, the optimal trajectory} 
\text{is found in polynomial time. This sweep-plane algorithm is presented in Section 5.5.} 

---

**Algorithm 5.1** Overview of the trajectory planner for the tracker.

1: function Main(X)
2: $[\mathcal{V}, \mathcal{E}, \{t^d_i\}, \{t^a_i\}, \{\omega\}] \leftarrow \text{GenerateGraph}(X)$
3: $[\mathcal{V}, \mathcal{E}, \mathcal{V}_{\text{start}}, \mathcal{V}_{\text{end}}] \leftarrow \text{StartEndCond’s}(\mathcal{V}, \mathcal{E}, X)$
4: $([\Omega], \{\psi\}) \leftarrow \text{SweepPlane}(\mathcal{V}, \mathcal{E}, \{\omega\}, \{t^a_i\})$
5: $[U, F] \leftarrow \text{BackTrack}([\Omega], \{\psi\}, \mathcal{V}, \mathcal{E}, \mathcal{V}_{\text{start}}, \mathcal{V}_{\text{end}})$
6: return $[U = [\hat{Y}, T^a, T^d], F]$
Algorithm 5.2 Generate a spatiotemporal search graph of potential waypoint positions and times.

1: function GenerateGraph($X$)
2: Select potential stopping locations $p_i \in P$
3: Generate vertices $v_\eta = \{p_\eta, r_\eta^a, r_\eta^d\} \in V$
4: Find feasible edges $e_k = \langle v_i, v_j \rangle \in E$
5: Calculate edge times $t_{d_i}^i, t_{a_j}^j$ for each edge $e_k$
6: Calculate edge weight $\omega_{i,j}$ for each edge $e_k$
7: return $[V, E, \{t_{d_i}^i\}, \{t_{a_j}^j\}, \{\omega_{i,j}\}]$

5.4 Spatiotemporal search graph

This section describes the process of generating the spatiotemporal search graph, as summarised in Algorithm 5.2. Section 5.4.1 describes how to generate the graph vertices, where each vertex represents a time interval at a position. Section 5.4.2 describes how to generate edges, such that each edge represents the travel time and monitoring effectiveness for travelling between a pair of vertices. Finally, Section 5.4.3 discusses further adjustments to the graph so that the start and end conditions are met.

5.4.1 Vertices

A set of graph vertices is generated, with each vertex $v_i \in V$ representing a potential stopping location in time and space. This is achieved by selecting a discrete set of positions in space in the neighbourhood of the target’s path. Time is incorporated for each position by considering all times that the tracker is expected to be effectively monitoring the target. We first introduce the spatial dimensions in Section 5.4.1, followed by the temporal dimensions in Section 5.4.1.

Spatial dimensions

The set of positions $P \subseteq \hat{Y}$ is generated by first discretising the space, such that $P = \hat{Y} \cap P_1$, where $P_1$ is a discrete set of positions. The algorithm is optimal with respect to the discretisation used for $P_1$. The best choice for discretisation is problem specific; a uniform grid is used in all figures and most of the experiments, while
Section 5.8.1 uses an adaptive-resolution grid and Section 5.8.2 uses a probabilistic roadmap (PRM).

The search space can be further reduced by taking the intersection with two additional sets, such that \( \mathcal{P} = \hat{Y} \cap \mathcal{P}_1 \cap \mathcal{P}_2 \cap \mathcal{P}_3 \). The set \( \mathcal{P}_2 \) describes the monitoring region and \( \mathcal{P}_3 \) describes the convex hull of the mission. These sets are formally defined below and an example of these sets is illustrated in Figure 5.2. Under reasonable conditions (defined below), optimality is still guaranteed after performing this culling. If a particular condition does not hold for a specific problem instance, then the associated set may be omitted to guarantee optimality. This additional culling of the search domain is not essential; the algorithm has the same runtime complexity, but the culling will improve the efficiency of the algorithm in practice. In our several experiment formulations, discussed later in this chapter, we present several realistic models that satisfy these conditions.

The set \( \mathcal{P}_2 \) is the set of all points \( p_i \) that have a non-zero monitoring effectiveness (i.e., are within monitoring range) for part of the target’s trajectory, i.e.,

\[
\exists X_\eta \in X : \mathbb{E} \left[ \tilde{f}(\|X_\eta - p_i\|) \right] > 0. \quad (5.11)
\]

Lemma 5.1 shows optimality is maintained if the \( \mathcal{P}_2 \) culling is used, assuming Condition 5.1 holds. For \( \mathcal{P}_3 \), first define \( \text{CH} \) as the convex hull of the set of all possible locations visited by the target and the tracker start and end sets; i.e., the convex hull of the set

\[
\{ x : \exists i, \rho_i(x) > 0 \} \cup \hat{Y}_{\text{start}} \cup \hat{Y}_{\text{end}}. \quad (5.12)
\]
The set $\mathcal{P}_3$ is all points that are in CH or are a distance less than the $\mathcal{P}_1$ discretisation spacing away from the boundary of CH. Lemma 5.2 shows optimality is maintained if the $\mathcal{P}_3$ culling is used, assuming Conditions 5.2, 5.3 and 5.4 hold.

**Condition 5.1** (Triangle inequality). The travel times satisfy the triangle inequality, i.e., $\delta(\hat{y}_a, \hat{y}_b) \leq \delta(\hat{y}_a, \hat{y}_i) + \delta(\hat{y}_i, \hat{y}_b)$.

**Condition 5.2** (Convex hull is feasible). All positions in CH are feasible stopping locations, i.e., $\text{CH} \subseteq \hat{Y}$.

**Condition 5.3** (Monotonically decreasing observation value). The observation model $\hat{f}(r_i)$ is a monotonically decreasing function of distance.

**Condition 5.4** (Monotonically increasing travel time). The travel time monotonically increases with distance, for a fixed start or end position; i.e., if $\|\hat{y}_i - \hat{y}_a\| \geq \|\hat{y}_i - \hat{y}_b\|$ then $\delta(\hat{y}_i, \hat{y}_a) \geq \delta(\hat{y}_i, \hat{y}_b)$ and $\delta(\hat{y}_a, \hat{y}_i) \geq \delta(\hat{y}_b, \hat{y}_i)$.

**Remark 5.1** (Unbounded distributions). If the distributions $\rho_i(x)$ or the observation function $\hat{f}(r_i)$ have an unbounded support, then $\mathcal{P}$ is potentially an infinite set. However, this is not an issue since the reachability pruning (later in Section 5.4.3) ensures the search space is finite. For computational reasons, to further reduce the size of the graph it may be appropriate to approximate $\mathcal{P}_2$ and $\mathcal{P}_3$ using non-zero lower bounds, i.e., $\mathbb{E}\left[\hat{f}(\|X_\eta - p_i\|)\right] > LB_1$ for $\mathcal{P}_2$ and $\rho_i(x) > LB_2$ for $\mathcal{P}_3$. $\triangle$

**Lemma 5.1** (Stopping in effective monitoring region). If the travel times satisfy the triangle inequality (Condition 5.1), then an optimal solution trajectory $U$ only contains waypoints at locations $\hat{y}_i \in \hat{Y}$ which satisfy $\exists X_\eta \in X : \mathbb{E}\left[\hat{f}(\|X_\eta - \hat{y}_i\|)\right] > 0$.

**Proof.** Define two partial solution trajectories over the time interval $[t_i^a, t_i^b]$: (1) $\hat{Y} = (\hat{y}_i, \hat{y}_j, \hat{y}_k)$ and (2) $\hat{Y}^* = (\hat{y}_i, \hat{y}_k)$; with $\hat{y}_j$ satisfying $\mathbb{E}[\hat{f}(\|X_\eta - \hat{y}_j\|)] > 0$. It follows from this definition of $\hat{y}_j$ that the monitoring effectiveness $F$ while STOPPED at $\hat{y}_j$ is $F_{[t_i^a, t_i^b]} = 0$. Combined with the MOVING time intervals, $F_{[t_i^a, t_i^b]} = 0$, with interval length $L = t_k^a - t_i^a$. Therefore $F(\hat{Y}) = F_{[t_i^a, t_i^b]}(t_k^a, t_k^b)$. For $\hat{Y}^*$, the monitoring effectiveness while MOVING is $F_{[t_i^a, t_i^b]} = 0$, with interval length $L^* = t_k^a - t_i^a$. Therefore $F(\hat{Y}^*) = F_{[t_i^a, t_i^b]}(t_k^a, t_k^b)$. Since the triangle inequality holds (Condition 5.1), it follows that $L \geq L^*$. Therefore $\exists \{t_i^a, t_i^b\} : (t_i^a \geq t_i^b)$, where $\{t_i^a, t_i^b\} = \{t_i^a, t_i^b\}$ is
a feasible choice for departure and arrival times. The optimal choice for \(\{t^d_i, t^a_k\}\) will always result in a greater or equal monitoring effectiveness than if \(\{t^d_i, t^a_k\} = \{t^d_i', t^a_k'\}\) were chosen, therefore:

\[
F(\hat{Y}^\ast) \geq F[t^d_i, t^a_k] \cup [t^d_i', t^a_k']
\]

\[
= F(\hat{Y}) + F[t^d_i', t^a_k] \cup [t^d_i, t^a_k']
\]

\[
\geq F(\hat{Y}).
\]

It follows that \(F(\hat{Y})\) will never decrease if \(\hat{y}_j\) was removed from the sequence. This generalises to longer sequences since \(F\) is additive over partial sequences; therefore an optimal sequence exists with all \(\hat{y}_i\) in range of an \(X_\eta\).

**Lemma 5.2** (Stopping in convex hull). If \(\text{CH} \subseteq \hat{Y}\) (Condition 5.2), \(\tilde{f}\) is monotonically decreasing (Condition 5.3), and the travel time monotonically increases with distance (Condition 5.4), then an optimal solution trajectory \(U\) only contains waypoints at locations \(\hat{y}_i \in \hat{Y}\) which are in \(\text{CH}\).

**Proof.** Define a stopping position \(\hat{y}_a \notin \text{CH}\). By definition, there exists a half-plane \(\mathcal{H}\) such that \(\text{CH} \subset \mathcal{H}\), \(\hat{y}_a \notin \mathcal{H}\), and \(\hat{y}_a^\ast\) lies on the boundary of \(\mathcal{H}\) where \(\hat{y}_a^\ast\) is the closest point to \(\hat{y}_a\) in \(\text{CH}\). The line segment \(\hat{y}_a\) to \(\hat{y}_a^\ast\) is perpendicular to the boundary of \(\mathcal{H}\); therefore \(\hat{y}_a^\ast\) is closer than \(\hat{y}_a\) to any point in \(\mathcal{H}\), i.e.,

\[
\|\hat{y}_a^\ast - h\| < \|\hat{y}_a - h\|, \forall h \in \mathcal{H}. \tag{5.13}
\]

Therefore, since \(X \subset \mathcal{H}\) and \(\tilde{f}\) is monotonically decreasing (Condition 5.3):

\[
\tilde{f}(\|X_i - \hat{y}_a^\ast\|) \geq \tilde{f}(\|X_i - \hat{y}_a\|), \forall X_i \in X. \tag{5.14}
\]

It follows that the monitoring effectiveness of a solution that contains a waypoint at \(\hat{y}_a^\ast\) will never decrease if this waypoint were moved to \(\hat{y}_a\) instead. It is assumed that \(\hat{y}_a^\ast \in \hat{Y}\), which will hold if Condition 5.2 holds.

To be optimal, selecting \(\hat{y}_a^\ast\) instead of \(\hat{y}_a\) must also not result in a lower monitoring effectiveness at the previous and next waypoints in the sequence. Define the partial
solutions $\hat{Y} = (\hat{y}_i, \hat{y}_a, \hat{y}_j)$ and $\hat{Y}^* = (\hat{y}_i, \hat{y}_a^*, \hat{y}_j)$, where $\hat{y}_i, \hat{y}_j \in CH \subset H$. It follows from (5.13) and Condition 5.4 that the travel times will not increase by selecting $\hat{y}_a^*$ instead of $\hat{y}_a$, i.e.,

$$\delta(\hat{y}_i, \hat{y}_a^*) \leq \delta(\hat{y}_i, \hat{y}_a) \quad \text{and} \quad \delta(\hat{y}_a^*, \hat{y}_j) \leq \delta(\hat{y}_a, \hat{y}_j). \quad (5.15)$$

Therefore the departure from $\hat{y}_i$ need not be earlier and the arrival to $\hat{y}_j$ need not be later if $\hat{y}_a^*$ is chosen instead of $\hat{y}_a$; hence the monitoring effectiveness at $\hat{y}_i$ and $\hat{y}_j$ will not decrease if $\hat{y}_a^*$ is chosen instead of $\hat{y}_a$.

It follows that $F(\hat{Y}^*) \geq F(\hat{Y})$. Given that $\hat{y}_1, \hat{y}_M \in CH$, this generalises to longer sequences. Therefore an optimal solution trajectory has all $\hat{y}_i \in CH$. \hfill \Box

**Temporal dimensions**

Each vertex $v_\eta \in V$ represents a position $p_\eta \in P$ and a time interval $[\tau_{a\eta}, \tau_{d\eta}] \subseteq T$, denoted by the tuple $v_\eta := [p_\eta, \tau_{a\eta}, \tau_{d\eta}]$. We first describe how to select $\tau_{a\eta}, \tau_{d\eta}$ for the general problem, then describe a procedure that improves the efficiency for deterministic cases. Figure 5.3 shows an example set of generated vertices for (a) probabilistic and (b) deterministic problems.

For probabilistic problem instances, each vertex has a time interval length equal to the time discretisation, i.e., $\tau_{d\eta} - \tau_{a\eta} = \Delta_t$. For each position $p_\eta \in P$, a vertex is created for each time step $\tau_{a\eta} \in T$ where the target has a non-zero probability of being in range of the tracker, i.e.,

$$v_\eta = [p_\eta, \tau_{a\eta}, \tau_{d\eta} = \tau_{a\eta} + \Delta_t] \in V \text{ iff } \mathbb{E} \left[ f(\|X(\tau_{a\eta}) - p_\eta\|) \right] > 0. \quad (5.16)$$

This definition is referred to as the *probabilistic algorithm*. An example of this vertex generation is illustrated in Figure 5.3a overlaying a probabilistic target trajectory represented by a set of sample trajectories. Each vertical blue line segment is a vertex with the bottom at time $\tau_{a\eta}$ and the top at $\tau_{d\eta}$. In this example, the extreme samples are used to approximate the boundaries of the non-zero regions of the distribution.

For deterministic problem instances (defined in Section 5.2.4), only a single vertex needs to be created for each contiguous subsequence of times where the target is in
range of the tracker. More formally, \( T_i \subseteq \mathcal{T} \) denotes the set of all times where the target would be effectively monitored if the tracker were STOPPED at \( p_i \) at time \( t_i \), i.e.,

\[
T_i := \{ t_i \in \mathcal{T} : \tilde{f}(\|X_l - p_i\|) = 1 \}. \tag{5.17}
\]

Each \( T_i \) is then divided into non-overlapping subsequences, with each subsequence being a maximal run of consecutive timesteps \((t_j, t_{j+1}, \ldots, t_{j+k}) \subseteq T_i\). Each subsequence forms a new vertex \( v_\eta \) in the search graph with \( \tau^a_\eta \) and \( \tau^d_\eta \) chosen as the subsequence start and end times, i.e.,

\[
[p_\eta, \tau^a_\eta, \tau^d_\eta] = [p_l, t_j, t_{j+k} + \Delta_t]. \tag{5.18}
\]

This definition is referred to as the deterministic algorithm. Figure 5.3b illustrates that, in contrast to the probabilistic algorithm, each vertex can span multiple timesteps.

The deterministic algorithm is typically more efficient since fewer vertices are generated. This adjustment maintains optimality since, for the deterministic case, if an
optimal solution path contains the position $p_i$ at time $t_i$ then it is optimal to stay at position $p_i$ for all timesteps consecutive to $t_i$ when the target is still in range. A proof of this guarantee is provided later in Lemma 5.3 after the edges have been introduced.

5.4.2 Edges

A solution trajectory is represented by a path through the graph with consecutive vertices connected by directed edges $e_\eta \in \mathcal{E}$. An edge is denoted $e_\eta = (v_i, v_j)$ and describes travelling from vertex $v_i$ at position $p_i$ to vertex $v_j$ at position $p_j$ at some time in the solution trajectory.

Each edge has an associated departure time $t_{e_\eta}^d := t_i^d$ and arrival time $t_{e_\eta}^a := t_j^a$ which describes the exact time the tracker moves from $p_i$ to $p_j$. We require $t_{e_\eta}^a, t_{e_\eta}^d$ satisfy

$$t_{e_\eta}^a = \tau_{e_\eta}^a < t_{e_\eta}^d.$$  \hspace{1cm} (5.19)

The key advantage of having a fixed arrival time $t_{e_\eta}^a$ (5.19) for a vertex is that the calculations for an edge $e_\eta = (v_i, v_j)$ do not depend on the choice of arrival time for a previous edge $e_m = (v_h, v_i)$ or the path taken to or from an edge; therefore optimal sub-paths are additive and generally lead to globally optimal solutions. For the probabilistic algorithm, selecting $t_{e_\eta}^a = \tau_{e_\eta}^a$ is optimal relative to the temporal resolution since each vertex represents only stopping for a single time step at $p_\eta$. For the deterministic case, where a vertex represents a contiguous subsequence of in-range timesteps, this choice is still optimal (shown later in Lemma 5.3).

Each edge also has an associated weight $\omega_{i,j}$ which is defined as the monitoring effectiveness over the time interval $[\tau_i^a, \tau_j^a)$ if that edge were chosen, i.e.,

$$\omega_{i,j} := F_{[\tau_i^a, \tau_j^a)}.$$  \hspace{1cm} (5.20)

Each edge is in one of four categories, which determines the edge weight and moving times. The conditions are derived directly from the geometric properties illustrated in Figure 5.4. The calculations are listed in Algorithm 5.3 and described as follows.

1. **Infeasible** — An edge is included if and only if the vertex $v_j$ is reachable from
Figure 5.4 – Illustration of the edge categories described in Algorithm 5.3 and Section 5.4.2. From left to right: infeasible, same position, smaller gap and larger gap.

Algorithm 5.3 Edge weight and time calculations for the four categories illustrated in Figure 5.4 and described in Section 5.4.2.

1: function $\text{EdgeCalculation}(e_{\eta} = (v_i, v_j))$
2: \[
\rho = E \left[ f(\|X_\eta - p_i\|) \right] : t_\eta = \tau_i^a
\]
3: \[
t_j^a \leftarrow \tau_j^a
\]
4: if $\delta(p_i, p_j) \geq \tau_j^a - \tau_i^a$ then $\triangleright$ infeasible
5: \[
\text{Do not include } e_{\eta} \text{ in } \mathcal{E}
\]
6: else if $p_i = p_j$ then $\triangleright$ same position
7: \[
t_i^d \leftarrow \tau_j^a
\]
8: \[
\omega_{i,j} \leftarrow \rho \times \left( \tau_i^d - \tau_i^a \right)
\]
9: else if $\delta(p_i, p_j) \geq \tau_j^a - \tau_i^d$ then $\triangleright$ smaller gap
10: \[
t_i^d \leftarrow \tau_j^a - \delta(p_i, p_j)
\]
11: \[
\omega_{i,j} \leftarrow \rho \times \left( \tau_j^a - \tau_i^a - \delta(p_i, p_j) \right)
\]
12: else $\triangleright$ larger gap
13: \[
t_i^d \leftarrow \tau_j^a - \delta(p_i, p_j)
\]
14: \[
\omega_{i,j} \leftarrow \rho \times \left( \tau_i^d - \tau_i^a \right)
\]
15: return $[t_i^d, t_j^a, \omega_{i,j}]$ $\triangleright$ depart, arrive, weight

$v_i$, i.e., $\delta(p_i, p_j) \leq \tau_j^a - \tau_i^a$.

2. **Same Position** – The two vertices are at the same position and therefore merged into a single waypoint.

3. **Smaller Gap** – The gap between the vertices is smaller than $\delta(p_i, p_j)$; therefore there will be no time spent in the STOPPED state while not effectively monitoring.

4. **Larger Gap** – The gap is larger than $\delta(p_i, p_j)$; therefore there must be some time spent in the STOPPED state while not effectively monitoring.
Remark 5.2 (Underestimates due to pass-through vertices). The trajectory represented by an edge \( \langle v_i, v_j \rangle \) may implicitly pass through another vertex \( v_\eta \) where \( v_\eta = [p_\eta, \tau^a_\eta, \tau^d_\eta] \). This occurs in two scenarios: (1) when there exists a \( v_\eta \) where \( p_\eta = p_i \) and the time interval \([\tau^a_\eta, \tau^d_\eta]\) overlaps with \([t^a_i, t^d_i]\), or (2) when there exists a \( v_\eta \) where \( p_\eta = p_j \) and the time interval \([\tau^a_\eta, \tau^d_\eta]\) overlaps with \([t^a_j, t^d_j]\). For efficient computation, and without loss of optimality, the computations in Algorithm 5.3 ignore the value of the pass-through vertex \( v_\eta \), if one exists. Therefore, \( \omega_{i,j} \) may underestimate \( F[\tau^a_i, \tau^a_j] \). However, this case will also be realised by an alternate path that visits the pass-through vertex \( v_\eta \). This alternate path, \( \langle \langle v_i, v_\eta \rangle, \langle v_\eta, v_j \rangle \rangle \) will have a value estimate consisting of \( \omega_{i,\eta} \) and \( \omega_{\eta,j} \), and therefore will provide a correct estimate of (5.20). The underestimate of \( \omega_{i,j} \) in the pass-through case does not result in suboptimal solutions since a maximum-weight search algorithm will always choose the alternate path \( \langle \langle v_i, v_\eta \rangle, \langle v_\eta, v_j \rangle \rangle \) instead. △

Lemma 5.3 (Optimal arrival time for deterministic cases). For deterministic problem instances (defined in Section 5.2.4), if a path passes through \( v_\eta \), then it is optimal for the solution trajectory to arrive at \( p_\eta \) with \( t^a_\eta \) chosen as \( \tau^a_\eta \) (using the definitions for \( t^a_\eta \) and \( \tau^a_\eta \) from Sections 5.4.1 and 5.4.2).

**Proof.** Consider the path consisting of a feasible edge \( \langle v_i, v_j \rangle \), where \( p_i \neq p_j \), for three cases: A - choose \( t^a_j = t^a_{ja} \) where \( \tau^a_j < t^a_{ja} \leq \tau^d_j \); B - choose \( t^a_j = t^a_{jb} \) where \( t^a_{jb} = \tau^a_j \); and C - choose \( t^a_j = t^a_{jc} \) where \( t^a_{jc} < \tau^a_j \). By this definition,

\[
t^a_{ja} > t^a_{jb} = \tau^a_j > t^a_{jc}.
\] (5.21)

The following proof shows that B has a monitoring effectiveness greater than or equal to A and C.

Firstly, we show that B is a feasible choice; i.e., it does not require departing \( p_i \) before the start time \( \tau^a_i \). Consider the pair of start times \( (\tau^a_i, \tau^a_j) \). When the target moves in a straight line at maximum speed \( \|\dot{x}\|_{\text{max}} \) (i.e., gradient in Figure 5.3b), the vertices will have start times with this same gradient between pairs, i.e.,

\[
|\tau^a_j - \tau^a_i| = \frac{\|p_j - p_i\|}{\|\dot{x}\|_{\text{max}}},
\] (5.22)
5.4 Spatiotemporal search graph

If the target turns (e.g. upper half of Figure 5.3b), or moves slower, this time difference must be larger; therefore generally

$$|\tau^a_j - \tau^a_i| \geq \frac{||p_j - p_i||}{\|\dot{x}\|_{\text{max}}}.$$  \hspace{1cm} (5.23)

Applying the speed assumption yields

$$|\tau^a_j - \tau^a_i| \geq |t^a_j - t^d_i| = \delta(p_i, p_j).$$  \hspace{1cm} (5.24)

An exception could occur at the beginning of the mission (since $|\tau^a_j - \tau^a_i| = 0$ if $\tau^a_j = \tau^a_i = 0$); however the vertex adjustments described later in Section 5.4.3 ensure this will not prevent an optimal path from being chosen. From (5.24), it follows that if $t^d_j = t^a_j$ then $t^d_i \geq \tau^a_i$, and therefore $B$ is a feasible choice.

For $A$, the tracker departs $p_i$ at a time $\partial := t^a_{ja} - t^a_{jb}$ later than for $B$. Therefore $B$ will spend $\partial$ less time at $p_i$ and $\partial$ more time at $p_j$ than $A$. In $B$, the extra time spent at $p_j$ is the interval $[t^a_{ja}, t^a_{ja}])$. By definition of a vertex for the deterministic case (Section 5.2.4), $F((t^a_{ja}, t^a_{ja}]) = \partial$, which is maximal. $A$ can not improve on this during the extra time at $p_i$, and therefore $B$ has a greater or equal monitoring effectiveness than $A$. Note this assumes $\tau^a_i < \tau^a_j$; however, it follows from (5.24) and the triangle inequality assumption from Lemma 5.1 that an optimal path will not contain $\langle v_i, v_j \rangle$ if $\tau^a_i \geq \tau^a_j$, since $v_j$ would also be reachable from the vertex preceding $v_i$.

To achieve $C$, the tracker will spend more time at $p_j$ than for $B$. This extra time is before $\tau^a_j$, and therefore by definition of a vertex, $F([t^a_{ja}, \tau^a_{ja}]) = 0$, which is minimal; hence $B$ has a greater or equal monitoring effectiveness than $C$. This shows that $t^a_j = \tau^a_j$ is optimal. Note that this assumes that there is no vertex $v_k$ where $p_k = p_j$ and $\tau^a_k < \tau^a_j$; however, if a $v_k$ exists then the planner has the option to choose the subsequence $(v_i, v_k, v_j)$ if this is feasible. For $(v_i, v_k, v_j)$, this proof also shows that $t^a_k = \tau^a_k$ is the optimal arrival time for $v_k$, and $t^a_j$ is irrelevant since $p_j = p_k$. \hfill \Box

### 5.4.3 Start and end conditions

The graph needs to be adjusted to ensure that the solution path satisfies the start and end constraints. This includes trimming or removing some of the edges, as well
Algorithm 5.4 Vertex set adjustments for the start condition.

1: for each $v_i \in V \setminus V_{\text{start}}$ do
2:   if $\exists p_{\text{start}} \in P_{\text{start}} : \tau_i^a \geq \delta(p_{\text{start}}, p_i)$ then
3:      continue \quad \triangleright \text{keep } v_i$
4:   else if $\forall p_{\text{start}} \in P_{\text{start}} : \tau_i^d \leq \delta(p_{\text{start}}, p_i)$ then
5:      $V \leftarrow V \setminus v_i$ \quad \triangleright \text{remove } v_i$
6:   else
7:      $\tau_i^a \leftarrow \min_{p_{\text{start}} \in P_{\text{start}}} \delta(p_{\text{start}}, p_i) + t_1$ \quad \triangleright \text{trim } v_i$

as defining the sets of start $v_{\text{start}} \in V_{\text{start}}$ and end $v_{\text{end}} \in V_{\text{end}}$ vertices.

The set of allowable start positions is calculated as $P_{\text{start}} = \hat{Y}_{\text{start}} \cap P$. If a problem instance requires start positions that are not already in $P$, then these positions should be added to $P$ and corresponding vertices and edges added to the graph. For each $p_{\text{start}} \in P_{\text{start}}$, if it is not in-range at time $t_1$ (i.e. $E[\hat{f}(\|X(t_1) - p_{\text{start}}\|)] = 0$), then additional vertices are generated between time $t_1$ and the first timestep where $p_{\text{start}}$ is in-range. For each $p_{\text{start}} \in P_{\text{start}}$, the vertex with $\tau_i^a = t_1$ is included in the set of possible start vertices $v_{\text{start}} \in V_{\text{start}}$.

To ensure the search always selects a path that begins at $v_{\text{start}} \in V_{\text{start}}$, all other vertices $v_i$ are adjusted using the rules described in Algorithm 5.4. In the first case, $v_i$ is reachable from at least one $p_{\text{start}} \in P_{\text{start}}$ and therefore no adjustment is made. The second case removes unreachable vertices. The third case trims all $v_i$ that are reachable only at some time after $\tau_i^a$. The third case is necessary for the deterministic algorithm where each vertex may span multiple timesteps, in order to ensure the Lemma 5.3 result holds. For simplicity, the reachability calculations assume the triangle inequality holds for travel times (Condition 5.1), although the algorithm could readily be adapted for other cases.

Similarly, the set of allowable end positions is calculated as $P_{\text{end}} = \hat{Y}_{\text{end}} \cap P$. For each position $p_{\text{end}} \in P_{\text{end}}$, the vertex with the latest departure time $\tau_i^d$ is added to the set of possible end vertices $v_{\text{end}} \in V_{\text{end}}$. The optimal start and end positions (for non-singleton $P_{\text{start}}$ or $P_{\text{end}}$) are found by the algorithm in the following section.
5.5 Sweep-plane algorithm

The optimal tracker trajectory is found by searching for the longest-path through the spatiotemporal graph. For general graphs, a longest-path search is NP-hard. However, optimal polynomial-time algorithms exist if the graph is a directed acyclic graph (DAG), since a topological ordering\(^1\) of \(\mathcal{V}\) exists. The spatiotemporal graph generation defined in Section 5.4 reduces the problem to a longest-path search through a DAG, and therefore, the solution can be found in \(\mathcal{O}(|\mathcal{V}| + |\mathcal{E}|)\) time. This section describes a longest-path search, which can be visualised as a sweep-plane moving through time. The algorithm includes finding the optimal \(v_{\text{start}} \in \mathcal{V}_{\text{start}}\) and \(v_{\text{end}} \in \mathcal{V}_{\text{end}}\).

5.5.1 Forward pass

The longest-path search begins with a forward pass through the graph that visits the nodes in topological order. A topological ordering of the vertices can be found by visiting \(v_i\) in order of ascending time \(t = \tau_i\). This can be thought of as a sweeping plane as illustrated in Figure 5.5 and described in Algorithm 5.5. The sweep-plane

---

\(^1\)If there exists a path from vertex \(v_i\) to \(v_j\), then \(v_i\) precedes \(v_j\) in a topological sort.
Algorithm 5.5 Sweep-plane graph search: forward pass.

1: function \text{SweepPlane}(\mathcal{V}, \mathcal{E}, \{\omega\}, \{t^n\})
2: \begin{align*}
\Omega_{\text{start}} & \leftarrow 0, \quad \forall v_{\text{start}} \in \mathcal{V}_{\text{start}} \\
\text{for} \ t = t_1, t_2, \ldots, t_N \ \text{do} \\
\quad & \begin{align*}
\text{for each} \ v_i \in \mathcal{V} \setminus \mathcal{V}_{\text{start}} \text{ where} \ t^a_i = t & \text{ do} \\
\quad & \begin{align*}
\mathcal{E}_i & \leftarrow \{e : (v_{\text{start}}, v_i) \in \mathcal{E}\} \quad \triangleright \text{edges into} \ v_i \\
\psi_i & \leftarrow \arg\max_{e \in \mathcal{E}_i} [\Omega_e + \omega_{e,i}] \quad \triangleright \text{optimal edge} \\
\Omega_i & \leftarrow \Omega_{\psi_i} + \omega_{\psi_i,i} \quad \triangleright \text{path weight}
\end{align*}
\end{align*}
\end{align*}
3: \text{return} \ [(\{\Omega\}), \{\psi\}] \quad \triangleright \text{path weights, back-pointers}

Algorithm 5.6 Backtracking to find the optimal trajectory.

1: function \text{BackTrack}(\{\Omega\}, \{\psi\}, \mathcal{V}, \mathcal{E}, \mathcal{V}_{\text{start}}, \mathcal{V}_{\text{end}})
2: \begin{align*}
\rho_i & \leftarrow \mathbb{E} \left[ \hat{f}(||X_\eta - p_i||) : t_\eta = \tau^a_{\text{end}}, \quad \forall v_i \in \mathcal{V}_{\text{end}} \right] \\
v_\eta & \leftarrow \arg\max_{v_i \in \mathcal{V}_{\text{end}}} [\Omega_i + \rho_i \times (\tau^d_i - \tau^a_i)] \quad \triangleright \text{end vertex}
\end{align*}
3: \text{return} \ [(\{\hat{y}\}), \{t^a\}, \{t^d\}, F] \quad \triangleright \text{trajectory, path weight}

represents a plane covering \(\mathcal{P}\) at a particular time \(t\), and moves linearly through increasing time \(T\) (line 3). A vertex \(v_i\) is explored once the sweep-plane reaches \(t = \tau^a_i\) (line 4). For efficient evaluation of the vertex set in line 4, \(\mathcal{V}\) should be pre-sorted by ascending \(\tau^a_i\). When \(v_i\) is explored (line 5), all edges \(e_{\psi_i}\) leading in to \(v_i\) are compared (line 6) and the optimal previous vertex with an edge into each \(v_i\) is denoted \(\psi_i\). The sum of weights along the optimal path leading to vertex \(v_i\) through edge \(e_{\psi_i}\) is calculated recursively and denoted \(\Omega_i\) (line 7).

5.5.2 Backtracking

Lastly, the optimal solution path is found by backtracking from a \(v_{\text{end}} \in \mathcal{V}_{\text{end}}\) to a \(v_{\text{start}} \in \mathcal{V}_{\text{start}}\), as described in Algorithm 5.6. The end vertex is chosen as the
vertex in $V_{\text{end}}$ with the highest path weight (line 3). The algorithm proceeds by recursively following the back-pointers $\psi$ until a $v_{\text{start}} \in V_{\text{start}}$ is reached (lines 5-12). Backtracking will always lead to a $v_{\text{start}} \in V_{\text{start}}$ due to the adjustments in Section 5.4.3. The expected monitoring effectiveness is $F = \Omega_{\text{end}} + \rho \times (\tau_{\text{end}}^d - \tau_{\text{end}}^a)$, since the weight of the end vertex is not accounted for by the edge weights.

5.6 Analysis

In this section, we analyse the optimality and time complexity of the proposed algorithm, and remark on practical considerations.

5.6.1 Optimality

The proposed algorithm is optimal with respect to the discretisation. We formally analyse the optimality as follows.

**Lemma 5.4** (Optimality of graph search algorithm). The graph search algorithm described by the sweep-plane forward pass in Algorithm 5.5 and backward pass in Algorithm 5.6 finds the optimal solution through a spatiotemporal graph $(V, E)$.

**Proof.** For each $v_i$, the forward pass calculates the preceding vertex $\psi_i$ and the sum of edge weights $\Omega_i$ for the optimal path from any $v_{\text{start}} \in V_{\text{start}}$ to $v_i$, if the mission were to end at time $\tau_{i}^a$. The algorithm recursively solves optimal sub-problems by iterating through $v_i \in V$ in a topological order. The sub-problems are optimal since the objective function $F$ is additive. The backtracking phase selects the optimal $v_{\text{end}} \in V_{\text{end}}$ and the backpointers $\psi_i$ directly map to the optimal path from $v_{\text{start}}$ to $v_{\text{end}}$. \hfill \Box

The main optimality result follows directly from Lemma 5.4 and the lemmas regarding the graph generation in Section 5.4.

**Theorem 5.1** (Optimality of spatiotemporal optimal stopping). Algorithm 5.1 finds the optimal tracker trajectory (as defined in Problem 5.1) with respect to the temporal ($\mathcal{T}$) and spatial ($\mathcal{P}_1$) discretisation.
Proof. The space of all possible tracker trajectories is first discretised over time and space. Lemma 5.1 and Lemma 5.2 (and Lemma 5.3 for the deterministic case) collectively state that the subsequent pruning phases of the graph generation algorithm (Algorithm 5.2) do not prune the optimal solution trajectory from the search graph \((V, E)\).

Lemma 5.4 states that the graph search algorithm finds the optimal solution for a given graph \((V, E)\) and thus Algorithm 5.1 finds the optimal solution to Problem 5.1.

\[\square\]

5.6.2 Time complexity

The algorithm runs in polynomial time, which is analysed formally as follows.

**Theorem 5.2** (Polynomial runtime complexity). The runtime of Algorithm 5.1 is \(O(|P|^2 \cdot |T|^2)\), where \(|P|\) is the spatial resolution and \(|T|\) is the temporal resolution of the problem.

The claim in Theorem 5.2 is justified as follows. Let the number of vertices be \(|V|\) and the number of edges be \(|E|\). The complexity for generating the set of vertices is \(O(|V|) = O(|P| \cdot |T|)\) and for the edges is \(O(|E|) = O(|V|^2)\). Therefore the computation time for generating the graph is \(O(|V|^2) = O(|P|^2 \cdot |T|^2)\). The topological sort has complexity \(O(|V| \log |V|)\), the graph search forward pass has complexity \(O(|V| + |E|)\), and the backtracking has complexity \(O(|T|)\). Therefore the computation time of the sweep-plane algorithm overall is \(O(|E|) = O(|P|^2 \cdot |T|^2)\).

5.6.3 Practical considerations

In practice there is a trade-off between solution quality and runtime of the algorithm. Firstly, the algorithm requires finite spatial and temporal resolutions, since the computation requires a finite set of vertices. This is not limiting since, in practice, there is little benefit in having a resolution higher than the positioning accuracy of the tracker vehicle. Secondly, for some prediction models it may be impractical to solve the observation value integral (5.6) exactly. In our implementation for the following
experiments we solve (5.6) using Monte Carlo integration, such that the belief of the target’s trajectory is approximated by a set of particles.

5.6.4 Stopping frequency

The algorithm automatically selects how many times the monitoring vehicle will stop by controlling the length of the stopping intervals. In most practical scenarios, it would typically be desirable to make few long stops rather than many short stops. Favouring few long stops over many short stops can be achieved by defining the travel time function $\delta$ in an appropriate manner. Constant time penalties can be introduced, which indirectly results in favouring longer stops. We provide an appropriate example definition of $\delta$ in Section 5.7.1.

5.7 Experiments

This section describes simulation experiments that illustrate the behaviour of the algorithm and advantages of the probabilistic formulation. First, we compare the current planning algorithm to previous algorithms for deterministic AUV missions (Best and Anstee, 2014). Second, we give an illustrative example that demonstrates the advantage of the probabilistic formulation of the problem. Third, we evaluate planning with probabilistic predictions that model temporal uncertainty. Later in Section 5.8, we demonstrate planning with two realistic trajectory and observation model implementations for application case studies.

5.7.1 Missions and parameters

The following simulations were performed using the same target trajectories and parameter values as the real AUV missions described in Best and Anstee (2014). Two hour-long AUV missions are considered as target trajectories, named Middle Harbour (depicted in Figure 5.6) and Jervis Bay. In Best and Anstee (2014), these missions were executed by a REMUS 100 AUV. Both missions alternate between densely scanning local regions of interest and moving in straight lines between these
regions. Two extreme cases for the trajectory are also considered: circular is a circular path with radius slightly less than the monitoring range, and linear is a straight path.

Parameter values are as follows: \( r = 200 \text{ m} \) monitoring range for an \( r \)-disk communication model (defined in Section 5.2.4), \( 2 \text{ m/s} \) constant target speed, \( 25 \text{ m} \) grid spacing, \( \Delta_t = 10 \text{ s} \) time steps, travel time between tracker waypoints

\[
\delta(\hat{y}_i, \hat{y}_j) = \frac{\|\hat{y}_j - \hat{y}_i\|}{\|\hat{y}\|} + T_{\text{pen}},
\]

with \( \|\hat{y}\| = 5 \text{ m/s} \) tracker speed and \( T_{\text{pen}} = 30 \text{ s} \) constant penalty for deploying and retrieving the monitoring hardware, and fixed start and end conditions, \( \hat{y}_1 = \mathbb{E}[X_1] \) and \( \hat{y}_M = \mathbb{E}[X_N] \) respectively, so that the tracker is in an appropriate position to deploy and recover the target vehicle. It is important to note that the travel time definition in (5.25) satisfies the triangle-inequality condition (Condition 5.1).

Computation times are shown for an unoptimised MATLAB implementation, running on a single core of an Intel i7 processor.

### 5.7.2 Deterministic target trajectory

Table 5.1 shows simulation results for four deterministic target trajectories. The deterministic and the probabilistic algorithms output the same solution trajectories,
Table 5.1 – Simulation results for deterministic target trajectories. The two planners output identical solution trajectories.

| Mission       | F/T | $M$ | $|V|$ | time (s) | $|V|$ | time (s) |
|---------------|-----|-----|------|----------|------|----------|
| Middle Harbour| 79.5| 8   | 2860 | 0.5      | 49483| 101      |
| Jervis Bay    | 79.2| 7   | 3121 | 0.6      | 59146| 144      |
| circular      | 95.8| 3   | 1203 | 0.3      | 27338| 32       |
| linear        | 52.2| 6   | 430  | 0.2      | 8062 | 3.7      |

Columns: Monitoring effectiveness as a percentage of mission duration; Number of stopping locations $M$; Number of vertices $|V|$; Computation time (s).

however the deterministic algorithm had a lower computation time due to the reduced number of vertices. The linear trajectory resulted in the lowest monitoring effectiveness since a straight-line target trajectory means the tracker cannot cut corners to reduce travel time. The linear trajectory required the fewest number of vertices since the convex hull of a straight line has the smallest area. The computation time is approximately quadratic in $|V|$ (regression fit with $R^2 > 0.99$), which agrees with the theoretical analysis. The algorithm shows a small improvement in the monitoring effectiveness over the greedy algorithm and genetic algorithm results reported in Best and Anstee (2014). The key advantage of the proposed sweep-plane algorithm is guaranteed and faster runtime (the deterministic algorithm is approximately 50 times faster than the genetic algorithm), and provably optimal solutions, as well as the applicability to probabilistic scenarios as demonstrated in the following experiments.

5.7.3 Planning with uncertainty

We demonstrate how planning while taking into account an accurate model for the uncertainty of the target trajectory improves the monitoring effectiveness. Figure 5.7 presents a target mission that alternates between sections with high spatial uncertainty and low spatial uncertainty.

Figure 5.7a shows the optimal stopping locations for the tracker if there were no uncertainty in the target trajectory. Figure 5.7b shows the solution when planning with a probability distribution $D_i$ that accurately models the uncertainty. The advantage
5.7 Experiments

![Figure 5.7](image)

**Figure 5.7** – Comparing planning with a deterministic model to planning with a probabilistic model. Green lines are sample target trajectories drawn from the probabilistic model. Red regions represent the monitoring range around the chosen stopping locations. Probabilistic planner achieves higher monitoring effectiveness since it selects regions with low spatial uncertainty.

of the probabilistic planning is that it chooses to stop at and stay longer in the regions with lower spatial uncertainty. For a Monte Carlo simulation drawing 10000 sample target trajectories, the deterministic planner has a mean monitoring effectiveness (as a percentage of mission duration) \( F/T = 47.5 \% \), while the probabilistic planner improves on this with \( F/T = 54.1 \% \). The solution path length given by the deterministic planner overestimates the expected monitoring effectiveness; conversely, the probabilistic planning accurately predicts the expected monitoring effectiveness.

### 5.7.4 Probabilistic trajectory with temporal uncertainty

Now we consider an example probabilistic target trajectory for an agent with uncertain speed. For a target with accurate localisation, uncertainty in position is usually due to variance in speed, rather than deviation from the path. To describe this, at time \( t_i \) the target is a distance \( d_i \) along the path from the start. We define \( d_i \) by the recursive equation

\[
d_{i+1} = d_i + \Delta t \dot{d}_i,
\]

(5.26)
where the speed $\dot{d}_i$ along the path at any time instance is assumed to be drawn from a Gaussian distribution that is independent of other time instances:

$$
\dot{d}_i \sim \mathcal{N}\left(\|\dot{x}\|_{\text{ave}}t_i, \sigma^2/\Delta t\right).
$$

The general solution to (5.26) (distance travelled along the path), for $d_1 = 0$ with zero uncertainty, is also Gaussian, with mean and variance increasing linearly over time:

$$
\mu_i = \|\dot{x}\|_{\text{ave}}t_i \quad \text{and} \quad \Sigma_i = \sigma^2 t_i.
$$

Figure 5.8 shows the results of Monte Carlo simulations performed by drawing 10000 sample target trajectories directly from (5.26), with the objective function evaluated for the planned tracker trajectory. Planning was performed using the mean only with no uncertainty (left bars) or using the uncertainty model (5.28) (right bars). The
5.8 Application case studies

horizontal axes shows varying speed uncertainty $\sigma \propto \sigma_{\text{rate}}$, where $\sigma_{\text{rate}}$ is the standard deviation of completion time in minutes for a 1 hour mission.

The monitoring effectiveness is significantly higher when planning using the uncertainty model, since the planner can choose to stop longer in regions with low spatial uncertainty. A single-tailed paired $t$-test confirms this performance improvement ($p < 0.001$) for all 20 missions except linear with $\sigma_{\text{rate}} \geq 4$. The probabilistic planning did not achieve significant improvements for the linear missions since there are no mission portions with relatively low spatial uncertainty (e.g., where the path folds back on itself). The monitoring effectiveness for all scenarios decreased as the uncertainty increased, since the probability of the target being within the 200 m communication radius at any position decreases. The monitoring effectiveness for the circular mission is less affected by increasing uncertainty since all possible positions for the target are within 200 m of a stopping location in the centre of the circle. The circular mission does not quite reach 100% monitoring effectiveness since the tracker is required to travel from the start location and to the end location.

5.8 Application case studies

In this section we present application case studies in autonomous underwater vehicle (AUV) and pedestrian monitoring to demonstrate the relevance and applicability of the problem formulation and algorithm to real-world scenarios. The purpose of the case studies is to: (1) demonstrate feasibility of the problem formulation and algorithm for realistic scenarios; (2) formulate implementations of realistic probabilistic prediction and observation models; (3) detail choices made while implementing our planner; (4) present extensive simulated experiments under various modelling assumptions; and (5) evaluate and discuss the simulation results with real data and Monte Carlo simulations.

5.8.1 AUV mission monitoring

Our first case study is for a mission monitoring scenario where an AUV is monitored by a surface vessel. We first formulate a realistic probabilistic model of the scenario,
(a) Eight sample paths from middle right (orange star) to bottom right. Magnified inset (right) illustrates the path uncertainty (not considering temporal uncertainty) in the middle region of the mission.

Figure 5.9 – Example predicted AUV trajectory given by the prediction model described in the case study, for the Middle Harbour mission (Figure 5.6).

(b) Spatial uncertainty over the mission duration. Measured as standard deviation of position for 1000 sample trajectories.

Figure 5.10 – Underwater acoustic communication model used in the AUV mission monitoring case study.

and then evaluate the performance of the algorithm under various modelling assumptions. The probabilistic formulation features a realistic probabilistic trajectory model (illustrated in Figure 5.9) and an underwater acoustic communication model (Figure 5.10). The formulation takes into account various causes of uncertainty in typical AUV missions in ocean environments, including localisation error, ocean currents, unpredictable mission pauses and unreliable communication. The formulation is motivated by our experiences with a REMUS 100 AUV in ocean environments, although is general enough to be adapted for other scenarios. The following simulation results show the advantages of taking into account these uncertainties when planning the trajectory of the monitoring surface vessel.
5.8 Application case studies

Simulation scenarios

We compare planning with and without a probabilistic AUV trajectory model, and with and without an acoustic communication model. The probabilistic trajectory model, detailed below, generates a set of 100 Monte Carlo sample paths to represent the belief of the AUV’s trajectory. The deterministic trajectory model is generated with a single simulation that assumes zero uncertainty. The acoustic communication model, detailed below, describes the probability of successful communication for a given distance between the AUV and the surface vehicle. The planning scenarios without the communication model use an $r$-disk model with $r = 264.2$ m, so that the two models have equal centroids. The tracker parameters are the same as in Figure 5.7.1, and planning is performed using the probabilistic algorithm in all cases. The following subsections describe in detail our formulation for implementing realistic prediction and communication models.

AUV prediction model implementation

The prediction model for the AUV’s motion is formulated probabilistically by adding various random disturbances to the deterministic mission plans in Section 5.7. The sequence of spatial probability distributions $(D_1, D_2, ..., D_N)$ of the AUV’s position is represented as a set of Monte Carlo sample trajectories of the following model. Each sample trajectory is calculated by iteratively: (1) updating the state by sampling a stochastic kinematics model, (2) adding localisation noise, and then (3) updating the controller using either a closed-loop control policy or executing a surfacing behaviour. Figure 5.9a shows an example set of sample paths and Figure 5.9b shows the spatial uncertainty (caused by path uncertainty and temporal uncertainty) over the duration of the Middle Harbour mission.

The stochastic kinematics model for the AUV is described as follows. The AUV moves through $\mathbb{R}^2$ with position coordinates $(x, y)$ relative to a fixed earth frame and heading $\theta$. The standard unicycle robot kinematics model, with speed $v$ and angular velocity $\omega$, is extended to include varying ocean currents $(\dot{c}_x, \dot{c}_y)$, speed and angular velocity control uncertainty ($\varepsilon_v$ and $\varepsilon_\omega$ respectively) and a maximum angular
velocity control \( \bar{\omega} \), giving the first-order equations:

\[
\begin{align*}
\dot{x} &= (v + \varepsilon_x) \cos \theta + \dot{c}_x \\
\dot{y} &= (v + \varepsilon_y) \sin \theta + \dot{c}_y \\
\dot{\theta} &= \omega + \varepsilon_\omega, \quad -\bar{\omega} \leq \omega \leq \bar{\omega}.
\end{align*}
\] (5.29)

The exact position of the AUV is not known and therefore the following controller is instead a function of the position estimate \((\tilde{x}_x, \tilde{x}_y)\). The position estimate has localisation error \((\varepsilon_x, \varepsilon_y)\), such that \((\tilde{x}_x, \tilde{x}_y) = (x_x + \varepsilon_x, x_y + \varepsilon_y)\).

We model the AUV’s controller using a non-linear feedback control policy (de Wit et al., 2012, Chapter 9.3) that controls the angular velocity \( \omega \) (which appears in (5.29)). This policy has the goal of reducing the difference between the current position/heading and the projection onto the desired path of the deterministic mission plan. This policy is asymptotically stable for the standard unicycle model under standard assumptions (de Wit et al., 2012), and simulations suggest it is also suitable for our extended kinematics model. More specifically, this policy is defined as follows.

Define the point \( M \) as the orthogonal projection of \((\tilde{x}_x, \tilde{x}_y)\) onto the mission path. The position error \( \tilde{l} \) is defined as the signed distance between \((\tilde{x}_x, \tilde{x}_y)\) and \( M \). At \( M \), the path has heading \( \theta_M \), and \( \tilde{\theta} \) denotes the heading error \( \tilde{\theta} := \theta - \theta_M \). The curvature of the path at \( M \) is \( C_M \) (we have \( C_M = 0 \) for the straight line segments considered in our example paths). The desired speed \( v \) is assumed to be constant and therefore the angular velocity \( \omega \) is the only control variable. Using these variables and parameters, the feedback control policy is defined as

\[
\omega = \begin{cases} 
-k_1 \tilde{l} \sin \tilde{\theta} - k_2 \tilde{\theta} + v \cos \tilde{\theta} \frac{C_M}{1 - C_M}, & \text{if } \tilde{\theta} \neq 0 \\
-k_1 \tilde{l}, & \text{if } \tilde{\theta} = 0
\end{cases}
\] (5.30)

where \( k_1 \) and \( k_2 \) are constants.

Some AUVs require pausing the mission and surfacing to, for example, receive a GPS fix or transmit data. We model unpredictable surfacing events as a stationary Poisson point process (Daley and Vere-Jones, 2003) with average rate \( \lambda \). The length of time that the AUV surfaces for each event is drawn from a known probability distribution. The AUV is unpowered while surfacing and therefore drifts with the ocean currents,
i.e., $\dot{x}_x = \dot{c}_x$, $\dot{x}_y = \dot{c}_y$ and $\dot{\theta} = 0$. Additionally, we assume that the localisation uncertainty is reset to zero while surfacing, i.e., $\epsilon_x = \epsilon_y = 0$.

The parameters of the model for our simulations are defined as follows. All probability distributions are Gaussian with mean $\mu$ and standard deviation $\sigma$, except where stated. The ocean currents in each axis have $\mu = 0$ m/s, $\sigma = 1$ m/s (although environment-specific models could be used if available; e.g. Witt and Dunbabin (2008)), speed has $\mu = 2$ m/s, $\sigma = 0.5$ m/s, maximum angular velocity $\bar{\omega} = \pi/32$ rad/s, angular velocity error has $\mu = 0, \sigma = \pi/64$ rad/s, localisation error accumulates linearly over time (e.g. due to using dead-reckoning) according to

\[
\epsilon_x^{t+1} = \epsilon_x^t + N \left( 0, \frac{\varsigma^2}{\Delta t} \right) \tag{5.31}
\]

\[
\epsilon_y^{t+1} = \epsilon_y^t + N \left( 0, \frac{\varsigma^2}{\Delta t} \right), \tag{5.32}
\]

with $\varsigma = 0.3$ m and $\epsilon_x = \epsilon_y = 0$ initial conditions, average surfacing rate $\lambda = 1$ /hour and a surfacing period uniformly distributed between 0 and 5 min.

**Acoustic communication model implementation**

Acoustic communication in ocean environments between an AUV and a surface vessel is highly unreliable. We simulate communication using a realistic underwater acoustic communication model proposed by Hollinger et al. (2011b). This model defines the probability $P_c$ of successful communication of a packet of data as a function of distance $d$ between two locations as

\[
P_c(d) = \left( 1 - \frac{A_0 d^k a(f_0)^d N(f_0) B}{4P} \right)^N. \tag{5.33}
\]

The model parameters that appear in (5.33) are functions of various characteristics of the environment and the communication hardware, such as transmission power, frequency, water depth, wind speed and shipping noise (Hollinger et al., 2011b). For the following simulated experiments, we chose these parameters arbitrarily, which results in the model shown earlier in Figure 5.10. This model has communication success probabilities of $P_c(100\, \text{m}) = 85\%$, $P_c(200\, \text{m}) = 52\%$ and $P_c(400\, \text{m}) = 6\%$, and has a centroid of 132.1 m.
**Planner implementation**

The planner requires evaluating the expected monitoring effectiveness (5.6) for stopping and monitoring at each candidate stopping location and time. For this case study, we compute the integral (5.6) as the normalised sum of the communication probabilities (given by the communication model) at the sampled AUV positions (given by the AUV trajectory prediction samples).

The planner also requires a predefined set of candidate stopping locations, and the algorithm is optimal with respect to this set. For this case study we employ an adaptive grid spacing so that the planner focuses more attention on promising regions. This is achieved by setting the grid spacing to 25 m in regions and times where the probability of successful communication $E[\tilde{f}]$ is greater than 0.6, 50 m for $E[\tilde{f}] > 0.4$, 100 m for $E[\tilde{f}] > 0.2$, and ignoring regions where $E[\tilde{f}] \leq 0.2$.

**Results and discussion**

Table 5.2 shows the simulation results for the four example missions and four planning scenarios. For each planning scenario, the first results column shows the monitoring effectiveness (as a percentage of mission duration) of the planned tracker trajectory when simulating the full prediction and communication models. For all missions, planning with the full prediction and communication models (1st row) achieves the highest monitoring effectiveness, which highlights the advantage of the probabilistic formulation.

Planning with deterministic predictions and the acoustic model (3rd row) achieves a moderate improvement in monitoring effectiveness over planning with probabilistic predictions and the $r$-disk model (2nd row). This suggests the communication model is more valuable than the probabilistic predictions in this instance; however, using both is the most valuable.

The objective function computed by the planner (2nd results column), which is measured relative to the current planning scenario rather than the full model, significantly overestimates the monitoring effectiveness relative to the full probabilistic model (1st column). This overestimate is worse in the 4th rows since the deterministic planning scenario is most different to the full probabilistic model.
Table 5.2 - Simulation results with the probabilistic AUV prediction model and underwater acoustic communication model.

<table>
<thead>
<tr>
<th>Mission</th>
<th>Planning Scenario</th>
<th>Results</th>
<th>Computation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$F/T$</td>
<td>$F/T$ planner</td>
</tr>
<tr>
<td>Middle Harbour</td>
<td>Prob. Acoustics</td>
<td>68.05</td>
<td>67.90</td>
</tr>
<tr>
<td></td>
<td>Prob. r-disk</td>
<td>64.14</td>
<td>82.06</td>
</tr>
<tr>
<td></td>
<td>Det. Acoustics</td>
<td>65.13</td>
<td>73.64</td>
</tr>
<tr>
<td></td>
<td>Det. r-disk</td>
<td>57.78</td>
<td>87.51</td>
</tr>
<tr>
<td>Jervis Bay</td>
<td>Prob. Acoustics</td>
<td>65.83</td>
<td>65.57</td>
</tr>
<tr>
<td></td>
<td>Prob. r-disk</td>
<td>57.19</td>
<td>81.94</td>
</tr>
<tr>
<td></td>
<td>Det. Acoustics</td>
<td>61.18</td>
<td>74.43</td>
</tr>
<tr>
<td></td>
<td>Det. r-disk</td>
<td>53.58</td>
<td>86.19</td>
</tr>
<tr>
<td>circular</td>
<td>Prob. Acoustics</td>
<td>57.68</td>
<td>57.68</td>
</tr>
<tr>
<td></td>
<td>Prob. r-disk</td>
<td>48.23</td>
<td>90.43</td>
</tr>
<tr>
<td></td>
<td>Det. Acoustics</td>
<td>52.37</td>
<td>67.46</td>
</tr>
<tr>
<td></td>
<td>Det. r-disk</td>
<td>50.06</td>
<td>97.29</td>
</tr>
<tr>
<td>linear</td>
<td>Prob. Acoustics</td>
<td>55.35</td>
<td>54.32</td>
</tr>
<tr>
<td></td>
<td>Prob. r-disk</td>
<td>53.20</td>
<td>60.97</td>
</tr>
<tr>
<td></td>
<td>Det. Acoustics</td>
<td>53.53</td>
<td>60.85</td>
</tr>
<tr>
<td></td>
<td>Det. r-disk</td>
<td>50.15</td>
<td>66.94</td>
</tr>
</tbody>
</table>

Columns: Monitoring effectiveness as a percentage of mission duration for probabilistic simulator and communication model; Objective function value (as a percentage of mission duration) output by planner, with respect to the planning scenario rather than the full model; Number of stopping locations $M$; Number of vertices $|V|$; Computation time (s). Best results with respect to full model are in bold.
The number of stopping locations $M$ is similar for all planning scenarios for three of the missions. However, for the circular mission, planning with the acoustics model results in more than double the number of stops than the $r$-disk model. This is because the acoustics model favours being close to the AUV and therefore the planner chooses to make many short stops near the predicted AUV position. The $r$-disk model treats all samples within the communication range as equally effective, and (for the deterministic trajectory case) all samples are within communication range of a single stopping location in the centre of the circle. Although the time spent deploying and retrieving the monitoring hardware is longer for the acoustic model case, the monitoring effectiveness is improved since the chosen stopping locations and times have a higher probability of successful communication.

The computation time is approximately quadratic in the number of vertices in the search graph $|V|$ (quadratic regression has $R^2 = 0.97$), which agrees with the time complexity analysis. In most cases, using deterministic predictions is slightly faster than probabilistic predictions, since it takes less time to generate the trajectory model and evaluate the observation value (5.6). Note that due to the filtering for the adaptive grid-spacing, the acoustics model was significantly faster than the $r$-disk model. The computation time for the fourth scenario could be reduced by using the deterministic planner instead (tested in Section 5.7.2) without compromising performance. In practical applications where computation time is more important than monitoring effectiveness, this planning scenario may be more desirable.

### 5.8.2 Pedestrian monitoring in cluttered environments

Our second case study is for a mission monitoring scenario where an aerial or ground vehicle monitors or aids a pedestrian or other similarly behaving agent. Similar to the first case study, we first formulate a realistic probabilistic model of the scenario, and then evaluate the performance of the algorithm under various modelling assumptions. The probabilistic formulation features a multi-modal intention-inference trajectory prediction model illustrated in Figure 5.11, and a visibility-based observation model that takes into account occlusions in a cluttered environment. Motivating scenarios include filming a sporting event with a mobile camera but stationary filming locations,
Figure 5.11 – The simulated office environment with an example observed trajectory (solid-orange), current position (yellow circle), 15 goal regions (green rectangles, with shade proportional to prediction probability), future trajectory prediction particles (colour changes over time), and the ground-truth future trajectory (dashed orange). Full details of this model are presented in Appendix A.

tracking an animal, monitoring boats in a cluttered harbour, aiding a disabled person, or monitoring other robots moving around a warehouse.

Simulation scenarios

The simulation results compare planning with and without taking into account the probabilistic prediction model, as well as with and without taking into account the occlusions in the observation model. The probabilistic predictions, detailed below, are represented by a set of 100 Monte Carlo sample paths, which are biased random walks through a probabilistic roadmap (PRM). The deterministic prediction is defined as the maximum likelihood estimate of the probabilistic model, which is the shortest path to the most likely goal region. Predictions were performed after the target moved through 5 edges of the PRM to improve the estimation precision. The tracker moves on average 5 times faster than the target, and both agents avoid collisions with static obstacles. The tracker starts at the same position as the target while the end position is to be optimised by the planner.

We consider two environments and ground truth trajectories: a footpath environment with a real pedestrian trajectory dataset with 442 trajectories (Lerner et al., 2007) (illustrated later in Figure A.6), and a more cluttered office environment with 100
random trajectories drawn from the same dynamics model (Figure 5.11).

**Trajectory prediction model implementation**

The prediction model for the target uses an intention inference trajectory prediction model. We provide a brief summary of this model here; full details are provided in Appendix A.

Figure 5.11 shows an example environment, trajectory and prediction. In this model, the target is assumed to be driven by the high-level intention to move to an unknown goal region within a cluttered environment. Prediction is based on the observed trajectory and a static environment map. The prediction algorithm first estimates the intended goal region using a recursive Bayes’ approach, and then uses the resulting probability distribution to perform Monte Carlo sampling of random walks through a PRM. Each random walk biases towards shortest paths to the estimated goal regions. The random walks are interpolated using a stochastic speed to give a set of 100 particles representing the predicted position of the target at every future timestep.

**Observation model implementation**

The tracker observes the target only when within an observation range ($r$-disk model) and the line-of-sight is not occluded by the static obstacles in the environment. As a comparison, we also consider an $r$-disk model that ignores occlusions. For the simulations we let $r$ be 20% of the width of each environment.

**Planner implementation**

For this case study, the tracker plans a path through a probabilistic roadmap that respects static obstacles in the environment. To achieve this, the spatial discretisation set $\mathcal{P}_1$ used during the vertex generation phase of the planner is defined as the same vertices in the PRM used by the prediction algorithm. The travel time between waypoints for the tracker is proportional to the shortest path through the euclidean-distance PRM plus a constant.
Since the PRM respects static obstacles, there are regions of the environment where the tracker cannot stop. This violates the condition $\text{CH} \subseteq \hat{Y}$ (Condition 5.2) required for Lemma 5.2 and therefore the convex hull culling $P_3$ cannot be used. This may increase runtime but does not affect optimality.

**Results and discussion**

The simulation results are shown in Table 5.3 averaged over the set of trajectories for each environment. A single-tailed paired $t$-test supports the hypothesis that in both environments, planning with the probabilistic model and the occlusions achieves a significantly higher monitoring effectiveness than the other planning scenarios. The hypothesis achieved statistical significance ($p < 0.001$ in all cases) when measured assuming either the full probabilistic trajectory prediction (1$^{st}$ results column) or the ground truth trajectories (2$^{nd}$ results column).

Planning while taking into account occlusions achieved a significant improvement in monitoring effectiveness compared to planning while ignoring the occlusions, and therefore shows the benefit of planning with a more accurate observation model. These improvements were more significant in the office environment compared to the footpath, since the office environment is more cluttered and therefore a larger portion of the in-range region was occluded from most positions.

In the office environment, the ground truth trajectories were samples generated from the probabilistic model, and therefore the monitoring effectiveness improvements between planning with the probabilistic to the deterministic model were similar when measured relative to the full model or to the ground truth. In the footpath environment, the ground truth trajectories were taken from the real pedestrian dataset, and the probabilistic planning still had a higher monitoring effectiveness than the deterministic planning. This suggests the intention inference model was a better estimator for the pedestrian trajectories than the shortest path model, and therefore improved the planning performance. Note that the ground truth monitoring effectiveness was much higher than the full model, since the former was measured up until the time when the sample trajectory reached the goal region, while the latter was measured until the longest sample reached its goal region.
Table 5.3 – Simulation results with the probabilistic pedestrian prediction model and $r$-disk with occlusions observation model. The pedestrian dataset had 442 pedestrian trajectories, and the office environment had 100 random trajectories.

<table>
<thead>
<tr>
<th>Environment</th>
<th>Planning Scenario</th>
<th>Results</th>
<th>Computation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Predictions</td>
<td>Occlusions</td>
<td>$F/T$ full model</td>
</tr>
<tr>
<td>Pedestrian dataset</td>
<td>Prob. Yes</td>
<td>56.11 (6.55)</td>
<td>66.74 (12.84)</td>
</tr>
<tr>
<td></td>
<td>Prob. No</td>
<td>55.55 (7.09)</td>
<td>65.86 (13.95)</td>
</tr>
<tr>
<td></td>
<td>Det. Yes</td>
<td>51.14 (9.07)</td>
<td>63.03 (16.22)</td>
</tr>
<tr>
<td></td>
<td>Det. No</td>
<td>50.14 (9.51)</td>
<td>62.36 (16.76)</td>
</tr>
<tr>
<td>Office environment</td>
<td>Prob. Yes</td>
<td>35.94 (9.45)</td>
<td>64.73 (23.77)</td>
</tr>
<tr>
<td></td>
<td>Prob. No</td>
<td>30.09 (9.08)</td>
<td>56.86 (23.88)</td>
</tr>
<tr>
<td></td>
<td>Det. Yes</td>
<td>29.54 (9.06)</td>
<td>50.47 (26.17)</td>
</tr>
<tr>
<td></td>
<td>Det. No</td>
<td>25.52 (8.23)</td>
<td>45.85 (23.05)</td>
</tr>
</tbody>
</table>

Columns: Mean monitoring effectiveness as a percentage of mission duration for probabilistic predictions and $r$-disk with occlusions observation model; Mean monitoring effectiveness as a percentage of mission duration for ground truth trajectory and $r$-disk with occlusions observation model; Num. vertices $|V|$, Computation time (s). Standard deviation in parenthesis. Best results with respect to full model and ground truth in bold.
5.9 Summary

We have proposed a spatiotemporal optimal stopping formulation and a polynomial-time sweep-plane algorithm for the stochastic mission monitoring problem. The algorithm solves the problem with a reduction to a longest-path search through a directed acyclic graph. The graph construction phase further reduces the size of the search space by exploiting geometric characteristics of the problem under reasonable assumptions. The simulation results validate the performance of our algorithm, show the value of the probabilistic formulation and describe implementations of probabilistic models for realistic applications. The algorithm admits a general class of probabilistic trajectory prediction models and probabilistic observation models, and therefore is applicable to a variety of real-world problems, such as the demonstrated AUV monitoring and pedestrian monitoring applications. Our implementation is unoptimised, but still exhibits reasonable clock-time performance ranging from milliseconds to a few minutes.
Chapter 6

Decentralised mission monitoring

In this chapter we consider a multi-tracker generalisation of the mission monitoring problem introduced in Chapter 5. An illustration of this generalised problem was presented earlier in Figure 1.5. The presented algorithm is a new decentralised planning algorithm that borrows and extends elements of Dec-MCTS (Chapter 3) and spatiotemporal optimal stopping (Chapter 5).

6.1 Overview

We formulate and solve a variant of mission monitoring where multiple tracker robots must monitor a single target robot. Various multi-robot problem settings are possible, but the case of multiple trackers observing a single target is of immediate practical value. Optimal single-tracker algorithms guarantee the best solution given a stochastic target trajectory, but do not necessarily guarantee any absolute level of quality. The target trajectory or communication channel may be subject to severe uncertainty that limits the probability of success of any single-tracker solution. Also, the tracker may be relatively slow-moving and therefore not be able to achieve the desired spatial coverage alone. Utilising multiple trackers provides a pathway for improvement by enabling the observation of multiple disparate possible target positions simultaneously.
One challenge in considering the multi-tracker case is that the single-tracker algorithm does not extend naturally. It is not useful for trackers to plan independently, because it is likely that all trackers would choose to make the same, rather than complementary, observations. Instead, each tracker must compute its actions jointly with the actions of the others. Hence, it is necessary to solve the resulting coordination problem. Ideally, this problem should be solved in a decentralised and asynchronous manner to distribute the computational effort, to avoid having a single point of failure, and to be robust to unreliable communication links.

In this chapter, we formulate the multi-tracker variant of mission monitoring as an extension of the formulation in Chapter 5, and propose a decentralised solution algorithm. The solution is motivated by the optimal single-robot planner in Chapter 5 with several modifications. For the multi-tracker scenario, the algorithm must be extended to consider the trajectories of the other agents. However, this alone is not enough to ensure successful coordination due to the cyclic dependencies between agents. We overcome this challenge by defining plans as probability distributions over trajectories that are optimised in a decentralised manner; this formulation reduces the likelihood of the algorithm getting stuck in a cycle of suboptimal solutions and is motivated by the findings in Chapter 3. The proposed algorithm has similar analytical properties as Dec-MCTS (Section 3.4), but has stronger convergence properties due to the use of the optimal single-agent planner. The algorithm also has the useful properties of being any-time, polynomial runtime per iteration, and small communication bandwidth usage.

We present simulated experimental results for a similar setting to the marine robotics setting from Section 5.8.1. The results demonstrate that the trackers must coordinate to adequately solve this problem rather than plan independently, there is significant benefit of using a probabilistic rather than deterministic plan representation, and our algorithm outperforms a generic decentralised planner. Overall, we show the approach is viable for practical use in multi-tracker mission monitoring.
### 6.1.1 Chapter outline

The remainder of this chapter is organised as follows. Section 6.2 discusses the contributions of this chapter in the context of the previous chapters. Section 6.3 formulates the multi-tracker mission monitoring problem. Section 6.4 presents a new decentralised planning algorithm for this problem. Section 6.5 discusses analytical properties of the algorithm. Section 6.6 presents new simulated experiments for an AUV mission monitoring scenario. Finally, Section 6.7 summarises the chapter.

### 6.2 Relationship to previous chapters

This chapter presents the first formulation and solution for decentralised multi-tracker mission monitoring. While this chapter builds upon the approaches and analyses presented in previous chapters, namely single-tracker mission monitoring and Dec-MCTS, this chapter contributes more than simply a combination of these two distinct ideas. We discuss the relationship between these ideas as follows.

#### 6.2.1 Single-agent mission monitoring (Chapter 5)

Relative to the contributions of Chapter 5, this chapter contributes a multi-tracker formulation of the mission monitoring problem, a modified single-tracker algorithm designed for efficiently performing multiple queries, a non-trivial decentralised generalisation of the single-tracker algorithm, and new analytical and empirical results.

#### 6.2.2 Dec-MCTS (Chapter 3)

Relative to the contributions of Chapter 3, this chapter contributes a new decentralised algorithm that is specifically designed for mission monitoring, has stronger analytical properties due to the use of the optimal single-agent planner, and is empirically demonstrated to significantly outperform Dec-MCTS at this problem.
6.3 Problem formulation

In this section we formulate the multi-tracker mission monitoring problem as a generalisation of the single-tracker mission monitoring problem (see Problem 5.1). We state the full problem here for completeness, with minor changes to the notation appropriate for this generalised problem.

The problem involves a team of mobile agents: 1) a target agent which follows a probabilistic trajectory defined by a mission plan, and 2) a team of tracker agents that seek to effectively monitor the target throughout the mission. At each time instant, to monitor effectively, at least one tracker must be stationary and within observation/communication range of the target. The trajectory for each of the trackers can therefore be characterised as a sequence of stopping waypoints in time and space. The optimisation problem is to find the trajectories for the team of trackers that maximises the expected monitoring effectiveness. This problem is to be solved in a decentralised manner. We formally define this problem as follows.

6.3.1 Target

The predicted future trajectory of the target is represented as a sequence of random variables $X := (X_1, X_2, ..., X_N)$ with associated timesteps $(t_1, t_2, ..., t_N) = \mathcal{T}$. The timesteps are evenly spaced at $\Delta t$ intervals, with $t_1 = 0$, and $t_N = T$ is the mission duration or a time horizon. Each $X_i$ represents the predicted location of the target at time $t_i$ and has a known probability density function $\rho_i(X_i = x)$ over the domain $\mathcal{X}$.

6.3.2 Tracker team

The target is monitored by a team of $R$ tracker agents $\{1, 2, ..., R\} = \mathcal{R}$. The trajectory of tracker $r \in \mathcal{R}$ is represented by a sequence of positions $Y^r = (y^r_1, y^r_2, ..., y^r_N)$ and states $S^r = (s^r_1, s^r_2, ..., s^r_N)$, with associated timesteps $\mathcal{T}$. The trajectory of the tracker is characterised as alternating between two states $\{\text{STOPPED, MOVING}\}$. If $s^r_i = \text{STOPPED}$, then at time $t_i$ tracker $r$ is stationary at position $y^r_i$. Conversely, if $s^r_i = \text{MOVING}$, tracker $r$ is moving between waypoints and therefore not monitoring.
The trajectory of tracker \( r \) is equivalently represented by the tuple \( \pi^r = [\hat{Y}^r, T^{ar}, T^{dr}] \), where \( \hat{Y}^r := (\hat{y}^r_1, \hat{y}^r_2, ..., \hat{y}^r_{M^r}) \) is a sequence of waypoint positions with sequences of associated arrival times \( T^{ar} := (t^{ar}_1, t^{ar}_2, ..., t^{ar}_{M^r}) \) and departure times \( T^{dr} := (t^{dr}_1, t^{dr}_2, ..., t^{dr}_{M^r}) \). We have \( \hat{y}^r_i \in \hat{Y} \), where \( \hat{Y} \) is a discrete set of positions where the trackers may stop.

During the time interval \([t^{ar}_i, t^{dr}_i]\), tracker \( r \) is in the STOPPED state and is stationary at the waypoint position \( \hat{y}^r_i \in \hat{Y} \). During the time interval \([t^{dr}_i, t^{ar}_{i+1}]\), tracker \( r \) is in the MOVING state and is travelling between consecutive waypoints \( \hat{y}^r_i, \hat{y}^r_{i+1} \). The sequences of arrival and departure times satisfy the constraints: \( t^{dr}_i \geq 0, t^{ar}_{M^r} \leq T \), and \( t^{ar}_i < t^{dr}_i < t^{ar}_{i+1}, \forall i \).

The required travel time between two waypoints \( \hat{y}^r_a, \hat{y}^r_b \) is defined by a function \( \delta(\hat{y}^r_a, \hat{y}^r_b): \hat{Y} \times \hat{Y} \rightarrow \mathbb{R}_{\geq 0} \). The proposed algorithm does not depend on the exact trajectory taken to achieve this travel time, but is likely to involve travelling at maximum speed to maximise the amount of time spent in the STOPPED state. We require \( \delta(\hat{y}^r_a, \hat{y}^r_b) = 0 \) iff \( \hat{y}^r_a = \hat{y}^r_b \). For clarity, we assume \( \delta \) is the same for all trackers, although this could be easily generalised. We do not plan for collision avoidance between trackers; however, this could typically be handled by a low-level controller during execution with only slight changes to the travel times.

The start position \( \hat{y}^r_1 \) for each tracker \( r \) is a known constant, while the end position \( \hat{y}^r_{M^r} \) is to be selected by the planner from a set \( \hat{Y}_{\text{end}} \subseteq \hat{Y} \). Alternative start/end assumptions are formulated for the related problem in Section 5.2.2; in this chapter we address this specific case, although the algorithm could be modified for other cases in a similar way to the methods in Section 5.4.3.

The trajectories for the team of robots collectively is denoted: \( \pi = \{\pi^1, \pi^2, ..., \pi^R\} \). The trajectory of all trackers except \( r \) is denoted \( \pi^{(r)} \), i.e., \( \pi^{(r)} := \pi \setminus \pi^r \). These superscript conventions are also used for all tracker variables: \( s_i, S, y_i, Y, \hat{y}, \hat{Y}, T^{ar} \) and \( T^{dr} \).
6.3 Problem formulation

6.3.3 Monitoring effectiveness

At time $t_i$, the monitoring effectiveness for tracker $r$ only is described by a function $f^r$. This function is defined as the probability of monitoring effectively:

$$f^r(X_i, y^r_i, s^r_i) := \begin{cases} \tilde{f}(\|X_i - y^r_i\|) & \text{if } s^r_i = \text{STOPPED} \\ 0 & \text{if } s^r_i = \text{MOVING} \end{cases}$$ (6.1)

where $\tilde{f}(d) : \mathbb{R}_{\geq 0} \rightarrow [0, 1]$ is the observation (or communication) model. This model $\tilde{f}$ describes the probability of successfully observing the target from a distance of $d$, although other interpretations of $\tilde{f}$ are possible (Section 5.2). This function may be defined as a simple binary $r$-disk model or a more realistic observation model; we presented example definitions earlier in Section 5.8. For clarity, we define the observation model as tracker-, translation-, orientation- and time-invariant, however the approach can readily be extended for more general models.

The goal of the team of trackers is to collectively monitor the target. At time $t_i$, the monitoring effectiveness for the team is described by a function $f$, defined as follows. There is no additional reward for multiple trackers monitoring at the same time. However, having multiple trackers STOPPED at the same time increases the probability that at least one tracker is effectively monitoring. By assuming observation independence, and following a similar formulation to Sec. 3.3 of Best and Fitch (2016), we define $f$ as

$$f(X_i, y_i, s_i) := 1 - \prod_{r \in \mathcal{R}} [1 - f^r(X_i, y^r_i, s^r_i)],$$ (6.2)

which specifies the probability that at least one tracker is effectively monitoring at time $t_i$. The motivation for this formulation of $f$ is that this model encourages the different trackers to observe different parts of the prediction model distribution $X_i$; if $f$ was instead simply a sum of $f^r$ then all trackers would aim to observe the most likely realisation of $X_i$ only, which is likely to result in an undesirable behaviour of the trackers.

The objective function $F$ is defined as the expected monitoring effectiveness over the
duration of the mission:

\[
F(X, \pi) := \mathbb{E}_X \left[ \Delta_t \sum_{i=1}^{N} f(X_i, y_i, s_i) \right] = \Delta_t \sum_{i=1}^{N} \mathbb{E}_{X_i} \left[ f(X_i, y_i, s_i) \right] \tag{6.3}
\]

where \(\{y_i, s_i\}\) are the trajectories derived from the plan \(\pi\), and the expectation is computed with respect to the probabilistic target trajectory \(X\).

### 6.3.4 Problem statement

The optimisation problem to be solved is stated as follows.

**Problem 6.1** (Decentralised mission monitoring). For a given probabilistic model of the predicted target trajectory \(X\), a set of possible waypoint locations \(\hat{Y}\), the start locations \(\hat{y}_r \in \hat{Y}\), \(\forall r \in \mathcal{R}\), and the set of feasible end locations \(\hat{Y}_{\text{end}} \subseteq \hat{Y}\), find for each tracker \(r\) the set of stopping waypoints \(\pi^r\) with positions \(\hat{y}^r_i \in \hat{Y}\), \(\hat{y}^r_{M^r} \in \hat{Y}_{\text{end}}\), arrival times \(T^a_r\) and departure times \(T^d_r\), such that the travel time constraints

\[
t^a_{i+1} - t^d_i = \delta(\hat{y}_i^r, \hat{y}_{i+1}^r), \forall i \in \{1, ..., M^r - 1\}, \forall r \in \mathcal{R} \tag{6.5}
\]

are satisfied, and the expected monitoring effectiveness for the team \(F(X, \pi)\), as defined in (6.4), is maximised over the mission duration.

Problem 6.1 is to be solved in a decentralised manner. Specifically, each tracker \(r\) optimises its own trajectory \(\pi^r\) based on only the information known to tracker \(r\). We assume tracker \(r\) knows the target prediction model \(X\), but does not necessarily know the trajectories \(\pi^{(r)}\) selected by the other robots. The trackers can communicate during planning-time to improve coordination, but this communication channel may be unpredictable and intermittent.
6.4 Decentralised planning algorithm

In this section we present our decentralised planning algorithm as a solution to the multi-tracker mission monitoring problem. The algorithm runs simultaneously and asynchronously on all tracker robots; we present the algorithm from the perspective of tracker $r$.

The algorithm cycles repeatedly between three phases: (1) find the optimal solution $\pi^*_r$ with respect to the currently known information about the other trackers’ plans, (2) maintain a set $\Pi'$ of possible solutions for $\pi^r$ and optimise a probability distribution $q^r$ over the set $\Pi'$, and (3) communicate probability distributions with the other robots. These three phases continue regardless of whether or not the communication was successful, until a computation budget is met or the algorithm converges. Pseudocode is provided in Algorithm 6.1.

Algorithm 6.1 Decentralised planning algorithm that optimises $\pi^r$ on-board tracker $r$.

1: $G \leftarrow \text{GenerateGraph}(X)$
2: $\Pi' \leftarrow \emptyset$ \hfill \textcircled{Set of solutions}
3: define $q^r$ as a probability distribution over $\Pi'$
4: $\beta \leftarrow \beta_0$ \hfill \textcircled{Temperature parameter}
5: loop
6: \hfill \textcircled{Phase 1: Spatiotemporal optimal stopping}
7: $\pi^*_r \leftarrow \text{OptimalStopping}(G, \Pi'(r), q(r))$
8: \hfill \textcircled{Phase 2: Probability distribution optimisation}
9: $\Pi'.\text{RemoveMin}(q^r)$ \hfill \textcircled{Remove least likely $\pi^r$}
10: $\Pi'.\text{Add}(\pi^*_r)$
11: \textbf{for each} $\pi^r \in \Pi'$ \textbf{do}
12: \quad $q^r(\pi^r) \leftarrow \text{Update}(q^r(\pi^r), \beta)$ \hfill \textcircled{Eqn. (6.8)}
13: $\beta \leftarrow \text{Cool}(\beta)$
14: \textbf{end for}
15: \hfill \textcircled{Phase 3: Communication}
16: $\Pi'(r), q(r) \leftarrow \text{CommunicateReceive}$

return $\pi^r \leftarrow \arg\max_{\pi^r \in \Pi'} q^r(\pi^r)$

6.4 Decentralised planning algorithm
6.4 Decentralised planning algorithm

6.4.1 Probability distributions over trajectories

The algorithm maintains a probability distribution for each tracker, which represents the predicted plan of each tracker. Specifically, we define a probability mass function \( q^r \), such that \( q^r(\pi^r) \) defines the probability that robot \( r \) will select the trajectory \( \pi^r \). The domain of \( q^r \) is restricted to a *dynamically* selected subset \( \Pi^r \) of all possible solution trajectories. As the algorithm progresses, both the domain \( \Pi^r \) and the probability distribution \( q^r \) are optimised. The product distribution of all trackers is denoted \( (\Pi, q) \), and of all trackers except \( r \) is denoted \( (\Pi^{(r)}, q^{(r)}) \).

6.4.2 Spatiotemporal optimal stopping

The first phase of the algorithm finds the solution \( \pi^*_r \) that is optimal with respect to the current information (probability distributions) available to tracker \( r \). This solution gets incorporated into the set \( \Pi^r \), which defines the domain of the probability distribution that is optimised later in phase 2.

We find \( \pi^*_r \) by extending the spatiotemporal optimal stopping algorithm for the single-tracker problem presented in Chapter 5. The algorithm consists of generating a search graph over time and space, followed by a longest-path search through the graph to find the optimal trajectory for the tracker. We extend the single-tracker algorithm to also consider the current plans for the other trackers when evaluating the new reward function (6.2). We also modify the graph generation of Chapter 5 to enable more efficient repeated queries, which is particularly useful in the context of Alg. 6.1. We summarise the algorithm as follows and highlight the main differences to the related algorithm of Chapter 5.

Graph generation

During a precomputation step (Algorithm 6.1 line 1), a graph \( G = (V, E) \) is generated such that any path through this graph represents a trajectory \( \pi^r \) for tracker \( r \). Each vertex \( v_\eta \in V \) represents a potential stopping location in time and space. The set \( V \) is generated by first considering the set of potential stopping locations \( \hat{Y} \). This set is then culled by only keeping positions that are within the observation range of any
part of the mission $X$, or within the convex hull of the mission $X$. An example of these sets and the resulting stopping locations $\mathcal{P}$ is depicted earlier in Figure 5.2.

Each vertex $v_\eta \in \mathcal{V}$ represents a position $p_\eta \in \mathcal{P}$ and a time interval $[\tau_\eta, \tau_\eta + \Delta \tau] \subseteq \mathcal{T}$, denoted by the tuple $v_\eta := [p_\eta, \tau_\eta]$. For each position $p_\eta \in \mathcal{P}$, a vertex is created for each time step $\tau_\eta \in \mathcal{T}$ where the target has a non-zero probability of monitoring the tracker. An example of this vertex generation is illustrated earlier in Figure 5.3 overlaying a probabilistic target trajectory represented by a set of sample trajectories.

A solution trajectory is represented by a path through the graph with consecutive vertices connected by directed edges $e_\eta \in \mathcal{E}$. An edge $e_\eta = (v_i, v_j)$ describes travelling from vertex $v_i$ at position $p_i$ to vertex $v_j$ at position $p_j$ at some time in the solution trajectory. Edges are connected between each pair of vertices that have feasible travel times, i.e., edge $(v_i, v_j) \in \mathcal{E}$ iff $\delta(p_i, p_j) \leq \tau_j - \tau_i$. The arrival time at $p_j$ is selected as $t_{ar}^j = \tau_j$ and the departure time from $p_i$ is $t_{dr}^i = \tau_i - \delta(p_i, p_j)$. Any vertex (and associated edges) that has no feasible path back to the start vertex $[\hat{y}_r^t, 0]$ is excluded from $\mathcal{V}$.

Unlike in the approach presented in Chapter 5, to make repeated queries more efficient, we make a further adjustment to the set $\mathcal{E}$. For a fixed vertex $v_j$, if there are multiple feasible edges $(v_i, v_j)$ with different $v_i$ that all have the same location $p_i$, then only the edge with the latest $\tau_i$ is kept, while all others are excluded. Optimality is maintained after this adjustment (corollary of Remark 5.2 from Chapter 5). As discussed later in Section 6.5.1, this improvement yields a runtime complexity for phase 1 that is linear in the resolution of the temporal discretisation (in contrast to the quadratic runtime achieved in Chapter 5).

**Graph edge weights**

In the main loop of the algorithm (line 5), rewards are assigned to the graph edges and then the optimal $\pi_r^*$ is found. This is performed while considering the plans $(\Pi^{(r)}, q^{(r)})$ of the other trackers, which change each time a communication message is received.

The reward $\omega_\eta$ for edge $e_\eta = (v_i, v_j)$ represents the relative value of including edge $e_\eta$ in the solution path $\pi_r^*$. Specifically, we define $\omega_\eta$ as the expected increase in
6.4 Decentralised planning algorithm

probability of effective monitoring if tracker \( r \) were to stop at location \( p_i \) at time \( \tau_i \), i.e.,

\[
\omega_\eta := \mathbb{E}_{X, y_i^{(r)}, s_i^{(r)}} \left[ f(X_i, y_i, s_i^{(r)} \cup s_i^{r} = \text{STOPPED}) - f(X_i, y_i, s_i^{(r)} \cup s_i^{r} = \text{MOVING}) \right]
\]

(6.6)

where \( y_i^{r} = p_i \), and \( f \) is defined in (6.2). The purpose of computing the increase in probability rather than absolute probability is to focus on the additional utility contributed by tracker \( r \) only and be less affected by noise caused by uncertainty in the other trackers’ plans. This expectation (6.6) evaluates to 0 during the timesteps that tracker \( r \) is moving from \( p_i \) to \( p_j \) and thus these timesteps do not need to be evaluated.

The expectation in (6.6) is computed with respect to several random variables. The \( y_i^{(r)}, s_i^{(r)} \) variables represent the position and state of the other trackers, which can be considered by summing over the discrete probability distribution \( (\Pi^{(r)}, q^{(r)}) \), while evaluating the positions \( p_i^{(r)} \) of \( \pi^{(r)} \in \Pi^{(r)} \) at time \( \tau_i \). If the number of robots or the cardinality of \( \Pi \) is large, then this summation becomes intractable, and therefore should instead be approximated using sampling; typically, a small number of samples would be adequate. The \( X_i \) variable is the location of the target, and this variable is considered by integrating with respect to the PDF \( \rho_i \). The best way to compute this integral would depend on the representation used by the prediction model; we use a sampled representation for \( X \) and evaluate using Monte Carlo integration.

We note that, unlike the definition of \( \omega_\eta \) in Chapter 5, in (6.6) we have ignored the effect of having departure times \( t_i^{r} \) that fall between the discrete time indices. This reduces the computation time since edges \( \langle v_i, v_a \rangle \) and \( \langle v_i, v_b \rangle \) will have the same weight \( \omega_\eta \), and thus (6.6) only needs to be evaluated once for each vertex, rather than for each edge.

**Graph search**

The optimal tracker trajectory \( \pi^{*}_r \) is found by searching for the longest-path through the graph \( G \). For general graphs, a longest-path search is NP-hard. However, \( G \) is a directed acyclic graph, and thus we can find the optimal longest-path in polynomial time. This search can be thought of a sweep-plane moving forwards through time.
As each vertex \( v_j \) is visited, the optimal edge \( \langle v_i, v_j \rangle \) is stored, which represents the optimal path if the trajectory were to finish at \( v_j \). The vertex with location in \( \hat{\mathcal{Y}}_{\text{end}} \) that has the largest accumulated reward at time \( t_N \) is selected as the end vertex. Finally, the trajectory \( \pi^*_r \) is found by backtracking from the end vertex to the start vertex.

### 6.4.3 Decentralised coordination

In phase 2, the trackers coordinate their plans by jointly optimising a probability distribution over their trajectories in a decentralised manner. The domain of the probability distribution \( \Pi^r \) for tracker \( r \) is constructed using trajectories generated in phase 1. The probability distribution \( q^r \) optimised in phase 2 is communicated to the other trackers during phase 3, then used by the other trackers when planning their own trajectories.

**Domain construction**

The domain \( \Pi^r \) is constructed by adding the trajectory \( \pi^*_r \) each time phase 1 is run. The set \( \Pi^r \) should be a small set to keep communication packets small and computation efficient; thus, once a fixed size has been reached, a trajectory is also removed each time one is added. The trajectory with the lowest probability \( q^r(\pi^*_r) \) is selected to be removed. When a new trajectory \( \pi^*_r \) is added, it is assigned a probability

\[
q^r(\pi^*_r) = \max_{\pi^r \in \Pi^r} q^r(\pi^r),
\]

and then \( q^r \) is renormalised. This is performed in Algorithm 6.1 lines 9–10. This construction of \( \Pi^r \) is motivated by, but distinctly different to, Dec-MCTS since the phase 1 of this algorithm generates a new, optimal solution at each iteration; in contrast, Dec-MCTS periodically resets \( \Pi^r \) since the phase 1 equivalent for Dec-MCTS is an incremental, converging planner. The benefits of defining \( \Pi^r \) as a compact set with cardinality greater than 1 is analysed in Section 6.5 and demonstrated empirically in Section 6.6.
6.4 Decentralised planning algorithm

Distribution optimisation

The probability distribution $q^r$ is optimised using a decentralised gradient descent scheme that is equivalent to the second phase of Dec-MCTS (Algorithm 3.3), which is an adaptation of probability collectives (Wolpert and Bieniawski, 2004). We chose to use this approach since it has interesting theoretical and practical properties, though other similar optimisation processes for $q^r$ could also be considered here. This descent scheme is formulated as finding the $q^r$ that has minimum KL-divergence to the optimal joint probability distribution. Thus, this formulation indirectly optimises the joint plans of the team in a distributed manner. This is performed in Algorithm 6.1 lines 11–12. The formulation is equivalent to the descent scheme presented in Algorithm 3.3; for completeness, we restate it as follows using the notation specific to decentralised mission monitoring.

Specifically, this distribution optimisation is defined such that during each phase 2, each component $q^r(\pi^r)$ of $q^r$ is updated as

$$
q^r(\pi^r) \leftarrow q^r(\pi^r) - \alpha q^r(\pi^r) \times \left[ \frac{E_\pi[F(X, \pi)] - E_{\pi(r)}[F(X, \pi) | \pi^r]}{\beta} + H(q^r) + \ln(q^r(\pi^r)) \right],
$$

where $H$ is Shannon entropy, $\alpha$ is a small constant step size, and $\beta$ is a temperature parameter that is cooled to slowly decrease the entropy of the distribution. The intuition behind this update step (6.8) is that the probability of selecting $\pi^r$ is increased if selecting $\pi^r$ would result in a larger expected reward $E_{\pi(r)}[F(X, \pi) | \pi^r]$ compared to the expected reward $E_\pi[F(X, \pi)]$ if tracker $r$ were to sample a trajectory from $q^r$. The last two terms in (6.8) control the entropy of the distribution, which is reduced slowly to avoid making a decision too quickly and getting stuck in local optima. The expectations in (6.8) are computed as

$$
E_\pi[F(X, \pi)] := \sum_{\pi \in \Pi} F(X, \pi) \prod_{r' \in R} q^{r'}(\pi^{r'})
$$

(6.9)
and similarly for $\mathbb{E}_{\pi^{(r)}}[F(X, \pi) \mid \pi^r]$ except tracker $r$’s trajectory is fixed as $\pi^r$, i.e.,

$$
\mathbb{E}_{\pi^{(r)}}[F(X, \pi) \mid \pi^r] := \sum_{\pi^{(r)} \in \Pi^{(r)}} \left[ F(X, \pi) \prod_{r' \in R \setminus r} q^{\pi^r}(\pi^{r'}) \right].
$$

(6.10)

Typically, it is necessary and satisfactory to approximate these summations in (6.9) and (6.10) by sampling from the joint distribution $q$. The distribution is renormalised after each update.

### 6.4.4 Communication

In phase 3 (Algorithm 6.1 lines 15–16), tracker $r$ communicates its current probability distribution $(\Pi^r, q^r)$ to the other trackers. If tracker $r$ receives an updated distribution $(\Pi'^r, q'^r)$ from tracker $r'$, then this replaces the locally stored distribution for $r'$. The updated distribution is used during phases 1 and 2 of the next iteration. If messages are lost, e.g. due to an unreliable communication channel, then each tracker will continue planning based on the most recently received distributions. In Section 3.7, we presented an extension to this communication phase approach that incorporates communication scheduling; this extended approach could also be directly applied to the decentralised algorithm presented here.

### 6.5 Analysis

This section analyses the runtime and optimality properties of the proposed algorithm by leveraging analytical results from the previous chapters.

#### 6.5.1 Runtime

The proposed algorithm is an iterative algorithm and a feasible solution is computed at each iteration, and thus is any-time. It is difficult to determine how many iterations are required before the algorithm reaches a satisfactory solution (as discussed in Section 6.5.2), however we can analyse the runtime per iteration as follows.
As stated earlier in Theorem 5.2, the computation time for the single-tracker spatiotemporal optimal stopping algorithm is $O(|P|^2|T|^2)$ where $|P|$ is the spatial resolution and $|T|$ is the temporal resolution of the problem. For the multi-tracker problem, we have split this algorithm into a precomputation step, which has runtime $O(|P|^2|T|^2)$, and phase 1, which has runtime

$$O(\psi|V| + |E|) = O(\psi|P||T| + |P|^2|T|)$$

where $|V|$ is the number of graph vertices, $|E|$ is the number of edges and $\psi$ is the time taken to compute the expectation (6.6). We note that this runtime for phase 1 is linear in $|T|$, rather than quadratic as seen in the proof of Theorem 5.2; this improved runtime is due to the removal of unnecessary edges from $E$ during the precomputation step (see Sec. 6.4.2).

While these runtimes are polynomial, $\psi$ is exponential in the number of robots if (6.6) is computed exactly, but can be efficiently and adequately approximated using sampling, as discussed in Section 6.4.2. The runtime of phase 2 is dominated by computing the expectations (6.9), (6.10) and thus should also be approximated using sampling.

### 6.5.2 Optimality and convergence

Due to the inherent challenges of decentralised planning, it is difficult to provide any guarantee of global optimality (as discussed in the related discussion in Section 3.4.3). However, we can analyse algorithms to provide insight into their behaviour; in this subsection we analyse the two main components of the algorithm and their interaction to support the use of these components in our algorithm.

As stated earlier in Theorem 5.1, the single-agent spatiotemporal optimal stopping algorithm is optimal, including when performing the vertex culling in the graph generation phase. This optimality result directly applies to the phase 1 of the decentralised algorithm, where optimality is measured with respect to the current information (distributions $\Pi^{(r)}, q^{(r)}$) available to tracker $r$.

However, we emphasise that this optimality result does not imply global optimality for the joint plan of the team. In fact, if $\Pi^{(r)}, q^{(r)}$ changes then tracker $r$ may change
its decision and vice versa, and this may continue indefinitely. This observation motivates the need for phase 2 of the algorithm, which maintains and optimises a probability distribution over trajectories. This formulation ensures that each tracker gradually settles upon a solution as the entropy is lowered over time, rather than responding drastically to changes in $\Pi^{(r)}$, $q^{(r)}$, and thus will typically overcome the cyclic dependency problem. Additionally, our phase 2 is closely related to phase 2 of Dec-MCTS, and thus the analysis of Proposition 3.1 holds here: the product distribution $q$ asymptotically converges to a distribution that locally minimises the KL-divergence to the optimal joint distribution, assuming that the sets $\Pi^r, \forall r \in \mathcal{R}$ are selected sufficiently. It is unclear what defines a sufficient selection of solutions for $\Pi^r$; however, we argue that our method in Sec. 6.4.3, which uses the candidate solutions generated by phase 1, is an appropriate heuristic for this purpose. The following experiments empirically support these claims.

6.6 Experiments: AUV mission monitoring

These experiments demonstrate the behaviour and performance of the planning algorithm for AUV mission monitoring, and support our theoretical claims. We simulate a scenario where an AUV is monitored by a team of surface vessels that communicate with the AUV via acoustic communication. The AUV follows a mission plan, and its motion is predicted using a probabilistic model with multiple sources of uncertainty. The surface vessels coordinate their plans using the proposed approach. A geometric representation of this scenario is illustrated in Figure 6.1. This scenario is a multi-tracker generalisation of the AUV case study presented earlier in Section 5.8.1.

6.6.1 Scenario

The prediction model for the AUV probabilistically simulates the trajectory of a path-following mission; we use the 7 km Middle Harbour mission introduced earlier in Section 5.7.1 and Figure 5.6. The AUV is simulated to follow this mission with various random disturbances, as described by the model in Section 5.8.1. We add an adaptive behaviour to this model to create a multi-modal distribution and thus
6.6 Experiments: AUV mission monitoring

(a) With the proposed algorithm, trackers successfully coordinate their actions to achieve a monitoring effectiveness of 60%.

(b) Without coordination, the trackers’ planned monitoring regions significantly overlap, resulting in a lower monitoring effectiveness of 37%.

Figure 6.1 – Comparing planning with and without communication by a team of 5 trackers. See Figure 1.5 caption for a description of this graphical representation.

requires using a team of trackers to monitor effectively; three decision points are added to the mission where the AUV may deviate from the path to visit another fixed point 200 m away. Observations are modelled as acoustic communication between the AUV and the surface vessels using the model in Section 5.8.1 but with a shorter range to further necessitate using multiple trackers.

Between 4 and 8 surface vessels coordinate in each scenario. Parameters are chosen
such that the mission has 1 hour duration, the AUV moves at 2 m/s, the trackers move at 3 m/s with a 60 s time penalty for each occasion the tracker stops, the trackers start near the AUV positioned 50 m apart, and the communication model parameters are such that the probability of an observation is defined as a continuous function of distance with 100% probability at 0 m, 75% at 30 m, 50% at 50 m and 0% for ≥ 250 m. The AUV trajectory is represented as a set of 100 particles. The feasible stopping locations $\hat{Y}$ form a grid with 50 m spacing, and time is discretised at 60 s intervals. Communication between trackers is performed asynchronously and assumed to be reliable.

6.6.2 Results

Benefits of coordination

First we demonstrate that the trackers must coordinate their actions in order to achieve a reasonable monitoring effectiveness. Figure 6.1a shows an example solution where the trackers successfully coordinate using the proposed algorithm to observe most of the AUV trajectory prediction model samples. In contrast, in the scenario shown in Figure 6.1b all trackers planned independently of other trackers, resulting in most trackers choosing overlapping observations in time and space, and therefore achieved poor monitoring effectiveness.

Probabilistic plans

The next results, shown in Figure 6.2, demonstrate the benefit of planning with a probabilistic representation of the plan. These results empirically support the theoretical analysis in Section 6.5.2. When the set of trajectories $\Pi^r$ is restricted to only keeping the current trajectory $\pi^r_*$, the solution switches back and forth between multiple suboptimal solutions as each tracker immediately reacts to the other trackers changing their plan. The trackers do not settle on a single solution as the algorithm progresses, and the monitoring effectiveness remains relatively low. On the other hand, planning with a probabilistic representation quickly converges to a stable solution that clearly outperforms the deterministic case. Similar results are achieved
between planning with a distribution size limited to 5 or keeping all past trajectories, and thus it is recommended to use a small size since it takes less samples to adequately estimate the expectations (6.9), (6.10).

Comparison to Dec-MCTS

Finally, we compare the proposed approach to Dec-MCTS (Chapter 3) as an example generic decentralised planner. These two planners have a similar three-phase cycle with similar phases 2 and 3. The key difference is that in phase 1 Dec-MCTS employs a generally applicable, incremental planner, which is a novel variant of MCTS. For our decentralised mission monitoring algorithm, our phase 1 is a problem-specific solution that is optimal with respect to the current information available.

The results are shown in Figure 6.3. Our proposed approach achieves the fastest convergence and the best overall performance. Dec-MCTS has interesting theoretical convergence properties, but its practical performance is largely dependent on the choice of rollout policy used for guiding the sampling of the search tree (James et al., 2017). Dec-MCTS with a rollout policy equivalent to our phase 1 was slower to converge but achieved similar results given sufficient iterations. Dec-MCTS with a purely random rollout policy clearly achieved the worst results. Dec-MCTS with a 50% mixture of the two rollout policies achieved results somewhere in the mid-
6.7 Summary

We have formulated and solved the multi-tracker mission monitoring problem. The problem is formulated as maximising expected observation time with respect to prob-

---

Figure 6.3 – Comparison between our proposed method and the generic Dec-MCTS decentralised algorithm with three different rollout policies (in parenthesis). 4 robots are simulated in a smaller problem instance than the Figure 6.2 scenario. Averages and quartiles shown from 20 trials.

dle. These results show our approach outperforms a generic planner at multi-tracker mission monitoring, and also show that our approach can be utilised as a guiding heuristic to greatly improve the performance of a generic planner.

We note that since the problem is new, we do not have algorithms for direct comparison other than generic planners such as Dec-MCTS. It would be interesting to compare to an optimal centralised algorithm though a key challenge that would need to be overcome first is the intractability of searching over this joint solution space that grows exponentially in the number of robots.

**Computation time**

The experiments were simulated in MATLAB on a standard desktop computer, and the computation times were on the order of several seconds to minutes for all scenarios. We note that, in practice, computation time can be tuned by varying the number of iterations and the discretisation resolution to meet the requirements of an application.

6.7 Summary

We have formulated and solved the multi-tracker mission monitoring problem. The problem is formulated as maximising expected observation time with respect to prob-
abilistic models of the target dynamics and communication. This has broad practical applications, especially for performing real-world marine robotics missions. The solution is a novel decentralised algorithm that inherits and extends several useful analytical and practical properties from approaches for the single-tracker problem and generic decentralised planning. In our experiments, the solutions are reached after only a small number of communication messages are broadcast by each tracker.
Chapter 7

Conclusions and future work

The active perception methodology of utilising planned motion to better perceive the environment has great potential to improve information gathering performance in a wide range of scenarios. Scaling this methodology up for teams of robots enables achieving an improved set of viewpoints in time and space; however, multi-robot systems introduce the additional challenge of interconnecting active-perception modules that are distributed over multiple robots. In this thesis, we have particularly focused on improving system-level performance by proposing new algorithms suitable for the planning module of multi-robot active perception systems.

The contribution of this thesis is a suite of planning algorithms for multi-robot active perception. The algorithms are designed for a variety of active perception formulations with different perception objectives. We consider both centralised and decentralised coordination, and emphasise online computation and anytime solutions. Several of the algorithms are also inherently relevant to single-robot active perception. A variety of analytical results describe useful properties of the behaviour of the algorithms. Empirical results with realistic perception objectives provide strong evidence that our proposed solutions are suitable for real-world applications.

In this concluding chapter, we provide a summary of this thesis in Section 7.1 and a summary of the main contributions in Section 7.2. In Section 7.3 we discuss areas for future work, and in Section 7.4 we end this thesis with an outlook to the future of multi-robot active perception systems in real-world scenarios.
7.1 Thesis summary

This thesis addressed the problem of selecting valuable sequences of viewpoints for a team of information-gathering robots. We addressed this problem for a variety of generic and task-specific perception objectives. We summarise the active perception formulation considered in each chapter of this thesis as follows, as well as the associated proposed planning algorithms and results.

7.1.1 Dec-MCTS (Chapter 3)

We first considered a generic formulation where the rewards are defined by any arbitrary objective function over the action sequences of the robots. This formulation is relevant to any active perception task, such as object classification and target search.

We solved this problem in a decentralised manner, such that each robot plans its own sequence of actions, while using available communication resources to coordinate their plans with other robots. We proposed Dec-MCTS as a powerful and generally-applicable solution algorithm for this context. In Dec-MCTS, each robot expands a search tree over possible action sequences using a novel variant of MCTS. This tree is periodically compressed in the form of a probability distribution over plans and shared with other robots. The tree expansions are performed while considered any information provided by other robots.

We then presented several analytical results for Dec-MCTS that supported the use of the tree search and probability distribution optimisation components of the algorithm. The tree search phase is shown to successfully trade off between exploration and exploitation in this generalised scenario where the reward distributions change dynamically. The probability distribution optimisation approximates an importance-sampling variant of probability collectives, and therefore we have the proposition that the product distributions converge towards the optimal joint distribution. These two results do not yield any strong guarantees for global optimality, due to the complex interaction between these components of the algorithm; however, this result motivates the use of the two main algorithmic components of Dec-MCTS.

Empirical results were presented for two contexts: generalised team orienteering,
and active object classification. The empirical results demonstrated a robustness to communication loss, improved performance over a centralised MCTS, and the need to perform non-myopic planning. We also presented an extended algorithm that trades computation efficiency for more effective use of limited communication resources.

7.1.2 SOM (Chapter 4)

We then considered a more specific active perception formulation that can be thought of as a generalisation of the team orienteering problem. The robots are tasked to observe a discrete set of features in the environment by visiting overlapping polygonal regions of the environment. Correlations between viewpoints are modelled by the overlaps between the regions and the property that a region can only have its associated reward collected once. We demonstrate that this formulation forms an efficient representation of tasks such as coverage, scene reconstruction and object classification.

We addressed this problem in a centralised manner such that the plans for all robots are jointly optimised by a centralised processing unit. We proposed a SOM algorithm for this problem, which is a special type of neural network that exploits the geometry of the problem. This algorithm efficiently searches over a continuous space of candidate viewpoints to maximise the weighted sum of visited reward regions. It is an efficient heuristic algorithm, and hence does not provide any strong optimality guarantees, but we have proven the algorithm converges in polynomial time.

We provide empirical results for an extensive set of test cases and formulations, where the main findings show the benefits of joint multi-robot planning, the benefits of defining continuous regions, significant performance improvements over the generic Dec-MCTS algorithm, and the ability to efficiently replan online as information changes.

7.1.3 Spatiotemporal optimal stopping (Chapter 5)

The next active perception formulation we considered was the mission monitoring problem. This problem involves a target robot that stochastically follows a mission plan, and a tracker robot that aims to maximise expected observation time of the
7.1 Thesis summary

The key problem characteristic that we consider is that the tracker vehicle must be stationary to observe the target, which is motivated by operational practices in marine robotics. The problem is to plan the trajectory of the tracker, which is defined as a sequence of stopping intervals and their associated locations. We refer to this new problem as the spatiotemporal optimal stopping problem as a multi-dimensional generalisation of the well-known optimal stopping problems.

Our solution algorithm exploits geometric properties of the formulation by sliding a sweep-plane forwards through time, and connecting graph-edges that maximise the expected overlap between the probabilistic target trajectory and the probabilistic observation model. This algorithm is a type of maximum-weight graph search, where the key novelty is the construction of an efficient spatiotemporal graph that maintains solution optimality. The algorithm has polynomial runtime and is resolution complete.

We provided a set of case studies for scenarios with different models for the target trajectory and communication. The primary scenario is motivated by marine robotics, where the target is an AUV and the tracker is a monitoring surface vessel. The AUV’s predicted trajectory is modelled using Monte Carlo sampling that considers uncertainty in the localisation, ocean currents, and the mission plan. The results show the benefits of planning with respect to the probabilistic models, confirm the runtime is polynomial, and demonstrate significant improvements over prior work.

7.1.4 Decentralised mission monitoring (Chapter 6)

The final active perception formulation considered was a multi-tracker extension of the mission monitoring problem. For this scenario, we envision a team of surface vessels tracking an AUV, where the use of multiple trackers allows improved spatial and temporal coverage, particularly when there are large sources of uncertainty or the trackers are slow.

We addressed this generalised problem in a decentralised manner, in a similar context to Dec-MCTS. The proposed algorithm uses a similar three-phase cycle to Dec-MCTS except where MCTS is replaced with the optimal single-tracker algorithm for mission monitoring. Several other modifications were made to both Dec-MCTS and
mission monitoring to form this new algorithm, such as a new subset construction method relevant to this formulation. The algorithm has similar analytical properties to Dec-MCTS, but we expect faster and stronger convergence due to the use of the optimal single-tracker algorithm.

We demonstrated the behaviour of this algorithm for a multi-tracker AUV mission. The results showed the benefits of using a distribution of possible solutions, the algorithm significantly outperforms Dec-MCTS, and the algorithm can also be used as a heuristic with Dec-MCTS for improving the tree expansion.

7.2 Summary of contributions

In this section, we elaborate on and reiterate the significance of the claimed contributions of this thesis (enumerated earlier in Section 1.4).

7.2.1 Multi-robot active perception problem formulations

Throughout this thesis we have presented many new problem formulations for the planning component of multi-robot active perception systems. These new formulations reflect real-world tasks, while also being in a suitable form for designing and presenting effective planning algorithms. The formulations of Problem 1.1 and Problem 3.1 are generic in that an objective function can be defined for these problems to be suitable for any task. Most algorithms for multi-robot active perception planning are solving an instance of these problems.

In Section 3.5 and Problem 4.1 we presented a new formulation that generalises the orienteering problem. This formulation can efficiently encode viewpoint correlations and thus can be used as a heuristic for tasks such as weighted coverage (see Section 4.5), object recognition (see Section 4.5.1) and exploration (see Section 4.6). This formulation also motivated efficient algorithms that plan paths over continuous space. While we propose a solution algorithm for this problem, we also hope that this formulation motivates further development of generalised problems and new algorithms for these types of tasks.
In Section 5.2 and Section 6.3 we presented optimal stopping formulations of the mission monitoring problem. The formulations are probabilistic such that they admit probabilistic models for trajectory prediction and communication. This formulation is suitable for a variety of real-world tasks, particularly marine robotics operations (see Section 5.8.1) and pedestrian tracking (see Section 5.8.2). This formulation was also designed to enable algorithms with interesting theoretical (particularly optimality) and practical properties, to provide further algorithmic insight into the problem of mission monitoring. The example scenarios of Section 5.8 also featured several new or adapted methods for modelling probabilistic trajectories, communications and observations; most notable is the AUV trajectory prediction model in Section 5.8.1.

We also adapted an existing model for active object recognition tasks. This adapted formulation was presented in Section 3.6 as an example test case for Dec-MCTS. This formulation is suitable for demonstrating the behaviour of our algorithm, while also being representative of existing object recognition perception models and objectives.

### 7.2.2 Dec-MCTS algorithm

We proposed the Dec-MCTS algorithm as a new, generally-applicable algorithm for decentralised multi-robot planning. This algorithm admits an objective function defined over the action sequences of the robots, and thus is suitable for multi-robot active perception. Our analysis has shown that Dec-MCTS has many practically-useful properties, such as being anytime, online, non-myopic, balances exploration and exploitation of the search space, robust to unreliable communication, and allows incorporating prior knowledge. Dec-MCTS is the first decentralised variant of the widely-used MCTS. Thus, Dec-MCTS can solve multi-robot generalisations of many single-robot problems that have previously been solved by MCTS, such as object recognition, patrolling and manipulation (see Section 2.3.4). An extended algorithm is also presented that incorporates an additional phase for communication scheduling, which is designed to more effectively use limited communication resources.
7.2 Summary of contributions

7.2.3 SOM algorithm

We presented a new SOM algorithm designed for centralised multi-robot active perception. The algorithm is a new learning procedure for a special type of neural network, and is designed to solve a new active perception formulation. It is a heuristic algorithm that efficiently searches over continuous space and a long time-horizon, and has guaranteed polynomial runtime before convergence. The algorithm has similarities to related SOM algorithms, but offers new ways of meeting budget constraints, dividing the workload between robots, addressing non-uniform observation rewards, and performing online replanning.

7.2.4 Spatiotemporal optimal stopping algorithm

We proposed the spatiotemporal optimal stopping algorithm for single-tracker mission monitoring. The main novelty in the approach is a new spatiotemporal graph construction method, with associated analytical results, that is then exploited by a longest-path graph search. The algorithm plans with respect to probabilistic motion prediction and communication models. It has guaranteed optimality over a long planning horizon, has polynomial runtime, and empirically and analytically outperforms existing approaches. This is the first algorithm for solving a probabilistic formulation of single-tracker mission monitoring.

7.2.5 Decentralised mission monitoring algorithm

We proposed a new decentralised algorithm multi-tracker mission monitoring. This is the first solution algorithm for this problem. The algorithm combines and extends elements of Dec-MCTS and spatiotemporal optimal stopping in a non-trivial manner. The algorithm is decentralised, anytime, non-myopic, robust to unreliable communication, and plans under various sources of uncertainty. It has stronger convergence properties and empirical performance than Dec-MCTS for this particular problem. The similarities between this algorithm and Dec-MCTS further motivate the general three-phase approach used in both algorithms.
7.2.6 Analytical results

Each of the algorithms in this thesis is presented alongside an associated set of analytical results. These results are intended to analyse the behaviour of the algorithm, motivate and guide the use of the algorithms in practice, and provide insight into the multi-robot active perception planning problems. We highlight the main analytical results as follows, which are in addition to the general algorithmic properties (such as non-myopic, anytime, and robust to communication loss) mentioned above.

Our analysis of Dec-MCTS focussed on describing the behaviour of the main algorithmic components. The tree search component balances between exploring unknown regions of the search space and exploiting learnt information; a complete proof of this result was omitted from this thesis, but was presented previously in Best et al. (2018a). This result extends known results for standard MCTS for our generalised scenario where the reward distributions are changing. For the probability distribution optimisation component, we proposition that our approach to subspace selection approximates importance sampling. Thus, the algorithm has similar convergence properties to probability collectives, such that the product distribution converge towards minimising the KL divergence to the optimal joint distribution. These results do not yield global optimality guarantees, however they are relatively strong properties for this context of this general and decentralised problem formulation.

Our main analytical result for our SOM algorithm showed the algorithm converges in polynomial time. The proof of this result also provides insight into the runtime dependence on the number of robots (Remark 4.1) and the possibility of early convergence (Remark 4.2). This result supports the similar analysis of related SOM algorithms.

Our main analytical result for our spatiotemporal optimal stopping algorithm is that it finds the optimal solution trajectory in polynomial time. The proof of this result relies on several intermediate results that show geometrically that the graph construction phase maintains optimality. Interestingly, this result also shows that our optimal graph search algorithm is performed in polynomial time; a longest-path graph search is generally NP-hard, but can be performed in polynomial time in this context. The existence of our polynomial-time algorithm also immediately implies that our mission
monitoring problem formulation is in the P complexity class.

Our analysis for our decentralised mission monitoring algorithm builds on these previous results. In particular, the optimality result for spatiotemporal optimal stopping applies to each run of phase 1 of the algorithm. Phase 2 of the algorithm has similar convergence properties to the probability collectives component of Dec-MCTS. Our analysis also motivates the need for this phase of the algorithm, as it aims to avoid getting stuck in cycles of local minima. These results are similar to, but stronger than, the results of Dec-MCTS, due to the use of the optimal first phase that is designed specifically for mission monitoring.

7.2.7 Empirical results

Each algorithm is also presented alongside various empirical results for different scenarios. These results are intended to empirically confirm the analytical claims, show how the algorithms can be applied to various problems, and demonstrate the behaviour and performance of the algorithms. For all algorithms, the computation time is shown to be relatively fast and can be tuned to meet the requirements of a particular application. These experiments were performed in simulation within ROS or MATLAB, and use real-world datasets, perception models and objective functions.

For Dec-MCTS, we presented experiments for the context of generalised orienteering and active object recognition. These results showed our algorithm outperforms a centralised algorithm, outperforms a greedy algorithm, is robust to communication loss, and that it can be used for online replanning. Results are also shown for our extended algorithm that incorporates communication scheduling, which demonstrate that coordination performance can be maintained with significantly reduced communication by judiciously selecting when to communicate.

For our SOM algorithm, we presented an extensive set of experiments in a variety of problem instances. These results demonstrated the benefits of planning jointly as a team, the benefits of planning with non-uniform rewards, the benefits of planning with viewpoint regions, the convergence, several illustrative examples, improvements over Dec-MCTS for this problem, the applicability of online replanning for exploration, and the benefits of planning over a long time-horizon. These simulations used a
real 3D point cloud dataset of an outdoor scene, and example point-cloud processing methods.

For single-tracker mission monitoring, we presented an extensive set of experiments for simple scenarios, AUV monitoring scenarios, and pedestrian tracking scenarios. Overall, these results demonstrated the benefits of planning while considering the uncertainty, how to apply the algorithm with various prediction and communication models, improvements over previously proposed algorithms, and polynomial runtime. These experiments were based on real AUV missions, a real pedestrian trajectory dataset, and several new or adapted prediction and communication models.

We empirically evaluated our decentralised mission monitoring algorithm in similar AUV monitoring scenarios generalised for multiple trackers. These results demonstrated the need to coordinate plans, the benefits of the probabilistic planning, convergence of the algorithm, and that the algorithm significantly outperforms Dec-MCTS for this task.

7.3 Future work

The contributions of this thesis motivate many avenues of future work. We separate this section into (1) ideas for extending algorithms to potentially improve performance at the considered problems, (2) ideas for new generalisations of the considered problems that may be more appropriate for some applications, and (3) future hardware experiments.

7.3.1 Improving performance

We begin this section by discussing several ideas for potentially improving the performance of each of the proposed algorithms for the perception tasks considered in this thesis.
Dec-MCTS

The MCTS component of the proposed Dec-MCTS algorithm is a generalisation of the commonly-used UCT algorithm (Kocsis and Szepesvári, 2006). UCT has been widely used in the literature due to the associated analytical results, and also perhaps due to the widely-publicised empirical success in other domains. There has been some debate as to whether UCT really is the most suitable MCTS variant (Domshlak and Feldman, 2013). It may be worth investigating other MCTS variants that could be used within Dec-MCTS. For example, BRUE (Feldman and Domshlak, 2014) is an MCTS variant formulated using a different definition of regret. A challenge would be to generalise MCTS variants, similar to our D-UCT algorithm, for our domain that has changing reward distributions. It would be interesting to see if similar analytical results can be derived for these cases.

Another interesting line of inquiry is to incorporate coalition forming into Dec-MCTS. This may allow robots that depend more on each other to compute plans in a more tightly-coupled manner, while devoting less attention to less-dependent robots. As formulated, static coalitions of agents can be formed by generalising the product distributions in our framework to be partial joint distributions. The product distribution described in Section 3.3.4 would be defined over groups of robots rather than individuals. Each group acts jointly, with a single distribution modelling the joint actions of its members, and coordination between groups is conducted as in our algorithm. Just as our approach corresponds to mean-field methods, this approach maps nicely to generalised mean field inference (Xing et al., 2004) or region-based variational methods (Yedidia et al., 2005), and guarantees from these approaches may be applicable. It would also be interesting to study dynamic coalition forming, where the mapping between agents and robots is allowed to change, and to develop convergence guarantees for this case. A key challenge would be to determine which robots’ plans are more tightly coupled and therefore would benefit from planning within a coalition. Our approach of deciding which robots should communicate (Section 3.7) could help address this challenge.

In practice, the performance of MCTS is largely dependent on a suitable choice of rollout policy (Browne et al., 2012; James et al., 2017). Rollout policies that compute near-optimal paths would typically be too slow to compute, while purely random
policies typically result in requiring a large number of rollouts before convergence. In Chapter 6 we showed how Dec-MCTS can incorporate spatiotemporal optimal stopping as a rollout policy. It would also be interesting to consider if our SOM algorithm would be a useful rollout policy in some scenarios.

It would also be interesting to consider alternatives to PC for performing the probability distribution optimisation step. PC has interesting theoretical properties, but can be slow to compute due to the required sampling of expectations. Alternative approaches could be developed that more efficiently find reasonable distributions. It would also be interesting to reconsider how the subsets of candidate paths are selected, such as using other selection measures, or dynamically resizing the set.

**SOM**

Our SOM algorithm is designed to be an efficient heuristic algorithm and thus the focus is on having scalable runtime. Our algorithm scales well in terms of number of robots and number of viewpoint regions. However, one potential limitation is the runtime dependence on the relative reward weights; in instances where both very small and very large weights exist, the algorithm will be relatively slow. It would be interesting to develop alternative algorithms that overcome this potential issue. One approach could be to avoid node duplication by proposing an alternative adaptation function that is scaled by the weights. This approach is likely to be slower in many cases since the adaptations are smaller, but is likely to be more efficient in cases where there is a large variance in rewards.

As the SOM is a heuristic approach, it provides guarantees for runtime until convergence, but unfortunately does not provide any optimality or convergence-rate guarantees. An interesting line of inquiry would be to develop SOM-like algorithms that provide these guarantees, even potentially at the expense of practical performance since it may provide insight as to how to improve SOM algorithms in general.

**Mission monitoring**

The spatiotemporal optimal stopping algorithm performs optimally with respect to the model of the world, which may be probabilistic. If this model changes, then
replanning can be performed by repeating the algorithm when new information is obtained. However, it would be interesting to extend the approach to allow replanning for partially-known mission trajectories that are discovered over time. If the mission discovery depends on the tracker observations, the planner may need to consider the value of obtained information when planning sequences of stopping locations and times. In this case, the problem is likely to be intractable, and therefore extensions that give approximately optimal solutions should be developed, such as MCTS.

7.3.2 Problem variants and applications

Another promising line of future work is to consider new variants, generalisations and applications of the formulations considered in this thesis.

Dec-MCTS

It would be interesting to apply Dec-MCTS to multi-agent scenarios where standard algorithms already exist for associated single-agent scenarios. Problem-specific single-agent planning algorithms could replace the MCTS component of Dec-MCTS, while still performing the distributed product distribution optimisation phase, in order to provide stronger theoretical guarantees or algorithmic efficiency for special cases. Scenarios where this could be applicable include multi-robot persistent monitoring (Alamdari et al., 2014), cooperative wildlife localisation (Cliff et al., 2015), collision avoidance (Otte and Correll, 2013), and dynamic coverage problems (Hönig and Ayanian, 2016). We already demonstrated success at achieving this for decentralised mission monitoring.

The problem formulation we consider for Dec-MCTS is general in that we are interested in planning sequences of actions to optimise a joint objective function, without requiring assumptions such as submodularity. A straightforward extension to our approach would be to adapt the algorithm to address the Dec-POMDP formulation. This could be achieved by generalising the MCTS component of our algorithm to POMCP (Silver and Veness, 2010) (an MCTS-based solution for single-agent POMDPs) while still using our proposed D-UCT tree expansion policy. A difficulty
would be to efficiently find good-quality solutions while also considering probabilistic transition models and having the search tree branch for both actions and observations.

Incorporating collision avoidance objectives would also be particularly interesting. However, including this as a Dec-MCTS objective is likely to be particularly difficult since this imposes hard inter-robot constraints that may be difficult to capture with our probabilistic representation of paths. In practice, collision avoidance could be handled by a lower-level planner, but it would be interesting to consider how to include this directly within the higher-level planning algorithms.

Recently, we have seen an explosion of interest in applying deep learning methods to robotics problems (Sündenhauf et al., 2018). So far this has mostly been for the perception components of systems, but we expect to see this interest to increase for planning components also. There are downsides of using deep learning methods, such as requiring large amounts of training data, difficulties in guaranteeing robustness and predictability, and the models are difficult for humans to interpret, understand and learn from. We think that the deep learning planners that will achieve the most success in robotics are those that are combined with more-classical algorithms. The computer Go programs use a combination of MCTS and neural networks trained from human games (Silver et al., 2016, 2017); Dec-MCTS could potentially be extended in a similar way.

Our robotic systems for the foreseeable future will have some form of human-in-the-loop. The humans may be assisting the robots complete tasks, or provide instructions to robots. This human can be thought of as another agent in this decentralised system of agents. Dec-MCTS could work where humans communicate information to and from the robots in the same way that robots already communicate their intentions to other robots. A key challenge here would be translating between human-interpretable messages and the probability distributions of Dec-MCTS.

**Communication scheduling**

Our approach to communication scheduling during decentralised planning (Section 3.7) could be extended in several ways. It would be interesting to consider cases where we have models for predicting network integrity to more effectively
use available resources. Our current approach does not enforce hard constraints on bandwidth, however our method of measuring information value solves a necessary step towards enforcing bandwidth constraints while considering task performance. Our approach could be extended to the constrained bandwidth case by swapping the objectives and constraints; however, the interesting challenge here is to suitably generalise the communication decision making step without incurring the substantial time penalty of considering a full decision tree in this case.

It would also be interesting to consider alternative prediction models for predicting the value of communication messages. For Dec-MCTS, it would be useful to be able to efficiently predict the effect of the periodic domain changes. Our approach could also be adapted to be suitable for other decentralised planning algorithms; the key challenge here would be to develop suitable ways of efficiently performing the communication prediction step.

**SOM**

Our SOM algorithm is designed particularly for environments with Euclidean-distance costs. We are interested in extending the approach for scenarios with obstacles or non-holonomic constraints. Several ideas have been proposed for extending SOM algorithms for such scenarios, typically by combining an SOM approach with other planning algorithms, such as RRT (Faigl, 2016). In non-holonomic scenarios, it would be an interesting challenge to incorporate orientation-dependent observations; a promising approach may be to approximate the problem in a high-dimensional Euclidean space (Kulich et al., 2016). Inter-robot collision avoidance is likely to be challenging to incorporate into an SOM algorithm due to the temporal constraints, but a decoupled approach could be an appropriate solution. However, we note that the SOM approach tends to find solutions where the robots’ paths do not cross; therefore additional collision avoidance planning may not be necessary.

Our formulation is motivated by the fact that the performance of perception algorithms is sensitive to the choice of viewpoints. Viewpoint dependencies can be conveniently expressed in our formulation such that all viewpoints within a region are considered correlated, and partial correlation can be expressed with overlapping
regions. We discussed the relationship between viewpoint dependencies and TSP formulations in Section 2.2.2. While this formulation is generally applicable, it may also be convenient to express dependencies by varying the rewards for each polygon, which may be addressed with a modification to the adaptation procedure (Faigl and Váňa, 2016). Other modifications to the reward function formulation could include extensions for teams of robots with heterogeneous sensing, which could readily be addressed by defining a different set of viewpoint regions and rewards for each robot. Other interesting problem generalisations include time-varying objectives for moving targets (Hönig and Ayanian, 2016), and perception models with probabilistic viewpoint regions (like mission monitoring). Also, while our experiments used a generic perception model to define the regions and rewards, this data processing can be adapted for the perception task at hand, such as by using other formulations for modelling 3D objects and predicting observations (Martens et al., 2017).

Another interesting research direction would be to extend the SOM algorithm for infinite horizon persistent monitoring tasks. The SOM algorithm can already be setup to produce cyclic solution paths, but it would be interesting to consider how to incorporate additional challenges presented by particular persistent monitoring tasks. These may include temporal constraints on the time between consecutive visits to a regions (Alamdari et al., 2014). Additionally, considering other dependency models found in some persistent monitoring formulations (Lan and Schwager, 2016; Yu et al., 2016) would be an interesting challenge. SOMs would be particularly suitable for efficiently adapting the path over multiple cycles as the information changes (Hefferan et al., 2016). SOMs could potentially offer solutions to multi-robot generalisations of single-robot persistent monitoring formulations.

In many practical scenarios, such as farms and warehouses with permanent infrastructure, multi-robot coordination can be performed by a centralised server. However, in other scenarios it is necessary to decentralise the planning efforts and consider communication constraints, which presents new algorithmic and practical challenges. A decentralised version of our SOM algorithm may be formulated by combining decentralised robot-node allocation with single-robot SOMs or small teams of multi-robot SOMs. These two components could interact in a similar way to Dec-MCTS to optimise the joint-action space.
Mission monitoring

Mission monitoring is primarily motivated by AUV operations, but several other real-world scenarios also motivate this problem. These scenarios include: flying robots that must land during observational periods to conserve energy (Brockers et al., 2011), pedestrian tracking for hands-free filming and photography (Naseer et al., 2013; Hönig and Ayanian, 2016), acoustically-covert surveillance for tracking animals (Dunbabin and Tews, 2012), ground-based mobile recharge stations for aerial vehicles (Mathew et al., 2013), aerial robots that must be stationary to achieve accurate measurements of radio-tagged wildlife (Cliff et al., 2015), and underwater robots that need to stop and surface to communicate or observe some phenomenon (D’Este et al., 2015). It would be particularly interesting to apply our proposed algorithms to these other scenarios.

Additionally, it would be interesting to further generalise the observation models. An orientation-dependent observation model, such as a narrow field-of-view camera, can be accommodated by adding a tracker-orientation dimension to the search space. Time-varying observation models and dynamic communication rates (Kassir et al., 2015) can readily be addressed by redefining the observation value as a function of time. Also, other definitions for monitoring effectiveness could be considered, such as worst-case as opposed to expected observation time.

Our mission monitoring formulation makes several assumptions about the problem. Many of these assumptions are motivated by our experiences with real robots. However, our assumption of independent probability distributions for the target locations at different timesteps was primarily introduced for algorithmic convenience. Introducing dependencies would probably not require changing the algorithm for the currently considered objective function that sums expected values. If alternative objectives were considered, such as for improving the worst-case performance, then these dependencies would become important, and would also likely make the problem intractable.

Larger scale operations can feature multiple tracker and target agents, such as when multiple AUVs survey an area while being supervised by multiple surface vessels. We addressed the multi-tracker case in Chapter 6. For multi-target scenarios, a naive formulation that sums all targets’ observation times is tractable, but likely to
give undesirable plans that only follow a single target. An objective function that favours dividing the observation time between the targets, such as a minimax function, would be more beneficial, but is likely to be intractable and approximations should be considered.

Our current mission monitoring formulation considers decoupled planning where the target trajectory is first optimised independently, then the tracker trajectories are optimised with respect to the target. In many scenarios, this is reasonable, and it reflects current AUV operations. However, it would also be interesting to consider the case where both the target and tracker trajectories are optimised jointly. In other words, the target trajectory would be designed to not only meet the mission objectives, but also facilitate good monitoring performance. Our general framework proposed in Chapter 6 could potentially be extended for this case.

7.3.3 Hardware experiments

The empirical results presented in this thesis were from a range of simulated experiments. Simulated experiments provide the benefit of being able to isolate particular components of a robotic system; this allows us to analyse the behaviour and performance of the components that are particularly relevant to our contributions. However, we acknowledge that simulated experiments do not fully capture all of the complexities present in real robotic systems. These complexities may include dealing with system failures, decentralised data fusion, localisation uncertainty, and unreliable communication hardware.

As such, we would like to see our algorithms being demonstrated onboard real robot systems. Scenarios we are currently working on hardware implementations for after writing this thesis include: non-myopic task allocation with time-varying rewards by a team of UAVs (Smith et al., 2018); detecting, segmenting and classifying fruit in trees using RGB-D cameras mounted on robot arms (Sukkar et al., 2018) (generalisation of the system by Ramon Soria et al. (2018)); surveillance and patrolling scenarios for a team of ground vehicles; and mission monitoring trials for collaborating surface and undersea robots (extensions of the seatrials by Best and Anstee (2014)). We would like to see many more of the applications discussed earlier in Section 1.2 be performed.
7.4 Outlook

We feel the timing is now right to be seeing much more widespread use of the active perception methodology onboard robots in the real world. As argued in Chapter 1, society is beginning to recognise and invite the important role robotics can play in a wide range of industries. There has been tremendous recent advances in developing robotic hardware systems that can reliably maneuver through, sense and perceive many environments. These innovations have encouraged the development of new algorithms, both for planning and perception, that can run onboard these robotic systems. These new algorithms, in conjunction with modern communication hardware, increased computing power, and reliable robotics hardware, are enabling us to scale these ideas up for multi-robot systems. Use of the active perception methodology will allow these robots to efficiently collect the perceptual data and information necessary to complete important tasks.

We look forward to seeing autonomous and intelligent teams of robots playing an increasingly important role in a wide range of scenarios, including: increasing the output and efficiency of our farms; providing emergency response to natural and humanitarian disasters; and advancing our scientific knowledge of the depths of our oceans and the far reaches of our solar system.
List of References


Appendix A

Intention inference model for trajectory prediction

In this appendix we present a trajectory prediction algorithm that makes predictions by first inferring the plan of an agent. While this problem, as presented here, is not an active perception problem, it may serve as a dynamics prediction model for problems such as target tracking, or as an inference model for inferring the plans of other agents in a multi-robot team. The model presented here was used as an example prediction model in the mission monitoring experiments described in Section 5.8.2.

A.1 Introduction

A wide variety of application areas such as agriculture, defence and domestic service involve robots that must operate alongside people, animals, human-driven vehicles and other robots. One core capability of these robots is to be able to predict the future trajectories of fellow moving agents (Bandyopadhyay et al., 2012; Aoude et al., 2013; Chiang et al., 2014; Kim et al., 2015; Schreier et al., 2014), at a minimum for implementing successful collision avoidance. Incorrect predictions can result in collisions while imprecise predictions can impact the robot’s ability to satisfy its primary objectives. We are interested in exploiting the context of the application, in the form of the agent’s intent, to improve trajectory prediction.
A.1 Introduction

Models that predict the motion of dynamic agents critically rely on contextual assumptions. Examples of common simple assumptions include near-constant velocity or acceleration and predefined trajectories. However, these simple assumptions can lead to weak predictors in complex environments populated by many obstacles and unknown factors that may influence the agent’s short-term behaviour. More complex contextual assumptions, such as the agent’s intent, allow for reasoning about the interaction between the agent and its environment over a long time period and thus can lead to more robust predictors. We provide an extended discussion and review of these assumptions in Section 2.1.2.

We consider the case where an agent is driven by the high-level intention to move to some unknown goal region within a cluttered environment, and is rational in the sense that it aims to take a short path. This case occurs in many scenarios such as pedestrians walking through a train station or robots delivering packages. The agent’s intention is initially unknown, but can be estimated given previous observations of the agent’s motion and a map of the environment. An example is shown in Figure A.1. The challenge is how to develop a probabilistic formulation of this estimation problem along with computationally efficient solutions.

We approach the trajectory prediction problem using a novel mathematical formulation that first estimates the agent’s intention and then uses the resulting probability distribution to predict the position of the agent in the future. Central to our for-
mulation is a probabilistic dynamics model that is used to compute a probability distribution representing the next position of the agent conditional on the current position and intention. The intention inference phase employs a Bayesian estimation framework that can be computed efficiently. The trajectory prediction phase extrapolates the agent’s position recursively, but is difficult to solve analytically. We therefore propose a probabilistic roadmap discretisation of the environment and a Monte Carlo sampling technique that converges to the true predictive distribution as more sample paths are drawn. We show empirically that the technique converges with few samples and performs efficiently in practice.

We demonstrate the behaviour of our algorithms in the scenario of predicting paths of pedestrians moving through a busy environment, using a real-world dataset with 442 pedestrian trajectories. Results show that our method results in high accuracy and prediction certainty compared to a method that does not consider the intention of the agent. Our results also demonstrate the feasibility of our method for integration with collision avoidance and other types of planning algorithms.

A.2 Problem formulation

We address the problem of predicting the future trajectory of an intelligent agent. We consider the case where the agent has a higher-level intention to move to a goal region of the environment, which is only known to the agent itself. The agent is assumed to take the shortest path to the goal region with some uncertainty while avoiding static obstacles.

More formally, at time $t_i$, we seek to estimate the position $X$ of the agent at time steps $t_{i+1}, t_{i+2}, ..., t_{i+k} := t_i + 1: i+k$. The position estimates are described by the probability distributions $\Pr(X_{i+1}), \Pr(X_{i+2}), ..., \Pr(X_{i+k})$. The estimates are calculated based on the observed sequence of positions of the agent $x_1, x_2, ..., x_i$, and therefore is described by the conditional distribution $\Pr(X_{i+j} \mid X_{1:i} = x_{1:i})$.

The agent moves through a bounded environment $\Psi$ that contains known static obstacles, (e.g., the grey obstacles in Figure A.1). The agent is assumed to navigate around the static obstacles and the set of all feasible positions of the agent is denoted
A.2 Problem formulation

Figure A.2 – Graphical model of the agent’s trajectory. The previous and current positions \((X_1, X_2, \ldots, X_i)\) are fully observable, while the future positions \((X_{i+1}, X_{i+2}, \ldots)\) are to be predicted. The trajectory is conditional on the intention of the agent which is described by the latent variable \(\theta\).

For the sake of notation we assume the space \(\mathcal{X}\) has been sufficiently discretised. The speed of the agent at time \(t_i\) is assumed to be drawn from a normal distribution with known parameters \(|\dot{X}_i| \sim \mathcal{N}(\mu, \sigma^2)\). It is assumed that the agent is holonomic, however the proposed approach can be generalised for motion models of non-holonomic agents.

A.2.1 Intention of the agent

The agent’s trajectory is conditional on the intention of the agent which is only known to the agent itself. We describe this intention by the latent variable \(\theta\), and the trajectory of the agent is conditional on \(\theta\) as depicted in Figure A.2. If \(\theta\) is known precisely then this information can be used to give a more accurate estimate of the trajectory.

We assume that the agent’s intention is to move along a path to a particular region \(\theta\) within the environment. The goal region \(\theta\) is one of finitely many predefined regions \(\theta_1, \theta_2, \ldots \in \Theta\), such that each \(\theta_\eta \subset \mathcal{X}\).

A.2.2 Probabilistic dynamics model

We assume that the agent will most likely take the shortest path to the goal region, with some uncertainty as described by the following transition model. The shortest path is calculated by taking into account the set of known static obstacles. We use \(\delta(a, b)\) to denote the distance of the shortest path from position \(a\) to position or region...
\( \delta(x_i, X_{i+1}) \)

\( \delta(X_{i+1}, \theta) \)

\( \delta(x_i, \theta) \)

\( \delta(x_i, X_{i+1}, \theta) \)

Figure A.3 – An illustration of the probabilistic dynamics model. The next position \( X_{i+1} \) along the trajectory relative to the shortest path (blue) from the current position \( x_i \) to a known goal region \( \theta \). The new path distance is \( \delta(x_i, X_{i+1}, \theta) = \delta(x_i, X_{i+1}) + \delta(X_{i+1}, \theta) \).

We define the probability distribution for the position at the next timestep as being exponential in the negative of the increase in the shortest path distance to a given goal \( \theta \): 

\[
\Pr(X_{i+1} \mid X_i = x_i, \theta = \theta) := K^{-1} \exp \left[ -\alpha \left( \delta(x_i, X_{i+1}, \theta) - \delta(x_i, \theta) \right) \right]. \tag{A.1}
\]

The normalising constant \( K \) is defined as

\[
K = \sum_{x_{i+1} \in \chi^+} \Pr(X_{i+1} = x_{i+1} \mid X_i = x_i, \theta = \theta), \tag{A.2}
\]

where \( \chi^+ \subset \chi \) is the set of all \( x_{i+1} \) that can be reached from \( x_i \) in one time step.

We define the distance of the shortest path from \( a \) to \( c \) that passes through \( b \) as 

\[
\delta(a, b, c) = \delta(a, b) + \delta(b, c)
\]

Similarly, \( \delta(a, b, c) = \delta(a, b) + \delta(b, c) \) denotes the distance of the shortest path from \( a \) to \( c \) that passes through \( b \).

This proposed model specifies that, for a given goal, movements that lead to shorter paths are more likely, while movements that result in longer paths are less likely. We wish to emphasise that this intuition is for a given goal and therefore it does not follow that the agent’s intention is necessarily more likely to be a closer goal region than a further region.

The parameter \( \alpha \) describes how likely the agent is to take the shortest path to the goal region. The value is constrained to \( \alpha > 0 \). As \( \alpha \to \infty \), the agent will almost certainly take the shortest possible path to the goal. Conversely, as \( \alpha \to 0 \), all possible paths to the goal are equally likely. The most appropriate value of \( \alpha \) may be selected based
on training trajectories or vary according to other factors in the environment, such as the presence of other agents. We assume $\alpha$ is known and fixed, but we leave this open to future work.

## A.3 Bayesian trajectory prediction

We formulate a Bayesian estimation framework based on the graphical model in Figure A.2. At each timestep we first update an estimate of the intention $\theta$ and then use this to update the predicted future trajectory of the agent.

### A.3.1 Joint distribution

The probability distribution for the estimate of position $X_{i+1}$ at time $t_i$ is conditional on the previously observed position $X_i = x_i$ and the intention $\theta$ of the agent. From Figure A.2, the joint distribution of the probabilistic model is given as

$$
\Pr(X_{1:i}, X_{i+1:i+k}, \theta) = \Pr(\theta) \prod_{j=1}^{i+k-1} \Pr(X_{j+1} | X_j, \theta). \tag{A.3}
$$

### A.3.2 Intention inference

The probability distribution for the estimate of $\theta$ given the observed trajectory $x_{1:i}$ up to time $t_i$ can be calculated using Bayes’ theorem and applying the Markov assumption of the graphical model, such that the posterior is given by

$$
\Pr(\theta \mid X_{1:i} = x_{1:i}) \propto \Pr(X_i = x_i \mid X_{i-1} = x_{i-1}, \theta) \Pr(\theta \mid X_{1:i-1} = x_{1:i-1}), \tag{A.4}
$$

with a uniform initial distribution $\Pr(\theta \mid X_1 = x_1)$. The first factor in the right hand side of (A.4) is the likelihood of an observation and can be computed directly from the transition model (A.1). The second factor is the prior which is recursively updated as the previous posterior.
### A.3.3 Trajectory prediction

The future trajectory $X_{i+1:i+k}$ of the agent is predicted based on the current estimate for $\theta$ (A.4). This is achieved by marginalising the transition model (A.1) over $\theta$, i.e., for 1 timestep into the future:

$$
\Pr(X_{i+1} \mid X_{1:i} = x_{1:i}) = \sum_{\theta_\eta \in \Theta} \Pr(X_{i+1} \mid X_i = x_i, \theta = \theta_\eta) \Pr(\theta = \theta_\eta \mid X_{1:i} = x_{1:i}) .
$$

(A.5)

The first factor in the right hand side can be computed directly from the transition model (A.1) and the second factor is the estimate of the intention (A.4).

This model is recursively extrapolated $j$ timesteps into the future by marginalising over all possible unobserved trajectories at each timestep, yielding

$$
\Pr(X_{i+j+1} \mid X_{1:i} = x_{1:i}) = \sum_{x_{i+j} \in \chi^-} \left[ \Pr(X_{i+j+1} \mid X_{i+j} = x_{i+j}) \Pr(X_{i+j} = x_{i+j} \mid X_{1:i} = x_{1:i}) \right],
$$

(A.6)

where the sum argument $x_{i+j} \in \chi^-$ denotes the set of all positions such that $X_{i+j+1}$ is reachable from $x_{i+j}$ in one time step. An analytical evaluation of (A.6) is difficult due to the exponential branching factor $|\chi|$ at each timestep. Therefore in the following section we propose a sampling technique which iteratively converges to the true distribution.

### A.4 Sampling-based algorithm

In this section we propose an efficient algorithm that outputs probability distributions for the intention estimates and trajectory predictions based on the theoretical solution developed in the previous section. The algorithm begin by discretising the environment and the set of intentions using a graph representation. Prediction is performed after each position observation $X_i = x_i$ in two phases: (1) update the estimate of the intention $\Pr(\theta \mid X_{1:i} = x_{1:i})$, and (2) perform trajectory prediction
Algorithm A.1 Trajectory prediction algorithm.

1: \(\triangleright\) Precomputation
2: \([V, E] \leftarrow \text{GENERATE GRAPH } (\Psi)\)
3: \(\theta_\eta' \leftarrow \{v_i : v_i \in (\theta_\eta \cap V), \forall \theta_\eta \in \Theta\}\)
4: \(\{\delta(v_i, v_j)\} \leftarrow \text{FLOYD-WARSHALL}(V, E)\)
5: \(\delta(v_i, \theta_\eta') \leftarrow \text{SHORTESTDISTANCE}(v_i, \{\delta(v_i, v_j)\}, \theta_\eta'), \forall v_i \in V, \theta_\eta' \in \Theta'\)

6: \(\triangleright\) At each timestep
7: for each \(t_i\) do
8: Observe \(X_i = v_i\)

9: \(\triangleright\) Intention Inference
10: for each \(\theta_\eta \in \Theta\) do
11: \(\text{Pr}(X_i = v_i \mid X_{i-1} = v_{i-1}, \theta' = \theta_\eta') \leftarrow \text{(A.1)}\)
12: \(\text{Pr}(\theta = \theta_\eta \mid X_{1:i} = v_{1:i}) \leftarrow \text{(A.4)}\)

13: \(\triangleright\) Trajectory Prediction
14: \(\text{count}(\hat{x}, j) \leftarrow 0, \forall \hat{x} \in \hat{X}, j \in \{i + 1, ..., \text{maxTime}\}\)
15: for each \(\theta_\eta \in \Theta\) do

16: \(\triangleright\) Draw \(N\) sample trajectories
17: \(N_\eta \leftarrow \text{Pr}(\theta = \theta_\eta \mid X_{1:i} = v_{1:i}) \times N\)
18: for each \(n = 1 : N_\eta\) do
19: \(v_{i:b} \leftarrow \text{DRAWSAMPLEPATH}(v_i, \theta_\eta')\)
20: \(x_{i+1:i+k} \leftarrow \text{INTERPOLATE}(v_{i:b})\)

21: \(\triangleright\) Bin the samples at each time step
22: for each \(j = 1 : k\) do
23: \(\hat{x} \leftarrow \text{ROUND}(x_{i+j})\)
24: \(\text{count}(\hat{x}, j) \leftarrow \text{count}(\hat{x}, j) + 1\)
25: \(\text{Pr}(X_j = \hat{x} \mid X_{1:i} = v_{1:i}) \leftarrow \frac{\text{count}(\hat{x}, j)}{N}, \forall \hat{x} \in \hat{X}, j \in \{i + 1, ..., \text{maxTime}\}\)

by evaluating the distribution \(\text{Pr}(X_{i+j+1} \mid X_{1:i} = x_{1:i})\) using Monte Carlo sampling. Pseudocode for the algorithm is shown in Algorithm A.1.

A.4.1 Precomputation

Discrete roadmap (line 2)

For non-trivial problems, the theoretical equations are computable only when time is discretised and there exists a finite set of possible transitions at each timestep. We therefore discretise the environment \(\Psi\) into a discrete set of feasible positions.
\( v_i \in \mathcal{V} \subseteq \mathcal{X} \) of the agent. Additionally, there is a finite set of feasible transitions 
\( e_\eta = \langle v_i, v_j \rangle \in \mathcal{E} \) that describes moving from the position \( v_i \) to the position \( v_j \) in one timestep. The sets \( (\mathcal{V}, \mathcal{E}) \) describe the vertices and edges of a graph that gives a discrete roadmap representation of the environment. The graph should be formed such that any path through the environment can be sufficiently approximated by a path through the graph. For the examples used in Section A.5 we utilise a probabilistic roadmap (PRM) formulation, however other roadmaps may also be considered.

**Set of intentions (line 3)**

The problem specification assumes there is a predefined finite set of possible goal regions \( \theta_\eta \in \Theta \). In the graph representation, each region is represented by a set of vertices
\[
\theta'_\eta := \{ v_i : v_i \in (\theta_\eta \cap \mathcal{V}) \}.
\] (A.7)

A trajectory to a goal region \( \theta_\eta \) is represented as a path through the graph \( (\mathcal{V}, \mathcal{E}) \) that ends at any \( v_i \in \theta'_\eta \).

**Shortest-path distances (lines 4–5)**

The intention inference and the sampling both require repeatedly evaluating the shortest path distances from various vertices to all goals. It is therefore worth precomputing a database of shortest path distances from every vertex \( v_i \in \mathcal{V} \) to every goal \( \theta_\eta \in \Theta \). This can be achieved efficiently using the Floyd-Warshall algorithm (Floyd, 1962). The distance \( \delta(v_i, \theta_\eta) \) is evaluated as the distance to the closest \( v_j \in \theta'_\eta \) (line 5).

**A.4.2 Intention estimates (lines 10–12)**

At every time step, the position of the agent is observed \( X_i = x_i \) and the current vertex \( v_i \) is chosen as the \( v_i \in \mathcal{V} \) which has the closest distance to \( x_i \) (line 8). This observation is used to update the intention estimate by directly evaluating and normalising (A.1) and (A.4) for each \( \theta_\eta \in \Theta \).
A.4 Sampling-based algorithm

A.4.3 Monte Carlo trajectory prediction (lines 14–25)

The general solution for the trajectory prediction (A.6) is difficult to evaluate due to the branching factor \(|\chi^+|\) of possible transitions at every step into the future. Therefore we propose a Monte Carlo sampling technique that converges to the true distribution as more samples are drawn.

Cells (line 14)

Firstly, the space \(\mathcal{X}\) is partitioned into small cells \(\hat{x} \in \hat{\mathcal{X}}\). The probability distribution \(\Pr(X_j \mid X_{1:i} = x_{1:i})\) for a time \(t_j\) in the future will be estimated over these cells \(\Pr(X_j \in \hat{x} \mid X_{1:i} = v_{1:i}), \forall \hat{x} \in \hat{\mathcal{X}}\). The partitioning structure and resolution may be chosen appropriately based on the application. The examples use a uniformly-spaced grid covering the environment.

Draw sample paths (lines 15–24)

The sampling proceeds by drawing \(N\) sample trajectories from \(v_i\) to a goal \(\theta_\eta\). The number of samples \(N_\eta\) allocated to each \(\theta_\eta\) is proportional to the posterior probability of \(\theta = \theta_\eta\) (line 17). Note that each sample may be computed in parallel.

A sample path from \(v_i\) to \(\theta'_\eta\) is drawn by recursively following edges \(\langle v_i, v_j \rangle\), where each \(v_j\) is drawn from the probabilistic transition model distribution (A.1), given a fixed goal region \(\theta'_\eta\) (line 19). In other words, sample paths are random walks over the graph that bias towards shorter paths to the goal region \(\theta'_\eta\). Given that the transition model specifies that shorter paths are more likely (for a given goal region), the recursion should always quickly reach \(\theta'_\eta\). However, an upper bound \(|v_{i:b}| < U\) on the number of iterations before halting a path may be required for some environments, and is reasonable since most applications are only interested in predicting up to some fixed time horizon.

Each sample path \(v_{i:b}\) is mapped to a sample trajectory defined as a set of positions \(x_{i+1:i+k}\) at evenly spaced timesteps (line 20). This interpolation takes into account the probabilistic speed model by drawing from the velocity distribution distribution...
given in the problem specification. It is assumed that after the agent reaches $\theta_\eta$ at time $t_{i+k}$ it is no longer relevant, for example because it then moves beyond the boundary of the environment.

**Trajectory prediction distributions (lines 22–25)**

For every $x_j \in x_{i+1;i+k}$ of a sample trajectory, the algorithm finds the cell $\hat{x}$ where $x_j \in \hat{x}$. A counter corresponding to the cell $\hat{x}$ and time $t_j$ is incremented by 1 (line 24). Each counter represents the number of times that cell $\hat{x}$ is visited at time $t_j$ by the $N$ sample trajectories. The probability distribution at each time $t_j$ in the future is therefore evaluated by dividing by the total number of samples (line 25).

**A.4.4 Analysis**

The estimated trajectory prediction distribution will converge towards the true distribution (A.6) as the number of vertices in the PRM increases and as the number of sample paths drawn for the sampling method increases. We empirically show convergence of the distribution as the number of samples increases in the experimental section.

The worst-case time complexity for the offline precomputation is cubic in the number of vertices of the PRM and the online trajectory prediction at each timestep is linear in the number of samples. More formally, we denote $B$ as the maximum number of edges out of any vertex and $R$ as the interpolation rate of the sample paths. The offline precomputation phase is dominated by building the spanning trees from all vertices which has complexity $O(|V|^3)$. Computation after each observation has complexity for: updating the posterior of the intention inference $O(\mathcal{B}\Theta)$, evaluating $N$ sample paths $O(N(BU + RU))$, evaluating the resulting prediction distributions $O(|\hat{\mathcal{X}}|)$ and therefore a total complexity of $O(\mathcal{B}\Theta + N(BU + RU) + |\hat{\mathcal{X}}|)$, which in most cases will be dominated by $O(NBU)$ since $|\Theta|$ and $|\hat{\mathcal{X}}|$ are typically smaller than $N$. 
A.5 Experiments

In this section we present experimental results to give an intuitive understanding of the algorithm, show its feasibility in practice for predicting human movements using a real-world pedestrian dataset and highlight advantages of the proposed intention inference concept over predictions that do not consider the agent’s intention.

A.5.1 Simulated environment

Figure A.4 shows a simulated agent moving from top left to middle right through an environment with 16 goal regions around the boundary. Figure A.4 shows the distribution of the trajectory prediction accumulated over all future timesteps, while Figure A.5 demonstrates how the algorithm gives probability distributions for individual timesteps in the future. The information shown in these figures can be used
by another agent to avoid collisions with the dynamic agent who’s trajectory is being predicted. The other agent should avoid the blue areas at particular times since otherwise there is some probability that a collision would occur.

We analyse the behaviour of the algorithm in this example as follows. (a) At time 1, the goal regions have a uniform prior and therefore the trajectory prediction almost covers the entire environment. (b) By time 4, the agent has started moving towards the right rather than downwards, and therefore the goal regions on the right side of the map now have a larger posterior probability and the trajectory has a higher density in this direction. (c) At time 7, the goal regions at the top and right of the map have a similar posterior probability, but the following movement (d) suggests that the top regions are unlikely. (e) The downward movement towards the centre suggests the agent is moving towards the bottom right, however it is not yet clear which direction it will move around the bottom right obstacle. (f)–(h) As the agent moves closer to the actual goal region, the predicted trajectory converges towards the actual trajectory. This increased certainty is due to the increasing posterior intention probability for the actual goal region. (h) At the end of the actual path, there is still
some probability that the agent will continue moving downwards to the bottom right region.

Figure A.5 shows the predicted position distribution at individual timesteps in the future from the same observed position as Figure A.4d. These figures show how the uncertainty in position increases when extrapolating to timesteps further ahead in the future. For the purpose of dynamic-obstacle collision avoidance, the predictions presented in this form shows how the algorithm can be used to tell another agent the probability of a collision at a particular time and position.

In this example the environment was represented using a 1,000 vertices PRM with edges between all pairs of vertices that are straight line visible and have a distance less than one-tenth the width of the environment. The trajectory prediction used $\hat{X}$ chosen as a grid of $20 \times 20$ cells. Each prediction step used 1,000 sample paths, which takes approximately 500 ms using un-optimised code and performed in parallel on a standard 4-core desktop processor. The distribution converges as more samples are taken, however with 1,000 samples the distribution has a mean squared error of $10^{-5}$ relative to a 100,000 sample distribution estimate for this example.

### A.5.2 Real-world pedestrian dataset

We show the feasibility and advantages of the proposed trajectory prediction using intention inference by performing simulated experiments with 442 pedestrian trajectories from a real-world dataset by Lerner et al. (2007)\(^1\). The pedestrians walk through the environment shown in Figure A.6a and the ground-truth trajectories were hand-labelled from a video. The simulated environment is shown in Figure A.6b with 50 example pedestrian trajectories and 12 selected goal regions around the boundary. Each pedestrian either walked along the footpath through the middle, crossed the road at the bottom boundary while avoiding the parked car, entered or exited the shop doorway in the top left, or walked down the driveways in the top right.

The accuracy of trajectory predictions averaged over all pedestrians is shown in Figure A.7a. For timesteps near in the future, the algorithm most often gives a relatively high probability to the cell of the ground-truth future position. The probability

\(^1\)Dataset published at: [graphics.cs.ucy.ac.cy/research/downloads/crowd-data](graphics.cs.ucy.ac.cy/research/downloads/crowd-data)
A.5 Experiments

(a) Outdoor environment.

(b) 50 example pedestrian trajectories and 12 goal regions around the boundary.

Figure A.6 – The dataset of pedestrian trajectories (Lerner et al., 2007).

(a) Accuracy of the trajectory predictions. Vertical axis is the fraction of pedestrians that have a high correct-prediction probability.

(b) Uncertainty of the trajectory predictions. Vertical axis is the entropy of the prediction distributions.

Figure A.7 – Comparison of predicting with and without taking into account the intention estimate posterior. Predictions made after 10 observed timesteps.

threshold is set to $p(\hat{x}^*) > 0.05$, which may represent a safety threshold for a collision avoidance planner. The accuracy drops when looking further in the future since the uncertainty grows when extrapolating to future timesteps.

Figure A.7a also shows that trajectory prediction with the posterior intention estimate outperforms predictions with a uniform intention estimate; a high accuracy is maintained for many more timesteps into the future. Intuitively, this is because the uniform case can only extrapolate outwards from the current position whereas the
intention inference case directs predictions towards the estimated goal regions.

Figure A.7b shows the uncertainty of the prediction distributions when using the posterior intention estimate results in distributions with lower entropy than the uniform case. This is important when applied to dynamic obstacle collision avoidance since a probabilistic planner has fewer cells that it must avoid and therefore has more freedom to find improved collision-free paths that satisfy other objectives.

A.6 Conclusions

In this appendix we have proposed a Bayesian framework for predicting the future trajectory of an agent by estimating its intention to move to a goal region in the environment, and have presented a computationally efficient solution. Some ideas for future work would be to explore other types of intentions and objective functions, model interaction with other agents, and consider observation uncertainty. We demonstrated the usefulness of this model for planning in the context of mission monitoring in Section 5.8.