ABSTRACT

Title of Dissertation: Efficient Environment Sensing and Learning for Mobile Robots

Varun Suryan
Doctor of Philosophy, 2022

Dissertation Directed by: Professor Pratap Tokekar
Department of Computer Science

Data-driven learning is becoming an integral part of many robotic systems. Robots can be used as mobile sensors to learn about the environment in which they operate. Robots can also seek to learn essential skills, such as navigation, within the environment. A critical challenge in both types of learning is sample efficiency. Acquiring samples with physical robots can be prohibitively time-consuming. As a result, when applying learning techniques in robotics that require physical interaction with the environment, minimizing the number of such interactions becomes a key. The key question we seek to answer is: How do we make robots learn efficiently with a minimal amount of physical interaction? We approach this question along two fronts: extrinsic learning and intrinsic learning. In extrinsic learning, we want the robot to learn about the external environment in which it is operating. In intrinsic learning, our focus is on the robot to learn a skill using reinforcement learning (RL) such as navigating in an environment. In this dissertation, we develop algorithms that carefully plan where the robots obtain samples in order to efficiently perform

intrinsic and extrinsic learning. In particular, we exploit the structural properties of Gaussian Process (GP) regression to design efficient sampling algorithms.

We study two types of problems under extrinsic learning. We start with the problem of learning a spatially varying field modeled by a GP efficiently. Our goal is to ensure that the GP posterior variance, which is also the mean square error between the learned and actual fields, is below a predefined value. By exploiting the underlying properties of GP, we present a series of constant-factor approximation algorithms for minimizing the number of stationary sensors to place, minimizing the total time taken by a single robot, and minimizing the time taken by a team of robots to learn the field. Here, we assume that the GP hyperparameters are known. We then study a variant where our goal is to identify the hotspot in an environment. Here we do not assume that hyperparameters are unknown. For this problem, we present Upper Confidence Bound (UCB) and Monte Carlo Tree Search (MCTS) based algorithms for a single robot and later extend them to decentralized multi-robot teams. We also validate their performance on real-world datasets.

For intrinsic learning, our aim is to reduce the number of physical interactions by leveraging simulations often known as Multi-Fidelity Reinforcement Learning (MFRL). In the MFRL framework, an agent uses multiple simulators of the real environment to perform actions. We present two MFRL framework versions, model-based and model-free, that leverage GPs to learn the optimal policy in a real-world environment. By incorporating GPs in the MFRL framework, we empirically observe a significant reduction in the number of samples for model-based and model-free learning.

Efficient Environment Sensing and Learning for Mobile Robots

by

Varun Suryan

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2022

Advisory Committee:
Dr. Pratap Tokekar, Chair/Advisor
Dr. Dinesh Manocha
Dr. Furong Huang
Dr. Jordan Boyd-Graber
Dr. Derek A. Paley

Dedication

To my parents.

Acknowledgments

I am grateful to my advisor Pratap Tokekar and all the lab members. Research meetings with Dr. Tokekar have always been helpful and fun in preparing this dissertation.

Thanks to my parents, brother, family members, and friends for always supporting me throughout the graduate school.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

CDOM	Chromophoric Dissolved Organic Material
DCT	DiskCoverTour
DL	Deep Learning
GPs	Gaussian Processes
IPP	Informative Path Planning
MAB	Multi-Armed Bandits
MCTS	Monte Carlo Tree Search
MDP	Markov Decision Processes
MFRL	Multi-Fidelity Reinforcement Learning
MI	Mutual Information
ML	Machine Learning
MRS	Multi-Robot Systems
MSE	Mean Square Error
OM	Organic Matter
RL	Reinforcement Learning
TSPN	Traveling salesperson with neighborhoods
UAV	Unmanned Aerial Vehicles
UCB	Upper Confidence Bound
UCT	Upper Confidence Tree
VI	Value Iteration

Chapter 1: Introduction

1.1 Learning in Robotics

There has been a huge interest recently in incorporating learning in robotics. This is not surprising. Given the advancements in the Machine Learning (ML) community [2–6], there has been a push to leverage powerful learning-based methods to provide robots more autonomy [3, 7–10]. The recent progress in machine learning techniques has been spurred, in part, by access to large datasets [11, 12]. However, when it comes to applying these techniques in robotics, acquiring this dataset itself is a challenge since it requires physical interaction. Dealing with robots presents unique challenges. For example, while the state-of-the-art ML techniques can achieve super-human performance on certain tasks, these algorithms often require billions of training samples by interacting with the environment [4, 13–15]. Acquiring these many samples with physical robots may not be feasible [16]. While it may be possible in specific cases, such as robot manipulators [17], mobile robots in general present a different proposition. They are costly to operate, have significant energy requirements, and suffer from hardware degradation that can result in failure if made to interact endlessly with the environment. Maintaining robots for long periods of time can be labor and cost-intensive. As a result, when applying

ML techniques in robotics that require physical interaction with the environment, minimizing the number of such interactions becomes key.

While there is work on reducing the sample complexity of ML algorithms [18–21], reducing just the number of samples may not be sufficient for physical agents. This is because obtaining a sample may require the mobile robot to travel to a new location [14, 22]. As such, we need algorithms that can directly take into account the cost to obtain a sample by a mobile robot. When there is a team of robots, these algorithms must factor in the communication capabilities between the robots.

The key contribution we make in this dissertation is how to make robots learn as efficiently as possible with a minimal amount of physical interaction. Current deployment of robots requires lots of guesswork in terms of how much data one should collect, how many robots to deploy, and where are the interesting regions in the environment from where one should collect the data? The algorithms presented in this dissertation provide a systematic way to answer these questions for a practitioner. For example, an environmental scientist may be interested in mapping the concentration of certain algae in a water body. One way could be to go and collect the data in a lawn-mower pattern. However, figuring out the resolution of the lawn-mower pattern is not trivial. Further, this can be inefficient in practice and does not come with any theoretical guarantees on the prediction accuracy. The algorithms proposed in this dissertation answer these questions about how much data should they collect and from which locations so that they can build an accurate map in such situations.

We measure the cost of collecting data as the total operating time for the

physical robot which consists of the travel time and measurement time. The time to obtain an individual measurement may be negligible (e.g., if the sensor is a camera) or may be substantial (e.g., if the sensor is a soil probe that needs to be physically inserted in the ground to get a sample). We devise algorithms to reduce the cost of data collection and eliminate the guesswork in robot deployment by answering two broad sets of questions: *where to collect the data from* and *when to collect the data*? The former is motivated by *extrinsic learning* scenarios where the robot is learning about the environment that it is operating in. In these scenarios, it may not be necessary for the robot to collect the data everywhere. We devise algorithms that carefully choose measurement locations to reduce the cost of data collection. Intrinsic learning deals with the problem of robot learning by itself a skill or solving a task. We develop algorithms that provably reduce the number of physical interactions needed by the robot to learn by leveraging simulators (where the cost of data collection is low) and carefully choosing when to collect data with physical robots.

1.1.1 Extrinsic Learning

In extrinsic learning, the goal for the robot is to learn about the external environment where it is operating. This problem is often known as Informative Path Planning (IPP) in the robotics literature [23–27]. In IPP, the robots are generally equipped with sensors and the objective for the robots is to learn the environment as quickly as possible. It has significant applications. For example,

robots can be used to learn the concentration of nitrogen present in the soil within a farm [28]. Knowing the content of various nutrients in the soil within a farm can help the farmers to improve the yield and reduce the application of fertilizers. Robots are increasingly used to learn about the environments, such as a chemical spill in a water body (Figure 1.1(a)) which can have a significant impact on marine life. Learning the spatial variation of rock minerals can help in efficient mining strategies [29]. Such applications can be abstracted as IPP where the robot equipped with appropriate sensors is tasked with learning the spatially and/or temporally varying representation of the environment.

There are many factors to consider when deploying robots to efficiently learn the external environment. Usually, a trade-off must be made between the number of sensing resources (e.g., number of deployed robots, energy consumption, mission time) and the quality of data collected. The robots can be deployed to act as stationary or mobile sensors depending on the application (Figure 1.1(b)). Deploying robots to function as mobile sensors is especially challenging because of the need for path planning. While deploying mobile robotic sensors, one needs to plan the most informative resource-constrained observation paths to minimize the uncertainty in modeling and tracking the spatial phenomena.

While there has been significant empirical work in this area, most results lack theoretical soundness. By exploiting the properties of Gaussian Process (GP) regression [30], we present several constant-factor polynomial-time approximation algorithms to learn a given spatial field. We validate these algorithms in various real-world scenarios as well. We study two related problems. We start by seeking

(a) An unmanned aerial vehicle (UAV) flying over a lake to find the chemical spill hotspots [31].

(b) A single quadcopter can fly over a farm and measure the height of the crop using a LIDAR sensor.

Figure 1.1: Robots used as mobile sensors to efficiently learn an environment.

to minimize the time required for mobile robots to learn a given spatial field at each location (Chapter 2). We assume that the underlying GP hyperparameters are known. In some applications, it is not crucial to learn the entire spatial field but to rather learn about the interesting points in the environment. Motivated by this, we seek to minimize the time in finding hotspots in a spatial field (Chapter 3). The hotspots are defined as the locations in the spatial field where it attains their maximum values. There can be multiple hotspot locations. In this case, we plan adaptively, *i.e.,* not assuming that the GP hyperparameters are known *a priori*. In both cases, time includes the time required by the robot to travel as well as the time required to obtain measurements.

1.1.2 Intrinsic Learning

As mentioned previously, in intrinsic learning, we focus on the robot to learn a skill such as navigating autonomously while avoiding collisions with obstacles. This falls under the umbrella of Reinforcement Learning (RL) [32].

RL can be traced back to the early days of machine learning and work in

statistics, psychology, neuroscience, and computer science [33]. An agent is linked to its environment through perception and action in a standard reinforcement-learning model. In robotics applications, the agent is usually the robot. The agent's goal is to find a strategy often termed as policy represented by π that maximizes some long-run measure of reward by mapping states to actions. A typical RL problem is mathematically formulated as a Markov Decision Process (MDP) [34]. In the last ten to fifteen years, it has attracted rapidly increasing interest in the machine learning and artificial intelligence communities due to the gains in computational resources and rise of deep learning [35]. Deep reinforcement learning has achieved superhuman performance on several benchmark tasks [36]. The strength of these approaches stems from the convergence of a well-established and powerful area of deep learning with the unique nature of RL methods.

There has also been a significant interest for using RL in robotics [3,14,37–39] recently. Reinforcement learning provides robotics with a structure and collection of methods for creating complex and difficult-to-engineer behaviors. Robotic problems, on the other hand, provide motivation, effect, and validation for advancements in reinforcement learning. The interdisciplinary relationship between RL and robotics has significant promise. A major limitation of using RL for planning with robots is the need to obtain a large number of training samples. Obtaining a large number of real-world training samples can be expensive and potentially dangerous. In particular, obtaining exploratory samples–which are crucial to learning optimal policies–may require the robot to collide or fail, which is undesirable. Motivated by these scenarios, we focus on minimizing the number of real-world samples required

for learning optimal policies.

One way to reduce the number of real-world samples is to leverage simulators [40]. Collecting learning samples in a robot simulator is often inexpensive and fast. One can use a simulator to learn an initial policy which is then transferred to the real world — a technique usually referred to as sim2real [40]. This lets the robot avoid learning from scratch in the real world and hence, reduces the number of physical interactions required. However, this comes with a trade-off. While collecting learning samples in simulators is inexpensive, they often fail to capture the real-world environments perfectly, a phenomenon called the reality gap. While simulators with increasing fidelity with respect to the real world are being developed, one would expect there to always remain some reality gap.

Our idea is to leverage not just one simulator, but rather a chain of simulators of increasing fidelities (with real hardware as the highest fidelity at the top of the chain). The slowest fidelity simulators can be used at the start to learn quickly, which can then be transferred to the next fidelity simulator. Furthermore, unlike previous work, the transfer is bidirectional. After transferring, if the robot realizes that the policy in its current state has high uncertainty, the robot drops to a lower fidelity simulator to collect additional samples. The key is to show when to switch simulators and how to transfer data across environments. This technique is often known as Multi-Fidelity Reinforcement Learning (MFRL) [41]. In the MFRL framework, an agent uses multiple simulators of the real environment to perform actions. These simulators have varying fidelity levels with the real environment. The lower fidelity simulators can be used at the start to learn quickly and subsequently the policy can

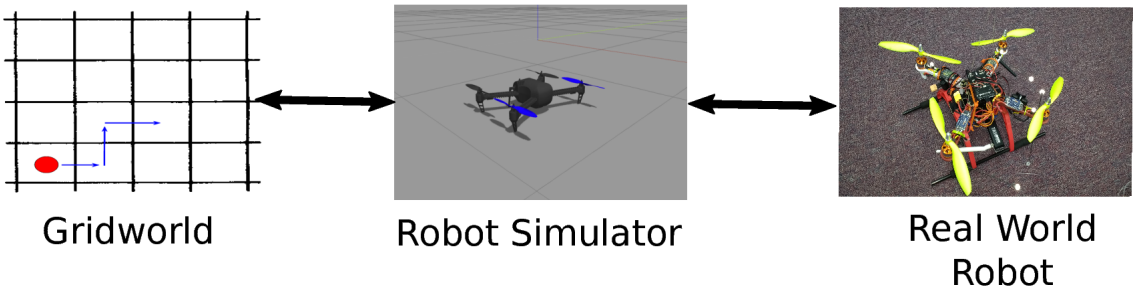


Figure 1.2: MFRL framework: The first simulator captures only grid-world movements of a point robot while the second simulator has more fidelity modeling the physics as well. Control can switch back and forth between simulators and real environment which is the third simulator in the chain.

be transferred to the next fidelity simulator. With increasing fidelity in a simulator chain, the number of samples used in successively higher simulators can be reduced. The simulators have increasing levels of fidelity with respect to the real environment. For example, the first simulator can be a simple simulator that models only the robot kinematics, the second one can model the dynamics as well as kinematics, and the highest fidelity simulator can be the real world (Figure 1.2). We provide more details and results on the MFRL framework in Chapter 5.

The common objective in both extrinsic as well as intrinsic learning settings is to minimize the cost of obtaining samples by the robots. Unlike traditional notion of sample complexity (that measures only the number of samples used for learning), we focus on the cost of sampling with a robot which includes time to physical travel to a measurement location and to actually obtain the measurement. We use structural properties of GPs to reduce this cost.

1.2 Contributions and Roadmap

This dissertation proposes several algorithms for efficient extrinsic learning as well as intrinsic learning. Next, we provide a brief description of the contributions made in this dissertation starting with extrinsic learning where the goal is to minimize the time required to learn a spatially varying entity.

1.2.1 Extrinsic Learning

We study two versions here – learning the entire environment and finding the hotspot by using GP regression. In the first version, we use the squared-exponential isotropic kernel [30] and assume that the kernel hyperparameters are known. In the second version, we use the anisotropic squared-exponential kernel and propose algorithms that do not assume that the hyperparameters are known. We focus on learning a stationary field in both cases.

Learning the Entire Environment. In many real-world robotics applications, it is crucial to learn about the entire environment efficiently. For example, in mapping the radiological activity inside a nuclear power plant, it is important to predict the nuclear radiation level at each point in the environment accurately. This can help in closely monitoring for any potential leakages or breakdowns [42]. In another application, stationary sensors are often deployed to monitor temperature in a geographical region [43]. Normally sensors are expensive. Hence, in these scenarios it is important to monitor the temperature using as few sensors as we can. Further,

a farmer may be interested in learning about the nutrient concentration at various locations in a farm. Learning about the nutrient concentration in the farm can help in better resource planning [44]. These are just a few examples but there are many others where learning of the entire environment is important [24, 45–47].

For learning the entire environment, we study three versions of the problem in Chapter 2. Our goal is to ensure that the GP posterior variance, which is also the mean square error between the learned and actual fields, is below a predefined value. In the placement version, the objective is to minimize the number of measurement locations while ensuring that the posterior variance is below a predefined threshold. In the mobile robot version, we seek to minimize the total time taken by the robot to visit and collect measurements from the measurement locations using a single robot. We further extend it to multi-robot systems (MRS). An MRS can exhibit better system reliability, flexibility, and versatility. Robots with diverse abilities can be combined together to deal with complex task [48]. In the multi-robot version, the objective is to minimize the time required by the last robot to return to a common starting location called a depot. By exploiting the properties of GP regression, we present constant-factor approximation algorithms. In summary, our main contributions here include:

- introducing stationary sensor placement and mobile sensor algorithms for ensuring that the prediction error at each location in the environment is below a predefined threshold,
- providing polynomial-time constant-factor approximation guarantees on their

performance, and

- showing their performance on a real-world dataset comprising of organic matter concentrations at various locations within a farm.

We show that it is possible to learn a given spatial field accurately with high confidence without planning adaptively.

The single robot and the stationary placement algorithms were presented at the 13^{th} International Workshop on the Algorithmic Foundations of Robotics [49]. The multi-robot version appeared in the IEEE Transactions on Robotics in 2020 [28].

Finding the Hotspot. In many real-world applications, finding the hotspot is more important than learning the entire environment accurately. The hotspot here is defined as the location where the underlying field achieves its maximum value. For example, consider the case of a chemical spill in a water body. This can have a significant impact on the marine life. We might be interested in finding the hotspot here to locate the source of the spill [31]. In another application, robots can be used as scouts to potentially identify the regions of high values. After the scouting mission, the robots can report the hotspot locations for further data collection from those regions [50]. For hotspot identification, our goal is to identify the hotspot in the environment using a mobile sensor(s). We again use GPs to model the underlying spatial field. In many practical applications, the GP hyperparameters are not known and must be optimized during the process. Optimizing GP hyperparameters is computationally prohibitive. We propose two algorithms that finds the hotspot with or without the need for hyperparameter optimization. In the latter case, we

provide a computationally inexpensive way of adapting the hyperparameters. The first algorithm follows an Upper Confidence Bound (UCB) [51] style exploration while the second algorithm is based on Monte Carlo Tree Search (MCTS) [52]. We compare their empirical performance on carefully designed synthetic spatial fields of varying complexities as well as two real-world datasets. The first dataset is the concentration of organic matter at various locations in a farm. The second dataset is the Chromophoric Dissolved Organic Material (CDOM) concentration collected from a subregion inside the Gulf of Mexico. In summary, our main contributions include:

- introducing two single and multi-robot adaptive path planning algorithms to minimize the terminal regret given that the robot is allowed to operate for a predefined time budget of T,
- showing their performance on two real-world datasets comprising of OM concentrations at various locations within a farm and Chlorophyll concentration in a Pacific Ocean subregion, and
- introducing a computationally efficient way of adapting the GP hyperparameters.

The terminal regret is defined as the difference between the actual maximum field value $f(x^*)$ and the actual field value $f(\hat{x})$ at the location $\hat{x}$ of maxima reported by the robot. We present a more thorough discussion on the terminal regret in later chapters.

The single robot and the multi-robot versions are presented in Chapters 2 and

3 respectively.

1.2.2 Intrinsic Learning

Often, in the task of learning, many times we may have access to robot simulators so we can ease the burden of learning only with physical robots to use simulators. For example, in training a robot to do certain maneuvers, we may use its dynamics simulator and collect some data in that simulator. This makes our foundation for intrinsic learning where we can leverage simulators to collect the data. In Chapter 5, we show how to combine function approximators with the MFRL framework. We leverage the GP regression as a function approximator to speed up learning in the MFRL framework. GPs can predict the learned function value for any query point, and not just for a discretized state-action pair. Furthermore, GPs can exploit the correlation between nearby state-action values by an appropriate choice of a kernel [53]. GPs have been extensively used to obtain optimal policies in simulation-aided reinforcement learning [16]. We take this further by using GPs in the MFRL setting for both versions, model-based and model-free [32]. In summary, our contributions for the single robot case include introducing:

1. a model-based MFRL algorithm, GP-VI-MFRL, which estimates the transition function and subsequently calculates the optimal policy using Value Iteration (VI); and
2. a model-free MFRL algorithm, GPQ-MFRL, which directly estimates the optimal Q-values and subsequently the optimal policy.

We verify the performance of the algorithms presented through simulations as well as experiments with a ground robot. Our empirical evaluation shows that the GP-based MFRL algorithms learn the optimal policy faster than the original MFRL algorithm using even fewer real-world samples.

The model-free version appeared in the AAAI 2018 Fall Symposium on Reasoning and Learning in Real-World Systems for Long-Term Autonomy [54] and the model-free and model-based algorithms appeared in a special issue of Machine Learning in Robotics of the IEEE Robotics and Automation Magazine [53].

Chapter 2: Planning to Efficiently Learn an Environment

We start with the problem of efficiently learning a spatially-varying field. This is motivated by applications such as environmental monitoring (Figure 2.1) where robots are used to learn a model of an environmental phenomenon such as temperature.

The underlying field is typically modeled as a Gaussian Process [24,55,56]. The question we seek to answer in this chapter is where should be obtain measurements from so as to learn the entire field accurately using few measurements. In this chapter, we will study three versions of the problem: placing stationary sensors, using a single mobile sensor (robot), and using a team of mobile sensors (robots). Specifically, our objective will be to minimize the number of sensors place and time taken by the robot(s) while ensuring that the posterior GP is accurately estimated.

For all the problems, we present polynomial-time approximation algorithms to ensure that the mean square error in prediction the underlying spatial field is smaller than a pre-defined threshold at each point. We also derive the lower bounds on the performance of any algorithm (including optimal) to solve respective problems are provided. We show that it is possible to learn a given spatial field accurately with high confidence. Note that, if the kernel parameters are optimized online, then, one

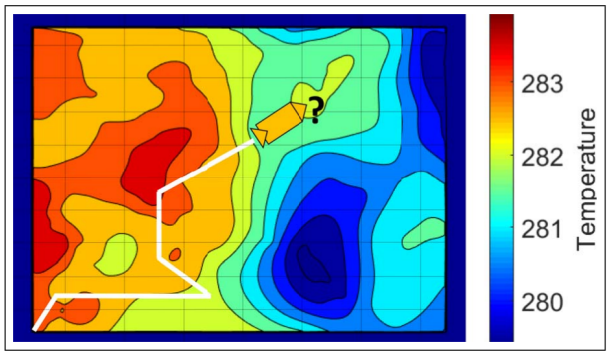


Figure 2.1: The environmental field here is the sea surface temperature of an area that is modeled as a GP in Sea of Japan on January 21, 2018 [1] from the data collected using a robotic boat.

would require an adaptive strategy. We will investigate this further in Chapter 3 but assume known hyperparameters for the algorithms presented in this chapter.

2.1 Related Work

We begin by reviewing the related work in sensor placement where the goal is to cover a given environment using sensors placed at fixed locations and mobile sensing where sensors can move and collect measurements from different locations.

Stationary Sensor Placement. When monitoring a spatial phenomenon, such as temperature or humidity in an environment, selection of a limited number of sensors and their locations is an important problem. The goal in this problem is to select the best k out of n possible sensor locations and use the measurements from these to make predictions about the spatial phenomenon. The typical formulation of a sensor selection problem makes it NP-hard [57]. Previous work used global optimization techniques such as branch and bound to exactly solve this problem [58,

59]. However, these exact approaches are often computationally intensive.

Since cameras are most commonly used as sensors, there has been significant work on stationary camera placement. This is typically formulated as an instance of the art-gallery problem [60, 61]: Find the minimum set of guards inside a polygonal workspace from which the entire workspace is visible. However, this version of the problem only covers vision-based sensors and does not consider noisy measurements [43].

An alternative approach from spatial statistics is to learn a model of the phenomenon, typically as a GP [62, 63]. The learned GP model can then be used to predict the effect of placing sensors at locations and thus optimize their placement. For a given GP model, many criteria including information-theoretic ones have been proposed to evaluate the quality of placement. Shewry and Wynn introduced the maximum entropy criterion [64] where the sensors are placed sequentially at the locations of highest entropy. Ko et al. [65] proposed a greedy algorithm by formulating the entropy maximization as maximizing the determinant of the covariance matrix. However, the entropy criterion tends to place the sensors at the boundary of the environment thus wasting sensed information [66]. Mutual information (MI) can be used as well [62, 67, 68]. Krause et al. [43] study the problem of maximizing MI for optimizing sensor placement problem. They present a polynomial-time approximation algorithm with constant factor guarantee by exploiting submodularity [69]. Eventually, they show that MI criterion leads to improved accuracy with a fewer number of sensors compared to other common design criteria such as entropy [64], A-optimal, D-optimal, and E-optimal design [70].

The above-mentioned methods estimate the prediction error indirectly. Nguyen et al. [71] consider choosing a set of n potential sensor measurements such that the root mean square prediction error is minimized. They present an annealing based algorithm for the sensor selection problem. Their algorithm starts by selecting a potential subset of cardinality k from the entire population of sensor locations. After that, it iteratively attempts to substitute the members of the selected subset by its neighbors according to an optimization criterion.

None of the criteria discussed above cannot directly make any guarantees on the MSE in predictions at each point in the environment. Instead, we design a sensor placement algorithm which results in an accurate reconstruction of the spatial field using the collected sensor measurements. Most works in the past have focused on optimizing an objective function (entropy, MI, etc.) given the resource constraints (limited energy, number of sensors, and time, etc.). We optimize the resource requirement given the objective constraint (MSE below a predefined threshold Δ), predictive accuracy more than a predefined threshold in our case.

Mobile Sensing. The goal in the mobile sensing problem, also known as Informative Path Planning (IPP), is to compute paths for robots acting as mobile sensors. Paths are being computed in order to accurately estimate some underlying phenomenon, typically a spatial field [24, 72]. A central problem in IPP is to identify the hotspots in a large-scale spatial field. Hotspots are the regions in which the spatial field measurements exceed a predefined threshold. In many applications, it is necessary to assess the spatial extent and shape of the hotspot regions accu-

rately. Low et al. presented a decentralized active robotic exploration strategy for probabilistic classification/labeling of hotspots in a GP-based spatial field [73]. The time needed by their strategy is independent of the map resolution and the number of robots, thus making it practical for *in situ*, real-time active sampling. Another formulation in hotspot identification is that of level set identification [74].

Previous works on level set boundary estimation and tracking [75–77] have primarily focused on communication of the sensor nodes, without giving much attention to individual sampling locations. Bryan et al. [78] proposed the straddle heuristic, which selects sampling locations by trading off uncertainty and proximity to the desired threshold level, both estimated using GPs. However, no theoretical justification had been given for its use and its extension to composite functions [79]. Gotovos et al. proposed a level set estimation algorithm, which utilizes GPs to model the target function and exploits its inferred confidence bounds to drive the selection process. They provided an information-theoretic bound on the number of measurements needed to achieve a certain accuracy, when the underlying function is sampled from a GP [80].

In many mobile sensing problems, it is not enough to identify only a few specific regions but estimate the entire spatial field accurately. It can be formulated as a path planning problem to observe a spatial field at a set of sampling locations, and then making inference about the unobserved locations [81]. Choosing and visiting the sample locations so that one can have an accurate prediction (point prediction and/or prediction interval) is of great importance in soil science, agriculture, and air pollution monitoring [68]. The objective functions used are usually submodular

and thus exhibit a diminishing returns property. Submodularity arises since nearby measurement locations are correlated [82]. Chekuri and Pal introduced a quasi-polynomial time algorithm [83] for maximizing a submodular objective along the path using a *recursive greedy strategy*. This algorithm was further extended by Binney et al. [84] for spatiotemporal fields using average variance reduction [85] as the objective function.

Zhang and Sukhatme proposed an adaptive sampling algorithm consisting of a set of static nodes and a mobile robot tasked to reconstruct a scalar field [46]. They assume that the mobile robot can communicate with all the static nodes and acquire sensor readings from them. Based on this information, a path planner generates a path such that the resulting integrated mean square error is minimized subject to the constraint that the boat has a finite amount of energy.

An important issue in designing robot paths is deciding the next measurement location [86–89], often referred to as the exploration strategy. Traditionally, conventional sampling methods [47] such as raster scanning, simple random sampling, and stratified random sampling have been used for single-robot exploration. Low et al. presented an adaptive exploration strategy called adaptive cluster sampling. It was demonstrated to reduce mission time and yield more information about the environment [45]. Their strategy performs better than a baseline sampling scheme called systematic sampling [90] using root mean squared error as a metric. A different adaptive multi-robot exploration strategy called MASP was presented in [91] which performs both wide-area coverage and hotspot sampling using non-myopic path planning. MASP allows for varying adaptivity and its performance is theo-

retically analyzed. Further, it was demonstrated to sample efficiently from a GP and *log*GP. However, the time complexity of implementing it depends on the map resolution, which limits its large-scale use. To alleviate this computational difficulty, an information-theoretic approach was presented in [92]. The time complexity of the new approach was independent of the map resolution and less sensitive to the increasing robot team size. Garnett et al. [93] considered the problem of active search, which is also about sequential sampling from a domain of two (or more) classes. Their goal was to sample as many points as possible from one of the classes.

Yilmaz et al. [94] solved the adaptive sampling problem using mixed integer linear programming. Popa et al. [87] posed the adaptive sampling problem as a sensor fusion problem within the extended Kalman filter framework. Hollinger and Sukhatme proposed a sampling-based motion planning algorithm that generates maximal informative trajectories for the mobile robots to observe their environment [95]. Their information gathering algorithm extends ideas from rapidly-exploring random graphs. Using branch and bound techniques, they achieve efficient optimization of information gathering while also allowing for operation in continuous space with motion constraints. Low et al. [27] presented two approaches to solve IPP for *in situ* active sensing of GP-based anisotropic spatial fields. Their proposed algorithms can trade-off active sensing performance with computational efficiency. Ling et al. [96] proposed a nonmyopic adaptive GP planning framework endowed with a general class of Lipschitz continuous reward functions. Their framework can unify some active learning/sensing and Bayesian optimization criteria and offer practitioners flexibility to specify choices for defining new tasks. Tan et al. [97]

introduced the receding-horizon cross-entropy trajectory optimization. Their focus was to sample around regions that exhibit extreme sensory measurements and much higher spatial variability, denoted as the region of interest. They used GP-UCB [51] as the optimization criteria which helps in exploring initially and converging on regions of interest eventually.

A naive implementation of GP prediction scales poorly with increasing training dataset size. Sparse GP frameworks can overcome this problem by using only a subset of the data to provide accurate estimates. A state-of-the-art sparse GP variant is SPGP [98–101]. The SPGP framework learns a pseudo subset that best summarizes the training data. Mishra et. al. introduced an online IPP framework AdaPP [1] which uses SPGP.

Sensing with Multiple Robots. Mobile sensing can be made faster by distributing the task among several robots. Multi-robot systems can do complex tasks and have been widely used in environmental sampling [102], coverage [103]. Robots can use local communication or control laws to achieve some collective goals.

Singh et al. [89] proposed a sequential allocation strategy that uses GP regression, which can be used to extend any single robot planning algorithm for the multi-robot problem. Their procedure approximately generalizes any guarantees for the single-robot problem to the multi-robot case. However, the approach works only when MI is the optimization objective. Cao et al. [27] presented two approaches along with their complexity analysis addressing a trade-off between active sensing performance and time efficiency. Luo et al. [104] combined adaptive sampling with

information-theoretic criterion into the coverage control framework for model learning and simultaneous locational optimization. They presented an algorithm allowing for collaboratively learning the generalized model of density function using a mixture of GPs with hyperparameters learned locally from each robot. Kemna et al. [105] created a decentralized coordination approach which first splits the environments into Voronoi partitions and makes each vehicle then run within their own partition. Other multi-robot approaches used in other domains, e.g. exploration and estimation with ground vehicles, include auction-based methods [106–108] and spatial segregation, typically through Voronoi partitioning [25, 109].

Tokekar et al. [110] presented a constant factor approximation algorithm for the case of accurately classifying each point in a spatial field. The first step in the algorithm is to determine potentially misclassified points and then to find a tour visiting neighborhoods of each potentially misclassified point. In this chapter, we study a regression version of the problem where every point is of interest. We exploit the properties of GP and squared-exponential kernel to find a constant-factor approximation algorithm.

2.2 Problem Formulation

We use GPs for formulating our problems.

2.2.1 Gaussian Processes

GPs are Bayesian non-parametric function approximators. GPs can be defined as a collection of infinitely many random variables, any finite subset $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_k\}$ of which is jointly Gaussian with mean vector $\mathbf{m} \in \mathbb{R}^k$ and covariance matrix $\mathbf{K} \in \mathbb{R}^{k \times k}$ [30].

Let $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_k\}$ denote the set of the training inputs. Let $\mathbf{y} = \{y_1, \ldots, y_k\}$ denote the corresponding training outputs. GPs can be used to predict the output value at a new test point, $\mathbf{x}$, conditioned on the training data. Predicted output value at $\mathbf{x}$ is normally distributed with mean $\hat{\mu}(\mathbf{x})$ and variance $\hat{\sigma}^2(\mathbf{x})$ given by,

$$\hat{\mu}(\mathbf{x}) = \mu(\mathbf{x}) + \mathbf{k}(\mathbf{x}, \mathbf{X})\Big[\mathbf{K}(\mathbf{X}, \mathbf{X}) + \omega^2\mathbf{I}\Big]^{-1}\mathbf{y}, \tag{2.1}$$

$$\hat{\sigma}^2(\mathbf{x}) = k(\mathbf{x}, \mathbf{x}) - \mathbf{k}(\mathbf{x}, \mathbf{X})\Big[\mathbf{K}(\mathbf{X}, \mathbf{X}) + \omega^2\mathbf{I}\Big]^{-1}\mathbf{k}(\mathbf{X}, \mathbf{x}), \tag{2.2}$$

where $\mathbf{K}(\mathbf{X}, \mathbf{X})$ is the kernel. The entry $\mathbf{K}_{\mathbf{x}_l, \mathbf{x}_m}$ gives the covariance between two inputs $\mathbf{x}_l$ and $\mathbf{x}_m$. $\mu(\mathbf{x})$ in Equation (2.1) is the prior mean of output value at $\mathbf{x}$. Some of the commonly used kernels are squared-exponential and Matern kernel [111, 112].

In this section, we formally define the problems and the algorithms. We assume that the environment is a two dimensional area $U \subset \mathbb{R}^2$ and the underlying spatial field is an instance of a GP, F [113]. F has an isotropic covariance function of the form,

$$C_Z(x, x') = \sigma_0^2 \exp\left(-\frac{(x - x')^2}{2l^2}\right); \forall x, x' \in U, \tag{2.3}$$

defined by a squared-exponential kernel where the hyperparameters σ_0^2 and l are known *a priori*. Let X denote the set of measurement locations within U produced by an algorithm.

Problem 1 (Placement)**.** *Find the minimum number of measurement locations, such that the MSE at each location in U is below $\Delta < \sigma_0^2$, i.e.,*

$$\begin{aligned} &\textit{minimize} \quad |X|, \\ &\textit{subject to} \quad MSE(x) \leq \Delta, \forall x \in U, \end{aligned}$$

where $|X|$ is the cardinality of X and $MSE(x)$ is the MSE at location x.

Problem 2 (Mobile)**.** *Find the minimum time trajectory for a mobile robot that obtains a finite set of measurements at one or more locations in U, such that the MSE at each location in U is less than Δ, i.e.,*

$$\begin{aligned} &\textit{minimize} \quad len(\tau) + \eta n(X), \\ &\textit{subject to} \quad MSE(x) \leq \Delta, \forall x \in U. \end{aligned}$$

τ denotes the tour of the robot. Robot travels at unit speed, obtains one measurement in η units of time and obtains $n(X)$ total measurements.

The robot may be required to obtain multiple measurements from a single location. Therefore, the number of measurements $n(X)$ can be more than $|X|$. For multiple robots, their tours can start at the same starting location (often referred to as a *depot*) or can start at different locations. In this chapter, our focus is on the former case. The latter case is more appropriate when the robots must persistently

monitor the environment.

Problem 3 (multi-robot)**.** *For k robots starting from a given starting location (depot), design a set of trajectories that collectively obtain a finite set of measurements at one or more locations in U, such that the MSE at each location in U is less than Δ, i.e.,*

$$\begin{aligned} &\textit{minimize} \quad \max_{i \in \{1,\ldots,k\}} len(\tau_i) + \eta n(X_i), \\ &\textit{subject to} \quad MSE(x) \leq \Delta, \forall x \in U. \end{aligned}$$

τ_i denotes the tour of the i^{th} robot and X_i the subset of measurement locations covered by the i^{th} robot. The robots travel at unit speed, obtain one measurement in η units of time. i^{th} robot obtains $n(X_i)$ total measurements.

The solution for Problem 1 is a subset of the solution to Problem 2. Further, the solution for Problem 3 is derived from the solution for Problem 2. The three algorithms build on top of each other by: (1) finding a finite number of measurement locations for the robot; (2) finding a tour to visit all the measurement locations; and (3) splitting the tour from step 2 in multiple sub-tours for k robots. We exploit the properties of squared-exponential kernel to find the measurement locations. By knowing the value at a certain point within some tolerance, values at nearby points can be predicted albeit up to a larger tolerance.

2.3 Algorithms

Before we discuss our algorithms, we provide a mathematical proof that Mean Squared Error (MSE) is equal to the posterior GP variance if the hyperparameters

are known. MSE measures the expected squared difference between an estimator and the parameter the estimator is designed to estimate [114]. The MSE at a location x for an estimator $\hat{f}$ is,

$$MSE\left(\hat{f}(x)\right) = Var\left(\hat{f}(x)\right) + \left(E[\hat{f}(x) - f(x)]\right)^2, \tag{2.4}$$

where the Equation 2.4 is the bias-variance expression for the estimator. GP predicted value $\hat{f}(x)$ at a location x is an unbiased estimator of the true value $f(x)$ [113] and has a normal distribution with mean given by Equation 2.1, and variance given by Equation 2.2. Among all linear and non-linear estimators, GP is the best in terms of minimizing MSE [115, 116]. Further, GPs are unbiased and hence, the MSE at a location x is equal to the posterior variance of the predicted value, *i.e.,*

$$MSE(x) = \hat{\sigma}^2_{x|X}. \tag{2.5}$$

From Equation 2.5, one can deduce that MSE for GPs is same as the posterior variance and hence, any guarantees for the posterior variance hold for MSE as well. Further, we derive two mathematical results for our algorithms. These two conditions provide a framework to prove the theoretical guarantees for the presented algorithms.

2.3.1 Necessary and Sufficient Conditions

We start by deriving necessary conditions on how far a test location can be from its nearest measurement location. A test location corresponds to a point in the environment where we would like to make a prediction.

Lemma 1 (Necessary Condition)**.** *For any test location x, if the nearest measurement location is at a distance r_{max} away, and,*

$$r_{max} > l\sqrt{-\log\left(1 - \frac{\Delta}{\sigma_0^2}\right)}, \tag{2.6}$$

then it is not possible to bring down the MSE below Δ at x.

Proof. Consider the posterior variance $\hat{\sigma}_x^2(n)$ at x (which is also equal to the MSE at x from Equation 2.5) after collecting n measurements, possibly from different locations. A lower bound on $\hat{\sigma}_x^2(n)$ can be obtained by assuming that all measurements were collected at the nearest location x_i to x. This is based on the fact that the closer the observation, lower the predictive variance. This can be proved as follows:

Consider two measurement locations $x_1,\ x_2$ and a test location x such that x_1 is closer to x. The posterior variance at x if a measurement was collected at x_1 can be computed as follows:

$$\hat{\sigma}_{x|x_1}^2 = k(x,x) - k(x,x_1)K(x_1,x_1)^{-1}k(x_1,x) \tag{2.7}$$

$$= \sigma_0^2\left(1 - \exp\left(-\frac{||x - x_1||^2}{l^2}\right)\right). \tag{2.8}$$

Similarly, the posterior variance at x if a measurement was collected at x_2,

$$\hat{\sigma}^2_{x|x_2} = \sigma_0^2 \left(1 - \exp\left(-\frac{||x - x_2||^2}{l^2}\right)\right). \tag{2.9}$$

From $||x-x_1||^2 < ||x-x_2||^2$ and $f(x) = -\exp(-x)$ being a monotonically increasing function, we have,

$$-\exp\left(-\frac{||x - x_1||^2}{l^2}\right) < -\exp\left(-\frac{||x - x_2||^2}{l^2}\right). \tag{2.10}$$

Using this to compare Equations 2.8 and 2.9 one can easily see that $\hat{\sigma}^2_{x|x_1} < \hat{\sigma}^2_{x|x_2}$.

Let the nearest measurement location x_i is distance r away from x. Assuming that all n measurements were collected at x_i, lower bound for posterior variance at x can be calculated using Equation 2.2,

$$\hat{\sigma}^2_x(n) \geq \sigma_0^2 - \begin{bmatrix} k(x,x_i), \ldots, k(x,x_i) \end{bmatrix} \begin{bmatrix} \sigma_0^2 + \omega^2 & & \sigma_0^2 \\ & \ddots & \\ \sigma_0^2 & & \sigma_0^2 + \omega^2 \end{bmatrix}^{-1} \begin{bmatrix} k(x,x_i) \\ \vdots \\ k(x,x_i) \end{bmatrix}. \tag{2.11}$$

It is worth mentioning that the square matrix in Equation 2.11 is of order $n \times n$ since there are n measurements. Substituting the value for $k(x, x_i) = \sigma_0^2 \exp\left(-\frac{r^2}{2l^2}\right)$

in Equation 2.11 and performing the required matrix operations, we get,

$$\hat{\sigma}_x^2(n) \geq \sigma_0^2 - \frac{\sigma_0^4}{\omega^2} \exp\left(\frac{-r^2}{l^2}\right) \left[1, \ldots, 1\right] \times \begin{bmatrix} 1 - \frac{1}{n+\frac{\omega^2}{\sigma_0^2}} & & \frac{-1}{n+\frac{\omega^2}{\sigma_0^2}} \\ & \ddots & \\ \frac{-1}{n+\frac{\omega^2}{\sigma_0^2}} & & 1 - \frac{1}{n+\frac{\omega^2}{\sigma_0^2}} \end{bmatrix} \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}. \tag{2.12}$$

Therefore,

$$\hat{\sigma}_x^2(n) \geq \sigma_0^2 - \frac{\sigma_0^4}{\omega^2} \left(n \left(1 - \frac{1}{n + \frac{\omega^2}{\sigma_0^2}} \right) - \frac{n(n-1)}{n + \frac{\omega^2}{\sigma_0^2}} \right), \tag{2.13}$$

$$\geq \sigma_0^2 \left(1 - \frac{\exp\left(-\frac{r_{max}^2}{l^2}\right)}{1 + \frac{\omega^2}{n\sigma_0^2}} \right). \tag{2.14}$$

Even if we had collected infinitely many measurements at the nearest location x_i, the posterior variance will still be lower bounded as,

$$\hat{\sigma}_x^2(n) > \lim_{n\to\infty} \sigma_0^2 \left(1 - \frac{\exp\left(-\frac{r_{max}^2}{l^2}\right)}{1 + \frac{\omega^2}{n\sigma_0^2}} \right), \tag{2.15}$$

$$= \sigma_0^2 \left(1 - \exp\left(-\frac{r_{max}^2}{l^2}\right) \right). \tag{2.16}$$

If the posterior variance at x even with the infinitely many measurements collected

at the nearest measurement location x_i (Equation 2.16) is greater than Δ, *i.e.*,

$$\Delta < \sigma_0^2 \left(1 - \exp\left(-\frac{r_{max}^2}{l^2}\right)\right) \implies r_{max} > l\sqrt{-\log\left(1 - \frac{\Delta}{\sigma_0^2}\right)}, \tag{2.17}$$

then it is not possible to bring down the MSE at x below Δ in any circumstance. □

Next, we prove a sufficient condition that if every point in the environment where no measurement is obtained (*test* location) is sufficiently close to a measurement location, then we can make accurate predictions at each point.

Lemma 2 (Sufficient Condition)**.** *For a test location $x \in U$, if there exists a measurement location $x_i \in X$, r distance away from x with n_{suff} measurements at x_i, such that,*

$$r \leq l\sqrt{-\log\left(\left(1 + \frac{\omega^2}{n_{suff}\sigma_0^2}\right)\left(1 - \frac{\Delta}{\sigma_0^2}\right)\right)}, \tag{2.18}$$

then GP predictions at x will be accurate, i.e., MSE at x will be smaller than Δ.

Proof. From Equation 2.14, we have an expression for variance of the posterior predictive distribution at x. Taking the other measurement locations into consideration can not increase the posterior variance at x. Information never hurts [117]! To prove the sufficiency, we consider n_{suff} measurements at x_i only and discard others knowing that the other locations can not increase the posterior variance at x. Bounding

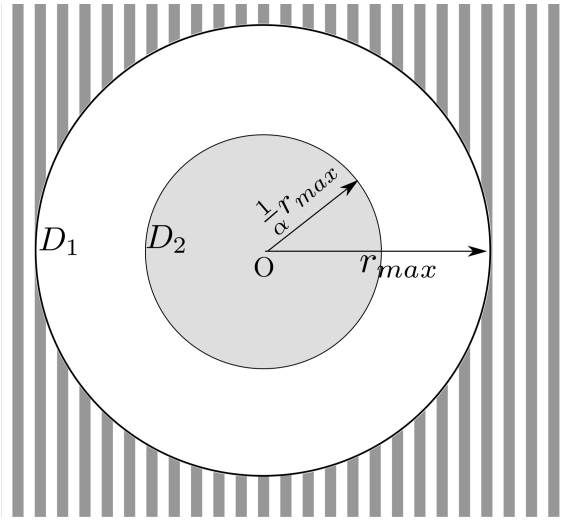


Figure 2.2: Collecting n_α measurements at O suffices to make accurate predictions at all points inside disk D_2 (Sufficient condition). No number of measurements at O can ensure predictive accuracy on points outside disk D_1 (Necessary condition).

the expression in Equation 2.14 with Δ results in,

$$\Delta \geq \sigma_0^2 \left(1 - \frac{\exp\left(-\frac{r^2}{l^2}\right)}{1 + \frac{\omega^2}{n_{\text{suff}}\sigma_0^2}}\right), \tag{2.19}$$

$$\exp\left(-\frac{r^2}{l^2}\right) \geq \left(1 + \frac{\omega^2}{n_{\text{suff}}\sigma_0^2}\right)\left(1 - \frac{\Delta}{\sigma_0^2}\right), \tag{2.20}$$

$$r \leq l\sqrt{-\log\left(\left(1 + \frac{\omega^2}{n_{\text{suff}}\sigma_0^2}\right)\left(1 - \frac{\Delta}{\sigma_0^2}\right)\right)}. \tag{2.21}$$

□

Lemma 2 gives a sufficient condition for GP predictions to be accurate at any given test location $x \in U$. The following lemma shows that a finite number of measurements $n_{\text{suff}} = n_\alpha$, are sufficient to ensure predictive accuracy in a smaller disk of radius $\frac{1}{\alpha} r_{max}$ around x_i, where $\alpha > 1$ (Figure 2.2).

Lemma 3. *Given a disk of radius $\frac{1}{\alpha} r_{max}$ centered at x_i, n_α measurements at x_i*

suffice to make accurate predictions for all points inside the disk, where,

$$n_\alpha \geq \left\lceil \frac{\omega^2}{\sigma_0^2} \frac{1}{\left(1 - \frac{\Delta}{\sigma_0^2}\right)^{\frac{1}{\alpha^2}-1} - 1} \right\rceil . \tag{2.22}$$

Proof. We want a sufficiency condition on the number of measurements n_α inside a disk of radius $\frac{1}{\alpha} r_{max}$. Lemma 2 gives an upper bound on the radius of a disk such that all points inside the disk will be accurately predicted after n_α measurements at the center. We construct a disk (D_2 in Figure 2.2), whose radius is equal to $\frac{1}{\alpha} r_{max}$ such that,

$$\frac{1}{\alpha} r_{max} \leq l \sqrt{-\log\left(\left(1 + \frac{\omega^2}{n_\alpha \sigma_0^2}\right)\left(1 - \frac{\Delta}{\sigma_0^2}\right)\right)}. \tag{2.23}$$

Plugging in the value of r_{max} from Lemma 1, squaring both sides in Equation 2.23 and re-arranging for n_α gives the required bound stated in Lemma 3. Ceiling function in Equation 2.22 accounts for the fact that n_α is an integer. □

A packing of disks of radius r_{max} gives a lower bound on the number of measurements required to ensure predictive accuracy. On the other hand, a covering of disks of radius $\frac{1}{\alpha} r_{max}$ gives us an upper bound on the number of measurements required. To solve Problem 1, what remains is to relate the upper and lower bound and present an algorithm to place the disks of radii $\frac{1}{\alpha} r_{max}$.

2.3.2 Placement of Sensors for Problem 1

We use an algorithm similar to the one presented by Tekdas and Isler [118] for stationary sensor placement in order to track a target using bearing sensors. In their case, the goal is to place sensors such that irrespective of where the target is in the environment, there are at least three sensors forming a triangle that get good quality bearing information of the target. They show how to cover the environment with disks and place a triangle of sensors within each disk. The setup is different from the one we have; however, we use a similar disk coverage strategy as a subroutine here. The exact procedure is outlined in Algorithm 1.

Algorithm 1 DiskCover

1: **procedure**
2: **Input:** An environment.
3: **Output:** Measurement locations.
4: **begin**
   1. Design a set $\mathcal{X}$ of disks of radii r_{max} which covers the environment and calculate a Maximal Independent Set (MIS) $\mathcal{I}$ of $\mathcal{X}$ greedily *i.e.*, $\mathcal{I} = \text{MIS}(\mathcal{X})$.
   2. Place disks of radii $3r_{max}$ concentric with disks in $\mathcal{I}$. Let the set of $3r_{max}$ radii disks is $\bar{\mathcal{X}}$.
   3. Cover each disk in $\bar{\mathcal{X}}$ with disks of radii $\frac{1}{\alpha}r_{max}$ as shown in Figure 2.3 and label centers of all disks of radii $\frac{1}{\alpha}r_{max}$.
   4. Return all the labeled points in previous step as measurement locations.
5: **end procedure**

Theorem 1. DiskCover *(Algorithm 1) gives an* $18\alpha^2$*-approximation for Problem 1 in polynomial time.*

Proof. Denote the set of measurement locations computed by the optimal algorithm to solve the Problem 1 by X^*. The function MIS in Step 1 of Algorithm 1 computes

a maximally independent set of disks: the disks in $\mathcal{I}$ are mutually non-intersecting (independent) and every disk in $\mathcal{X}\backslash\mathcal{I}$ intersects at least one disk in $\mathcal{I}$ (maximal). The set $\mathcal{I}$ can be computed by a simple polynomial greedy procedure: choose an arbitrary disk d from $\mathcal{X}$, add it to $\mathcal{I}$, remove all disks in $\mathcal{X}$ which intersect d, and repeat the procedure until no such d exists.

An optimal algorithm will collect measurements from at least as many measurement locations as the cardinality of $\mathcal{I}$. This can be proved by contradiction. Suppose an algorithm visits measurement locations fewer than the number of disks in $\mathcal{I}$. In that case, there will exist at least one disk of radius r_{max} in $\mathcal{I}$ which will not contain a measurement location. This means that there will be at least a point in that disk which will be more than r_{max} away from each measurement location. From Lemma 1, the robot can never make accurate predictions at that point and hence violating the constraint in Problem 1. Hence,

$$|\mathcal{I}| \ \leq \ |X^*|. \tag{2.24}$$

Every disk in $\mathcal{X}$ intersects at least one disk in $\mathcal{I}$ and hence, lies within $3r_{max}$ of the center of a disk in $\mathcal{I}$. As a result, $\bar{\mathcal{X}}$ disks cover all the $\mathcal{X}$ disks and hence, the entire environment.[1]

Collecting measurements from $18\alpha^2$ locations inside a $3r_{max}$ disk suffice to make accurate predictions in that disk (satisfying the Problem 1 constraint for

[1]Note that $3r_{max}$ is the minimum radius of the bigger disks to guarantee that the entire environment is always covered. In specific instances, it may be possible to cover the environment with smaller than $3r_{max}$ by selecting the MIS using a well-designed heuristic. However, there are environments where $3r_{max}$ will be necessary.

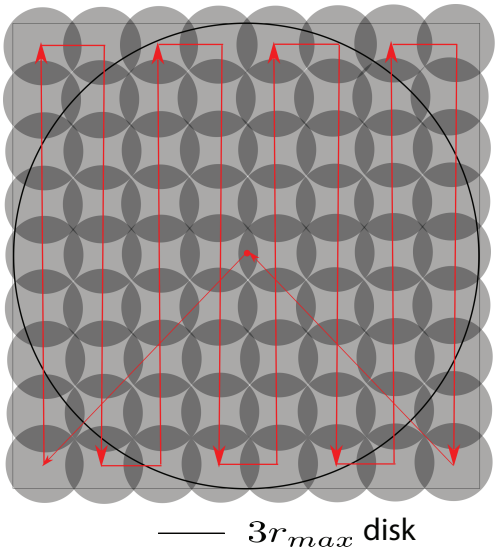


Figure 2.3: We cover each $3r_{max}$ radius disk with $\frac{1}{\alpha}r_{max}$ radii disks (smaller gray disks) in lawn-mower pattern. $18\alpha^2$ disks suffice to cover the bigger disk. The locations of disks of radii $\frac{1}{\alpha}r_{max}$ inside a disk of radius $3r_{max}$ are obtained by covering the square circumscribing bigger disk with smaller squares inscribed in smaller disks. The centers of smaller squares coincide with the centers of smaller disks.

points belonging to that disk) as illustrated in Figure 2.3. DiskCover collects measurement from $18\alpha^2$ such locations per disk in $\bar{\mathcal{X}}$. It collects measurements from a total of $18\alpha^2|\bar{\mathcal{X}}|$ locations, hence, satisfying the constraint for all points in the area covered by union of $\bar{\mathcal{X}}$ disks. Since, union of $\bar{\mathcal{X}}$ disks covers the entire environment, DiskCover satisfies the constraint for all points in the environment. Multiplying both sides of Equation 2.24 with $18\alpha^2$, we get, $18\alpha^2|\mathcal{I}| \leq 18\alpha^2|X^*|$. Note that $|\bar{\mathcal{X}}| = |\mathcal{I}|$. Hence,

$$18\alpha^2|\bar{\mathcal{X}}| \leq 18\alpha^2|X^*|, \tag{2.25}$$

$$n_{\text{DiskCover}} \leq 18\alpha^2|X^*|, \tag{2.26}$$

where, $n_{\text{DiskCover}}$ is the number of measurement locations for DiskCover. □

2.3.3 Finding an Approximate Optimal Trajectory for Problem 2

The algorithm for Problem 2 builds on the algorithm presented in the previous section. The locations where measurements are to be made become the locations that are to visited by the robot. The robot must obtain at least n_α measurements at the center of each disk of radius $\frac{1}{\alpha}r_{max}$. A pseudo-code of the algorithm is presented in the Algorithm 2.

Algorithm 2 DISKCOVERTOUR

1: **procedure**
2: **Input:** A set of measurement locations calculated from Algorithm 1.
3: **Output:** An approximate optimal tour visiting all the measurement locations.
4: **begin**
   1. Calculate approximate TSP tour visiting centers of the $3r_{max}$ radius disks (set $\bar{\mathcal{X}}$) disks.
   2. Cover $\bar{\mathcal{X}}$ disk containing the starting location in lawn-mower pattern visiting the centers of corresponding disks of radius $\frac{1}{\alpha}r_{max}$ and make n_α measurements at each center point.
   3. Move to the center of next $\bar{\mathcal{X}}$ disk along the tour calculated in Step 1.
   4. Repeat Steps 2 and 3 until all $\bar{\mathcal{X}}$ disks are covered.

5: **end procedure**

Theorem 2. DISKCOVERTOUR *(Algorithm 2) yields a constant-factor approximation algorithm for Problem 2 in polynomial time.*

Proof. From Theorem 1, we have a constant approximation bound on number of measurement locations. Let the time (travel and measurement time) taken by the optimal algorithm be T_1^*. Using notation from Theorem 1, we assume that the optimal traveling salesperson with neighborhoods (TSPN) time to visit disks in $\mathcal{I}$ be $T_{\mathcal{I}}^*$. In TSPN, we are given a set of geometric neighborhoods, and the objective is to find the shortest tour that visits at least one point in each neighborhood (disks

in this case) [110]. The optimal algorithm will visit at least all disks once in $\mathcal{I}$ which gives the following minimum bounds on the optimal travel time (T^*_{travel}) and optimal measurement time ($T^*_{measure}$),

$$T^*_{\mathcal{I}} \leq T^*_{travel}; \eta|\mathcal{I}| \leq T^*_{measure}. \tag{2.27}$$

Let the optimal time to visit the centers of disks in $\mathcal{I}$ be $T^*_{\mathcal{I}^C}$. An upper bound on $T^*_{\mathcal{I}^C}$ can be established by the fact that upon visiting each disk, the robot can visit the center of that disk and return back by adding an extra tour length of $2r_{max}$, *i.e.,* a detour of maximum length $|\mathcal{I}| \times 2r_{max}$ for all disks in $\mathcal{I}$. As a result: $T^*_{\mathcal{I}^C} \leq T^*_{\mathcal{I}} + 2r_{max}|\mathcal{I}|$. Using inequality from Equation 2.27: $T^*_{\mathcal{I}^C} \leq T^*_{travel} + 2r_{max}|\mathcal{I}|$. For any disk in $\bar{\mathcal{X}}$, the length of lawn-mower path starting from the its center and return back (Figure 2.3) after visiting all center points of $\frac{1}{\alpha}r_{max}$ disks will be of order $\mathcal{O}(\alpha^2)r_{max}$. Hence, the total travel time for DiskCoverTour is: $T_C + |\mathcal{I}|\mathcal{O}(\alpha^2)r_{max}$, where T_C is the $(1+\epsilon)$-approximated time with respect to the optimal TSP tour returned by the $(1+\epsilon)$-approximation algorithm to visit the centers of the disks in $\bar{\mathcal{X}}$ (or $\mathcal{I}$ disks since they are concentric). T_C can be calculated in polynomial time [119] having bounds: $T_C \leq (1+\epsilon)\ T^*_{\mathcal{I}^C}$, with $T^*_{\mathcal{I}^C}$ being the optimal TSP time to visit the centers of $\bar{\mathcal{X}}$ disks. Measurement time for DiskCoverTour is $18\alpha^2\eta n_2|\mathcal{I}|$. Hence,

the total time T^1_{alg} for DiskCoverTour is,

$$T^1_{alg} = T_C + |\mathcal{I}|\mathcal{O}(\alpha^2)r_{max} + 18\alpha^2\eta n_2|\mathcal{I}| \tag{2.28}$$

$$\leq (1+\epsilon)\ T^*_{\mathcal{I}^C} + \mathcal{O}(\alpha^2)r_{max}|\mathcal{I}| + 18\alpha^2\eta n_2|\mathcal{I}|, \tag{2.29}$$

$$\leq (1+\epsilon)\ \ (T^*_{travel} + 2r_{max}|\mathcal{I}|) + \mathcal{O}(\alpha^2)r_{max}|\mathcal{I}| + 18\alpha^2\eta n_2|\mathcal{I}|. \tag{2.30}$$

n_2 is the number of sufficient measurements required inside a disk of radius $\frac{1}{2}r_{max}$ (Lemma 3 with $\alpha = 2$). Length of any tour that visits k non-overlapping equal size disks of radii r is at least $0.24kr$ [120], which gives $0.24r_{max}|\mathcal{I}| \leq T^*_{\mathcal{I}}$. Combining this result with Equation 2.27 modifies the bounds in Equation 2.30 as,

$$T^1_{alg} \leq \left((1+\epsilon)\left(1+\frac{2}{.24}\right) + \frac{\mathcal{O}(\alpha^2)}{.24}\right)T^*_{travel} + 18\alpha^2 n_2 T^*_{measure}, \tag{2.31}$$

$$\leq \max\left(9.33(1+\epsilon) + \tfrac{82}{.24}, 72n_2\right)\left(T^*_{travel} + T^*_{measure}\right), \tag{2.32}$$

$$\leq \max\left(9.33(1+\epsilon) + \frac{\mathcal{O}(\alpha^2)}{.24}, 18\alpha^2 n_2\right)T^*_1 \tag{2.33}$$

$$\leq cT^*_1, \tag{2.34}$$

where c, a constant, is larger one of the two quantities inside the bracket in Equation 2.34. □

Note that the Algorithm 2 collects same number of measurements from each measurement location. There may be another algorithm that collects different num-

ber of measurements from different locations which may result in better performance. This modification is an avenue for future work.

2.3.4 Finding an Approximate Optimal Trajectory for Problem 3

When one robot can not handle a large territory, to speed up the task, k robots can be sent to collectively visit1 all measurement locations. A natural objective is to ensure that no robot has too large of a task. Hence, We choose our optimization criterion as minimizing the maximum of the k-robot tour costs. This is equivalent to minimizing the time taken by the last robot to return back to the common starting location. Our proposed algorithm only works if the robots start and return back to the same location – called depot. Any measurement location can be chosen as the depot but in our case, we assume that the robots start from and return back to a pre-defined depot.

We now describe an algorithm which employs a tour-splitting heuristic to plan for k robots. We modify the heuristic proposed by Frederickson et al. [121] to account for the measurement time, and not just the travel time.

Let the output tour of the robot from Algorithm 2 be denoted by τ and l_{max} be the distance of farthest measurement location from the depot.

Theorem 3. k-DiskCoverTour *(Algorithm 3) yields a* $(c+2)$ *approximation algorithm for Problem 3 in polynomial time, given a c-approximation algorithm for Problem 2.*

Proof. First, we prove that the time taken along every subtour is bounded and

Algorithm 3 k-DiskCoverTour

procedure
Input: Tour calculated from Algorithm 2, Depot location x_1.
Output: k approximate optimal paths visiting all measurement locations collectively.
begin
1. For j^{th} robot, $1 \leq j < k$, find the last measurement location $x_{p(j)}$ such that the time taken to travel from x_1 to $x_{p(j)}$ along τ is not greater than $\frac{j}{k}\left(T^1_{alg} - (2l_{max} + \eta n_2)\right) + (l_{max} + \eta n_2)$.
2. Obtain k subtours as $R_1 = (x_1, \ldots, x_{p(1)}, x_1)$, $R_2 = (x_1, x_{p(1)+1}, \ldots, x_{p(2)}, x_1)$, $\ldots$ $R_k = (x_1, x_{p(k-1)+1} \ldots, x_n, x_1)$.

end procedure

eventually show that the bound is within a constant factor of the optimal time. With k robots, let the subtours for 1^{st} and k^{th} robot are $x_1 \to x_{p(1)} \to x_1$ and $x_1 \to x_{p(k-1)+1} \to x_1$ respectively (an example with $k = 5$ is shown in Figure 2.4). Subtours for the remaining robots can be denoted by $x_1 \to x_{p(j-1)+1} \to x_{p(j)} \to x_1$, where $1 < j < k$.

Substituting $j = 1$ in Algorithm 3, the time to travel from x_1 to $x_{p(1)}$ along τ, $T(x_1 \xrightarrow{\tau} x_{p(1)})$ is no greater than $\frac{1}{k}\left(T^1_{alg} - (2l_{max} + \eta n_2)\right) + (l_{max} + \eta n_2)$. Time for the first subtour is hence bounded by $T(x_1 \xrightarrow{\tau} x_{p(1)}) + T(x_{p(1)} \to x_1)$, *i.e.*, $\frac{1}{k}\left(T^1_{alg} - (2l_{max} + \eta n_2)\right) + (l_{max} + \eta n_2) + l_{max}$. From the condition in Algorithm 3, we know that $x_{p(k-1)}$ is the last location such that,

$$T\left(x_1 \xrightarrow{\tau} x_{p(k-1)}\right) \leq \frac{k-1}{k}\left(T^1_{alg} - (2l_{max} + \eta n_2)\right) + (l_{max} + \eta n_2), \tag{2.35}$$

and hence,

$$T\left(x_1 \xrightarrow{\tau} x_{p(k-1)+1}\right) \geq \frac{k-1}{k}\left(T^1_{alg} - (2l_{max} + \eta n_2)\right) + (l_{max} + \eta n_2)\,. \tag{2.36}$$

Subtracting both sides from T^1_{alg},

$$T^1_{alg} - T\left(x_1 \xrightarrow{\tau} x_{p(k-1)+1}\right) \leq T^1_{alg} - \frac{k-1}{k}\Big(T^1_{alg} - (2l_{max} + \eta n_2)\Big) + (l_{max} + \eta n_2)\,, \tag{2.37}$$

which gives,

$$T\left(x_{p(k-1)+1} \xrightarrow{\tau} x_1\right) \leq \tfrac{1}{k}\Big(T^1_{alg} - (2l_{max} + \eta n_2)\Big) + (3l_{max} + 2\eta n_2), \tag{2.38}$$

and hence, time for the last subtour is bounded by $T\left(x_1 \to x_{p(k-1)+1}\right) + T\left(x_{p(k-1)+1} \xrightarrow{\tau} x_1\right)$, *i.e.*, $\tfrac{1}{k}\Big(T^1_{alg} - (2l_{max} + \eta n_2)\Big) + (3l_{max} + 2\eta n_2) + l_{max}$. Similar inequalities can be derived for remaining subtours as follows. For $1 \leq j \leq k-2$, following inequalities hold from Algorithm 3,

$$T(x_1 \xrightarrow{\tau} x_{p(j)+1}) \geq \tfrac{j}{k}\left(T^1_{alg} - (2l_{max} + \eta n_2)\right) + (l_{max} + \eta n_2)\,, \tag{2.39}$$

$$T(x_1 \xrightarrow{\tau} x_{p(j+1)}) \leq \tfrac{j+1}{k}\left(T^1_{alg} - (2l_{max} + \eta n_2)\right) + (l_{max} + \eta n_2)\,. \tag{2.40}$$

Subtracting Equation 2.39 from 2.40 results in,

$$T(x_{p(j)+1} \xrightarrow{\tau} x_{p(j+1)}) \leq \frac{1}{k}\left(T^1_{alg} - (2l_{max} + \eta n_2)\right), \tag{2.41}$$

i.e., the time taken along remaining subtours, $T\left(x_1 \to x_{p(j-1)+1} \xrightarrow{\tau} x_{p(j)} \to x_1\right)$, where $1 < j < k$, is also bounded by $\frac{1}{k}\left(T^1_{alg} - (2l_{max} + \eta n_2)\right) + 2l_{max}$. Hence, we can conclude that the time taken along each subtour does not exceed $\frac{1}{k}\left(T^1_{alg} - (2l_{max} + \eta n_2)\right) + (4l_{max} + 2\eta n_2)$.

Let T^k_{alg} be the time taken for largest of the k subtours generated by the Algorithm 3, and T^*_k be the cost of the largest subtour in an optimal solution to Problem 3. We have,

$$T^k_{alg} \leq \frac{1}{k}\left(T^1_{alg} - (2l_{max} + \eta n_2)\right) + (4l_{max} + 2\eta n_2) \tag{2.42}$$

$$\leq \frac{T^1_{alg}}{k} + (2l_{max} + \eta n_2)\left(2 - \frac{1}{k}\right). \tag{2.43}$$

From the triangle inequality, $T^*_k \geq \frac{1}{k}T^*_1$. It is natural to think that at least one robot will have to go to the farthest location from the depot and come back from there after collecting n_2 measurements which gives us a lower bound on the output of the optimal algorithm, *i.e.,* $2l_{max} + \eta n_2 \leq T^*_k$. Combining these results with

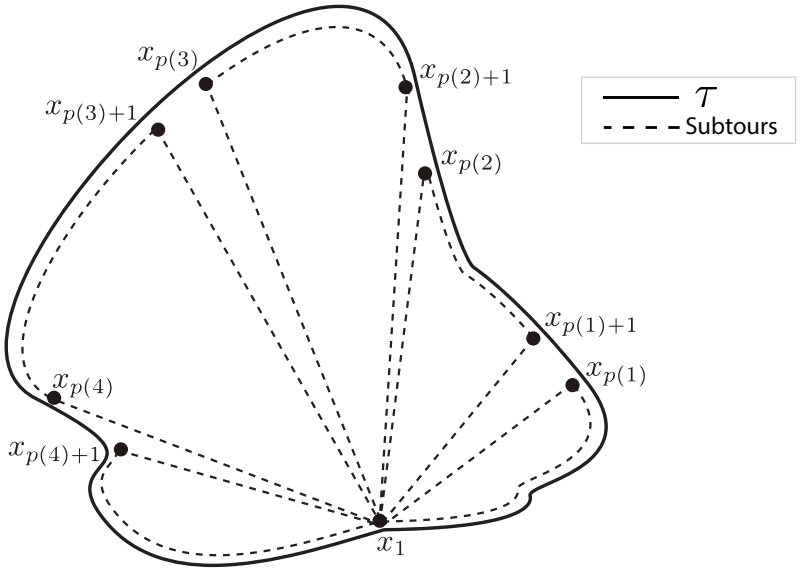


Figure 2.4: Splitting the tour for one robot (τ) into 5 subtours. The solid line shows an initial single robot tour τ starting and ending at x_1. The dotted lines denote the individual robot subtours starting and ending at x_1 obtained by splitting the single tour τ.

Equation 2.34, we get,

$$T_{alg}^{k} \leq \frac{c}{k} T_1^* + T_k^* \left(2 - \frac{1}{k}\right) \tag{2.44}$$

$$\leq \left(c + 2 - \frac{1}{k}\right) T_k^* \tag{2.45}$$

$$\leq (c+2)\, T_k^*. \tag{2.46}$$

□

2.4 Empirical Evaluation

In this section, we report results from empirical evaluation of the theoretical results. We show qualitative and quantitative comparison of our algorithms with

other baseline strategies through simulations using precision agriculture as our motivating example.

Dataset We use a real-world dataset [122], collected from a farm, consisting of organic matter (OM) measurements manually collected from several hundred locations within the farm. The maximum and minimum values of the underlying field are 54.6 *parts per million (ppm)* and 25.4 *ppm* respectively shown by the colorbar (Figure 2.5(a)). Taking this into account, we set Δ to be equal to 4 which is 10% of the average of maximum and minimum field values. We use a simulated sensor that returns a noisy version of the ground truth measurement with an additive Gaussian noise of variance, $\omega = 0.0361$.

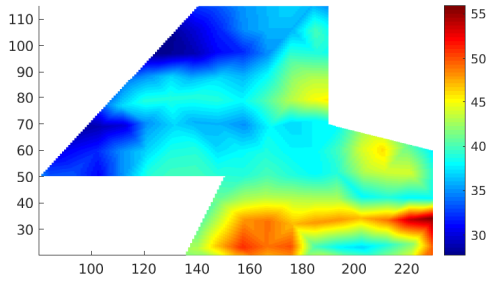
The squared-exponential kernel has three hyperparameters: length scale (l), signal variance (σ_0^2), and noise variance (ω^2). The values of l, σ_0, and ω^2 were estimated to be 8.33 meter, 12.87, and 0.0361 respectively by minimizing the negative log-marginal likelihood of the manually collected data. We assume that the estimated values are the true values of the kernel hyperparameters. In a general application where some prior data is available, the hyperparameters can be estimated in a similar way. We used the GPML toolbox to perform the necessary GP operations [123].

2.4.1 Qualitative Example

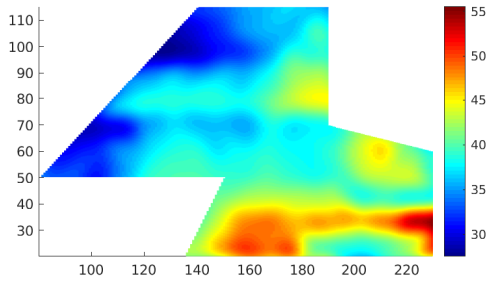
Stationary Sensor Placement The final predicted OM content after performing inference using the measurements obtained is shown in Figure 2.5(b). This predicted

OM content is the average of ten trials. In each trial, the reported value by the sensor can be different even at the same location because of the simulated noise. Figure 2.5(c) shows a plot of the prediction error averaged over those ten trials. We observe that the average prediction error is below $\Delta = 4$ *ppm* at each location in the environment. It is important to mention that average prediction error is not same as the MSE. The MSE at a location is the expected squared error in prediction at that location. The average prediction error, referred as empirical MSE in the following text, is an empirical estimate of that expectation. As the number of runs increases, the empirical MSE will converge to the actual MSE. We verify it through simulations and report the results later. Our theoretical guarantees hold for the MSE and not for the empirical MSE. However, one can expect that the empirical MSE will also be less than the pre-defined threshold Δ given enough trials. The regions where the OM content changes sharply tend to be more erroneously predicted as shown by the lighter colored regions in Figure 2.5(c). This can be attributed to the inherent smoothness assumptions of a squared-exponential kernel.

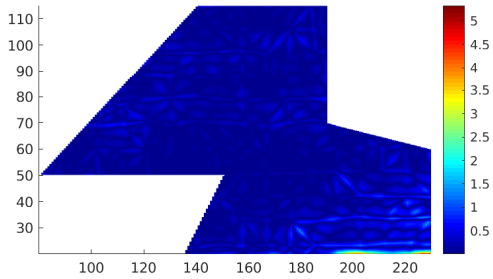
Single and Multi-Robot Tours The measurement locations computed by the DiskCoverTour are shown in Figure 2.6(a). As post-processing, we removed the redundant measurement locations in overlapping $3r_{max}$ radii disks. After performing this step, the total number of measurement locations was 2320. A covering of the farm with disks of radii $3r_{max}$ and an approximate optimal tour visit the centers of those disks calculated by DiskCoverTour is shown in Figure 2.6(b). We compute the optimal TSP tour since this is a reasonably sized instance. The lawn-mower detours



(a) The actual organic matter content (*ppm*).



(b) The predicted organic matter content (*ppm*).



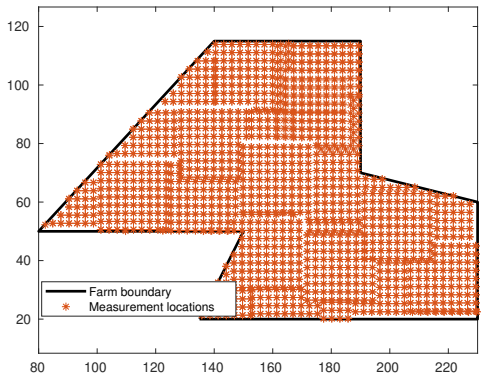
(c) Prediction error between the actual and the predicted OM content (*ppm*).

Figure 2.5: Actual and predicted OM content comparison. The farm is shown as the colored region with the colorbar denoting concentrations at different locations. All distance units are in *meter*.

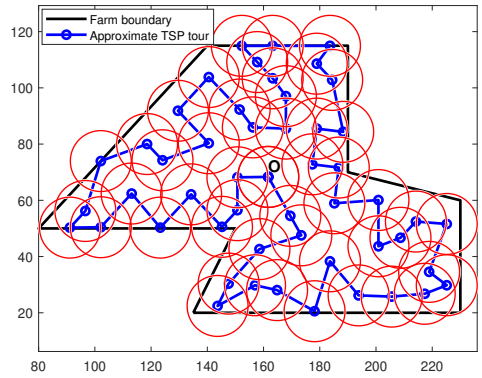
visiting individual $3r_{max}$ disks have been omitted to make the figure more legible. For the multi-robot version, we assume that we have three robots. Splitting of a single robot tour (Figure 2.6(b)) in three subtours is shown in Figure 2.6(c). The robots start from a common depot.

Varying values of Δ: In some applications, one may be interested in having more accurate predictions in some parts of the environment than others. Our algorithms provides a way to choose locations and plan paths in such applications as well. To demonstrate this, we divide the farm in three sub-environments that have different Δ tolerances. The left-most, middle, and right-most regions have thresholds of $\Delta = 6,\ 4$ and 2 *ppm* respectively. We solve for the measurement locations independently in each region. The corresponding r_{max} values were calculated to be 4.97, 3.93, and 2.70 meters respectively using Equation 2.17. Figure 2.7 shows the measurement locations. One can qualitatively observe that the algorithm places fewer measurement locations in the left-most sub-environment which allows for the highest error tolerance.

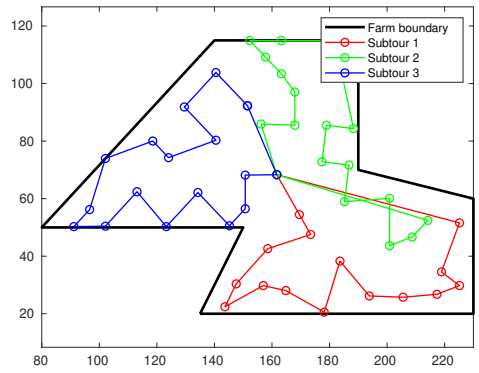
An approximate TSP tour visiting the centers of all $3r_{max}$ disks, in all three regions, is shown in Figure 2.8. The size of the disks shrinks as one moves to the right-most sub-region which has the least tolerance for prediction error. The TSP tour goes outside the environment in this case, which may be feasible if an aerial robot is used to monitor the farm. In case of applications, where the robot must stay inside the environment, we can enforce this constraint by replacing the Euclidean edge weights in the TSP input graph with the length of the shortest path between



(a) Measurement locations calculated by the DiskCover algorithm. There are 1012 measurement locations.



(b) The red disks are of radii $3r_{max}$ which are concentric with disks of radii r_{max} in $\mathcal{I}$. The depot is denoted by O.



(c) All robots start from and return to the depot after making measurements.

Figure 2.6: Measurement locations and the tours computed by DiskCover and k-DiskCoverTour. For Figures 2.6(b) and 2.6(c), the complete tours that take detours to visit all the locations in Figure 2.6(a) have been omitted to make the figures more legible.

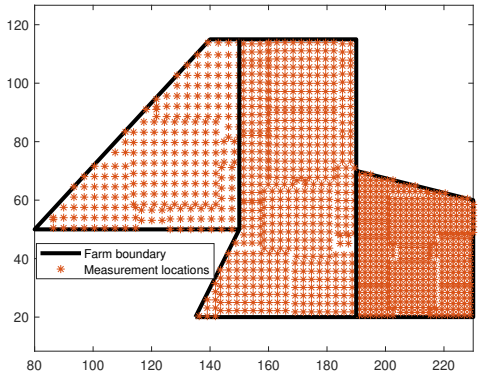


Figure 2.7: Measurement locations for different values of Δ. .

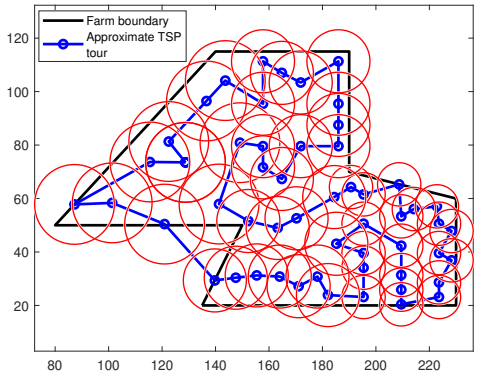


Figure 2.8: An approximate TSP tour to visit the $3r_{max}$ disks. r_{max} values depend on Δ (Equation 2.17) and hence, vary in different Δ sub-regions. Note the shrinking size of disks as one moves towards right.

.

two vertices inside the environment.

2.4.2 Comparisons with Pre-defined Lawn-mower Tours

One can observe from Figure 2.6(a) that the measurement location pattern closely resembles a lawn-mower pattern. It motivated us to compare the performance of our algorithms and with lawn-mower plan. Figure 2.9 and 2.10 show the average posterior variance and average empirical (for ten trials) MSE respectively for a

pre-defined lawn-mower pattern with varying grid resolutions on a semi-logarithmic scale. Note that the posterior variance at a test location is always same in each trial because it is not a function of the actual measurement value. The blue horizontal line corresponds to DCT and is shown for the sake of comparison.

A plot of the time taken by the robot to cover lawn-mower patterns with various grid resolutions is shown in Figure 2.11. The lawn-mower lines in Figures 2.9, 2.10, and 2.11 intersect the DCT lines at approximately a resolution of 2 meters. It suggests that one would need to create a grid of approximately that resolution to achieve same performance as DCT. Figure 2.12 shows the average posterior variance for DCT and a pre-defined lawn-mower of resolution 2.4 meter as a function time elapsed along a deployment (averaged over 10 deployments). We chose a resolution of 2.4 meters since a lawn-mower planner with this resolution has approximately the same number of measurement locations as DiskCoverTour. We observe that both perform almost the same empirically.

One may wonder why we cannot simply use the lawn-mower pattern, instead of DiskCoverTour. To create a lawn-mower pattern, one would need to pick a grid resolution. There is no systematic way of picking this resolution without enumerating a few combinations to analyze the trade-off between time and posterior variance or MSE. This can be wasteful. Instead, we present a systematic way of planning the measurement locations and give explicit theoretical guarantees on time and MSE or variance.

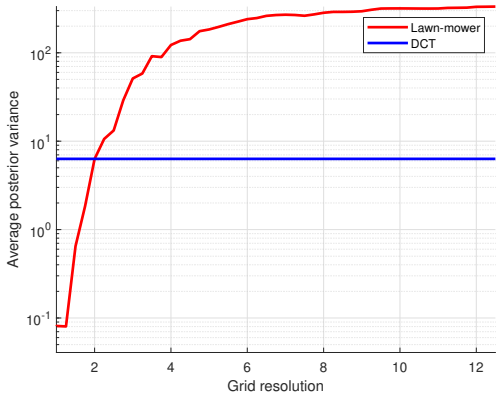


Figure 2.9: Average posterior variance for varying degree of lawn-mower resolutions.

.

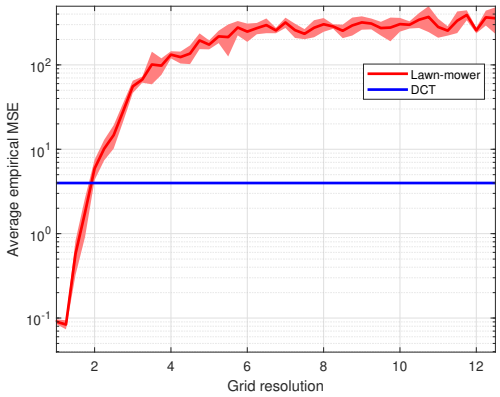


Figure 2.10: Average empirical MSE for varying degree of lawn-mower resolutions.

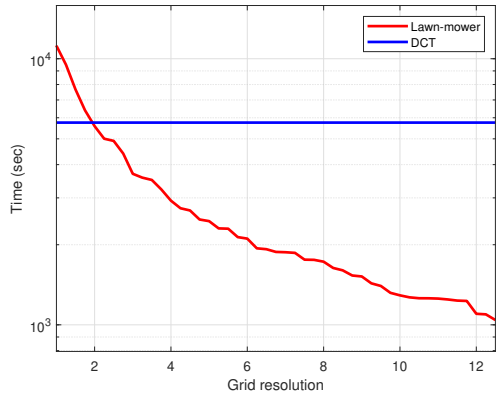


Figure 2.11: Time spent by the robot with lawn-mower planners of different grid resolutions.

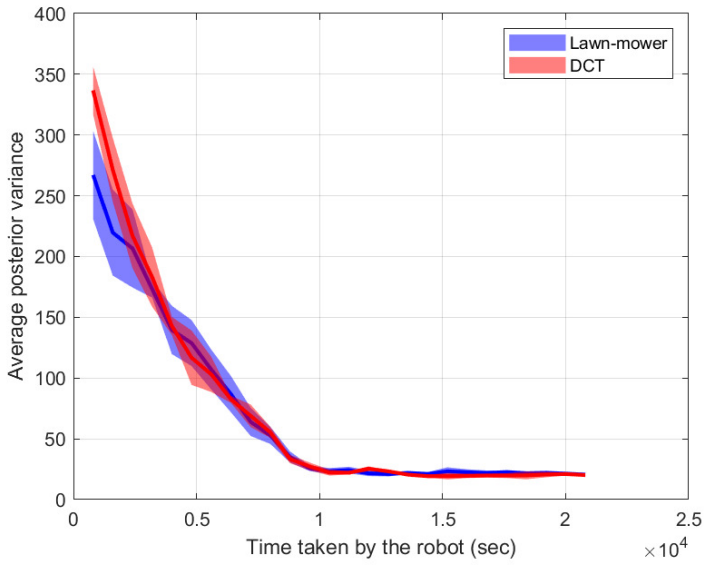
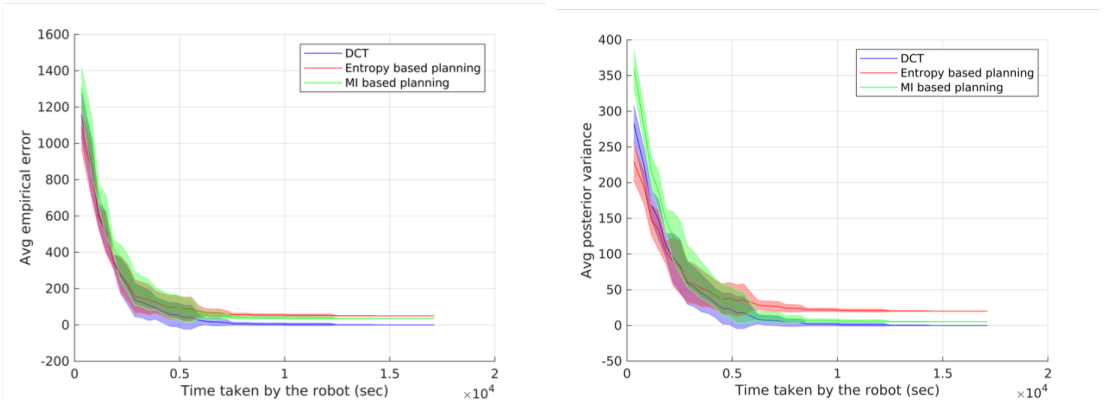


Figure 2.12: Average posterior variance as a function of time spent by the robot.

2.4.3 Comparison With Other Baselines

A comparison between DiskCoverTour and two baselines, entropy-based, and MI-based planner is shown in Figure 2.13. The measurement locations for the entropy-based and MI-based planners were calculated greedily, *i.e.,* picking the next location at the point of maximum entropy and MI respectively as described in [43]. We study the average posterior variance and average empirical MSE in prediction as a function of the total time (measurement plus traveling) spent by the robot on the farm for each planner. After finding the measurement locations for each planner separately, TSP tours visiting those locations were calculated. The X axis in Figure 2.13 shows the time taken along a tour and the Y axis shows the respective metrics based on measurements collected until that point in time along the tour (averaged over ten trials). We observe that DiskCoverTour performs at par with



(a) Average empirical MSE (b) Average posterior variance

Figure 2.13: DiskCoverTour performs comparably with entropy-based and MI-based strategies. The shaded regions correspond to the standard deviation taken over ten trials.

other planners. The entropy-based planner results in the most significant reduction in posterior variance and average empirical MSE initially. This can be explained by the fact that the entropy-based planning tends to spread the measurement locations far from each other resulting in covering a bigger portion of the environment initially. However, DiskCoverTour converges to a lower value of average empirical MSE and average posterior variance.

2.4.4 MSE and Variance

We verify our hypothesis that MSE is equal to the posterior variance for GPs. A plot of the mean percent difference between the empirical MSE and the posterior variance is shown in Figure 2.14. The mean is computed over approximately 5600 test locations which are different from the measurement locations and placed on a grid. As the number of trials increases, the mean difference between empirical MSE, which is essentially the MSE given enough number of trials, and the posterior vari-

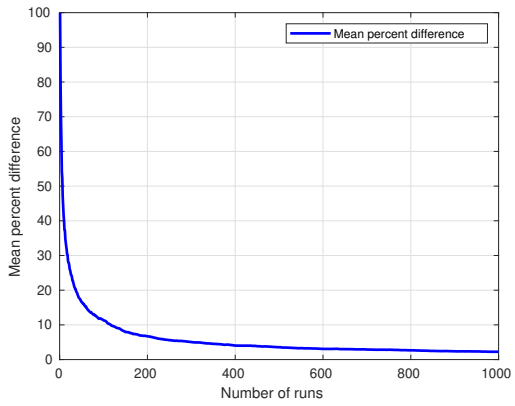


Figure 2.14: The mean percentage difference between the empirical MSE and the posterior variance.

.

ance decreases implying that the empirical MSE converges to the posterior variance asymptotically. In each trial, the measurement locations, test locations, and the hyperparameters are same, and therefore the variance estimates are same as well. However, the predicted value in each trial, and hence the prediction error, may be different since the actual measurement collected can be different in each trial due to the simulated noise. The effect of noise will decrease as one computes empirical estimate over a larger number of trials.

2.5 Conclusions

In this chapter, we study several problems: Placing the minimum number of stationary sensors to track a spatial field, mapping a spatial field by a single as well as multiple robots while minimizing the time taken by the robots. For all the problems, we propose polynomial-time approximation algorithms to ensure that the mean square error in prediction the underlying spatial field is smaller than a pre-

defined threshold at each point. We also derive the lower bounds on the performance of any algorithm (including optimal) to solve respective problems are provided. We show that it is possible to learn a given spatial field accurately with high confidence without planning adaptively. Note that, if the kernel parameters are optimized online, then, one would require an adaptive strategy.

The algorithms suggested in this chapter perform comparatively with the baseline planners developed earlier. Our algorithms have theoretical bounds on their performance. The algorithms can also be generalized to 3D mapping, even though we illustrate using 2D examples. The disks in the 2D case will be replaced by spheres in 3D. The disk packing/covering problem becomes a sphere packing/covering. The tour will need to visit points in 3D, as opposed to 2D. The existing TSP algorithms already apply to the 3D case [81].

Chapter 3: Adaptive Planning for Finding Hotspot in an Environment with a Single Robot

In this chapter, we study an environmental monitoring problem where the goal is to identify a hotspot in the environment using a mobile sensor. Identifying hotspots is a crucial problem [124, 125]. For example, consider deploying robots in a nuclear power plant to monitor potential leakages by measuring radiation levels. Radiation levels will relatively be higher near the sites of potential leakages [126]. Hence, by identifying the sites of higher nuclear radiations using robot sensors, we can efficiently find any potential leakage [127]. In these scenarios, one would be better off just by identifying the hotspot (i.e., maxima) instead of learning the entire environment accurately as we did in Chapter 2. We consider the case where the robot operates for a pre-defined time in the environment. As in the previous chapter, we will use Gaussian Processes (GPs) to model the underlying spatial field. We also present an algorithm that relaxes the assumption of GP hyperparameters being known. In general, GP hyperparameters are optimized during the process and can be a computationally prohibitive task. We introduce a Monte Carlo Tree Search (MCTS) [128] based algorithm that uses an Upper Confidence Bound (UCB) [51] style exploration and finds a hotspot with or without the need to do hyperparameter

optimization. In the latter case, we provide a computationally inexpensive way of adapting the hyperparameters. We compare their empirical performance on a carefully designed synthetic spatial field as well as a real-world dataset of Chlorophyll density from a Pacific ocean subregion. Our results suggest that we may not always need to accurately find the hyperparameters if finding a hotspot is the only goal in a robotic mission.

3.1 Monte Carlo Tree Search (MCTS)

MCTS has gained popularity in various AI applications in last couple of decades. MCTS is a best-first strategy. It explores the nodes in a tree which are most promising to have higher rewards first. This is opposed to brute-force exploration strategies such as Depth First Search and Breadth First Search which require to traverse through all the nodes in a tree and do not take into account the information collected from the already explored component of the tree. MCTS progressively explores a tree, guided by the results of previous exploration of that tree. MCTS is an anytime algorithm for which more computing power leads to better exploration. A general MCTS approach has four fundamental steps: Selection, Expansion, Simulation, and Backpropagation.

1. Selection: Starting at the root node, a child selection policy is applied recursively to descend through the nodes until an expandable node is reached. A node is expandable if it represents a non-terminal state and has unvisited children.

2. Expansion: An unvisited child node of the expandable node is added to expand the tree.
3. Simulation: A simulation is run from the new node according to a random policy to get the rewards.
4. Backpropagation: The simulation result is backpropagated through the newly added node of the expanded node to the root node.

We explain these steps with one of the most popular and provably efficient MCTS algorithm called UCT [128]. UCT is inspired from the idea of Upper Confidence Bound (UCB) algorithm UCB1 from the multi-armed bandit literature [129]. UCB1 has been shown to have the best possible bound on the growth of regret. It therefore makes a promising candidate to address the exploration-exploitation dilemma in MCTS.

The UCT algorithm is shown in Algorithm 1. For each node v, we keep track of the total reward collected ($Q(v)$) and the number of times that node has been visited (n_v). When deciding in the Selection step among all the possible children we compare their UCT values in Line 11. The UCT expression contains the summation of an exploitation term (average reward) and an exploration term (number of times that child node has been visited). This is inspired from the idea of UCB1 and is shown to achieve the best regret bound. In the Simulation step, we randomly sample nodes starting from the new child node of the expanded node and then backpropagate the reward collected from the new node to the root node. Note that here we just use the reward at the end of the rollouts but in many implementations,

one can use different reward functions, such as the average of the rewards collected while doing the rollouts.

Algorithm 4 UCT

Input: root node v_0 of the tree.
for $iter \leq MaxIter$
    $v_{expandable} \leftarrow Selection(v_0)$
    $v_{new} \leftarrow Expansion(v_{expandable})$
    $reward \leftarrow Simulation(v_{new})$
    $Backpropagate(reward, v_{new})$
return $\text{argmax}_{child \in v_0.children} \frac{Q(child)}{n_{child}} + 2\sqrt{\frac{\log(n_{v_0})}{n_{child}}}$

function Selection(v)
    while v is fully expanded
        $v \leftarrow \text{argmax}_{child \in v.children} \frac{Q(child)}{n_{child}} + 2\sqrt{\frac{\log(n_v)}{n_{child}}}$
    return v

function Expansion(v)
    Randomly pick and return an unvisited children of v

function Simulation(v)
    while $roll \leq MaxRollout$
        $v \leftarrow$ Randomly pick a child from $v.children$
        $roll += 1$
    return v

function Backpropagate(r, v)
    while v is not None
        $Q(v) = Q(v) + r$
        $n_v = n_v + 1$
        Backpropagate(r, $v.parent$)

3.2 Related Work

The hotspot identification problem is closely related to several other common problems in the IPP literature including the source-seeking problem. The aim of the source-seeking problem is to direct an autonomous vehicle to move towards

the source of the considered phenomenon and to reach it, in order to determine its position [130, 131]. Marchant and Ramos used Bayesian Optimization (BO) to effectively identify the ozone concentration over the contiguous USA [132]. They also used BO to take continuous path planning into account. Unfortunately, these works estimate the kernel a priori in an offline manner that requires some prior access to at least a few data samples.

For hotspot mapping, Chen and Liu presented an anytime multi-objective informative planning method called Pareto Monte Carlo tree search which allows the robot to handle potentially competing objectives such as exploration versus exploitation [133]. However, their approach focuses on all the regions that have a value higher than the the median value of the spatial field and not reporting one of the hotspot locations. Tan et. al., focused on adaptive sampling on a GP using the receding-horizon Cross-Entropy trajectory optimization. By using GP-UCB as the optimization criteria they adaptively planed sampling paths that leads to regions of high values [97]. Sung et. al., studied the problem of hotspot identification in limited time using a UAV [134].

There has been a lot of work on informative planning where the hyperparameters are assumed to be known [28,43,49,77,84,95,135]. However, this may not always be the case in real-world applications. In online planning, the hyperparameters are estimated during the execution of the algorithm. Binney et al. [84] estimated hyperparameters by using data from an initial run executed before running any other sampling routines. However, they do not specify how much prior data one would need to estimate the hyperparameters well enough. Thompson et al. [136] initially

estimated the hyperparameters by starting every adaptive mission with fixed-time straight-line paths, and then periodically re-estimated the hyperparameters during the sampling. This may make the sampling computationally expensive if the hyperparameters are estimated too frequently. They also assume that that the measurements collected initially during the straight-line paths are representative of the entire field. Kemna et. al. [137] ran some pilot surveys to initialize the hyperparameters and included the time taken during these pilot surveys into overall planning. Garg and Ayanian [138] used the particle filtering approach to estimate the randomly initialized hyperparameters during execution. Albeit, this approach allowed them to learn spatio-temporal fields as well, the performance of the approach is sensitive to random initialization. Contrary to the above approaches, we are interested in finding a hotspot rather than learning the entire field.

Multi-Armed Bandits (MAB) [91,139] have also been used for hotspot identification [51,140]. Tan et. al. [55] used cross-entropy as the optimization criterion in a MAB setting. However, much recent work on multi-armed bandit problems focuses on the cumulative expected regret and assumes that the GP hyperparameters are known.

Monte Carlo Tree Search has been commonly used in informative path planning and hotspot identification [128, 133, 141, 142]. MCTS including the Upper Confidence Tree (UCT) algorithm and its variants [143], have been investigated to handle sequential decision problems. They have been shown to have consistencies in balancing the exploration-exploitation efficiently in many applications [144,145]. Since exploration and exploitation are two conflicting objectives, they can not be

achieved simultaneously generally [133]. Hence, it is crucial to devise a metric that balances this trade-off. Our algorithm AdaptGP-MCTS uses GP-UCB values as the reward heuristics and balances the exploration-exploitation. The performance of UCB planners has been shown to be sensitive with respect to β value [146]. In order to avoid, subjective tuning, a time-varying β_t is used. A slower growth of β_t can result in exploitative behavior while a faster growth can result in overly exploratory behavior. Audibert et. al. [147] show that a growth rate faster than the logarithmic growth [51] is more efficient for the terminal regret. For a better performance on terminal regret, Tolphin and Shimony recommend using. In this work, we use a squared root growth of β which has been proved to achieve better performance on terminal regret [148].

We present the MCTS based algorithm that does not require the GP hyperparameters to be known *a priori* as opposed to much of the past research [149, 150] that assume the hyperparameters to be known. We show that adapting the hyperparameters in a specific way without optimizing as described in latter sections suffices to find the hotspots. Further, from a practical point of view, a robot operator may not always have access to very accurate sensors. Hence, we also provide exhaustive empirical studies about how the sensor quality affects the performance of the presented algorithms.

3.3 Problem Formulation

In this section, we formally define the problem and the assumptions made. We assume that the spatial field under consideration defined over a 2-dimensional environment $U \in \mathbb{R}^2$ is an instance of a GP, F. F is defined by a covariance function of the form,

$$C_Z(x, x') = \sigma^2 \exp\left(-\frac{(x - x')^2}{2l^2}\right); \forall x, x' \in U, \tag{3.1}$$

defined by a squared-exponential kernel and the hyperparameters σ^2 and l are not known. We now formally describe the problem and the algorithms proposed in this chapter.

Problem 4 (Terminal Regret)**.** *Given an operating time budget T, plan a trajectory under budget T for a mobile robot that obtains measurements from U, and reports the location of maxima of the spatial field f at the end, i.e.,*

$$\begin{aligned} &\textit{minimize} \quad f(x^*) - f(\hat{x}), \\ &\textit{subject to} \quad len(\tau) + n\eta \leq T. \end{aligned}$$

τ denotes the tour of the robot. Robot travels at unit speed, obtains one measurement in η units of time and collects n total measurements. $\hat{x}$ is the location of the maxima of the predicted field while x^* is the location of maxima of the true spatial field. We do not know x^* and we also do not know f. We only know the GP prediction $\hat{f}$. The task is to use $\hat{f}$ to be able to predict x^*.

3.4 Algorithm

Our algorithm AdaptGP-MCTS uses an MCTS-based planner [149]. AdaptGP-MCTS (Algorithm 5) shows the main function that calls the planner MCTS shown in Line 4. Once the planner returns the next measurement location (Line 4), the robot goes there and collects the measurement. We do a full GP hyperparameter optimization only at the end when the robot has used all the time budget.

Algorithm 5 AdaptGP-MCTS

1: **Input:** Initial hyperparameters $\sigma_0 = 1$ and $\mathbf{l_0} = diam(Env)$, $\mathbf{X} = \{\}$, $\mathbf{y} = \{\}$, Planner().
2: **while** $t \leq$ Total time budget T
3: $\quad \hat{\mu}_t(x), \hat{\sigma}_t(x) \leftarrow GP.Predict(\mathbf{X}, \mathbf{y})$
4: $\quad x_t \leftarrow Planner(\hat{\mu}_t(x), \hat{\sigma}_t(x), t)$
5: $\quad y_t = f(x_t) + \epsilon$
6: $\quad \mathbf{X}.append(x_t); \mathbf{y}.append(y_t)$
7: $\quad$ Update $\sigma_t = \sigma_0 \log(t); \mathbf{l_t} = \mathbf{l_0}/\log(t)$
8: Do a full GP hyperparameter optimization with $(\mathbf{X}, \mathbf{y})$
9: Estimate the posterior mean $\hat{\mu}$
10: return $\text{argmax}_{x \in U} \hat{\mu}(x)$

AdaptGP-MCTS does not optimize for GP hyperparameters (*e.g.*, maximum likelihood estimation) at each time step. Instead, it changes the hyperparameters starting with an initial l_0 and σ_0^2 for the length scale and the signal variance, respectively. Over the iterations, it monotonically decreases the length scale and monotonically increases the signal variance so that the GP model can capture more complex function candidates (Figure 3.1) [149]. As the length scale l decreases and signal variance σ^2 increases, we can capture rapidly changing and potentially more complex functions. While the correct values of GP hyperparameters are important in modeling the entire field correctly, that is not the case if we just have to monitor

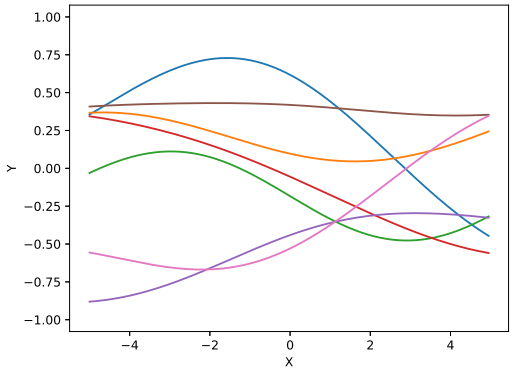
a few specific locations, *i.e.,* a hotspot in this work. Eliminating the need to optimize hyperparameters at each step by using AdaptGP-MCTS alleviates the cubic complexity of the hyperparameter optimization. AdaptGP-MCTS starts with an initial σ_0 and l_0 of the GP hyperparameters. We provide more description on the choosing the σ_0 and l_0 in the section where we discuss our experimental results.

As AdaptGP-MCTS collects more measurements during the procedure, we monotonically decrease the length scale and increase the signal variance in Line 7 of the Algorithm 5. The new values of hyperparameters are used to get the mean and variance estimate in the next iteration in Line 3. In Line 5, we collect the measurement at location x_t. This measurement is corrupted by the sensor noise ϵ which is modeled as a standard normal distribution with mean zero mean and ω^2 variance. The sensor noise variance ω^2 is assumed to be known *a priori*. Once the operating budget is exhausted, at the end we do a full GP hyperparameter optimization (Line 8) using the measurements collected previously in Line 6. Finally, the location of the predicted maxima is reported (Line 10) where the posterior mean attains its maximum value.

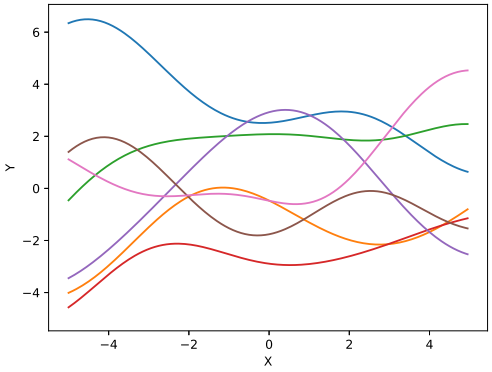
Now we discuss the planner which is based on the idea of MCTS and uses GP-UCB values as the reward heuristics to balance the the exploration-exploitation trade-off.

3.4.1 Planner

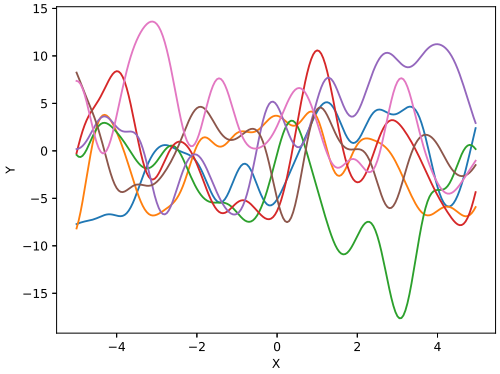
The pseudocode for the planner is given in the Algorithm 6. In the Backprop-



(a) $l = 5$ and $\sigma = 0.5$.



(b) $l = 2.5$ and $\sigma = 2.5$.



(c) $l = 0.5$ and $\sigma = 5$.

Figure 3.1: GP samples from a squared exponential kernel with various values of hyperparameters. Notice the difference in the range of the functions on y-axis. l controls the overall smoothness of the functions and σ controls the range of values the sampled function can take.

Algorithm 6 GP-MCTS

Input: $\hat{\mu}_t(x), \hat{\sigma}_t(x), t$.
while *within budget*
  $v' \leftarrow$ Selection(v)
  $v_{new} \leftarrow$ Expansion(v')
  $r_\mu + \beta^{1/2} r_\sigma \leftarrow$ Simulation(v_{new})
  Backpropagation $\left(v_{new}, r_\mu + \beta^{1/2} r_\sigma\right)$
end procedure
function Selection(v)
  while v is fully expanded
    $v \leftarrow \text{argmax}_{child \in v.children} \frac{Q(child)}{n_{child}} + 2\sqrt{\frac{\log(n_v)}{n_{child}}}$
  return v

agation step, we use the GP-UCB values to update the values for ancestral nodes. For the hotspot identification problem considered in this chapter, for reward calculation, we use a root squared growth of $\beta^{1/2}$ in terms of the number of measurements collected:

1. Mean: To encourage the exploitation, *i.e.,* $r_\mu = \hat{\mu}_t(x)$,
2. Variance: To encourage the exploration, *i.e.,* $r_\sigma = \hat{\sigma}_t(x)$.

The reward function is the summation of these two values. Now we experimentally validate our algorithm on a real-world dataset and a carefully designed synthetic spatial field. Next, we present our empirical results staring with the case where the GP hyperparameters are assumed to be known. We call this strategy TrueGP-MCTS.

3.5 Empirical Evaluation

In our application, the locations inside the environment are the search tree nodes. At a given location, the robot has five motion primitives that serve as the

five children of the current node. The maximum number of iterations to build the MCTS was set to 50. A random policy was used for performing rollouts to backpropagate the average GP-UCB values as the rewards. For rollouts, we do not fix the number of simulation steps. Instead, we do the simulations for the remaining time budget at the current instance minus the depth of the expanded node from the root node. This captures the intuition that in the beginning, the robot should explore more often which is encouraged by the higher number of simulation steps. However, as the robot comes closer to finish the mission, it has learned a good estimate of the underlying environment and will not benefit from higher number of simulation steps [128]. An instance of an MCTS tree for a robot is shown in Figure 3.2. The green arrows represent the entire tree and the blue arrows represent the best trajectory based on this built tree. The blue path shows the the robot path until that moment in time and background heatmap represents the learned GP mean by the robot of the underlying spatial field. For the Expansion Step in Algorithm 6 (Line 4), we expand randomly on any of the unvisited child.

We perform the empirical evaluation of the algorithms presented. One of the most important features of our algorithm is that it does not estimate the hyperparameters at each iteration by running the computationally expensive optimization of the marginal likelihood. Hence, we provide a quantitative comparison of our algorithms with the strategies that optimize the hyperparameters adaptively at each time step or use the true underlying hyperparameters. The experimental results suggest that one would not lose much by not optimizing hyperparameters at each time step but gain a significant advantage in computational time. Our experiments

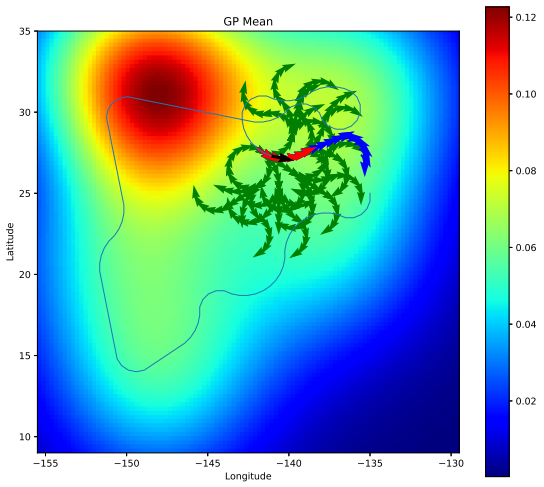


Figure 3.2: The robot has five motion primitives.

on the real-world dataset are on Chlorophyll density data collected from the a square subregion in Pacific ocean spanning the geographical coordinates, longitude expansion from -155.5 to -129.5 and latitude expansion from 9 to 35. Hence, we construct an synthetic spatial field over the same lon/lat coordinates and start by doing a study on this synthetic spatial field.

3.5.1 Synthetic Field

We construct a complex synthetic spatial field (Figure 3.3) over the given lon/lat coordinates. This spatial field has four locations of maxima, three of which are local maxima. For our experiments, we start the robot near lower left corner from (-149, 16) so as to trick it into collecting measurements and spending time near one of the local maxima. The actual hotspot is located near the top right corner at (-135.6, 29) where the field attains a maximum value of 1. The lowest field

value is 0. We estimated the hyperparameters *apriori* using a 30×30 grid on this

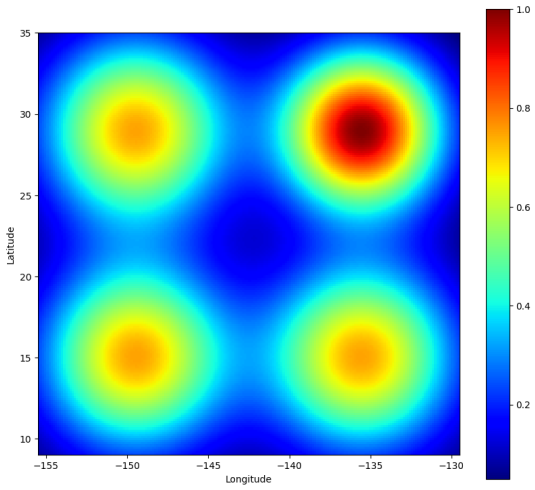


Figure 3.3: The environment has four locations of maxima, three of which are local maxima.

field and minimizing the negative log marginal likelihood of the values at those grid locations. The GP squared-exponential hyperparameters $\sigma_0, l1, l2, \omega^2$ for this field were estimated to be $0.251, 5.04, 5.04, 10^{-5}$ respectively.

We studied the effect of the sensor noise on the performance of the algorithms. The sensor values are simulated as a normal distribution with mean as the actual value at the measurement location. The sensor noise standard deviation was varied to take values 0.01, 0.05, 0.1, and 0.30 (1, 5, 10, and 30% respectively of the spatial field range) respectively, *i.e.,* from using a very accurate sensor to a very bad sensor. In all experiments, the robots plan the path using an MCTS planner with GP-UCB values as the node rewards where the GP variance was multiplied with the square root of $2\sqrt{t}\log\left(\frac{|D|\pi^2}{6\delta}\right)$ as β_t (termed as GP-UCBE in the plots). Here, $|D|$ denotes the number of grid locations used for estimating the GP mean and variance. We

used a grid of resolution 130×130. Hence, $|D|$ is 16900 in our case and we choose δ to be equal to 0.1 [51]. For each noise standard deviation, we run ten trials for the robot that starts from (-149, 16). We compare the performance of TrueGP-MCTS planner with a Boustrophedon (BST) path.

Table 3.1 shows the average mission Percent Terminal Regret, Percent Average Cumulative Regret, Percent Root Mean Squared Error (RMSE) all with respect to the range of the spatial field (*i.e.,* 1), Percent Distance with respect to maximum possible distance between any two points in the environment (*i.e.,* diagonally opposite locations) for four noise scenarios. For each noise case, we ran ten missions with standard deviation shown over those ten missions. The TrueGP-MCTS outperforms the BST on all metrics. We can see that for low-noise scenarios, BST tends to have a higher standard deviation which can be attributed to the fact that depending on the type of BST pattern (horizontal or vertical), the robot may get lucky sometimes and find the hotspot very quickly. On the other extreme, it may get really unlucky and not find the hotspot for a really long time. However, TrueGP-MCTS tends to uniformly explore the environment and find the hotspot.

For a more exhaustive comparison, Figures 3.4, 3.5, 3.6, and 3.15 show the Percent Terminal Regret, Percent Average Cumulative Regret, Percent Root Mean Squared Error (RMSE) all with respect to the range of the spatial field values, Percent Distance with respect to maximum possible distance between any two points in the environment (*i.e.,* diagonally opposite locations) for four noise scenarios. We can see that in the beginning for the low to moderate noise cases, BST and TrueGP-MCTS have almost same performance in terms of the terminal regret and

Table 3.1: Mission average percentage values of four metrics. Time budget is 350 units. The first subcolumn in each column corresponds to the BST pattern and the second to TrueGP-MCTS.

	BST	TrueGP-MCTS
	1% Noise	
Terminal Regret	12.5072 ± 3.6851	5.7712 ± 1.8836
Avg Cumulative Regret	63.7503 ± 0.6186	57.5540 ± 1.7510
RMSE	11.7810 ± 3.3208	6.7574 ± 0.7483
Distance	23.2970 ± 6.7337	9.7482 ± 3.7384
	5% Noise	
Terminal Regret	11.7130 ± 4.8586	5.3964 ± 2.1146
Avg Cumulative Regret	63.6402 ± 0.7974	54.8814 ± 1.9327
RMSE	11.9767 ± 4.2813	8.2699 ± 1.0573
Distance	19.7206 ± 9.3801	9.7927 ± 4.8182
	10% Noise	
Terminal Regret	15.5591 ± 4.7759	11.4409 ± 3.4391
Avg Cumulative Regret	63.7503 ± 0.6186	59.5152 ± 2.1726
RMSE	14.5763 ± 2.9520	11.4094 ± 1.0819
Distance	25.9312 ± 9.3238	18.2340 ± 7.5541
	30% Noise	
Terminal Regret	21.2261 ± 4.6456	15.1915 ± 6.8276
Avg Cumulative Regret	63.8433 ± 0.5179	58.5647 ± 2.4262
RMSE	18.4003 ± 1.7002	16.9728 ± 2.7193
Distance	33.8347 ± 9.8928	15.4725 ± 10.3717

the distance. However, with medium budget the TrueGP-MCTS explores the environment efficiently and converges quickly to report the hotspot location. We also observe that for a high-noise setting (Figure 3.7), both algorithms have a lower performance and high variation in the performance among different trials. We also notice that as the sensor noise increases it takes longer for the TrueGP-MCTS to find the hotspot.

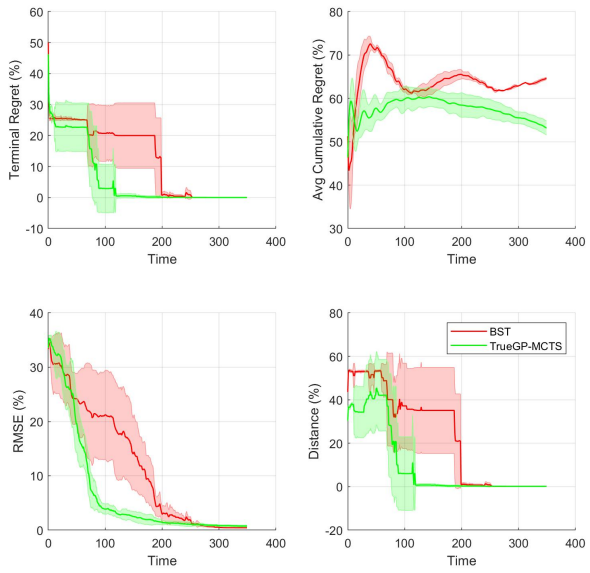


Figure 3.4: The sensor noise standard deviation was set to 1% of the spatial field values range.

Next, we empirically evaluate our algorithm on a synthetic spatial field defined over the an environment with the same geo-coordinates.

3.5.2 Chlorophyll Dataset

We evaluate the performance of our algorithms on a real-world dataset of Chlorophyll concentration measured on Oct 8, 2021 obtained from NASA Earth

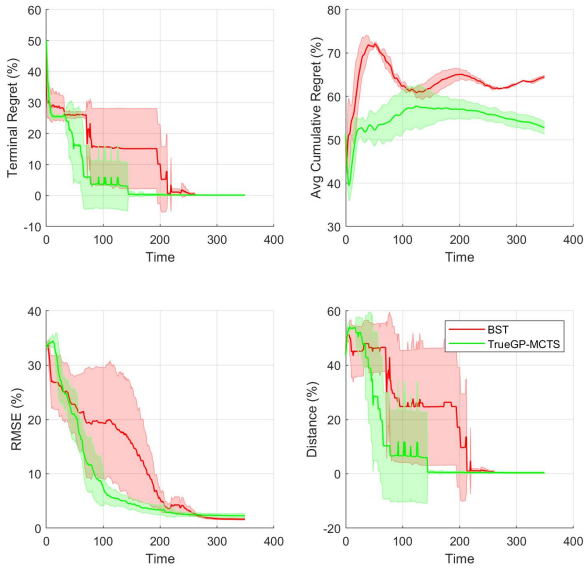


Figure 3.5: The sensor noise standard deviation was set to 5%.

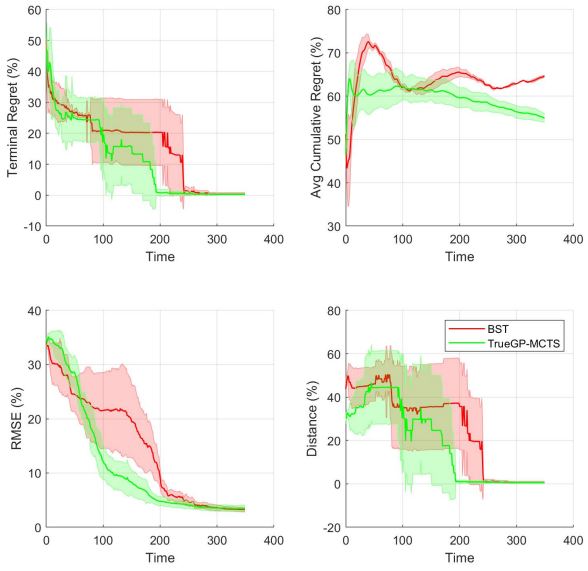


Figure 3.6: The sensor noise standard deviation was set to 10%.

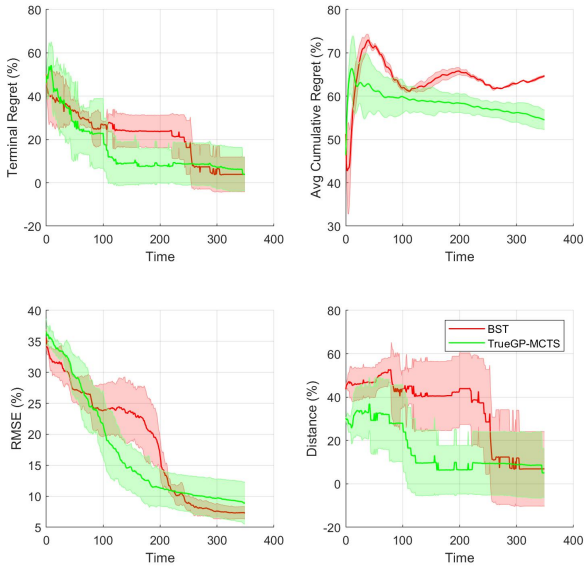
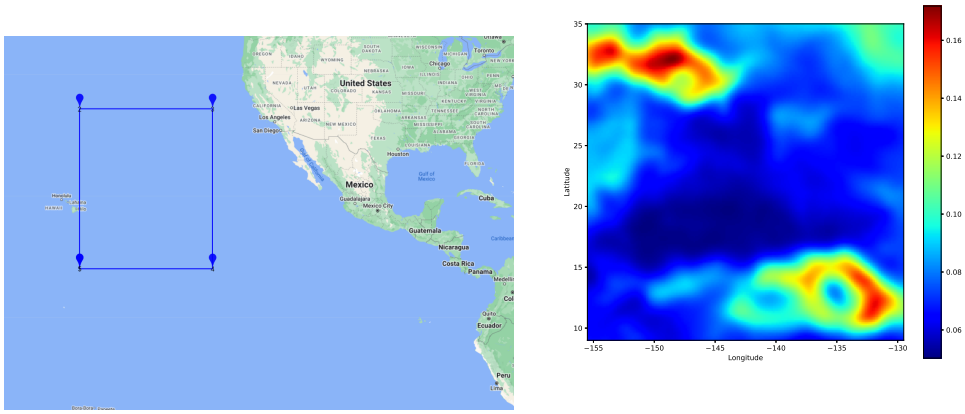


Figure 3.7: The sensor noise standard deviation was set to 30%.

Observations from a Pacific ocean subregion shown in Figure 3.8(a). The actual Chlorophyll concentration (mg/m^3) is shown in Figure 3.8(b). The data collected is from the a square region spanning the geographical coordinates, longitude expansion from -155.5 to -129.5 and latitude expansion from 9 to 35 (Figure 3.8(a)) at 0.5 degree geo-coordinate grid resolution. To query a value at any non-grid location, we used a radial basis function for interpolating and assume that the interpolated values are the true values at that non-grid location.

The hotspot is located at (-148.67, 32.11) where the Chlorophyll density attains the maximum value equal to 0.17 mg/m^3 and lowest density value is 0.05 mg/m^3. We estimated the hyperparameters *apriori* using a 30×30 grid on this field and minimizing the negative log marginal likelihood of the values at those grid locations. The GP squared-exponential hyperparameters $\sigma_0, l1, l2, \omega^2$ for this field were estimated to be $0.0483, 2.33, 1.99, 10^{-5}$ respectively.



(a) Geographical Subregion (b) Chlorophyll Concentration (mg/m^3)

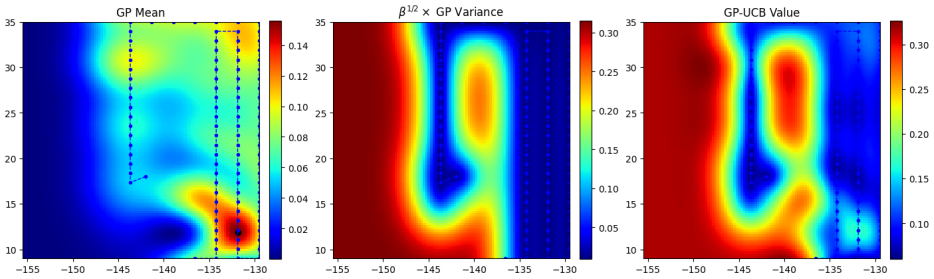
Figure 3.8: The environment has longitude expansion from -155.5 to -129.5 and latitude expansion from 9 to 35.

Here also, we studied the effect of the sensor noise on the performance of the algorithms. The sensor values are simulated as a normal distribution with mean as the actual value at the measurement location. The sensor noise standard deviation was varied to take values 0.0012, 0.006, 0.012, and 0.036 (1, 5, 10, and 30% respectively of the spatial field range) respectively, *i.e.,* from using a very accurate sensor to a very bad sensor. In all experiments, the robots plan the path using an MCTS planner with GP-UCB values as the node rewards where the GP variance was multiplied with the square root of $2\sqrt{t}\log\left(\frac{|D|\pi^2}{6\delta}\right)$ as β_t (termed as GP-UCBE in the plots). Here, $|D|$ denotes the number of grid locations used for estimating the GP mean and variance. We used a grid of resolution 130×130. Hence, $|D|$ is 16900 in our case and we choose δ to be equal to 0.1 [51].

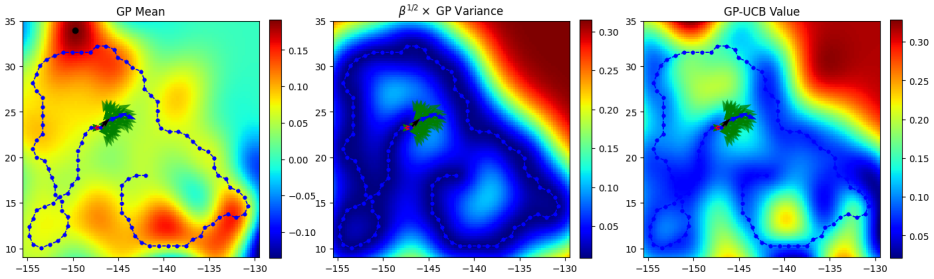
For each noise standard deviation, we run ten trials for the robot that starts from (-142, 18). This starting location was chosen keeping in mind that it is closer to the local maxima in the bottom right corner and hence more likely to trick the robot

from identifying the actual hotspot. We compare the performance of TrueGP-MCTS planner with a Boustrophedon (BST) path. For a qualitative analysis, we observe the path of the robot in a low-noise setting (5%) for both strategies after the robot has spent 100, 200, and 300 units of time in the environment in Figures 3.9, 3.10, and 3.11 respectively. The leftmost subfigures in each figure shows the learned GP mean μ_t as the heatmap. The black bubble in each leftmost subfigure denotes the location of the GP mean maxima. This is the location we report as the hotspot location. The middle subfigure shows the $\beta^{1/2}\sigma_t$, where σ_t is the GP variance at the given time. The right subcolumn shows the summation of both quantities, *i.e.,* GP-UCBE value. We can see from Figure 3.10 that the TrueGP-MCTS planner finds the hotspot quickly and collects more samples there but also explores other regions after it has collected the measurements near the actual hotspot. Even though it collects measurements at other locations, the reported hotspot location does not change. However, the Boustrophedon path finds the hotspot after the robot has spent approximately 300 units of time in the environment. Although, a BST pattern may get really lucky a few times and find the hotspot relatively quickly. Hence, we did multiple runs to observe the average performance of the BST and TrueGP-MCTS planner.

Table 3.2 shows the average mission Percent Terminal Regret, Percent Average Cumulative Regret, Percent Root Mean Squared Error (RMSE) all with respect to the range of the spatial field (*i.e.,* 1), Percent Distance with respect to maximum possible distance between any two points in the environment (*i.e.,* diagonally opposite locations) for four noise scenarios. For each noise case, we ran ten missions with

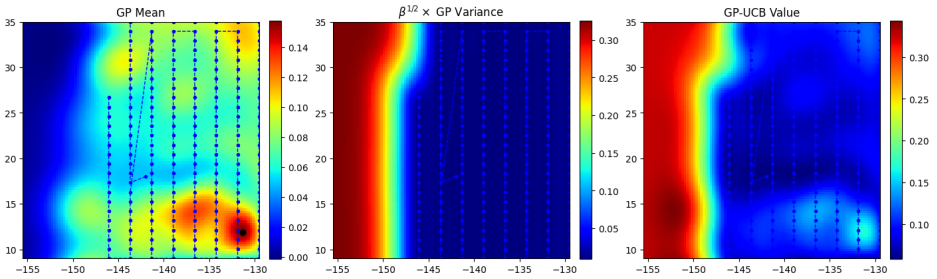


(a) Boustrophedon path after the robot has spent 100 units of time.

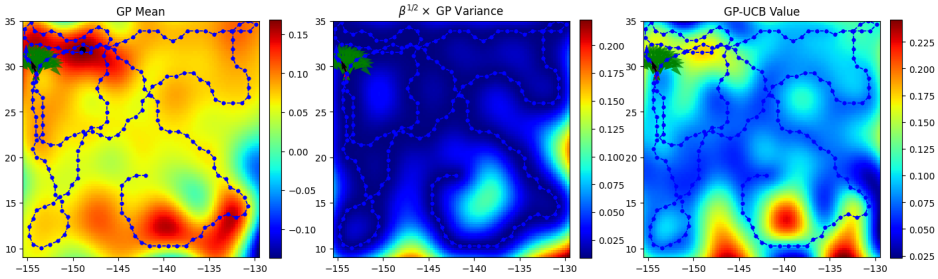


(b) TrueGP-MCTS path after the robot has spent 100 units of time.

Figure 3.9: Boustrophedon vs TrueGP-MCTS comparison at time 100.

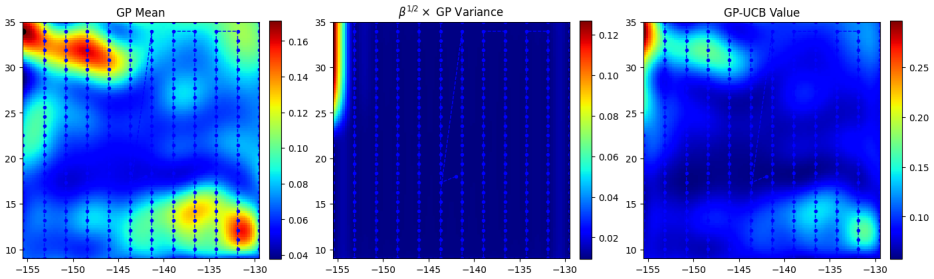


(a) Boustrophedon path after the robot has spent 200 units of time.

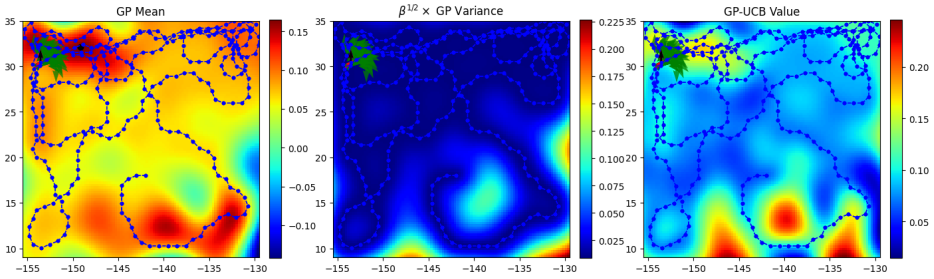


(b) TrueGP-MCTS path after the robot has spent 200 units of time.

Figure 3.10: Boustrophedon vs TrueGP-MCTS comparison at time 200.



(a) Boustrophedon path after the robot has spent 300 units of time.



(b) TrueGP-MCTS path after the robot has spent 300 units of time.

Figure 3.11: Boustrophedon vs TrueGP-MCTS comparison at time 300.

Table 3.2: Average percentage values of four metrics. Time budget is 350 units. The first subcolumn in each column corresponds to the Boustrophedon pattern and the second to TrueGP-MCTS.

	1% Noise		5% Noise	
	BST	TrueGP-MCTS	BST	TrueGP-MCTS
Terminal Regret	34.99 ± 5.46	27.50 ± 6.15	32.21 ± 16.38	26.37 ± 7.76
Avg Cumulative Regret	77.56 ± 1.04	71.44 ± 3.32	76.52 ± 1.01	68.1 ± 1.91
RMSE	28.37 ± 5.86	36.01 ± 6.55	26.59 ± 5.16	20.58 ± 1.08
Distance	35.51 ± 11.57	13.39 ± 6.18	34.54 ± 12.14	16.84 ± 8.57
	10% Noise		30% Noise	
Terminal Regret	23.68 ± 11.92	22.36 ± 6.67	39.82 ± 13.42	27.39 ± 9.57
Avg Cumulative Regret	69.47 ± 0.98	70.63 ± 1.69	77.56 ± 1.04	62.78 ± 6.77
RMSE	28.38 ± 4.54	19.8 ± 2.8	32.59 ± 4.5	33.29 ± 6.26
Distance	31.13 ± 20.17	18.7 ± 8.7	48.50 ± 21.72	33.31 ± 15.7

standard deviation shown over those ten missions. The TrueGP-MCTS outperforms the BST in terms of the percentage distance and the terminal regret for each noise case. The TrueGP-MCTS planner outperforms Boustrophedon on Terminal Regret, RMSE, and Distance and comparably on Cumulative Regret. The TrueGP-MCTS outperforms the BST path and keeps accumulating the cumulative regret by continuously exploring the environment even though it has already found the hotspot. Hence, while it might not be always traveling in the high-value regions (resulting in a higher cumulative regret) but its GP-Mean estimate still has the maxima aligned with the actual hotspot location.

For a more exhaustive comparison, Figures 3.12, 3.13, 3.14, and 3.15 show the Percent Terminal Regret, Percent Average Cumulative Regret, Percent Root Mean Squared Error (RMSE) all with respect to the range of the spatial field values, Percent Distance with respect to maximum possible distance between any two points in the environment (*i.e.,* diagonally opposite locations) for four noise scenar-

ios. We can see that in the beginning for the low to moderate noise cases, BST and TrueGP-MCTS have almost same performance in terms of the terminal regret and the distance. However, with medium budget the TrueGP-MCTS explores the environment efficiently and converges quickly to report the hotspot location. We also observe that for a high-noise setting (Figure 3.7), both algorithms have a lower performance and high variation in the performance among different trials. We also notice that as the sensor noise increases it takes longer for the TrueGP-MCTS to find the hotspot.

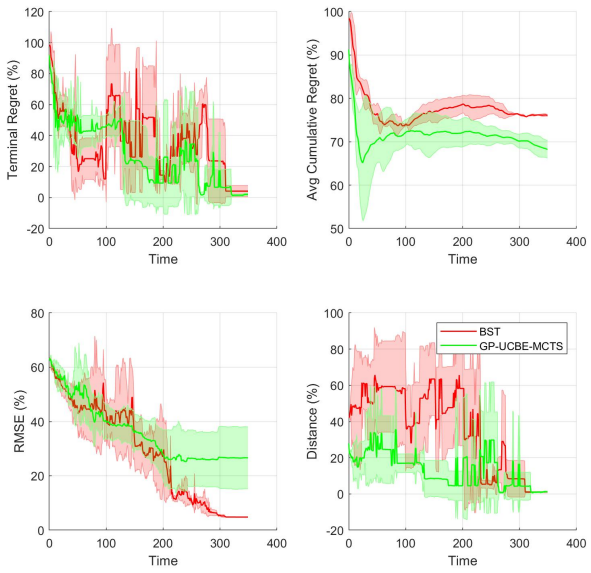


Figure 3.12: The sensor noise standard deviation was set to 1% of the Chlorophyll values range.

3.6 Unknown GP Hyperparameters

In many practical applications, one may not have any prior data from the underlying environment and henco no knowledge of true GP hyperparameters. We

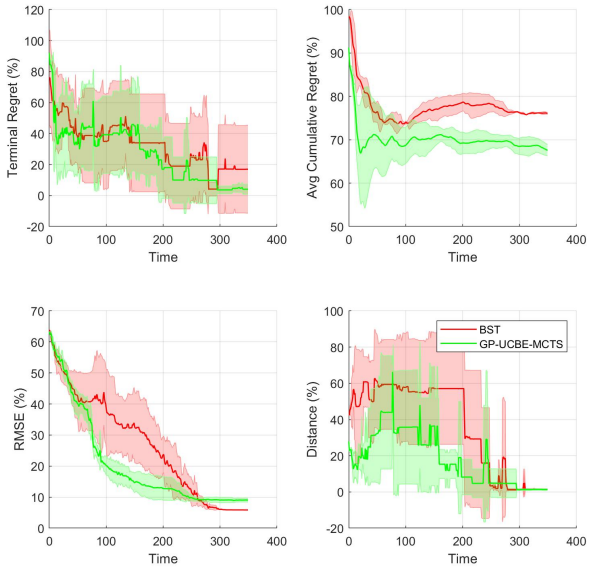


Figure 3.13: The sensor noise standard deviation was set to 5%.

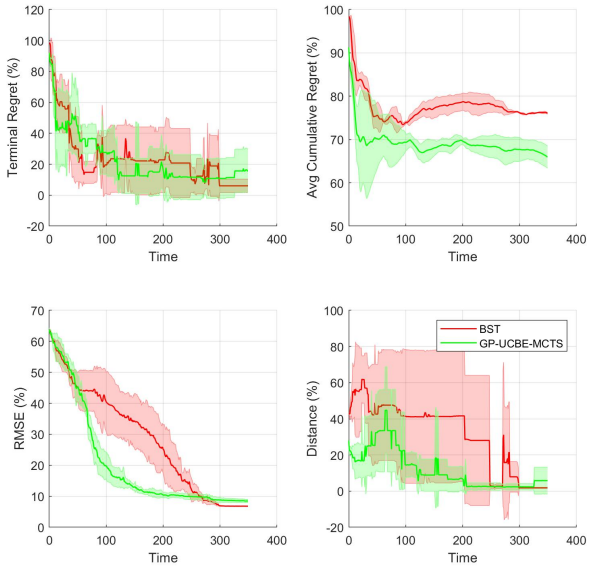


Figure 3.14: The sensor noise standard deviation was set to 10%.

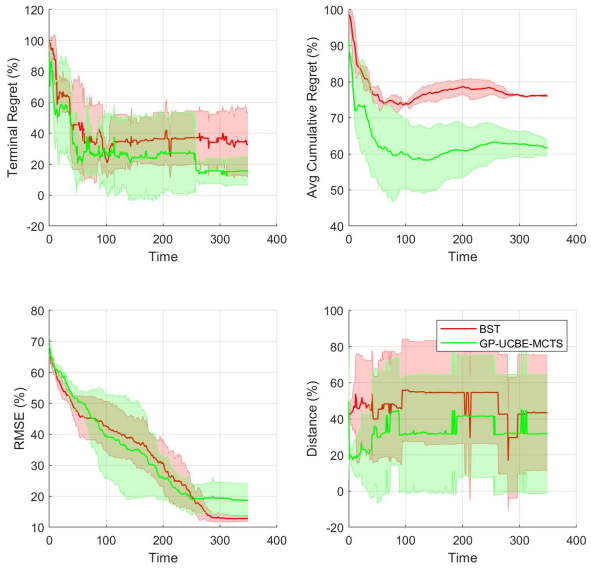


Figure 3.15: The sensor noise standard deviation was set to 30%.

propose a computationally efficient algorithm AdaptGP-MCTS and compare its performace with the case if the hyperparameters are known or if they are optimized at every timestep.

3.6.1 Adaptive planning with AdaptGP-MCTS

We did the experiments with a single robot on the synthetic spatial field and observe the performance of AdaptGP-MCTS with OptGP-MCTS that optimizes for the hyperparameters at each timestep and TrueGP-MCTS as well for a low time budget, medium, and high time budget. Table 3.3 shows the average mission Percent Terminal Regret, Percent Average Cumulative Regret, Percent Root Mean Squared Error (RMSE) all with respect to the range of the spatial field (*i.e.,* 1), Percent Distance with respect to maximum possible distance between any two points in the environment (*i.e.,* diagonally opposite locations) for 5% noise. We ran ten missions

with standard deviation shown over those ten missions. In the beginning, TrueGP-MCTS outperforms other two but only a slight advantage over OptGP-MCTS. There is a large variation in the OptGP-MCTS performance in the beginning. This can be attributed to the fact that the hyperparameters for the OptGP-MCTS depends on the path itself which can be different in each mission and hence more variation. Hence, if one has low operating time budget and does not know the underlying hyperparameters, they can use OptGP-MCTS. Figure 3.16 shows the cumulative GP operations time as a function of the operating budget for three algorithms. We observe that TrueGP-MCTS and AdaptGP-MCTS have almost same computation time but OptGP-MCTS complexity increases significantly. However, as the robot spends more time in the environment, the AdaptGP-MCTS catches up and the performance difference diminishes to less than 3%. Hence, if a user has sufficient time to operate their mission with computational constraints due to the robot hardware, they will be good with using AdaptGP-MCTS.

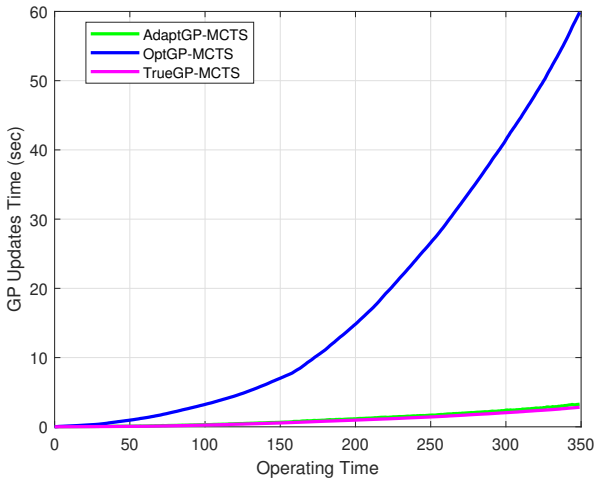


Figure 3.16: The sensor noise standard deviation was set to 5%.

Table 3.3: Average percentage values of four metrics. The first subcolumn in each column corresponds to the AdaptGP-MCTS and the second to OptGP-MCTS and the third subcolumn to TrueGP-MCTS.

	AdaptGP-MCTS	OptGP-MCTS	TrueGP-MCTS
	Time Budget 1-100		
Terminal Regret	58.02 ± 13.10	17.58 ± 8.30	17.15 ± 3.31
Avg Cumulative Regret	59.27 ± 4.62	53.50 ± 2.80	50.70 ± 2.46
RMSE	34.33 ± 4.67	22.19 ± 4.98	21.06 ± 3.73
Distance	48.38 ± 4.13	27.64 ± 16.46	28.88 ± 5.21
	Time Budget 101-200		
Terminal Regret	$18.13.20 \pm 14.51$	2.33 ± 3.34	0.56 ± 0.66
Avg Cumulative Regret	62.58 ± 0.96	59.71 ± 3.12	54.25 ± 3.85
RMSE	8.92 ± 1.00	5.98 ± 1.09	4.43 ± 0.47
Distance	21.23 ± 20.68	4.38 ± 7.20	1.10 ± 1.26
	Time Budget 201-350		
Terminal Regret	3.13 ± 2.07	0.07 ± 0.10	0.17 ± 0.22
Avg Cumulative Regret	62.39 ± 1.14	57.19 ± 1.61	53.88 ± 1.11
RMSE	6.36 ± 0.29	2.62 ± 0.38	2.71 ± 0.24
Distance	3.28 ± 2.84	0.30 ± 0.21	0.44 ± 0.34

3.7 Conclusions

In this chapter, we studied the problem of hotspot identification for various sensor qualities. We also presented an algorithm AdaptGP-MCTS for the unknown GP hyperparameters. Optimizing hyperparameters adaptively is a computationally prohibitive task that scales cubically with the number of measurements [30]. We show that if a user has sufficient mission time but not powerful computing hardware on their robotic system, they can use AdaptGP-MCTS and will not lose much in terms of the performance. For a low to medium mission time, the user should use OptGP-MCTS if they prior hyperparameters of the kernel are not known. We compared TrueGP-MCTS with a Boustrophedon pattern for each sensor type, from a very accurate sensor to a very bad sensor. We also validated their performance

on a real-world dataset of Chlorophyll density.

Chapter 4: Planning for Finding Hotspots in an Environment with Multiple Robots

In the previous chapter, we focused on using a single robot to identify the hotspot in an environment. However, if the environment is large a single robot may not be able to cover the entire environment. For these scenarios, we propose a decentralized multi-robot coordination approach for hotspot identification in unknown environments. As was the case in previous chapters, we use Gaussian Process regression to model the stationary field whose hotspot (i.e., maxima) needs to be found.

The field of coordination of multi-robot systems (MRS) has gained considerable attention in recent years. MRS can reduce the time required to explore an area significantly by distributing the task among several robots. Using a MRS has several advantages as well over a single-robot system. An MRS has a better spatial distribution and can achieve better overall system performance. An MRS is more robust and can benefit from information sharing among the robots. For example, multiple robots can localize themselves more efficiently if they exchange information about their position whenever they sense each other [151, 152]. An MRS can have a lower cost. Using a few simple robots can be simpler to program, cheaper to build

than using a single powerful robot to perform a task.

Our goal is to plan the paths for robots to identify the hotspot quickly. Of course, dealing with MRS presents unique challenges. There are two main challenges that MRS present in comparison to the single-robot case that are particularly relevant for our work:

1. Coordination and communication: It is often desired that the robots can operate in a decentralized fashion when possible. This is especially crucial, for example, in underwater environments where communication is severely limited. We cannot assume continuous access to a reliable communication channel. They can only communicate when coming to the surface and therefore have access to a satellite link [153, 154].
2. Planning: In a multi-robot system, each robot should take into account for the paths of the others in order to avoid any redundancy. Hence, the planner should ensure that the robots are not very close to each other. As such, we need some way of partitioning or covering the environment. However, this partitioning cannot be done naively. For example, if a large part of the environment has low values, then having too many robots covering that area is not useful when the goal is to find the hotspot. Therefore, one would need an efficient way to partition that is specifically tuned for finding hotspots.

We address these challenges by presenting a dynamic partitioning scheme which splits the environment amongst the robots such that no robot is required to cover an especially large portion of the environment. However, instead of parti-

tioning the environment just based on the size, we use the GP estimates and the size of the environment to determine the partitions. Specifically, our partitioning is based on Voronoi tessellation [155] and the Upper Confidence Bound metric [51,97]. This partitioning scheme can work with several planners and finds hotspots efficiently. We also allow for the robots to operate in a decentralized fashion with periodic connectivity for coordination. We propose several planners and compare their performance on a real-world data using terminal regret and average prediction error as the metric. We also compare the performance of our partitioning algorithm with a non-partitioning one.

4.1 Related Work

MRS can do complex tasks and have been widely deployed in many applications such as environmental sampling, coverage, and others, in which the robots employ local communication to achieve some common objective [156–160]. MRS have been actively deployed in number of domains, such as precision agriculture [161,162], and environmental monitoring and exploration [163–165]. One of the major challenges in MRS is dividing the task between robots efficiently especially in practical scenarios when the robots operate in a decentralized manner [166].

Voronoi partitioning is a common approach for multi-robot coordination used in various domains, such as, exploration and mapping with ground vehicles, include spatial partitioning [167–171]. The foundational work on multi-robot Voronoi-based partitioning and coverage was done by Cortes et al. [103]. The key idea in this paper

was to show how Voronoi tessellation can be used to partition the environment amongst a set of robots. In particular, they presented a control law that drives the robots towards the centroid of their Voronoi cells and showed that this converged to equitable partition of the environment.

Building on this work, several algorithms have been proposed over the years. Cortes et al. presented area-constrained locational problems where a group of robots seeks to optimize a notion of environmental coverage by partitioning the space into Voronoi regions [158,172]. They characterized the critical points of this optimization problem as center generalized Voronoi configurations. Martinez et al., focused on the motion coordination and encoding various coordination tasks in MRS [173,174] using Voronoi partitions.

Although the multi-robot coverage problems and its variants have been extensively studied with the optimal solutions, the results are often based on the assumption that the density function is known beforehand [175], which may not be applicable in real-world situations where the robots operate in unknown environments. This motivates the need for taking samples on the fly and efficiently learn the distribution of environmental phenomenon via statistical models such as a GP.

Kemna et. al., used a dynamic Voronoi partitioning approach based on the entropy in a decentralized fashion [176]. They repeatedly calculate weighted Voronoi partitions for the space. Each vehicle then runs informative adaptive sampling within its partition. The vehicles can share information periodically. Wenhao et. al., presented an adaptive sampling algorithm for learning the density function in multi-robot sensor coverage problems using a Mixture of Gaussian Processes models [56].

They also partitioned the environment in Voronoi cells and plan the robot paths to move towards the centroid of their Voronoi cells.

Instead of using entropy as the weight function in estimating Voronoi regions, we use GP-UCB criteria with a squared root growth of β_t term. GP-UCB criterion has been shown more efficient in hotspot identification tasks in comparison to entropy [97]. Further, unlike the work by Wenhao et. al., we use a planner to plan the robot paths in their respective partitions.

4.2 Problem Formulation

In this section, we formally define the multi-robot hotspot identification problem. We assume that the spatial field under consideration defined over a 2-dimensional environment $U \in \mathbb{R}^2$ is an instance of a GP, F. F is defined by a covariance function of the form,

$$C_Z(x, x') = \sigma^2 \exp\left(-\frac{(x - x')^2}{2l^2}\right); \forall x, x' \in U, \tag{4.1}$$

defined by a squared-exponential kernel and the hyperparameters σ^2 and l are not known. We now formally describe the problem and the algorithms proposed in this chapter.

Problem 5 (Multi-robot Hotspot ID)**.** *Given an operating time budget T, plan a set of trajectories under budget T for a set of k mobile robots that obtain measurements from the environment U, and report the location of maxima of the spatial field f at*

the end, i.e.,

$$\textit{minimize} \quad f(x^*) - f(\hat{x}),$$

$$\textit{subject to} \quad \max_{i \in \{1,\ldots,k\}} len(\tau_i) + n_i\eta \leq T.$$

τ_i denotes the tour of the i^{th} robot. Robots travel with unit speed, obtain one measurement in η units of time. Here, let i^{th} robot collect n_i total measurements. $\hat{x}$ is the location of the maxima of the predicted field while x^* is the location of maxima of the true spatial field. We do not know x^* and we also do not know f. We only know the GP prediction $\hat{f}$. The task is to use $\hat{f}$ to be able to predict x^*.

Note that while there may be individual GP models for each robot operating in the environment, there is one combined GP model as well and we will report the maxima of this combined GP model at the end. The combined GP model gets the data from all the robots periodically but this communication can not occur each time the robot gets a new measurement and can only occur after every few measurements.

4.3 Algorithm

Our algorithm uses Voronoi regions for dynamic partitioning after each epoch and uses the single robot TrueGP-MCTS (from Chapter3) as the planner. Voronoi diagram is defined in the following manner:

Definition 1. *Given a set of points $p_1, p_2, \ldots, p_n$ in the plane S, a Voronoi diagram divides the plane S into n Voronoi regions with the following properties [155]:*

- *Each point p_i lies in exactly one region.*

- *If a point $q \in S$ lies in the same region as p_i, then the Euclidian distance from p_i to q will be shorter than the Euclidean distance from p_j to q, where p_j is any other point in S.*

The points $p_1, \ldots, p_n$ are called generator points for the Voronoi partitions. Similarly Voronoi regions partitions be defined for line segments as well where line segments acts as generators. More formally,

Definition 2. *Given a set of line segments $l_1, l_2, \ldots, l_n$ in the plane S, a Voronoi diagram divides the plane into n Voronoi regions with the following properties [155]:*

- *Each line segment l_i lies in exactly one region.*
- *If a point $q \in S$ lies in the same region as l_i, then the distance of the closest point on l_i to q will be shorter than the distance of the closest point on l_j to q, where l_j is any other line segment.*

We use UCB values defined in [51] (denominator in Equation 4.2) as the weights from our GP model to estimate the weighted centroids of a Voronoi cell. Let $(x_1^1, x_2^1), \ldots, (x_1^m, x_2^m)_i$ be the set of m points in i^{th} Voronoi partition. Then its centroid can be calculated as follows,

$$Centroid(Vor_i) = \frac{\sum_{k=1}^{k=m} (x_1^k, x_2^k)_i (\hat{\mu}_t(x_1^k, x_2^k)_i + \beta_t \hat{\sigma}_t(x_1^k, x_2^k)_i)}{\hat{\mu}_t(x_1^k, x_2^k)_i + \beta_t \hat{\sigma}_t(x_1^k, x_2^k)_i}. \tag{4.2}$$

Here, $\hat{\mu}_t(x_1^k, x_2^k)_i$, and $\hat{\sigma}_t(x_1^k, x_2^k)_i$ are the GP mean and variance at location $(x_1^k, x_2^k)_i$ respectively, and β_t is the parameter that controls the exploration-exploitation.

A Voronoi partitioning of a unit square with some random generator points is shown in Figure 4.1. Similarly, the Voronoi partitioning can be defined when the

generators are the line segments. In that case, each point in a particular Voronoi cell is closest to the generator line segment of that partition.

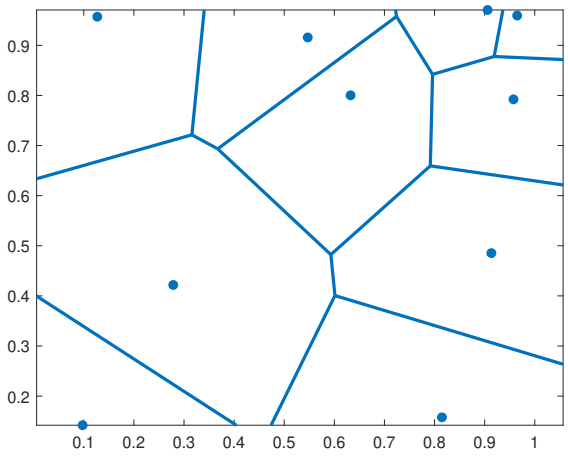


Figure 4.1: Voronoi partitions. The blue bubbles denote the point generators.

Algorithm 7 Voronoi-TrueGP-MCTS

procedure
Input: Hyperparameters σ_0 and l_0, Planner().
for epoch = [1:n]
    Create k individual copies $GP1, GP2, \ldots, GP_k$ of the combined GP model $GP_{combined}$
    Calculate Voronoi regions using robot locations as the point generators
    for t = [1:m]:
        Plan and execute the next robot step computed using Planner() in their respective Voronoi region.
        Collect the next environment samples, $(x_1^t, x_2^t)_1, \ldots, (x_1^t, x_2^t)_k$
        $GP1.update(x_1^t, x_2^t)_1$, $GP2.update(x_1^t, x_2^t)_2, \ldots,$ $GPk.update(x_1^t, y_2^t)_k$
    Combine the samples from all robots and update the combined GP model using all the samples.
end procedure

In Algrothm 5, we assume that the robots run for n epochs where in each epoch each robot takes m steps. Initially, the robots start from predefined starting locations. We calculate the Voronoi regions for these robots using their current

position as generators. In each epoch, we plan the robot trajectories using Planner() in their respective Voronoi region. Note that during the epoch the robots can not share information with each other so the Planner() uses only the local information known to the corresponding robot in planning the trajectory during a particular epoch. The measurement collected is subscripted by the robot index. For example, $(x_1^t, x_2^t)_1, (x_1^t, x_2^t)_2$ denote the measurements collected by Robot 1 and 2 respectively at timestep t. At the end of the epoch, the robots share information with each other and we update the combined GP model $GP_{combined}$ using all the measurements collected cumulatively by every robot. The Voronoi partitions are recomputed using the current robot locations as the point generators (Line 6). If the hyperparameters of the underlying GP are not known, we can use the AdaptGP-MCTS planner for a single robot described in Chapter 3.

4.4 Experiments

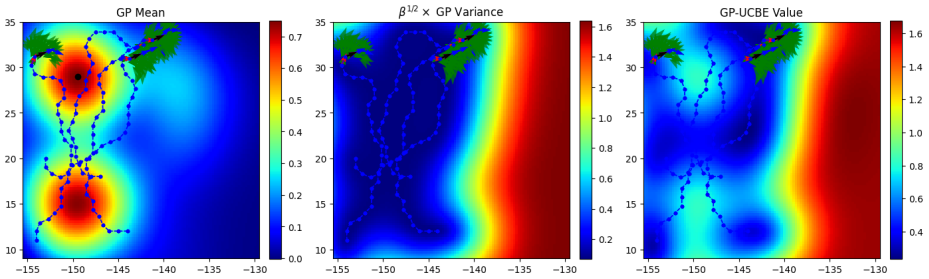
We do the experiments using four robots setting on the synthetic spatial field from Chapter 3 in Figure 3.3.

4.4.1 Synthetic Environment

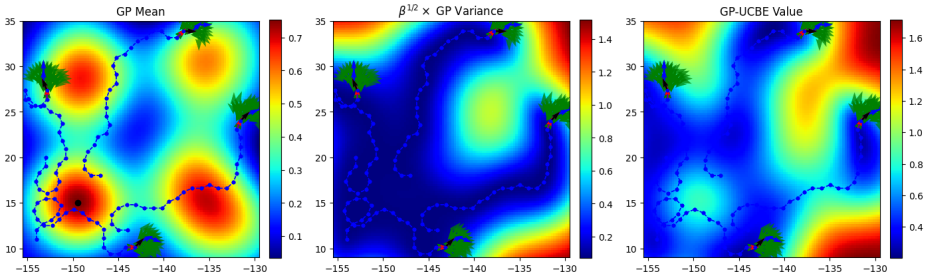
We start the robots near lower left corner so as to trick them into collecting measurements and spending time near one of the maxima. The actual hotspot is located near the top right corner at (-135.6, 29) where the field attains a maximum value of 1. The four robots start from (-147, 18), (-144, 12), (-153, 20), (-154,

11). First we show a qualitative example showing the comparison between using partitioning and not using partitioning. We observe the path of the robots for both strategies after the robot has spent 25, 50, and 75 units of time in the environment in Figures 4.2, 4.3, and 4.4 respectively. Voronoi partitioning achieves much better uniform exploration and reduces the variance in in almost the entire environment by the time the robots have spent 50 units of time (Figure 4.3) while in case of no partitioning, the robots have overlap between the regions they explore and there are still regions in the environment that have high variance.

Table 4.1 shows the average mission Percent Terminal Regret, Percent Average Cumulative Regret, Percent Root Mean Squared Error (RMSE) all with respect to the range of the spatial field (*i.e.,* 1), Percent Distance with respect to maximum possible distance between any two points in the environment (*i.e.,* diagonally opposite locations) for four noise scenarios. For each noise case, we ran ten missions with standard deviation shown over those ten missions. For all the noise values, the two TrueGP-MCTS algorithms (with or wothout partitioning) perform better than the Boustrophedon (BST) pattern. Using Voronoi partitioning provides an advantage in comparison to not using Voronoi partitioning at all. This can be attributed to the fact that with no partitioning, the robots might end up collecting measurements in the same area bringing redundancy. Voronoi partitioning helps in dividing the exploration task efficiently and uniformly between robots and hence has low standard deviation across missions. That is why we also observe the width of the shades narrower for Voronoi partitioning. However, as the sensor becomes bad, we observe that the performance difference between using partitioning and no

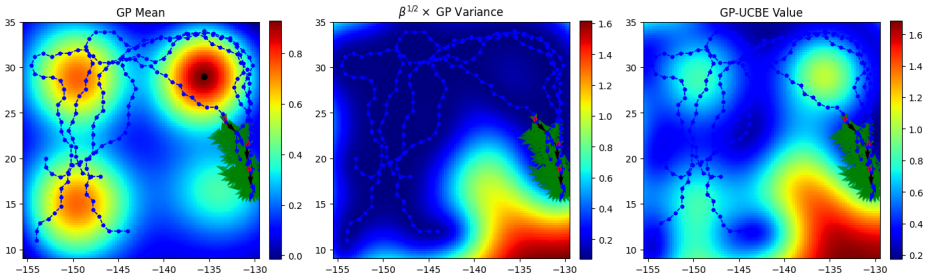


(a) No partitioning path after the robots have spent 25 units of time.

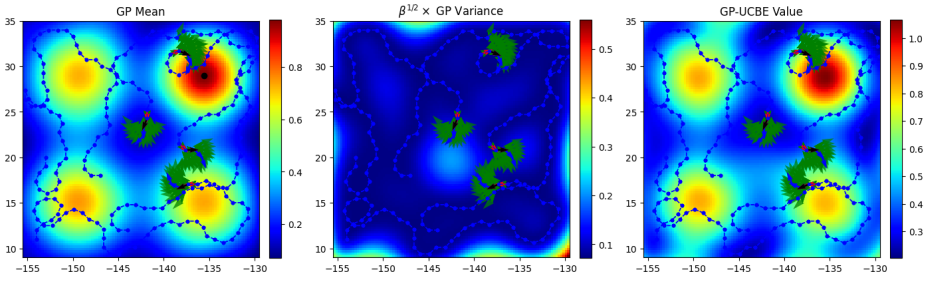


(b) Voronoi partitioning path after the robots have spent 25 units of time.

Figure 4.2: Voronoi partitioning vs No partitioning comparison at time 25.

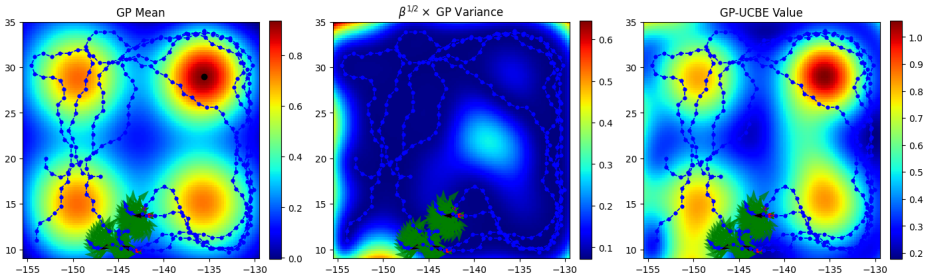


(a) No partitioning path after the robots have spent 50 units of time.

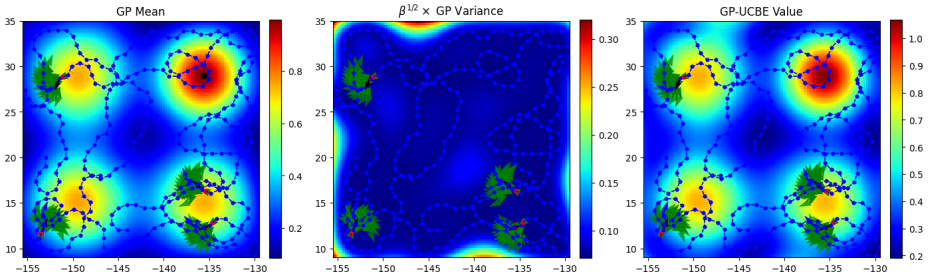


(b) Voronoi partitioning path after the robots have spent 50 units of time.

Figure 4.3: Voronoi partitioning vs No partitioning comparison at time 50.



(a) No partitioning path after the robots have spent 75 units of time.



(b) Voronoi partitioning path after the robots have spent 75 units of time.

Figure 4.4: Voronoi partitioning vs No partitioning comparison at time 75.

partitioning diminishes. This is due to the fact that when the noise is higher we would need more measurement at a particular location to become confident about the prediction. With no partitioning, four robots may go and collect measurement at the hotpot location and become confident very quickly. However, with partitioning, only one robot can collect the measurement at a given location at a given time. Hence, using partitioning will not provide significant advantage if one has a very noisy sensor.

For a more exhaustive comparison, Figure 4.5, 4.6, 4.14, and 4.15 show the Percent Terminal Regret, Percent Average Cumulative Regret, Percent RMSE all with respect to the range of the spatial field values, Percent Distance with respect to maximum possible distance between any two points in the environment (i.e., diagonally opposite locations) for three noise scenarios. For all the noise values, the two algorithms perform better than the Boustrophedon pattern (red plot). Using Voronoi partitioning provides an advantage in comparison to not using Voronoi partitioning at all (Green plot). This can be attributed to the fact that with no partitioning, the robots might end up collecting measurements in the same area bringing redundancy. Voronoi partitioning helps in dividing the exploration task efficiently between robots. That is why we also observe the width of the shades narrower for Voronoi partitioning.

We also compare the Voronoi partitioning and No partitioning in terms of the time taken by them in finding all the hotspots (total 4 for the synthetic field). Table 4.2 shows the earliest time for four robots to detect 1, 2, 3, and 4 hotspots. The Voronoi partitioning achieves better exploration and outperforms No Partitioning

Table 4.1: Average percentage values of four metrics. Time budget is 130 units. The first subcolumn in each column corresponds to the Boustrophedon pattern and the second to No Partitioning and the third subcolumn to Voronoi partitioning.

	BST	No Partition	Voronoi
	1% Noise		
Terminal Regret	37.88 ± 0.53	12.6441 ± 10.7720	7.3440 ± 2.8197
Avg Cumulative Regret	72.4830 ± 1.4048	67.7062 ± 8.2693	72.7852 ± 1.5970
RMSE	15.2483 ± 1.0781	15.3704 ± 8.7345	18.2134 ± 0.6778
Distance	31.1011 ± 0.7507	26.1653 ± 21.3366	16.5787 ± 1.6818
	5% Noise		
Terminal Regret	40.0625 ± 1.8991	8.9615 ± 6.8710	6.5880 ± 2.0447
Avg Cumulative Regret	73.7904 ± 0.7451	63.4555 ± 1.5669	62.1019 ± 0.5412
RMSE	16.4405 ± 1.0196	8.2900 ± 0.5418	6.4329 ± 0.5843
Distance	31.2840 ± 1.0031	15.0227 ± 10.0851	12.4295 ± 3.5505
	10% Noise		
Terminal Regret	44.3363 ± 7.2250	6.3146 ± 1.1188	7.3816 ± 1.6857
Avg Cumulative Regret	72.2413 ± 1.7654	64.7636 ± 1.0904	62.3564 ± 1.0278
RMSE	18.0199 ± 0.8690	9.8770 ± 0.4835	8.1581 ± 0.7311
Distance	30.2471 ± 2.0042	10.6846 ± 2.1218	12.8723 ± 2.9458
	30% Noise		
Terminal Regret	48.7565 ± 1.8755	13.9792 ± 7.4808	12.9057 ± 3.2586
Avg Cumulative Regret	72.6008 ± 1.6435	62.4641 ± 2.0551	62.1961 ± 0.8107
RMSE	20.3622 ± 0.4279	14.6819 ± 1.0296	13.8190 ± 0.6759
Distance	28.7675 ± 1.9315	17.7905 ± 12.5539	18.3297 ± 7.7563

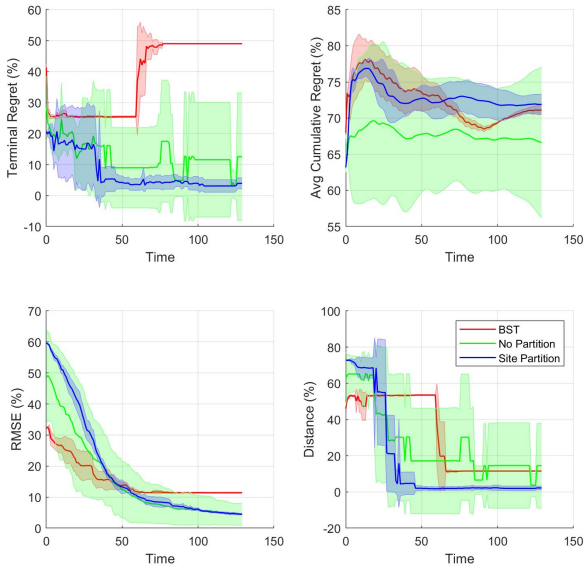


Figure 4.5: The sensor noise standard deviation was set to 1%.

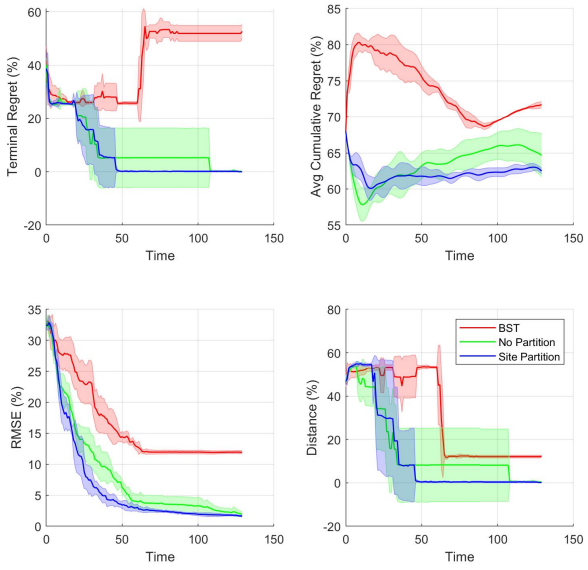


Figure 4.6: The sensor noise standard deviation was set to 5%.

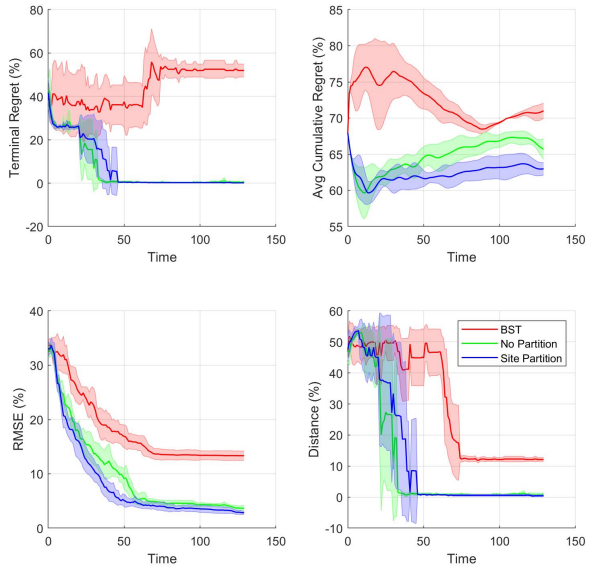


Figure 4.7: The sensor noise standard deviation was set to 10%.

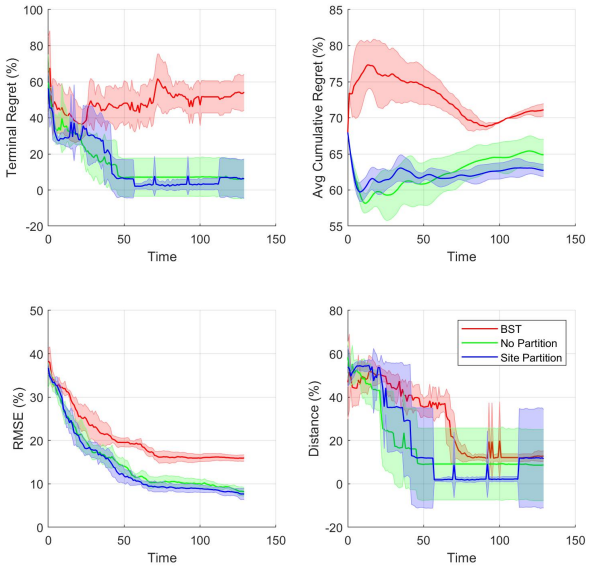


Figure 4.8: The sensor noise standard deviation was set to 30%.

Table 4.2: Averaged over 15 trials for four robots and the noise was set to 5%.

	No Partition	Voronoi
1 Hotspot	3.70 ± 1.15	4.60 ± 1.77
2 Hotspots	25.40 ± 8.09	11.40 ± 5.22
3 Hotspots	37.40 ± 6.96	21.10 ± 5.70
2 Hotspots	55.70 ± 14.10	34.20 ± 9.35

Table 4.3: Averaged over 15 trials for three robots and the noise was set to 5%.

	No Partition	Voronoi
1 Hotspot	4.73 ± 2.15	4.86 ± 1.88
2 Hotspots	31.46 ± 5.73	11.93 ± 4.00
3 Hotspots	50.06 ± 12.69	20.53 ± 8.81
2 Hotspots	71.13 ± 15.00	44.33 ± 9.80

when it comes to finding multiple hotspots. Table 4.3 shows the earliest time for three robots to detect 1, 2, 3, and 4 hotspots. We observed similar trends for three robots as well.

4.4.2 Chlorophyll Dataset

We evaluate the performance of our algorithms on a real-world dataset of Chlorophyll concentration on Oct 8, 2021 obtained from NASA Earth Observations from a Pacific ocean subregion shown in Figure 3.8(a). The actual Chlorophyll concentration (mg/m^3) is shown in Figure 3.8(b). The hotspot is located at (-148.67, 32.11) where the Chlorophyll density attains the maximum value equal to 0.17 mg/m^3 and lowest density value is 0.05 mg/m^3. We estimated the hyperparameters *apriori* using a 30×30 grid on this field and minimizing the negative log marginal likelihood of the values at those grid locations. The GP squared-exponential hyperparameters $\sigma_0, l1, l2, \omega^2$ for this field were estimated to be $0.0483, 3.33, 2.99, 10^{-5}$

respectively.

Here also, we studied the effect of the sensor noise on the performance of the algorithms. The sensor values are simulated as a normal distribution with mean as the actual value at the measurement location. The sensor noise standard deviation was varied to take values 0.0012, 0.006, 0.012, and 0.036 (1, 5, 10, and 30% respectively of the spatial field range) respectively, *i.e.,* from using a very accurate sensor to a very bad sensor. In all experiments, the robots plan the path using an MCTS planner with GP-UCB values as the node rewards where the GP variance was multiplied with the square root of $2\sqrt{t}\log\left(\frac{|D|\pi^2}{6\delta}\right)$ as β_t (termed as GP-UCBE in the plots). Here, $|D|$ denotes the number of grid locations used for estimating the GP mean and variance. We used a grid of resolution 130×130. Hence, $|D|$ is 16900 in our case and we choose δ to be equal to 0.1 [51].

We performed experiments with four robots. In each scenario, we compared the following three algorithms.

1. Boustrophedon (BST): All the robots follow a boustrophedon pattern individually.
2. No partition: The robots can traverse the entire environment at any given time without any constraint of being only in its Voronoi partition.
3. Site partition: The robots are constrained to be collecting measurement only in their Voronoi partitions generated using robot sites at the last surfacing event.

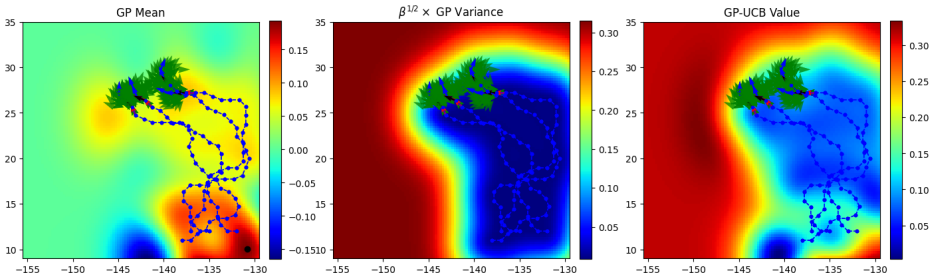
For each noise, we run ten trials for four robots that start with starting loca-

tions (-135, 12), (-132, 12), (-137, 12), (-138, 11). These locations were chosen to be close to the local maxima as to trick the robots to spend more time away from the hotspot location. First we show a qualitative example showing the comparison between using partitioning and not using partitioning. We observe the path of the robots for both strategies after the robot has spent 25, 50, and 75 units of time in the environment in Figures 4.9, 4.10, 4.11 respectively. The leftmost subfigures in each figure shows the learned GP mean μ_t as the heatmap. The black bubble in each leftmost subfigure denotes the location of the GP mean maxima. This is the location we report as the hotspot location. The middle subfigure shows the $\beta^{1/2}\sigma_t$, where σ_t is the GP variance at the given time. The right subcolumn shows the summation of both quantities, *i.e.,* GP-UCBE value. We can see from Figure 4.10 and 4.11 that the partitioning the environment among robots helps in relatively uniform exploration resulting in a lower GP variance. If we do not use the partitioning, the robots tend to collect the measurements in the same subregion of the environment and hence resulting in redundancy.

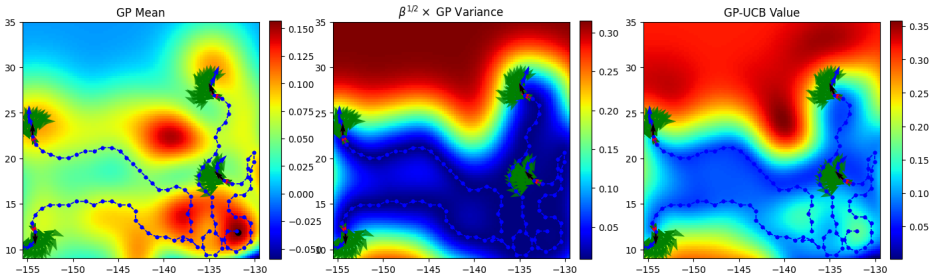
Table 4.4 shows the average mission Percent Terminal Regret, Percent Average Cumulative Regret, Percent Root Mean Squared Error (RMSE) all with respect to the range of the spatial field (*i.e.,* 1), Percent Distance with respect to maximum possible distance between any two points in the environment (*i.e.,* diagonally opposite locations) for four noise scenarios. For each noise case, we ran ten missions with standard deviation shown over those ten missions. Figures 4.12, 4.13, 4.14, and 4.15 show the Percent Terminal Regret, Percent Average Cumulative Regret, Percent RMSE all with respect to the range of the spatial field values, Percent Dis-

Table 4.4: Average percentage values of four metrics. Time budget is 130 units. The first subcolumn in each column corresponds to the Boustrophedon pattern and the second to No Partitioning and the third subcolumn to Voronoi partitioning.

	BST	No Partition	Voronoi
	1% Noise		
Terminal Regret	20.67 ± 0.19	17.57 ± 10.73	8.44 ± 1.67
Avg Cumulative Regret	$75.04 \pm .31$	73.12 ± 1.15	73.68 ± 2.06
RMSE	25.87 ± 1.32	20.83 ± 1.82	18.63 ± 0.82
Distance	63.94 ± 16.98	25.70 ± 12.55	20.45 ± 8.75
	5% Noise		
Terminal Regret	20.20 ± 1.91	14.1 ± 9.43	7.34 ± 2.81
Avg Cumulative Regret	77.71 ± 0.52	72.81 ± 1.60	72.78 ± 1.59
RMSE	40.14 ± 0.86	22.20 ± 1.42	18.21 ± 0.67
Distance	67.54 ± 9.49	30.81 ± 13.71	16.57 ± 1.68
	10% Noise		
Terminal Regret	22.28 ± 3.1	8.36 ± 4.02	8.08 ± 2.86
Avg Cumulative Regret	77.51 ± 0.40	73.29 ± 1.26	71.83 ± 2.07
RMSE	40.97 ± 1.69	21.59 ± 1.46	20.08 ± 1.06
Distance	48.13 ± 22.61	28.62 ± 18.1	34.73 ± 18.12
	30% Noise		
Terminal Regret	36.52 ± 3.74	23.16 ± 8.62	23.14 ± 10.00
Avg Cumulative Regret	77.63 ± 0.37	70.09 ± 2.32	72.71 ± 1.00
RMSE	44.97 ± 1.07	31.68 ± 3.96	28.26 ± 1.65
Distance	28.11 ± 5.12	41.28 ± 29.08	43.02 ± 18.36

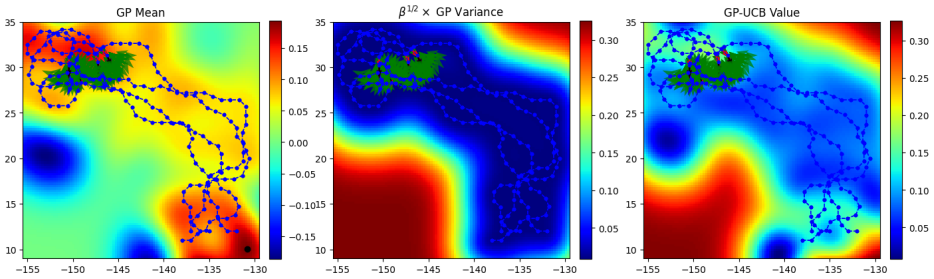


(a) No partitioning path after the robots have spent 25 units of time.

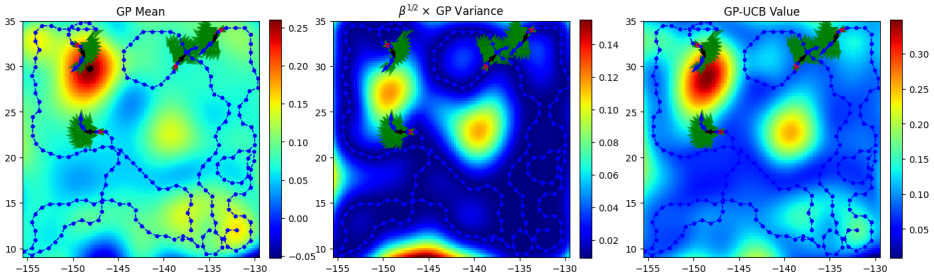


(b) Voronoi partitioning path after the robots have spent 25 units of time.

Figure 4.9: Voronoi partitioning vs No partitioning comparison at time 25.

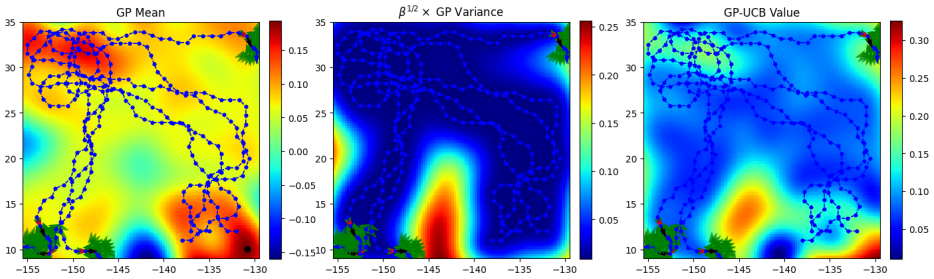


(a) No partitioning path after the robots have spent 50 units of time.

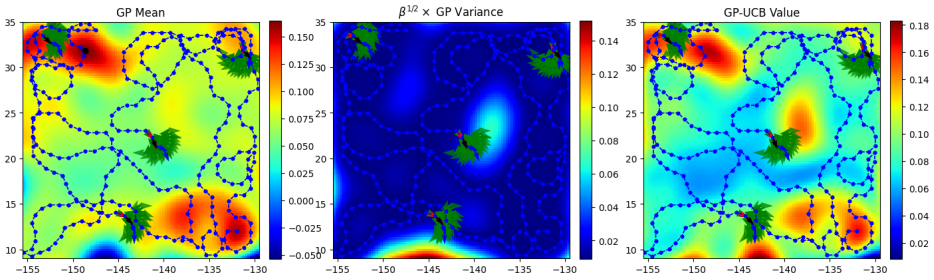


(b) Voronoi partitioning path after the robots have spent 50 units of time.

Figure 4.10: Voronoi partitioning vs No partitioning comparison at time 50.



(a) No partitioning path after the robots have spent 75 units of time.



(b) Voronoi partitioning path after the robots have spent 75 units of time.

Figure 4.11: Voronoi partitioning vs No partitioning comparison at time 75.

tance with respect to maximum possible distance between any two points in the environment (*i.e.,* diagonally opposite locations) for three noise scenarios. For all the noise values, the two algorithms perform better than the Boustrophedon pattern (Red plot). Using Voronoi partitioning provides an advantage in comparison to not using Voronoi partitioning at all (Green plot). This can be attributed to the fact that with no partitioning, the robots might end up collecting measurements in the same area bringing redundancy. Voronoi partitioning helps in dividing the exploration task efficiently between robots. The Voronoi partitioning performs best for low-noise cases. However, the difference between No Partitioning and Voronoi Partitioning diminishes as the noise increases. This can be attributed to the fact that as the noise increases one would require more measurement from the hotspot region to identify it. In case of No Partitioning, all four robots are allowed to collect measurements at any location. However, with partitioning only one robot can collect the measurements at a location at a given time. Hence, if we use the partitioning, it will require more time for the robots to become confident about a certain region.

4.5 Conclusions

In this chapter, we studied the problem of hotspot identification using a decentralized multi-robot system. We focused on primarily two aspects of the problem: Distributing the identification task between robots efficiently and planning the path for those robots. We used a Voronoi region-based partitioning scheme to assign the

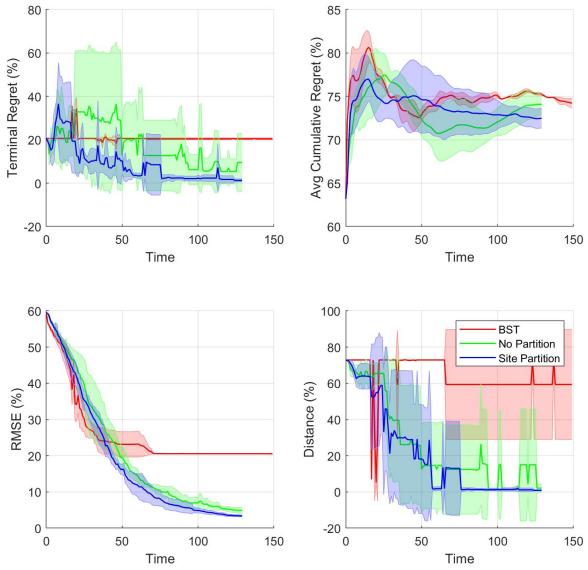


Figure 4.12: The sensor noise standard deviation was set to 1%.

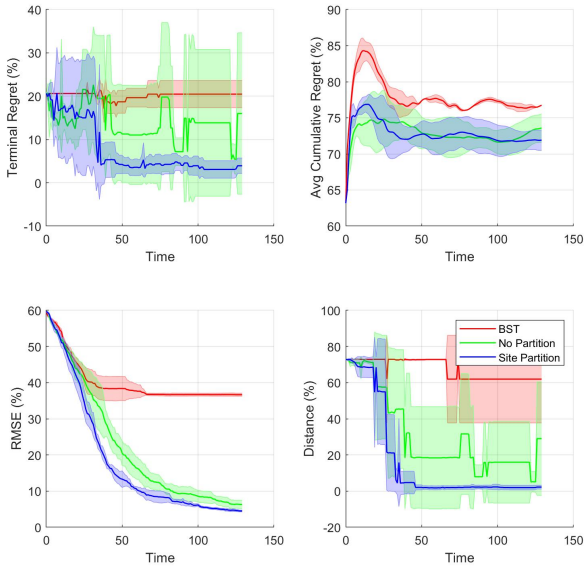


Figure 4.13: The sensor noise standard deviation was set to 5%.

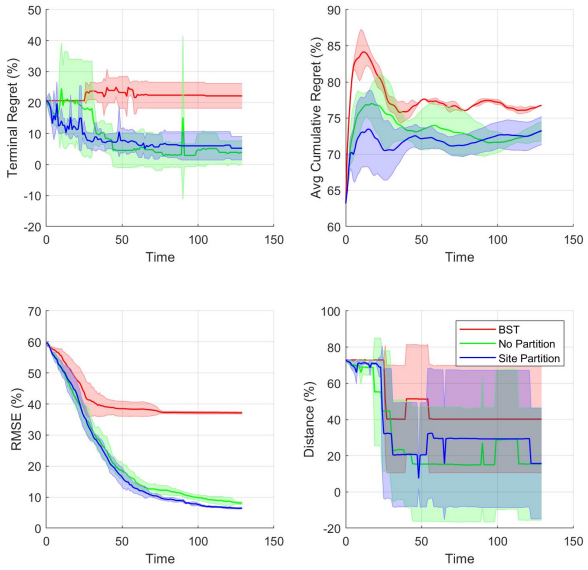


Figure 4.14: The sensor noise standard deviation was set to 10%.

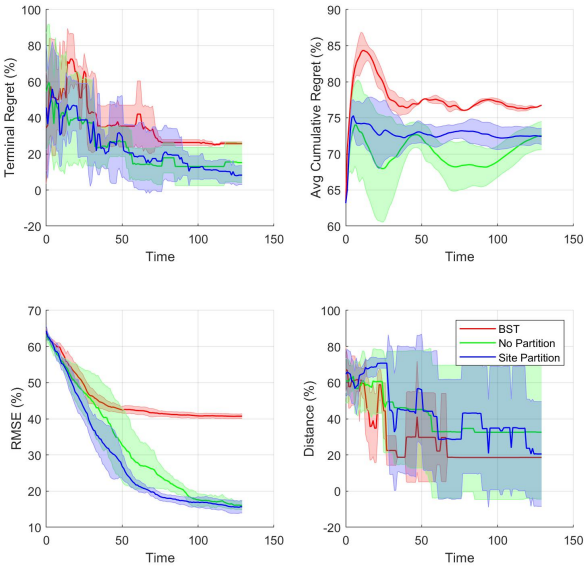


Figure 4.15: The sensor noise standard deviation was set to 30%.

subregions of the environment for individual robots to explore. For planning in these Voronoi partitions, we make use of the UCB values of the learned GP model. This encourages individual robots to to balance the exploration-exploitation trade-off. Further, our partitioning algorithm can be combined with several planners to plan the individual paths for the robots. We also validate our algorithm on a real-world Chlorophyll dataset collected from a subregion of the Pacific ocean.

Chapter 5: Multi-Fidelity Reinforcement Learning for Navigation in an Environment

In previous chapters, we focused on learning an environment or identifying the hotspot in an environment. We showed that when the underlying function is represented by a GP, by effectively choosing where to obtain measurements we can speed up the learning process. In this chapter, we will focus on the same objective but consider a different type of learning problem.

A crucial component of this endeavor is the robot mobility, *i.e.,* how should a robot navigate autonomously if instructed to collect data from an environment. Hence, in this chapter, we focus on efficient ways for autonomous robot navigation through an environment. More specifically, we will focus on Reinforcement Learning (RL) using Gaussian Processes (GPs) as function approximators. As in the previous chapters, we will exploit the structural properties of GPs to efficiently learn the unknown function. The difference is that in the previous chapters we used GPs to model a spatially varying field, here GPs will be used to model the robot dynamics and reward (model-based) or Q-value (model-free) function in the context of RL.

A naive application of RL can be inefficient in real-world scenarios. We present two versions of Multi-Fidelity Reinforcement Learning (MFRL), model-based and

model-free, that leverage GPs to learn the optimal policy in a real-world environment. In the MFRL framework, an agent uses multiple simulators of the real environment to perform actions. With increasing fidelity in a simulator chain, the number of samples used in successively higher simulators can be reduced. We also examine the performance of our algorithms through simulations and through real-world experiments for navigation with a ground robot.

5.1 Related Work

There has been a significant interest in using RL in robotics [3]. While RL has achieved remarkable success in simulation environments its potential in robotics has been quite limited. Not every RL method is equally suitable for each robotics domain [3]. Hester and Stone [177] identify four properties of an RL algorithm that would make it generally applicable to a broad range of robot control tasks:

1. The algorithm must learn from very few samples (which may be expensive and time-consuming).
2. It must learn tasks with continuous state representations.
3. It must learn good policies even with unknown sensor or actuator delays (*i.e.,* selecting an action may not affect the environment instantaneously).
4. It must be computationally efficient enough to take actions continually in real-time.

Using simulators in the RL can alleviate many of the issued mentioned above [41]. The simulators can be used to train the agents, with real-world experience used later

to update the simulator or the agent's policy. The idea of simulator-based RL is closely related to that of using multi-fidelity methods to optimize complex functions in various applications.

Multi-fidelity methods are prominently used in various engineering applications. These methods are used to construct a reliable model of a phenomenon when obtaining direct observations of that phenomenon is expensive. The assumption is that we have access to cheaply obtained, but possibly less accurate observations from an approximation of that phenomenon. Multi-fidelity methods can be used to combine those observations with expensive but accurate observations to construct a model of the underlying phenomenon [16]. For example, learning the dynamics of a robot using real-world observations may cause wear and tear of the hardware [3]. Instead, one can obtain observations from a simulator that uses a, perhaps crude, approximation of the true robot dynamics [22].

Let $f : \mathcal{X} \to \mathcal{Y}$ denote a function that maps the input $\mathbf{x} \in \mathcal{X} \subset \mathbb{R}^d$ to an output $\mathbf{y} \in \mathcal{Y} \subset \mathbb{R}^{d'}$, where $d, d' \in \mathbb{N}$. Multiple approximations are available that estimate the same output with varying accuracy and costs. More generally, K different fidelity approximations, $f^{(1)}, \ldots, f^{(K)}$, are available representing the relationship between the input and the output, $f^{(i)} : \mathcal{X} \to \mathcal{Y}, i = 1, \ldots, K$. Obtaining observations from the i^{th} approximation incurs cost $c^{(i)}$, and typically $c^{(i)} < c^{(j)}$ for $i < j$.

There has been a significant surge in multi-fidelity methods research following the seminal work on autoregressive schemes [178]. They used GPs to explore ways in which runs from several levels of a computer code can be used to make inferences about the output of the complex computer code. A complex code approximates the

reality better, but in extreme cases, a single run of a complex code may take many days. GP framework is a natural candidate to estimate a phenomenon by combining data from various fidelity approximations [179, 180].

The work by Kennedy and O'Hagan [178] focuses only on the scenarios where low-fidelity approximations can capture the right trends, and the low- and high-fidelity approximation outputs exhibit strong linear correlation across the input space. Perdikaris et. al. extended the work to deal with the scenarios where low-fidelity approximations are correlated to their high-fidelity counterparts only in a specific range of inputs [179]. Cutajar et. al. developed a Deep GP [181] extension for multi-fidelity methods [182]. They used the nested structure of GPs in a DGP to propagate information where individual GPs model the transition from one fidelity to the next. There is some work on the use of multi-fidelity methods in Bayesian optimization [183] setting as well. Kandasamy et. al. formalized the notion of regret in multi-fidelity BO setting [184]. Inspired by GP-UCB [51], their algorithm MF-GP-UCB has theoretical guarantees on finding the optimum value at high-fidelity levels. Further, MF-GP-UCB was tested on three hyperparameter tuning tasks and one inference problem in Astrophysics.

Multi-fidelity methods have been widely used in RL applications to automatically optimize the parameters of control policies based on data from simulations and experiments. Transfer learning is a similar approach that aims to generalize between different tasks [22, 185]. Cutler et al. [41] introduced a framework that specifies the rules on when to collect observations from various fidelity simulators. Marco et al. [186] introduced a Bayesian Optimization algorithm which uses entropy [187] as

a metric to decide which simulator to collect the observations from. They used their algorithm for a cart-pole setup with a Simulink model as the simulator.

Two closely related techniques addressing sim2real are domain randomization and domain adaptation. In both cases, the simulators can be controlled via a set of parameters. The underlying hypothesis is that some unknown set of parameters closely matches the real-world conditions. In domain randomization, the simulator parameters are randomly sampled and the agent is trained across all the parameter values. In domain adaptation, the parameter values are updated during learning. However, it is important to note that the goal of these methods is to reduce the reality gap by using a parameterized simulator [40, 188]. This restricts the use of such approaches to scenarios where altering the parameters for a simulator itself is trivial. Further, only one type of simulator is used. In contrast, MFRL techniques leverage multiple simulators with varying fidelity levels as well as cost to operate. Furthermore, in MFRL the policy learned in the highest fidelity simulator (real-world) uses data from the same environment, unlike sim2real. As such, these approaches are beneficial when there is a significant reality gap where maneuvers learned in simulators may not translate to the real world.

5.2 Problem Formulation

We use Reinforcement Learning for formulating our problems.

5.2.1 Reinforcement Learning

RL problems can be formulated as a Markov Decision Process (MDP): $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$; with state space $\mathcal{S}$; action space $\mathcal{A}$; transition function $\mathcal{P}(s_t, a_t, s_{t+1}) \mapsto [0, 1]$; reward function $\mathcal{R}(s_t, a_t) \mapsto \mathbb{R}$ and discount factor $\gamma \in [0, 1)$ [54, 189]. A policy $\pi : \mathcal{S} \to \mathcal{A}$ maps states to actions. Together with the initial state s_0, a policy forms a trajectory $\zeta = \{[s_0, a_0, r_0], [s_1, a_1, r_1], \ldots\}$ where $a_t = \pi(s_t)$. r_t and s_{t+1} are sampled from the reward and transition functions, respectively.

We consider a scenario where the goal is to maximize the infinite horizon discounted reward starting from a state s_0. The value function for a state s_0 is defined as $\mathcal{V}^\pi(s_0) = \mathbb{E}[\sum_{t=0}^{t=\infty} \gamma^t r_t(s_t, a_t) | a_t = \pi(s_t)]$. The state-action value function or Q-value of each state-action pair under policy π is defined as $Q^\pi(s, a) = \mathbb{E}[\sum_{t=0}^{t=\infty} \gamma^t r_{t+1}(s_{t+1}, a_{t+1}) | s_0 = s, a_0 = a]$ which is the expected sum of discounted rewards obtained starting from state s, taking action a and following π thereafter. The optimal Q-value function Q^* for a state-action pair (s, a) satisfies $Q^*(s, a) = \max_\pi Q^\pi(s, a) = \mathcal{V}^*(s)$ and can be written recursively as,

$$Q^*(s_t, a_t) = \mathbb{E}_{s_{t+1}}\big[r(s_t, a_t) + \gamma \mathcal{V}^*(s_{t+1})\big]. \tag{5.1}$$

Our objective is to find the optimal policy $\pi^*(s) = \text{argmax}_a Q^*(s, a)$ when $\mathcal{R}$ and $\mathcal{P}$ are not known to the agent. In model-based approaches, the agent learns $\mathcal{R}$ and $\mathcal{P}$ first and then finds an optimal policy by calculating optimal Q-values from Equation (5.1). The most commonly used model-based approach is Value Iteration

(VI) [190, 191]. We can also directly estimate the optimal Q-values, often known as model-free approaches [112] or directly calculate the optimal policy, often known as policy-gradient approaches [32]. The most commonly used model-free algorithm is Q-learning.

Our multi-fidelity problem can be formally defined as follows: Given a chain of simulators $\Sigma_1, \ldots, \Sigma_d$, minimize the number of learning samples taken in Σ_d.

5.3 Algorithms

In this section, we describe both versions of our algorithm. We compare the proposed algorithms with baseline strategies through simulations. A flow chart of our algorithms is shown in Figure 5.1. We make the following assumptions for both algorithms.

1. The reward function is known to the agent. We make this assumption for the ease of exposition. In general, one can use GPs to estimate the reward function as well. This assumption is required only for GP-VI-MFRL algorithm.
2. State-space in simulator Σ_{i-1} is a subset of the state-space in simulator Σ_i. The many-to-one mapping ρ_i maps states from simulator Σ_i to states in simulator Σ_{i-1}. We give an example of such mapping in subsequent sections. Let ρ_i^{-1} denote the respective inverse mapping (which can be one-to-many) from states in Σ_{i-1} to states in Σ_i. The action space is discrete and the same for all the simulators and the real world.

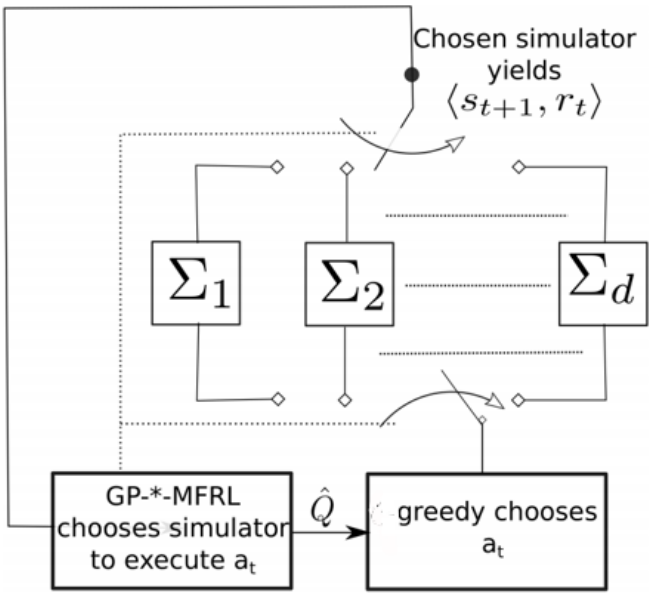


Figure 5.1: The simulators are represented by $\Sigma_1, \Sigma_2, \ldots, \Sigma_d$. The algorithms decide the simulator in which the current action is to be executed by the agent. Also, the action values in the chosen simulator are updated and used to select the best action using the information from higher as well as lower simulators.

5.3.1 GP-VI-MFRL Algorithm

GP-VI-MFRL consists of a model learner and a planner. The model learner learns the transition functions using GP-regression. We use VI [32] as our planner to calculate the optimal policy with learned transition functions. Let $\mathbf{s}_{t+1} = f(\mathbf{s}_t, a_t)$ be the (unknown) transition function that must be learned. We observe transitions: $\mathcal{D} = \{\langle \mathbf{s}_t, a_t, \mathbf{s}_{t+1} \rangle\}$. Our goal is to learn an estimate $\hat{f}(\mathbf{s}, a)$ of $f(\mathbf{s}, a)$. We can then use this estimated $\hat{f}$ for unvisited state-action pairs (in place of f) during VI to learn the optimal policy. For a given state-action pair (s, a), the estimated transition function is defined by a normal distribution with the mean and variance given by Equations (2.1) and (2.2). Algorithm 8 gives the details of the proposed framework.

Algorithm 8 GP-VI-MFRL Algorithm

procedure
Input: confidence parameters σ_{th} and σ_{th}^{sum}; simulator chain $\langle \boldsymbol{\Sigma}$, fidelity parameters $\boldsymbol{\beta}$, state mappings $\boldsymbol{\rho} \rangle$; $\mathcal{L}$.
Initialize: Transition functions $\mathcal{P}_{ss'}^{a}(i)$ and $\mathcal{D}_i$ for $i \in \{1, \ldots, d\}$; *change* = FALSE.
Initialize: $i \leftarrow 1$; $\hat{Q}_i \leftarrow$ PLANNER $(\mathcal{P}_{ss'}^{a}(i))$.
while terminal condition is not met
    $a_t \leftarrow \text{argmax}_a \hat{Q}_i(s, a)$
    if $\sigma_i(s_t, a_t) \leq \sigma_{th}$: *change* = TRUE
    if $\sigma\left(\rho_i(s_t), a_t\right) \geq \sigma_{th}$ and *change* and $i > 1$
        $s_t \leftarrow \rho_i(s_t)$, $i \leftarrow i - 1$, continue
    $\langle s_{t+1}, r_{t+1} \rangle \leftarrow$ execute a_t in Σ_i
    append $\langle s_t, a_t, s_{t+1}, r_t \rangle$ to $\mathcal{D}_i$
    $\mathcal{P}_{ss'}^{a}(i) \leftarrow$ update GP_i using $\mathcal{D}_i$
    $\hat{Q}_i \leftarrow$ call PLANNER with input $\mathcal{P}_{ss'}^{a}(i)$
    $t \leftarrow t + 1$
    if $\sum_{j=t-\mathcal{L}}^{j=t-1} \sigma_i(s_j, a_j) \leq \sigma_{th}^{sum}$ and $t > \mathcal{L}$ and $i < d$
        $s_t \leftarrow \rho_{i+1}^{-1}\left(s_t\right)$, $i \leftarrow i + 1$;
        change = FALSE
end procedure

procedure PLANNER$(\mathcal{P}_{ss'}^{a}(i))$
Initialize: $Q(s, a) = 0$ for each (s, a), $\Delta = \infty$
while $\Delta > 0.1$
    $\Delta \leftarrow 0$
    for every (s, a)
        $temp \leftarrow Q(s, a)$, $Q(s, a) = \hat{Q}_{i-1}\left(\rho_i(s), a\right) + \beta_i$
        for $k \in \{i, \ldots, d\}$
            $s_k = \rho_k^{-1} \ldots \rho_{i+2}^{-1} \rho_{i+1}^{-1}(s)$
            if $\sigma_k(s_k, a) \leq \sigma^{th}$: $\mathcal{P}_{ss'}^{a}(i) = \mathcal{P}_{ss'}^{a}(k)$
        $Q(s, a) \leftarrow \sum_a \sum_{s'} \mathcal{P}_{ss'}^{a}[\mathcal{R}_{ss'}^{a} + \gamma \text{max}_a Q(s', a)]$
        $\Delta \leftarrow \max(\Delta, |temp - Q(s, a)|)$
return $Q(s, a)$
end procedure

Before executing an action, the agent checks (Line 8) if it has a sufficiently accurate estimate of the transition function for the current state-action pair in the previous simulator, Σ_{i-1}. Specifically, we check if the variance of the current state-action pair in previous simulator is less than σ_{th}. If not, and if the transition model in the current environment has changed, it switches to Σ_{i-1} and executes the action in the potentially less expensive environment. The agent lands in the state $\rho_i(s)$ in lower fidelity simulator.

We also keep track of the variance of the $\mathcal{L}$ most recently visited state-action pairs in the current simulator. If the running sum of the variances is below a threshold (Line 15), this suggests that the robot is confident about its actions in the current simulator and can advance to the next one. In the original work [41], the agent switches to the higher fidelity simulator after a certain number of known state-action pairs were encountered. In our implementation (Line 7), the model of current environment changes if the posterior variance for a state-action pair drops below a threshold value (*i.e.,* agent has a sufficiently accurate estimate of the transitions from that state). Lines 10–13 describe the main body of the algorithm, where the agent executes the greedily chosen action and records the observed transition in $\mathcal{D}_i$. The GP model for the transition function is updated after every step (Line 12). New Q-value estimates are computed every time after an update of the transition function (Line 13). Note that we use a separate GP to estimate the transition function in each simulator.

One can use a number of termination conditions (Line 5), *e.g.*, maximum number of steps, changes in the value function or maximum number of switches. In

our implementation, Algorithm 8 terminates if the change in new estimates of value functions in the real-world environment is no more than a certain threshold (ten percent) compared to the previous estimates.

The planner utilizes the knowledge of transitions from higher simulators (Lines 26–28) as well as lower simulators (Line 25) to encourage exploration in the current simulator. For every state action-action pair (s, a), the planner looks for the maximum fidelity simulator in which a known estimate of transitions for (s, a) is available and uses them to plan in the current simulator. An estimate is termed known if the variance is below a threshold. If no such simulator is available, then it uses the Q-values learned in the previous simulator plus a fidelity parameter β. This parameter models the maximum possible difference between the optimal Q-values in consecutive simulators. The higher fidelity simulator values will always be trusted over the lower fidelity ones as long as we have low enough uncertainty in those estimates. We assume that, for two consecutive simulators, the maximum difference between the optimal action value of a state-action pair in Σ_i and corresponding pair in Σ_{i-1} is not more than β_i. Note that one needs to apply a state-space discretization to plan the actions. However, the learned transition function is continuous.

5.3.2 GPQ-MFRL Algorithm

The agent learns optimal Q-values using GPs directly, instead of learning the model first. The underlying assumption is that nearby state-action pairs will produce similar Q-values. This assumption can also be applied to problems where

the states and actions are discrete but the transition function implies some sense of continuity. We choose the squared-exponential kernel because it models the spatial correlation we expect to see in a ground robot. However, any appropriate kernel can be used. We use a separate GP per simulator to estimate the Q-values using only data collected in that simulator. Algorithm 9 gives the details of the proposed

Algorithm 9 GPQ-MFRL Algorithm

procedure
Input: confidence parameters σ_{th} and σ_{th}^{sum}; simulator chain $\langle \boldsymbol{\Sigma}$, fidelity parameters $\boldsymbol{\beta}$, state mappings $\boldsymbol{\rho} \rangle$; $\mathcal{L}$.
Initialize: $\hat{Q}_i$ = initialize GP for $i \in \{1, \ldots, d\}$; state s_0 in simulator Σ_1; $i \leftarrow 1$; *change* = FALSE.
Initialize: $t \leftarrow 0$; $\mathcal{D}_i \leftarrow \{\}$ for $i \in \{1, \ldots, d\}$.
while terminal condition is not met
    $a_t \leftarrow$ CHOOSEACTION(s_t, i)
    if $\sigma_i(s_t, a_t) \leq \sigma_{th}$: *change* = TRUE
    if $\sigma\left(\rho_i(s_t), a_t\right) > \sigma_{th}$ and *change* and $i > 1$
        $s_t \leftarrow \rho_i(s_t)$, $i \leftarrow i - 1$, continue
    $\langle r_t, s_{t+1} \rangle \leftarrow$ execute action a_t in Σ_i
    append $\langle s_t, a_t, s_{t+1}, r_t \rangle$ to $\mathcal{D}_i$
    $\mathcal{Y}_i \leftarrow \{\}$
    for $\langle s_t, a_t, s_{t+1}, r_t \rangle \in \mathcal{D}_i$ //batch training//
        $y_t \leftarrow r_t + \gamma \max_a \hat{Q}_i\left(s_{t+1}, a\right)$
        append $\langle s_t, a_t, y_t \rangle$ to $\mathcal{Y}_i$
    $\hat{Q}_i \leftarrow$ update GP$_i$ using $\mathcal{Y}_i$
    if $\sum_{j=t}^{j=t-\mathcal{L}} \sigma_i(s_j, a_j) \leq \sigma_{th}^{sum}$ and $t > \mathcal{L}$ and $i < d$
        $s_t \leftarrow \rho_{i+1}^{-1}\left(s_t\right)$, $i \leftarrow i + 1$
        change = FALSE
end procedure

procedure CHOOSEACTION(s, i)
for $a \in \mathcal{A}(s)$
    $Q(s, a) = \hat{Q}_{i-1}\left(\rho_i(s), a\right) + \beta_i$
    for $k \in \{i, \ldots, d\}$
        $s_k = \rho_k^{-1} \ldots \rho_{i+2}^{-1} \rho_{i+1}^{-1}(s)$
        if $\sigma_k(s_k, a) \leq \sigma^{th}$: $Q(s, a) = \hat{Q}_k(s_k, a)$
return $\operatorname{argmax}_a Q(s, a)$
end procedure

framework. GPQ-MFRL continues to collect samples in the same simulator until the agent is confident about its optimal actions. If the running sum of the variances is below a threshold (Line 17), this suggests that the robot has found a good policy with high confidence in the current simulator and it must advance to the next one (Line 18).

GPQ-MFRL uses similar thresholds (σ_{th} and σ_{th}^{sum}) as GP-VI-MFRL to decide when to switch to a lower or higher fidelity simulator. GP-VI-MFRL checks if the agent has a sufficiently accurate estimate of the transition function in the previous simulator while GPQ-MFRL checks if the agent has a sufficiently accurate estimate of optimal Q-values in the previous simulator (Line 8). Lines 10–15 describe the main body of the algorithm where the agent records the observed transitions in $\mathcal{D}_i$. We update target values (Line 14) for every transition as more data gets collected in $\mathcal{D}_i$ (Line 13). The GP model is updated after every step (Line 16).

The agent utilizes the experiences collected in higher simulators (Lines 25–27) to choose the optimal action in the current simulator (Line 6). Specifically, it checks for the maximum fidelity simulator in which the posterior variance for (s, a) is less than a threshold σ_{th}. If one exists, it utilizes the Q-values from the highest known simulator to choose next action in the current simulator. If no such higher simulator exists, the Q-values from the previous simulator (Line 24) are considered to choose the next action in the current simulator with an additive fidelity parameter β.

GPQ-MFRL performs a batch retraining every time the robot collects new sample in a simulator (Lines 13–15). During the batch retraining, the algorithm updates the target values in previously collected training data using the knowledge

gained by collecting new samples. Then these updated target values are used to predict the Q-values using GPs (Line 16). As the amount of data grows, updating the GP can become computationally expensive. However, we can prune dataset using sparse GP techniques [192]. It is non-trivial to choose values for confidence bounds but for the current experiments we chose the σ_{th}^{sum} to be ten percent of the maximum Q-value possible and σ_{th} to be one fifth of σ_{th}^{sum}.

5.4 Empirical Evaluation

We use two environments to simulate GP-VI-MFRL and three environments for GPQ-MFRL. For GP-VI-MFRL, the goal is learning to navigate from one point to another while avoiding the obstacles. Σ_1 is a 21×21 grid-world with a point robot whereas Σ_2 is Gazebo (discretized in 21×21 grid) which simulates the kinematics and dynamics of a quadrotor operating in 3D. For GPQ-MFRL, the goal is to learn avoiding the obstacles while navigating through the environment. Σ_1 is Python-based simulator Pygame, Σ_2 is a Gazebo environment, and Σ_3 is the real world. We further use sparse GPs to speed up the computations required to perform GP inference. We report the improvements in computational time as well as a direct comparison between GP-VI-MFRL and GPQ-MFRL on an obstacle avoidance task.

5.4.1 GP-VI-MFRL Algorithm

The task of the robot is to navigate from the start state to goal state. The start and goal states and the obstacles for the environment used are shown in Figure 5.2.

The state of the robot is given by its X and Y coordinates whereas the action is a 2D velocity vector. Both simulators have the same state-space, therefore, ρ_i is an identity mapping. The robot gets a reward of zero for all transitions except when it hits the obstacles in which case it gets a reward of -50 and a reward of 100 for landing in the goal state.

Since the state space $\mathcal{S} \in \mathbb{R}^2$ and the action space (velocity) $\mathcal{A} \in \mathbb{R}^2$, the true transition function is $\mathbb{R}^4 \to \mathbb{R}^2$. However, generally GP regression allows for single-dimensional outputs only. Therefore, we assume independence between the two output dimensions and learn two components (along X and Y) of the transition functions separately, $x_{i+1} = f_x(x_i, y_i, a_x)$ and $y_{i+1} = f_y(x_i, y_i, a_y)$, where (x_i, y_i) and (x_{i+1}, y_{i+1}) are the current and next states of the robot, and (a_x, a_y) is the velocity input. The GP prediction is used to determine the transitions, $(x_i, y_i, a_x) \to x_{i+1}$ and $(x_i, y_i, a_y) \to y_{i+1}$ where (x_{i+1}, y_{i+1}) is the predicted next state with variances σ_x^2 and σ_y^2 respectively.

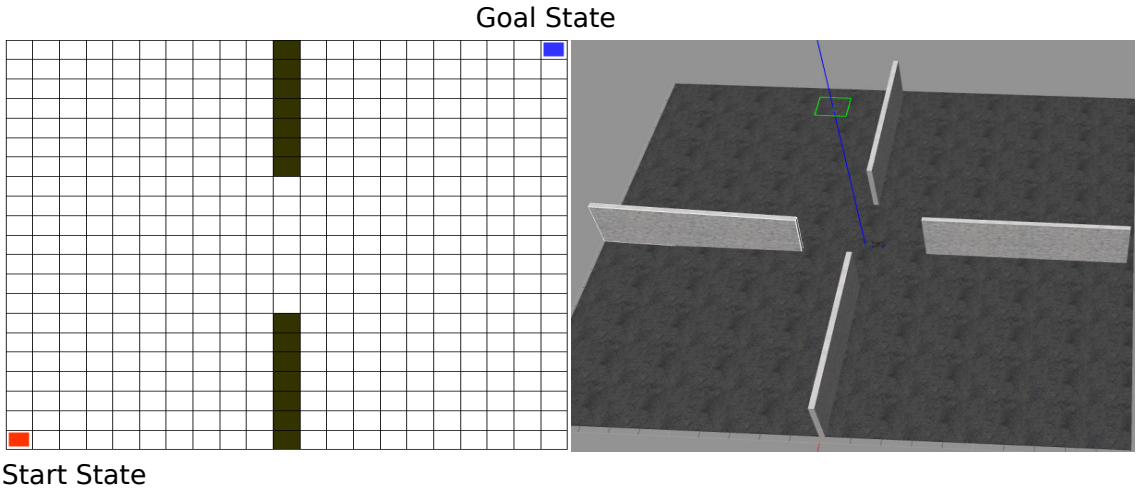


Figure 5.2: The environment setup for a multi-fidelity simulator chain. The grid-world simulator (Σ_1) has two walls whereas the Gazebo simulator (Σ_2) has four walls as shown.

Figure 5.3 shows the switching between the simulators for one run of the GP-VI-MFRL algorithm on the simulators shown in Figure 5.2. Unlike unidirectional transfer learning algorithms, GP-VI-MFRL agent switches back-and-forth in simulators collecting most of the samples in the first simulator initially. Eventually, the robot starts to collect more samples in the higher fidelity simulator. This is the case when the algorithm is near convergence and has accurate estimates for transitions in lower fidelity simulator as well. Next, we study the effect of the parameters used in GP-VI-MFRL and the fidelity of the simulators on the number of samples until convergence.

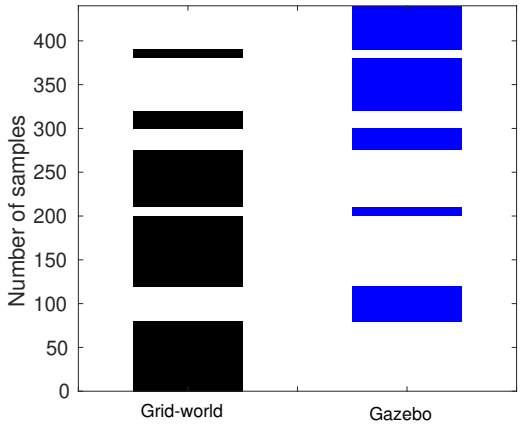
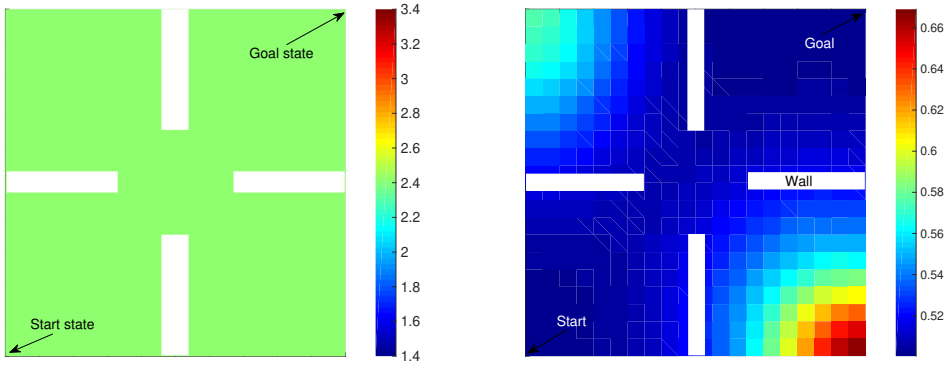


Figure 5.3: The figure represents the samples collected at each level of the simulator for a 21×21 grid in a grid-world and Gazebo environments. σ_{th}^{sum} and σ_{th} were kept 0.4 and 0.1 respectively.

5.4.1.1 Variance in learned transition function

To demonstrate how the variance of the predicted transition function varies from the beginning of the experiment to convergence, we plot "heatmaps" of the posterior variance for Gazebo environment transitions. The GP prediction for a



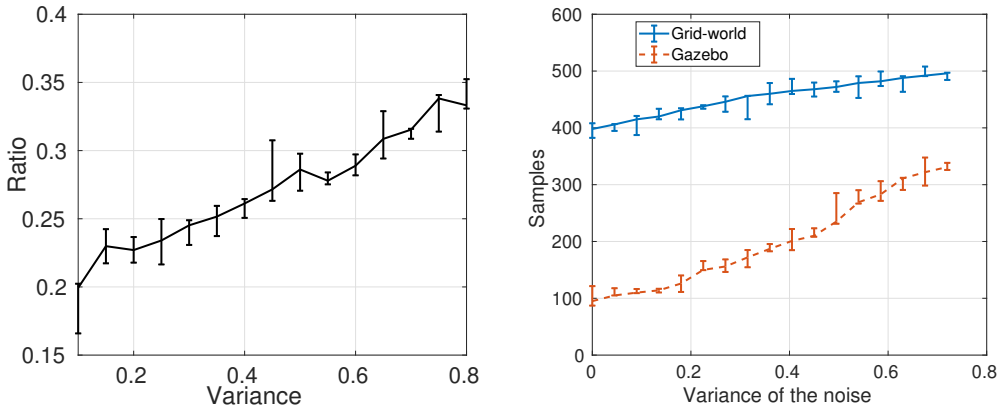
(a) Initialization (b) After convergence

Figure 5.4: Variance plot for Gazebo simulator after transition function initialization and after the algorithm has converged. Colored regions show the respective $\sqrt{\sigma_x^2 + \sigma_y^2}$ values for the optimal action returned by the planner in each state.

state-action pair gives the variance, σ_x^2 and σ_y^2, respectively for the predicted state. After convergence (Figure 5.4), the variance along the optimal (*i.e.,* likely) path is low whereas the variance for states unlikely to be on the optimal path from start to goal remains high since those states are explored less often in Gazebo environment. Hence utilizing the experience from lower fidelity simulator results in more directed exploration of the higher fidelity simulators.

5.4.1.2 Effect of fidelity on the number of samples

Next, we study the effect of varying the fidelity on the total number of samples and the fraction of the samples collected in the Gazebo simulator. Our hypothesis is that as the fidelity of the first simulator decreases, the agent will need more samples in Gazebo. In order to validate this hypothesis, we varied the noise added to simulate the transitions in the grid-world. The transition model in Gazebo remains the same.



(a) The ratio of samples collected in Gazebo and total samples (y-axis) as a function of the fidelity of grid-world. We lower the fidelity of grid-world by increasing the variance of the simulated transition function.

(b) The number of samples collected (y-axis) in Gazebo increases more rapidly (as demonstrated by diminishing vertical separation between the two plots) than samples collected in grid-world.

Figure 5.5: As we lower the fidelity of grid-world by adding more noise in grid-world transitions, the agent tends to spend more time in Gazebo. The plots show the average and min-max error bars of 5 trials.

The total number of samples collected increases as we increase the noise in grid-world (Figure 5.5(b)). As we increase the noise in grid-world, the agent learns less accurate transition function leading to more samples collected in Gazebo. Not only does the agent need more samples, the ratio of the samples collected in Gazebo to the total number of samples also increases (Figure 5.5(a)).

5.4.1.3 Effect of the confidence parameters

GP-VI-MFRL algorithm uses two confidence parameters, σ_{th} and σ_{th}^{sum}, which quantify the variances in the transition function to switch to a lower and higher simulator, respectively. Figure 5.6 shows the effect of varying the two parameters on the ratio of the number of samples collected in Gazebo simulator to the total number of samples. Smaller σ_{th} and σ_{th}^{sum} results in the agent collecting more samples

in the lower fidelity simulator and may result in slow convergence. Depending on the user preference, one can choose the values of confidence bounds from the Figure 5.6.

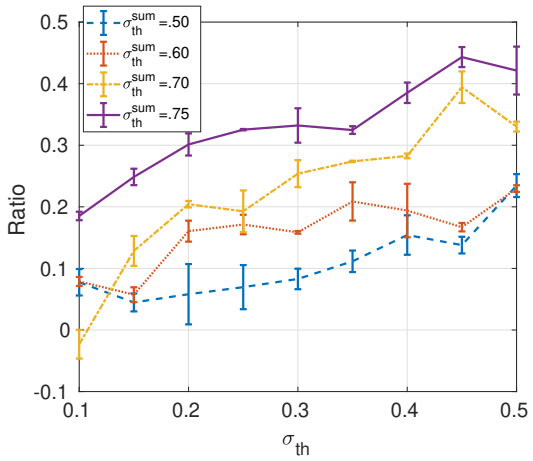


Figure 5.6: The ratio of samples collected in Gazebo to the total samples as a function of confidence parameter σ_{th} for four different values of σ_{th}^{sum}. The figure shows the average and standard deviation of 5 trials.

5.4.1.4 Comparison with RMax MFRL

Figure 5.7 compares GP-VI-MFRL with three other baseline algorithms,

1. RMax algorithm running only in Gazebo without grid-world (RMax),
2. GP-MFRL algorithm only in Gazebo with no grid-world present (GP-VI) and
3. Original MFRL algorithm [41] (RMax-MFRL).

Specifically, we plot the value of the initial state, $V(s_0)$, as a function of the number of samples in Gazebo, *i.e.,* Σ_2. We observe that GP-VI-MFRL uses fewer samples in Gazebo to converge to the optimal value than the other methods.

GP-VI-MFRL performs a GP update at each time step. GP update grows

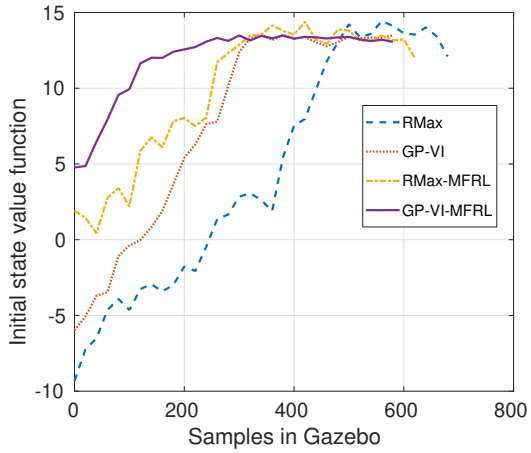


Figure 5.7: Comparison of GP-VI-MFRL with three baseline strategies. The Y-axis shows the value function for the initial state ($V(s_0)$) in Gazebo as a function of the number of samples collected in Gazebo. Value function estimation for GP-VI-MFRL converges fastest.

cubically with number of training samples which will make GP-VI-MFRL computationally infeasible beyond a certain number of training samples. However this issue can be addressed by using appropriate active learning strategies which select a subset of samples to retain thereby keeping the size of the dataset constant. The total computational time for GP-VI-MFRL to perform GP updates on collected samples accounts for approximately 10 minutes.

5.4.2 GPQ-MFRL Algorithm

We use three environments (Figure 5.8) to demonstrate the GPQ-MFRL algorithm. The task for the robot is to navigate through a given environment without crashing into the obstacles, assuming the robot has no prior information about the environments. There is no goal state.

The robot has a laser sensor that gives distances from obstacles along seven

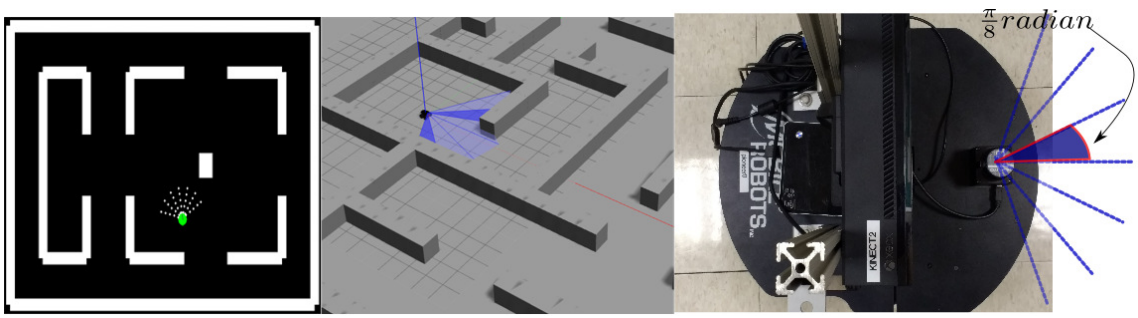


Figure 5.8: We use the Python-based simulator Pygame as Σ_1, Gazebo as Σ_2 and Pioneer P3-DX robot in real-world as Σ_3.

equally spaced directions. The angle between two consecutive measurement directions was set to be $\frac{\pi}{8}$ radians. The actual robot has a Hokuyo laser sensor that operates in the same configuration. Distance measurements along the seven directions serve as the state in the environments. Therefore we have a seven-dimensional continuous state space: $\mathcal{S} \in (0, 5]^7$. The linear speed of the robot was held constant at 0.2 m/sec. The robot can choose its angular velocity from nineteen possible options: $\{-\frac{\pi}{9}, -\frac{\pi}{8}, \ldots, \frac{\pi}{9}\}$. The reward in each state was set to be the sum of laser readings from seven directions except when the robot hits the obstacle. In case of a collision, it gets a reward of -50.

We train the GP regression, $Q(\mathbf{s}, a) : \mathbb{R}^8 \rightarrow \mathbb{R}$. Hyperparameters of the squared-exponential kernel were calculated off-line by minimizing the negative log marginal likelihood of 2000 training points which were collected by letting the robot run in the real world directly. The parameter values for experiments in this section are given in Table 5.1.

Table 5.1: Parameters used in GPQ-MFRL

Description	Type	Value
Hyperparameters	σ	102.74
	l	[2.1, 5.1, 14, 6.2, 15, 2, 2, 1]
	ω^2	20
Confidence parameters	σ_{th}^{sum}	60
	σ_{th}	15
Algorithm	$\mathcal{L}$	5

5.4.2.1 Average Cumulative Reward in Real-World

In Figure 5.9, we compare GPQ-MFRL algorithm with three other baseline strategies by plotting the average cumulative reward collected by the robot as a function of samples collected in the real world. Three baseline strategies are,

1. Directly collecting samples in real world without the simulators (Direct),
2. Collect hundred samples in one simulator and transfer the policy to the Pioneer robot with no further learning in the real world (Frozen Policy) and
3. Collect hundred samples in one simulator and transfer the policy to the robot while continuing to learn in the real world (Transferred Policy).

We observe that the Direct policy performs worst in the beginning. It can be attributed to the fact that the robot started to learn from scratch. The Frozen policy starts better since it has already learned a policy in the simulator. However, it tends to a lower value of average cumulative reward which suggests that the optimal policy learned in the simulator is not the optimal policy in the real world. Although, the Transferred policy seems to perform better at the beginning than the Frozen policy, it is difficult to dictate if it will always be the case. The Direct policy

has a large performance variance in the beginning. GPQ-MFRL outperforms the other strategies right from the beginning. We attribute this to the fact that the GPQ-MFRL collects more samples from the simulator in the beginning and hence starts better right from the start. If one would allow the Transferred Policy and the Frozen Policy to collect more samples from the simulator they might have performed the same as the GPQ-MFRL. However, deciding how many samples one should allow is a non-trivial task and problem-specific. GPQ-MFRL can decide the number of samples in each simulator by itself without the need for human intervention.

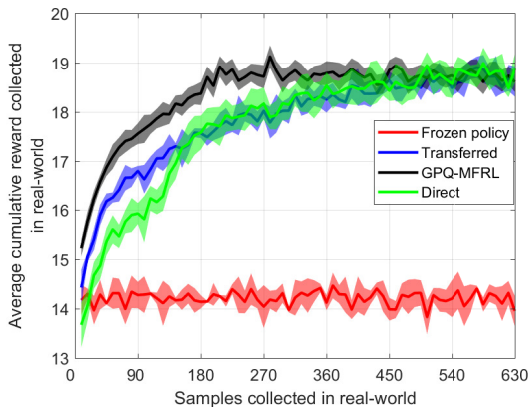
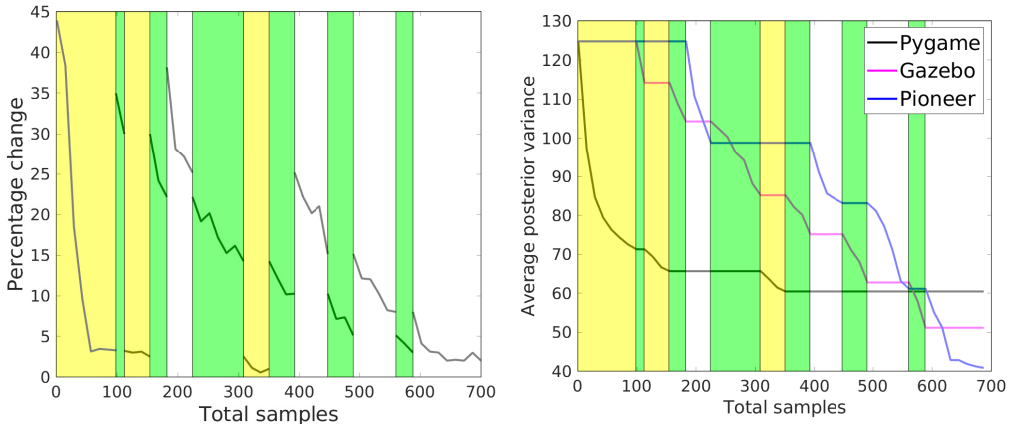


Figure 5.9: Average cumulative reward collected by the Pioneer robot in real-world environment as a function of the samples collected in the real world. The plot shows the average and standard deviation of 5 trials.

5.4.2.2 Policy Variation with Time

Figure 5.10 shows the absolute percentage change in the sum of the value functions with respect to last estimated sum of value functions and average predictive variance for states $\{1, 3, 5\}^7$ in all three simulators. Observe that initially most of the samples are collected in the simulator, whereas over time the samples are col-



(a) Sum of absolute change in value functions

(b) Average variances in value function estimations

Figure 5.10: Yellow, green and white regions correspond to the samples collected in the Pygame, Gazebo and real-world environments respectively. Plots are for state set $\{1, 3, 5\}^7$.

lected mostly in the real-world. The simulators help the robot to make its value estimates converge quickly as observed by a sharp dip in the first white region. Note that GP updates for i^{th} simulator ($\hat{Q}_i$) are made only when the robot is running in i^{th} simulator.

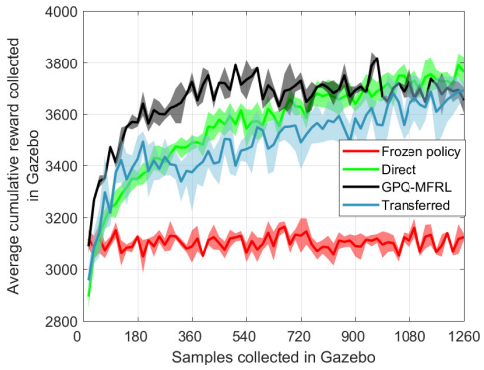
5.4.2.3 Higher-dimensional spaces and sparse GPs

One of the limitations of GPs is their computational complexity which grows cubically with the number of training samples. However, we use sparse approximations to address this limitation. We also increase the dimensionality of the state-space to verify if the proposed algorithms scale to higher dimensions. Specifically, we increase the number of laser readings to 180 equally spaced directions. Therefore, GP regression is used to estimate $Q(\mathbf{s}, a) : \mathbb{R}^{181} \rightarrow \mathbb{R}$.

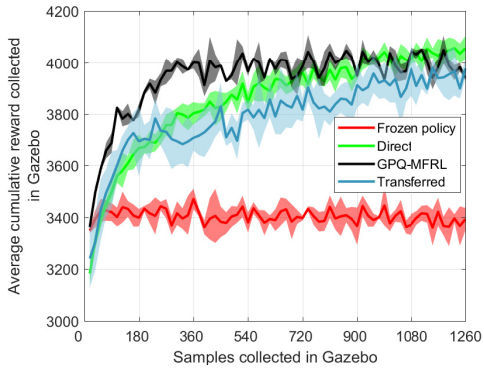
There are several methods for sparse GP approximations. We use the technique from [192]that finds a possibly smaller set of points (called inducing points) which fit the data best. The GP inference is conditioned on the smaller set of inducing points rather than the full set of training samples. Finding inducing points is closely related to finding low-rank approximations of the full GP covariance matrix. The inducing points may or may not belong to the actual training data.

We did several experiments with GPQ-MFRL to study the performance of the algorithm for number of inducing points. We use Pyro [193] to implement sparse GPs. Figure 5.11 shows the average cumulative reward collected by the robot in the Gazebo environment when GP inference is done with the number of inducing points set to 5%, 15%, 25% of the total training samples and using all the training samples. We observe a significant increase in cumulative reward collected going from 5% to 15% but not much from 15% to 25% (y-axes in Figure 5.11). A plot of the wall clock times to perform GP inference in Pygame is shown in Figure 5.12. The wall time to perform GP inference in Gazebo exhibits a similar trend which we omit for the sake of brevity. The wall clock time includes the time to perform all GP operations including the time to update the hyperparameters as well as finding the inducing points of both GPs. We update these after every ten new training samples in an individual simulator. The experiments were run on a machine running Ubuntu 16.04 with Intel(R) Core(TM) i7-5600U CPU @ 2.60 GHz, Intel HD Graphics 5500 and 16 GB RAM.

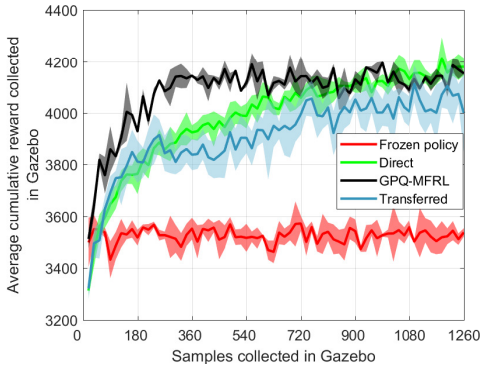
The results suggests that a small fraction of inducing points are sufficient and yields diminishing marginal gains in the performance when the number of inducing



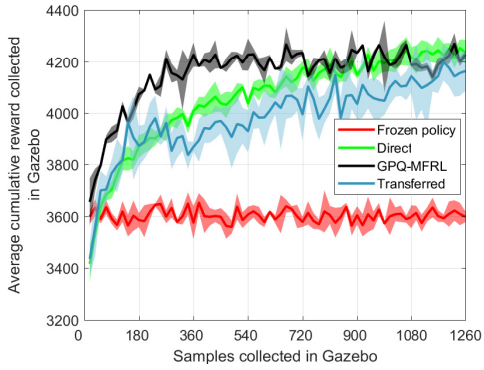
(a) Inducing points set to 5% of the training sample size



(b) Inducing points set to 15% of the training sample size



(c) Inducing points set to 25% of the training sample size



(d) Full GP inference

Figure 5.11: Average cumulative reward collected by the robot in Gazebo environment as a function of the samples collected in Gazebo for different percentage of inducing points. The plots show the average and standard deviation of 5 trials.

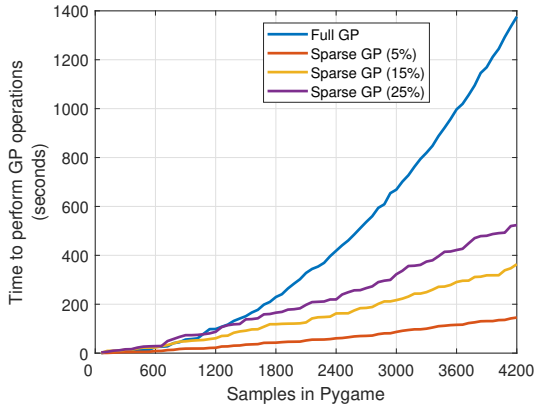


Figure 5.12: The wall clock time required to perform all GP related operations in Pygame for various degrees of GP sparse approximations in Pygame. The plot shows the mean time over five trials for each case.

points increases. Doing inference on inducing points with 25% of the training data performs almost as good as doing full GP regression, in terms of the reward collected by the learned policy (Figure 5.11(d)) but is significantly faster.

5.4.3 Comparison between GP-VI-MFRL and GPQ-MFRL

We compare GP-VI-MFRL and GPQ-MFRL using average cumulative reward collected by the robot in Gazebo as the metric in the obstacle avoidance task. To do this comparison, we use full GP regression to perform the inference. The laser obtains distance measurements from seven equally spaced directions and we train seven independent GPs to learn the transition function in GP-VI-MFRL (one GP corresponding to each direction). A performance comparison is shown in Figure 5.13. Though both algorithms seem to perform the same asymptotically, GP-VI-MFRL performs slightly better than GPQ-MFRL in the beginning.

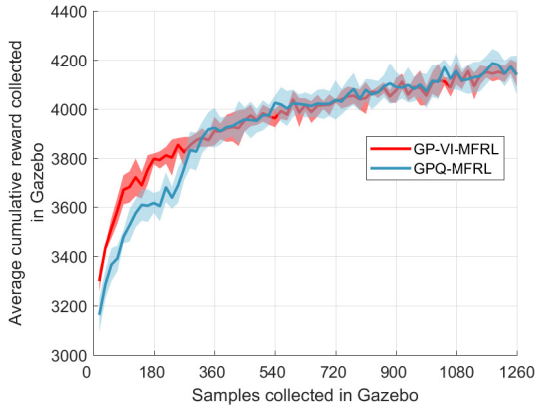


Figure 5.13: Average cumulative reward collected by the robot in Gazebo environment as a function of the samples collected in Gazebo for GP-VI-MFRL and GP-Q-MFRL. The plots show the average and standard deviation of 10 trials.

5.5 Conclusions

In this chapter, we present GP-based MFRL algorithms that leverage multiple simulators with varying fidelity and costs to learn a policy in the real world. We demonstrated empirically that the GP-based MFRL algorithms find the optimal policies using fewer samples than the baseline algorithms, including the original MFRL algorithm [41]. The computational limitations of sparse GPs can be mitigated to an extent by the use of sparse GP approximations. We also provided a head-to-head comparison between the two algorithms presented here. GP-VI-MFRL (model-based version) performs better than GPQ-MFRL (model-free version) in the beginning. This is consistent with the outcomes for traditional RL techniques where model-based algorithms tend to perform better than model-free ones.

Chapter 6: Conclusion and Future Work

In this dissertation, we focused on bridging the gap between learning algorithms and their application in the context of robotics. More broadly, the question that underlines the work reported in this dissertation was: How do we make robots learn as efficiently as possible with a minimal amount of physical interaction? Using learning algorithms naively in robotics presents unique challenges and is often not directly applicable to real-world robotics systems. This dissertation approached the problem of robot learning along two fronts: extrinsic learning and intrinsic learning. For both fronts, our goal was to develop algorithms that provably reduce the number of physical interactions needed by the robots to learn.

While the algorithms we have developed are for specific (albeit canonical) problems in extrinsic and intrinsic learning, we believe the techniques presented are of broad applicability. We predominantly used GPs in all problems. There has been work on sample efficiency in GP regression and classification [194]. However, our definition of sample complexity is broader than just the actual number of physical samples and we think of it in terms of the cost of learning samples. In extrinsic learning, the cost of samples is the travel and measurement time while in intrinsic learning is time spent on the real robot. It can be something different in other

robotics applications (e.g., energy consumption, wear-and-tear of the robot hardware, etc.). This dissertation has shown how to exploit the structural properties of GPs to reduce the cost of learning samples in any learning task with robots.

The majority of the work in extrinsic learning literature focuses on minimizing only the traveling time for robots to learn an environment. However, in many scenarios, not only the traveling time but the measurement time to collect the measurements also contributes significantly to the total robot operating time. For example, in the case of a robot chemical sensor attached to a measurement probe has to stop at a location for some time and wait until the probe makes the measurement. Our algorithms provide guarantees on the total time which is the summation of the traveling time as well as the measurement time. While there has been plenty of work on simulator-based RL, there is no standard way of determining when should the agents should switch the simulators. Our algorithms provide a concrete way of using the GP variance for determining when should the agents switch between simulators.

For extrinsic learning, we focused on two versions. In the first version, we studied the scenarios where we deploy robots to learn an environment in its entirety. This can be crucial in scenarios such as a farmer trying to identify the farm regions that need fertilizer boost. This is an NP-Hard problem. We modeled the underlying environment using Gaussian Processes (GPs), presented three polynomial-time approximation algorithms, and provided the theoretical guarantees of their performance. The first algorithm deals with placing a set of stationary sensors in an environment. The second algorithm called DiskCoverTour plans

a tour for a single robot to collect the measurements from the environment. The third algorithm called k-DiskCoverTour addresses the scenarios if we have multiple robots operating in the environment and how we distribute the task between them in a manner that gives us a lower bound on the performance of the slowest robot. We validated our algorithms using a real-world dataset of organic matter content on a farm.

In the second version, we are interested in finding the hotspot in a given environment. In many practical applications, finding the hotspot quickly is crucial. For example, we can identify the regions with a high concentration of certain chemicals in a water body, which can help scientists to study marine life. Here also we use GPs to model the underlying environment, relax the assumption of GP hyperparameters being known and study both single-robot as well as multi-robot scenarios. For the single-robot case, we provide two algorithms that vary the GP hyperparameters in a computationally tractable way and evaluate their performance on a real-world dataset of organic matter content in a farm. However, if the environment is large, one robot may not be sufficient to find the hotspot. Hence, we also study a multi-robot decentralized approach that uses Voronoi partitioning for planning and distributing the exploration task between robots. Our Voronoi partitioning makes use of the Upper Confidence Bound (UCB) values of the learned GP model. This makes sure that the robots get ample opportunity to explore the regions of higher values. Further, we compare various planners on top of the Voronoi partitioning. Some of the planners are a direct extension of the planners used in the single-robot case as well as some heuristic planners. We also provide a direct comparison between

using Voronoi partition and no partitioning.

While considering the problem of environmental monitoring, robot navigation is a crucial component to consider. How do we autonomously make robots navigate in unknown environments? Hence, our last chapter focused on learning optimal behaviors for mobile robots in unknown environments autonomously. We used an RL-based approach. RL has shown great promise in the context of robotics but remains limited to simulations mainly. We do real-world experiments where our task was to navigate in an indoor environment autonomously. We combined GPs with the Multi-Fidelity Reinforcement Learning (MFRL) framework and provide both model-based and model-free RL algorithms. These algorithms achieve up to 40% and 60% reduction in the real-world samples required respectively.

This dissertation sets the stage for some immediate and long-term future work. We dealt only with stationary environments here that do not evolve with time. However, in many practical applications that is not the case and the underlying environments can evolve quite a bit during the operation time. For example, a chemical spill in a water body can evolve as the robots are collecting measurements [29,31]. It would be interesting to extend and study our algorithms in such spatiotemporal environments. Leonard et al. [195] presented a formalization of the adaptive sampling problem using non-dimensional numbers for non-stationary fields. Specifically, the non-dimensional numbers included the size, shape, normalized robot speed, sensor noise, and sampling time interval. Investigating this in the context of the problems studied in this dissertation in extrinsic learning is an interesting direction for future work. Exploring other kernels such as signature kernels would be one direct exten-

sion [196]. Although we designed our planners keeping only stationary environments in mind, there is work on Voronoi partitioning for spatio-temporal fields [197, 198]. It would be interesting to extend our planners to such cases.

Deep learning-based approaches have been found to be very promising recently. While the bulk of the work has been on intrinsic learning [4, 13], there is potential for deep learning methods to be useful in extrinsic learning as well. For example, Rangwala et al. presented a Deep Learning-based pasture prediction model (DeepPaSTL) for predicting a spatiotemporal field in the context of precision agriculture [199]. DeepPaSTL can learn with new data accumulated over months, the model has an inherent capacity to effectively adapt to varying climatic and environmental conditions. Our planners can be integrated with such models and is a promising direction for future work.

Bibliography

[1] Rajat Mishra, Mandar Chitre, and Sanjay Swarup. Online informative path planning using sparse gaussian processes. In *2018 OCEANS-MTS/IEEE Kobe Techno-Oceans (OTO)*, pages 1–5. IEEE, 2018.

[2] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[3] Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[4] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[5] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016.

[6] O. Obulesu, M. Mahendra, and M. ThrilokReddy. Machine learning techniques and tools: A survey. In *2018 International Conference on Inventive Research in Computing Applications (ICIRCA)*, pages 605–611, 2018.

[7] Oliver Kroemer, Scott Niekum, and George Dimitri Konidaris. A review of robot learning for manipulation: Challenges, representations, and algorithms. *Journal of machine learning research*, 22(30), 2021.

[8] Harish Ravichandar, Athanasios S Polydoros, Sonia Chernova, and Aude Billard. Recent advances in robot learning from demonstration. *Annual Review of Control, Robotics, and Autonomous Systems*, 3:297–330, 2020.

[9] Andrei A Rusu, Matej Večerík, Thomas Rothörl, Nicolas Heess, Razvan Pascanu, and Raia Hadsell. Sim-to-real robot learning from pixels with progressive nets. In *Conference on Robot Learning*, pages 262–270. PMLR, 2017.

[10] Yuqing Du, Olivia Watkins, Trevor Darrell, Pieter Abbeel, and Deepak Pathak. Auto-tuned sim-to-real transfer. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1290–1296, 2021.

[11] Francesco Varini, Jordan Boyd-Graber, Massimiliano Ciaramita, and Markus Leippold. Climatext: A dataset for climate change topic detection. 2020.

[12] Huazhe Xu, Yang Gao, Fisher Yu, and Trevor Darrell. End-to-end learning of driving models from large-scale video datasets. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2174–2182, 2017.

[13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[14] OpenAI, Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, Jonas Schneider, Nikolas Tezak, Jerry Tworek, Peter Welinder, Lilian Weng, Qiming Yuan, Wojciech Zaremba, and Lei Zhang. Solving rubik's cube with a robot hand. *CoRR*, abs/1910.07113, 2019.

[15] Wenlong Mou, Zheng Wen, and Xi Chen. On the sample complexity of reinforcement learning with policy space generalization. *arXiv preprint arXiv:2008.07353*, 2020.

[16] M. Cutler and J. P. How. Efficient reinforcement learning for robots using informative simulated priors. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2605–2612, 2015.

[17] Dmitry Kalashnikov, Jacob Varley, Yevgen Chebotar, Benjamin Swanson, Rico Jonschkowski, Chelsea Finn, Sergey Levine, and Karol Hausman. Mt-opt: Continuous multi-task robotic reinforcement learning at scale. *arXiv preprint arXiv:2104.08212*, 2021.

[18] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016.

[19] Martin Pecka and Tomas Svoboda. Safe exploration techniques for reinforcement learning – an overview. In Jan Hodicky, editor, *Modelling and Simulation for Autonomous Systems*, pages 357–375, Cham, 2014. Springer International Publishing.

[20] Yanchao Sun, Xiangyu Yin, and Furong Huang. Temple: Learning template of transitions for sample efficient multi-task rl. *arXiv preprint arXiv:2002.06659*, 2020.

[21] Yanchao Sun and Furong Huang. Can agents learn by analogy? an inferable model for pac reinforcement learning. AAMAS '20, page 1332–1340, Richland, SC, 2020. International Foundation for Autonomous Agents and Multiagent Systems.

[22] Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. *arXiv preprint arXiv:1804.10332*, 2018.

[23] Marija Popović, Teresa Vidal-Calleja, Gregory Hitz, Jen Jen Chung, Inkyu Sa, Roland Siegwart, and Juan Nieto. An informative path planning framework for UAV-based terrain monitoring. *Autonomous Robots*, 44(6):889–911, February 2020.

[24] Andreas Krause and Carlos Guestrin. Nonmyopic active learning of gaussian processes: an exploration-exploitation approach. In *Proceedings of the 24th international conference on Machine learning*, pages 449–456. ACM, 2007.

[25] Daniel E Soltero, Mac Schwager, and Daniela Rus. Generating informative paths for persistent sensing in unknown environments. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2172–2179. IEEE, 2012.

[26] Stephanie Kemna, Oliver Kroemer, and Gaurav S Sukhatme. Pilot surveys for adaptive informative sampling. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6417–6424. IEEE, 2018.

[27] Nannan Cao, Kian Hsiang Low, and John M Dolan. Multi-robot informative path planning for active sensing of environmental phenomena: A tale of two algorithms. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pages 7–14. International Foundation for Autonomous Agents and Multiagent Systems, 2013.

[28] Varun Suryan and Pratap Tokekar. Learning a spatial field in minimum time with a team of robots. *IEEE Transactions on Robotics (TRO)*, 36(5):1562–1576, October 2020.

[29] Jan Blachowski. Spatial analysis of the mining and transport of rock minerals (aggregates) in the context of regional development. *Environmental Earth Sciences*, 71(3):1327–1338, Feb 2014.

[30] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

[31] Yoonchang Sung, Deeksha Dixit, and Pratap Tokekar. Online multi-robot exploration of a translating plume: Competitive algorithm and experiments. *arXiv preprint arXiv:1811.02769*, 2018.

[32] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 1998.

[33] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *J. Artif. Int. Res.*, 4(1):237–285, May 1996.

[34] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[35] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. The MIT Press, 2016.

[36] Aristotelis Lazaridis, Anestis Fachantidis, and Ioannis Vlahavas. Deep reinforcement learning: A state-of-the-art walkthrough. *Journal of Artificial Intelligence Research*, 69:1421–1471, 2020.

[37] Aaron M Roth, Jing Liang, and Dinesh Manocha. Xai-n: Sensor-based robot navigation using expert policies and decision trees. *arXiv preprint arXiv:2104.10818*, 2021.

[38] Utsav Patel, Nithish Kumar, Adarsh Jagan Sathyamoorthy, and Dinesh Manocha. Dynamically feasible deep reinforcement learning policy for robot navigation in dense mobile crowds. *arXiv preprint arXiv:2010.14838*, 2020.

[39] Yilin Wu, Wilson Yan, Thanard Kurutach, Lerrel Pinto, and Pieter Abbeel. Learning to manipulate deformable objects without demonstrations, 2020.

[40] Yevgen Chebotar, Ankur Handa, Viktor Makoviychuk, Miles Macklin, Jan Issac, Nathan Ratliff, and Dieter Fox. Closing the sim-to-real loop: Adapting simulation randomization with real world experience. *2019 International Conference on Robotics and Automation (ICRA)*, May 2019.

[41] M. Cutler, T. J. Walsh, and J. P. How. Real-world reinforcement learning via multifidelity simulators. *IEEE Transactions on Robotics*, 31(3):655–671, June 2015.

[42] Andrew West, Ioannis Tsitsimpelis, Mauro Licata, Anze Jazbec, Luka Snoj, Malcolm J Joyce, and Barry Lennox. Use of gaussian process regression for radiation mapping of a nuclear reactor with a mobile robot. *Scientific reports*, 11(1):1–11, 2021.

[43] Andreas Krause, Ajit Singh, and Carlos Guestrin. Near-optimal sensor placements in gaussian processes: Theory, efficient algorithms and empirical studies. *Journal of Machine Learning Research*, 9(Feb):235–284, 2008.

[44] DJ Mulla. Mapping and managing spatial patterns in soil fertility and crop yield. In *Proceedings of soil specific crop management*, number proceedingsofso, pages 15–26. American Society of Agronomy, Crop Science Society of America, Soil Science ..., 1993.

[45] Kian Hsiang Low, Geoffrey J Gordon, John M Dolan, and Pradeep Khosla. Adaptive sampling for multi-robot wide-area exploration. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 755–760. IEEE, 2007.

[46] Bin Zhang and Gaurav S Sukhatme. Adaptive sampling for estimating a scalar field using a robotic boat and a sensor network. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 3673–3680. IEEE, 2007.

[47] Mohammad Rahimi, Richard Pon, William J Kaiser, Gaurav S Sukhatme, Deborah Estrin, and Mani Srivastava. Adaptive sampling for environmental robotics. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, volume 4, pages 3537–3544. IEEE, 2004.

[48] Amanda Prorok, Alexander Bahr, and Alcherio Martinoli. Low-cost collaborative localization for large-scale multi-robot systems. In *2012 IEEE International Conference on Robotics and Automation*, pages 4236–4241, 2012.

[49] Varun Suryan and Pratap Tokekar. Learning a spatial field with gaussian process regression in minimum time. In *Algorithmic Foundations of Robotics XIII*, pages 301–317, Cham, 2020. Springer International Publishing.

[50] Jnaneshwar Das, Frédéric Py, Julio BJ Harvey, John P Ryan, Alyssa Gellene, Rishi Graham, David A Caron, Kanna Rajan, and Gaurav S Sukhatme. Data-driven robotic sampling for marine ecosystem monitoring. *The International Journal of Robotics Research*, 34(12):1435–1452, 2015.

[51] Niranjan Srinivas, Andreas Krause, Sham M Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. *arXiv preprint arXiv:0912.3995*, 2009.

[52] Weizhe Chen and Lantao Liu. Pareto monte carlo tree search for multi-objective informative planning. In *Proceedings of Robotics: Science and Systems*, FreiburgimBreisgau, Germany, June 2019.

[53] Varun Suryan, Nahush Gondhalekar, and Pratap Tokekar. Multi-fidelity reinforcement learning with gaussian processes: Model-based and model-free algorithms. *IEEE Robotics and Automation Magazine*, 2020. Special issue on Deep Learning and Machine Learning in Robotics.

[54] Varun Suryan, Nahush Gondhalekar, and Pratap Tokekar. Multi-fidelity model-free reinforcement learning with gaussian processes. In *AAAI 2018 Fall Symposium on Reasoning and Learning in Real-World Systems for Long-Term Autonomy*, pages 82–87, 2018.

[55] Y. T. Tan, A. Kunapareddy, and M. Kobilarov. Gaussian process adaptive sampling using the cross-entropy method for environmental sensing and monitoring. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6220–6227, 2018.

[56] Wenhao Luo and Katia Sycara. Adaptive sampling and online learning in multi-robot sensor coverage with mixture of gaussian processes. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6359–6364, 2018.

[57] Fang Bian, David Kempe, and Ramesh Govindan. Utility based sensor selection. In *Proceedings of the 5th international conference on Information processing in sensor networks*, pages 11–18. ACM, 2006.

[58] William J Welch. Branch-and-bound search for experimental designs based on d optimality and other criteria. *Technometrics*, 24(1):41–48, 1982.

[59] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[60] Dorit S Hochbaum and Wolfgang Maass. Approximation schemes for covering and packing problems in image processing and vlsi. *Journal of the ACM (JACM)*, 32(1):130–136, 1985.

[61] H. González-Banos. A randomized art-gallery algorithm for sensor placement. In *Proceedings of the Seventeenth Annual Symposium on Computational Geometry*, SCG '01, pages 232–240, New York, NY, USA, 2001. ACM.

[62] William F Caselton and James V Zidek. Optimal monitoring network designs. *Statistics & Probability Letters*, 2(4):223–227, 1984.

[63] Noel Cressie. Statistics for spatial data. *Terra Nova*, 4(5):613–617, 1992.

[64] Michael C Shewry and Henry P Wynn. Maximum entropy sampling. *Journal of applied statistics*, 14(2):165–170, 1987.

[65] Chun-Wa Ko, Jon Lee, and Maurice Queyranne. An exact algorithm for maximum entropy sampling. *Operations Research*, 43(4):684–691, 1995.

[66] Naren Ramakrishnan, Chris Bailey-Kellogg, Satish Tadepalli, and Varun N Pandey. Gaussian processes for active data mining of spatial aggregates. In *Proceedings of the 2005 SIAM International Conference on Data Mining*, pages 427–438. SIAM, 2005.

[67] Dale L Zimmerman. Optimal network design for spatial prediction, covariance parameter estimation, and empirical prediction. *Environmetrics*, 17(6):635–652, 2006.

[68] Zhengyuan Zhu and Michael L. Stein. Spatial sampling design for prediction with estimated parameters. *Journal of Agricultural, Biological, and Environmental Statistics*, 11(1):24, Mar 2006.

[69] George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. An analysis of approximations for maximizing submodular set functions—i. *Mathematical Programming*, 14(1):265–294, 1978.

[70] Anthony C. Atkinson. *Optimal Design*, pages 1–17. American Cancer Society, 2015.

[71] L. Van Nguyen, S. Kodagoda, R. Ranasinghe, and G. Dissanayake. Simulated annealing based approach for near-optimal sensor selection in gaussian processes. In *2012 International Conference on Control, Automation and Information Sciences (ICCAIS)*, pages 142–147. IEEE, Nov 2012.

[72] Syed Talha Jawaid and Stephen L Smith. Informative path planning as a maximum traveling salesman problem with submodular rewards. *Discrete Applied Mathematics*, 186:112–127, 2015.

[73] Kian Hsiang Low, Jie Chen, John M Dolan, Steve Chien, and David R Thompson. Decentralized active robotic exploration and mapping for probabilistic field classification in environmental sensing. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 105–112. International Foundation for Autonomous Agents and Multiagent Systems, 2012.

[74] Frédéric Galland, Philippe Réfrégier, and Olivier Germain. Synthetic aperture radar oil spill segmentation by stochastic complexity minimization. *IEEE Geoscience and Remote Sensing Letters*, 1(4):295–299, 2004.

[75] Aarti Singh, Robert Nowak, and Parmesh Ramanathan. Active learning for adaptive mobile sensing networks. In *Proceedings of the 5th international conference on Information processing in sensor networks*, pages 60–68. ACM, 2006.

[76] Karthik Dantu and Gaurav S Sukhatme. Detecting and tracking level sets of scalar fields using a robotic sensor network. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 3665–3672. IEEE, 2007.

[77] Sumana Srinivasan, Krithi Ramamritham, and Purushottam Kulkarni. Ace in the hole: Adaptive contour estimation using collaborating mobile sensors. In *Proceedings of the 7th international conference on Information processing in sensor networks*, pages 147–158. IEEE Computer Society, 2008.

[78] Brent Bryan, Robert C Nichol, Christopher R Genovese, Jeff Schneider, Christopher J Miller, and Larry Wasserman. Active learning for identifying function threshold boundaries. In *Advances in neural information processing systems*, pages 163–170, 2006.

[79] Brent Bryan and Jeff Schneider. Actively learning level-sets of composite functions. In *Proceedings of the 25th international conference on Machine learning*, pages 80–87. ACM, 2008.

[80] Alkis Gotovos, Nathalie Casati, Gregory Hitz, and Andreas Krause. Active learning for level set estimation. In *Twenty-Third International Joint Conference on Artificial Intelligence*, 2013.

[81] M. Wallat, T. Graepel, and K. Obermayer. Gaussian process regression: active data selection and test point rejection. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*, volume 3, pages 241–246 vol.3, July 2000.

[82] Andreas Krause and Carlos Guestrin. Submodularity and its applications in optimized information gathering. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(4):32, 2011.

[83] Chandra Chekuri and Martin Pal. A recursive greedy algorithm for walks in directed graphs. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS'05)*, pages 245–253. IEEE, 2005.

[84] Jonathan Binney, Andreas Krause, and Gaurav S Sukhatme. Optimizing waypoints for monitoring spatiotemporal phenomena. *The International Journal of Robotics Research*, 32(8):873–888, 2013.

[85] Abhimanyu Das and David Kempe. Algorithms for subset selection in linear regression. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 45–54. ACM, 2008.

[86] Naomi Ehrich Leonard, Derek A Paley, Francois Lekien, Rodolphe Sepulchre, David M Fratantoni, and Russ E Davis. Collective motion, sensor networks, and ocean sampling. *Proceedings of the IEEE*, 95(1):48–74, 2007.

[87] Dan O Popa, Muhammad F Mysorewala, and Frank L Lewis. Ekf-based adaptive sampling with mobile robotic sensor nodes. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2451–2456. IEEE, 2006.

[88] Robert Sim and Nicholas Roy. Global a-optimal robot exploration in slam. In *Proceedings of the 2005 IEEE international conference on robotics and automation*, pages 661–666. IEEE, 2005.

[89] Amarjeet Singh, Andreas Krause, Carlos Guestrin, and William J Kaiser. Efficient informative sensing using multiple robots. *Journal of Artificial Intelligence Research*, 34:707–755, 2009.

[90] A Tuchscherer. Sampling. *Biometrical Journal*, 35(7):848–848, 1993.

[91] Kian Hsiang Low, John M Dolan, and Pradeep Khosla. Adaptive multi-robot wide-area exploration and mapping. In *Proceedings of the 7th International Joint Conference on Autonomous agents and Multiagent systems-Volume 1*,

pages 23–30. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

[92] Kian Hsiang Low, John M. Dolan, and Pradeep Khosla. Information-theoretic approach to efficient adaptive path planning for mobile robotic environmental sensing. In *Proceedings of the Nineteenth International Conference on International Conference on Automated Planning and Scheduling*, ICAPS'09, pages 233—-240. AAAI Press, 2009.

[93] Roman Garnett, Yamuna Krishnamurthy, Xuehan Xiong, Jeff Schneider, and Richard Mann. Bayesian optimal active search and surveying. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, ICML'12, pages 843—-850. Omnipress, 2012.

[94] Namik Kemal Yilmaz, Constantinos Evangelinos, Pierre FJ Lermusiaux, and Nicholas M Patrikalakis. Path planning of autonomous underwater vehicles for adaptive sampling using mixed integer linear programming. *IEEE Journal of Oceanic Engineering*, 33(4):522–537, 2008.

[95] Geoffrey A Hollinger and Gaurav S Sukhatme. Sampling-based robotic information gathering algorithms. *The International Journal of Robotics Research*, 33(9):1271–1287, 2014.

[96] Chun Kai Ling, Kian Hsiang Low, and Patrick Jaillet. Gaussian process planning with lipschitz continuous reward functions: Towards unifying bayesian optimization, active learning, and beyond. In *AAAI*, pages 1860–1866, 2016.

[97] Yew Teck Tan, Abhinav Kunapareddy, and Marin Kobilarov. Gaussian process adaptive sampling using the cross-entropy method for environmental sensing and monitoring. In *International Conference on Robotics and Automation 2018*, pages 6220–6227, 05 2018.

[98] Edward Snelson and Zoubin Ghahramani. Sparse gaussian processes using pseudo-inputs. In *Advances in neural information processing systems*, pages 1257–1264, 2006.

[99] Lehel Csató and Manfred Opper. Sparse on-line gaussian processes. *Neural computation*, 14(3):641–668, 2002.

[100] Matthias Seeger, Christopher Williams, and Neil Lawrence. Fast forward selection to speed up sparse gaussian process regression. *Artificial Intelligence and Statistics 9*, 2003.

[101] Trong Nghia Hoang, Quang Minh Hoang, and Kian Hsiang Low. A unifying framework of anytime sparse gaussian process regression models with stochastic variational inference for big data. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 569–578, Lille, France, 2015. JMLR.org.

[102] Wenhao Luo, Shehzaman S Khatib, Sasanka Nagavalli, Nilanjan Chakraborty, and Katia Sycara. Distributed knowledge leader selection for multi-robot environmental sampling under bandwidth constraints. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5751–5757. IEEE, 2016.

[103] Jorge Cortes, Sonia Martinez, Timur Karatas, and Francesco Bullo. Coverage control for mobile sensing networks. *IEEE Transactions on robotics and Automation*, 20(2):243–255, 2004.

[104] Wenhao Luo and Katia Sycara. Adaptive sampling and online learning in multi-robot sensor coverage with mixture of gaussian processes. In *IEEE International Conference on Robotics and Automation (ICRA)*, May 2018.

[105] Stephanie Kemna, John G Rogers, Carlos Nieto-Granda, Stuart Young, and Gaurav S Sukhatme. Multi-robot coordination through dynamic voronoi partitioning for informative adaptive sampling in communication-constrained environments. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2124–2130. IEEE, 2017.

[106] Robert Zlot, Anthony Stentz, M Bernardine Dias, and Scott Thayer. Multi-robot exploration controlled by a market economy. In *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292)*, volume 3, pages 3016–3023. IEEE, 2002.

[107] Reid G. Simmons, David Apfelbaum, Wolfram Burgard, Dieter Fox, Mark Moors, Sebastian Thrun, and Håkan L. S. Younes. Coordination for multi-robot exploration and mapping. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, pages 852–858. AAAI Press, 2000.

[108] Weihua Sheng, Qingyan Yang, Song Ci, and Ning Xi. Multi-robot area exploration with limited-range communications. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, volume 2, pages 1414–1419. IEEE, 2004.

[109] Alessandro Marino, Gianluca Antonelli, Antonio Pedro Aguiar, António Pascoal, and Stefano Chiaverini. A decentralized strategy for multirobot sampling/patrolling: Theory and experiments. *IEEE Transactions on Control Systems Technology*, 23(1):313–322, 2015.

[110] Pratap Tokekar, Joshua Vander Hook, David Mulla, and Volkan Isler. Sensor planning for a symbiotic UAV and UGV system for precision agriculture. *IEEE Transactions on Robotics*, 32(6):1498–1511, 2016.

[111] Carl Edward Rasmussen and Malte Kuss. Gaussian processes in reinforcement learning. In *NIPS*, 2003.

[112] Robert Grande, Thomas Walsh, and Jonathan How. Sample efficient reinforcement learning with gaussian processes. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1332–1340, 2014.

[113] Evangelos A Yfantis, George T Flatman, and Joseph V Behar. Efficiency of kriging estimation for square, triangular, and hexagonal grids. *Mathematical Geology*, 19(3):183–205, 1987.

[114] Trevor Hastie, Robert Tibshirani, Jerome Friedman, and James Franklin. The elements of statistical learning: data mining, inference and prediction. *The Mathematical Intelligencer*, 27(2):83–85, 2005.

[115] Johan Wagberg, Dave Zachariah, Thomas Schon, and Petre Stoica. Prediction Performance After Learning in Gaussian Process Regression. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54, pages 1264–1272. PMLR, 20–22 Apr 2017.

[116] Michael L Stein. *Interpolation of spatial data: some theory for kriging*. Springer Science & Business Media, 2012.

[117] Thomas M Cover and Joy A Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

[118] Onur Tekdas and Volkan Isler. Sensor placement for triangulation-based localization. *IEEE transactions on Automation Science and Engineering*, 7(3):681–685, 2010.

[119] S. Arora. Nearly linear time approximation schemes for euclidean tsp and other geometric problems. In *Proceedings 38th Annual Symposium on Foundations of Computer Science*, pages 554–563, Oct 1997.

[120] Onur Tekdas, Deepak Bhadauria, and Volkan Isler. Efficient data collection from wireless nodes under the two-ring communication model. *The International Journal of Robotics Research*, 31(6):774–784, 2012.

[121] G. N. Frederickson, M. S. Hecht, and C. E. Kim. Approximation algorithms for some routing problems. In *17th Annual Symposium on Foundations of Computer Science (sfcs 1976)*, pages 216–227, Oct 1976.

[122] D. J. Mulla, A. C. Sekely, and M. Beatty. Evaluation of remote sensing and targeted soil sampling for variable rate application of nitrogen. In P. C. Robert, R. H. Rust, and W. E. Larson, editors, *Proceedings of the 5th International Conference on Precision Agriculture*, pages 1–15, Madison, USA, 2000. American Society of Agronomy.

[123] Carl Edward Rasmussen and Hannes Nickisch. Gaussian processes for machine learning (gpml) toolbox. *Journal of machine learning research*, 11(Nov):3011–3015, 2010.

[124] JK d C Pinto, M Masuda, LC Magrini, JA Jardini, and MV Garbelloti. Mobile robot for hot spot monitoring in electric power substation. In *2008 IEEE/PES Transmission and Distribution Conference and Exposition*, pages 1–5. IEEE, 2008.

[125] Hadi Ardiny, Stefan Witwicki, and Francesco Mondada. Autonomous exploration for radioactive hotspots localization taking account of sensor limitations. *Sensors*, 19(2):292, 2019.

[126] Taylor Moore. In the usa, robotics technology used at the tmi-2 cleanup and at other nuclear plants has prompted interest and shaped research on how robots might best be used. *IAEA BULLETIN*, page 31, 1985.

[127] Taylor Moore. In the usa, robotics technology used at the tmi-2 cleanup and at other nuclear plants has prompted interest and shaped research on how robots might best be used. *IAEA BULLETIN*, page 31, 1985.

[128] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

[129] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. *Machine Learning*, 47(2/3):235–256, 2002.

[130] C. Mellucci, P. P. Menon, C. Edwards, and P. Challenor. Source seeking using a single autonomous vehicle. In *2016 American Control Conference (ACC)*, pages 6441–6446, 2016.

[131] Esther Rolf, David Fridovich-Keil, Max Simchowitz, Benjamin Recht, and Claire J. Tomlin. A successive-elimination approach to adaptive robotic sensing. *CoRR*, abs/1809.10611, 2018.

[132] Roman Marchant and Fabio Ramos. Bayesian optimisation for intelligent environmental monitoring. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2242–2249, 2012.

[133] Weizhe Chen and Lantao Liu. Pareto monte carlo tree search for multi-objective informative planning. In *Proceedings of Robotics: Science and Systems*, FreiburgimBreisgau, Germany, June 2019.

[134] Yoonchang Sung, Deeksha Dixit, and Pratap Tokekar. Environmental hotspot identification in limited time with a uav equipped with a downward-facing camera. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 13264–13270, 2021.

[135] Amarjeet Singh, Andreas Krause, Carlos Guestrin, William Kaiser, and Maxim Batalin. Efficient planning of informative paths for multiple robots. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, IJCAI'07, page 2204–2211, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[136] David R Thompson, David S Wettergreen, and Francisco J Calderóon Peralta. Autonomous science during large-scale robotic survey. *Journal of Field Robotics*, 28(4):542–564, 2011.

[137] Stephanie Kemna, Oliver Kroemer, and Gaurav S Sukhatme. Pilot surveys for adaptive informative sampling. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6417–6424. IEEE, 2018.

[138] Sahil Garg and Nora Ayanian. Persistent monitoring of stochastic spatio-temporal phenomena with a small team of robots. *arXiv preprint arXiv:1804.10544*, 2018.

[139] Keyu Nie, Zezhong Zhang, Ted Tao Yuan, Rong Song, and Pauline Berry Burke. Efficient multivariate bandit algorithm with path planning, 2020.

[140] P. B. Reverdy, V. Srivastava, and N. E. Leonard. Modeling human decision making in generalized gaussian multiarmed bandits. *Proceedings of the IEEE*, 102(4):544–571, 2014.

[141] Chenxi Xiao and Juan Wachs. Nonmyopic informative path planning based on global kriging variance minimization. *IEEE Robotics and Automation Letters*, 2022.

[142] Lorenzo Pisani. Multi-objective pareto monte carlo tree search for informative path planning of planetary rovers. 2021.

[143] Pierre-Arnaud Coquelin and Rémi Munos. Bandit algorithms for tree search. In *Uncertainty in Artificial Intelligence*, 2007.

[144] Sylvain Gelly, Levente Kocsis, Marc Schoenauer, Michele Sebag, David Silver, Csaba Szepesvári, and Olivier Teytaud. The grand challenge of computer go: Monte carlo tree search and extensions. *Communications of the ACM*, 55(3):106–113, 2012.

[145] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

[146] Roman Marchant, Fabio Ramos, and Scott Sanner. Sequential bayesian optimisation for spatial-temporal monitoring. In *Proceedings of the Thirtieth*

Conference on Uncertainty in Artificial Intelligence, UAI'14, page 553–562, Arlington, Virginia, USA, 2014. AUAI Press.

[147] Jean-Yves Audibert, Sébastien Bubeck, and Rémi Munos. Best arm identification in multi-armed bandits. In *COLT*, pages 41–53. Citeseer, 2010.

[148] David Tolpin and Solomon Shimony. Mcts based on simple regret. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 26, pages 570–576, 2012.

[149] Felix Berkenkamp, Angela P Schoellig, and Andreas Krause. No-regret bayesian optimization with unknown hyperparameters. *arXiv preprint arXiv:1901.03357*, 2019.

[150] Shushman Choudhury, Nate Gruver, and Mykel J Kochenderfer. Adaptive informative path planning with multimodal sensing. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, pages 57–65, 2020.

[151] S.I. Roumeliotis and G.A. Bekey. Distributed multirobot localization. *IEEE Transactions on Robotics and Automation*, 18(5):781–795, 2002.

[152] R. Madhavan, K. Fregene, and L.E. Parker. Distributed heterogeneous outdoor multi-robot localization. In *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, volume 1, pages 374–381 vol.1, 2002.

[153] Donny Sutantyo and Paul Levi. Decentralized underwater multi-robot communication using bio-inspired approaches. *Artificial Life and Robotics*, 20(2):152–158, March 2015.

[154] Lidia Furno, Mikkel Cornelius Nielsen, and Mogens Blanke. Centralised versus decentralised control reconfiguration for collaborating underwater robots. *IFAC-PapersOnLine*, 48(21):732–739, 2015. 9th IFAC Symposium on Fault Detection, Supervision andSafety for Technical Processes SAFEPROCESS 2015.

[155] Masaharu Tanemura, Tohru Ogawa, and Naofumi Ogita. A new algorithm for three-dimensional voronoi tessellation. *Journal of Computational Physics*, 51(2):191–207, 1983.

[156] Wenhao Luo, Shehzaman S. Khatib, Sasanka Nagavalli, Nilanjan Chakraborty, and Katia Sycara. Distributed knowledge leader selection for multi-robot environmental sampling under bandwidth constraints. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5751–5757, 2016.

[157] J. Cortes, S. Martinez, T. Karatas, and F. Bullo. Coverage control for mobile sensing networks. *IEEE Transactions on Robotics and Automation*, 20(2):243–255, 2004.

[158] Jorge Cortes. Coverage optimization and spatial load balancing by robotic sensor networks. *IEEE Transactions on Automatic Control*, 55(3):749–754, 2010.

[159] Alyssa Pierson, Lucas C Figueiredo, Luciano CA Pimenta, and Mac Schwager. Adapting to sensing and actuation variations in multi-robot coverage. *The International Journal of Robotics Research*, 36(3):337–354, 2017.

[160] Francesco Bullo, Jorge Cortes, and Sonia Martinez. *Distributed Control of Robotic Networks: A Mathematical Approach to Motion Coordination Algorithms*. Princeton University Press, 2009.

[161] Antonio Barrientos, Julian Colorado, Jaime del Cerro, Alexander Martinez, Claudio Rossi, David Sanz, and Joao Valente. Aerial remote sensing in agriculture: A practical approach to area coverage and path planning for fleets of mini aerial robots. *Journal of Field Robotics*, 28(5):667–689, 2011.

[162] Wajahat Kazmi, Morten Bisgaard, Francisco Garcia-Ruiz, Karl Damkjær Hansen, and Anders la Cour-Harbo. Adaptive surveying and early treatment of crops with a team of autonomous vehicles. In *Proceedings of the 5th European Conference on Mobile Robots ECMR 2011*, pages 253–258, 2011.

[163] Amit Dhariwal, Gaurav S Sukhatme, and Aristides AG Requicha. Bacterium-inspired robots for environmental monitoring. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, volume 2, pages 1436–1443. IEEE, 2004.

[164] Matthew Dunbabin and Lino Marques. Robots for environmental monitoring: Significant advancements and applications. *IEEE Robotics & Automation Magazine*, 19(1):24–39, 2012.

[165] Michael Ouimet and Jorge Cortés. Collective estimation of ocean nonlinear internal waves using robotic underwater drifters. *IEEE Access*, 1:418–427, 2013.

[166] George P. Kontoudis and Daniel J. Stilwell. Fully decentralized, scalable gaussian processes for multi-agent federated learning, 2022.

[167] R. Zlot, A. Stentz, M.B. Dias, and S. Thayer. Multi-robot exploration controlled by a market economy. In *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, volume 3, pages 3016–3023 vol.3, 2002.

[168] Carlos Nieto-Granda, III John G. Rogers, and Henrik I. Christensen. Coordination strategies for multi-robot exploration and mapping. *The International Journal of Robotics Research*, 33(4):519–533, 2014.

[169] Jing Yuan, Yalou Huang, Tong Tao, and Fengchi Sun. A cooperative approach for multi-robot area exploration. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1390–1395, 2010.

[170] Daniel E. Soltero, Mac Schwager, and Daniela Rus. Generating informative paths for persistent sensing in unknown environments. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2172–2179, 2012.

[171] Alessandro Marino, Gianluca Antonelli, António Pedro Aguiar, António Pascoal, and Stefano Chiaverini. A decentralized strategy for multirobot sampling/patrolling: Theory and experiments. *IEEE Transactions on Control Systems Technology*, 23(1):313–322, 2015.

[172] Jorge Cortés. Area-constrained coverage optimization by robotic sensor networks. In *2008 47th IEEE Conference on Decision and Control*, pages 1018–1023. IEEE, 2008.

[173] Sonia Martinez, Jorge Cortes, and Francesco Bullo. Motion coordination with distributed information. *IEEE Control Systems Magazine*, 27(4):75–88, 2007.

[174] C. Belta and V. Kumar. Abstraction and control for groups of robots. *IEEE Transactions on Robotics*, 20(5):865–875, 2004.

[175] Qiang Du, Maria Emelianenko, and Lili Ju. Convergence of the lloyd algorithm for computing centroidal voronoi tessellations. *SIAM Journal on Numerical Analysis*, 44(1):102–119, 2006.

[176] Stephanie Kemna, John G. Rogers, Carlos Nieto-Granda, Stuart Young, and Gaurav S. Sukhatme. Multi-robot coordination through dynamic voronoi partitioning for informative adaptive sampling in communication-constrained environments. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2124–2130, 2017.

[177] Todd Hester and Peter Stone. TEXPLORE: real-time sample-efficient reinforcement learning for robots. *Machine Learning*, 90(3):385–429, October 2012.

[178] Marc C Kennedy and Anthony O'Hagan. Predicting the output from a complex computer code when fast approximations are available. *Biometrika*, 87(1):1–13, 2000.

[179] Paris Perdikaris, Maziar Raissi, Andreas Damianou, ND Lawrence, and George Em Karniadakis. Nonlinear information fusion algorithms for data-efficient multi-fidelity modelling. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 473(2198):20160751, 2017.

[180] Andreas Damianou. *Deep Gaussian processes and variational propagation of uncertainty*. PhD thesis, University of Sheffield, 2015.

[181] Andreas Damianou and Neil Lawrence. Deep gaussian processes. In *Artificial Intelligence and Statistics*, pages 207–215, 2013.

[182] Kurt Cutajar, Mark Pullin, Andreas Damianou, Neil Lawrence, and Javier González. Deep gaussian processes for multi-fidelity modeling. *arXiv preprint arXiv:1903.07320*, 2019.

[183] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

[184] Kirthevasan Kandasamy, Gautam Dasarathy, Junier B Oliva, Jeff Schneider, and Barnabás Póczos. Gaussian process bandit optimisation with multi-fidelity evaluations. In *Advances in Neural Information Processing Systems*, pages 992–1000, 2016.

[185] Matthew E Taylor, Peter Stone, and Yaxin Liu. Transfer learning via inter-task mappings for temporal difference learning. *Journal of Machine Learning Research*, 8(Sep):2125–2167, 2007.

[186] Alonso Marco, Felix Berkenkamp, Philipp Hennig, Angela P Schoellig, Andreas Krause, Stefan Schaal, and Sebastian Trimpe. Virtual vs. real: Trading off simulations and physical experiments in reinforcement learning with bayesian optimization. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1557–1563. IEEE, 2017.

[187] Philipp Hennig and Christian J Schuler. Entropy search for information-efficient global optimization. *Journal of Machine Learning Research*, 13(Jun):1809–1837, 2012.

[188] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30. IEEE, 2017.

[189] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

[190] Tobias Jung and Peter Stone. Gaussian processes for sample efficient reinforcement learning with rmax-like exploration. In *Proceedings of the European Conference on Machine Learning*, September 2010.

[191] Ronen I Brafman and Moshe Tennenholtz. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3(Oct):213–231, 2002.

[192] Joaquin Quiñonero-Candela and Carl Edward Rasmussen. A unifying view of sparse approximate gaussian process regression. *Journal of Machine Learning Research*, 6(Dec):1939–1959, 2005.

[193] Eli Bingham, Jonathan P. Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul Szerlip, Paul Horsfall, and Noah D. Goodman. Pyro: Deep Universal Probabilistic Programming. *arXiv preprint arXiv:1810.09538*, 2018.

[194] Marc Peter Deisenroth. *Efficient reinforcement learning using Gaussian processes*, volume 9. KIT Scientific Publishing, 2010.

[195] Naomi Ehrich Leonard, Derek A. Paley, Francois Lekien, Rodolphe Sepulchre, David M. Fratantoni, and Russ E. Davis. Collective motion, sensor networks, and ocean sampling. *Proceedings of the IEEE*, 95(1):48–74, 2007.

[196] Cristopher Salvi, Thomas Cass, James Foster, Terry Lyons, and Weixin Yang. The signature kernel is the solution of a goursat pde. *SIAM Journal on Mathematics of Data Science*, 3(3):873–899, 2021.

[197] Xiaotian Xu and Yancy Diaz-Mercado. Multi-robot control using coverage over time-varying domains: Extended abstract. In *2019 International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*, pages 179–181, 2019.

[198] Sung G Lee, Yancy Diaz-Mercado, and Magnus Egerstedt. Multirobot control using time-varying density functions. *IEEE Transactions on robotics*, 31(2):489–493, 2015.

[199] Murtaza Rangwala, Jun Liu, Kulbir Singh Ahluwalia, Shayan Ghajar, Harnaik Singh Dhami, Benjamin F. Tracy, Pratap Tokekar, and Ryan K. Williams. Deeppastl: Spatio-temporal deep learning methods for predicting long-term pasture terrains using synthetic datasets. *Agronomy*, 11(11), 2021.