Routing for Persistent Exploration in Dynamic Environments with Teams of Energy-Constrained Robots

Derek Mitchell

Sept 14, 2020

CMU-RI-TR-20-52

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

Thesis Committee:
Nathan Michael, CMU, Chair
Katia Sycara, CMU
Maxim Likhachev, CMU
Stephen L. Smith, University of Waterloo

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Robotics.

Abstract

Disaster relief scenarios require rapid and persistent situational awareness to inform first-responders of safe and viable routes through a constantly shifting environment. Knowing what roads have become flooded or are suddenly obstructed by debris can significantly improve response time and ease the distribution of resources. In a sufficiently large environment, deploying and maintaining fixed camera stands would be ineffective and prohibitively expensive, so we look to deploying teams of robots equipped with sensors to persistently cover the region of interest. In this case, the main challenge is determining how to distribute the robots to cover the entire region with limited travel speed and duration.

Basic multi-robot coverage and exploration methods take a passive approach that direct robots to evenly cover the space and populate a map of the environment with the observations the robots acquire as they move. More reactive frontier-based approaches will continuously guide robots towards unobserved regions until the environment is fully known. These approaches, however, are less effective when the environment changes over time. When the number of robots is limited and can only operate for finite durations, the planner must prioritize which regions to visit in order to provide the most accurate map possible.

In this thesis, we propose to plan deployments of teams of quadrotors equipped with range sensors to cooperatively cover an environment such that a map can be persistently updated as the environment topography evolves. Here, we present a systems-based approach that breaks up planning into stages and computes feasible plans over a sliding-window horizon. These plans are extended over subsequent horizons up to the limit that each robot's battery capacity will allow, ending with robots returning to home base and recharging. We initially show that the proposed planner is able to outperform greedy frontier assignment in terms of map accuracy and confidence. We then show how the objective function we initially used to distribute robots can be modified to incorporate the 'goodness-of-fit' of the environment dynamics model by biasing robots towards regions whose dynamics are less understood. The updated objective results in a system that quickly converges to robots revisiting regions only as often as they are expected to change.

While there is a physical limitation to how much area a team of energy-constrained robots can cover persistently, the system we present leverages an understanding of environment dynamics to maximize model improvement over time in a computationally tractable fashion. This is shown with a comprehensive study of the computational complexity of each component of the proposed system and an evaluation of how overall performance evolves over time relative to the choice of system parameters and environment conditions. We additionally show that the system can be tuned to better address environments experiencing changes of greatly varying magnitudes and scale. The result is a complete system that enables perpetual deployment and efficient distribution of robots throughout the environment to ensure no changes go unobserved for too long.

Acknowledgments

I can confidently say I could not have come this far without the support of my mentors over the years. I have been extremely fortunate to have constant access to teachers, supervisors, colleagues, and advisors that see potential in me and provide ample support to help draw it out. In particular I am most grateful to my advisor, Nathan Michael, who not only provided the opportunities, resources, and advice I needed to polish my skills and delve deeply into engaging research, but also gave me many chances to put my efforts into public view. His constant drive to push the boundaries of the state of the art in a manner that is visible and accessible to those it may interest has had significant impact in shaping my approach to research. I would also like to thank my thesis committee members, Katia Sycara, Maxim Likhachev, and Stephen Smith, who have been very helpful and accommodating throughout this process, with special thanks to Katia who helped co-advise me during my time as a Masters student.

Next, I would like to thank all my friends and colleagues at the Resilient Intelligent Systems Lab, whose care and advice has made this long journey enjoyable and rewarding. Foremost, I would like to thank my office mates, Ellen Cappo, Micah Corah, and the fish for their constant care, advice, and company (especially the fish). I have also received ample research advice and help managing robots from Arjav Desai, Matt Collins and Curtis Boirum, without whom much of my more interesting demonstrations would never have come to fruition. Many others in the lab, past and present, have helped me to polish my research and presentations through discussion and peer review, sacrificing their time and attention to help better my work. In particular, I offer my gratitude to Wennie Tabib, Kumar Shaurya Shankar, John Yao, Xuning Yang, Cormac O'Meadhra, Aditya Dhawale, Alex Spitzer, Lauren Lieu, Mosam Dabhi, Vishnu R. Desaraju, Vibhav Ganesh, Erik Nelson and Shihyun Lo.

Outside of the lab, I want to give special thanks Allie Del Giorno, whose friendship over the last seven years has helped me maintain some level of sanity and kept me from disappearing into the solitude of my apartment. I also have to offer my gratitude to Chuck Whittaker and James Teza, who worked directly with me to build many of the complex charging interfaces I needed to test my long duration deployments, as well as Karen Widmaier, Ashley McClinton, Nora Kazour, and Suzanne Lyons Muth who have provided ample administrative support to help me navigate the Ph.D. process smoothly.

Finally, I would like to thank my friends and family back home, who have been extremely patient and allowed me to disappear into my work for the past half a decade or so. While I appreciate the constant support and encouragement, the greatest help has been their forgiveness for missing the occasional holiday. Knowing that I can pour all my effort into my work when deadlines loom overhead, certain in the knowledge that my family is supporting me from home and will be there when I need them, has given me the strength and freedom to tackle challenges however suits me best.

Contents

1 Introduction **1**
- 1.1 Core Challenges . . . 4
- 1.2 Summary of Contributions . . . 5

2 Background **7**
- 2.1 Environment Mapping . . . 9
  - 2.1.1 Dynamic Environments . . . 11
  - 2.1.2 Active Perception . . . 12
- 2.2 Motion Planning . . . 13
- 2.3 Long-Duration Autonomy . . . 16
  - 2.3.1 Capacitated Routing . . . 17

3 Persistent Multi-Robot Mapping in an Uncertain Environment **23**
- 3.1 Problem Formulation . . . 23
- 3.2 Methodology . . . 25
- 3.3 Simulation Evaluation . . . 38
- 3.4 Conclusion . . . 43

4 Allocating Limited Sensing Resources to Accurately Map Dynamic Environments **44**
- 4.1 Problem Definition . . . 44
- 4.2 Methodology . . . 47
  - 4.2.1 HMM Occupancy Grid . . . 47
  - 4.2.2 Observation Utility Measure . . . 49
- 4.3 Results . . . 51
  - 4.3.1 Experiment Setup . . . 51
  - 4.3.2 Comparison of Objective Functions . . . 52
  - 4.3.3 Varying Parameters . . . 55
  - 4.3.4 Varying Environments . . . 57
- 4.4 Discussion . . . 58
- 4.5 Conclusion . . . 58

5 Deployment Planning for Online Mapping of Dynamic Environments **59**
- 5.1 Environment Model . . . 62
  - 5.1.1 Computational Complexity . . . 63

5.2 Sensor Model . . . 63
5.2.1 Computational Complexity . . . 65
5.2.2 Observation Utility . . . 65
5.2.3 Computational Complexity - Utility Function . . . 67
5.3 Waypoint Selection . . . 68
5.3.1 Computational Complexity . . . 70
5.4 Path Planning . . . 71
5.4.1 Trajectory Generation . . . 72
5.4.2 Computational Complexity . . . 73
5.5 Waypoint Assignment . . . 74
5.5.1 Greedy Assignment . . . 74
5.5.2 Minimum Cost VRP . . . 76
5.5.3 Min-Makespan VRP . . . 77
5.5.4 Ensuring Feasible Configurations . . . 78
5.5.5 Computational Complexity . . . 80
5.6 Conclusion . . . 81

6 Experimental Evaluation in Simulated Environments 82
6.1 Setup Details . . . 83
6.2 Approach Comparison . . . 85
6.3 Parameter Evaluation . . . 91
6.4 Physical Limits . . . 99
6.5 Environment Conditions . . . 106
6.6 Urban Environment . . . 110
6.7 Conclusion . . . 113

7 Conclusion 117
7.1 Summary of Contributions . . . 118
7.2 Future Work . . . 119

A Online Energy-Constrained Adaptation and Scheduling of Persistent Coordinated Behavior-Based Multi-Robot Deployments 123
A.1 System Overview . . . 124
A.2 Experimental Evaluation . . . 136
A.3 Conclusion . . . 141

Bibliography 142

List of Figures

1.1 Images showing the devastation that can be caused by natural disaster. 2

2.1 Visualization of the intersection of related fields to which the proposed work belongs. 8
2.2 Three distinct models of environment topography. 9
2.3 Enforcing long duration operation trajectory swapping. 17
2.4 Graphical representation of a Vehicle Routing Problem. 20

3.1 Deployment planner system diagram 24
3.2 The planning pipeline expressed in graph form 26
3.3 The observation model 27
3.4 Confidence decay for the log-odds and probability representation of occupancy likelihood 29
3.5 *Waypoint Selection* uses the *Sensor Model* to determine the optimal observation locations 30
3.6 Representation of the simulated environment populated by static and dynamic objects 34
3.7 Evaluation of system performance under various conditions 36
3.8 Evaluation of dynamics modeling and energy expenditure of a sample run 42
3.9 Evaluation for an environment with objects that move at a random frequency . . . 42

4.1 This work explores an example scenario where N_c cells are directly observed by N_r sensors and $N_r < N_c$. 46
4.2 A time series comparison of binary-state models given sparse observations. . . . 48
4.3 The set of test environments. 51
4.4 Comparison of allocation objectives. 52
4.5 Evolution of priority as the environment model is learned when allocating 150 observations per time-step according to the proposed objective. 55

5.1 Updated System Diagram 60
5.2 The variables associated with the *Sensor Model*. 64
5.3 Planning grid visualization 70
5.4 Motion primitive library 73

6.1 We evaluate on randomly generated environments, where cells oscillate between states at a regular frequency. 83

6.2 Diagram showing the differences between evaluated approaches. 86
6.3 Comparison of approaches evaluated in the environment from Fig. 6.1a. 88
6.4 Vignette of the environment models for simulated runs for the Min-Makespan and pure greedy approaches. 89
6.5 Comparison of computation times for the various approaches. 91
6.6 Performance analysis relative to varying numbers of waypoints processed per horizon. 93
6.7 Comparison of the numbers of waypoints assigned to deployed robots. 94
6.8 Performance analysis relative to the maximum number of robots active at any given time. 95
6.9 Performance analysis relative to the horizon advance duration 97
6.10 Performance analysis relative to the horizon window duration 98
6.11 Performance analysis relative to the mixed objective weight. 100
6.12 Performance analysis relative to energy capacity 102
6.13 Performance analysis relative to travel speed. 103
6.14 Performance when varying the maximum sensor range. 105
6.15 Performance when varying the span of a sensor sweep action. 106
6.16 Performance when varying the number of beams per sensor sweep. 107
6.17 Testing environments differentiated by the percentage of dynamic cells outside of the safety region. 109
6.18 Comparison of varying environment dynamics and distributions. 110
6.19 Urban test environment. 111
6.20 Urban scenario vignette 113
6.21 Urban environment dynamics. 114
6.22 Comparison of performance between the system-based and pure greedy approaches in the urban flooding scenario. 115
6.23 A comparison of environment models at key time frames for the Min-Makespan, system-based approach and the pure greedy assignment approach. 116

7.1 Visual representation of multiple charging areas. 122

A.1 System architecture for persistent formation flight. 124
A.2 Updating the set of energy feasible behaviors for a 30 robot team 126
A.3 Three voltage traces from three different quadrotors during flight. 130
A.4 Trajectory generation process. 131
A.5 Representative example of an exchange operation. 135
A.6 Images of robot behaviors in simulation and in hardware. 136
A.7 Parameter learning from simulated voltage data. 137
A.8 Robot trajectories as a function of distance from origin. 137
A.9 Parameter learning from live trials. 140

List of Tables

3.1 Evaluation of performance over various parameters 40

4.1 Comparison of approaches for runs in the 30% dynamic environment for 10,000 seconds. 54
4.2 Evaluation of performance as a function of parameter choice. 56
4.3 Comparison of performance for different concentrations of dynamics. 57

6.1 System used in the simulation experiments and their default values. 85
6.2 Physical limits used in the simulation experiments and their default values. 85
6.3 Comparison of approaches. 91
6.4 Evaluation of system performance while varying N_w and N_r. 96
6.5 Evaluation of performance as horizon variables vary over several runs. 99
6.6 Evaluation of performance as the objective weight α varies over several runs. . . 101
6.7 Comparison of steady-state performance while varying capacity L and travel speed v_{travel}. 104
6.8 Steady-state performance when varying sensor parameters. 108

Chapter 1

Introduction

Autonomous exploration is a well studied problem with many applications in domains such as search-and-rescue, surveillance, and mapping. The exploration problem involves deploying one or more robots to build an accurate model of an unknown environment. In dynamic environments, the unknown regions can grow over time as the environment evolves without being observed. This is of particular relevance to disaster response, where a persistent understanding of the environment is necessary to avoid dangerous regions and preserve infrastructure. An example of one of the major threats to infrastructure is flooding (Fig. 1.1), as roads needed to transport supplies are initially blocked or destroyed. However, once the water ebbs and rescue teams effect repairs, the roads become passable again and the environment model must be updated. Areas exhibiting potential threats, such as fire or shifting rubble, must be constantly monitored in case the danger spreads. Efficient coverage in such domains requires a reactive and online robot deployment strategy that is robust to changes in the environment despite the limited number of robots available to deploy.

The obvious solution for achieving coverage is to evenly spread robots over a boustrophedon path [1] that serpentines over the whole environment. However, we require a more directed approach when the environment is mostly static, with sparse dynamic regions that evolve independently over time. When the consequences of recognizing that a road has been made impassible

(a) Flooding in Wales, 2012[1] (b) Hurricane Sandy, 2012[2]

Figure 1.1: Images showing the devastation that can be caused by natural disaster. Note how accessability (a) changes as a result of flooding and (b), reverts to some degree as the flood abates.

are that survivors fail to receive the help they need in time, it is imperative that changes to the environment are recognized quickly. To address these concerns, a planner must satisfy some critical requirements. First, the planner must be able to differentiate between static and dynamic regions, prioritizing visitation of those areas more likely to change. Second, plans must extend far beyond the limited duration a robot is able to operate as changes can occur sporadically over long periods of time. Thus, a robot deployment planner must generate routes through the environment that eventually return to a charging station and account for the expected downtime while the robot is recharging.

Similar planning problems are often addressed in the field of Operations Research [2, 3, 4], where the choice of how to route agents to service a selected set of targets, subject to constraints, can be expressed as a combinatorial optimization problem. This class of problems, which includes the Traveling Salesman Problem (TSP) [5], the Vehicle Routing Problem (VRP) [6], and their many variants, has been the subject of many years of research to find provably optimal minimal cost routes that can be generated quickly for large sets of targets. However, due to the NP-hard [7] complexity of optimal solvers, varying the problem by adding constraints can significantly increase computation time for many targets. Additionally, many works focus on

[1]https://www.bbc.com/news/uk-wales-18378978

[2]https://en.wikipedia.org/wiki/Effects_of_Hurricane_Sandy_in_New_Jersey

finding computationally tractable ways to search for the optimal solution of complex problem configurations, without considering how to choose targets or determine how to travel between them [8, 9].

In the single and multi-robot exploration domain, robots are deployed to collect observations of an environment and generate a model that can be queried for the state of the environment at any location or time. Some exploration approaches focus on the modeling aspect, leveraging the more consistent qualities of the environment to predict how the model should look in unobserved regions [10, 11]. Others focus on efficient deployments by determining which regions of the environment to prioritize for observation [12, 13, 14, 15]. These approaches tend to prefer the myopic strategy of continuously directing robots towards 'unknown' regions where observations collected will best inform the model. However, this manner of myopic approach is less suitable when robots cannot be expected to operate for the full duration of interest and must periodically return to recharge. Instead, a planner must be able to balance the potential reward of collecting an informative observation against the cost of traveling too far from the charging station.

In this thesis, we present a systems-based approach to generate plans by combining the routing strategy of traditional combinatorial optimization problem solvers with the identification of informative targets and path generation used in robotic exploration algorithms. We mitigate the additional computational complexity of combining these approaches and improve responsiveness to changes in the environment by computing plans over finite time horizons, replanning as new information is collected. As information is acquired over each new horizon, our planner infers what regions in the environment exhibit more frequent changes and should be given higher priority for future visits. Utilizing this knowledge, the proposed approach deploys robots to observe changes in the environment far more quickly than greedy, myopic approaches. We show that, with the proper choice of planning horizon and other relevant parameters, our strategy can generate plans whose duration is less than the computation time, allowing the proposed planner to operate online. Using the proposed approach, robots are deployed to quickly acquire situational awareness in an unknown and dynamic environment and effectively capture long-scale changes

whose periodicity cannot be captured within a single robot's flight time.

1.1 Core Challenges

The objective of this thesis is to formulate a computationally tractable planning methodology for persistent deployment of teams of energy constrained robots to efficiently map an unknown, dynamic environment. As such, there are three major challenges we address in this work:

- **Responsiveness:** The amount of time that passes after a change in the environment occurs before it is observed must be minimized.
- **Energy Feasibility:** Each single-robot plan cannot expect energy expenditure to exceed battery capacity.
- **Computational Tractability:** Plans must be generated faster than they are executed to ensure continuous, online operation.

When deploying teams of energy constrained robots in unknown, dynamic environments, the immediate priority is to understand how the environment evolves over time. With limited resources, in terms of robot team size and battery capacity, there is a finite rate at which observations can be collected. Without understanding the concentration and distribution of dynamic regions in an environment, the deployment planner will waste time observing regions that are persistently static and be unable to respond to changes elsewhere. This issue is compounded by the need to learn the dynamics model online. Similar to the problem of aliasing in signal-processing, insufficient sampling can result in a misidentification of the inherent environment dynamics. If a region changes at a higher frequency than the model predicts, the planner will not be able to correct the error unless it is capable of recognizing the discrepancy and redistributing robots accordingly. This can have drastic consequences when monitoring a disaster scenario if robots only visit a road that is vital to the distribution of supplies when it is clear and fail to observe it flooding once every six hours for two hours straight. With observations only acquired

when the road is passable, the system would believe that the road is statically clear and direct robots there less frequently, ensuring that any further changes would go unobserved.

1.2 Summary of Contributions

This thesis develops a planning strategy that leverages standard techniques in task allocation and path planning to devise a system of algorithms capable of persistently generating and updating deployment plans for a team of robots operating in a dynamic environment. We present a system structure that interleaves path planning and task assignment over an infinitely repeating sequence of finite time horizons in a computationally tractable manner. This approach is made computationally tractable not only by confining planning to a finite horizon, but also by decoupling the task selection, path planning, and task assignment components of deployment planning as an alternative to planning informative paths in a high-dimensional, multi-robot state-space. Listed below are brief descriptions of subsequent chapters and the contributions they present:

- **Chapter 3 - Persistent Multi-Robot Mapping in an Uncertain Environment**: We introduce the core system framework that decouples path planning and task allocation components to enable tractable reactive planning. We show how a simple, structured planning approach can generate more accurate maps more quickly than sequential greedy assignment. [16]
- **Chapter 4 - Allocating Limited Sensing Resources to Accurately Map Dynamic Environments**: We present an improved objective function for selecting viewing targets by factoring both the known/unknown state of the environment model and the learned environment dynamics. This approach is applied to a Hidden Markov Model (HMM) occupancy grid and shows better performance than prior selection objectives.
- **Chapter 5 - Deployment Planning for Online Mapping of Dynamic Environments**: The system framework presented in Chapter 3 is updated with the new model and utility

score from the Chapter 4 and revised to plan through more waypoints per horizon. A complexity analysis is provided for each component of the system.

- **Chapter 6 - Experimental Evaluation in Simulated Environments**: The system is evaluated over many runs with varying parameter settings and in different environments. We show that the proposed approach significantly outperforms greedy assignment and highlight how performance might be inferred based on the robot parameters and environment conditions.

With the proposed system, we are now able to deploy robots to autonomously explore dynamic environments, prioritizing attention based on the dynamics model learned over many planning horizons. By incorporating energy constraints, we ensure that deployments can persist long enough to capture the periodicity of slowly evolving environment dynamics. As a result, robots quickly learn the dynamics model and converge to deployments that cover regions relative to their frequency of change. We expect future works will build on this approach to assess the time-varying density of environment dynamics from the model, determine the necessary number of robots (demand) for a desired level of model accuracy, and adjust deployment team size online relative to the energy reserves of available robots (supply). As a result, the system will be able to achieve a high level of model accuracy despite the limited availability of robots and a demand that varies over time.

Chapter 2

Background

The multi-robot persistent exploration problem is a complex amalgamation of three major robotics research domains, namely: Environment Mapping 2.1, Motion Planning 2.2, and Long-Duration Autonomy 2.3. Figure 2.1 shows a visualization of these domains and the related topics considered in this work. This thesis shows how to leverage, combine, and build upon techniques from these domains to create a system capable of persistently generating deployment plans for a team of robots to persistently monitor a dynamic environment. We provide the necessary background for understanding these component techniques in this chapter.

To aid in understanding how the component domains interact in this work, we provide a brief description of the multi-robot persistent exploration problem. Robots are continuously deployed from a collective charging area to build and maintain a model of the environment topography. Once the robots are deployed, they must return before completely exhausting their battery supply, completing a route through the environment. At each iteration of a planning cycle, multi-robot routes are generated by extending existing plans or creating new ones. After each planning cycle, robots traverse their routes, collect observations, and update the environment model for a fixed duration before starting the next cycle. The more effective the planning strategy, the faster the model converges to high confidence and accuracy. More details on this formulation are provided in Chapter 3. Specific to this work, we make several simplifying assumptions:

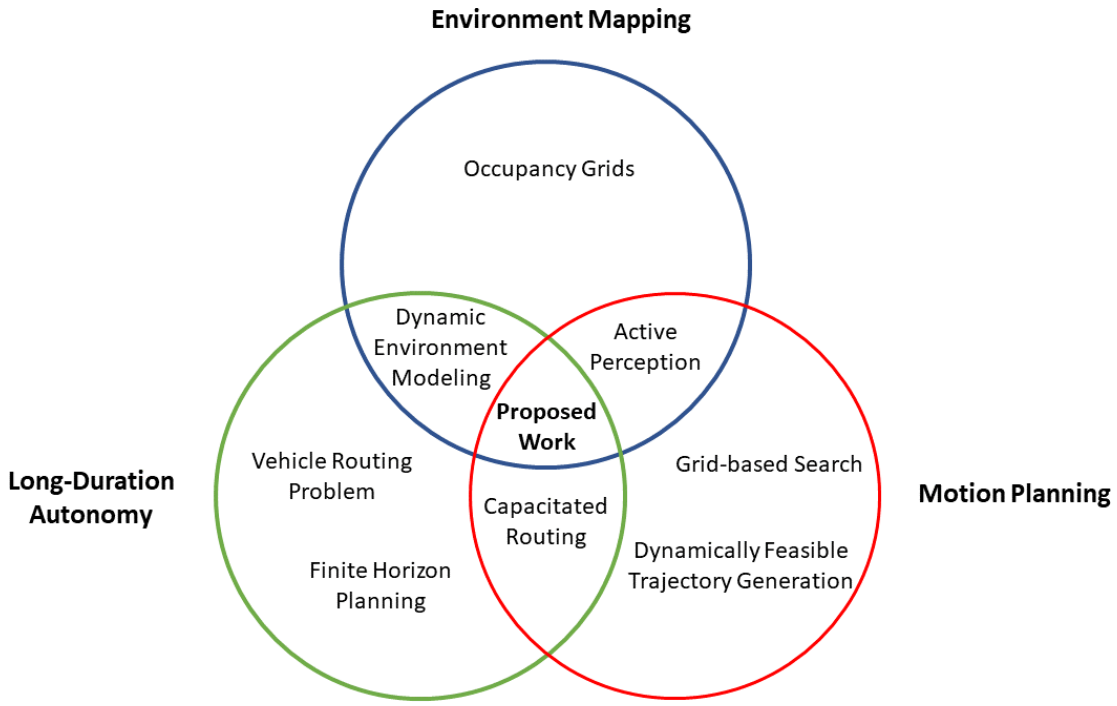
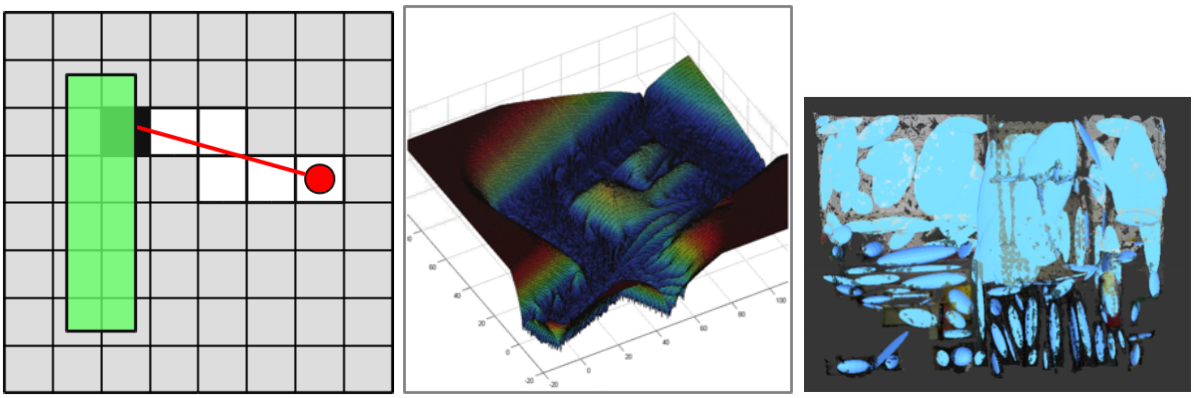


Figure 2.1: Visualization of the intersection of related fields to which the proposed work belongs. The topics listed inside the circles represent domains of particular relevance to the approach described in this thesis.

- **No robot-robot collisions** - The proposed application considers the use of aerial robots in outdoor settings, where we can assume there is sufficient open space to allow for reactive collision avoidance with minimal impact on energy expenditure.
- **Perfect communications** - The proposed approach uses a centralized planning strategy. To accommodate the sharing of sensor input and controls, we assume robots can communicate with the centralized planner instantaneously with zero cost.
- **Perfect localization** - To reduce the complexity of integrating sensor measurements into the environment model, we assume robots have perfect knowledge of their location in the environment.
- **Fixed orientation** - Robots in this work are equipped with downward facing sensors that exhibit rotational symmetry about the vertical axis. We assume robots maintain a fixed orientation to limit the dimensionality of path planning.



(a) Occupancy Grid (b) Gaussian Process [10] (c) Hierarchical GMM [18]

Figure 2.2: Three distinct models of environment topography. (a) The occupancy grid shows the occupancy likelihood via shading (free cells are white, occupied cells are black). (b) A plot of the variance of a Gaussian Process representation of an environment. Higher values indicate unknown regions (few data points). (c) Hierarchical Gaussian Mixture Models represent surfaces in the environment with clusters of Gaussians.

2.1 Environment Mapping

The key problem to solve in the environment mapping domain is how to accurately model the topographical structure of an environment. In the field of robotics, most mapping algorithms are probabilistic [17]. Even with perfect knowledge of robot position, the inherent noise in sensor observations necessitates a probabilistic environment representation to fuse observations into a coherent map. While sensors can come in many forms (camera, force sensor, infrared, depth sensor, etc.) this work primarily considers range sensors, which measure the time of flight of beams cast into the environment at different orientations to determine how far the robot is from visible surfaces.

Occupancy Grids [19] are one of the more prevalent mapping techniques in robotics as they are easy to implement, do not rely on specific environment features, can be queried quickly, and can easily represent unobserved areas as unknown, which is of vital importance for exploration tasks. A typical occupancy grid is expressed as a discrete grid of cells superimposed on the environment, where each cell is associated with a likelihood of intersecting a physical obstruction. The grid can be expressed as a map m consisting of a set of N cells,

$m = \{m_1, \cdots, m_i, \ldots, m_N\}$, each storing the occupancy likelihood of the form:

$$p(m_i = occ|o_1, \ldots, o_n) \in [0, 1],$$

where m_i can be in either the occupied (occ) or free ($free$) state, and $\{o_1, \ldots, o_n\}$ is a set of n observations where $o_j = \{hit, miss\}$ indicates whether or not the jth observation recognizes cell m_i as occupied. An example of this sensing action is shown in Fig.2.2a, with a single beam impacting a green obstacle. For the sake of brevity, we assume that the robot location and sensor action are directly correlated, thus any probability conditioned on o_i is also conditioned on the robot location when the observation is collected. Note that

$$p(m_i = occ|o_1, \ldots, o_n) = 1 - p(m_i = free|o_1, \ldots, o_n)$$

and that when

$$p(m_i = occ|o_1, \ldots, o_n) = p(m_i = free|o_1, \ldots, o_n) = 0.5$$

a cell's state is completely unknown. The goal of exploration algorithms is to prioritize the observation of these unknown cells, and cells with occupancy likelihoods close to 0.5, in order to maximally decrease the uncertainty in the environment model.

Gaussian Processes (GPs) are a fairly common technique for regression or classification that can robustly model functions with sparse and noisy data [20]. GPs represent functions as a Gaussian probability distribution in function space, using a kernel function to describe the correlation between inputs. O'Callaghan and Ramos [10] show how the GP can be used to model environment topography by incorporating beam measurements in a similar fashion as the occupancy grid. However, since a GP does not discretize the space, the point of impact is treated as a point with a high likelihood of occupancy and the space the beam passed through as free space. As a result, the map a GP generates is much more detailed and much smoother than an occupancy grid could generate. Additionally, the variance of the GP at points in space, as visualized in

Fig. 2.2b, can serve to identify unexplored regions, as regions with fewer data points will have higher variance. However, querying the occupancy state of points in the space scales poorly with the number of input observations. Even though there has been significant advances in maintaining the same quality of representation with sparser data [21], the sheer volume of multiple robots taking many measurements over long periods of time precludes the use of GPs for the proposed application.

Hierarchical Gaussian Mixture Models (HGMMs) have also shown promise in representing environment topology [18]. HGMMs work by converting the sensor observations into point clouds and using Expectation Maximization to learn the Gaussian parameters that best fit the cloud. If a measurement is taken in the presence of an HGMM, the point cloud is evaluated for novelty. Data that is determined to belong to the HGMM is integrated into the model, while novel data is used to spawn another Gaussian. The result is an approach that models surfaces in the environment with Gaussians of various sizes that adapt to the resolution required for high accuracy. Unfortunately, the model does not represent unknown space well, making it difficult to determine where new measurements are required. Additionally, the model is not easily extended to handle dynamic environments.

2.1.1 Dynamic Environments

The main challenge in modeling dynamic environments is that these models, which require effort to be learned, require a comparable effort to be unlearned when the environment changes. If the changes in the environment occur as the result of objects moving, one solution is to model the objects explicitly and remove their effects from the static model [22]. Another solution is to model the propagation of change using particle filters [23]. While these approaches improve the accuracy of the model, they provide us little information about the nature of the environment dynamics. To benefit from long duration deployments, a model must be able to recognize patterns in the dynamics and use them to inform planning. Recall that the purpose of this work is to

identify regions that change rapidly and prioritize those regions to ensure they remain covered. From the exploration perspective, the desired effect is an increase in uncertainty relative to the frequency at which regions experience change. The more often a region is in the unknown state, the more frequently it will be visited by exploring robots. In this work, we investigate different means of growing uncertainty and their effect on the overall performance of the proposed system.

2.1.2 Active Perception

When planning deployment strategies to efficiently build an environment model, the key challenge becomes: how to choose where to deploy robots to maximize the accuracy of our model. More specifically, we need to determine the set of observations, given the sensors available on the robot platforms, which collectively best improve the model accuracy. Selecting behaviors in this manner is referred to as *active perception*. The most intuitive approach continuously drives robots towards unknown frontiers, which are defined as the boundaries between known and unknown space [12, 24]. However, these approaches only consider the space a sensor action would reveal, not the degree of improvement a measurement would provide.

Alternatively, information theoretic techniques [25] can be used to determine the most informative set of observations while also accounting for the diminishing returns of redundant, overlapping measurements. Observations are chosen based on the Mutual Information (MI) between the known map and the candidate observation, expressed as the difference in entropy of the map m and the map entropy conditioned on the observation o:

$$I_{MI}[m; o] = H[m] - H[m|o]$$

If m is an occupancy grid, entropy can be expressed as:

$$H[m] = -\sum_{m_i \in m} p(m_i) \log_2 p(m_i) + (1 - p(m_i)) \log_2 (1 - p(m_i)),$$

which can be interpreted as a measure of how uncertain we are about the true state of all cells. As these component probabilities converge towards 0 or 1, the entropy approaches zero, signifying a high level of confidence in the model. Thus, controllers that maximize MI will steadily increase model confidence until the entire environment is known. The proof of this is provided by Julian et al. [26].

Subsequent works look towards approximating MI with a more tractable Cauchy-Schwarz Quadratic Mutual Information (CSQMI) score [27], applying MI to continuous occupancy maps in the form of Gaussian Processes [13], and coupling state-lattice motion primitives with CSQMI to enable real-time exploration of extensive environments [15]. Following the trend of applying Information Theory to new models and planning strategies, this work investigates the application of these techniques to dynamic occupancy grids with multi-robot, energy-constrained deployments.

2.2 Motion Planning

Motion planning, or path planning, involves defining a sequence of states a robot can move through to progress from its starting location to a specified goal. Since this work builds a system tightly integrated with an occupancy grid, the motion planning techniques of most interest are graph-based searches. For these algorithms, the grid m is treated as a graph with $|m|$ nodes and connectivity from each cell to neighboring cells, which typically include all cells in direct contact. The neighborhood can differ between implementations, but rarely includes more than the direct neighbors as the graph-search algorithms scale poorly with connectivity. If we assume that motion is possible between unoccupied cells, then a solution path consists of the sequence of states, s, whose positions correspond to the centroids of cells from map, m, a robot travels through to reach the goal state, s_n:

$$P = [s_1, \ldots, s_n | m].$$

Most grid based path planners are derived from Dijkstra's algorithm [28] or one of its variants. Dijkstra's algorithm is a graph search algorithm that begins at the starting node and iteratively extends paths from the shortest current path to the next closest node, unless a shorter path already leads to that node. As a result, paths grow in a wave from the start location until the goal is found or there are no more paths to find. Each path found is, by construction, the shortest path from the start location. This quality is particularly attractive for the proposed problem, as we are interested in deploying multiple robots to simultaneously observe locations in the environment. Rather than compute a single path for each possible motion from start to observation or from observation to observation, it is possible to grow paths from each location and extract those paths that are required to compute the high-level routes. As this algorithm is used in the proposed system, a more detailed description is provided in Algorithm 2.

The next extension of Dijkstra's algorithm is the A* [29] algorithm, which uses a heuristic cost-to-go to choose the nodes to extend. As a result the algorithm finds the solution with less nodes considered. However, the multi-goal planning capability of Dijkstra's algorithm is better suited for the proposed system. Further extensions include D* [30] and Lifelong Planning A* [31], which are tuned to quickly resolve changes in graph edge costs during run-time. An interesting extension to this work would be the application of these methods when the system must plan for scenarios where robots interact with dynamic obstacles. Similarly, Wagner and Choset [32] present a multi-robot A*, known as M*, that efficiently handles robot to robot collisions through subdimensional expansion, which only couples planners when and where they intersect. Otherwise, each robot computes an individual A* path. Considering the explosion of dimensionality when using A* to compute multi-robot paths, which must necessarily include the joint configuration space of each robot, M* might offer insight into the tractable computation of safe paths with the proposed system when robot to robot collisions are allowed.

For quadrotors, defining a sequence of feasible states in state space is difficult. The state space can be expressed in terms of position, velocity, and orientation of the robot's center of

mass and angular velocity:

$$s = [x, y, z, \phi, \psi, \theta, \dot{x}, \dot{y}, \dot{z}, p, q, r],$$

which is far too large to search, both in terms of complexity and memory requirements. Typical trajectory generation algorithms leverage a result provided by Mellinger and Kumar [33] that exploits the differential flatness of quadrotors to define a much more manageable search space:

$$f = [x, y, z, \psi],$$

which consist of the center of mass position and yaw. Any curve in the space of the flat outputs of the form

$$f(t) : [t_0, t_m] \rightarrow \mathcal{R}^3 \times SO(2)$$

that is smooth up to the third derivative can be followed by a quadrotor. Many state of the art techniques use a hierarchical approach to decouple aspects of the planning problem. For instance, Richter et al. [34] use a sampling-based planner to generate a piecewise-linear path to a desired goal. Then, the path is updated by using Quadratic Programming to solve for minimal snap polynomials that pass through the path nodes. However, as the resulting polynomial differs in shape from the original path, further refinement is required to avoid collisions.

Alternative approaches seek to avoid this issue by defining a library of feasible motions that can be concatenated into piecewise-cubic splines

$$S(t) = \begin{cases} f_1(t) : t \in [t_0, t_1] \\ f_2(t) : t \in [t_1, t_2] \\ f_3(t) : t \in [t_2, t_3] \\ f_4(t) : t \in [t_3, t_4] \end{cases}.$$

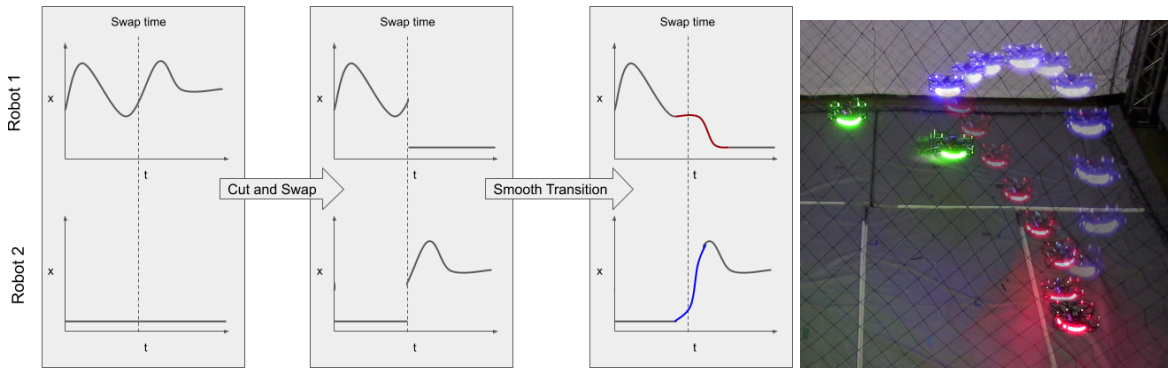
The set of polynomials, $f_1(t), \cdots, f_n(t)$, known as a motion primitive library, has proven effective in both single-robot [35] and multi-robot [36] planning. For this work, we construct a library that describes all possible motion between cells in a grid such that any path generated by the Dijkstra's algorithm can be quickly converted into a dynamically feasible, smooth polynomial spline. The details of this effort are listed in Sect. 5.4.

2.3 Long-Duration Autonomy

When robots have a limited battery capacity and are assigned to tasks that require more than this limit, persistent deployment plans must consider the periodic replenishment of expended energy. The most obvious solution is to compute energy-agnostic plans, segment them into many energy-feasible plans, and assign the new set to available robots [37]. In this manner, we can divide the effort of performing the plan among available robots at the cost of planning the corresponding approaches and departures for each segment. Once the segments are determined, the remaining challenge lies in computing collision-free trajectories that effect the exchange. This process is highlighted in Fig. 2.3, where exchange trajectories are computed at a specific swap in a manner which minimizes the divergence from the original plan.

This approach is of particular relevance when the divergence from the plan must be minimized, such as in theatrical scenarios [38], where the visual effect depends on robots following a specific trajectory. However, in cases where the main objective is to service targets with significant separation and there are few restrictions on how robots move through the environment, segmenting a single-robot plan into a multi-robot, energy-feasible plan will result in more excess travel through the environment than if targets are appropriately clustered before being assigned to robots.

An alternative approach [39] applies Control Barrier Functions (CBFs) to the control input to force the robot towards charging stations when energy is low. When a robot's stored energy exceeds the energy required to travel to the charging station, the control input is entirely influenced



(a) Trajectory swapping process (b) Robot swapping action [38]

Figure 2.3: Enforcing long duration operation by computing trajectories that extract active robots with low battery and replace them with fully charged robots. (a) Splines for the exchange are computed by splitting trajectories at the desired time, t_{swap}, exchanging the trajectory components at $t > t_{swap}$, and computing smooth trajectories to reconnect the splines. (b) The process shown on robots in formation flight. Green robots are in the active formation, the red robot is leaving to charge, and the blue robot is entering the formation.

by the desired control. As the energy depletes, the safe region a robot can inhabit shrinks, which limits the control input such that the robot can always reach the charging station with the energy remaining. In this manner, the robot is drawn to the charging station until the energy is so low that the only available control input drives the robot onto the charging station. At this point, the robot is constrained to continue charging until it reaches a user specified level. Afterwards, the restriction on control input is relaxed and normal operation can resume. While this approach is very flexible and generally applicable to any control input, it requires a clear path to the charging station at all times to ensure feasibility.

2.3.1 Capacitated Routing

The above approaches enforce energy constraints by repairing non-constrained plans or controls. Alternative approaches directly compute energy feasible plans by periodically routing robots back to the charging station. The Vehicle Routing Problem (VRP) [6] and its variants are particularly suited to these approaches as routes, by definition, are cycles through a set of locations that start and end at the same place.

This problem is heavily explored in the operations research community, where routes between cities are computed to efficiently distribute resources from a centralized depot [2, 40, 41]. The core problem is solved using branch and bound techniques to efficiently search through the space of feasible solutions, but handling the additional complexity of travel distance, resource capacity, and time window constraints requires heuristic approaches to find reasonable solutions within practical time limits. Some heuristics define a simple improvement function, such as 2-opt exchanges, to iteratively increase the optimality of the current best solution [42], while others rely on Genetic Algorithms [9] or Tabu search [8] to refine the search process.

For robotics problems, these approaches have been adapted to operate in unknown environments with stochastic operation costs [43, 44]. Rather than expending effort to find optimal solutions to the entire problem, VRP instances are defined over finite horizons and solved in succession as new information becomes available. In particular, Stump and Michael [45] present a solution to the VRP with Time Windows (VRPTW) that pursues the periodic visitation of a set of sites by defining the VRPTW over a sliding-window horizon. In the proposed work, we adapt these proven combinatorial optimization techniques to incorporate the predicted changes in cost and demand derived from the learned environment model.

We first approach the persistent exploration problem by investigating how the combinatorial optimization approaches can adapt to unknown or changing costs. The problem is formulated as an energy-constrained Vehicle Routing Problem. A typical VRP instance consists of a graph $G = \{V, E\}$ where V is a set of nodes an agent must visit and E is the set of edges connecting each of the nodes. The solution is then found by determining the optimal value of the objective function:

$$\min_{x_{ij}^k} \sum_{k \in K} \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij}^k \tag{2.1}$$

subject to the following constraints:

$$\sum_{k \in K} \sum_{j \in V} x_{ij}^k = 1 \qquad \forall i \in V \tag{2.2}$$

$$\sum_{i \in N} x_{ih}^k - \sum_{j \in N} x_{hj}^k = 0 \qquad \forall h \in V, k \in K \tag{2.3}$$

$$\sum_{j \in V \cup S_g} x_{oj}^k \leq 1 \qquad \forall k \in K \tag{2.4}$$

$$\sum_{i \in S_o \cup V} x_{ig}^k \leq 1 \qquad \forall k \in K \tag{2.5}$$

$$x_{ij}^k \in \{0, 1\} \qquad \forall i, j \in V \tag{2.6}$$

where $c_{ij} \in \mathcal{R}^+$ is the cost of traversing the edge (i, j) between nodes i and j, and x_{ij}^k is the binary decision variable that determines whether or not (i, j) is is traversed by robot k in the set of K robots. Constraint (2.2) ensures that only one edge leaves node i for all i. Constraint (2.3) forces the number of edges leaving a node to equal the number entering. The last two constraints represent the depot departure (2.4) and arrival (2.5) constraints, ensuring that each agent leaves and enters the collective starting node at most once.

This formulation is known as an Integer Linear Program (ILP), which is an extension of the more general Linear Program (LP). A typical LP is represented by a linear objective and a series of linear constraints. The constraints effectively define an n-dimensional polytope with the objective function describing the vector along which the optimal solution resides. One way to solve for the optimal solution, known as the Simplex algorithm, is to walk along the edges of the polytope in the direction of vertices with non-decreasing objective values until the optimum is reached. When the decision variables are constrained to be integers, a technique known as branch and bound is often employed. This approach begins by relaxing the integer constraints (2.6) and solving the resulting LP to optimality. If the resulting decision variables are not integer, we recursively split the search space by forcing the non-integer components to take the closest integer values. At each iteration, the problem is *branched* by generating two new problems, one

with a non-integer variable rounded up and one rounded down. As integer solutions are found, we track the minimum objective value and cease branching when relaxed solution's objective value is greater than the current best solution. This is known as *bounding* and prevents exploring space that cannot contain a more optimal solution.

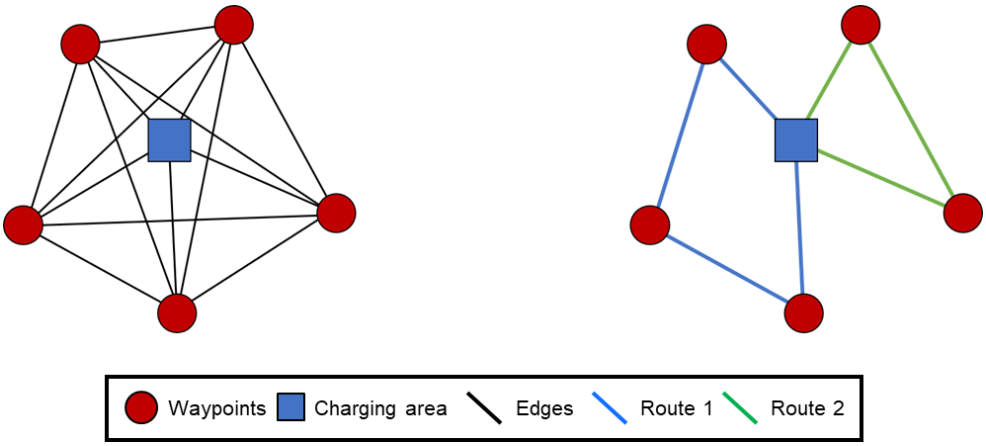


Figure 2.4: Graphical representation of a Vehicle Routing Problem. (Left) A graph is represented by nodes that require visitation and edges that denote the cost to transition between nodes. (Right) The solution to a VRP is a series of routes that specify the sequence of nodes each robot is assigned to visit.

Figure 2.4 shows a visual representation of a VRP. After solving the ILP (2.1)- (2.5), the decision variables x_{ij}^k describe a series of routes through the node set. A route, in this context, is defined as the sequence of nodes assigned to a single robot. We derive the sequence from x_{ij}^k by recursively adding a node to the sequence, then extending along the only active edge (where $x_{ij}^k = 1$) to the next node, starting from the start node S_o and ending at the goal node S_g.

Capacitated VRP is particularly applicable as robots are given a limited capacity of a resource (e.g. energy), $B_{\text{max}} \in \mathcal{R}^+$, and routes are constrained to deliver or expend less than this amount. For energy-constrained routing, there are several expressions of these constraints that apply. The

simplest describes the energy constraints as proportional travel cost:

$$\min_{x_{ij}^k} \sum_{k\in K}\sum_{i\in V}\sum_{j\in V} c_{ij}x_{ij}^k$$

s.t. VRP Constraints, (2.2) - (2.6)

$$\zeta_k \sum_{i\in V}\sum_{j\in V} c_{ij}x_{ij}^k \leq B_{\text{max},k} \qquad \forall k \in K$$

where $\zeta_k \in \mathcal{R}^+$ is a constant that describes the rate of energy expenditure (per unit travel cost) for robot k and $B_{\text{max},k}$ is the capacity specific to robot k.

Mitchell et al. [46] present an alternative formulation that draws from the single robot energy constraints in [47]. There are some slight differences in the formulation, given that the objective pursues a minimum makespan of travel and node visitation cost:

$$\min_{x_{ij}^k} \max_{k\in K} \sum_{i\in N}\sum_{j\in N} (c_{ij} + d_i)x_{ij}^k,$$

and the scenario allows for multiple depots, requiring a set of subtour elimination constraints. However, the constraints of interest, related to energy capacity are as below:

$$r_j - r_i + f_{ij} \leq M(1 - x_{ij}^k) \quad \forall i,j \in V, k \in K \tag{2.7}$$

$$r_j - r_i + f_{ij} \geq -M(1 - x_{ij}^k) \quad \forall i,j \in V, k \in K \tag{2.8}$$

$$r_j - B_{\text{max}} + f_{ij} \geq -M(1 - x_{ij}^k) \quad \forall i \in D \text{ and } j \in V, k \in K \tag{2.9}$$

$$r_j - B_{\text{max}} + f_{ij} \leq M(1 - x_{ij}^k) \quad \forall i \in D \text{ and } j \in V, k \in K \tag{2.10}$$

$$r_i - f_{ij} \geq -M(1 - x_{ij}^k) \quad \forall i \in V \text{ and } j \in D, k \in K \tag{2.11}$$

$$0 \leq r_i \leq B_{\text{max}} \quad \forall i \in T. \tag{2.12}$$

In these constraints, the decision variable r_i represents the amount of energy left in a robot when it departs node i, f_{ij} is the energy cost to travel between nodes i and j, D is the set of depot

nodes, and $M \in \mathcal{R}^+$ is a large constant value that can be chosen to be $B_{\text{max}} + \max_{ij \in V \cup D} f_{ij}$. The constraints are constructed such that a pair of constraints can represent a single constraint that is only active if a certain edge is included in the solution. For instance, constraints (2.7) and (2.8) can be represented by the constraint $r_i - r_j = f_{ij}$ if $x_{ij} = 1$. This pair of constraints ensures that the energy lost between two nodes is equal to the energy cost of traveling between them. Constraints (2.9) and (2.10) can be expressed as $B_{\text{max}} - r_j = f_{ij}$ and establish the condition that the energy level at a target visited after leaving a depot is equal to the energy capacity minus the energy cost of traversing the connecting edge. Constraint (2.11) similarly can be represented as $r_i \geq f_{ij}$ and restricts the energy lost in approaching a depot to being at most the cost to travel from the preceding target. Finally, constraints (2.12) restricts the energy level, r_i, to $[0, B_{\text{max}}]$. Expressing capacity constraints in this manner allows robots to recharge multiple times per cycle. However, the formulation is so complex that it requires a heuristic algorithm to generate a feasible starting point before it can be applied [46].

In this work, the set of nodes, V, through which robots are routed each planning cycle changes as the locations that are most unknown change. If we choose to plan over a finite horizon that is less than the flight duration of robots on maximum charge, then a robot will have no need to visit a charging station mid-horizon. As such, we can assume that no robots take off after the beginning of a new planning horizon, significantly reducing the formulation to

$$\min_{x_{ij}^k} \sum_{k \in K} \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij}^k$$

s.t. VRP Constraints, (2.2) - (2.6)

$$\zeta_k \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij}^k \leq B_{\text{max},k} \qquad \forall k \in K.$$

This formulation serves as the core of the routing problems addressed in this thesis.

Chapter 3

Persistent Multi-Robot Mapping in an Uncertain Environment

This chapter presents a system developed to enable persistent exploration with a team of energy-constrained robots. Typical occupancy map approaches assume a static world; however, we introduce a decay in confidence that degrades the occupancy probability of grid cells and promotes revisitation. Further, sections of the map whose occupancy differs between observations are visited more frequently, while unchanging areas are scheduled less frequently. While planning informative, energy-feasible paths is intractable through the entire space of multi-agent spatiotemporal states, the proposed algorithm decouples planning such that constraints are resolved separately by solving tractable subproblems. We evaluate this approach in simulation and show how the uncertainty of the world model is maintained below an acceptable threshold while the algorithm retains a tractable computation time.

3.1 Problem Formulation

Addressing the persistent exploration problem requires a map representation that enables learning the dynamics of an environment and a motion planner that directs robots to collect observa-

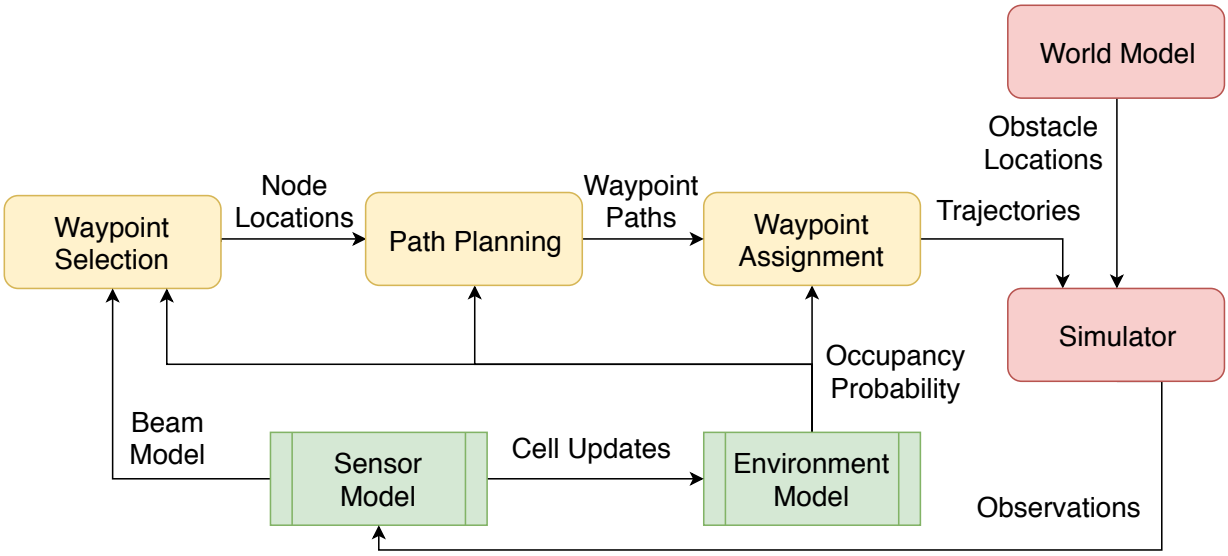


Figure 3.1: System Diagram. The planning pipeline (yellow blocks) generates plans executed by the simulation environment (red blocks), which generates observations for the observation model (green blocks) that provides map information to the planning pipeline.

tions that best inform the map. The system we propose utilizes a 3D occupancy grid representation of the environment, where $p(m_i) \in [0, 1]$ denotes the probability that a cell i in grid m is occupied by an object. The occupancy state is determined by threshold values γ_{free} and γ_{occ}, where cell i is occupied if $p(m_i) \geq \gamma_{\text{occ}}$, free if $p(m_i) \leq \gamma_{\text{free}}$, and unknown otherwise. The likelihood of occupancy given a series of observations is defined as

$$p(m_i|o_{1:n}) = \frac{p(m_i|o_n)p(o_n)}{p(m_i)} \frac{p(m_i|o_{1:n-1})}{p(o_n|o_{1:n-1})}, \tag{3.1}$$

where o_n represents the last in a series of n sequential observations and $o_{1:n} = \{o_1, \ldots, o_n\}$. The standard assumptions for occupancy grid mapping include: 1) Cells are independent of one another ($p(m|o_{1:n}) = \prod_i p(m_i|o_{1:n})$) 2) robots have perfect localization, and 3) unobserved cells begin with a uniform prior occupancy likelihood ($p(m_i) = p(\neg m_i) = 0.5$). Given this representation, the mapping aspect of the exploration problem in dynamic environments can be redefined as: learn a function $p(t, m_i|o_{1:n})$ that describes the time dependence on the occupancy likelihood given a set of observations of the environment.

We simplify the planning aspect of this problem with a sliding-window framework, where plans are generated for an established time window from t_{start} to t_{end} that is advanced by a duration d_{adv} after each iteration. Per horizon window, we determine the optimal plan to deploy a team of energy-constrained robots to collect the set of observations that best inform $p(t, m_i|o_{1:n})$. This is a complex, multi-faceted problem that is made tractable by decomposing it into the following three component problems: 1) Define a set of observations, $o_{1:n}$ of high utility whose locations, $x_{1:n}$ (hereafter referred to as waypoints), are reachable from a starting location, x_s, 2) compute trajectories that direct robot motion between waypoints without entering unknown or occupied regions, and 3) determine the optimal routes through waypoints such that energy expenditure does not exceed robot capacity.

3.2 Methodology

The system we propose in this work is outlined in Figure 3.1, with the planning pipeline presented in graphical form in Figure 3.2. A planning cycle begins with the *Waypoint Selection* algorithm which iteratively selects the N_w most informative waypoints from a set of reachable locations. Intuitively, a waypoint scores high if the corresponding observation, provided by the *Sensor Model*, measures a large number of uncertain cells with high probability, with diminishing returns provided for repeated coverage. Note that we limit the number of waypoints selected to reduce the computational burden of the rest of the pipeline and restrict the number of robots deployed over the planned horizon.

Once waypoints are selected, the *Path Planning* algorithm generates a set of shortest-distance paths between all pairs of waypoints. The focus is to provide a fast estimate of cost and feasibility to inform *Waypoint Assignment*, as opposed to investing in high-quality paths that may never be traversed. To simplify collision avoidance, we assume all dynamic objects exist outside a user-defined safety region, designated by a radius of r_{safe} around the starting location, x_s, and all locations above a height of h_{safe}. Given the proposed scenario, this is a reasonable assumption

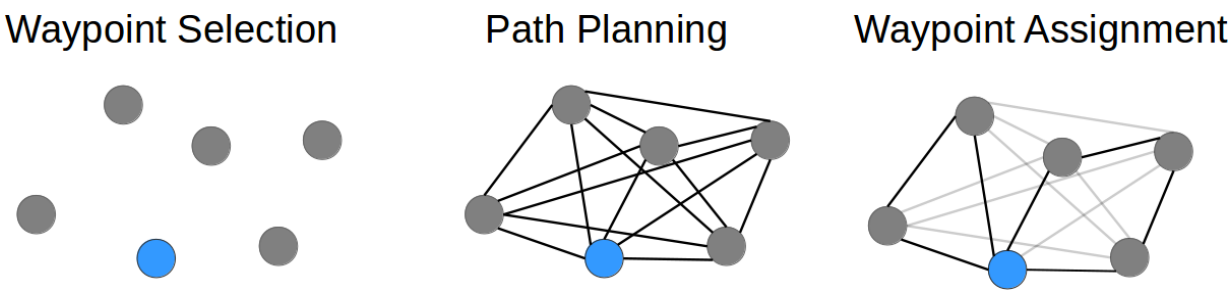
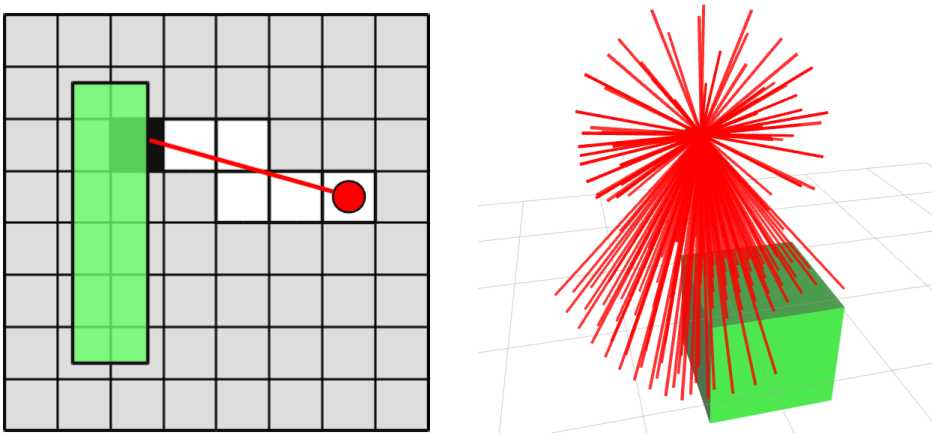


Figure 3.2: The planning pipeline expressed in graph form. (Left) A set of informative waypoints (grey), reachable from the starting location (blue), are selected. (Middle) The paths are computed between all waypoints. (Right) Efficient, energy-constrained routes through the waypoints are chosen.

that allows *Path Planning* to assume an accurate *Environment Model*.

The *Waypoint Assignment* algorithm uses these path costs to generate an instance of the Vehicle Routing Problem with Time Windows (VRPTW). The time windows, which define when a waypoint can be visited, open the instant any cell viewed from a waypoint transitions to unknown and close the last instant the path to x_s can be traversed, ensuring feasible motion while maximizing information gain. Once the routes are generated, trajectories are formed by concatenating paths between the waypoints along each route. The *Simulator* then executes these plans, using the *Sensor Model* to generate observations at a constant rate along trajectories traversed, for a duration of d_{adv}. The *Environment Model* is then updated and the planning pipeline starts over with a new horizon.

Under the traditional occupancy grid formulation, the *Environment Model* would simply process these observations to update the occupancy probability of all cells observed. However, the proposed model searches for discrepancies in the sequence of observations of each cell and tracks a period of change, $\mathcal{T}_i$, to suggest when the occupancy probability should transition to unknown. As observations are collected and $\mathcal{T}_i$ is learned, $p(t, m_i|o_{1:n})$ will decay exponentially towards $p(m_i)$ at a rate corresponding to the dynamics of cell i, allowing us to focus the efforts of our limited system only where it is needed.



(a) Beam Model (b) Sensor Model

Figure 3.3: The observation model. (Left) A single beam update marks all cells passed through as a miss ($p(m_i|o_n) = \gamma_{\text{miss}}$) and the final cell as a hit ($p(m_i|o_n) = \gamma_{\text{hit}}$) if it impacts and obstacle. (Right) The sensor model consists of one upward facing range sensor with a wide angle and a short range and one downward facing sensor with a close angle and long range.

Sensor Model In this chapter, we consider a robotic platform equipped with multiple range sensors. The model for a single beam is depicted in Figure 3.3a, where each cell a beam passes through is registered as a miss ($p(m_i|o_n) = \gamma_{\text{miss}}$) and any cell in which the beam impacts an object registers a hit ($p(m_i|o_n) = \gamma_{\text{hit}}$). The *Sensor Model* shown in Figure 3.3b simulates two range sensors by extending beams upwards 0.825m in a 270 degree arc about the x and y axes and downward 1.75m in a 60 degree arc. This configuration allows for the robot to explore the space it travels while focusing its attention towards the dynamic area below h_{safe}. Given the symmetry about yaw and the slow motion typical of quadrotors performing exploration tasks, we assume robots operate at a fixed orientation.

Environment Model Our key innovation for the occupancy grid representation lies in the introduction of *confidence decay*, which drives known cells towards the unknown state in accordance with environment dynamics. We modify the occupancy likelihood update rule (3.1) to generate the recursive summation:

$$\text{logit}(p(m_i|o_{1:n})) = \text{logit}(p(m_i|o_n)) - \text{logit}(p(m_i)) + \text{logit}(p(m_i|o_{1:n-1}), \tag{3.2}$$

also known as the *inverse measurement model* [48]. Note that, $\text{logit}(p) = \log \frac{p}{1-p}$ for a probability p and is known as the log-odds expression of p, which not only simplifies the update rule, but also improves robustness to floating-point errors for small probabilities. Given that this representation shifts the domain of occupancy likelihood from $[0, 1]$ to $(-\infty, \infty)$, we can directly apply an exponential decay to cause cells to asymptotically approach $p(m_i) = 0.5$

$$\text{logit}(p(t, m_i|o_{1:n})) = e^{-\alpha_i(t-t_{\text{last,i}})} \text{logit}(p(m_i|o_{1:n})), \tag{3.3}$$

where $t_{\text{last,i}}$ is the last time cell i was updated and α_i is a parameter that defines the rate of decay. For the purpose of tracking environment dynamics, it is convenient to define α_i in terms of the time it takes a cell to transition to the unknown state from a high occupancy likelihood, γ_{min} or γ_{max}

$$\alpha_i = \begin{cases} \frac{1}{\mathcal{T}_i} \log(\frac{\gamma_{\text{max}}}{\gamma_{\text{occ}}}) & \text{if } p(m_i|o_{1:n}) > 0.5 \\ \frac{1}{\mathcal{T}_i} \log(\frac{\gamma_{\text{min}}}{\gamma_{\text{free}}}) & \text{if } p(m_i|o_{1:n}) < 0.5 \\ 0 & \text{otherwise} \end{cases}. \tag{3.4}$$

Thus, the occupancy probability evolves over time, as depicted in Figure 3.4, crossing the threshold between states after a duration of $\mathcal{T}_i$, the decay period.

The decay period is tracked from a sequence of observations by recognizing two types of measurements. The first occurs when the observation of a cell disagrees with the model, for instance, if a cell is in the free state and registers a hit:

$$\mathcal{T}_{\text{meas}} = t - t_{\text{last,i}}. \tag{3.5}$$

This can be interpreted as an observation that there has been a change between t and $t_{\text{last,i}}$, therefore, the period of change must be at least $\mathcal{T}_{\text{meas}}$. The second measurement occurs when an

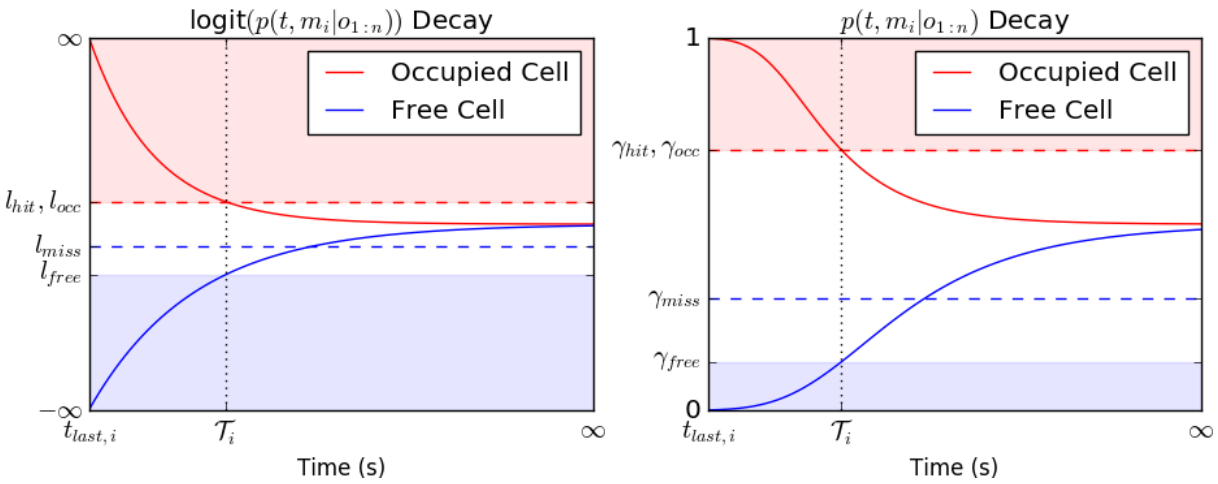


Figure 3.4: Confidence decay for the log-odds (left) and probability (right) representation of occupancy likelihood. To simplify labels, l represents log-odds parameters ($l_{\text{free}} = \text{logit}(\gamma_{\text{free}})$). Plots show the progression of a cell beginning in the free (blue) or occupied (red) state at high confidence and decaying over time. The decay function is designed such that cells with high model confidence transition to the unknown state within a duration of $\mathcal{T}_i$.

observation is made after the decay period ($\mathcal{T}_{\text{meas}} > \mathcal{T}_i$) that agrees with the model:

$$\mathcal{T}_{\text{meas}} = t - t_{\text{shift,i}}, \tag{3.6}$$

where $t_{\text{shift,i}}$ is the previous time that cell i changed to a free or occupied state from some other state. We interpret this as an observation that the cell has been consistent since $t_{\text{shift,i}}$, therefore, the decay period must be at least $\mathcal{T}_{\text{meas}}$.

Updating $\mathcal{T}_i$ is handled using the Kalman filter update rule:

$$d\mathcal{T} = \mathcal{T}_{\text{meas}} - \mathcal{T}_i, \quad d\rho = \rho_i + \rho_{\text{proc}}$$
$$K = \frac{d\rho}{\rho_{\text{meas}} + d\rho}$$
$$\mathcal{T}_i = \mathcal{T}_i + K d\mathcal{T}, \quad \rho_i = (1 - K)d\rho$$

where ρ_i is the variance of the estimate of $\mathcal{T}_i$, ρ_{meas} is the measurement noise, and ρ_{proc} is the process noise. Both noise parameters can be tuned to suit the expected variance in dynamics of

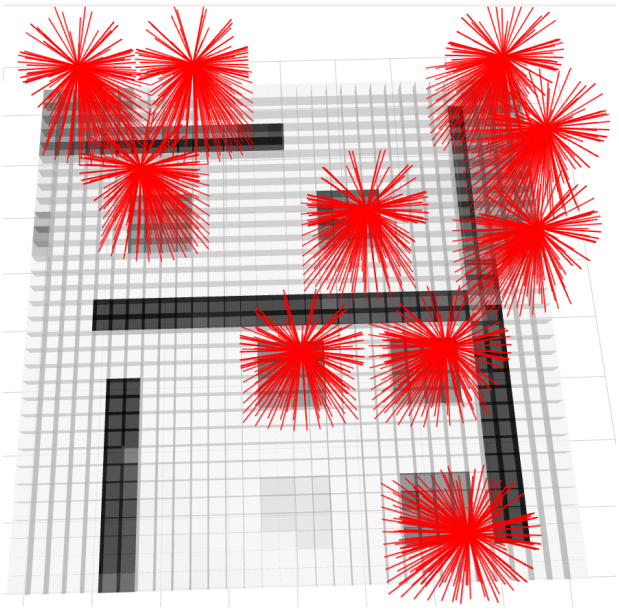


Figure 3.5: *Waypoint Selection* uses the *Sensor Model* to determine the optimal observation locations. In this top-down horizontal slice of the occupancy grid, where the occupancy probability of a cell is indicated by the shade of gray, the eight most informative locations extend beams through cells that are predominantly gray (unknown), as opposed to white (free) or black (occupied).

the environment, where increased measurement noise can blend widely varying measurements and increased process noise can prevent the estimate from converging too quickly. While this is a somewhat simple and imprecise approach to modeling periodicity in the environment, it is effective at differentiating between static and dynamic regions while dismissing transient changes as noise.

Waypoint Selection The first stage of the planning pipeline identifies a set of up to N_w reachable locations from which informative observations can be made. The informative quality of a set of locations is determined from the Mutual Information (MI) in the observations taken at all

locations, defined as:

$$\begin{aligned} I_{MI}[m;o] &= H[m] - H[m|o] \\ &= -\sum_{i,j} p(m_i, o_j) \log \frac{p(m_i)p(o_j)}{p(m_i, o_j)} \end{aligned}$$

where $H[m]$ is the map entropy and $H[m|o]$ the map entropy conditioned on the set of all observations. Intuitively, this utility score prefers a set of observations that cover the largest volume of unknown space, with diminishing returns for overlap, as shown in Figure 3.5. Cauchy-Schwarz Quadratic Mutual Information (CSQMI) [27] provides an alternative measure of MI in the form

$$I_{CS}[m;o] = -\log \frac{(\sum_m \int_o p(m,o)p(m)p(o)do)^2)}{\sum_m \int_o p^2(m,o)do \sum_m \int_o p^2(m)p^2(o)do}.$$

The key benefit of CSQMI is that the integrals can be computed analytically, resulting in a faster solution than the numerical integration of MI. Additionally, Charrow et al. [27] present an approach to further marginalize joint probability distributions between multiple beams intersecting the same cells by ignoring beams that impact cells already being observed. We leverage this measure to score the utility of observations, where the observation set with the largest CSQMI value provides the best potential improvement of map confidence. For convenience, we assume a locally static map, where CSQMI is evaluated against $p(t_{\text{start}}, m|o_{1:n})$ derived from (3.3).

Algorithm 1 shows the details of the *Waypoint Selection* algorithm. First, function DijkstraReachables uses the path planning approach to find the set of all cells reachable from the start location x_s and store their centroids in L. In lines 3 to 17 the CSQMI function is used to determine the information value of a set of waypoints. The waypoint which provides the set W with the most additional information is appended to W. Note that if the added value is less than a threshold ϵ, the algorithm breaks early and returns all waypoints found. This prevents the added computational burden of scheduling uninformative waypoints.

Algorithm 1 Waypoint Selection

1: **procedure** WAYPOINTSELECT(m, x_s, N_w, ϵ)
2: $L \leftarrow$ DijkstraReachables(m, x_s)
3: $W \leftarrow \emptyset$ ▷ Waypoint position set
4: **while** $|W| < N_w$ **do**
5: $w_{\text{next}} = \arg\max_{l \in L}$ CSQMI$(m, t_{\text{start}}, W \cup l)$
6: $d =$ CSQMI$(m, t_{\text{start}}, W \cup w_{\text{next}})$
7: $-$CSQMI(m, t_{start}, W)
8: **if** $d < \epsilon$ **then**
9: **return** W
10: **end if**
11: $W = W \cup w_{\text{next}}$
12: **end while**
13: **return** W
14: **end procedure**

Path Planning Algorithm 2 presents Dijkstra's algorithm modified to generate shortest-distance paths to all reachable cells through our time-dependent occupancy grid. Feasibility of motion is determined by assuming a constant travel speed, v_{travel}, and checking if cells are free at the time of arrival. Dijkstra's algorithm typically begins with an open set of nodes Q, initially only containing the start node s (line 2), corresponding to the cell from which paths are generated. Then neighboring cells are evaluated, as in lines 12 to 20, to determine if the path through the current cell belongs to the shortest path. Two vectors, dist and prev, store the cost-to-come to cell i from cell s and the previous cell in the shortest path available, respectively. After all reachable cells have been processed, the closed set of nodes, P, contains the set of all reachable cells and the shortest paths are derived by iterating backwards through predecessors until s is reached. The feasibility of transitioning to a cell is checked in line 15, where the boolean function isTraversable(m_j, t) confirms the following conditions relative to cell j at the time of arrival, t are satisfied:

$$\text{height}(j) > h_{\text{safe}} \text{ or } \text{distance}(j, x_s) < r_{\text{safe}}$$

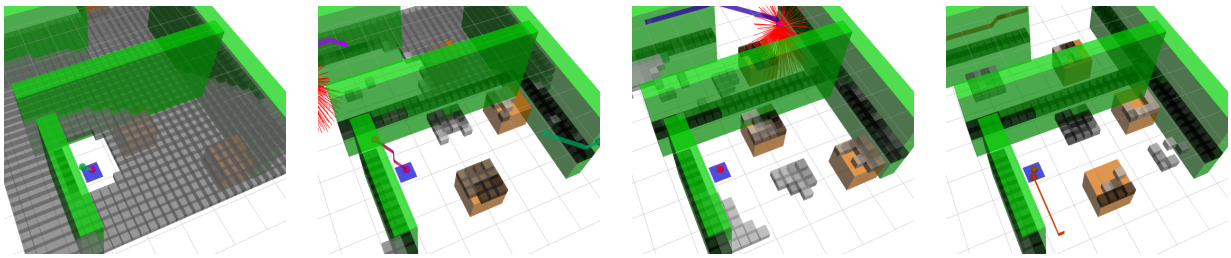
$$p(t, m_j | o_{1:n}) < \gamma_{\text{free}}.$$

Algorithm 2 Dijkstra's Algorithm with Dynamic Occupancy

procedure DYNAMICDIJKSTRAS($m, s, t_s, v_{\text{travel}}$)
  $Q = \{s\}$ ▷ Open nodes
  $P = \emptyset$ ▷ Closed nodes
  for $i = 0, \ldots, |m|$ **do**
    dist$[i] = \infty$ ▷ Distance from start
    prev$[i] = \emptyset$ ▷ Previous node
  end for
  dist$[s] = 0$
  while $Q \neq \emptyset$ **do**
    $i \leftarrow Q$
    $Q = Q \setminus i$, $P = P \cup i$
    for each $j \in$ Neighbors(i) **do**
      $d =$ dist$[i] +$ length(i, j)
      $t = d/v_{\text{travel}} + t_s$
      if $d <$ dist$[j]$ and isTraversable(m_j, t) **then**
        dist$[j] = d$
        prev$[j] = i$
        $Q = Q \cup j$
      end if
    end for
  end while
  return dist, prev
end procedure

The first condition checks if cell j is within the safety region, comparing the height of cell j against h_{safe} and the radial distance of j from the vertical axis through x_s against r_{safe}. The second condition ensures that cell j is free.

In *Waypoint Selection*, the set of all reachable cells is determined by evaluating DynamicDijkstras$(m, x_s, t_{\text{start}}, v_{\text{travel}})$ to grow paths from the start location and extracting reachable cells from P. Once the set of waypoints W has been chosen, pairwise paths are computed by evaluating DynamicDijkstras$(m, w, t_w, v_{\text{travel}})$ for all $w \in W$, where $t_w = t_{\text{start}} + \text{dist}(x_s, w)/v_{\text{travel}}$ is the earliest instance waypoint w can be reached. Paths begun after t_w are not guaranteed feasibility, but we determine the instant a path becomes infeasible and schedule routes accordingly in *Waypoint Assignment*.



(a) ($t = 5$) (b) ($t = 320$) (c) ($t = 580$) (d) ($t = 1200$)

Figure 3.6: Representation of the simulated environment populated by static (green) and dynamic (orange) objects, with the starting location, x_s, indicated in blue. Robots are displayed as cylinders with tails showing the last 5 seconds of motion. The small gray blocks indicate occupancy probability by their shade, with free cells being excluded for visibility. (a) Initially the world is populated by unknown cells ($p(m_i) = 0.5$). (b) As robots explore the environment, the decay rate is learned from repeated measurements. (c) Cells in static areas decay via the same process, but the rate of decay decreases with subsequent concurring observations. (d) When the model reaches a steady state, static cells become consistent and the only unknown cells are those that periodically contain objects.

Waypoint Assignment For the *Waypoint Assignment* algorithm, we leverage the work of Stump and Michael [45] to schedule visitations by formulating an instance of the Vehicle Routing Problem with Time Windows (VRPTW) for the current horizon and solving it using the Stabilized Cutting-Plane Algorithm (SCPA) [49]. The main contribution of [45] was to incorporate virtual nodes corresponding to the intended locations of robots active at t_{start} into the VRPTW formulation as a means of extending previously planned routes. This feature, coupled with the energy capacity constraints inherent in the VRPTW definition, permits the iterative extension of energy-aware routes over many horizons.

To construct a VRPTW instance, we define a graph $G = (A, E)$ of nodes A and edges E. The set A contains two nodes S_o and S_g corresponding to the start location, x_s, at the beginning and end of the route, N_w nodes corresponding to each waypoint in W, and one virtual node for each robot whose route can be extended. Extendable routes are determined by evaluating how much energy would remain if a robot were to continue operating until t_{start}, adding a virtual node corresponding to the last scheduled waypoint before t_{start}.

The edge costs are derived from the paths computed using Algorithm 2. We use the paths

computed in *Waypoint Selection* for the edges between x_s and W. The pairwise paths computed in *Path Planning* for the set W provide the corresponding edge costs. The paths between the virtual nodes and W are computed in a similar fashion, using the planned visit time of the last scheduled waypoint as the path start time t_w. Edges between x_s and the virtual nodes are given a travel cost of zero, to mimic the effect of a robot starting at the virtual node. Edges extending from W to the virtual nodes are disabled to force activation of edges extending from S_o to the virtual nodes, ensuring they are visited first in their routes.

The VRPTW instance for this problem is formally defined as:

$$\min_x \sum_{k \in K} \sum_{(i,j) \in A} c_{ij} x_{ij}^k \tag{3.7}$$

$$s.t. \sum_{k \in K} \sum_{j \in V} x_{i,j}^k = 1 \qquad \forall i \in V \tag{3.8}$$

$$\sum_{j \in V \cup S_g} x_{oj}^k = 1 \qquad \forall k \in K \tag{3.9}$$

$$\sum_{i \in S_o \cup V} x_{ig}^k = 1 \qquad \forall k \in K \tag{3.10}$$

$$\sum_{i \in N} x_{ih}^k - \sum_{j \in N} x_{hj}^k = 0 \qquad \forall h \in V, k \in K \tag{3.11}$$

$$t_i^k - t_j^k + d_{ij} - Z(1 - x_{ij}^k) \leq 0 \qquad \forall (i,j) \in A, k \in K \tag{3.12}$$

$$a_i \leq t_i^k \leq b_i \qquad \forall i \in N, k \in K \tag{3.13}$$

$$\sum_{(i,j) \in A} B_{ij} x_{ij}^k \leq B_{\max} \qquad \forall k \in K \tag{3.14}$$

where $x_{ij}^k \in 0, 1$ indicates if the edge between node i and j is active for robot k from the set of robots K, $t_i^k \in \mathbb{R}^+$ represents the time at which node i is visited by robot k, c_{ij} represents the cost activating the edge between i and j, $V = A \setminus \{S_o, S_g\}$ is a subset of nodes excluding the start and goal, and a_i and b_i define the lower and upper bounds of the time-window for node

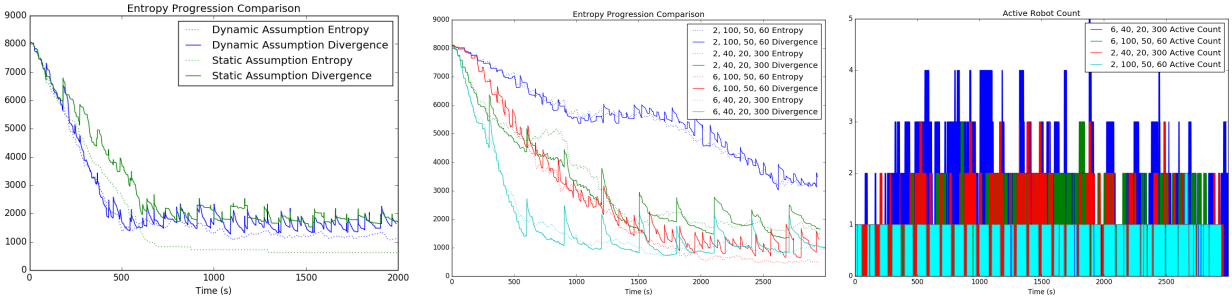


Figure 3.7: Evaluation of system performance under various conditions. (Left) System operating under the static assumption ($\alpha_i = 0$) exhibits a lower entropy and higher divergence than with the dynamic assumption (α_i learned online), suggesting that confidence decay increases accuracy and mitigates overconfidence. (Middle) Entropy and divergence using select parameter settings from Table 3.1. Convergence and steady-state improve as N_w increases and d_{hor} decreases. (Right) Active robot for varying parameters. More robots are deployed per horizon as N_w increases and d_{hor} decreases, to the limit that robots are capable of completely counteracting the entropy growth, given the environment dynamics.

i. By setting c_{ij} to equal the travel time along the path between the corresponding waypoints, the objective (3.7) will minimize the sum of travel time over all robots. Constraints (3.8)-(3.11) ensure that only one robot leaves each node, each robot begins at the start node S_o and ends at the goal node S_g, and any robots entering a node must leave it. Constraint (3.12) ensures that a robot that travels the edge (i, j) arrives at j after t_i^k plus the duration of travel d_{ij}. Constraint (3.13) enforces the time-window bounds, a_i and b_i, and constraint (3.14) limits energy expenditure to the battery limit B_{max}, where $B_{ij} = \max(t_{ij}, a_j - a_i)$ to conservatively estimate the expenditure of energy assuming a constant discharge rate. Note that B_{max} will be equal to the energy capacity of freshly charged robots, but, for a virtual node, will instead be equal to the remaining capacity of the active robot when visiting the last planned waypoint before t_{start}. To ensure the existence of a feasible solution, we solve this system for a team size of $|K| = |V|$.

The upper time-window bound b_i is set to the latest moment the edge (i, g) can be traversed before cells encountered along the path transition to the unknown state. The formula to compute the instant a cell becomes unknown is a simple inversion of the exponential decay from equation (3.3):

$$t_{\text{unknown},i} = \frac{1}{\alpha_i} \log \left(\frac{\text{logit}(p(m_i|o_{1:n}))}{\text{logit}(\gamma_{\text{free}})} \right) + t_{\text{last,i}}. \tag{3.15}$$

The latest departure from waypoint i to reach x_s before the path closes can then be expressed as:

$$b_i = \min_{j \in \text{path}(i,x_s)} t_{\text{unknown},j} - \frac{\text{distance}(i,j)}{v_{\text{travel}}},$$

where $\frac{\text{distance}(i,j)}{v_{\text{travel}}}$ denotes the travel time between nodes i and j. The lower time-window bound a_i corresponds to the earliest moment a cell observed from waypoint i transitions to the unknown state:

$$a_i = \min_{j \in o_i} t_{\text{unknown},j},$$

where each cell j for an observation o_i taken from waypoint i is provided by the *Sensor Model*. Note that if $a_i > b_i$ then we set $a_i = b_i$, forcing the waypoint to be visited at the last possible moment.

Solving a VRPTW in the above form is NP-hard and typically solved using branch and bound methods to efficiently search the space of feasible solutions. Leveraging the work in [49], however, we relax constraint (3.8) and decompose the problem into more tractable subproblems. Note that, excepting (3.8), all constraints are attributable to a single robot. Relaxing (3.8), we generate a Lagrangian Dual objective of the form:

$$z_{LD}(\lambda) = |K| \left(\min_x (c_{ij} - \lambda_i) x_{ij}^k + \sum_{i \in V} \lambda_i \right), \tag{3.16}$$

Note that $\min_x (c_{ij} - \lambda_i) x_{ij}^k$, combined with constraints (3.9)-(3.14), is an expression of the Elementary Shortest Path Problem with Resource Constraints (ESPPRC) [50] for a single robot using costs reduced by the Lagrangian multipliers λ. Maximization of this dual objective is then a matter of testing values of λ against the set of feasible single-robot routes through the waypoint set.

As it is impractical to test the set of all feasible routes, we rely on SCPA [49] to iteratively build a set of Pareto-optimal paths while searching for the optimal λ. In each iteration, the

ESPPRC is solved for a given set of λ values and the resulting path(s) are appended to the constraint set bounding (3.16). When the gap between $z_{LD}(\lambda)$ and the minimal-cost path is small enough, the set of paths that actively constrain the dual objective become the solution set. If each node is visited exactly once in the set of paths, then computation is complete. For multiple visitations, we solve a set of problem formulations with an additional constraint that either forces or excludes one of the transitions x_{ij}^k, applied recursively until all feasible routes are considered.

3.3 Simulation Evaluation

We evaluate the proposed system through simulation experiments designed to address four questions - **Q1**: How well does the system perform under varying environment dynamics? **Q2**: How do the parameter settings affect the evolution of map accuracy and confidence over extended operations? **Q3**: How closely can the system model environment dynamics? **Q4**: Do robots make efficient use of their limited capacity in the planned routes?

Setup Details In an effort to replicate a real-world scenario, we generated the simulated environment depicted in Figure 3.6 consisting of 4m x 4m x 2m area populated by static walls and dynamic objects. The dynamic objects remain still for a duration $\mathcal{T}_{obs}$ before transitioning to another location, causing the cells they intersect to alternate between the free and occupied state.

Robots depart from and return to the starting location, indicated by the blue square in Figure 3.6. They are given an energy capacity corresponding to a maximum flight duration of 100 seconds. Note that it is convenient to describe energy capacity in terms of flight duration for the purposes of evaluation and suffices under the assumption of constant energy expenditure. In the described experiments, robots are added to the system as they are needed and removed when they are finished.

Confidence and Accuracy The main measures of success considered in this work are model confidence and accuracy. Confidence in the probabilistic model is evaluated by computing the evolution of entropy over time:

$$H(t, m) = -\sum_{m_i \in m} p(t, m_i) \log_2 p(t, m_i) + (1 - p(t, m_i)) \log_2 (1 - p(t, m_i)). \tag{3.17}$$

As more cells trend towards $p(t, m_i) = 0.0$ or $p(t, m_i) = 1.0$ entropy will decrease, indicating a high level of confidence in the representation.

Model accuracy is evaluated using the Kullback-Leibler divergence between the learned model and an oracle model constructed from the true state of the world. In this oracle model, cells that intersect an object at time t are given the occupancy probability of $p_o(t, m_i) = 1.0$, whereas cells that do not intersect objects are given $p_o(t, m_i) = 0.0$. The Kullback-Leibler divergence between the learned model and the oracle can be expressed as

$$D_{KL}(t, m) = \sum_{m_i \in m} p_o(t, m_i) \log_2 \frac{p_o(t, m_i)}{p(t, m_i)} + (1 - p_o(t, m_i)) \log_2 \left(\frac{1 - p_o(t, m_i)}{1 - p(t, m_i)} \right), \tag{3.18}$$

where this value will decrease as $p(t, m_i)$ approaches $p_o(t, m_i)$ for all cells.

The left plot of Figure 3.7 shows how these measures evolve under the sample set of parameters: the number of waypoints per horizon, $N_w = 8$, horizon window size, $d_{\text{hor}} = 40$ seconds, horizon advance, $d_{\text{adv}} = 20$ seconds, and the period of change for objects in the environment, $\mathcal{T}_{\text{obs}} = 60$ seconds. Assuming a static environment ($\alpha_i = 0$), we see that entropy falls sharply to a low steady-state value, while divergence remains relatively high. This suggests a overconfidence in a less accurate model representation. Conversely, learning α_i during operation (the dynamic assumption) causes divergence to closely follow entropy, resulting in a higher accuracy at a minor cost of confidence.

We explore **Q1** and **Q2** with a series of test runs operating under various parameter configurations and world dynamics. In the middle plot of Figure 3.7, the entropy and divergence

Table 3.1: Parameter evaluation. We vary N_w, d_{hor}, d_{adv}, and $\mathcal{T}_{\text{obs}}$ over multiple runs and collect: the average number of robots deployed, N_r, the time divergence reaches 10% of the steady-state value, t_{conv}, and the entropy and divergence values averaged over the last 100 seconds of operation, μ_H and μ_D. Trends suggest faster convergence to more accurate maps occurs with higher N_w and lower d_{hor} while higher $\mathcal{T}_{\text{obs}}$ converges slower to higher accuracy.

$N_w, d_{\text{hor}}, d_{\text{adv}}, \mathcal{T}_{\text{obs}}$	N_r	t_{conv}	μ_H	μ_D
2, 40, 20, 60	1.34	1337	1542	2013
3, 40, 20, 60	1.53	948.0	1578	1788
4, 40, 20, 60	1.81	1049	972.4	1502
5, 40, 20, 60	2.04	629.0	1215	1796
6, 40, 20, 60	2.32	499.0	1013	1640
2, 70, 35, 60	1.06	2225	2020	2268
3, 70, 35, 60	1.40	1853	667.1	1111
4, 70, 35, 60	1.54	1779	521.5	1161
5, 70, 35, 60	1.59	1410	519.4	1226
6, 70, 35, 60	1.73	954.0	433.8	1223
2, 100, 50, 60	0.77	2542	3220	3256
3, 100, 50, 60	1.05	2255	1445	1816
4, 100, 50, 60	1.25	2064	706.8	1323
5, 100, 50, 60	1.36	1669	517.4	1050
6, 100, 50, 60	1.40	1458	519.7	1115
2, 40, 20, 300	1.30	1351	1614	1754
3, 40, 20, 300	1.53	896.0	1203	1180
4, 40, 20, 300	1.69	685.0	1136	1295
5, 40, 20, 300	1.93	502.0	959.0	1089
6, 40, 20, 300	2.12	599.0	992.2	1043
2, 70, 35, 300	0.99	2077	2007	2073
3, 70, 35, 300	1.36	1788	593.3	757.6
4, 70, 35, 300	1.48	1475	471.3	582.6
5, 70, 35, 300	1.54	1378	462.6	602.6
6, 70, 35, 300	1.75	1141	482.3	606.8
2, 100, 50, 300	0.76	2640	2912	2739
3, 100, 50, 300	1.09	2557	1246	1205
4, 100, 50, 300	1.26	2101	719.0	798.4
5, 100, 50, 300	1.36	1668	596.6	699.4
6, 100, 50, 300	1.26	1711	530.2	648.5

decrease towards steady-state as the environment is explored, with the rate of convergence and steady-state value changing depending on the chosen parameters. In Table 3.1 we see a general trend of entropy and divergence to converge more quickly to lower values as N_w increases and

d_{hor} decrease, which parallels an increase in the average number of robots deployed. Figure 3.7 (right) corroborates this pattern of deploying more robots when more waypoints are computed for shorter horizons. We interpret from these results that by tuning parameters, we can drive the system to deploy enough robots to saturate the environment with observations, allowing faster convergence to a more accurate representation (**Q2**). Table 3.1 also shows a trend of lower steady-state divergence (μ_D) for slower environment dynamics (higher $\mathcal{T}_{\text{obs}}$). However, the convergence rate suffers as it takes longer to observe the period of change (**Q1**).

Decay Period The precision of the environment dynamics model (**Q3**) is examined by sampling the tracked decay period, $\mathcal{T}_i$, of cells that periodically intersect objects. The progression of $\mathcal{T}_i$ is portrayed in the left plot of Figure 3.8 for a sample run. The naive update rules (3.5) and (3.6) may not closely approximate $\mathcal{T}_{obs}$, but serve to differentiate between static and dynamic regions well. As such, cells that periodically contain objects maintain a consistent decay period, while the decay period of static cells constantly grows.

Capacity Expenditure We check the duration of routes assigned to robots over the course of an operation by compiling a histogram of expended energy for a sample run (Figure 3.8, right). Note that the majority of plans operate for over 80% of the established flight duration, suggesting that the *Waypoint Assignment* algorithm is taking full advantage of the limited energy capacity (**Q4**).

Aperiodic Environments To further validate robustness to dynamic environments (**Q1**), we test the system where $\mathcal{T}_{\text{obs}}$ is chosen randomly between 60s and 300s for every object every cycle. Despite this inconsistent periodicity, Figure 3.9 shows that we maintain a consistent progression of entropy and divergence while maintaining the ability to differentiate between static and dynamic cells.

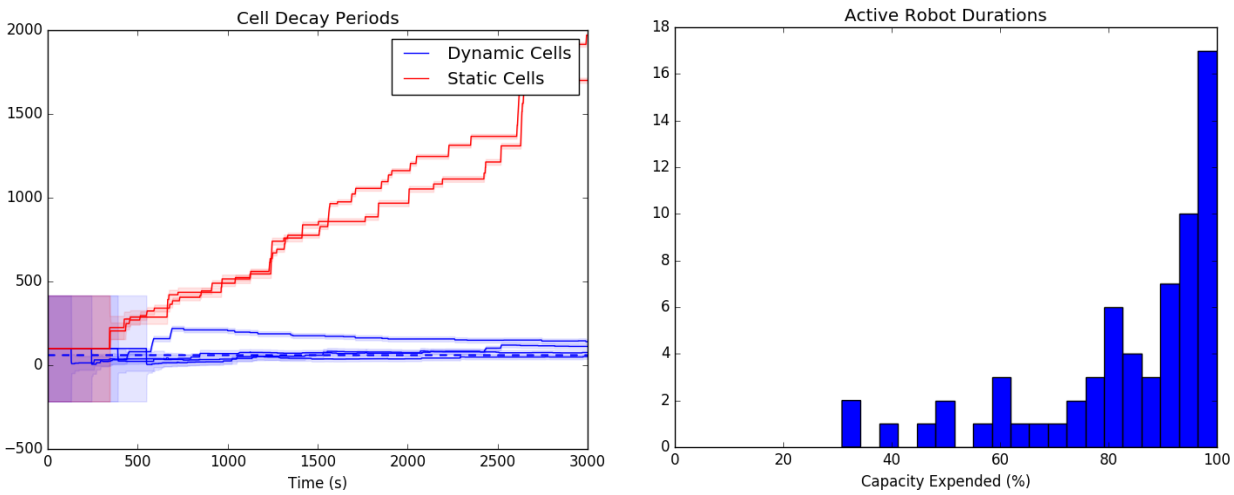


Figure 3.8: Evaluation of dynamics modeling and energy expenditure of a sample run, $N_w = 4$, $d_{\text{hor}} = 40$, $d_{\text{adv}} = 20$, and $f_{obj} = 60$. (Left) The tracked decay period, $\mathcal{T}_i$ for a select set of cells. The solid lines indicate mean $\mathcal{T}_i$ and the shaded areas indicate standard deviation. Cells that periodically intersect objects approximate $\mathcal{T}_{obs}$, while cells in static locations consistently increase $\mathcal{T}_i$. (Right) Histogram suggests that robots prefer to expend the majority of their capacity. Short duration routes are avoided to improve efficiency.

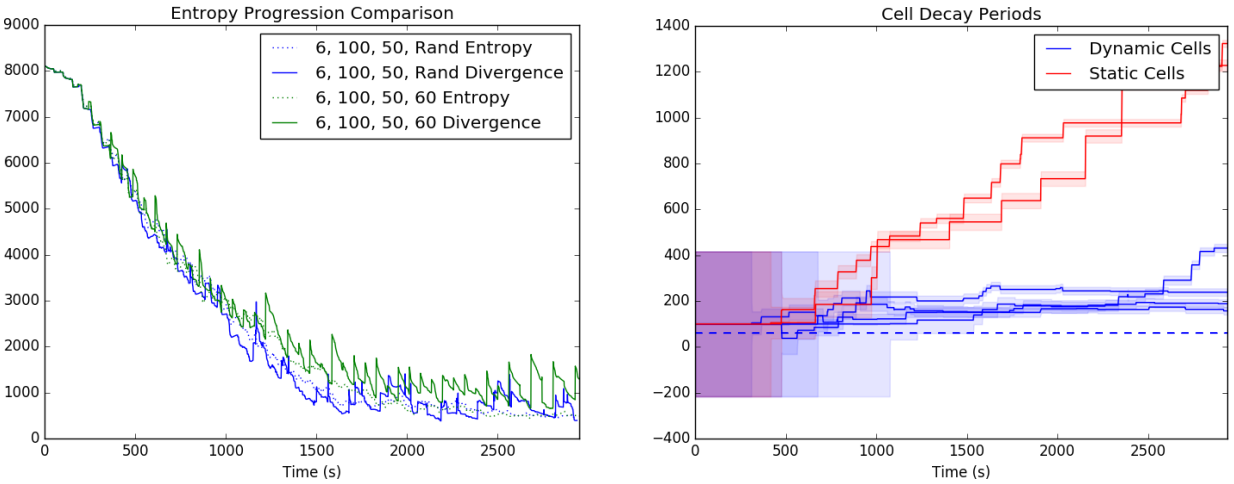


Figure 3.9: Evaluation for an environment with objects that move at a random frequency with parameters, $N_w = 6$, $d_{\text{hor}} = 100$, $d_{\text{adv}} = 50$. (Left) Comparison between aperiodic (dotted line) and periodic (solid line) dynamics. Spikes are no longer regularly spaced, but the convergence remains consistent. (Right) Decay periods differ enough to distinguish static and dynamic cells despite the unpredictable environment.

3.4 Conclusion

The method presented in this chapter addresses the base problem of deploying a multi-robot team to persistently map a dynamic environment. We pursue deployments that actively search for regions exhibiting discrepancies in observations over time and induce decay in the model confidence for these regions to promote revisitation at a rate corresponding to the perceived rate of change. In this manner, we are able to distribute our limited team to the places and times they will be most effective. This is evidenced by many runs over varying parameter configurations and environment dynamics that show the system consistently improve map accuracy by identifying and revisiting dynamic regions. Additionally, we showed how the system may be tuned to deploy more robots for faster convergence by increasing number of waypoints computed per horizon N_w or decreasing the horizon window d_{hor}.

Chapter 4

Allocating Limited Sensing Resources to Accurately Map Dynamic Environments

This chapter focuses on the problem of allocating limited resources to efficiently map an environment. First, we show how the *Waypoint Selection* objective is modified to factor in the fidelity of the dynamics model when prioritizing regions to visit. This is evaluated in a simple example, where independent cells are observed directly with no occlusion. We show that, using the updated objective, robots initially prioritize understanding static space, then transition to distributing observations among dynamic regions. This results in a better steady state model quality than MI, with fewer cells being abandoned once they are known with high confidence. Then, we present an extension to this approach which incorporates the updated objective into the multi-beam sensor model so that the utility function correctly balances CSQMI and the dynamics model fit.

4.1 Problem Definition

In this chapter, we seek to accurately model the physical structure of a dynamic environment where the number of sensing actions that can be taken at any given time is limited. When the

environment is initially unknown, the inherent dynamics must be discovered online before observations can be allocated appropriately. We choose to model the environment as a time-varying occupancy grid $m_t = \{m_{1,t}, \ldots, m_{N_c,t}\}$, where any cell i can be in the occupied $(m_{i,t} = occ)$ or free $(m_{i,t} = free)$ state at any time-step t. To model both the environment topography and how it changes, each cell stores a unique and independent HMM parameterized by the probability the cell is occupied

$$p(m_{i,t} = occ) = 1 - p(m_{i,t} = free) \in [0, 1]$$

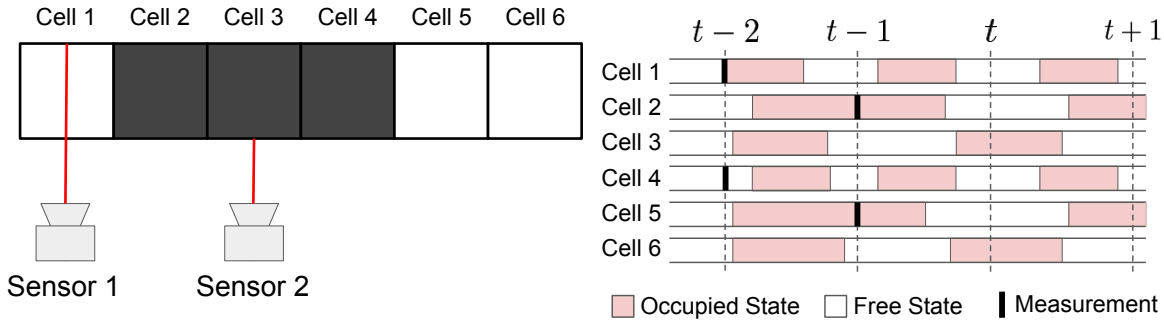
and the probabilities that the cell transitions between states

$$p(m_{i,t}|m_{i,t-1}) \in [0, 1] \quad \forall m_{i,t}, m_{i,t-1} \in \{free, occ\}.$$

Note that we assume the transition probabilities are locally stationary, where the value of $p(m_{i,t}|m_{i,t-1})$ is held consistent for all t in the absence of observations, but can change when the observed behavior disagrees with the model.

At each time-step, N_r sensing actions are taken that update the parameters of each HMM in m_t, as depicted in Fig. 4.1. Each sensing action produces an observation $o_j \in \{hit, miss\}$, where $\gamma_{\text{hit}} = p(o_j = hit|m_{i,t} = occ)$ and $\gamma_{\text{miss}} = p(o_j = miss|m_{i,t} = occ)$ describe the likelihood of sensing the corresponding observation conditioned on the cell being occupied. For this work, we assume each observation observes one cell directly with no occlusion and that there is no additional cost incurred when transitioning between sensing actions (as would occur when robots travel between locations to collect observations). While these assumptions are restrictive, the results we generate provide clearer intuition as to the influence of limitations and parameter choice in our approach directly.

The problem we address, then, is how to allocate the $N_r < N_c$ observations available at each time step to maximize the quality of the resulting model m_t, where quality is defined relative to



(a) Diagram of example scenario

(b) Occupancy state time-series

Figure 4.1: This work explores an example scenario where N_c cells are directly observed by N_r sensors and $N_r < N_c$. The objective is to appropriately allocate measurements, where N_r are available at each time-step t, such that the inherent environment dynamics can be accurately modeled.

the following three measures: 1) *Entropy*, expressed as:

$$H(m_t) = -\sum_{m_{i,t} \in m_t} p(m_{i,t}) \log_2 p(m_{i,t}) + (1 - p(m_{i,t})) \log_2 (1 - p(m_{i,t})), \tag{4.1}$$

provides a measure of model confidence as it decreases while the occupancy likelihood of all cells trend towards 0 or 1. 2) Model accuracy is expressed through the *Kullback-Leibler Divergence* between the learned environment model $p(m_{i,t})$ and an oracle model where $p_o(m_{i,t} = occ) = 0$ or 1 according to the ground truth state:

$$D_{KL}(t, m_t) = \sum_{m_i \in m} p_o(m_{i,t}) \log_2 \frac{p_o(m_{i,t})}{p(m_{i,t})} + (1 - p_o(m_{i,t})) \log_2 \left(\frac{1 - p_o(m_{i,t})}{1 - p(m_{i,t})} \right). \tag{4.2}$$

3) The responsiveness of our approach is expressed through the *response time* $d^i_{r,t}$ which denotes the amount of time that passes after cell i changes state before it is observed.

$$d^i_{r,t} = (\max(t^i_{obs} - t^i_{change}, t - t^i_{change}))dt, \tag{4.3}$$

where dt is duration between time-steps. In this work, we show how our approach outperforms standard techniques according to these measures despite the limited observations available.

4.2 Methodology

This work proposes an observation allocation strategy that scores cells based on the utility an additional observation would provide and chooses which N_r of N_c cells to observe each time-step based on that score. Section 4.2.1 describes the occupancy grid formulation in detail. Section 4.2.2 describes the utility measure which factors in the projected information gain and current *goodness-of-fit* to allocate observations. In Section 4.3, experiments highlight the benefits of this approach and evaluate the effect of varying parameters on the method's performance.

4.2.1 HMM Occupancy Grid

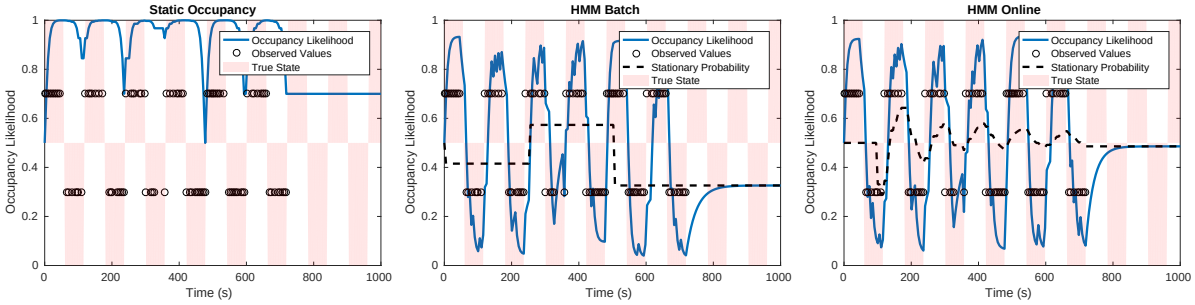
The model we use in this work is chosen to mimic the typical implementation of an occupancy grid. However, a static model regresses all measurements over time directly into the occupancy likelihood value, making it unsuitable for dynamic environments. As a result, historical bias will have the dominant effect on occupancy likelihood, eventually forcing the model to stay in one state, as depicted in Fig. 4.2a. In Chapter 3 we applied an exponential decay to drive the likelihood back towards 0.5; however, in this chapter we investigate a more principled means of representing the transition between states.

Meyer-Delius et al. [51] generalize the occupancy grid approach to account for environment dynamics by defining an HMM at each cell. In this manner, the occupancy likelihood of each cell evolves over a series of discrete time steps as an independent Markov process according to the recursive update function:

$$p(m_{i,t}|o_{1:n}) = \eta p(o_n|m_{i,t}) \sum_{m_{i,t-1}\in\{free,occ\}} p(m_{i,t}|m_{i,t-1})p(m_{i,t-1}|o_{1:n-1}), \tag{4.4}$$

where η is a normalization constant.

Typical HMMs [52] are updated using Expectation Maximization (EM) to maximize the



(a) Static Single Cell Model (b) Windowed Single Cell HMM (c) Online Single Cell HMM

Figure 4.2: A time series comparison of binary-state models given sparse observations. A static model has difficulty matching the true value when it oscillates, but the HMM versions learn transition probabilities that allow for the change. The batched HMM performs sufficiently well at the cost of storing a window of observations, but we see similar performance with the online HMM approach without needing to store as much data.

accuracy of the model over a series of observations:

$$p(m_{i,t}|m_{i,t-1}) = \frac{\sum_{\tau=1}^{t} p(m_{i,\tau-1} = \alpha, m_{i,\tau} = \beta | o_{1:n}, \theta)}{\sum_{\tau=1}^{t} p(m_{i,\tau-1} = \alpha | o_{1:n}, \theta)}$$

where $\alpha, \beta \in \{free, occ\}$ and θ are the HMM parameters learned via the standard forward-backward procedure. We can implement this method as an online approach by computing transition probabilities over a sliding window, resulting in the batched HMM update depicted in Fig. 4.2b. While this may be sufficient in most cases, it still requires storing a significant amount of data for each cell to accurately represent transition probabilities. Instead, we rely on the online version derived by Mongillo et al [53] to update parameters as each data point arrives, as shown in Fig. 4.2c.

Given that the measurements of cells arrive asynchronously, we define a regular time step and apply measurements with $p(o_n, m_{i,t}) = 0.5$ at each time step in between observations, effectively applying a no-observation update:

$$p(m_{i,t}|o_{1:n}) = \sum_{m_{i,t-1} \in \{free, occ\}} p(m_{i,t}|m_{i,t-1}) p(m_{i,t-1}|o_{1:n-1}), \tag{4.5}$$

which serves to drive the occupancy likelihood asymptotically towards the stationary probability:

$$p_{\text{stat}} = \frac{p_{o|f}}{p_{o|f} + p_{f|o}},$$

where $p_{o|f} = p(m_{i,t} = occ | m_{i,t-1} = free)$ and $p_{f|o} = p(m_{i,t} = free | m_{i,t-1} = occ)$. We can see this process in the progression of occupancy likelihood in Fig. 4.2, where the likelihood exponentially decays towards the stationary probability in the absence of observations.

4.2.2 Observation Utility Measure

Mutual Information is a useful measure of utility as it indicates the amount of information gained through an additional sensing action. It can be expressed in the form:

$$I_{MI}[m; o] = -\sum_{i,j} p(m_i, o_j) \log \frac{p(m_i)p(o_j)}{p(m_i, o_j)},$$

or interpreted as the change in entropy, or uncertainty, as the result of a measurement:

$$I_{MI}[m; o] = H[m] - H[m|o],$$

where $H[m]$ is the entropy of the map and $H[m|o]$ is the map entropy conditioned on the new observation.

Given that this expression of MI is designed for static environments, it is insufficient when the occupancy state is subject to change. While MI can be formulated to sufficiently address the influence of transition probabilities, the additional parameters make the computation complex and prohibitively expensive. If we used a dynamic model that guaranteed cells return to 0.5, as in Chapter 3, then cells would never stall at high likelihood and MI alone would suffice. However, since the HMM model can exhibit stationary probabilities with high confidence, we consider using the *goodness-of-fit* of the transition probability parameters to augment the utility

measure.

As a measure of the *goodness-of-fit*, we leverage the Pearson's χ^2 test to determine how well the transition probabilities match the history of observations. The χ^2 test is a used to evaluate whether or not an observed frequency distribution conforms to a theoretical distribution. Kalbfleisch and Lawless [54] show how this test can be used to determine how accurate the transition probabilities are modeled.

The χ^2 value is computed using:

$$\chi^2(t) = \sum_{ij \in \{occ, free\}} \frac{(n_{ijt} - e_{ijt})^2}{e_{ijt}},$$

where n_{ijt} is the number of observed transitions from state i to state j from time step $t - L_w$ to t and e_{ijt} is the expected number of transitions, where e_{ijt} can be expressed as:

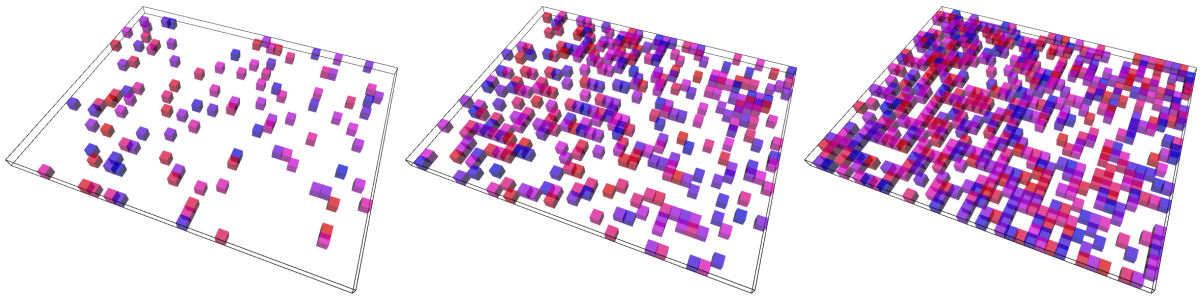
$$e_{ijt} = (p(m_{i,t})L_w)p(m_{i,t}|m_{i,t-1}).$$

The variable L_w denotes the window size, or the total number of time steps we search for transitions. One can interpret this value as a scaled, square-distance measure of the transition probability parameter. As we see more or less transitions than expected, we can expect the value of χ^2 to increase. If we use this value to modulate the observation rate of a cell, then cells that see the incorrect number of observations will receive higher priority.

To incorporate the *goodness-of-fit* into our allocation objective, we compose the weighted objective function:

$$u_i = I_{MI}[m_i; o_j] + \alpha \frac{\chi^2}{L_w}, \tag{4.6}$$

using the value α to balance the relative influence of the component objectives. Note that we normalize χ^2 by the window size L_w to mitigate its influence on the choice of α. This ensures that the effect of α is consistent as the window grows over time (until $t > L_w$) or when L_w is changed between runs, as is evidenced in Sect. 4.3.3.



(a) 10% dynamic (b) 30% dynamic (c) 50% dynamic

Figure 4.3: The set of test environments. Each dynamic cell is attributed an oscillation period between 300s and 2000s, drawn from a uniform random distribution. The percentage of dynamic cells varies between environments as listed.

4.3 Results

In this section, we compare the performance of the proposed allocation objective against alternatives and evaluate performance, as defined in Sect. 4.1, when parameters and conditions vary. Section 4.3.1 describes the general format of experiments and provides details on the evaluation criteria. Section 4.3.2 shows how the proposed approach outperforms random selection and the component objectives used alone. Section 4.3.3 highlights the change in performance as the relevant parameters are adjusted. Finally, Section 4.3.4 evaluates how performance varies as a function of environment conditions.

4.3.1 Experiment Setup

In this work, we consider scenarios where the occupancy state of cells evolve over time at regular intervals, where the state of each cell is represented by a square wave with the frequency f_i. As such, each cell will begin in the free state for a duration of $T_i/2$, where $T_i = 1/f_i$ is the associated square wave period, before transitioning to the occupied state for the same duration. Static cells are a special case where the cell exists in its initial state for all time. Excepting the illustrative example in Fig. 4.4, environments are constructed in a grid of 1073 cells with the set of dynamic cells and their T_i values sampled randomly. For each environment we test, a ratio of

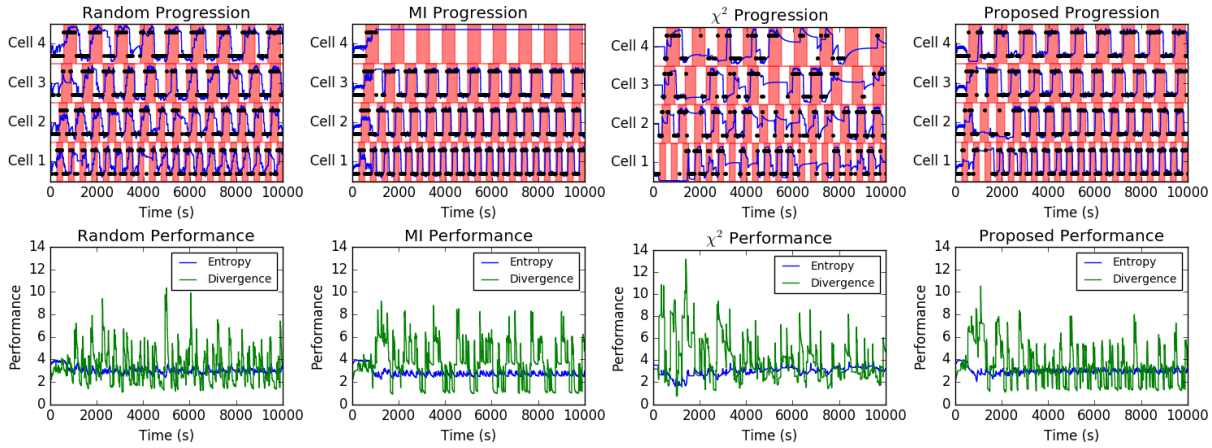


Figure 4.4: Comparison of allocation objectives. The top row shows a time series in the same manner as Fig. 4.2 where measurements are applied to one of four cells every 10 seconds and each row within the plot corresponds to a different HMM cell. The bottom row plots entropy and KL-Divergence for the set of cells in each approach. Random allocation serves as a baseline for comparison. Allocating based on MI results in accurate models, but can abandon cells once confidence is high, resulting in low entropy but high divergence. While χ^2 alone produces poor results, the proposed balanced objective can distribute observations based on the learned dynamics resulting in low entropy and a fast decrease in divergence whenever a change induces a spike.

dynamic to static cells is chosen and a random sampling of this fraction are assigned a value of T_i sampled from a uniform distribution between 300 and 2000 seconds. To perform a run, we allocate N_r observations to the cells chosen based on the specified allocation objective each time step, advancing $dt = 10s$ after each step. Cells that are observed are updated according to (4.4), while other cells evolve according to (4.5).

4.3.2 Comparison of Objective Functions

We evaluate performance according to the entropy (4.1), KL-Divergence (4.2), and response time (4.3) as formulated in Sec. 4.1 with one caveat. While we seek to reduce the response time for all cells, the value which best reflects our response to changes over the whole map is the

average maximum response time fraction

$$E[d_r] = \frac{1}{N_c} \sum_{i=1:N_c} \max_t \bar{d}^i_{r,t}.$$

over all cells, where we define the response time fraction as:

$$\bar{d}^i_{r,t} = \frac{d^i_{r,t}}{T_i},$$

interpreted as the response time normalized by the cell's associated period of change. As a cell changes state at each $0.5T_i$, values of $\bar{d}^i_{r,t} > 0.5$ indicate that cell i is receiving observations less frequently than changes occur. Providing the average maximum $\bar{d}^i_{r,t}$, as opposed to the maximum over all cells, permits outliers to reduce the quality of performance without being the dominant influence.

Figure 4.4 shows how the proposed allocation objective compares against alternatives. A small test of four independent cells is run for each allocation objective with $N_r = 1$. A random allocation objective provides a solid baseline, which exhibits a reasonable approximation of the time-series occupancy likelihood, but is still capable of missing changes. We further evaluate performance in Table 4.1, where tests are performed on the grid shown in Fig. 4.3b. The listed values for average, steady-state entropy μ_H, divergence μ_D, and response time fraction $E[d_r]$ for the random allocation case provide a target to exceed for our desired model confidence, accuracy, and response time. Additionally, the number of cells that go unobserved in the steady-state N_u highlights when an objective is not effectively distributing focus.

Allocation by MI, while appropriate for maximizing the confidence of individual cells, performs significantly worse than random allocation when applied to the environment model as a whole. Once an HMM reaches a stationary probability with high enough confidence, priority is given to cells whose observations suggest dynamics. As time passes without subsequent observations for a cell, it appears to be static and is ignored. This results in high accuracy for a

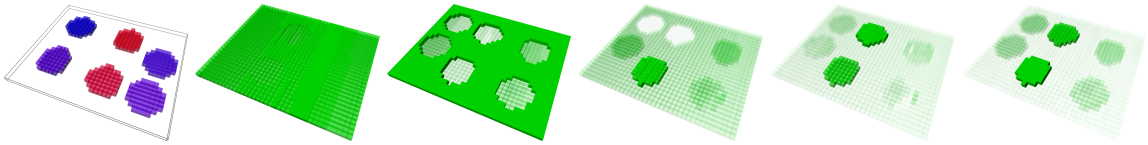
Table 4.1: Comparison of approaches for runs in the 30% dynamic environment (Fig 4.3b) for 10,000 seconds. Each row corresponds to a separate run with the listed parameters. The μ_H and μ_D terms correspond to the average entropy and KL-divergence of the last 2000s of operation. $E[d_r]$ is evaluated over the last 2000s, where a value of greater than 1.0 suggests that a majority of state changes go unobserved. N_u indicates the number of cells that are unobserved over the 2000s window. Note that the proposed approach outperforms all other allocation objectives, with significantly smaller μ_H, μ_D, and $E[d_r]$ values as well as a sufficiently small N_u value.

Objective	N_r	μ_H	μ_D	$E[d_r]$	N_u
Random	100	337.7	340.4	0.2644	0
Random	150	284.6	299.8	0.1634	0
Random	200	250.5	270.9	0.1238	0
MI only	100	572.5	438.2	0.9315	98
MI only	150	490.1	319.0	0.4498	38
MI only	200	392.8	264.9	0.4319	33
χ^2 only	100	276.1	320.8	0.7028	12
χ^2 only	150	249.0	282.2	0.4779	1
χ^2 only	200	229.3	245.9	0.3470	2
Proposed	100	297.6	325.7	0.1238	1
Proposed	150	222.3	217.3	0.0570	0
Proposed	200	**177.8**	**157.8**	**0.0333**	0

limited subset of cells, but low accuracy and confidence overall. We see the same trend in Table 4.1, where N_u is significantly higher than other approaches. As many cells go unobserved in the steady-state, the environment model exhibits a higher steady-state entropy and divergence as well as poor response times.

Alternatively, using the χ^2 *goodness-of-fit* measure alone results in measurements being distributed more evenly, but with less focus on ensuring high confidence. Measurements are drawn to cells that do not observe the number of transition their HMMs predict. However, measurements that occur immediately succeeding a change will drastically reduce model confidence for the cell, making it more difficult to recognize subsequent transitions.

In using the proposed objective function, we can infer that the χ^2 component serves to draw attention to cells whose dynamics are inappropriately modeled while MI recovers confidence lost in adapting to the newly recognized dynamics. Additionally, we note that our approach will bias early measurements towards static cells, as these produce no transitions, until the transition prob-



(a) Heatmap (b) $t = 400$ (c) $t = 1200$ (d) $t = 3000$ (e) $t = 6000$ (f) $t = 10000$

Figure 4.5: Evolution of priority as the environment model is learned when allocating 150 observations per time-step according to the proposed objective. A heatmap of the true dynamics is shown in a, where red indicates high frequency changes and blue indicates low frequency changes. The subsequent images reflect the relative number of measurements assigned to the cell over a 1000s time window preceding time t, with cells appearing more transparent the fewer measurements they are allocated over the window. Initially, measurements are spread evenly over all cells. Then focus is directed towards finding the static cells in the environment. Finally, measurements are distributed with a bias towards dynamics cells until the distribution accurately reflects the underlying dynamics.

abilities are low enough and confidence high enough that other cells draw focus. This process is highlighted in Fig 4.5, where our approach is applied to an environment with cells of corresponding dynamics artificially clustered to show how attention is allocated. By expending effort to initially identify static regions, we allow ourselves significantly more observations for dynamic regions, eventually settling on a distribution of attention that reflects the true environment dynamics.

4.3.3 Varying Parameters

Table 4.2 shows how performance changes relative to parameter choice. The first thing to note is that altering the time window L_w does not appear to have an appreciable effect on performance. As L_w simply defines the range of time in which we search for transitions, it is sufficient to define L_w large enough to account for the expected dynamic range and small enough to quickly forget data collected before the model has converged.

While we treat N_r as a parameter, it is more accurate to consider N_r as a problem constraint as the rate of observation collection is usually limited by the available number of robots and how quickly they can respond. As can be expected, increasing this value results in a drastic improvement in performance.

Table 4.2: Evaluation of performance as a function of parameter choice. Again, each run operates for 10,000 seconds on the environment in Fig. 4.3b. The objective performs as expected, with L_w not having a significant impact, N_r improving performance as more observations are available, and α mimicking the component objectives as it is tuned in either direction.

L_w	N_r	α	μ_H	μ_D	$E[d_r]$	N_u
1000	100	10	420.0	416.8	1.332	117
3000	100	10	392.9	451.0	1.632	130
5000	100	10	392.8	452.4	1.604	132
1000	150	10	311.0	343.0	1.202	85
3000	150	10	306.8	350.2	1.298	88
5000	150	10	305.2	350.6	1.369	88
1000	200	10	261.6	268.0	1.011	54
3000	200	10	256.0	267.7	1.052	53
5000	200	10	256.4	267.8	0.9191	50
1000	100	100	290.9	253.5	0.06865	0
3000	100	100	297.6	325.7	0.1238	1
5000	100	100	297.8	347.8	0.1504	1
1000	150	100	219.5	163.9	0.0434	2
3000	150	100	222.3	217.3	0.05703	0
5000	150	100	229.4	238.8	0.06176	0
1000	200	100	185.6	**136.7**	0.04156	2
3000	200	100	**177.8**	157.8	**0.03326**	0
5000	200	100	182.6	169.1	0.03602	0
1000	100	200	334.5	287.9	0.1551	0
3000	100	200	314.3	347.9	0.2521	0
5000	100	200	318.5	345.1	0.2763	0
1000	150	200	228.5	172.2	0.0362	0
3000	150	200	242.3	259.2	0.0742	0
5000	150	200	246.5	265.0	0.1001	0
1000	200	200	197.5	148.4	0.04378	2
3000	200	200	192.0	194.5	0.04598	0
5000	200	200	198.1	206.6	0.05416	0

Varying the objective weight predictably scales the influence of the associated component objectives. As α becomes smaller, performance mimics a pure MI objective with poorer performance measures and a drastic increase in steady-state N_u. Larger α values mimic the pure χ^2 objective, with reasonable performance, but less than when properly balanced.

Table 4.3: Comparison of performance for different concentrations of dynamics. Tests are performed on the environments depicted in Fig. 4.3 for 10,000s each. The weight α is varied for each run, while the remaining parameters are fixed to $L_w = 3000$ and $N_r = 150$. Environments with less dynamics cells prefer an allocation objective closer to pure MI while more dense dynamic cells lean towards pure χ^2.

α	Dyn %	μ_H	μ_D	$E[d_r]$	N_u
10	10	114.4	**46.56**	**0.01938**	0
50	10	96.60	49.41	0.02549	0
100	10	**94.87**	57.34	0.03464	0
150	10	99.82	67.95	0.04216	0
200	10	106.3	80.01	0.06319	0
10	30	306.8	350.2	1.298	88
50	30	232.3	235.9	0.1475	6
100	30	**222.3**	**217.3**	**0.05703**	0
150	30	230.8	235.1	0.05913	0
200	30	242.3	259.2	0.07420	0
10	50	442.5	704.2	2.286	247
50	50	**383.3**	551.3	0.5649	58
100	50	388.7	479.9	0.1110	7
150	50	395.0	**472.5**	**0.09815**	1
200	50	401.2	482.4	0.1074	0

4.3.4 Varying Environments

To evaluate performance as environment conditions change, we run simulations on randomly generated environments of varying density of dynamic cells. Figure 4.3 shows the three environments tested in this work, with 10%, 30%, and 50% of cells oscillating between the occupied and free state with periods drawn from a uniform distribution between 300s and 2000s. As expected, the more the environment exhibits change, the harder it is to model. However, we note that the best choice of α appears to vary depending on the environment. More static environments prefer an approach closer to pure MI, while environments with dense dynamics rely more heavily on the χ^2 objective.

4.4 Discussion

There are two main limitations when using this approach that we note in this section. First, the approach is sensitive to an accurate determination of the number of observed transitions, n_{ijt}. If the sensor noise is significant, false positives can be registered, which promote further measurements of false positives, creating a self-sustaining loop that biases priority towards cells that produce noisy measurements. Strategies for determining n_{ijt} in the presence of noise will allow this approach comparable performance that is robust to noisy measurements.

Second, cells with low transition probabilities will be revisited at very low frequencies. This trait is acceptable when the dynamics do not change, but less so when a static region becomes dynamic at some future time (e.g. a region is designated for parking). We expect that inducing decay in the confidence of our dynamics model as well as the occupancy likelihood will serve to promote revisitation of cells that have been considered static for too long a duration.

4.5 Conclusion

This work presents a strategy for allocating limited sensing resources to mapping a dynamic environment. As expected, the best performance is achieved when the distribution of observations reflects the dynamics being observed, which is most easily achievable when the dynamics are learned quickly. The results show that our proposed approach outperforms both the pure MI and χ^2 objectives, which suggests that environments can only be modeled accurately if there is a balanced effort between reinforcing poorly modeled dynamics and reducing uncertainty in occupancy likelihood. When this balance exists, we are capable of ensuring an average response time significantly less than $0.5T_i$ when as little as 14% of cells are observed each time step. While this value will vary based on the dynamic properties of the environment, future works can extend these results to provide performance guarantees relative to the available number of robots and their limited energy capacities while considering the influence of travel time.

Chapter 5

Deployment Planning for Online Mapping of Dynamic Environments

In this chapter, we combine the framework described in Chapter 3 with the HMM-based environment modeling strategy and associated waypoint selection objective in Chapter 4. The new objective is extended to work with a multi-beam sensor (and multiple observations with the same sensor) to allow for seamless integration within the updated framework. Additionally, we investigate an updated planning strategy that considers multiple planning iterations per horizon as well as alternative *Waypoint Assignment* formulations, both to improve performance and to ensure the discovery of feasible routes when the full set of N_w waypoints is not reachable by available robots.

The updated system we propose in this chapter is outlined in Figure 5.1. As before, a planning cycle begins with the *Waypoint Selection* algorithm, which iteratively selects the N_w most informative waypoints from a set of reachable locations. However, in the updated version of our approach we allow for multiple planning cycles each horizon, drawing a new set of N_w waypoints each cycle. This change allows us to extend plans up to the limits of the horizon duration and/or robot energy capacity. For efficient computation of these planning cycles, it is convenient to maintain a pool of candidate waypoints, greedily updating the cost relative to the previously

selected waypoints and selecting the next best candidate. This process is described in detail in Sect. 5.3.

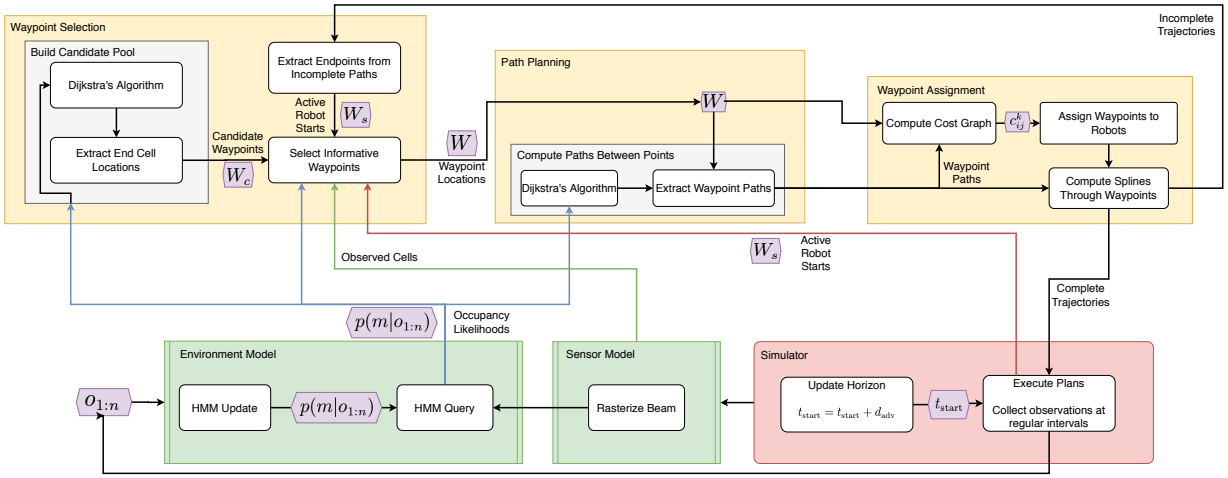


Figure 5.1: System Diagram. The planning pipeline (yellow blocks) generates plans executed by the simulation environment (red blocks), which generates observations for the observation model (green blocks) that provides map information to the planning pipeline. Note that, after the *Waypoint Assignment* block, the planner loops back to *Waypoint Selection* to extend plans if possible. Otherwise, the plans are sent to simulator for evaluation.

Once waypoints are selected, the *Path Planning* algorithm, Sect. 5.4, generates a set of shortest-distance paths between all pairs in the set of N_w waypoints. The focus, at this stage, is to provide a fast estimate of cost and feasibility to inform *Waypoint Assignment*, as opposed to investing in high-quality paths that may never be traversed. To simplify collision avoidance, we assume all rapid changes due to transient motion occur outside a user-defined safety region, designated by a radius of r_{safe} around the starting location, x_s, and all locations above a height of h_{safe}. Given the disaster response application, this is a reasonable assumption that allows *Path Planning* to assume an accurate *Environment Model*. We continue to rely on Dijkstra's algorithm to compute energy-feasible paths, but guarantee dynamic feasibility, in terms of velocity and acceleration limits, through the use of motion primitives to describe the motion between cells.

While the *Waypoint Assignment* algorithm, Sect. 5.5, still relies on solving an ILP-based graph of the waypoints and corresponding pair-wise paths it has been simplified to remove the time window constraints, reflecting a reliance on the assumption that our environment is locally static. These constraints are of less significance without a time-dependent utility measure, as the exact time an observation is collected will not heavily impact overall performance if environment changes occur infrequently over many horizons. Additionally, we present several ILP alternatives and discuss their effect on the overall performance.

Once the routes are generated, trajectories are formed by concatenating paths between the waypoints along each route. If these trajectories fill the planning horizon, then they can be sent to the robots and resulting observations can be integrated into the *Environment Model*. However, if it is possible to add more waypoints, we repeat the planning cycle as detailed in Algorithm 3. The key difficulty lies in determining the set of pseudo-waypoints W_s, which serve as virtual nodes in *Waypoint Assignment* that robots must visit first to ensure continuity across planning iterations. At the first planning iteration each horizon, shown in line 4, this set of waypoints corresponds to the next set of waypoints each robot intends to visit after t_{start} in the current plan. Further waypoints are discarded and active robots start from these locations. In subsequent planning iterations, starting at line 13, W_s includes the last assigned waypoint for each active robot, allowing *Waypoint Assignment* to append new waypoints to each robot's plan. The intention is to simply extend each plan until robots expend their capacity or the planning horizon ends.

Under the traditional occupancy grid formulation, the *Environment Model* would simply process these observations to update the occupancy probability of all observed cells. However, our model, described in Sect. 5.1, incorporates the Hidden Markov Model approach introduced in Sect. 4.2.1, where we leverage existing techniques to learn an HMM online by incorporating irregularly timed observations, allowing the confidence in the model to decay as time progesses with no observations.

Algorithm 3 Extend Plans over a Finite Horizon

procedure COMPUTEHORIZON($t_{\text{start}}, d_{\text{adv}}, P$)
  $W_s \leftarrow \emptyset$ ▷ First waypoint to be used in updated plans for active robots
  for each $p_i \in P$ **do** ▷ Collect the set of waypoints first visited after t_{start}
    $W_s \leftarrow W_s \cup \{\text{nextVisitedWaypoint}(p_i, t_{\text{start}})\}$
  end for
  while 1 **do**
    ComputeSchedules(t_{start}, W_s, P) ▷ Main loop to compute schedules
    if isExtendable($P, t_{\text{start}} + d_{\text{adv}}$) **then**
      break
    else
      $W_s \leftarrow \emptyset$
      for each $p_i \in P$ **do** ▷ Collect the set of waypoints last visited after t_{start}
        $W_s \leftarrow W_s \cup \{\text{lastVisitedWaypoint}(p_i, t_{\text{start}})\}$
      end for
    end if
  end while
  $t_{\text{start}} = t_{\text{start}} + d_{\text{adv}}$
end procedure

5.1 Environment Model

The system described in this chapter requires a spatiotemporal representation of occupancy probability, but has a large degree of flexibility regarding what form this representation takes. While it is possible to consider other forms of probabilistic environment models, for this work we continue to utilize the HMM occupancy grid described in Sect. 4.2.1 as it allows for a simple extension to the complete system described in Chapter 3, which incorporates sensors with long-range beams and planning that considers movement limitations. Recall the HMM update function for individual cells:

$$p(m_{i,t}|o_{1:n}) = \eta p(o_n|m_{i,t}) \sum_{m_{i,t-1} \in \{free, occ\}} p(m_{i,t}|m_{i,t-1}) p(m_{i,t-1}|o_{1:n-1}), \tag{5.1}$$

which describes how an observation is incorporated into the environment model. Although we are no longer observing cells individually, the process by which a multiple-cell observation informs each single-cell model does not change.

At the beginning of each planning horizon, the HMM for each cell is queried for its value at time t_{start} by advancing the model by time steps according to (5.1) and incorporating the appropriate observed states. We then use the occupancy likelihoods for each cell at this time for all subsequent queries, enforcing a locally static assumption. This assumption can be restrictive for excessively long horizons, but obviates the need for a computationally intractable evaluation of time-varying observation utility (as described in Sect. 5.2.2).

5.1.1 Computational Complexity

As we have elected to use an online approach to environment modeling, as described in Sect. 4.2.1, both the updates and queries can be executed in constant time ($O(1)$). If we had used the windowed HMM update, complexity would be dependent on window size, but the online approach is both faster and more reactive to changes in the environment. Each individual cell update evaluates a constant number of values, which is repeated for each observed cell for the number of time steps between the current and previous observation.

5.2 Sensor Model

The sensor model used in this work represents a range sensor that measures occupancy by extending beams into the environment, each collecting a set of distance measurements at various angles. The traditional approach to modeling range sensors updates each cell a beam passes through as observed in the *free* state and the cell in which a beam impacts an obstruction is observed in the *occupied* state with each observation action. We apply a simplification, applicable in a simulated environment, in which each cell a beam passes through is updated according to its ground truth state. This circumvents the case where a cell is only partially occupied and a beam passing through the empty space provides an inaccurate observation of the cell state. While this assumption may be somewhat restrictive when considering real-world conditions, it is possible

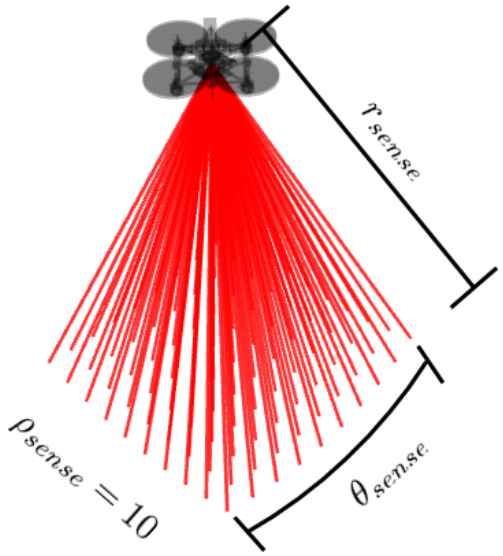


Figure 5.2: The variables associated with the *Sensor Model*. The maximum sensor range, r_{sense}, describes the maximum distance a beam is extended, the field of view, θ_{sense}, describes how wide the sensor scans vertically or horizontally, and ρ_{sense} describes the number of beams per vertical or horizontal scan.

to devise a model for $p(o_n|m_{i,t})$ that appropriately accounts for this pathological case without requiring any change to the overall environment model.

Figure 5.2 shows the parameters used to describe the sensor model. Given that range sensors typically scan vertically and horizontally, the key variables of interest are:

- The maximum beam range r_{sense}.
- The scan width θ_{sense}.
- The number of beams per vertical or horizontal scan ρ_{sense}.

While both θ_{sense} and ρ_{sense} can have separate definitions for each axis of the scan, we choose the same scan width and number of beams for both the vertical and horizontal axes to simplify the evaluation in Chapter 6. Although we define a specific sensor configuration to facilitate evaluation, other configurations would be equally applicable with no appreciable change to the overall system.

5.2.1 Computational Complexity

To determine which cells a beam passes through, we use a ray-tracing algorithm presented by Amanatides and Woo [55] that works by iteratively stepping along grid cells from the ray start to the ray end, ending at the map border if the beam range has yet to be exceeded. Thus, the complexity of determining the cells an observation intersects is $O(BC)$, where B is the number of beams in an observation action and C is the number of cells each beam intersects.

5.2.2 Observation Utility

Given the sensor model chosen for this approach, we define a utility function to prioritize observations that best improve the accuracy of our environment model. The static components of the model are represented using the occupancy likelihood parameters, while the dynamics are captured by the transition probabilities. An appropriate utility function will account for both when determining where to allocate observations. As such, we define a utility function:

$$u_i = I_{CS}[m_i; o_j] + \alpha \frac{\chi^2}{L_w}, \tag{5.2}$$

which balances a information theoretical objective $I_{CS}[m_i; o_j]$ against a *goodness-of-fit* for the dynamics model χ^2 using a user-defined weighting parameter α. The rest of this section will elaborate on the components of this mixed objective.

The first component of the proposed objective uses the CSQMI-based utility score introduced in Sect. 3.2 to prioritize waypoints that maximize information gain relative to the map in its current state. Recall that CSQMI operates as a measure of the reduction of entropy that occurs as a result of the observation action, expressed in the form:

$$I_{MI}[m; o] = H[m] - H[m|o] = -\sum_{i,j} p(m_i, o_j) \log \frac{p(m_i)p(o_j)}{p(m_i, o_j)}.$$

For a static environment, or in cases where observations can be collected immediately, selecting

observations based on this score is the most efficient way to increase model confidence.

However, CSQMI alone is insufficient to define the utility of a measurement for our environment model that also tracks dynamics. To prioritize the observation of regions that are poorly modeled with respect to dynamics, we augment the utility function with a *goodness-of-fit* measure introduced in Chapter 4. Recall that this measure is expressed the form:

$$\chi^2(t) = \sum_{ij \in \{occ, free\}} \frac{(n_{ijt} - e_{ijt})^2}{e_{ijt}},$$

where n_{ijt} is the number of observed transitions from state i to state j from time step $t - L_w$ to t and e_{ijt} is the expected number of transitions, where e_{ijt} can be expressed as:

$$e_{ijt} = (p_{\text{stat}} L_w) p(m_{i,t} | m_{i,t-1}).$$

The variable L_w, denotes the window size, or the total number of time steps we search for transitions.

Two final considerations for our utility function are: first, how to address multiple observations of the same cell and second, how to incorporate occlusion into our objective. We account for these by considering the visibility of cells a beam passes through. Visibility is expressed as:

$$\text{vis}_b(c_i) = \prod_{j < i \in b} (1 - p(c_j)) \tag{5.3}$$

where $j < i$ defines the set of cells in c that precede i along beam b's direction vector. Using this value, we can determine the expected χ^2 value of visible cells using the equation:

$$E[\chi_i^2] = \sum_i \text{vis}_b(c_i) \chi_i^2. \tag{5.4}$$

While this formulation addresses occlusion, we must also consider how multiple observations from different beams can be incorporated into the utility function. If we were to sum these

contributions directly, then the utility function would favor multiple views of the same regions despite the fact that multiple observations provide limited benefit. To preserve the submodular property of our objective, we use the maximum visibility across all beams for each cell:

$$E_o[\chi_i^2] = \sum_{i \in o} \max_{b \in o} \text{vis}_b(c_i)\chi_i^2, \tag{5.5}$$

which has the benefit of considering the utility of observing each cell only once at the cost of ignoring the potential benefit of viewing a cell multiple times. Incorporating this form of expected χ^2 into the utility function, the final form of our waypoint selection objective function is expressed as:

$$u_i = I_{CS}[m_i; o_j] + \alpha \frac{E_{o_j}[\chi_i^2]}{L_w}. \tag{5.6}$$

5.2.3 Computational Complexity - Utility Function

Charrow et al. [27] proposes an efficient approach to computing CSQMI with an evaluation of the computational complexity. It is shown that by assuming cell lengths are larger than the variance in beam measurement the components corresponding to joint probabilities between cells in (3.2) can be simplified such that the CSQMI of a single beam can be computed in $O(C)$, where C is the number of cells the beam intersects. When multiple beams are considered simultaneously the complexity becomes $O(TBC)$, where T is the number of poses from which an observation is taken and B is the number of beams cast in a single sensing action. Note that this accounts for multiple observations taken, as long as the map does not change between observations.

We can apply a similar analysis to the χ^2 component of the utility function, given that we are generating another submodular objective that modifies the contributions of observing cells based on the source beam. As each cell from the set of C cells provides a χ^2 measure which is multiplied against each of B beams in observations from T poses, we again have a complexity of $O(TBC)$, which when added to the CSQMI complexity yields a cumulative complexity of $O(TBC)$. From this result we can conclude that, while the joint objective is more costly than

the single objective, is scales no more poorly than the single objective relative to the number of cells or observations.

5.3 Waypoint Selection

The updated version of our *Waypoint Selection* approach has been split into two component algorithms to accommodate building a pool of candidates to iteratively draw from each planning cycle. Algorithm 4 first builds the candidate pool once per horizon, then Algorithm 5 selects N_w waypoints from this pool each planning iteration. Algorithm 4 is essentially a single iteration of Dijkstra's algorithm to determine the set of feasible paths from the starting location, x_s, followed by storing the set of end locations as candidate waypoints. This process begins with the DijkstraReachables function, which uses the path planning approach (described in Sect. 5.4) to find the set of all cells reachable from the start location x_s and store their centroids in C_l. Each waypoint is paired with a utility score, initialized as ∞, and stored in the candidate pool W_c, from which we draw the set of N_w points each planning iteration.

Algorithm 5 then performs this drawing while lazily updating the scores relative to the currently selected set W. This output set is initialized with the set of pseudo-waypoints selected at the beginning of each planning iteration, as expressed in Algorithm 3, and updated each iteration of the while loop in line 3 as each new waypoint is added. The basic loop starts with the SortByScore function by sorting the candidate pool W_c relative to the utility score such that the first drawn waypoint is the highest scoring waypoint. Then, we iterate through each waypoint in the loop, updating the associated score stored in W_c relative to the set W, until we reach a waypoint with a score less than the current best score. Given that the utility score is monotonically non-decreasing with the addition of new measurements, the score of any given waypoint w will never increase with the addition of more waypoints to W. We can interpret this as:

$$\text{computeUtility}(m, w, t_{\text{start}}, W) \geq \text{computeUtility}(m, w, t_{\text{start}}, W') \quad \forall \quad W \subset W',$$

which suggests that there is no merit to recomputing the score of waypoints whose last computed score is less than the current best. Also, given that W_c is sorted according to last computed utility score, the current best waypoint w_{next} is the highest scoring waypoint of the current iteration. Thus, the algorithm can exit the loop and transition w_{next} from W_c to W, continuing this process until N_w waypoints have been added to W. In this manner, we are able to select the N_w best waypoints in each planning iteration without significantly increasing the computational complexity.

Algorithm 4 Build waypoint candidate pool

procedure BUILDCANDIDATEPOOL(m, x_s)
    $C_l \leftarrow$ DijkstraReachables(m, x_s) ▷ Candidate locations
    $W_c \leftarrow W_s$ ▷ Scored candidate waypoint
    for each $c_i \in C_l$ **do**
        $W_c \leftarrow [c_i, \infty]$
    end for
    return W_c
end procedure

Algorithm 5 Lazy Waypoint Selection

procedure WAYPOINTSELECT($m, x_s, t_{\text{start}}, N_w, W_c, W_s$)
    $W \leftarrow W_s$ ▷ Waypoint position set
    while $|W| < N_w$ **do**
        SortByScore(W_c)
        for each $[w, s] \in W_c$ **do**
            if $s < s_{\text{next}}$ **then** ▷ Exit if utility is less than the current best
                break
            end if
            $s =$ computeUtility$(m, w, t_{\text{start}}, W)$ ▷ Update utility relative to W
            if $s > s_{\text{next}}$ **then** ▷ Track the highest scoring waypoint
                $s_{\text{next}} = s$
                $w_{\text{next}} = w$
            end if
        end for
        $W_c = W_c \setminus w_{\text{next}}$ ▷ Move waypoint from candidate to output
        $W = W \cup w_{\text{next}}$
    end while
    return W
end procedure

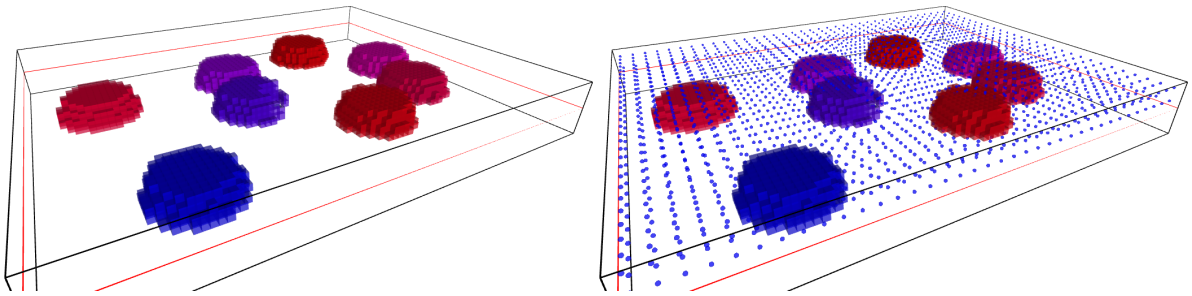


Figure 5.3: (Left) An example of the testing environment. The dimensions are defined by the black wireframe box, with the safety height defined by the red square. Dynamic cells are shown differentiated by color relative to frequency of change, while static cells are not shown in the figure. Note that all dynamic cells are below the safety height. (Right) The example environment with the planning grid nodes superimposed as blue spheres. In this figures, the planning grid is half the resolution of the environment model to simplify the search algorithms. Note that all planning nodes are above the safety height.

Additionally, to allow for a higher resolution occupancy grid, we select waypoints from a low resolution planning grid that is superimposed on the environment. This is a similar approach to subsampling potential candidates that ensures an even spread of potential observations over the environment. While many cells will not be visited with this approach, a sensor with an appropriately small beam separation angle will be able to view any cell in the environment model from a nearby planning grid node. An example of this separation of grid specifications is shown in Fig. 5.3, where a planning grid is superimposed on an environment model with twice the resolution.

5.3.1 Computational Complexity

As Algorithm 4 consists of a single application of Dijkstra's algorithm, the complexity is $O((E+V)\log(V))$, as described in detail in Sec. 5.4.2. The complexity of Algorithm 5 is largely a function of how many utility score evaluations must be performed. In the worst case, each candidate waypoint must be re-evaluated every time a waypoint is selected. This results in $N_w N_l$ evaluations of utility score, where $N_l = |C_l| = |W_c|$ is the number of candidate waypoints. This can become expensive as the number of candidates, drawn from the set of cells reachable

from x_s, will be large for any environment model of significant size or resolution. The algorithm shows that each evaluation of utility considers $|W| + 1$ poses, but it is possible to compute the relative score of adding an additional waypoint if one stores the likelihood of intersecting cells with observations from W. As such, the utility computed each iteration only costs $O(BC)$, resulting in a worst case complexity for *Waypoint Selection* of $O(N_w N_l BC)$. Adding to this the complexity of sorting W_c each cycle of the while loop in line 4 yields a total complexity of $O(N_w N_l BC + N_w N_l \log N_l)$. In practice, however, the lazy-greedy approach only evaluates a small subset of candidates per iteration, resulting in a significant decrease in computation time.

5.4 Path Planning

Path Planning between selected waypoints is handled using Dijkstra's algorithm in the same manner as presented in Chapter 3. By computing Dijkstra's paths from each waypoint to each reachable cell, we can extract all possible paths that could be traveled from any permutation of waypoint visitations. The proposed system employs the standard implementation of Dijkstra's algorithm, with a few adjustments to account for the specific application. First, collision constraints are handled using the learned environment model. As we assume a locally static environment, we evaluate the model at the planning horizon start time t_{start} and plan paths only through cells with a high probability of being free: $p(m_{j,t_{\text{start}}}|o_{1:n}) < \gamma_{\text{free}}$, where γ_{free} is a user-defined parameter. Second, we restrict movement to a safe region where we assume no fast-moving obstacles exist. This is modeled as a cylindrical region surrounding x_s of radius r_{safe} and all cells above a height h_{safe}. This restriction is not significantly limiting when operating with quadrotor vehicles in a disaster scenario, as the majority of dynamic obstacles will exist at street level and we can safely assume high model accuracy above a certain height.

5.4.1 Trajectory Generation

To enforce dynamic constraints, we build the splines used to describe robot motion from a motion primitive library. It is trivial to define trajectories that can safely move robots from one cell to the next and concatenate these primitive motions to generate a spline from the computed Dijkstra's path. In the proposed approach, we compute polynomial splines that minimize jerk relative to endpoint constraints, which is a simple solution of a set of linear equations of the form:

$$\min_{f} \quad \int_0^T \left(\frac{d^3 f(t)}{dt^3} \right)^2 dt \tag{5.7}$$

$$\text{s.t.} \quad \frac{d^k f(\tau)}{dt^k} = x_k(\tau) \qquad \forall \tau = 0, T; k = 0, 1, 2, \tag{5.8}$$

where $f(t)$ is a nth order polynomial, $x_k(0)$ is the kth derivative starting condition (position: $k = 0$, velocity: $k = 1$, acceleration: $k = 2$), with $x_k(T)$ corresponding to the kth derivative ending condition, and T as the time it takes to travel from start point to end point at the user specified constant speed, v_{travel}. Generating a full set of primitives for the proposed approach requires defining all possible motions a robot may take through a cell. These include:

1. motions that accelerate from 0 to v_{travel} from the cell centroid to each exit point
2. motions that move from each entrance to each exit starting and ending at v_{travel}
3. motions as in 2, but which pass through the centroid at $\tau = 0.5T$
4. motions that decelerate from v_{travel} to zero from each entrance to the centroid

Note that each of these motions is constrained to zero acceleration at each endpoint ($x_2(0) = x_2(T) = 0$). Given these motions, a path begins with a motion taken from 1, concatenates motions from 2 with motions from 3 used when passing through a waypoint, ending with a motion taken from 4. A visualization of these primitives is provided in Fig. 5.4.

While it is possible to formulate dynamic constraints that enforce limits across the full spline by adding inequalities for values of $\tau = \beta T$ for $0 < \beta < 1$, significantly increasing the number of constraints will make the problem harder to solve and may result in ill behaved trajectories.

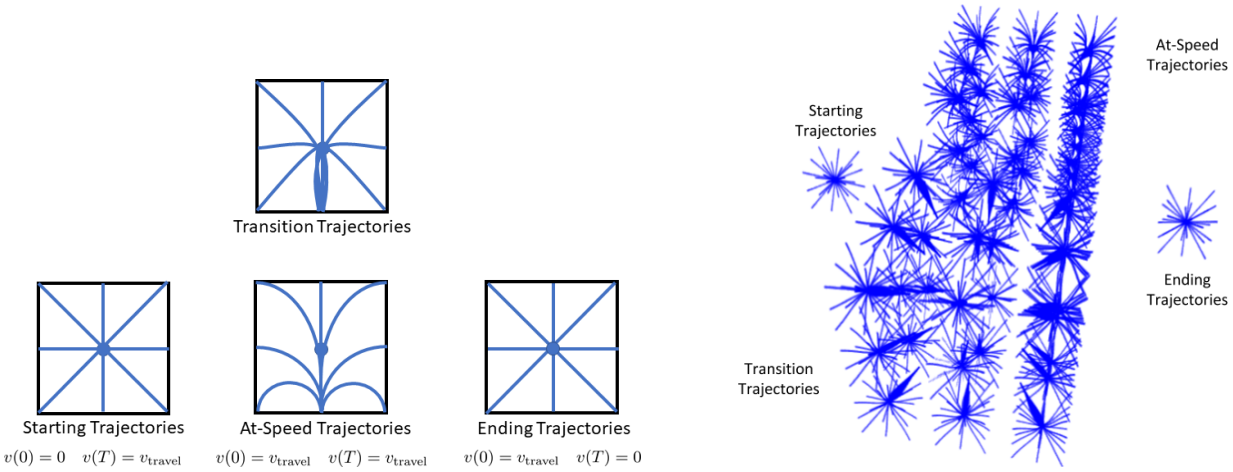


Figure 5.4: (Left) A sample set of 2D primitives. The Starting trajectories accelerate to v_{travel} from the center to the corners and edges. The At-Speed trajectories enter from corners and edges and exit from corners and edges at v_{travel}. Transition trajectories perform the same action, but are constrained to pass through the centroid. Ending trajectories enter from corners and edges, decelerating to zero at the centroid. (Right) The full set of 3D primitives used in this work. The motion primitive library can be maintained at a manageable size if motions are constrained to a single speed at the transition points.

It is sufficient to compute trajectories for a given value of v_{travel} then lower v_{travel} if any dynamic constraints are violated. In this manner, any path generated from the planning grid and constructed by concatenating motion primitives will, by definition, be smooth and adhere to velocity and acceleration limits. Note that it is possible to generate the motion primitive library using any preferred method without affecting the system process as long as the primitives are dynamically feasible and defined for all required motions.

5.4.2 Computational Complexity

A single Dijkstra's operation has been shown to compute in $O((E + V)\log(V))$ time for a graph of V vertices and E edges if the graph is stored as an adjacency matrix and paths are grown according to a priority queue. As Algorithm 2 operates on a regular 3D grid, with the neighborhood determining adjacency, and a priority queue stored in Q, this complexity applies. Computing paths for all waypoints requires $O(N_w * (E + V)\log(V))$, where V is the number of cells reachable from the starting cell, s, and E is proportional to V relative to the neighborhood

size.

5.5 Waypoint Assignment

Given the selected waypoints and the computed paths, we next formulate a Vehicle Routing Problem (VRP) and solve for optimal routes. Traditionally, VRPs are expressed in graphical form $G = V, E$, where each node in V corresponds to a waypoint and each edge in E corresponds to a path between waypoints. Optimal routes through G are cycles joined at the node corresponding to the starting location that, together, visit all waypoints exactly once. Additionally, we constrain the energy expended in each route to ensure long-term feasibility. This results in a variant of the VRP known as the Capacitated VRP, which includes a constraint that compares the resources expended traversing each edge and visiting each node to the assigned robot's capacity.

There are three main approaches to solving this problem formulation that we consider in this paper. A greedy, finite-horizon approach serves as a baseline for comparison. Then we present two Mixed-Integer Program (MIP) formulations with differing objectives and slightly different constraints to evaluate which better fits the proposed problem. Finally, we discuss how these MIP formulations might be augmented to ensure feasible problem configurations even when there are more waypoints than can be assigned given team size and capacity limitations.

5.5.1 Greedy Assignment

First we establish the greedy finite-horizon assignment approach, which differs from pure greedy assignment in that waypoints are selected as per Sect. 5.3 in batches and iteratively assigned to robots, extending active plans if they exists and have excess energy capacity. Then the horizon is extended and the process repeats. This allows us to implement a greedy approach to assignment while utilizing the horizon formulation and waypoint selection algorithm. The approach used is expressed in Algorithm 6

Algorithm 6 Greedy Waypoint Assignment

1: **procedure** WAYPOINTASSIGN(x_s, N_r, W, A)
2: $P_{out} \leftarrow \emptyset$ ▷ Solution path set
3: $P \leftarrow A$ ▷ Set first $|A|$ paths to currently active paths
4: **for** $i = |A| : N_r$ **do**
5: $P_i \leftarrow \{x_s\}$ ▷ Initialize remaining paths to start location
6: **end for**
7: **while** $|W|$ **do**
8: $p_{next} = \min_{p \in P}$ pathEndTime(p) ▷ Earliest ending path
9: $L_{rem} = L -$ (pathEndTime$(p_{next}) -$ pathStartTime$(p_{next}))$ ▷ Remaining capacity
10: $W_{reachable} = \{w \in W | L_{rem} >$ Cost$(p_{next}(end), w) +$ Cost$(w, x_s)\}$
11: **if** $W_{reachable} = \emptyset$ **then** ▷ No more reachable waypoints, end path
12: $p_{next} \leftarrow x_s$
13: $P_{out} \leftarrow p_{next}$
14: $P \leftarrow P \setminus p_{next}$
15: **else** ▷ Append next closest waypoint
16: $w_{next} \leftarrow \min_{w \in W_{reachable}}$ Cost$(p(end), w)$
17: $p_{next} \leftarrow p_{next} \cup w_{next}$
18: $W \setminus w_{next}$
19: **end if**
20: **end while**
21: **for each** $p \in P$ **do**
22: $p \leftarrow x_s$
23: $P_{out} \leftarrow p$
24: **end for**
25: **return** P_{out}
26: **end procedure**

The algorithm takes in as parameters the start location, x_s, a desired number of robots to be deployed, N_r, the selected waypoints, W, and the active paths A. The outputs are a set of paths, P_{out}, represented as a list of waypoint locations. Lines 2-6 initialize the set of paths being processed P as the currently active paths with remaining capacity, generating new paths starting at x_s until $|P| = N_r$. Lines 7-20 greedily assign waypoints from W until there are none left. Once it is determined that a path can no longer be extended, paths are extracted from P and the process continues (11-15). Then, once all waypoints are exhausted, all remaining paths in P are routed back towards x_s and appended to P_{out}.

5.5.2 Minimum Cost VRP

Recall that the typical VRP formulation constructs a Mixed Integer Program of the following form:

$$\min_{x} \sum_{k \in K} \sum_{(i,j) \in A} c_{ij} x_{ij}^{k} \tag{5.9}$$

$$s.t. \sum_{k \in K} \sum_{j \in V} x_{i,j}^{k} = 1 \qquad \forall i \in V \tag{5.10}$$

$$\sum_{j \in V \cup S_g} x_{oj}^{k} = 1 \qquad \forall k \in K \tag{5.11}$$

$$\sum_{i \in S_o \cup V} x_{ig}^{k} = 1 \qquad \forall k \in K \tag{5.12}$$

$$\sum_{i \in N} x_{ih}^{k} - \sum_{j \in N} x_{hj}^{k} = 0 \qquad \forall h \in V, k \in K \tag{5.13}$$

$$t_i^k - t_j^k + d_{ij} - Z(1 - x_{ij}^k) \leq 0 \qquad \forall (i,j) \in A, k \in K \tag{5.14}$$

$$\zeta \sum_{(i,j) \in A} d_{ij} x_{ij}^{k} \leq B_{\text{max}} \qquad \forall k \in K \tag{5.15}$$

where $x_{ij}^{k} \in 0, 1$ indicates if the edge between node i and j is active for robot k from the set of robots K, c_{ij} represents the cost activating the edge between i and j, d_{ij} is the duration required to traverse edge i, j, and $V = A \setminus \{S_o, S_g\}$ is a subset of nodes excluding the start and the goal. By setting $c_{ij} = d_{ij}$, the objective (5.9) will minimize the sum of travel time over all robots. Constraints (5.10)-(5.13) ensure that only one robot leaves each node, each robot begins at the start node S_o and ends at the goal node S_g, and any robots entering a node must leave it. Constraints (5.14) define t_i^k as the time node i is visited by robot k and ensures that node j is visited no earlier than t_i^k plus the time required to travel the edge between i and j. These constraints are traditionally used when enforcing constraints on the time a node is visited, but also act as subtour elimination constraints, ensuring that no independent loops exists in the final routes, disconnected from the starting location. Constraints (5.15) limit energy expenditure,

computed by multiplying the durations required to traverse edges d_{ij} and the discharge rate $\zeta \in \mathcal{R}^+$, to the battery limit B_{max}.

This formulation has been used in many multi-agent routing approaches including our previous work [16] in developing the initial framework for the systems-based approach presented in this chapter. As it solves for the minimal cost incurred by the entire team, we can easily solve for routes that expend minimal energy or have robots deployed for the minimal collective time. However, for a finite horizon approach that seeks to maximize the rate of waypoint visitation, it is more effective to minimize the time to completion. As such, we augment this formulation with a minimum makespan objective.

5.5.3 Min-Makespan VRP

The Min-Makespan VRP is a less common variant that is used to minimize the completion time of plans. It can be derived from the VRP formulation by replacing the objective (5.9) with

$$\min_{x} \max_{k \in K} \sum_{(i,j) \in A} d_{ij} x_{ij}^k, \tag{5.16}$$

which serves to minimize the maximum duration of any robot k's route. To maintain the Mixed-Integer Linear Problem formulation, it is convenient to represent this objective in the form

$$\min \theta$$

with the additional constraint

$$\theta \geq \sum_{(i,j) \in A} d_{ij} x_{ij}^k \quad \forall k \in K,$$

which represents the duration of the multi-robot plan for this horizon with the variable θ, also referred to as the makespan.

5.5.4 Ensuring Feasible Configurations

One major difficulty in these MIP formulations lies in determining ahead of time if a given problem formulation is feasible. It is entirely possible that a selection of waypoints cannot be fully visited given the number of robots available and their remaining capacities. To avoid this scenario, we augment our formulation to closer model the Orienteering Problem [4], which seeks to determine the maximum reward achievable by visiting waypoints. By not constraining all waypoints to be visited and providing reward for each waypoint visited, the MIP is maximized when the set of waypoints with the maximum total reward are included in the feasible solution.

The typical Orienteering Problem formulation can be expressed similarly to the above problems, with a slightly different objective

$$\max_{x} \sum_{k \in K} \sum_{(i,j) \in A} r_i x_{ij}^k \tag{5.17}$$

and waypoint visitation constraint

$$\sum_{k \in K} \sum_{j \in V} x_{ij}^k \leq 1 \quad \forall i \in V \tag{5.18}$$

which replace equations (5.9) and (5.10), respectively, leaving the remaining constraints (5.11)-(5.15) as listed. It is possible to combine this with either of the previous objectives, weighing the contribution of each objective separately in a mixed objective form. With proper tuning of the weights, it is possible to ensure that the Orienteering objective does not interfere with the desired objective, resulting in an optimization routine that allows for a subset of waypoints to be excluded based on merit. For example, a mixed-objective Min-Makespan/Orienteering problem

would be of the form

$$
\begin{aligned}
&\max_{x} \sum_{k\in K} \sum_{(i,j)\in A} r_i x_{ij}^k - C \max_{k\in K} d_{ij} x_{ij}^k \\
&s.t. \sum_{k\in K} \sum_{j\in V} x_{ij}^k \leq 1 && \forall i \in V \\
&\sum_{j\in V\cup S_g} x_{oj}^k = 1 && \forall k \in K \\
&\sum_{i\in S_o\cup V} x_{ig}^k = 1 && \forall k \in K \\
&\sum_{i\in N} x_{ih}^k - \sum_{j\in N} x_{hj}^k = 0 && \forall h \in V, k \in K \\
&t_i^k - t_j^k + d_{ij} - Z(1 - x_{ij}^k) \leq 0 && \forall (i,j) \in A, k \in K \\
&\zeta \sum_{(i,j)\in A} d_{ij} x_{ij}^k \leq B_{\text{max}} && \forall k \in K
\end{aligned}
$$

where r_i is the reward for including waypoint i in the solution. With our proposed application, it is sensible to represent $r_i = u_i$ as the utility of collecting an observation from waypoint i, as in Equation (5.2), assuming the observation is independent of other planned observations. There is a slight loss of optimality due to the independence assumption, however, this loss is minimized with the appropriate selection of waypoints (waypoints are chosen to be maximally informative, minimizing overlap), as per Sect. 5.3. The objective weight $C = \min_{i\in V} r_i \zeta / B_{\text{max}}$ is specifically chosen to decouple the objectives. We leverage the fact that no route will exceed the maximum duration ($L = B_{\text{max}}/\zeta$) to ensure that no matter how much the min-makespan objective is decreased, it will never increase the objective more than including the waypoint with lowest utility ($\min_{i\in V} r_i$). Effectively, the min-makespan objective is constrained to the values $[0, \min_{i\in V} r_i]$ and each decision variable in the orienteering objective increments the objective by values greater than (or equal to) $\min_{i\in V} r_i$, resulting in a complete disassociation of objectives.

5.5.5 Computational Complexity

The greedy assignment approach is the least complex of the presented options, as it is a straightforward evaluation of the minimum cost assignment for each waypoint. At each iteration, the next ending path is selected from the set of robot paths, then the next closest waypoint is selected from the set of unassigned waypoints. Without consideration of energy capacity limits, the complexity would be $O((N_r + N_w)N_w)$ as each of the N_w cycles would result in the assignment of a waypoint. With the capacity limits, there may be an additional cycle when determining that a path can no longer be extended. However, this results in decreasing N_r for subsequent cycles resulting in faster computation. As such, the complexity does not significantly change with the addition of capacity limits.

For the additional approaches, we consider the VRP and its variants, which belong to the NP-hard class of problems. As such, these problems are not guaranteed solvable in polynomial time and their solutions may not be verifiable in polynomial time. In practice, MIPs may be solvable in real time with few decision variables, but rapidly become computationally intractable as the number of decision variables increases. However, our approach fixes the number of waypoints assigned per planning iteration, which prevents the number of decision variables from scaling beyond a few virtual nodes for active robots as the known space expands. As such, the complexity of the assignment algorithm scales with the complexity of the environment, namely the configuration of the cost matrix, c_{ij}, relative to the constraint parameters, B_{ij} and B_{max}. Most MIP solvers perform a directed search through the space of feasible solutions, using branch and bound techniques to iteratively exclude sections of the solution space and quickly arrive at the optimal solution. While these can be fast for problems where there is a singular optimal solution, in the worst case, a solver may require an exhaustive search of the entire set of feasible solutions.

5.6 Conclusion

In this chapter, we presented the details of our proposed approach, combining the core framework from Chapter 3 with the environment model and utility function from 4, along with a section by section analysis of computational complexity. In compiling these details, we note several key relationships between parameters, model representations, and algorithm choices. Most importantly, we recognize that performance, in terms of information gained per horizon, is most directly influenced by the number of waypoints assigned each horizon and the informative quality of those waypoints. Other choices tend to indirectly affect performance through their influence on these values. For instance, the choice of number of waypoints processed per planning cycle N_w affects the amount of coupling between waypoints considered each cycle. The greater the amount of coupling considered, the more efficient the resulting plan, and the more waypoints that can be assigned per horizon. Alternatively, the choice of objective weight, α, affects the quality of waypoints selected by biasing the selection process towards cells whose dynamics model is poorly fit, which serves to compensate for environment changes that violate the locally static assumption. Making effective use of the proposed strategy relies on understanding the influence component algorithms and their associated parameters have on the number and quality of waypoints visited.

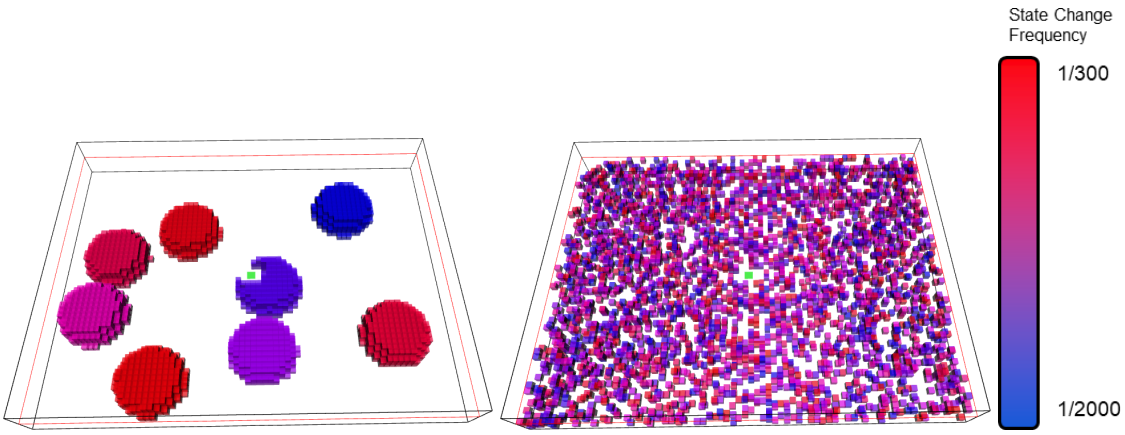
Chapter 6

Experimental Evaluation in Simulated Environments

While the previous chapter focused on establishing the theory behind the proposed system and describing how the component algorithms influence each other, in this chapter we focus on thorough evaluation of the proposed approach as influenced by parameter settings. We address three types of parameters: 1) algorithm variables that are chosen by a user, 2) physical constraints and limitations relative to the robotic platform used in deployments, and 3) conditions of the environment being explored. In this manner, we show how performance can be tuned with proper choice of parameters and what a user can expect in terms of performance relative to the choice of robotic platform or the dynamic properties of the environment. Additionally, we provide a comparison study of the proposed approach against the alternative *Waypoint Assignment* formulations highlighted in Section 5.5 as well as a pure greedy selection approach that iteratively adds waypoints to available robots.

6.1 Setup Details

To evaluate the performance of our approach, we devised a simulation environment where grid cells change state at set intervals. Fig. 6.1 shows several examples of the types of environments generated. We seek to evaluate performance in randomly generated worlds with key qualities, namely a specified range of dynamics and type of distribution. As such, we generated environments where the state of each cell oscillates at a random frequency, chosen from the range $[1/2000, 1/300]$ Hz, where the distribution of dynamic cells is chosen at random. We primarily test in an environment where cell states of a given oscillation period are clustered in groups (Fig. 6.1a), but also consider environments where the distribution of dynamics is random (Fig. 6.1b).



(a) Clusters in static empty space (b) Pure random

Figure 6.1: We evaluate on randomly generated environments, where cells oscillate between states at a regular frequency. Most tests are performed on a map where cells in clusters change state at the same frequency and phase (a). However, Sect. 6.5 also compares performance when the distribution of dynamics is purely random (b). The frequency of change in the above images is indicated by color, with the associated values indicated by the gradient on the right. Each environment is bounded by the black box, with robots deploying from the central green square and limited to flying in the region above the red line.

Robots depart from and return to the starting location, indicated by the green square in Figure 6.1. We assert that they are given an energy capacity corresponding to their maximum flight duration, noting that it is convenient to describe energy capacity in terms of flight duration for

the purposes of evaluation and suffices under the assumption of constant energy expenditure. In the described experiments, robots are added to the system as they are needed and removed when they are finished, effectively ensuring enough robots will always be available for the system to deploy. Observations are collected at the instant a robot visits a waypoint to ensure the map is updated at the planned locations. To also include observations collected when transitioning between waypoints, additional observations are collected at a frequency of one per second.

To further simplify computation, cells within the safety region are given a prior of $p(m_i) = 0.001$ or $p(m_i) = 0.999$ based on the ground truth state of m_i. Given that these cells are not subject to the confidence decay of the environment model and that the sensor model is always accurate by design, the traversability of this region will remain consistent over all time. As such, the area robots traverse remains constant and, as a result, the evaluations below do not reflect the conditions of traditional exploration techniques where the space a robot can traverse grows over time. Instead, this work focuses on exploring the speed and optimality of solutions given full access to traversable space. While the system would be perfectly capable of operating in a fully unknown environment, subject to the restrictions on the locations of dynamic objects noted earlier, it is reasonable in our desired application to assume enough knowledge of free and occupied space to obviate the need for this early stage of exploration.

The following set of tables list the parameters evaluated in this chapter. Unless otherwise specified, the listed values are used as default in the subsequent experiments. Table 6.1 lists the system parameters whose effects on performance are evaluated in Sect. 6.3, where N_w is the number of waypoints processed each planning iteration, N_r is the maximum number of robots active at any given time, d_{adv} is the amount of time the planning horizon advances after each instance of updating the environment model, d_{hor} is the duration of the planning horizon, and α is the weight in the *Waypoint Selection* objective function that biases the utility score towards CSQMI or χ^2-based objectives. Table 6.2 lists the physical parameters of the robot team whose effects on performance are evaluated in Sect. 6.4, where v_{travel} is the standard speed robots at which travel between waypoints (though the speed may change as robots accelerate and deceler-

Table 6.1: System used in the simulation experiments and their default values.

N_w	N_r	d_{adv}	d_{hor}	α
6	3	100s	120s	150

ate as necessary), L corresponds to the duration a robot can operate on a single battery, r_{sense} is the maximum range of beams the sensor projects, θ_{sense} is the sensor's field of view in both the horizontal and vertical directions, and ρ_{sense} is the number of beams per slice of the field of view both horizontally and vertically.

Table 6.2: Physical limits used in the simulation experiments and their default values.

v_{travel}	L	r_{sense}	θ_{sense}	ρ_{sense}
6	300s	12	40°	10

6.2 Approach Comparison

This section evaluates the performance of the chosen approaches for *Waypoint Assignment* the proposed system utilizes. As a comparison, we implement a pure greedy approach, in which a new waypoint is selected the instant a robot reaches its next assigned target, and compare against the proposed system using the greedy assignment approach, as described in Sect. 5.5.1, the VRP assignment described in 5.5.2, and the Min Makespan assignment approach, described in Sect. 5.5.3. The pure greedy approach we consider uses the mixed objective:

$$w_{next} = \arg\min_{w \in W_r} \text{CSQMI}(m, t, w \cup W_{planned}) + C_d(\text{Cost}(x, w)) \tag{6.1}$$

that combines the CSQMI utility measure, which computes the value of observing from a candidate waypoint w given the planned future waypoints $W_{planned}$, with a weighted path-length measure. The variable W_r corresponds to a set of candidate waypoints for which a path to the starting location exists and traversing the path from the current location x, to the waypoint, and

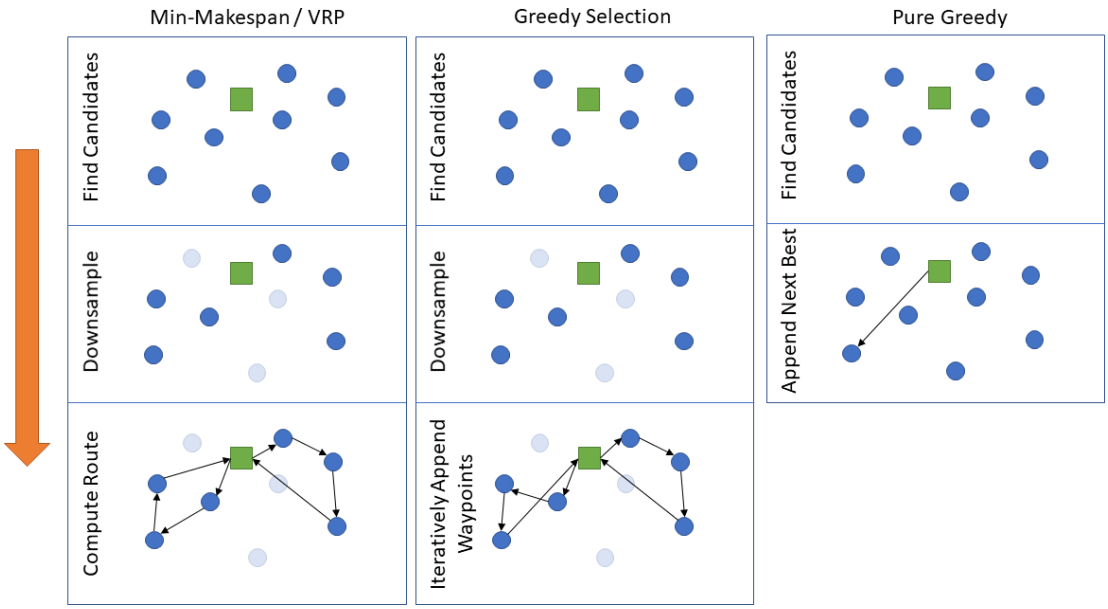


Figure 6.2: Diagram showing the differences between evaluated approaches. All approaches begin with computing the utility score of reachable waypoints to find candidates. The Min-Makespan, VRP, and greedy selection approaches then downsample to the N_w best subset and compute routes, with the greedy selection approach iteratively extending routes by selecting the next closest robot-candidate pair. Alternatively, the pure greedy approach cycles directly between computing utility score and assigning the highest scoring waypoint to the next available robot.

then to x_s is possible given the remaining capacity

$$W_r = \{w : L - L_{current} > \text{Cost}(x, w) + \text{Cost}(w, x_s)\},$$

where $L_{current}$ is the current remaining capacity. The distance weight C_d defines how much the distance objective is emphasized relative to the utility objective. Refer to Fig. 6.2 for a visual representation of how pure greedy assignment differs from the proposed approach and its associated *Waypoint Assignment* algorithms.

Figure 6.3 shows the evolution of the performance of the approaches compared in this section. Recalling that we compute entropy as:

$$H(m_t) = -\sum_{m_{i,t} \in m'_t} p(m_{i,t}) \log_2 p(m_{i,t}) + (1 - p(m_{i,t})) \log_2 (1 - p(m_{i,t})),$$

and KL-Divergence as:

$$D_{KL}(t,m) = \sum_{m_i \in m'_t} p_o(m_{i,t}) \log_2 \frac{p_o(m_{i,t})}{p(m_{i,t})} + (1 - p_o(m_{i,t})) \log_2 \left(\frac{1 - p_o(m_{i,t})}{1 - p(m_{i,t})} \right),$$

the entropy and divergence plots show performance increasing over time as these values decrease. Note that entropy and divergence use a modified map $m'_t \subset m_t$ that excludes any cells within the dynamic clusters that will always be occluded when they are in the occupied state and, thus, only observed in the free state. We exclude these cells as they will only degrade model accuracy despite the fact that there is little difference between a hollow and solid object for mapping purposes.

The response progression plot in Fig. 6.3 shows the evolution of response fraction over time, where response fraction is computed using:

$$\bar{d}^i_{r,t} = max(t^i_{obs} - t^i_{change})/T_i \tag{6.2}$$

each time an observation is collected. The response fraction in the plot is displayed as a windowed maximum to simplify the plot, showing only the greatest value of $\bar{d}^i_{r,t}$ for each observation collected in a 100 second window. This is a slightly modified version of the response fraction introduced in Chapter 4 that expresses the multiples of period of change, T_i, that exist within the duration between the first instance a cell changed state after the previous measurement, t^i_{change}, and the next time an observation is taken, t^i_{obs}. By plotting $\bar{d}^i_{r,t}$ against t^i_{obs} for each measurement collected, we can see where the system begins to converge and observations are regularly collected within the period of change.

From Fig. 6.3, we can conclude that the Min-Makespan, VRP, and Greedy selection approaches outperform pure greedy assignment. Entropy and divergence converge more quickly to much lower values for the system-based approaches and the response fraction is much more well behaved. While the system-based approaches converge to low response fraction early (around

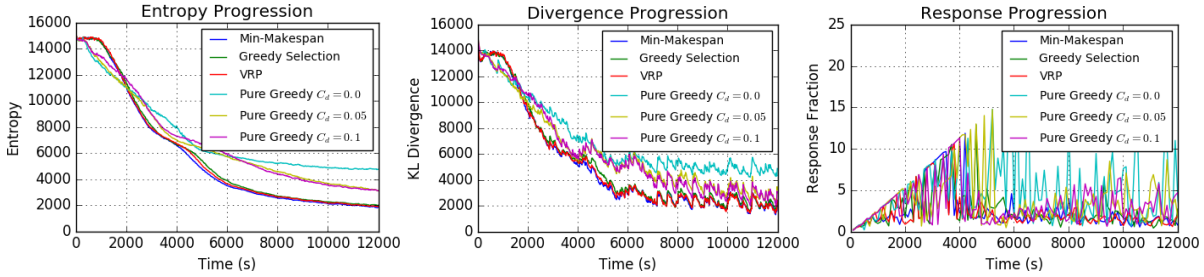


Figure 6.3: Comparison of approaches evaluated in the environment from Fig. 6.1a using the default values listed in Tables 6.1 and 6.2, exhibiting entropy (left), KL-Divergence (middle), and the response fraction (right). Both the entropy and KL-Divergence plots show the pure greedy approach at significantly higher values than the system-based approaches, for each value of C_d, suggesting lower model confidence and accuracy. Additionally, the response fraction plot shows many new observations of cells after they have gone through more than 5 cycles of changes, suggesting that the pure greedy approach is slow to respond to regions that have already been observed. The choice of the *Waypoint Assignment* approach, however, appears to have minimal impact on performance at the scales visible in these plots. Refer to Table 6.3 for a more precise comparison of steady-state values.

$t = 4000$) and stay low after that, the pure greedy approaches exhibit spikes of greater than 5 well after $t = 8000$. Figure 6.4 shows a series of images of the occupancy grid at several time steps for both the pure greedy approach and the Min-Makespan approach. From these images, we see the Min-Makespan approach learning the static regions much more quickly than the pure greedy approach. As such, the dynamic regions are discovered earlier and the steady-state is reached much more quickly. The pure greedy approach focuses on the dynamic clusters until they are known with high confidence, but applying many observations in a short period of time is less effective when the environment exhibits changes on a slow time scale.

With a CSQMI-based objective, the robots tend to prefer sensing locations with an unobstructed view with many cells of low certainty at the edge of the field of view. As such, robots may tend to "carve" away at areas they have already begun to observe rather than observe a new area with a flat surface of unknown cells. When the environment is static, this behavior is acceptable as the map will eventually become entirely known. However, in dynamic environments uncertainty continues to grow in the regions already explored, continuously drawing robots back to these areas. By batching the selection of target waypoints, without allowing multiple obser-

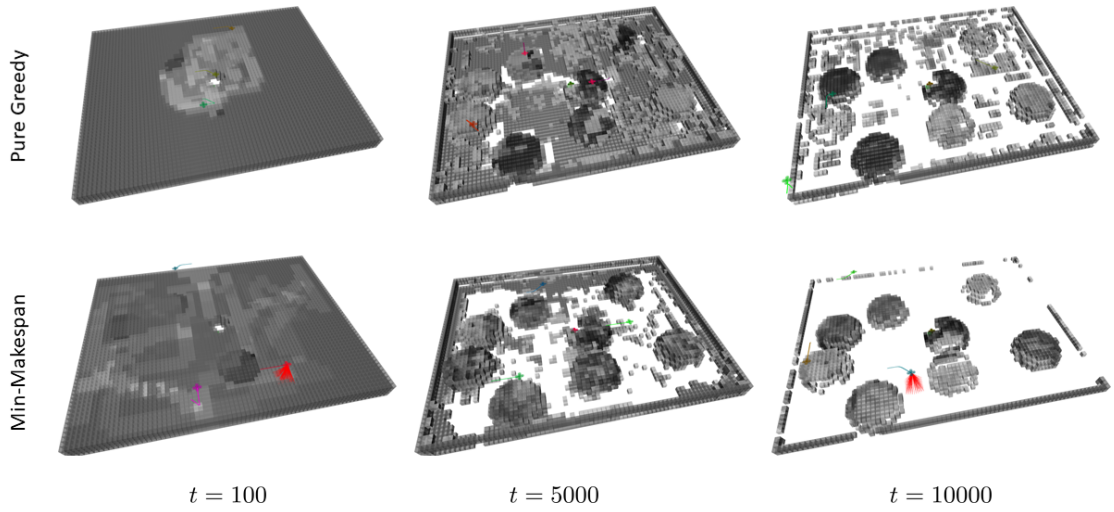


Figure 6.4: Vignette of the environment models for simulated runs for the Min-Makespan and pure greedy approaches. Note that the Min-Makespan approach learns more of the map faster than the pure greedy approach, while the pure greedy approach expends extra effort on observing the dynamic clusters until they are represented with high confidence. The Min-Makespan approach is preferable for the proposed scenario as the entire static region is uncovered earlier, ensuring stray clusters will not be missed while robots focus on nearby clusters.

vations from the same position in a single planning horizon, the proposed approach promotes the visitation of a more varied set of locations resulting in a more even spread of focus. In this manner, the system learns the dynamic model of the environment more quickly, slowing down the growth of uncertainty enough to allow for the small amount of robots to more effectively cover the dynamic regions at the appropriate frequencies.

Table 6.3 provides a more detailed view into key evaluation criteria used to differentiate the performance of the individual approaches. The criteria evaluated are as listed below:

- $E(N_r)$: the average number of robots deployed over the entire run. This value decreases as more time is spent after one robot lands before another takes off, whether through longer durations or more frequent instances.
- t_{conv}: the time it takes for the model entropy to converge to 10% of the steady-state value.
- μ_H: the steady-state entropy value averaged over the last 1500 seconds of operation
- μ_D: the steady-state KL-Divergence value averaged over the last 1500 seconds of operation

- $E(d_r)$: the average response fraction at the end of the run. This value is computed using (6.2) where t^i_{obs} is set to the run end time, averaged over the set of unobstructed cells, m'_t. Compared to the windowed maximum in the Response Progression plot in Fig. 6.3, which shows the worst performing cell at each instance of time, $E(d_r)$ expresses the average response performance of the model.
- N_u: the number of cells in m'_t that are unobserved at the end of the run.

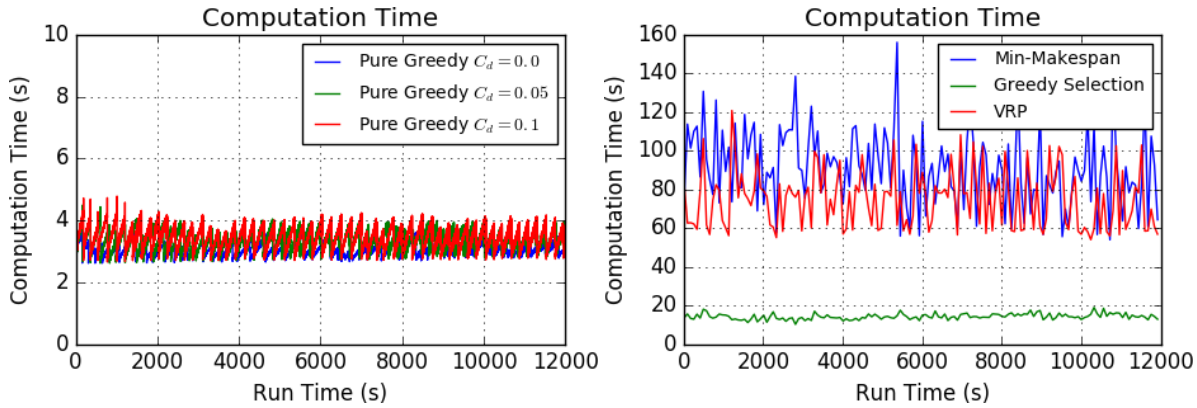
With the system-based approaches, there is a distinct improvement in model performance evident in the μ_H and μ_D values for the thin Min-Makespan approach. Additionally, N_u is lower for Min-Makespan, suggesting that the approach exhibits more complete coverage than VRP or Greedy Selection. While the average response fraction $E(d_r)$ is slightly higher for Min-Makespan, it is still significantly less than 0.5, which would suggest that, on average, the system responds well enough to prevent aliasing effects.

Regarding the pure greedy approaches, there is a significant improvement to performance with the inclusion of a distance weighted objective, resulting in a lower average entropy, divergence, and response fraction when compared to the non-weighted approach ($C_d = 0$). The number of unobserved cells, N_u, also decreases with the addition of the distance weight, but begins to rise again as the weight is increased further ($C_d = 0.1$). As the weight increases, the planner becomes more myopic and constrains itself to nearby unknown space.

The computation time of the tested approaches are shown in Fig. 6.5. As the pure greedy approach only builds the candidate pool and selects the best option, each instance only requires 2-5 seconds. However, for a greedy approach, this can be prohibitively expensive as the computed splines may require significantly less time to traverse if the goal is nearby. The system-based approaches require significantly more time per iteration, but the plans generated allow for a planning duration of up to d_{adv} before the next set of plans will be required. While the displayed computation times for Min-Makespan and VRP occasionally exceed $d_{\text{adv}} = 100s$ in the plot, we can tune parameters to reduce computation time appropriately, as explored in Sect. 6.3.

Table 6.3: Comparison of approaches. In each run we evaluate: the average number of robots deployed, $E(N_r)$, the time for entropy to reach 10% of the steady-state value, t_{conv}, the entropy and KL-Divergence values averaged over the last 1500 seconds of operation, μ_H and μ_D, the average response fraction, $E(d_r)$, and the number of unobserved cells, N_u. The system-based approaches all outperform the pure greedy approaches, with Min-Makespan having a comparable μ_H and μ_D and the lowest N_u overall.

Approach	$E(N_r)$	t_{conv}	μ_H	μ_D	$E(d_r)$	N_u
Min-Makespan	2.87	7695	1688	1867	0.0018	440
Greedy Selection	2.88	7694	1831	1946	0.0011	495
VRP	2.93	7769	1678	1873	0.0012	501
Pure Greedy $C_d = 0$	2.94	7180	4652	4767	0.0345	1804
Pure Greedy $C_d = 0.05$	2.98	9802	2875	2800	0.0149	1009
Pure Greedy $C_d = 0.1$	2.98	9397	2912	2640	0.0089	1217



(a) Pure greedy approaches

(b) System-based approaches

Figure 6.5: Comparison of computation times for the pure greedy approaches (a) and the system-based approaches (b). The pure greedy approaches are faster to compute, but require more iterations. The greedy selection approach is the fastest of the system-based approaches, given that *Waypoint Assignment* requires the majority of the computation time for the Min-makespan and VRP.

6.3 Parameter Evaluation

This section shows the performance of our approach as we vary the core system parameters. Evaluations are performed in the environment with clustered dynamics shown in Fig. 6.1a using the Min-Makespan *Waypoint Selection* approach. The first set of parameters we focus on are the maximum number of robots deployed at any given time N_r and the number of waypoints selected per horizon N_w. For the most part, these parameters effect performance in a very intu-

itive fashion, which we show through plots of specific parameter sets followed by a table of key, steady-state values to highlight long term performance.

Considering the plots of performance for different values of N_w, as shown in Fig. 6.6, we see lower entropy and KL-Divergence as the coupling of waypoints per planning iteration is increased. The response fraction seems to converge to steady-state at roughly the same time for each value of N_w, suggesting that N_w (for the tested values) has little impact on timely coverage. However, we see that compute time increases drastically as N_w increases. For small N_w, the system invests most of its computation time in the *Waypoint Selection* and *Path Planning* phases, for which the required compute time is relatively consistent. However, as N_w increases, *Waypoint Assignment* becomes the dominant component and we see a large variance in compute times each cycle as the difficulty of the planning problem can vary greatly depending on the waypoints selected and their pairwise travel costs. We see from these graphs that increasing coupling results in a slight improvement to performance at a cost that appears to grow exponentially.

To investigate this further, we evaluate the ability of the planner to efficiently assign waypoints by counting the number of waypoints that are assigned to each robot, shown in Fig. 6.7. We can see that there is a general trend to assign more waypoints to each route the higher we set N_w. The more waypoints we consider in each planning cycle, the more efficient the resulting plan will be, resulting in more waypoints assigned to each robot. Given that there are somewhere between 20 and 70 waypoints assigned to each route, with 3 robots operating simultaneously (each traversing their own route of 20-70 waypoints), it is likely that even $N_w = 12$ is too small to generate a significant improvement in performance. Considering that computation is already prohibitively expensive ($85s > d_{\text{adv}} = 60s$), future improvements will have to rely on sub-optimal heuristic solvers to leverage coupled waypoint routing in real-time.

The most obvious improvements to performance we can see are when we vary the number of robots N_r, as shown in Fig. 6.8. While increasing N_w increases the number of waypoints each robot services, increasing N_r adds more agents to service waypoints independently. As such, the addition of more robots allows the system to distribute observations more widely across the map,

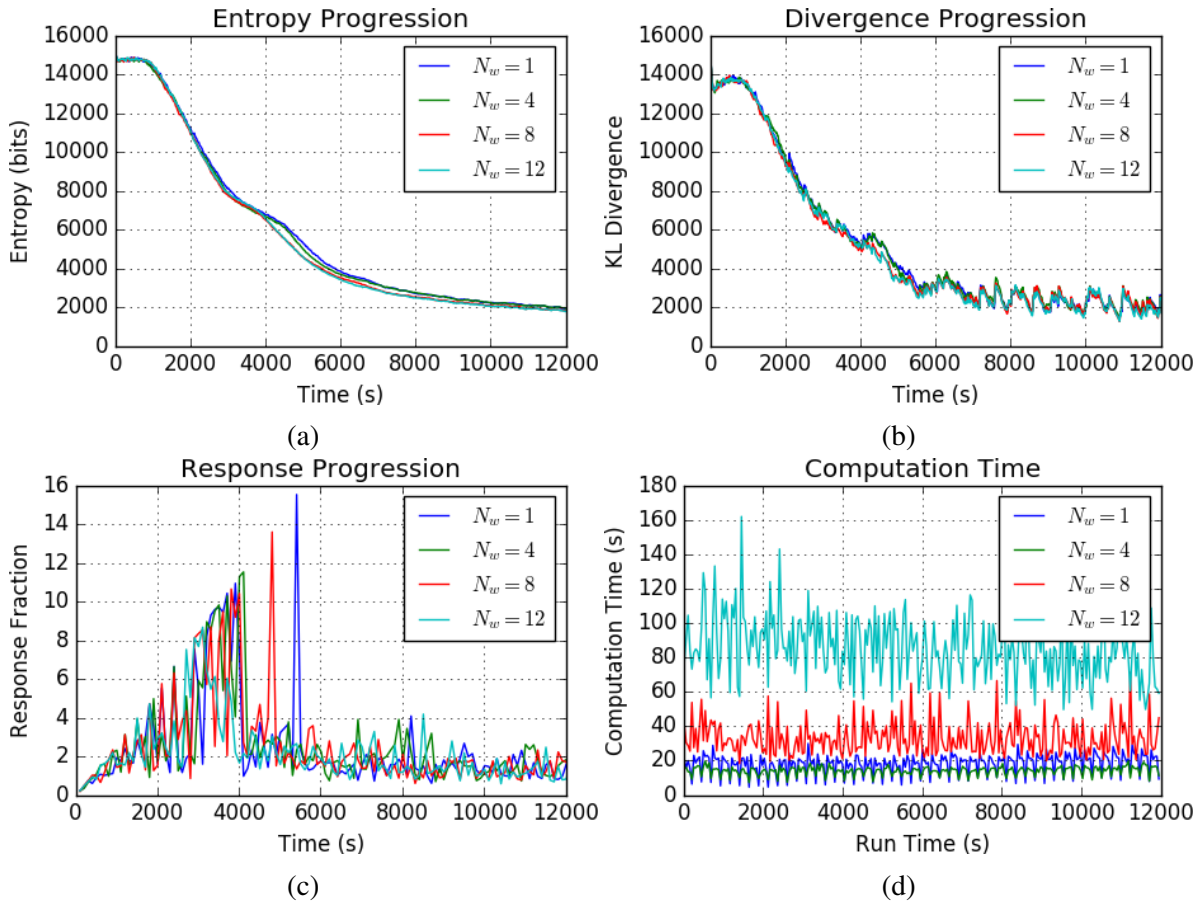


Figure 6.6: Performance analysis relative to varying numbers of waypoints processed per horizon N_w, with $N_r = 3$, $d_{\text{adv}} = 60s$, $d_{\text{hor}} = 120s$, $\alpha = 150$. While performance appears consistent over chosen values, both entropy (a) and KL-Divergence (b) decrease faster for higher N_w. The response fraction (c) is largely consistent, with some later spikes when there is less coupling between waypoints (lower N_w). Computation time (d) shows a minor baseline increase as N_w increases, but with large spikes as the *Waypoint Assignment* algorithm begins to dominate in complex MIP configurations. Also, note that there is a significant decrease in computation time whenever the set of active robots reach their full operating capacity ($L = 300s$). Planning complexity is reduced significantly in this case as there are less waypoints that can be added without exceeding L.

resulting in better compensation for the growth of uncertainty. This results in a faster convergence to lower entropy and KL-Divergence, with response fraction converging much faster the more robots we deploy (as more of the environment is observed well enough to track dynamics more quickly). For computation time, we note that the *Waypoint Selection* and *Path Planning* components are agnostic to N_r, so the main contributor to complexity, relative to N_r, is *Waypoint*

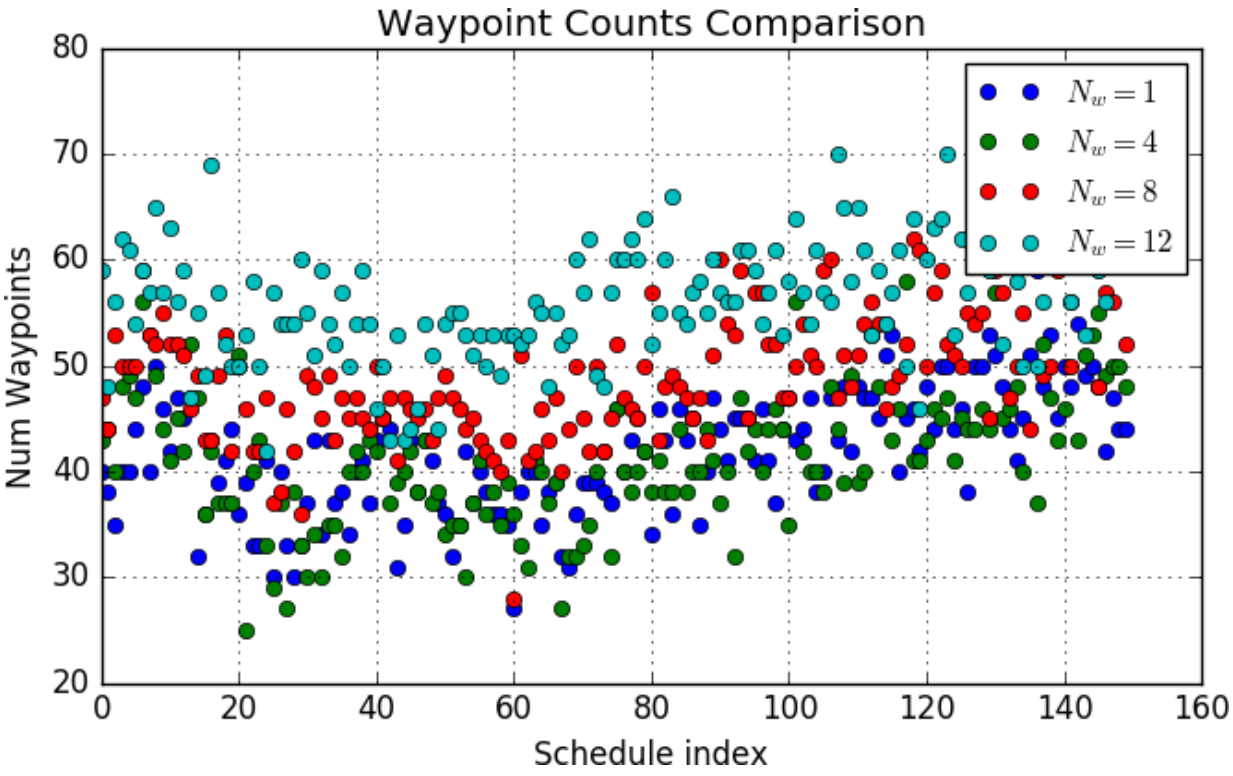


Figure 6.7: Comparison of the numbers of waypoints assigned to deployed robots. The ith schedule index corresponds to the ith route traversed in a given run (regardless of the specific robot traversing the route). For the above plot, the index also correlates the start time of the associated routes (for each N_w, the ith schedule starts at the same time). It is relatively obvious that increasing the coupling between waypoints (via N_w) results in a significant increase in the number of waypoints assigned to any given route. This result coincides with our intuition that more coupling generates more efficient routes that, in turn, allow for the assignment of more waypoints.

Assignment. In this case, the more decision variables we manage in the ILP, the more difficult the problem is to solve. Given that there are $N_w N_w N_r$ variables in x_{ijk}, the number of decision variables scales linearly with the number of robots deployed. While this is not as bad as scaling N_w, the computation time can grow quickly as N_r increases when the number of waypoints per planning iteration is high.

Table 6.4 reinforces our conclusions about these trends, as we see significant increases to performance over all values as N_r increases and minor improvements to convergence rate t_{conv} for higher N_w. One thing to note, though, is that the expected response fraction $E(d_r)$ and number of unobserved cells N_u exhibits diminishing returns as N_r increases. This is likely due

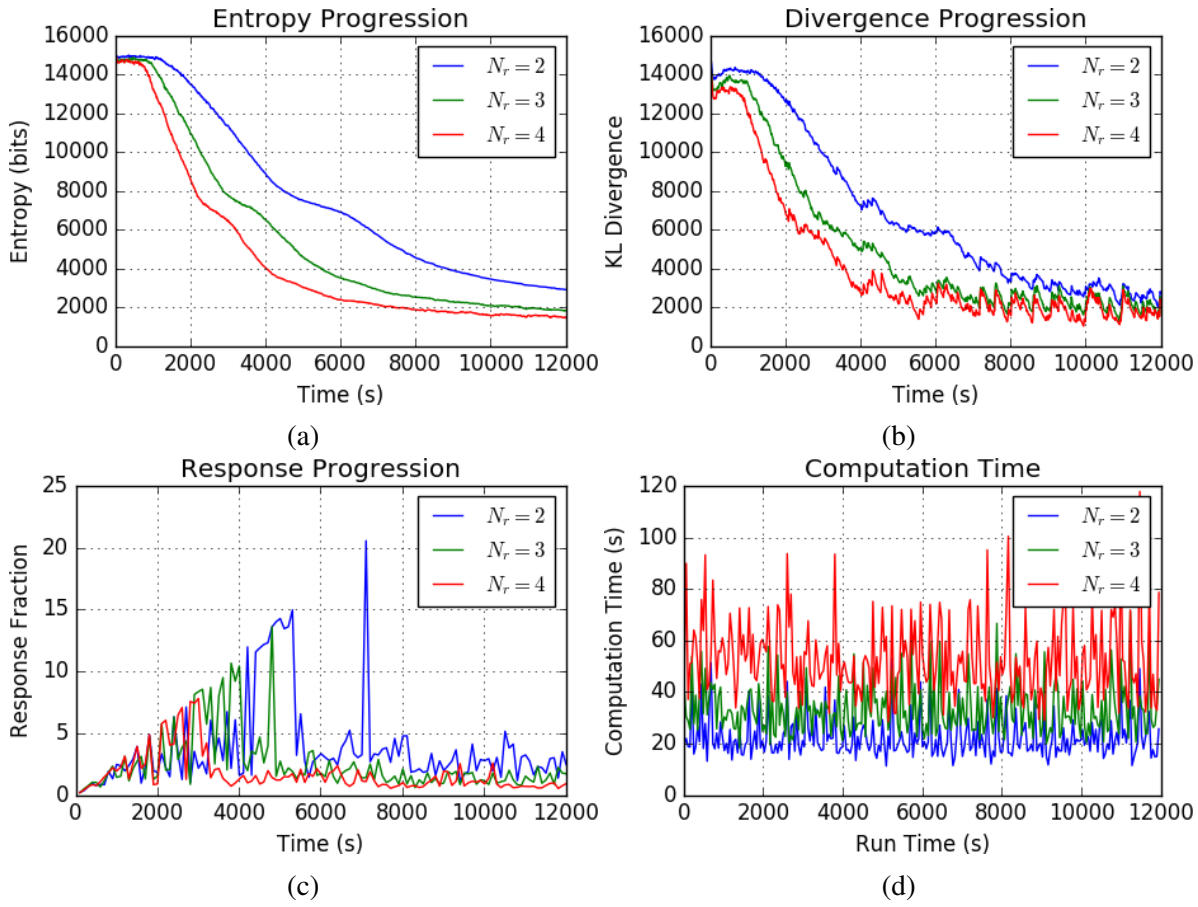


Figure 6.8: Performance analysis relative to the maximum number of robots active at any given time N_r, with $N_w = 8$, $d_{\text{adv}} = 60$, $d_{\text{hor}} = 120$, and $\alpha = 150$. Entropy (a) and divergence (b) converge much more quickly as N_r increases, allowing robots to cover more of the environment at any given time. We also see a decrease in the steady-state values as larger deployments are better suited to counter uncertainty growth. There is also a clear increase in the convergence of the response fraction (c) as the improved distribution allows for faster response to expected changes. However, these improvements come at the cost of computation time (d) as the complexity of solving an ILP is directly tied to the number of decision variables, of which there are $N_w^2 N_r$ in this problem. This results in a almost linear increase in computation time in the above plot, though the variance appears to grow the more difficult the problem becomes.

to the limited map size, resulting in a limited amount of work for any given set of robots to do. As we increase the team size, the closer we get to saturating the map with observations and the more redundant many of our observations will become.

The next set of parameters we consider are the horizon advance d_{adv} and horizon duration d_{hor}. For the horizon advance parameter plotted in Fig. 6.9, we see no significant change in

Table 6.4: Evaluation of system performance while varying N_w and N_r, for $d_{\text{adv}} = 60$, $d_{\text{hor}} = 120$, and $\alpha = 150$. Trends suggest faster convergence to more accurate maps occurs when N_w and N_r increase. The response fraction and coverage, shown by $E(d_r)$ and N_u, show a marked increase with higher N_r, but less so with higher N_w. These trends indicate that the system can greatly improve performance with more robots, but will require more development to reap the benefits of improving the coupling between waypoints.

N_r	N_w	$E(N_r)$	t_{conv}	μ_H	μ_D	$E(d_r)$	N_u
2	1	1.91	10270	2636	2564	0.0066	633
3	1	2.87	8039	1741	1958	0.0022	434
4	1	3.80	6435	1443	1663	0.0013	303
2	4	1.92	10290	2620	2559	0.0061	651
3	4	2.85	7980	1748	1988	0.0023	454
4	4	3.81	6225	1436	1664	0.0009	320
2	8	1.95	10000	2557	2483	0.0045	620
3	8	2.86	7605	1689	1991	0.0015	494
4	8	3.82	6165	1412	1724	0.0008	325
2	12	1.90	10190	2424	2463	0.0039	647
3	12	2.88	7500	1671	1896	0.0010	453
4	12	3.80	6070	1411	1701	0.0012	312

performance as the parameter varies. Recalling the locally static assumption we make relative to the environment model, d_{adv} effectively denotes how new the information is that the system processes each planning horizon. Given that changes to the environment occur on the order of 300-2000 seconds, those changes that occur between each horizon are infrequent enough to not be significantly impacted by the locally static assumption. Also acknowledging the design choice that robots are only deployed at the beginning of each horizon, we always choose d_{adv} to be less than the robot capacity L to ensure that robots do not finish their run before the next horizon starts. When L is significantly less than the period of environment dynamics, no choice of $d_{\text{adv}} < L$ can be expected to reduce the performance of our system.

Another quirk related to design choice is that performance is maximized when d_{adv} is a multiple of L. In this case, each deployed robots capacity ends at the end of a planning horizon. Otherwise, robots fully deplete energy reserves in the middle of a horizon and must wait for the next before another robot can be deployed. However, given the complexity of including the deployment of new robots mid-horizon in the *Waypoint Assignment* phase, it is simpler to restrict

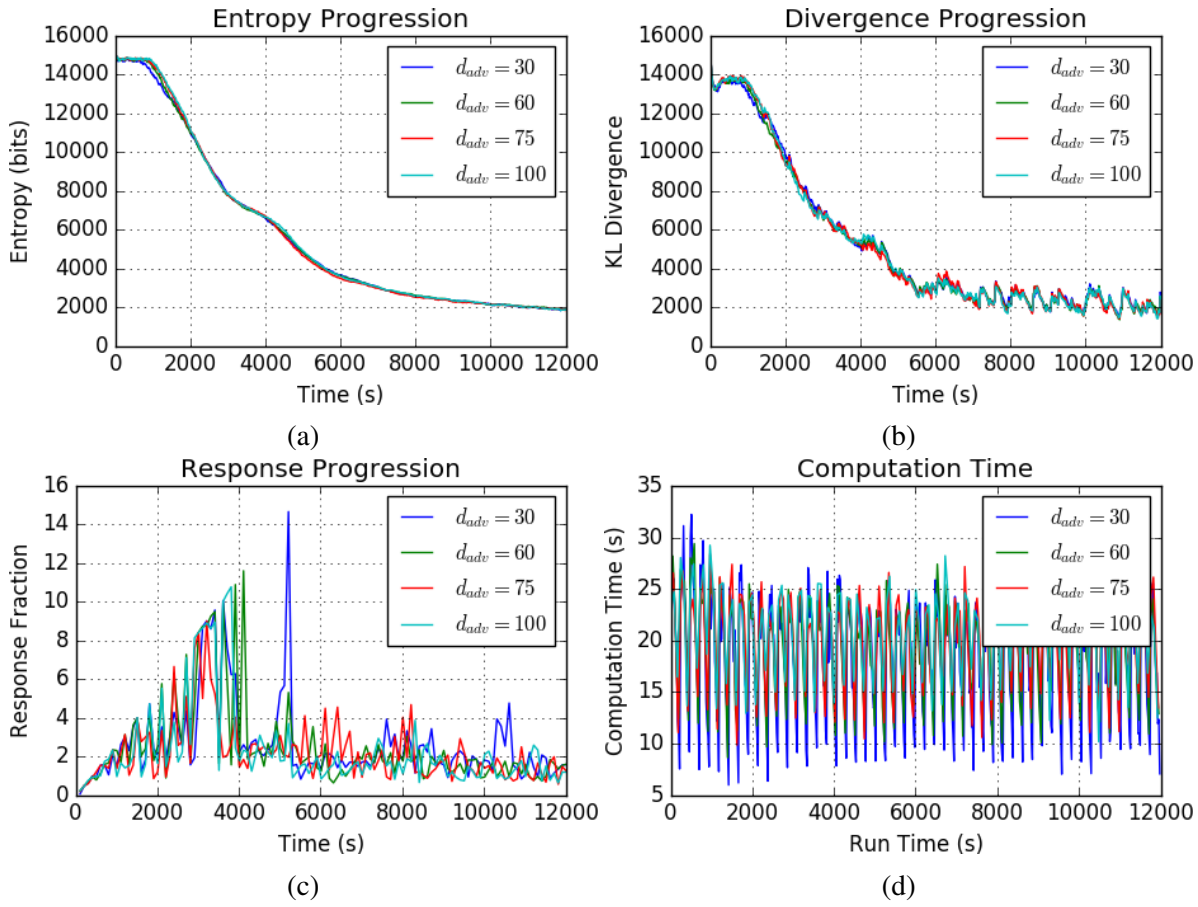


Figure 6.9: Performance analysis relative to the horizon advance duration d_{adv}, with $d_{\text{hor}} = 100$, $N_w = 8$, $N_r = 3$, and $\alpha = 150$. Performance does not significantly change for varying values of d_{adv}. This suggests that it is possible to update plans relatively infrequently, while still remaining responsive to the changes in the environment.

choices of d_{adv} to multiples of L.

For the horizon duration parameter d_{hor}, with evaluation results depicted in Fig. 6.10, we see a similar trend where performance is consistent in most cases. However, we note that the case where $d_{\text{adv}} = d_{\text{hor}}$ exhibits a slight decrease in performance. In this case, the next horizon starts after the current plans end and the system becomes unable to extend schedules over multiple horizons. As a result, the effective single-robot deployment duration becomes d_{hor} as opposed to the maximum-flight duration L, given that $d_{\text{hor}} < L$ by design. The decrease in performance occurs as a result of robots spending more time traversing to and from the charging station, as discussed in more detail in Sect. 6.4. To ensure the system operates properly, we must enforce

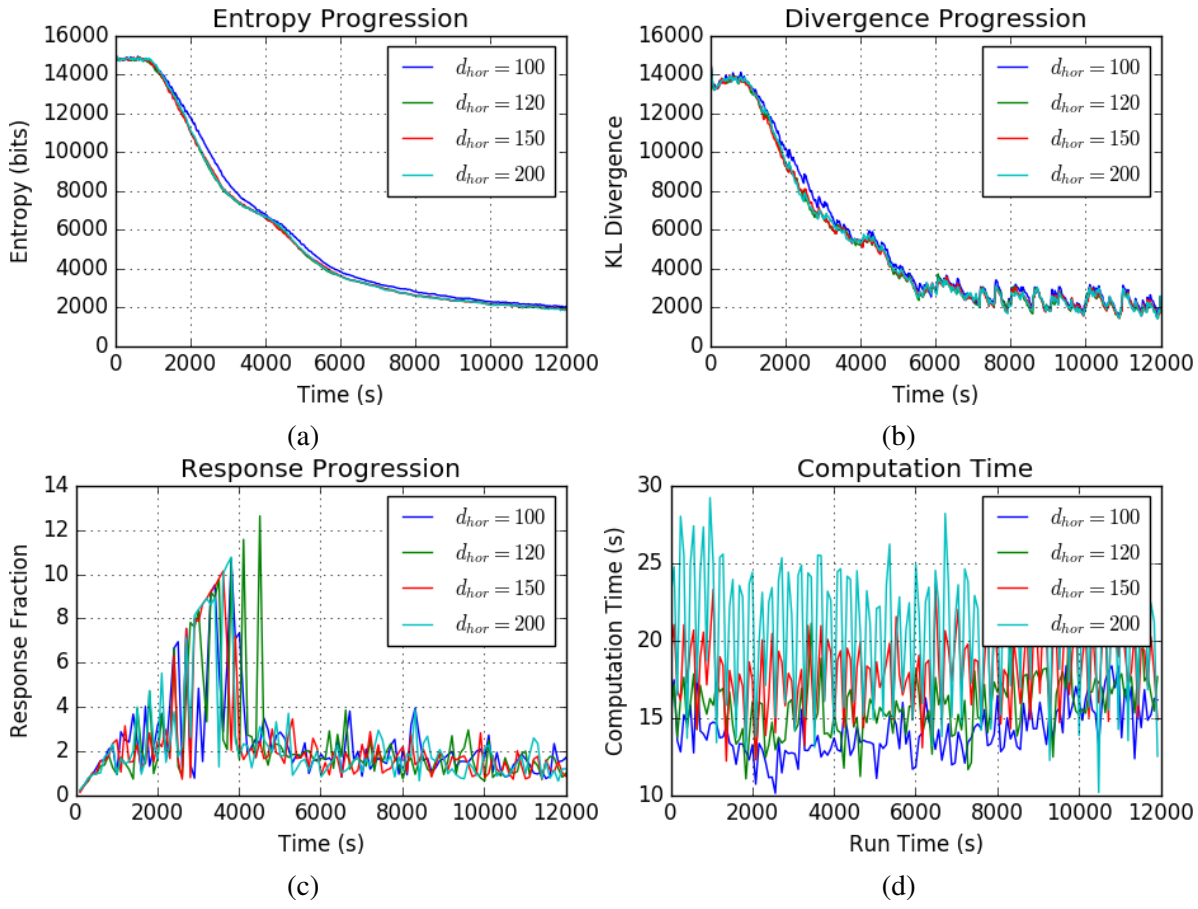


Figure 6.10: Performance analysis relative to the horizon window duration d_{hor}, with $d_{\text{adv}} = 100$, $N_w = 8$, $N_r = 3$, and $\alpha = 150$. We notice a slight dip in performance when $d_{\text{adv}} = d_{\text{hor}} = 100$. This can occur due to schedules not being properly extending across multiple horizons, resulting in schedules operating for only $d_{\text{hor}} < L$. The other significant difference to note is the computation time, as more time is required to compute longer horizons. This increase appears roughly linear as increasing d_{hor} simply increases the number of planning iterations.

$L > d_{\text{hor}} > d_{\text{adv}}$, with the horizon duration extending far enough beyond d_{adv} to ensure continuity across horizons.

Finally, we consider the objective function weight α introduced in Chapter 4. Figure 6.11 shows a distinct variation in performance as α changes. Low values of α, which favor the MI-based objective, quickly reduce entropy and divergence in the early stages. Then, once the environment dynamics start to become known, the χ^2 objective starts to become more effective, favoring high α values. Given the known influence of α from the experiments in Chapter 4, we can surmise that at this time step (roughly $t = 3800$) that high α values cause the system to

Table 6.5: Evaluation of performance as horizon variables vary over several runs with $N_w = 12$, $N_r = 3$, and $\alpha = 150$. Values are largely consistent over all runs. Note a dip in performance for $d_{\text{adv}} = d_{\text{hor}} = 100$ as the system ends schedules early when the horizon advances beyond the current set of plans.

d_{adv}	d_{hor}	$E(N_r)$	t_{conv}	μ_H	μ_D	$E(d_r)$	N_u
30	100	2.90	7889	1716	1945	0.0019	462
60	100	2.86	7665	1689	1986	0.0021	499
75	100	2.88	7784	1706	1975	0.0016	455
100	100	2.83	8010	1813	1993	0.0015	475
30	120	2.88	7634	1716	2026	0.0018	491
60	120	2.87	7680	1712	1986	0.0017	457
75	120	2.86	7725	1726	2025	0.0017	460
100	120	2.96	7809	1718	1949	0.0012	414
30	150	2.86	7545	1706	1978	0.0023	480
60	150	2.86	7740	1701	2006	0.0022	454
75	150	2.92	7754	1706	2019	0.0021	478
100	150	2.88	7800	1708	1930	0.0012	431
30	200	2.86	7695	1755	1927	0.0020	480
60	200	2.88	7770	1716	2010	0.0025	465
75	200	2.86	7650	1705	1961	0.0018	465
100	200	2.86	7785	1705	1890	0.0018	431

transition focus to favor static regions while low α values focus on observing regions that are somewhat known. Spreading focus in this manner allows robots to better distribute attention to allow for faster response to changes, as evidenced in Fig. 6.11c with high alpha values converging to a stable response fraction earlier than lower values. This trend is further evidenced in Table 6.6, with performance significantly improved for greater values of α. In particular, we note that the number of unobserved cells N_u decreases significantly for higher values of α as robots are directed to explore more of the environment.

6.4 Physical Limits

In this section, we evaluate parameters related to the limitations of the system and, as such, are not chosen by the user. The first we consider energy capacity parameter, as depicted in Fig 6.12. While it is a significant aspect of the proposed problem, the actual value has little effect on

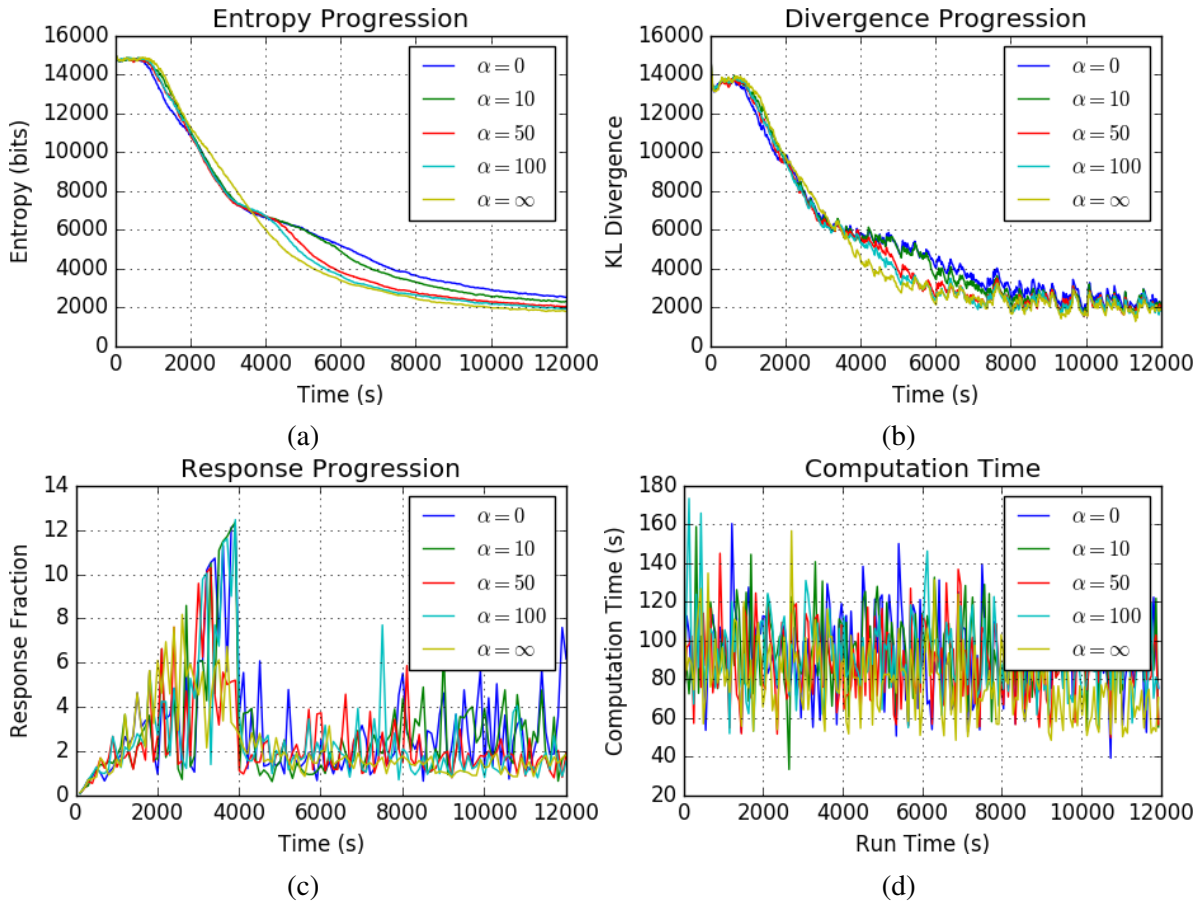


Figure 6.11: Performance analysis relative to the mixed objective weight α, with $N_w = 12$, $d_{\text{adv}} = 60$, $d_{\text{hor}} = 120$, and $N_r = 3$. This data set has an interesting progression, as the best parameter choice varies over time. Initially, when most of the environment is unknown, low α values, which bias towards the CSQMI objective, are preferable. Once the dynamics become known, however, better performance is achieved with a bias towards the χ^2 objective (higher α). When purely evaluating relative to the response fraction (c), though, we see a definitive bias towards high α values as these objectives bias observation focus towards regions where the observed dynamics differ from the learned model.

performance as there is no limit to the number of robots deployed. Instead, the current active limit N_r is maintained whenever a robot fully depletes its energy capacity by deploying another robot in the subsequent horizon. Any small fluctuations in entropy and divergence will occur as a result of robots with less capacity needing to return to the charging station more often, thus redundantly covering the area around the charging station. This effect is more prominently shown in the response fraction plot Fig. 6.12c, where higher values of L converge more quickly to the

Table 6.6: Evaluation of performance as the objective weight α varies over several runs with $N_w = 12$, $d_{\text{adv}} = 60$, $d_{\text{hor}} = 120$, and $N_r = 3$. We see the steady-state values prefer α to be higher, to better direct robots towards less well modeled regions.

α	$E(N_r)$	t_{conv}	μ_H	μ_D	$E(d_r)$	N_u
0	2.90	8220	2636	2310	0.0046	729
1	2.90	8244	2556	2224	0.0059	803
10	2.89	8004	2368	2167	0.0042	639
50	2.91	7284	2110	2050	0.0029	541
100	2.97	7067	2003	2038	0.0016	537
150	2.90	7020	1975	2021	0.0024	486
300	2.88	6924	1938	1951	0.0016	481
1000	2.91	6936	1880	1918	0.0022	497
∞	2.89	7020	1851	1911	0.0026	582

steady-state response fraction. Robots that are deployed for longer periods are better able to spread their coverage throughout the map and ensure each cell is visited regularly. Also note that the choice of L has little effect on compute time, as L only has minor influence as a limitation to the range of motion in each of the planning pipeline components.

The second physical limit considered is traversal speed, v_{travel}, which corresponds to the assumed constant velocity used in the planning phases. While the actual speed of robots will vary greatly during operation, this value helps to determine the potential visitation times given the computed path length. Figure 6.13 shows how the system generates high confidence and accuracy maps more quickly when robots are able to move faster. With higher v_{travel}, the plans generated each iteration can be traversed more quickly, resulting in more planning iterations per horizon. As a consequence, more waypoints are processed each horizon, requiring more time to compute, but providing more coverage of the environment. Further evidence of these trends are visible in Table 6.7.

In addition to movement limitations, we also evaluate sensor parameters and their effect on the proposed system. Given that the robots are equipped with beam sensors, the key parameters of interest are the maximum sensor range r_{sense}, the maximum sensor span θ_{sense}, and the number of beams per vertical or horizontal span ρ_{sense}. The effect on performance of r_{sense} is shown in

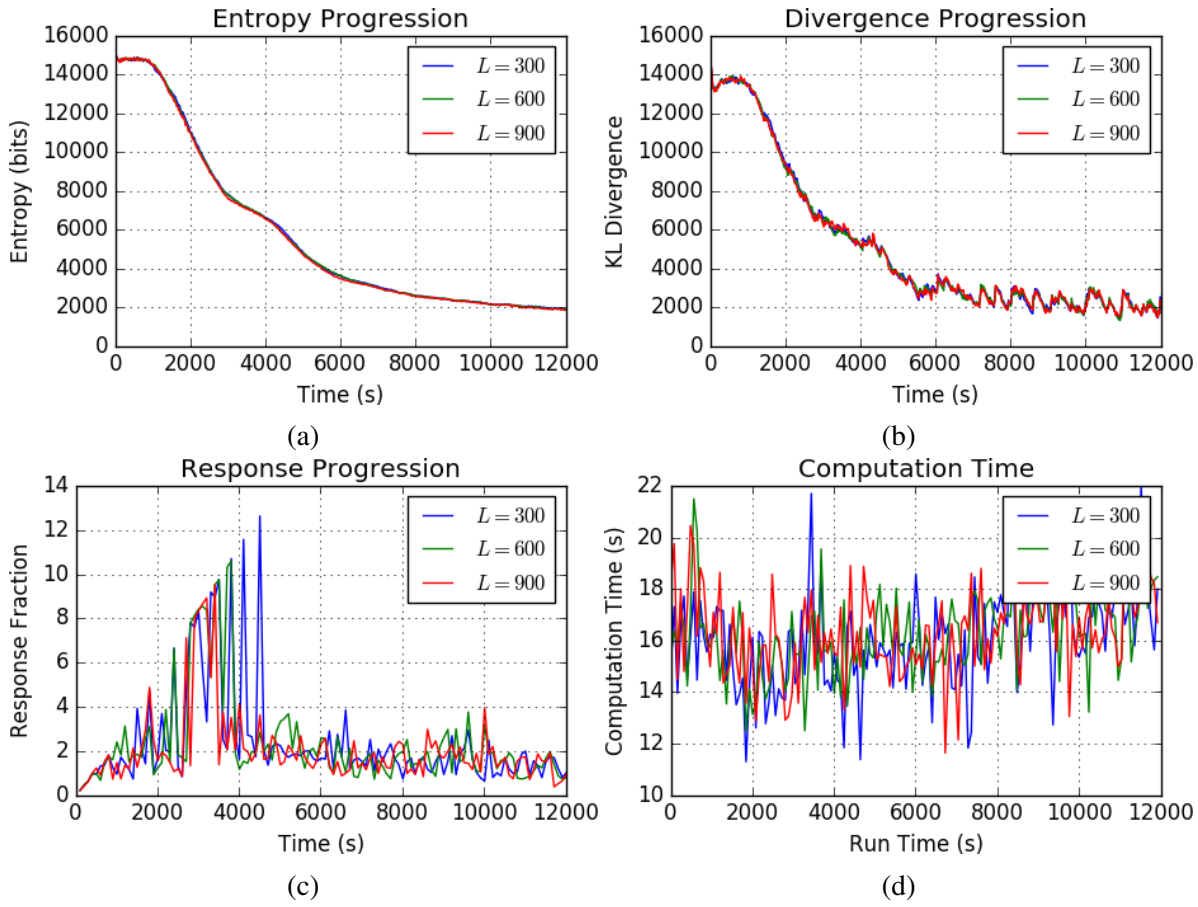


Figure 6.12: Performance analysis relative to energy capacity L expressed as flight duration, with $v_{\text{travel}} = 6$, $N_w = 6$, $N_r = 3$, $d_{\text{adv}} = 100s$, $d_{\text{hor}} = 120s$, and $\alpha = 150$. The influence of capacity is less significant than other parameters as we do not limit the number of robots deployed. However, note that the response fraction (c) converges to steady-state much more quickly when L is high. When robots are deployed for longer durations, they spend less time traveling to and from the charging station, allowing them more freedom to cover more of the environment. Also, note that compute time is consistent across all values of L as the complexity is minimally impacted by robot capacity.

Fig. 6.14, where we see entropy, divergence, and response fraction all converge significantly faster with longer range sensors. The approach, however, is still able to utilize sensors with smaller r_{sense} and learn the model well enough to converge to roughly the same steady-state entropy and divergence, albeit at a much slower pace. Computation time, however, does not exhibit significant variance, despite the additional cells that require processing as a result of farther reaching beams. As such, the proposed approach is capable of leveraging long-range

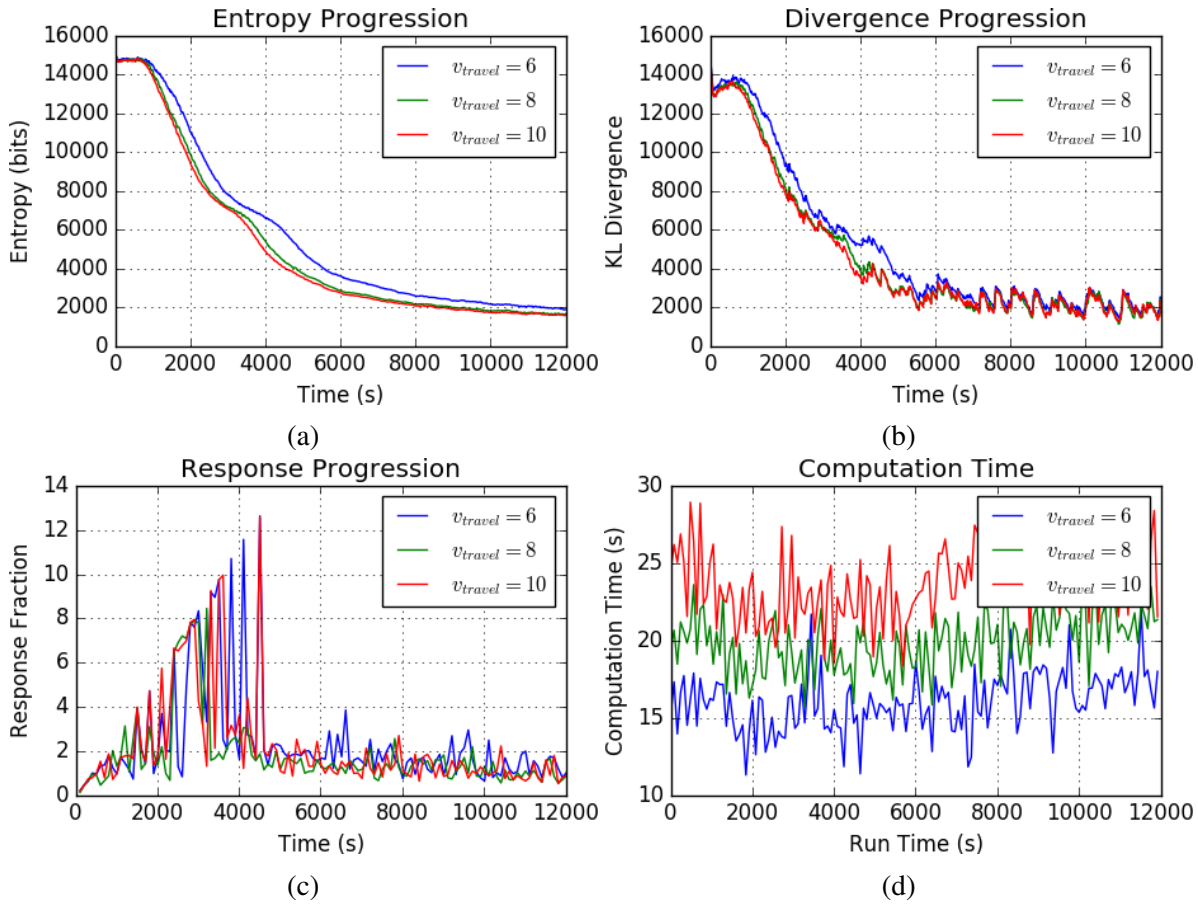


Figure 6.13: Performance analysis relative to travel speed v_{travel}, with $L = 300$, $N_w = 6$, $N_r = 3$, $d_{\text{adv}} = 100s$, $d_{\text{hor}} = 120s$, and $\alpha = 150$. Increased speed allows for routes to be completed faster, resulting in more planning iterations and more waypoints visited per horizon. As a result, entropy (a) and divergence (b) converge more quickly to lower steady-state values, while computation time (d) increases with the number of iterations performed per horizon. The response fraction (c), on the other hand, does not vary greatly with travel speed, except for a slight increase in steady-state response when v_{travel} is low.

sensors without significantly influencing the frequency of plan generation.

Figure 6.15 shows the progression of performance as a function of the span of the sensor sweep θ_{sense}. Given that beams are projected in sequence horizontally and vertically, this number represents the field of view of the sensor along the vertical and horizontal axes. As such, the wider the view used, the more cells a sensor can observe. In most cases, this results in an improvement to performance, with minimal impact on computation time as the number of additional cells observed per sensing action is minimal. Further discussion on how this effect is impacted by

Table 6.7: Comparison of steady-state performance while varying capacity L and travel speed v_{travel}. There is a clear trend of improved entropy and divergence reduction as v_{travel} increases, given that faster moving robots can cover more of the environment more quickly. There is also a minor improvement in steady-state entropy and divergence evident when L is increased, as robots expend less time traveling to and from the charging station.

L	v_{travel}	$E(N_r)$	t_{conv}	μ_H	μ_D	$E(d_r)$	N_u
300	6	2.96	7809	1718	1949	0.0012	414
600	6	2.98	7733	1719	1902	0.0012	413
900	6	2.98	7689	1706	1873	0.0013	455
300	8	2.86	6855	1545	1858	0.0013	507
600	8	2.98	6883	1540	1793	0.0012	485
900	8	2.99	6879	1521	1821	0.0010	521
300	10	2.85	6660	1503	1850	0.0011	435
600	10	2.93	6585	1495	1829	0.0010	415
900	10	3.00	6609	1492	1766	0.0008	430

ρ_{sense} is included in the subsequent paragraphs.

Although the number of beams per sweep, ρ_{sense}, is a function of the choice of sensor, a user can adjust this value by downsampling the number of beams used in processing an observation. This is an important property, as there is little benefit to having a high density of beams when multiple measurements of the same set of cells provides little extra information. In Fig. 6.16, we see a general trend of improved performance as a result of increasing ρ_{sense}, with diminishing returns observable at $\rho_{sense} = 10$. However, this comes at the cost of increased computational complexity as increasing the number of beams considered greatly increases the number of cells processed with each computation of the utility function.

In Table 6.8, we see that KL-Divergence suffers with increasing θ_{sense} when ρ_{sense} is low. Additionally note the increase in unobserved cells, N_u, as a result of the widening span. Consider the distance between the beam end locations as determined by the law of cosines:

$$a^2 = 2r_{sense}^2 \left(1 - \cos\left(\frac{\theta_{sense}}{\rho_{sense}}\right)\right).$$

Any cell with a span smaller than a in the horizontal or vertical plane of the sensor's field of view

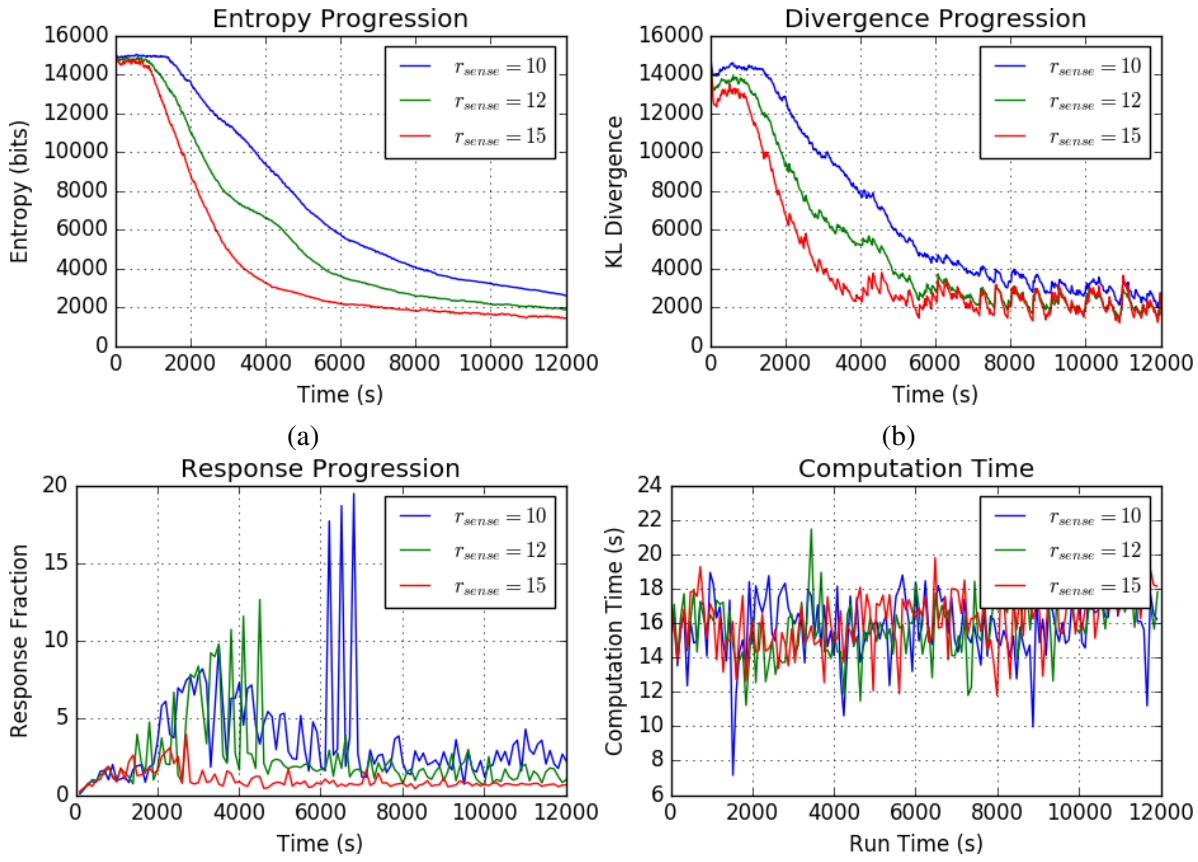


Figure 6.14: Performance when varying the maximum sensor range r_{sense}, with $\theta_{sense} = 40°$ and $\rho_{sense} = 10$. Entropy, divergence, and response fraction converge much more quickly to steady-state with long-range sensors. However, the potential addition of a few extra cells to the end of each beam does not appear to heavily influence computation time in this example.

is likely to be missed as the sensor scans across the planes. As such, the minimum number of beams per span to which an observation can be downsampled is given by:

$$\rho_{sense} = \frac{\theta_{sense}}{\cos^{-1}\left(1 - \frac{a^2}{2r_{sense}^2}\right)}.$$

With a cell length and width of 2 m, $\theta_{sense} = 60°$ and $r_{sense} = 15$, this value translates toroughly 5.5 beams per span, which corresponds to the reduction of performance we begin to see in Table 6.8.

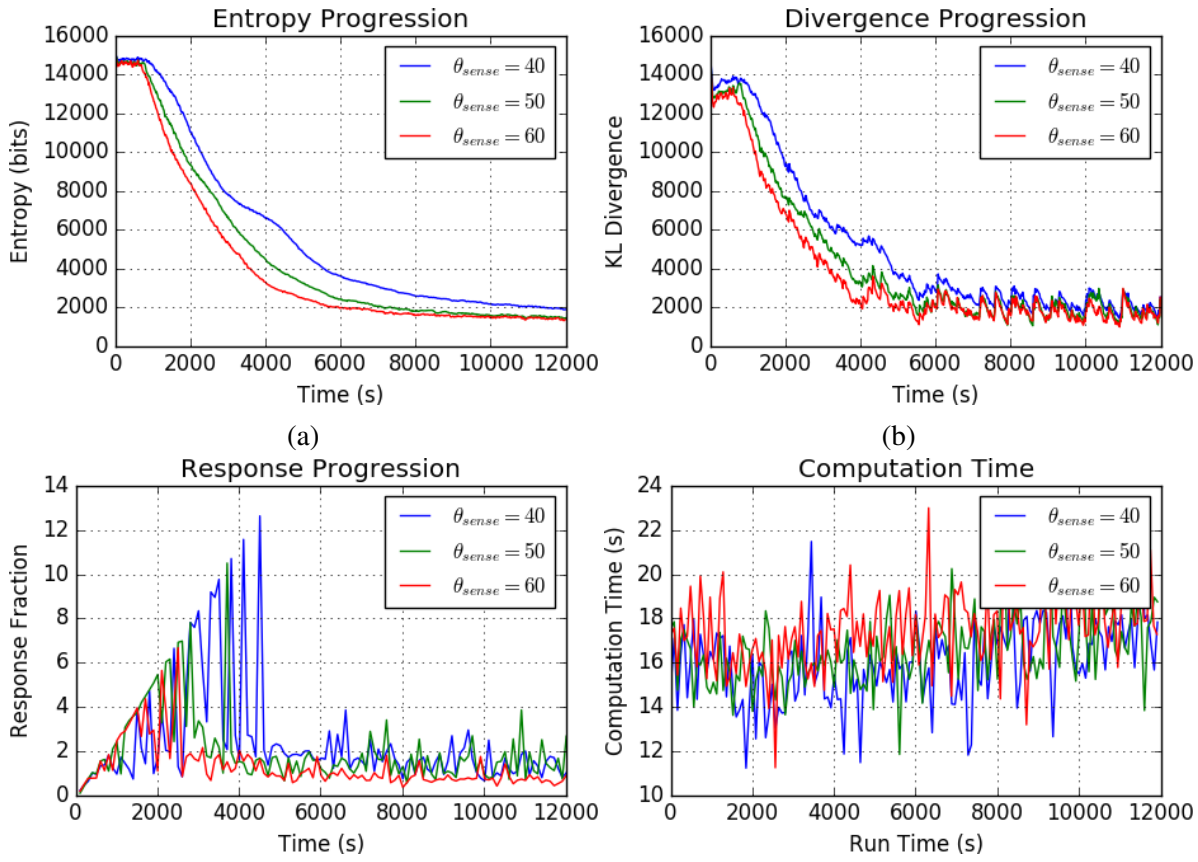


Figure 6.15: Performance when varying the span of a sensor sweep action θ_{sense}, with $r_{sense} = 12$ and $\rho_{sense} = 10$. The wider the span, the greater the width of observed space. If ρ_{sense} is high enough to keep from opening gaps in the observed region, a wider span will provide more information without significantly increasing computation time.

6.5 Environment Conditions

Another factor that can have significant impact on performance is the state of the environment and how we choose to model it. There are some model-specific parameters, such as HMM window size and occupancy likelihood prior, that we do not investigate in this work as there is little benefit to understanding their effects if we elect to change the model. Instead, we make the general comment that the more accurate a model we use, the more effective our approach. However, we can investigate with different environments under different conditions to give a sense of our performance as the environment becomes more difficult to process. As such, this section presents

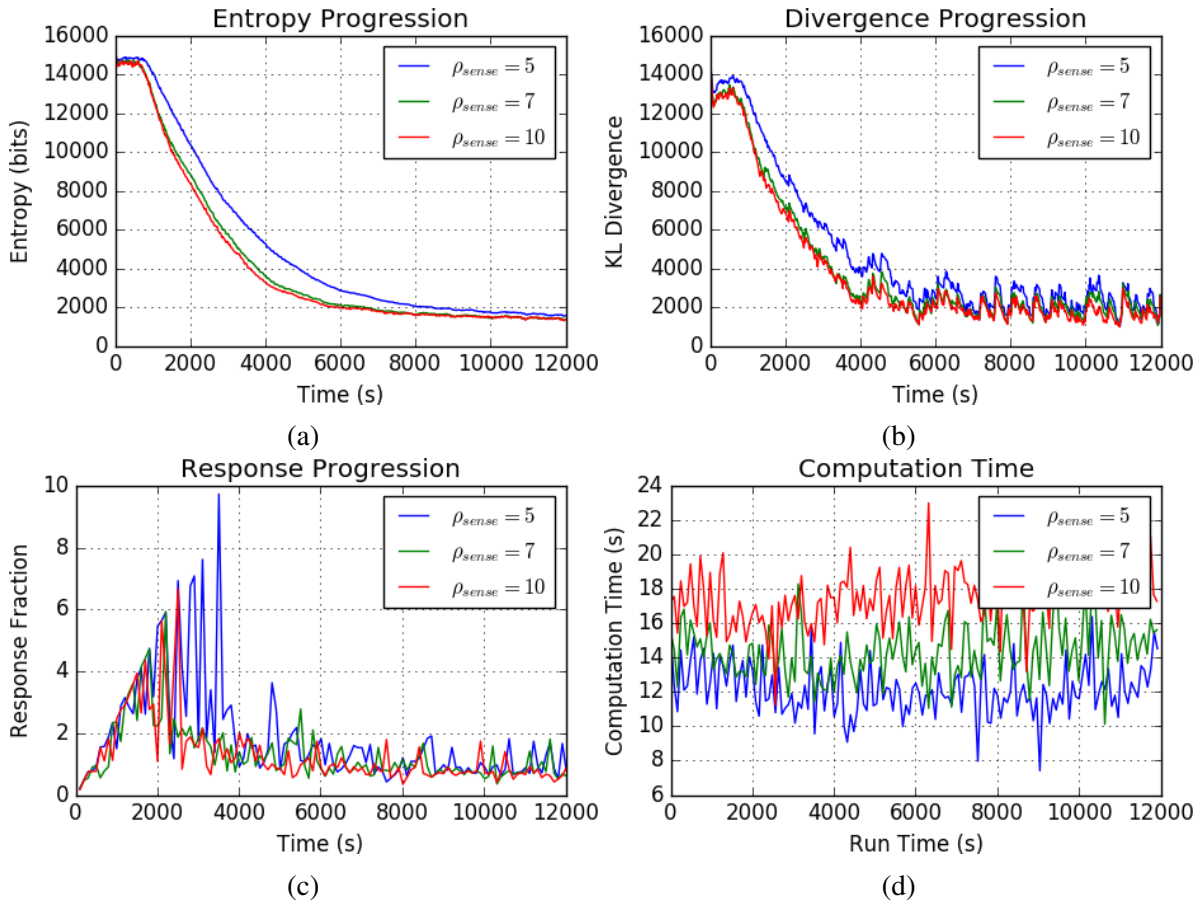


Figure 6.16: Performance when varying the number of beams per sensor sweep ρ_{sense}, with $r_{sense} = 12$ and $\theta_{sense} = 40°$. The higher this number, the more beams are extended into the same space. As a result, performance improves with diminishing returns approaching 10 beams per sweep, as more observations of the same cells will not provide significantly more information. However, each additional beam necessitates processing of another subset of cells, resulting in an observable increase in computation time.

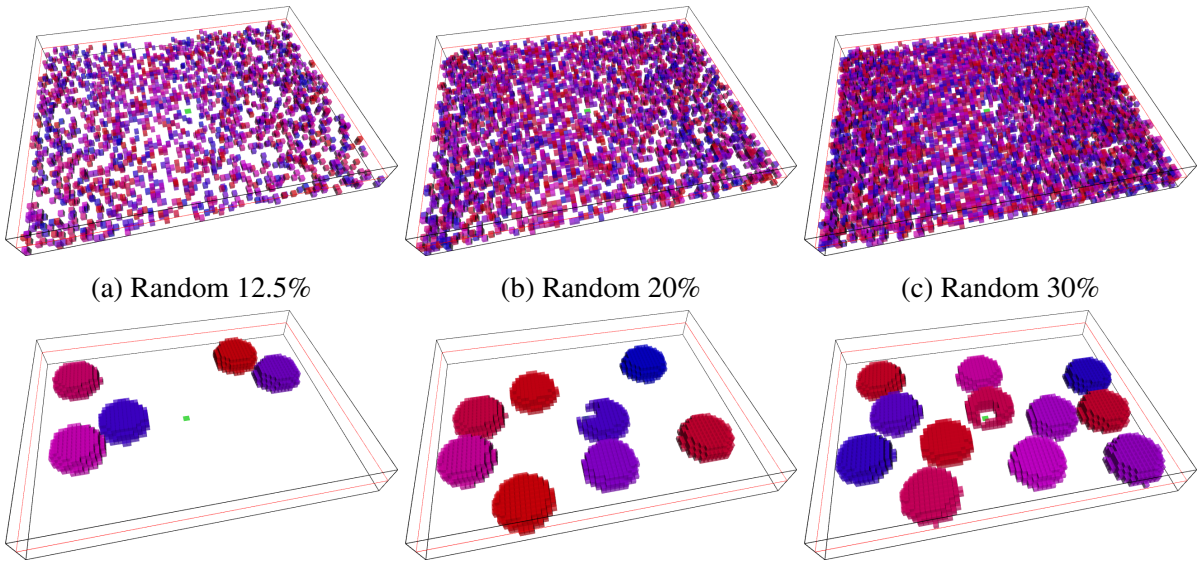
the results of testing the proposed approach under varied distributions and concentrations of dynamics.

Altering the distribution of dynamics, in this case, refers to how the dynamic cells are dispersed in the environment. We examine the six cases depicted in Fig. 6.17, for which half exhibit dynamic cells that are distributed randomly and half in which dynamic cells are clustered, with each cell in a cluster changing state at the same time as the rest of the cluster. The key difference lies in how rapidly the environment changes. When the dynamics are randomly distributed, changes are small and more evenly distributed across time. When they are clustered, the envi-

Table 6.8: Steady-state performance when varying sensor parameters. As expected, sensors with longer range, r_{sense}, and wider span, θ_{sense} significantly improve convergence rate and steady-state entropy. However, as evidenced in Figs. 6.14, 6.15, and 6.16, the steady-state convergence trends towards the same value after enough of the model has been observed. The sensor resolution, ρ_{sense}, also improves performance when increased, but we see diminishing returns for higher values as multiple measurements of the same cells provide little extra benefit. Additionally, note for lower ρ_{sense}, increasing θ_{sense} can result in an increase in the number of unobserved cells N_u in at the end of a run as beams can skip over cells, resulting worse coverage during each sensing action.

r_{sense}	θ_{sense}	ρ_{sense}	$E(N_r)$	t_{conv}	μ_H	μ_D	$E(d_r)$	N_u
10	40	5	2.94	10870	2215	2397	0.0046	528
12	40	5	2.86	8415	1853	2323	0.0023	379
15	40	5	2.97	6428	1470	2500	0.0017	193
10	50	5	2.96	7840	1657	2342	0.0015	824
12	50	5	2.96	6984	1537	2322	0.0017	709
15	50	5	2.85	5745	1335	2434	0.0014	337
10	60	5	2.97	8035	1638	2351	0.0022	962
12	60	5	2.97	6952	1478	2286	0.0015	783
15	60	5	2.86	5970	1315	2357	0.0013	324
10	40	7	2.98	10130	2025	2289	0.0032	615
12	40	7	2.96	8095	1669	2127	0.0024	426
15	40	7	2.87	5925	1287	2373	0.0012	187
10	50	7	2.95	7449	1477	2286	0.0018	785
12	50	7	2.95	6668	1369	2198	0.0016	604
15	50	7	2.97	4536	1166	2045	0.0006	374
10	60	7	2.97	7086	1429	2287	0.0014	850
12	60	7	2.97	5723	1332	1916	0.0009	658
15	60	7	2.96	4325	1123	2139	0.0009	379
10	40	10	2.94	10090	2138	2195	0.0036	588
12	40	10	2.96	7809	1718	1949	0.0012	414
15	40	10	2.96	5602	1354	1878	0.0006	191
10	50	10	2.96	7254	1469	2096	0.0015	870
12	50	10	2.97	6353	1392	1796	0.0011	593
15	50	10	2.96	4416	1145	1796	0.0006	378
10	60	10	2.85	6644	1410	2022	0.0014	862
12	60	10	2.96	5422	1360	1651	0.0010	665
15	60	10	2.85	4005	1112	1621	0.0004	370

ronment sees a significant instantaneous change, resulting in a spike in KL-Divergence. This evolution is evident in Fig. 6.18, where the clustered environments are initially as easy to process as random environments, due to the fact that most of the environment is unknown, but quickly



(a) Random 12.5% (b) Random 20% (c) Random 30%

(d) Cluster 12.5% (e) Cluster 20% (f) Cluster 30%

Figure 6.17: Testing environments differentiated by the percentage of dynamic cells outside of the safety region. These percentages do not include cells that the robots are permitted to traverse. The frequencies of change are expressed by color using the same range shown in Fig. 6.1

lose accuracy as the sudden changes begin to stack. Once the dynamics are learned, however, these spikes become less prominent and degrade faster, resulting in a more consistent representation of the environment.

To vary the concentration of dynamics, we adjust the percentage of cells in the dynamic region that exhibit changes. Figure 6.17 shows the set of environments with varying dynamic concentrations that are tested in this section. As one might expect, the more cells that are dynamic the less accurate and certain our model. This results in exacerbating the effects of clustering, causing more significant spikes in KL-Divergence. The key trait of interest, however is that there is a significant separation of the steady-state values, suggesting that there is a hard limit to how well the model can be learned for a team of robots in an environment of a certain concentration of dynamics. In future works, we will determine how to predict this limit based on environmental conditions and available resources to determine how well an environment can be modeled.

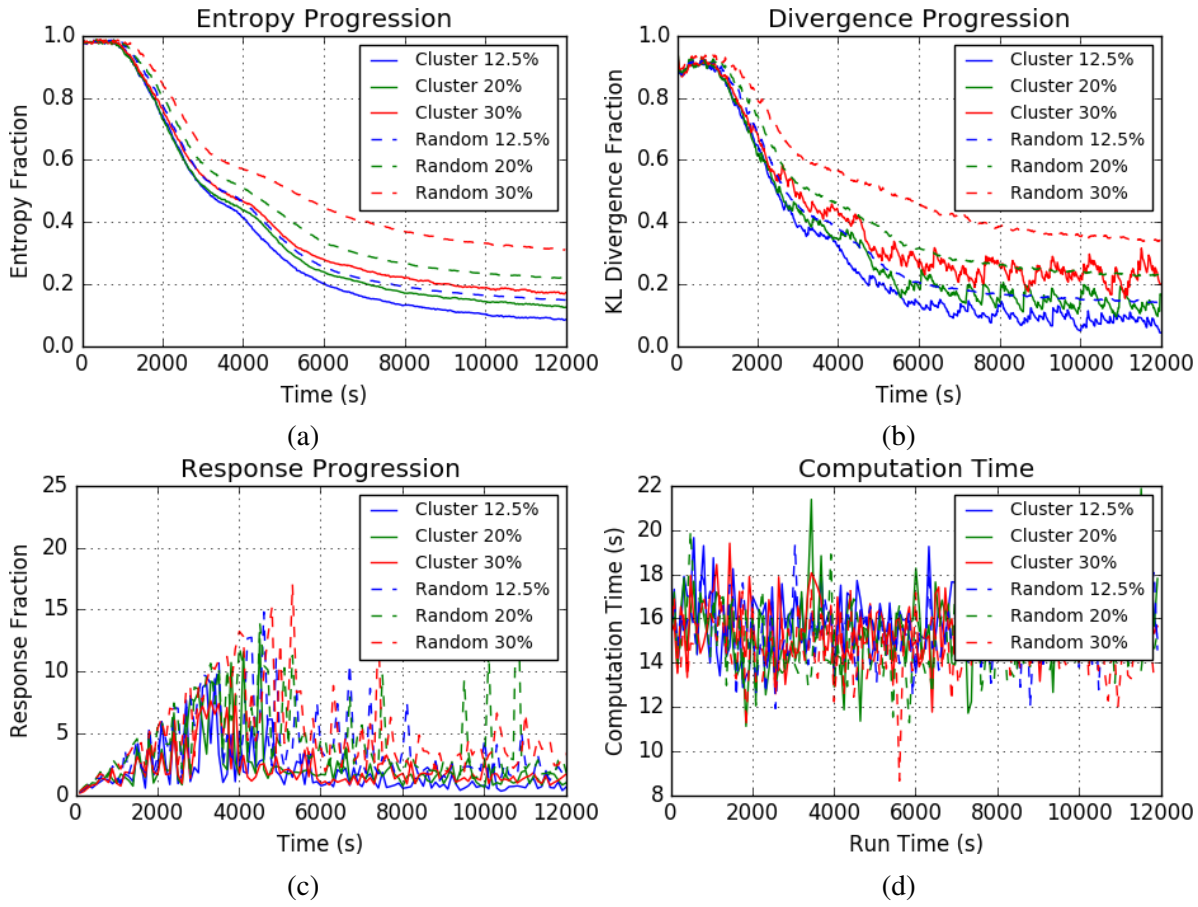
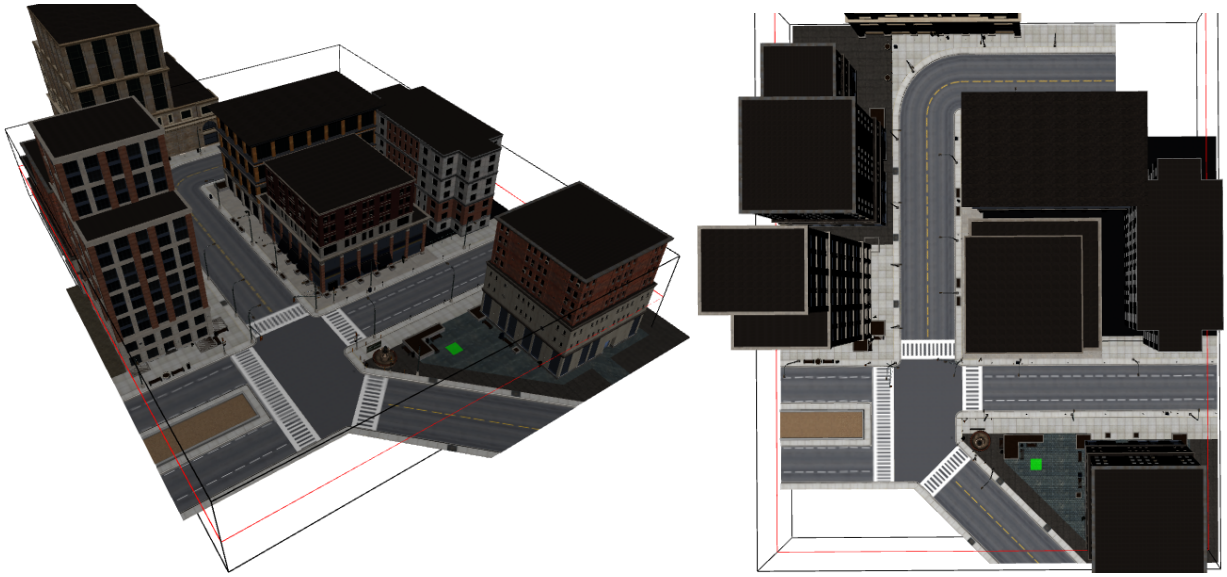


Figure 6.18: Comparison of varying environment dynamics and distributions. To mitigate the difference in entropy and divergence values for differing maps, we divide these values by the number of cells below the safety height that are not persistently occluded and show the corresponding entropy fraction and divergence fraction. As expected, environments with more dynamic cells are more difficult to process than environments with less, resulting in poorer performance for higher percentages of dynamic cells. Random environments exhibit more stable divergence than clustered environments, given that the changes are more staggered, but performance overall is worse in the random environments as cells are more often occluded. The cell dynamics can still be learned with partial occlusions, but many more observations are required to learn the dynamics accurately.

6.6 Urban Environment

To connect these results to a realistic scenario, we devised an urban environment experiencing tidal flooding that regularly impedes motion in a portion of the covered area. These tests express how the proposed approach performs in a scenario where dynamic cells are not clustered into spherical groups and the cells change states over irregular periods. Figure 6.19 shows the struc-



(a) Front view (b) Top-down view

Figure 6.19: Urban test environment, showing the front view (a) and top-down view (b). Robots deploy from the green square within the wire frame box (black lines) to map the environment. The buildings serve as static obstacles both above and below the safety line (red bordering line).

ture of the urban environment, which consists of static buildings and open pathways in the form of roads and sidewalks. The environment is represented using a $76 \times 60 \times 12$ occupancy grid where the lower $76 \times 60 \times 5$ sub-grid (below the safety height) encompasses the region where dynamics are modeled.

To express a more complex evolution of the environment topology, we expand on the previous representation of dynamics by allowing cells to exist in one state for a longer duration than the other. This property is expressed as the *duty cycle* of the cell's state transition frequency, where a cell m_i with a duty cycle $\delta_i \in [0, 1]$ and a frequency of change f_i corresponding to the period of change $\mathcal{T}_i = 1/f_i$ exists in the free state for $(1 - \delta_i)\mathcal{T}_i$ seconds before transitioning to the occupied state, then remains in the occupied state for $\delta_i\mathcal{T}_i$ seconds before transitioning back to the free state. The updated representation differs from earlier environments as each environment assumed a value of $\delta_i = 0.5$ for all cells $m_i \in m$.

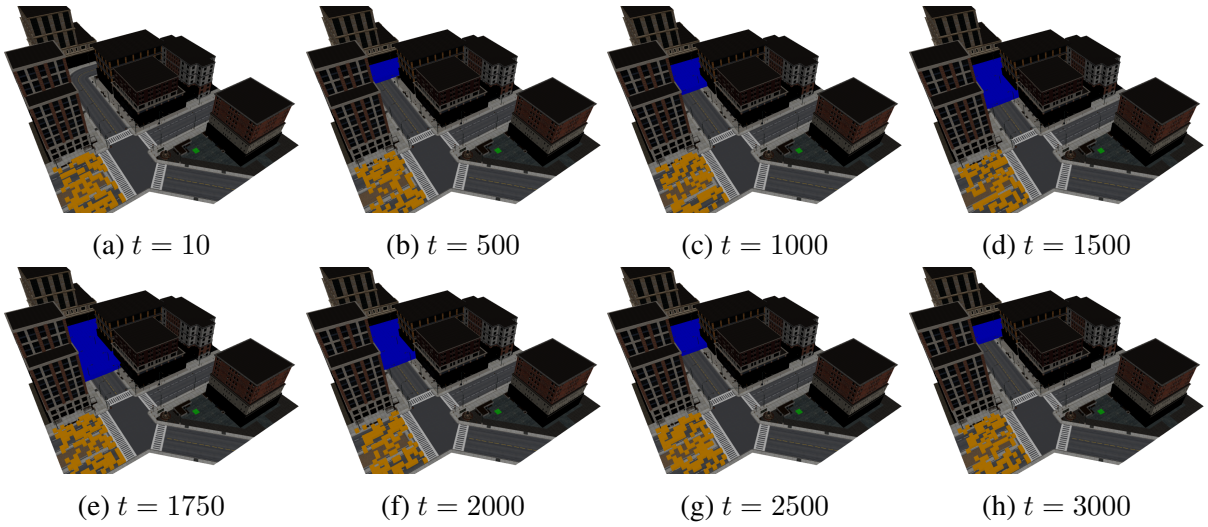
The dynamics for the scenario addressed in this section are expressed in two forms, visually represented by the vignette in Fig. 6.20, where blue cells indicate water periodically flooding

the top half of the map and orange cells represent people, vehicles and equipment being periodically moved through a region of the environment. We simplify the expression of these effects by selecting an independent frequency, phase, and duty cycle for each cell, instead of directly modeling the motion of objects through the environment. As shown in Fig. 6.21, the frequency is selected as $f_i = 1/3600$ for the entire flooded area at the top of the map, but the duty cycle decreases in stages as the flooded area extends downwards. Setting the phase for each cell to

$$\phi_i = -(1 - \delta_i)\frac{\mathcal{T}_i}{2},$$

where the phase, ϕ_i, represents the duration before cell m_i begins its first cycle in the proposed scenario, results in the staged flooding exhibited in Fig. 6.20. The region in the bottom left of the environment, corresponding to the movement of people and resources, consists of cells whose frequency, phase, and duty cycle are randomly chosen from the ranges $f_i \in [1/2000, 1/300]$, $\phi_i \in [0, 1/f_i]$, and $\delta_i \in [0, 1]$.

We evaluate the efficacy of our approach by comparing performance against the pure greedy assignment in the proposed urban flood scenario. As shown in Fig. 6.22, the greedy approach quickly decreases entropy and divergence early ($t = [2500, 6500]$) but at $t = 8000$ the Min-Makespan approach exceeds the performance of the greedy approach and is better able to respond to changes in the environment. In the Divergence Progression plot, we see the Min-Makespan approach consistently reducing divergence faster and to lower values than the greedy approach after each instance the environment undergoes a sudden change. Figure 6.23 shows how this is expressed in the environment model, as the Min-Makespan approach is capable of recognizing that the road at the top of the map is open at $t = 11000$ while the pure greedy approach models the road as mostly obstructed. This result highlights the importance of modeling environment dynamics and distributing robots appropriately, as an insufficient strategy might miss the opening of a key passageway in a scenario where the ability to move quickly through the environment is paramount.



(a) $t = 10$ (b) $t = 500$ (c) $t = 1000$ (d) $t = 1500$

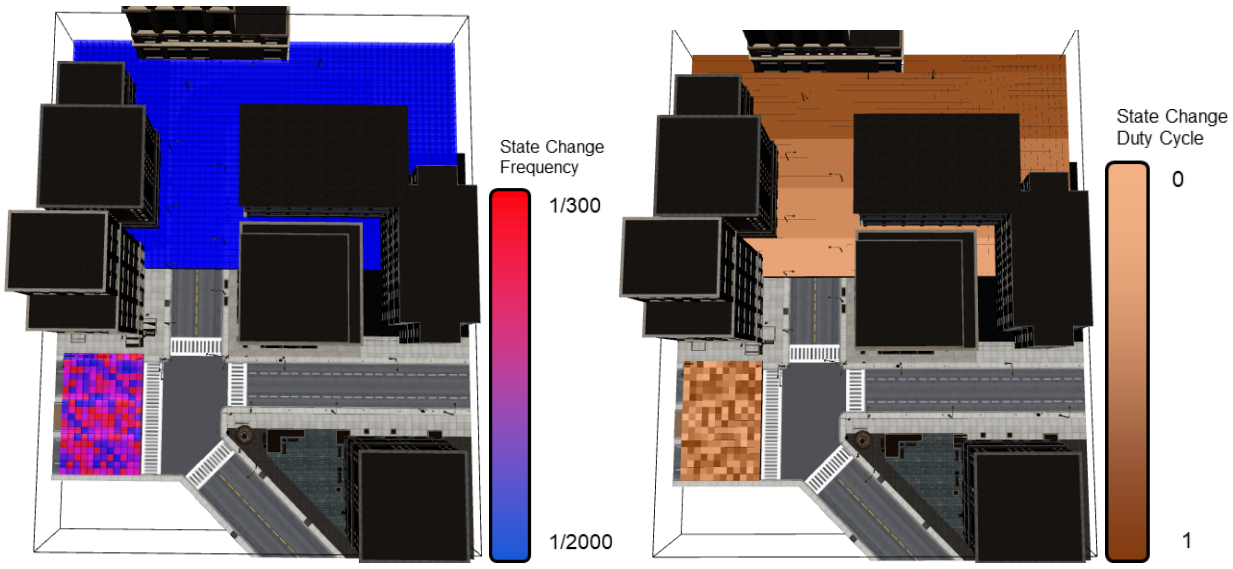
(e) $t = 1750$ (f) $t = 2000$ (g) $t = 2500$ (h) $t = 3000$

Figure 6.20: Urban scenario vignette. The representative disaster scenario expresses two different types of dynamic clusters, expressed in the vignette through cells that are periodically occupied by water (blue cells) and cells that are periodically occupied by people, vehicles, or equipment (orange cells). The area at the top of the map exhibits flooding in stages, repeatedly covering and uncovering half of the environment every hour. The area at the bottom left of the map exhibits the randomly distributed periodicity that could be expected of rescue workers and people in distress. The black cells surrounding the buildings represent the continuously occupied cells the buildings intersect.

6.7 Conclusion

In this chapter, we evaluated our systems-based approach to persistently deploying a team of robots to map a dynamic environment while accounting for each robot's limited energy capacity. We showed that pure greedy assignment is ill-suited to dynamic environments as the constant growth of uncertainty promotes the continuous observation of the same dynamic clusters. The proposed approach mitigates this issue by batch-selecting a large set of candidate waypoints that are spread throughout the environment. As a result, the proposed approach converges 22% faster to a steady-state with less than 60% of the entropy (model uncertainty) and 71% of the KL-Divergence (model inaccuracy). However, the proposed approach is heavily dependent on the appropriate selection of many parameters, which can make implementation difficult.

By evaluating the effect these parameters have on performance, though, we are able to provide the following insights to reduce this burden. First, the parameter which most directly affects



(a) Dynamic Heatmap (b) Duty Cycle Map

Figure 6.21: Urban environment dynamics. The heatmap (a) expresses dynamics as before, with red signifying fast changes and blue signifying slow changes, while the duty cycle map (b) expresses how much of each period of change a cell spends in the occupied state (an 0.8 duty cycle means a cell spends 20% of a period free and 80% occupied). The flooded area shows a consistent frequency of change across all stages, but exhibits a decreasing duty cycle as stages descend from the top of the map. This representation results in the evolution of occupancy depicted in Fig. 6.20 where the uppermost areas remain flooded (occupied) for longer than the areas below. The cluster in the bottom left exhibits randomly distributed cell dynamics, similar to the random environments tested in Sec. 6.5, but with randomly selected duty cycles as well.

multi-robot coordination, N_w, shows a minor improvement to performance as it is increased, at the cost of computational complexity. This suggests that further development is warranted in investigating the integration of more efficient, highly-coupled task assignment approaches. Second, attempting to compute short horizons quickly provides little benefit at a relatively significant cost. Instead, it is more prudent to compute over longer horizons, up to the limits the robot capacities allow. Third, the objective weight which blends CSQMI and goodness-of-fit objectives appears to vary in performance over time as the environment becomes known. It would be beneficial to incorporate dynamic parameter tuning to ensure optimal performance over all time. Fourth, performance as a function of sensor parameters can be improved by downsampling beams if there are too many to process, however the user must take care not to exclude so

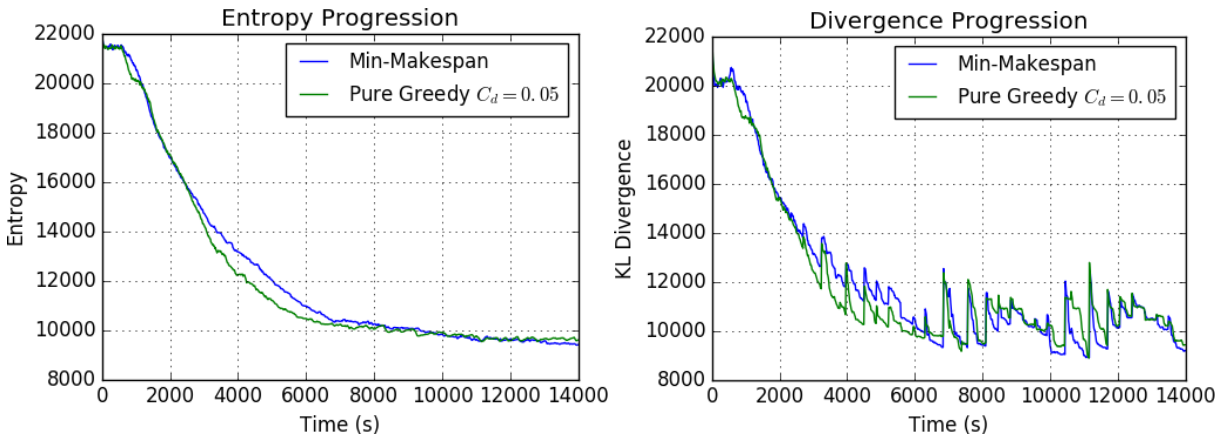
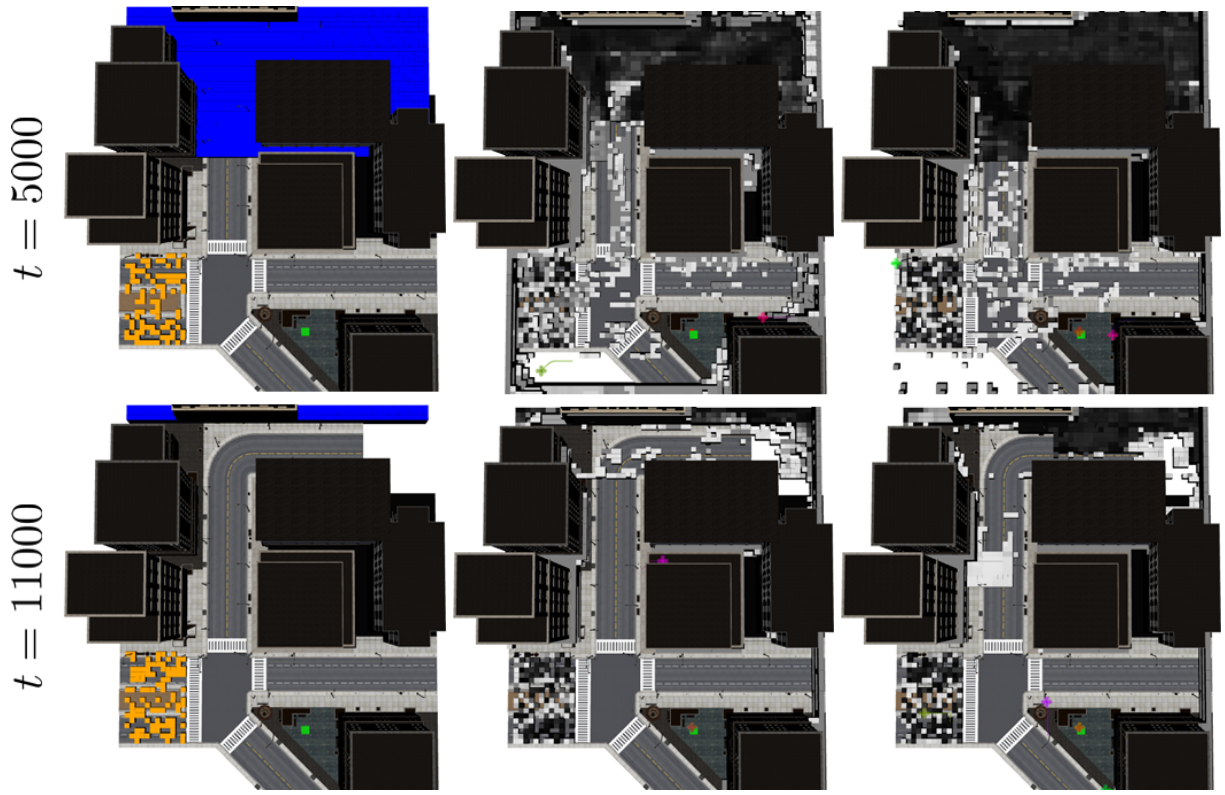


Figure 6.22: Comparison of performance between the system-based and pure greedy approaches in the urban flooding scenario, where the system parameters are chosen as: $N_w = 6$, $N_r = 3$, $d_{\text{adv}} = 100s$, $d_{\text{hor}} = 120s$, $\alpha = 10$ and the greedy distance weight is chosen as $C_d = 0.5$. While the greedy approach more quickly processes the static regions, resulting in faster convergence in the $t = [2500, 6500]$ range, the Min-Makespan system-based approach proposed in this work eventually reaches the same steady-state entropy. Divergence shows a similar trend, with the Min-Makespan more quickly converging to lower values after sudden changes.

many beams that the observation cone develops gaps. Finally, we are able to show consistent performance with convergence and steady-state values that vary over environments with differing composition and distribution of dynamics. As such, it is likely that future works will be able to leverage this information to determine roughly how accurate and confident a model can be developed in a given environment.



(a) True State (b) Min-Makespan (c) Pure Greedy

Figure 6.23: A comparison of environment models at key time frames for the Min-Makespan, system-based approach and the pure greedy assignment approach. The top row shows the true state of the environment (a) and the environment model from runs using the Min-Makespan approach (b) and the pure greedy assignment approach (c) at $t = 5000$. The bottom row shows these same visualizations at $t = 11000$. Earlier in the run, the Min-Makespan approach has a better model of the center road area (lighter and transparent blocks, indicating free space with high confidence) but models the dynamic regions more poorly than the greedy approach. Later, the Min-Makespan approach can better distribute robots to the dynamic regions and recognizes when the road at the top of the map has cleared faster than the greedy approach.

Chapter 7

Conclusion

The thesis objective is to perform persistent monitoring of dynamic environments using energy-constrained, multi-robot teams. Despite the complexity of the problem, the proposed system is able to persistently generate plans online by decoupling the per-horizon planning problem into several stages. First, candidate waypoints are collected into a list, sorted by the utility score presented in Chapter 4. Then a set of N_w waypoints are drawn from the top of the candidate pool. Pair-wise paths are computed through the environment to provide the path costs for travel between the waypoints, which are used to compute energy feasible routes through the environment. These routes extend existing plans, if possible, or create new plans that deploy the next available robot into the environment. If there is time left in the horizon or energy left in the active robots, a new set of N_w waypoints are selected from the candidate pool and the process is repeated. Otherwise, the plans are executed, the model is updated, and the planner advances to the next horizon. This process addresses the challenge of **Computational Tractability** when parameters are chosen to ensure that computation time is less than d_{adv}. **Energy Feasibility** is ensured by limiting the extension of plans by the maximum flight duration L. The last challenge, **Responsiveness** is fulfilled by the updated utility score, as measurements are distributed relative to the inherent environment dynamics once the environment is well enough known, ensuring that changes are observed soon after they occur.

7.1 Summary of Contributions

The main contributions of this thesis include:

- **The multi-robot persistent planning system**, which is capable of perpetually deploying robots to gather information efficiently enough to build maps 1.4 times more confident and accurate than greedy approaches. We discovered that the greedy approach, which allocates observation actions to robots based on the CSQMI value, is poorly suited to the HMM occupancy grid used in this work. CSQMI guarantees complete exploration if given enough time; however, the guarantee only applies for a static environment. In a dynamic environment, in which certainty is constantly decreasing, a robot can apply too much focus on nearby dynamic regions to the detriment of the rest of the environment. The proposed approach mitigates this issue by batch selecting waypoints to avoid a bias towards the top scoring candidates. By spreading attention wide instead of deep, the dynamics model is learned early, the growth of uncertainty degrades, and robots can focus on the more active regions.
- **An improved observation selection objective** that combines CSQMI and Pearson's χ^2 test to leverage the high confidence performance of CSQMI along with the improved coverage of χ^2. The key innovation is that CSQMI, which is designed for static environments, interacts poorly with an HMM-based occupancy grid. If too many measurements agree in too short a span of time (cell appears to be in one state too often), the model can appear to have high confidence in an incorrect model. The χ^2 component mitigates this issue by promoting more measurements in regions where observations disagree with the dynamics model.
- **A complete analysis of the system components**, including computational complexity and the influence of parameter settings on performance. From this analysis, we can conclude that the system would benefit from higher coupling (greater N_w), but may need to rely on suboptimal heuristics to ensure computational tractability. Additionally, the best per-

forming choice of selection objective weight, α, appears to vary in performance over time, suggesting that performance could be maximized with online tuning. Finally, we see a distinct variance in steady state performance as the density of dynamics changes. With enough data, it might be possible to predict the steady-state performance the system could provide based on the physical limits of the robot team and the environment conditions.

7.2 Future Work

As for future directions, there are many ways the proposed system can be improved, but four areas in particular could benefit from further investigation. First, the choice of environment model was made to simplify the planning problem in early stages of development, but there are some promising modeling approaches that could better represent a complex environment without fixing the scale or resolution. Second, the *Waypoint Assignment* approach we present relies on an optimal solver to distribute waypoints; however, tighter coupling between larger sets of waypoints might be achieved with sub-optimal, heuristic approaches that can quickly find solutions to such complex problems. Third, the proposed approach assumes a constant number of deployed robots over all time, which can ensure continuous operation by deploying a fixed fraction of the available team. Instead, a more efficient approach might predict the upcoming demand and adjust the deployment team size online to better react to particularly active phases, or keep more robots on standby when the environment is expected to remain stable. Finally, the range of coverage can be greatly increased by establishing multiple charging areas within reachable flight distance of one another, assuming we are able to mitigate the resulting increase in computational complexity for planning optimal routes. The following paragraphs provide a bit more detail on the challenges that can be expected in pursuing these goals.

Updating the Environment Model The main limitations of occupancy grids lie in the fixed resolution and the assumption of independence between cells. As a result, it can be difficult to

model environments that contain features of significantly varying scale. Octomaps are a promising option that will not require a significant change to the approach and Hierarchical Gaussian Mixture Models provide a similar benefit without fixing the location of each unit of volume; however, both come with two major limitations that must be resolved. First, without a fixed volume or location, the system must have some means preserving information across time-steps in order to encode environment dynamics. Every time a new measurement is taken, the model must decide whether the measurement belongs to previously observed volumetric units (cells or Gaussians within the hierarchy), or it corresponds to a new volumetric unit. Second, a utility measure (both CSQMI and χ^2, or some related measure) must be provided for this new form. A naive approach would be to locally generate a temporary, fixed-resolution occupancy grid from which to derive utility, but this approach would limit the benefit of a new model to a significant degree.

Efficient Waypoint Assignment There are many heuristic approaches to task assignment that provide fast solutions at the cost of optimality. The main challenge is determining the how much optimality must be compromised to ensure online planning. The greedy waypoint assignment approach presented in Sect. 5.5.1 is a naive option, but more efficient or more optimal approaches may exist. Alternatively, one could reduce the degree of coupling or the number of waypoints serviced each horizon and instead focus on upgrading the informative quality of trajectories traversed between waypoints. While the *Path Planning* phase of the proposed system is too deeply integrated to allow for the computation of complex, informative paths, it is possible to revise computed paths at the end of each planning iteration to improve their informative quality. Then, if the system is able to compute subsequent planning iterations relative to these updated paths, the impact of waypoint choice would be reduced allowing for the planner to generate high-utility plans with low values of N_w and/or fewer planning iterations.

Long-Duration Demand Management In order to efficiently distribute effort across many planning horizons, the system must determine how many robots to deploy in each planning horizon relative to the demand of the environment, where demand is a function of predicted utility of observations in the environment. This requires a definition of demand relative to utility and an objective function that determines team size relative to the demand. Appendix A presents an early approach to addressing the latter problem relative to a user defined demand in the form of desired formation flight behaviors. Future work would then look to defining demand in the presence of a probabilistic environment model and leveraging awareness of the system's full energy state balanced against demand to determine deployment team size at any given time. Doing so could significantly increase performance in scenarios where the environment experiences sparse, but regular instances of particularly high levels of activity by allowing the system to keep robots in reserve in preparation for periods of high demand.

Multiple Charging Areas In this thesis, we assume the existence of a single charging area from which all robots deploy. However, it may be desirable to establish multiple charging areas sparsely distributed throughout the environment to greatly increase the range that a team of robots can cover. Given that the range of a robot is limited to half the distance traversable on a single charge from a charging area, placing multiple charging areas within range of each other allows each robot access to the union of all reachable distances from nearby charging areas, as shown in Fig. 7.1. While the benefit is great, there is also a significant increase in complexity as the planner must decide from which area to deploy and to which area to return each time a plan is generated. Additionally, when the team size is finite the planner must appropriately manage the distribution of robots among charging areas to ensure a timely response to unexpected changes in the environment. As transitions between charging areas that are sufficiently far away necessitate a recharging cycle before a robot can continue exploring or transition to another charging area, redistributing robots over multiple transitions between charging areas quickly becomes prohibitively time consuming. A viable planner must be able to predict the demand at each charging

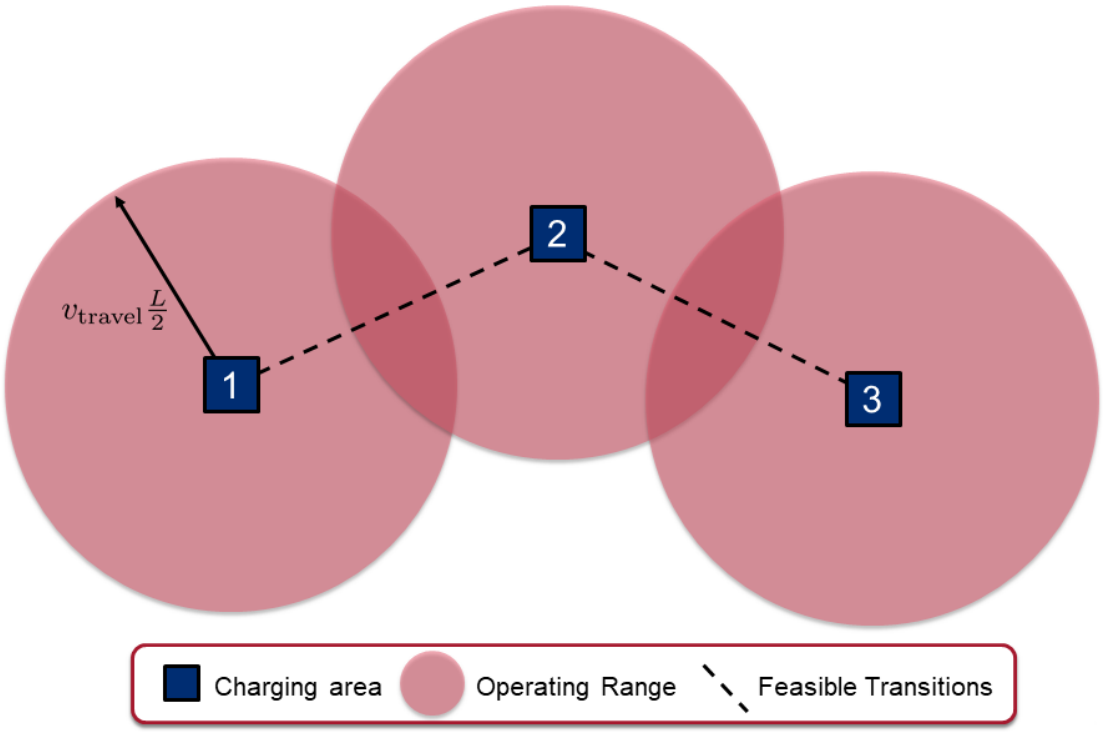


Figure 7.1: Visual representation of multiple charging areas. A robot is only capable of traveling a finite distance from a charging area ($v_{\text{travel}}\frac{L}{2}$ for a constant travel speed v_{travel} and a maximum flight time L) if it is to return to the charging area before depleting energy reserves. With multiple charging areas, a robot can transition between areas as long as the transition can be accomplished on a single charge. However, such transitions may require complex and time-consuming maneuvers as moving from one area to another may necessitate multiple transitions and recharging cycles (as would be necessary if transitioning from charging area 1 to area 3 in the figure above).

area and ensure that robots with full energy capacity are available where and when they are needed.

Appendix A

Online Energy-Constrained Adaptation and Scheduling of Persistent Coordinated Behavior-Based Multi-Robot Deployments

In this chapter, we present a collaborative effort with Ellen Cappo and Arjav Desai that highlights how a persistent deployments can be planned in a manner that factors in energy considerations over multiple planning horizons. In the proposed application, robots are tasked with performing formation flight behaviors that are decided by a user in an online fashion. Persistence is achieved using the trajectory refinement approach proposed in [38], which permits the planner to overwrite behaviors with robot swapping actions as necessary. The key contribution of this work is an approach that uses the energy state of the full robot team to determine the subset of a set of behaviors that are feasible. We expect this approach of planning relative to the multi-robot energy state to inform future development of the system presented in earlier chapters.

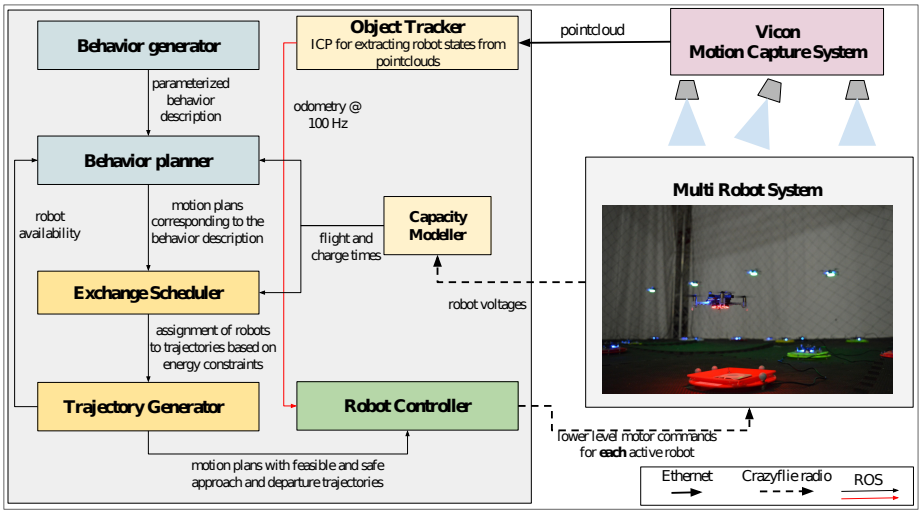


Figure A.1: System architecture.

A.1 System Overview

The proposed planning framework for achieving long duration autonomy with a multi-robot system subject to energy constraints can be subdivided into four components. The first two components, the behavior generator and the behavior planner, generate behavior commands and map the behavior commands to dynamically feasible and collision free trajectories that do not consider limitations on energy capacity. The third component, the exchange scheduler, cuts the trajectories into energy feasible subsections and assigns robots to individual trajectories. The fourth component, the exchange trajectory generator, computes the approach and departure trajectories for the assigned robots. These trajectories are then finely discretized into a set of position setpoints. The controller uses these position setpoints and the state-estimates of the robots received from a motion-capture system to compute the lower level commands for each active robot. These lower level commands are transmitted to the robots by means of a 2.5 GHz radio. This process is illustrated in Fig. A.1, and the subsequent subsections describe each of the components in detail.

Behavior Generation We choose to use *behavior commands* in order to direct multi-robot motions in an online context [56]. A behavior command, $\vec{b}$, is a fixed length vector containing M

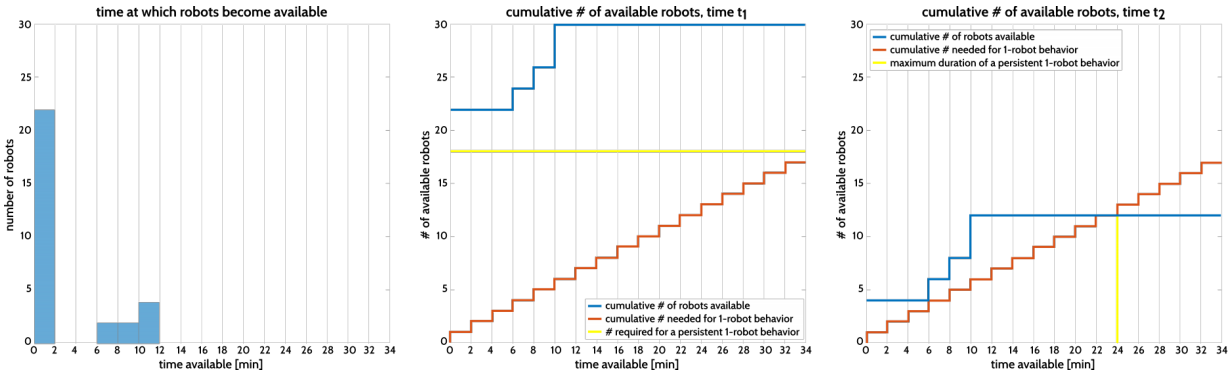
behavior descriptors: $\vec{b} = [b_1, \ldots, b_m], m \in \{1, \ldots, M\}$. Each descriptor, b_m, describes a facet of the desired overall multi-robot ensemble motion. To give an example, descriptor categories can include the number of robots making up a group, formation shapes for groups of robots, speeds governing group movement, or group actions (such as moving to a target destination or performing a repeated motion pattern, including traveling in a circle or moving side to side). Each behavior descriptor, b_m, may take a discrete number of values, and we denote B_m as the set of values associated with the behavior descriptor b_m. The total number of potential behaviors achievable by the system is therefore the product of the cardinality of each set:

$$\text{perm}(\vec{b}) = \prod_{m=1}^{M} |B_m| \, . \tag{A.1}$$

Behaviors allow a user to direct ensembles of robots without requiring the user to specifically consider low level details of multi-robot motion planning such as potential collisions between robots and the environment or the dynamic limits of the hardware platforms. Specifically considering the physical extents and actuator limits of a vehicle is non-trivial for a human user, especially in an online context where commands may need to issued quickly and without prior planning. The methodology presented in [56, 57] provides a planning system that translates user-specified behaviors in an online context into dynamically feasible and safe motion plans which reflect the user's motion intent.

Estimating Energy Feasible Behaviors

Another physical constraint which we seek to abstract from the user's consideration is a vehicle's battery capacity. In this section, we propose an update rule for the set of behaviors to reflect the energy constraints of a robot team. This allows a user to specify motions for homogeneous groupings of robots from an estimated energy-feasible subset of behaviors. The large set of potential behaviors makes precomputing all possible behavior transitions, even at discretized time instances, infeasible. Therefore, while the exact energy requirements of transitioning to



(a) Histogram of robot availability.

(b) Robot availability as a cumulative sum over time.

(c) Updated set of robot availabilities after a scheduled behavior.

Figure A.2: Updating the set of energy feasible behaviors for a 30 robot team. Each robot has a flight time of 2 minutes and a charge time of 34 minutes; 18 robots are required to perform any persistent behavior. (a) A histogram of the current energy state of the 30 robot team, where robots are allotted into bins based on time to full charge. (b) Energy requirements for a behavior can be compared to the energy state of the team using the *cumulative availability* of the robot team over time. The blue line shows the cumulative number of robots available with increasing time. The red line shows how many robots are needed with time to sustain a persistent behavior requiring a single flying robot. For reference, the yellow line shows the number of robots required for a persistent behavior. (c) The updated availability after a persistent behavior is assigned. Here, the blue curve does not remain above the red line, showing that there are not enough robots remaining to perform a second persistent behavior. However, a single-robot behavior could be performed for a maximum duration shown by the yellow line.

a behavior cannot be predetermined, the proposed methodology ensures with high probability that the energy capacity across the number of available robots is sufficient to cover the desired behavior trajectories.

In this work, we assume that all robots are homogeneous. This assumption allows a user to choose a quantity of robots to direct rather than specific robots. To represent an energy feasible behavior, we therefore create a descriptor set which combines the concepts of how many robots may be directed and for how long.

While the actual energy expenditure of a proposed behavior trajectory will depend on the vehicles' states at the time the command is issued, we can approximate behavior energy usage by reasoning in terms of proposed flight and charge times. A robot, on average, will be able to fly for time t_{flight} using a single battery. That battery, in turn, will recharge at a measurable rate,

allowing us to determine an average time to recharge, t_{charge}. The number of robots needed in order to cover a single robot flight plan persistently, N_p, is then a function of the ratio of charge time to flight time (rounded up to the nearest integer, to describe that we use integer numbers of robots and cannot use a portion of a robot).

$$N_p = \text{ceil}\left(\frac{t_{\text{charge}}}{t_{\text{flight}}}\right) + 1 \tag{A.2}$$

The "+1" term describes the fact that one additional vehicle beyond the capacity ratio is required to ensure full coverage. As a simple example, consider the case where a vehicle battery takes the same amount of time to recharge as to deplete. In this case, two robots would be needed to cover one job persistently; while one robot performed the assigned task, the other would recharge. Therefore we define a *persistent behavior* as a behavior which allows for the first robot in a set of N_p robots to have fully recharged its battery by the time the last robot has depleted its battery, allowing a set of N_p robots to fully cover a behavior plan of indefinite length.

To create a set of energy feasible behaviors, the behavior planner queries the exchange scheduler to learn the scheduled times at which each robot in the team is estimated to both have full battery capacity and become available, i.e., will finish covering any previously specified plans. Because the behavior planner treats robots as homogeneous, it does not need to keep track of the availability of specific robots, and can simply represent all availabilities across the team as a simple histogram, as shown in Fig. A.2a.

The series of figures in Fig. A.2 illustrates the concept of using robot availabilities to determine energy feasible behaviors, and the process of updating the set after choosing a behavior. Fig. A.2a shows a histogram of robot availabilities, given as the time duration from query time $t_1 = 0$ at which a robot will become available. Robots are sorted into histogram bins of duration t_{flight}, and illustrated is a 30 robot example, where robots have a flight time of 2 minutes and a charging duration of 34 minutes, requiring 18 robots to perform a persistent behavior. A persistent one-robot behavior is shown in Fig. A.2b as the cumulative robots needed over time;

this function is shown in comparison to the cumulative robot availabilities, as described in the histogram of Fig. A.2a.

We define the set of energy-feasible behaviors as behaviors having combined robot-time requirements which lie under the curve of cumulative robot availabilities (Figs. A.2b and A.2c). In Fig. A.2b, the team of 30 robots will have enough energy capacity, given that robots who expend their batteries immediately recharge, to cover a one-robot persistent behavior, as shown. If the user chooses to direct the system to perform a behavior using a single robot which requires t_{charge} or greater duration for recharging, the updated set of energy-feasible behaviors is shown in Fig. A.2c. At time t_2 (shown without loss of generality as $t_2 = 0$), 17 robots have been assigned persistently by the scheduler to cover the desired one-robot behavior. As shown, a second persistent behavior is infeasible, as the number of unscheduled robots are fewer than N_p. However, a user could choose to specify a single-robot behavior which only lasted 28 minutes in duration, as this behavior is fully supported by the scheduled availabilities across the robot system.

Online Capacity Estimation

In order to update the conservative estimates of the flight and charge times, we monitor voltage signals from each robot and modify flight durations such that certain predefined voltage thresholds are not crossed. Typical batteries operate nominally within a specified safe voltage band and the profile of voltage decay at a constant current is relatively consistent. As such, we present a simple approach to adjusting flight and charge duration based on updating these values when voltages outside of the safe band are sensed. Fig. A.3 shows an example of the voltage profile for four robots during flight. The white band indicates the "safe" region, where our batteries perform nominally. During flights, where the voltage is trending downwards, the times associated with the threshold crossings are recorded until we see a significant positive change signaling a switch to charging. The last crossing serves to identify the end of safe flight, where any extra flight time is considered detrimental to the system. The duration between takeoff and this safe flight end

time indicates the new safe flight duration that is now factored into the scheduling algorithm. When considering charge durations, however, we consider profiles that fail to cross the upper (blue) threshold. This indicates that a robot took off before it was finished charging. In this case, we increase charge duration by a small amount until robots are able to charge fully before being scheduled to take off.

We note that we operate under a homogeneous vehicle and battery assumption, and treat all voltage measurements from all robot-battery pairings as measurements of a "system wide" battery model. While individual vehicle-battery pairings might exhibit slightly different capacity profiles, we choose to use a single measure of t_{flight} and t_{charge} across all vehicles to simplify calculations and to gather a greater number of data points for faster system adaptation. We believe this is a reasonable assumption for two reasons. First, both vehicles and batteries are homogeneous (respectively) and the flight burden is shared approximately equally among the robot team; no one robot-battery pairing will see significantly greater use than any other, and we therefore expect batteries across the team to degrade similarly. Second, poorly performing vehicles and/or batteries can be repaired or replaced while the system is operating. This allows low-performing outliers to be corrected, and vehicle-battery pairings will therefore not fall far outside a common mean.

Exchange Scheduling Continuous operation is accomplished by scheduling robot exchanges such that the behavior trajectories are followed as strictly as possible while ensuring that no robot is scheduled to expend more energy than is available. This is accomplished by assigning subsections of the behavior trajectory to robots with enough energy to perform them. To ensure rapid response to human input behaviors, we rely on a greedy, online scheduling and assignment algorithm to quickly generate a feasible plan. The assignment algorithm is a straightforward application of the Hungarian algorithm [58]. Behavior trajectories are assigned to the closest available robots, where available refers to fully charged robots or robots who are in active operation. Robots in the process of charging are not considered for assignment until they completely

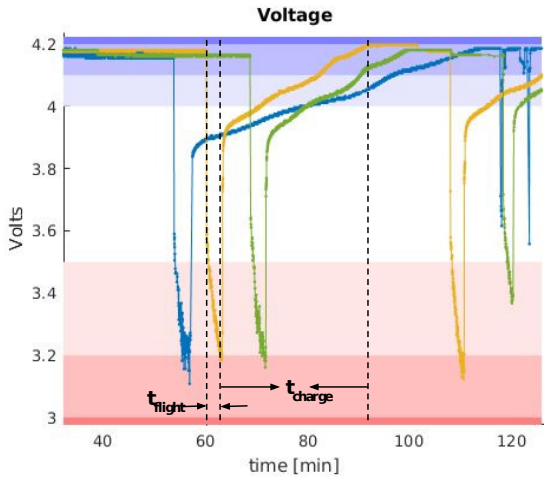
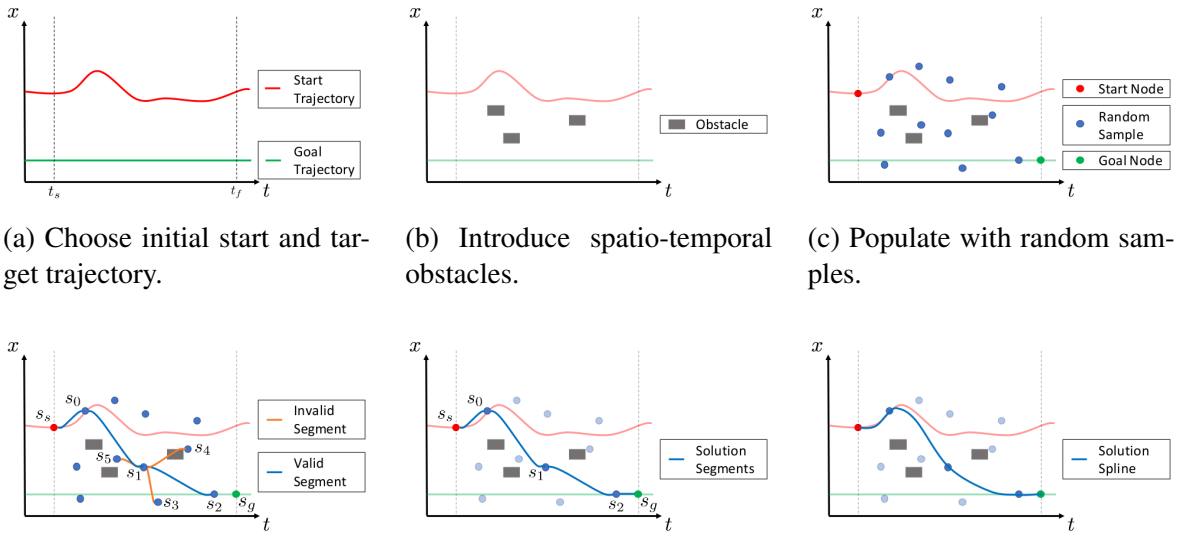


Figure A.3: Three voltage traces from three different quadrotors during flight. All batteries and robots are the same model, and have been operating over a similar lifespan. This plot shows the voltage drop for the various batteries during each robot's respective flight, and the rate at which recharging occurs. The red shaded areas of the figure indicate regions of low battery voltage, with increasingly depleted voltage shaded light to dark. The blue shaded areas of the figure indicate regions of high, or "full" battery voltage, with increasingly charged states shaded light to dark.

charge.

The scheduling algorithm is a recursive approach with three basic steps. First a depletion condition is discovered when a robot is scheduled to operate for longer than its capacity allows. The algorithm then iterates backwards in time from the full depletion time until a feasible exchange is found. An exchange is feasible if the departing robot is capable of reaching the target depot before the established depletion time and the robot on the target depot is fully charged. Once the exchange is found, dynamically feasible, collision-free trajectories for each robot are computed. These steps are repeated until no more depletion conditions are found.

Trajectory Generation As the majority of each robot path is determined by the assigned behavior, only the transitions between motions need to be computed. Such transitions can occur from charging station to behavior, behavior to charging station, or between separate behaviors. Viable trajectories for these transitions must be continuous, smooth and collision-free. To achieve these conditions, we rely on a specialized implementation of Batch Informed Trees (BIT*) [59], a



(a) Choose initial start and target trajectory.

(b) Introduce spatio-temporal obstacles.

(c) Populate with random samples.

(d) Some potential segments are invalid.

(e) Find base solution with static waypoints.

(f) Update solution with relaxed constraints.

Figure A.4: Trajectory generation process.

sample-based motion planner, to generate a series of waypoints[1]. The search space is customized, as described below, such that the resulting waypoints form a continuous, smooth piece-wise polynomial.

At the basic level, the described application can be modeled by choosing a state space that includes three Euclidean dimensions and one time dimension. Obstacles are represented as spatio-temporal regions in the state space through which an edge between nodes may not be drawn. A representative example with one spatial dimension is shown in Fig. A.4.

This example represents a robot in flight planning a trajectory to land. In Fig. A.4a, the red trajectory, $f_s(t)$, describes the planned robot flight path and the green trajectory, $f_g(t)$, shows the robot landed. States within this space are expressed as a combination of spatial and temporal components

$$s = \{t, x, y, z\}. \tag{A.3}$$

World spatial boundaries can be defined by the size of the space in which the robots are operating. The bounds on time within the context of the trajectory generation problem are defined by the

[1]The planner implementation itself is drawn directly from the OMPL library [60]

Algorithm 7 RRT

1: **procedure** RRT(s_{start}, s_{goal})
2: $V \leftarrow s_{start}$
3: $E \leftarrow \emptyset$
4: $\mathcal{T} \leftarrow (V, E)$
5: **for** $i = 1, \ldots, N$ **do**
6: $s_{\text{rand}} \leftarrow \text{SampleFree}_i$
7: $s_{\text{nearest}} \leftarrow \text{Nearest}(\mathcal{T}, s_{\text{rand}})$
8: **if** motionValid($s_{\text{nearest}}, s_{\text{rand}}$) **then**
9: $V \leftarrow V \cup \{s_{\text{rand}}\}$
10: $E \leftarrow E \cup \{s_{\text{nearest}}, s_{\text{rand}}\}$
11: **end if**
12: **end for**
13: **return** $\mathcal{T}$
14: **end procedure**

start and goal states, both drawn from their respective trajectories. These are designated in the plots by dashed lines at the start time t_s and the goal time t_f.

Obstacles, in the state space, are defined in both space and time. In Fig. A.4b, these are shown as boxes in x and t. However, the obstacles we typically consider are robot bounding volumes whose positions in space are determined piece-wise continuous polynomial splines. This is how we avoid pre-planned behavior while computing approach and departure trajectories for swapping robots.

Advancement through the RRT process is shown in Figs. A.4c - A.4e. For reference, the steps of basic RRT [61] are outlined in Algorithm 7. RRT evolves by sampling the space for valid states, matching each state to its nearest neighbor, and checking for valid motion along edges between states. Each valid state s_{rand}, with a valid edge $\{s_{\text{rand}}, s_{\text{nearest}}\}$, is added to the list of vertices V and edges E of the tree $\mathcal{T}$. Random sampling occurs in the SampleFree_i function on line 6 as shown in Fig. A.4c. Samples are drawn from a uniform distribution within the space and time bounds, but rejected where obstacles are defined. The start and goal nodes are states

constructed from the polynomial splines between which we are computing the transition

$$s_s = \{t_s, f_x(t_s), f_y(t_s), f_z(t_s)\} \tag{A.4}$$

$$s_f = \{t_f, f_x(t_s), f_y(t_s), f_z(t_s)\}. \tag{A.5}$$

Note that in this example, we show only one start and goal state. However, it is possible to seed multiple start and goal states from the splines if we wish to increase the flexibility of our approach. Each start state will act as the root of a separate tree and each will grow towards all goals, the optimal solution attributed to the start-goal pair which produces a viable path with the minimum path length. In practice, we define a buffer duration d_{buff} and a desired start $t_{s,\text{des}}$, where $t_s = t_{s,\text{des}} - d_{\text{buff}}$, and seed start states within this buffer window. The buffer also applies to the goal states, but we take advantage of the fact that seeding more goal states doesn't significantly impact the computation time of RRT and sample between $t_{s,\text{des}}$ and $t_f = t_{f,\text{des}} + d_{\text{buff}}$.

As we sample random intermediate states, we test their connectivity to $\mathcal{T}$. In line 7, the Nearest$(\mathcal{T}, s_{\text{rand}})$ function returns the nearest node in $\mathcal{T}$ to s_{rand} in terms of Euclidean distance. The function motionValid$(s_{\text{nearest}}, s_{\text{rand}})$ from line 8 then tests the validity of this edge as shown in Fig. A.4d. This figure describes a state where a path in $\mathcal{T}$ is drawn as $\{s_s, s_0, s_1\}$ and samples are being tested for valid motion from s_1. State s_5 is rejected simply because it occurs earlier on the timeline than its connecting point, s_1, despite being designated as a destination state on the edge $\{s_1, s_5\}$. State s_4 shows motion rejected due to obstacle interference. To detect this, the algorithm samples states at discrete times along a three-dimensional polynomial trajectory between these two positions and rejects the motion if any of these states occur with in the spatio-temporal region defined by the obstacle. State s_3 is rejected due to higher order limits. This is difficult to portray in the figure, but the polynomial trajectory which describes motion between these states can be evaluated for maximum velocity and acceleration along this edge and if these values exceed predetermined limits, the edge is rejected. State s_2 shows the only valid motion for this set of samples.

Fig. A.4e shows a possible solution derived from the RRT algorithm. Note that the edges are not represented by straight lines. As we require each trajectory defined in the configuration space to be continuously differentiable up to acceleration, the solution must be described as a piece-wise continuous set of polynomials. Computing the polynomial requires finding a solution to a set of equations of the form

$$\sum_{j=0}^{m} c_j \frac{d^i(t^j)}{dt^i}\bigg|_{t=0,d_f} = l_i \quad \forall j = 0, \ldots, m/2 \tag{A.6}$$

where m is the number of coefficients. The set of coefficients $c_0, \ldots, c_m$ are derived from solving this set of equations at both $t = 0$ and $t = d_f$, where d_f is the difference in time between edge states (duration of travel across the edge). Note that this computation is relatively expensive and would have to be computed for many edges before a viable solution is found. To circumvent this, we assume all randomly sampled states have all derivatives constrained to equal zero. Trajectories between these states can be represented by a set of polynomials that travel a unit distance over a unit length of time if the trajectory is scaled by distance and time. The scaling formula described below shows how the unit polynomial coefficients, $c_{j,\text{unit}}$, can be transformed to match any pair of static waypoints

$$c_j = l_p \left(\frac{1}{d_p}\right)^j c_{j,\text{unit}}. \tag{A.7}$$

The variables l_p and d_p here describe the length and duration of the edge between two nodes, respectively. Edges that connect to the goal or start state will still require full computation as the higher-order derivatives may be non-zero, however most other edges can be computed with the simple scaling formula in (A.7).

The solution this approach provides is dynamically viable, but we arrive at a smoother trajectory if we compute the polynomials from waypoints with unconstrained higher-order terms. Richter et al. [34] suggest computing a trajectory from a string of waypoints where some, if not all, of the higher-order derivative constraints are relaxed. As long as continuity is preserved,

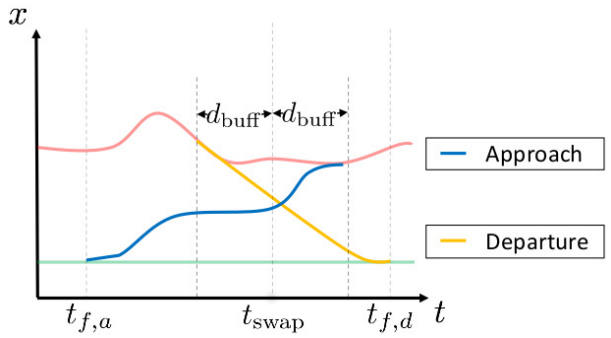
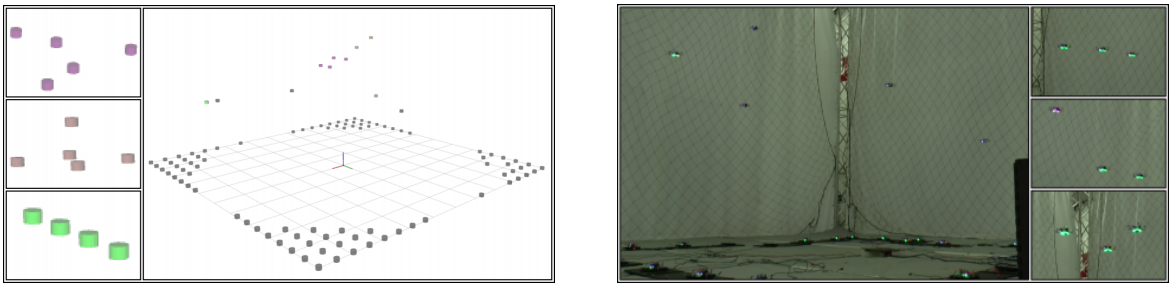


Figure A.5: Representative example of an exchange operation. Displays evolution of swap from the approach start time ($t_{f,a}$) to the departure end time ($t_{f,d}$). The approach end time and departure start time are derived through computation within a time buffer defined by $t_{\text{swap}} \pm d_{\text{buff}}$

the robot does not need to come to a full stop at each waypoint. Collision avoidance can be assured by checking for collisions along the smoothed trajectory and adding constraints to the higher-order terms of waypoints that occur immediately before and after a collision. As the RRT approach described earlier found these segments to be collision-free, we can be certain that enforcing this motion will prevent the detected collision from occurring. Resulting trajectories appear as shown in Fig. A.4f.

Although the above process is sufficient for computing the transition between behaviors, further consideration is required to handle the simultaneous motion of a two robot exchange. The scheduling algorithm establishes the instance in time in which an exchange is required. The optimal solution, which minimizes divergence from the behavior trajectory, would have both robots arriving at the exchange position as the same time. Here, we grow trees from the swap time t_{swap} towards the landing time, effectively growing the approach backwards in time. This allows us to vary the takeoff and landing time more drastically and sample the approach and departure start states at a few key points around the swap time. The diagram in Fig. A.5 depicts the timeline of a typical exchange operation.

Online Reintroduction In an effort to mitigate robot failure due to losses of reliable state estimation or outside interference, we implemented a recourse approach to plan paths that reintro-



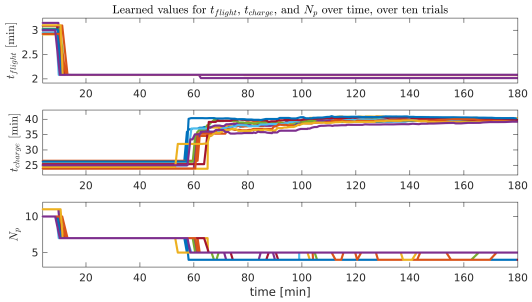
(a) Simulation (b) Hardware

Figure A.6: Images of robot behaviors in simulation and in hardware. (a) A view of a simulated 100 robot team, where ten robots are instructed to fly in various groups, in different formations. Close up images of examples of various formations are shown left. (b) The Crazyflie robot team, showing five flying robots; robots without LEDs are exchanging places in formation, so that robots with depleted batteries can return to charging stations. Additional images depicting robot formations are shown on the right.

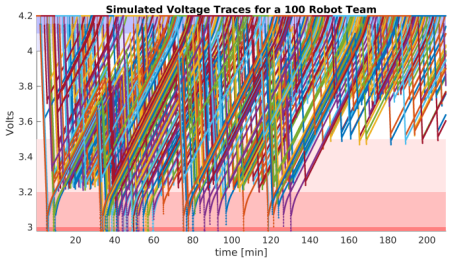
duce robots that have failed into active operation. Once a failure is detected, if there is sufficient time remaining before the subsequent exchange or departure a robot is selected from the set of charged and unscheduled robots to perform the remaining assigned trajectory. Then we perform a 'replacement' action, which involves computing a trajectory from the charging station into the behavior path such that it avoids collision with all active robots and conducting the robot along that trajectory. A 'reintroduction' action is executed when the failed robot is corrected and added to the set of active robots. Here, a path is computed from a launch position to its assigned charging station. Once a robot has landed on its charging station and recharged to full, it is available to be selected for subsequent jobs. The paths for both the replacement action and the reintroduction action are computed in the same manner as described in Sec. A.1.

A.2 Experimental Evaluation

The system proposed in this work enables a user to generate and maintain continuous motion plans over long durations for a team of robots while accounting for robot energy (battery) limitations. To evaluate our approach and system capabilities, we therefore dynamically instruct a robot team to perform behaviors online over a multi-hour timespan (a timespan exceeding 30



(a) Flight and charge time are adapted as the mission progresses. Flight time (top) converges relatively quickly as measurements are available often and in large quantity. Charge time (middle) changes quickly as initial measurements are available, then converges as the model is refined. Job count (bottom) changes as a function of flight and charge time.



(b) Voltage data collected from 100 simulated trials. Voltage levels initially exceed the safe region, but converge as parameters are learned.

Figure A.7: Parameter learning from simulated voltage data. Jobs are generated randomly for three hours.

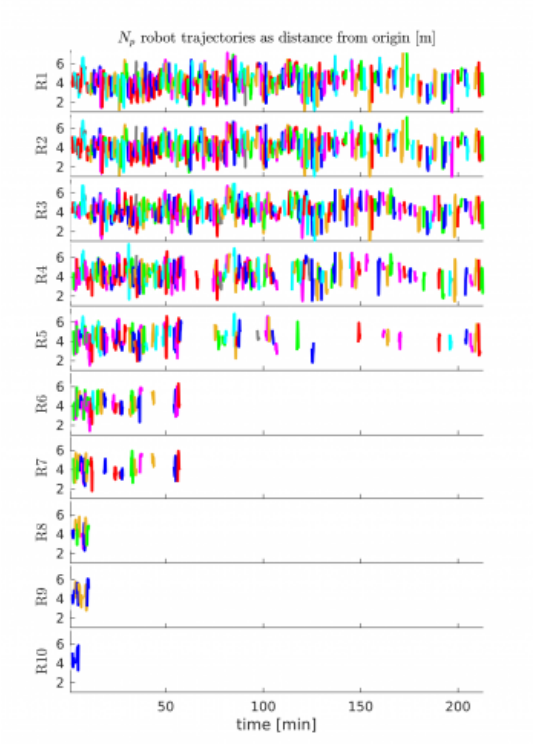


Figure A.8: Robot trajectories as a function of distance from origin. Potential active jobs listed as R1-R10. As parameters are refined, less jobs are made active and assigned robots.

times any individual vehicle energy capacity). We show that the system appropriately adapts system parameters to account for learned vehicle energy profiles, and consequently dynamically updates the number of operating robots to adapt to and maintain a continuous level of system operation.

We evaluate the proposed system via trials in both simulation and hardware, with both simulated and hardware experiments following the experimental approach described below:

- All motion plans during all trials were generated online from a centralized server and no trajectories are planned prior to testing. To simulate a human user who wishes to dynamically instruct a team of robots in an online setting, our system randomly chooses behaviors for the robot team from a set of energy-viable behaviors. This set of energy-viable behaviors is updated online based on the system's measurement of battery capacity and team-wide task assignment.
- The behaviors instruct robots to form formations and travel to goal locations or rotate in formation at a location, such as might be done when performing surveillance or coverage operations. Robots are instructed to travel at varying speeds over varying durations, form groups of varying number, transition between formation shapes, and merge or split formations. Further description of behaviors and their generation is provided in [56], and examples of robots performing behaviors in simulation and in hardware are shown in Figs. A.6a and A.6b.
- The system begins operation using estimated values of t_{flight}, the average length of time during which an individual robot has the energy capacity to remain airborne, and t_{charge}, the average length of time required to recharge a battery. Based on the voltage measurements of all robots observed during operation, t_{flight} and t_{charge} are updated online as described in Sect. A.1. In simulation, voltage measurements are generated from a nonlinear battery model, maintained for each robot in the team, following the lithium-polymer battery profile used on the hardware robots (profile shown in Fig. A.3). In hardware, voltage

measurements are measured directly at each robot.

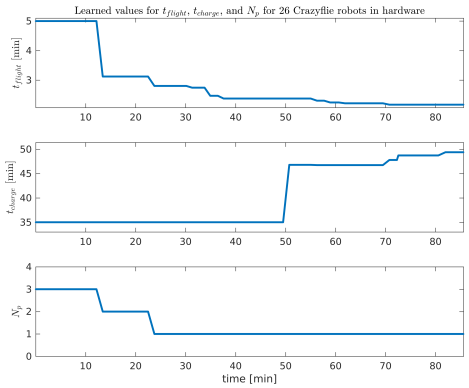
- The values for t_{flight} and t_{charge} are used to dynamically determine N_p, the number of robots required to maintain persistent operation, as defined in (A.2).

Over the course of operation in both simulation and in hardware, we show that the system only performs behaviors which fall within the energy capacity of the system, and adapts to maintain a dynamic number of persistent operations. We further show that system energy capacity limits are respected by showing that the voltage profiles of all agents in the team fall within stated limits as the system learns the true energy capacities of the vehicles.

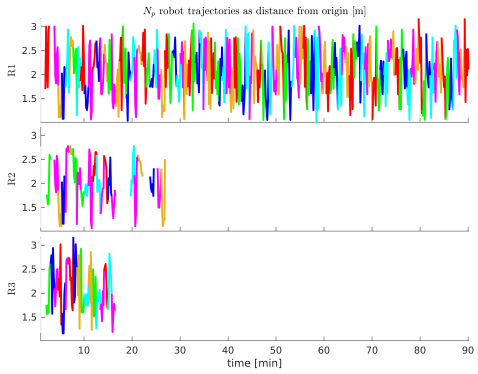
In simulation, we perform ten trials using a 100 robot team, with each trial lasting three hours, to illustrate that our method is robust, repeatedly converging to the given battery model; capable of online, real-time performance for large numbers of robots; and capable of maintaining persistent operation over timespans far exceeding those of an individual vehicle battery. We verify our simulation results through one and a half hour long hardware experiment using a team of 26 Crazyflie[2] robots within a Vicon[3] motion capture arena. Hardware results demonstrate that the described approach is viable with physical systems and verifies simulation results.

Simulation Simulated missions were run in 10 separate trials with each trial operating 100 robots for three hours. The evolution of t_{flight}, t_{charge}, and N_p is shown in Fig. A.7a. As individual flights end quickly and are performed often, the flight time sees significant change early. Also, the flight time will converge quickly from high values to more conservative values as a direct measurement of flight duration is available once the measured voltage crosses the lower voltage bound. Charge time, however, increases quickly once measurements are available, but takes a while to converge as the value increases by static increments each instance voltage fails to reach the upper threshold during charging. The job count changes as a function of flight and charge time, so we see major decreases at instances where flight or charge time see significant increase.

[2]https://www.bitcraze.io/crazyflie-2/
[3]http://www.vicon.com/



(a) Flight time (top), charge time (middle), and number of persistent jobs (bottom) adaptation mirrors simulation results.



(b) Trajectories of the robots over time. Number of persistent jobs supported drops from 3 to 2 to 1.

Figure A.9: Parameter learning from live trials. Jobs are generated randomly for over one hour.

The simulated voltage profiles in Fig. A.7b show how voltages initially cross into the unsafe region, but steadily confine themselves to the safe region as the parameters are updated. This effect will occur some time after the parameters are updated relative to how far ahead plans have been generated once parameters are updated. In the current approach, previously processed plans are not re-generated as the parameters are updated as this would be too computationally demanding for our system.

Fig. A.8 shows how jobs are assigned to robots over the course of a trial. For the sake of expressing activity, each job (R1-R10) is plotted as a one-dimension polynomial indicating distance from the origin relative to time. Robot assignments are differentiated by color and show how each job is assigned to many robots over the course of the mission. Initially many jobs are active when the parameters are set to their ambitious initial values. As the parameters grow more conservative, the number of active jobs dwindles until we only see 4-5 active robots at a time.

Hardware Similarly, testing was done in hardware with Crazyflie robots operating within our Vicon motion capture arena. The robots used are outfitted with reflective beads to assist Vicon in providing pose information and an inductive charging board to allow for passive charging, as seen in Fig. 3.1. With the added weight, the robots are capable flight times of roughly two

minutes and charge times of roughly 40 minutes. Fig. A.9a shows the minimum voltage relative to the minimum allowed voltage for the Lithium Polymer (LiPo) batteries with which the robots are equipped. Fig. A.9b shows how a team of 26 Crazyflies was flown for one and a half hours, and that the system correctly adapted parameters and plans to maintain persistent operation. In particular, we note that the evolution of the live trial mirrors what was seen in simulation and, as such, the proposed system is viable for operation for extended durations on hardware.

A.3 Conclusion

This work presented a comprehensive system capable of deploying a large team of robots for long-duration, persistent operation. Missions of this kind are particularly suited towards inspection and exploration and require fast, responsive generation of energy-feasible plans to properly express the user input behavior. The results generated in this work show that the system is capable of extended duration operations exceeding 1000 power cycles of constant motion. A subset of the team was shown persistently operating through robot exchanges, with larger subsets of the team operating at less frequent intervals. We showed extensive results in simulation that show the reliability of the proposed system given uncertainty in the energy capacity of individual platforms.

Bibliography

[1] Howie Choset and Philippe Pignon. Coverage path planning: The boustrophedon cellular decomposition. In Alexander Zelinsky, editor, *Field and Service Robotics*, pages 203–209, London, 1998. Springer London. 1

[2] Gilbert Laporte, Michel Gendreau, Jean-Yves Potvin, and Frédéric Semet. Classical and modern heuristics for the vehicle routing problem. *International Transactions in Operational Research*, 7(4):285 – 300, 2000. 1, 2.3.1

[3] Roberto Baldacci, Aristide Mingozzi, and Roberto Roberti. Recent exact algorithms for solving the vehicle routing problem under capacity and time window constraints. *European Journal of Operational Research*, 218(1):1 – 6, 2012. 1

[4] Aldy Gunawan, Hoong Chuin Lau, and Pieter Vansteenwegen. Orienteering problem: A survey of recent variants, solution approaches and applications. *European Journal of Operational Research*, 255(2):315 – 332, 2016. 1, 5.5.4

[5] David L. Applegate, Robert E. Bixby, Vašek Chvatál, and William J. Cook. *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, 2006. 1

[6] Paolo Toth and Daniele Vigo, editors. *The Vehicle Routing Problem*. Society for Industrial and Applied Mathematics, USA, 2001. 1, 2.3.1

[7] Jan Karel Lenstra and A. H. G. Rinnooy Kan. Complexity of vehicle routing and scheduling problems. *Networks*, 11(2):221–227, 1981. 1

[8] Paolo Toth and Daniele Vigo. The granular tabu search and its application to the vehicle-routing problem. *INFORMS Journal on Computing*, 15(4):333–346, 2003. 1, 2.3.1

[9] Mazin Abed Mohammed, Mohd Sharifuddin Ahmad, and Salama A. Mostafa. Using genetic algorithm in implementing capacitated vehicle routing problem. In *International Conference on Computer Information Science*, volume 1, pages 257–262, June 2012. 1, 2.3.1

[10] Simon T O'Callaghan and Fabio T Ramos. Gaussian process occupancy maps. *The International Journal of Robotics Research*, 31(1):42–62, 2012. 1, 2.2b, 2.1

[11] Chaoqun Wang, Teng Li, Max Q.-H. Meng, and Clarence De Silva. Efficient mobile robot exploration with gaussian markov random fields in 3d environments. In *Proc. of the IEEE Intl. Conf. on Robot. Auto.*, pages 5015–5021, May 2018. 1

[12] Shaojie Shen, Nathan Michael, and Vijay Kumar. Stochastic differential equation-based exploration algorithm for autonomous indoor 3d exploration with a micro-aerial vehicle. *The International Journal of Robotics Research*, 31(12):1431–1444, 2012. 1, 2.1.2

[13] Maani Ghaffari Jadidi, Jaime Valls Miro, and Gamini Dissanayake. Mutual information-based exploration on continuous occupancy maps. In *Proc. of the IEEE/RSJ Int. Conf. on Intelli. Robots and Systs.*, pages 6086–6092, 2015. 1, 2.1.2

[14] Benjamin Charrow, Gregory Kahn, Sachin Patil, Sikang Liu, Kenneth Y. Goldberg, Pieter Abbeel, Nathan Michael, and R. Vijay Kumar. Information-theoretic planning with trajectory optimization for dense 3d mapping. In *Robotics: Science and Systems*, 2015. 1

[15] Wennie Tabib, Micah Corah, Nathan Michael, and Red Whittaker. Computationally efficient information-theoretic exploration of pits and caves. In *Proc. of the IEEE/RSJ Int. Conf. on Intelli. Robots and Systs.*, pages 3722–3727, Oct 2016. 1, 2.1.2

[16] Derek Mitchell and Nathan Michael. Persistent multi-robot mapping in an uncertain environment. In *Proc. of the IEEE Intl. Conf. on Robot. Auto.*, pages 4552–4558, May 2019. 1.2, 5.5.2

[17] Sebastian Thrun. *Robotic Mapping: A Survey*, page 1–35. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003. 2.1

[18] Shobhit Srivastava and Nathan Michael. Approximate continuous belief distributions for precise autonomous inspection. In *IEEE Int. Sym. on Safety Security and Rescue Robotics*, pages 74–80, Oct 2016. 2.2c, 2.1

[19] Sebastian Thrun. Learning occupancy grid maps with forward sensor models. *Autonomous Robots*, 15(2):111–127, Sep 2003. 2.1

[20] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005. 2.1

[21] Edward Snelson and Zoubin Ghahramani. Sparse gaussian processes using pseudo-inputs. In Y. Weiss, B. Schölkopf, and J. C. Platt, editors, *Advances in Neural Information Processing Systems 18*, pages 1257–1264. MIT Press, 2006. 2.1

[22] Dirk Hähnel, Dirk Schulz, and Wolfram Burgard. Mobile robot mapping in populated environments. *Advanced Robotics*, 17(7):579–597, 2003. 2.1.1

[23] Dominik Nuss, Stephan Reuter, Markus Thom, Ting Yuan, Gunther Krehl, Michael Maile, Axel Gern, and Klaus Dietmayer. A random finite set approach for dynamic occupancy grid maps with real-time application. *The International Journal of Robotics Research*, 37 (8):841–866, 2018. 2.1.1

[24] Brian Yamauchi. Frontier-based exploration using multiple robots. In *Proceedings of the Second International Conference on Autonomous Agents*, AGENTS '98, page 47–53, New York, NY, USA, 1998. Association for Computing Machinery. 2.1.2

[25] *Entropy, Relative Entropy and Mutual Information*, chapter 2, pages 12–49. John Wiley & Sons, Ltd, 2001. 2.1.2

[26] Brian J. Julian, Sertac Karaman, and Daniela Rus. On mutual information-based control of range sensing robots for mapping applications. *The International Journal of Robotics Research*, 33(10):1375–1392, 2014. 2.1.2

[27] Ben Charrow, Sikang Liu, Vijay Kumar, and Nathan Michael. Information-theoretic mapping using cauchy-schwarz quadratic mutual information. In *Proc. of the IEEE Intl. Conf. on Robot. Auto.*, pages 4791–4798, May 2015. 2.1.2, 3.2, 5.2.3

[28] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959. 2.2

[29] Peter Hart, Nils Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2): 100–107, 1968. 2.2

[30] Anthony Stentz. Optimal and efficient path planning for unknown and dynamic environments. *International Journal of Robotics and Automation*, 10:89–100, 1993. 2.2

[31] Sven Koenig, Maxim Likhachev, and David Furcy. Lifelong planning a*. *Artificial Intelligence*, 155(1):93 – 146, 2004. 2.2

[32] Glenn Wagner and Howie Choset. Subdimensional expansion for multirobot path planning. *Artificial Intelligence*, 219:1–24, 2015. 2.2

[33] Daniel Mellinger and Vijay Kumar. Minimum snap trajectory generation and control for quadrotors. In *Proc. of the IEEE Intl. Conf. on Robot. Auto.*, pages 2520–2525, Shanghai, China, May 2011. 2.2

[34] Charles Richter, Adam Bry, and Nicholas Roy. *Polynomial Trajectory Planning for Aggressive Quadrotor Flight in Dense Indoor Environments*, pages 649–666. Springer International Publishing, Cham, 2016. 2.2, A.1

[35] Sikang Liu, Nikolay Atanasov, Kartik Mohta, and Vijay Kumar. Search-based motion planning for quadrotors using linear quadratic minimum time control. In *Proc. of the IEEE/RSJ Int. Conf. on Intelli. Robots and Systs.*, pages 2872–2879, Sep. 2017. 2.2

[36] Arjav Desai, Matthew Collins, and Nathan Michael. Efficient kinodynamic multi-robot replanning in known workspaces. In *Proc. of the IEEE Intl. Conf. on Robot. Auto.*, pages 1021–1027, 2019. 2.2

[37] Jonghoe Kim, Byung Duk Song, and James R. Morrison. On the scheduling of systems of uavs and fuel service stations for long-term mission fulfillment. *J. Intell. Robotics Syst.*, 70 (1-4):347–359, April 2013. 2.3

[38] Derek Mitchell, Ellen A. Cappo, and Nathan Michael. Persistent robot formation flight via online substitution. In *Proc. of the IEEE/RSJ Int. Conf. on Intelli. Robots and Systs.*, pages 4810–4815, Daejeon, Korea, October 2016. IEEE. 2.3, 2.3b, A

[39] Gennaro Notomista, Sebastian F. Ruf, and Magnus Egerstedt. Persistification of robotic tasks using control barrier functions. *IEEE Robotics and Automation Letters*, 3(2):758–763, April 2018. 2.3

[40] Paolo Toth and Daniele Vigo. *The Vehicle Routing Problem*, chapter 2. Branch-And-Bound Algorithms for the Capacitated VRP, pages 29–51. 2002. 2.3.1

[41] Jean François Cordeau, M. Gendreau, Gilbert Laporte, Jean Yves Potvin, and Frédéric Semet. A guide to vehicle routing heuristics. *Journal of the Operational Research Society*, 53(5):512–522, May 2002. 2.3.1

[42] Jean-Yves Potvin and Jean-Marc Rousseau. An exchange heuristic for routeing problems with time windows. *The Journal of the Operational Research Society*, 46(12):1433–1446, 1995. 2.3.1

[43] Derek Mitchell, Nilanjan Chakraborty, Katia Sycara, and Nathan Michael. Multi-robot persistent coverage with stochastic task costs. In *Proc. of the IEEE/RSJ Int. Conf. on Intelli. Robots and Systs.*, pages 3401–3406, Hamburg, Germany, September 2015. 2.3.1

[44] Saravanan Venkatachalam, Kaarthik Sundar, and Sivakumar Rathinam. A two-stage approach for routing multiple unmanned aerial vehicles with stochastic fuel consumption. *Sensors*, 18(11), 2018. 2.3.1

[45] Ethan Stump and Nathan Michael. Multi-robot persistent surveillance planning as a vehicle routing problem. In *IEEE Int. Conf. on Autom. Sci. Eng.*, pages 569–575, Aug 2011. 2.3.1, 3.2

[46] Derek Mitchell, Micah Corah, Nilanjan Chakraborty, Katia Sycara, and Nathan Michael. Multi-robot long-term persistent coverage with fuel constrained robots. In *Proc. of the IEEE Intl. Conf. on Robot. Auto.*, pages 1093–1099, Seattle, Washington, May 2015. 2.3.1, 2.3.1

[47] Kaarthik Sundar and Sivakumar Rathinam. Algorithms for routing an unmanned aerial vehicle in the presence of refueling depots. *IEEE Trans. Autom. Sci. Eng.*, 11(1):287–294, January 2014. 2.3.1

[48] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. MIT Press, 2008. 3.2

[49] Brian Kallehauge, Jesper Larsen, and Oli B.G. Madsen. Lagrangian duality applied to the vehicle routing problem with time windows. *Computers & Operations Research*, 33(5): 1464 – 1487, 2006. 3.2, 3.2, 3.2

[50] Dominique Feillet, Pierre Dejax, Michel Gendreau, and Cyrille Gueguen. An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. *Networks*, 44(3):216–229, 2004. 3.2

[51] Daniel Meyer-Delius, Maximilian Beinhofer, and Wolfram Burgard. Occupancy grid models for robot mapping in changing environments. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, AAAI'12, pages 2024–2030. AAAI Press, 2012. 4.2.1

[52] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, Feb 1989. 4.2.1

[53] Gianluigi Mongillo and Sophie Deneve. Online learning with hidden markov models. *Neural Comput.*, 20(7):1706–1716, July 2008. 4.2.1

[54] J. D. Kalbfleisch and J. F. Lawless. The analysis of panel data under a markov assumption. *Journal of the American Statistical Association*, 80(392):863–871, 1985. 4.2.2

[55] John Amanatides and Andrew Woo. A Fast Voxel Traversal Algorithm for Ray Tracing. In *EG 1987-Technical Papers*. Eurographics Association, 1987. 5.2.1

[56] Ellen A. Cappo, Arjav Desai, and Nathan Michael. Robust coordinated aerial deployments for theatrical applications given online user interaction via behavior composition. *Distributed Auton. Robot. Syst.*, November 2016. A.1, A.1, A.2

[57] Arjav Desai, Ellen A. Cappo, and Nathan Michael. Dynamically feasible and safe shape transitions for teams of aerial robots. In *Proc. of the IEEE/RSJ Int. Conf. on Intelli. Robots and Systs.*, pages 5489–5494, Daejeon, Korea, October 2016. A.1

[58] Harold W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955. A.1

[59] Jonathan D. Gammell, Siddhartha S. Srinivasa, and Timothy D. Barfoot. Batch informed trees (bit*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs. In *Proc. of the IEEE Intl. Conf. on Robot. Auto.*, pages 3067–3074, May 2015. A.1

[60] Ioan A. Şucan, Mark Moll, and Lydia E. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, December 2012. `http://ompl.kavrakilab.org`. 1

[61] Steven M. LaValle and Jr. James J. Kuffner. Randomized kinodynamic planning. *The International Journal of Robotics Research*, 20(5):378–400, 2001. A.1