UNIVERSITY OF CALIFORNIA, MERCED

Environmental Monitoring with Budget Constraints Using Reinforcement Learning

A dissertation presented for the degree of

Doctor of Philosophy

in

Electrical Engineering and Computer Science

by

Azin Shamshirgaran

2024

Committee in charge:
Professor David Noelle
Doctor Shijia Pan
Professor Stefano Carpin, chair

Submitted by:	Azin Shamshirgaran
Advisor:	Stefano Carpin, Ph.D. Electrical Engineering and Computer Science
Committee Members:	David Noelle, Ph.D. Cognitive and Information Sciences
	Shijia Pan, Ph.D. Electrical Engineering and Computer Science
	Stefano Carpin, Ph.D. (chair) Electrical Engineering and Computer Science

This dissertation is approved and accepted for publication by the committee.

David Noelle

Shijia Pan

Stefano Carpin

Abstract

This dissertation explores decision-making under uncertainty in robotics systems tasked with reconstructing a scalar field through sensing. In this task, each robot must determine its next decision considering surrounding uncertainties, and environmental and physical constraints. The complexity escalates in a multi-agent scenario, as each robot must not only assess its course of action but also predict and consider other robots' movements and plans. This is crucial to avoid collisions and prevent repetitive tasks among interacting agents. Our interest in this problem is motivated by applications in precision agriculture, where robots are used to collect measurements to estimate domain-relevant scalar parameters such as soil moisture or nitrates concentrations. In particular, we focus on the implementation of budget-aware algorithms, therefore casting our problem as an instance of constrained optimization, whereby the goal is to collect *good data*, while being subject to constraints on the available energy limiting the distance a robot can travel while fulfilling its mission. This problem is formally called informative path planning (IPP) and is NP-hard.

For this approach to be efficient, it is necessary for robots to coordinate their efforts to avoid unnecessary duplicate work or negative interference. As part of the measurement collection task, sample locations can be provided in advance, chosen by experts or randomly, or selected along the way in response to collected data. Central to our work is the necessity to perform task allocation being aware of the distance a robot can travel before its battery is depleted. From a practical standpoint, a robot running out of energy in the middle of its data collection mission is a major problem, as it will be necessary to manually recover it. Our task allocation strategy, therefore, focuses both on avoiding duplicate work and on managing the energy constraints.

The purpose of this dissertation is to address each of these variations of the IPP problem within the context of environmental map learning, in particular agricultural field map learning. Different offline and online learning based solutions are provided to address the IPP problem for single robot as well as multiple robots. The proposed algorithms are analyzed on real-world datasets, and simulations are constructed and shown to be efficient at guiding robots to sampling points under budget and environmental constraints. The feasibility and performance of a single robot solution method is evaluated and verified on a real Husky robot.

Related Publications

[74] A. Shamshirgaran and S. Carpin. "Reconstructing a spatial field with an au tonomous robot under a budget constraint". In 2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 8963–8970, 2022.

[76] A. Shamshirgaran, S. Manjanna, and S. Carpin. "Distributed multi-robot online sampling with budget constraints". In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), pages 12658-12664, 2024.

[75] A. Shamshirgaran and S. Carpin. "Environmental map learning with multi-robots". In Proceedings of the IEEE International Conference on Robotics and Automation, 2025 (to be submitted).

[22] Dechemi, D. Chatziparaschis, J. Chen, M. Campbell, A. Shamshirgaran, C. Mucchiani, A. Roy-Chowdhury, S. Carpin, and K. Karydis. "Robotic assessment of a crop's need for watering". IEEE Robotics and Automation Magazine (RAM), 30(4):52 – 67, 2023.

Acknowledgments

To begin with, I would like to thank Stefano Carpin for his continued support and advice as my PhD advisor and the opportunity to work in his robotics laboratory as a graduate student. Additionally, I would like to thank the other members of my advisory committee, Shijia Pan and David Noelle, for reviewing my dissertation.

I wish to thank the co-authors of each of my research papers and conference publications, which formed the basis of my research and extend my gratitude to the UC Merced community, of which I have been a member for four years.

I would also like to take this opportunity to express my gratitude to my family for their unwavering support throughout my entire life. I am forever grateful to have them as part of my life.

I also must acknowledge financial support by the USDA-NIFA under award number 2021-67022-33452 (National Robotics Initiative) and IoT4Ag Engineering Research Center funded by the National Science Foundation (NSF) under NSF Cooperative Agreement Number EEC-1941529.

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

1.1 Purpose

Over the past few years, there has been a steady increase in the use of collaborative team of robots for gathering information across multiple domains. Examples of applications include search and rescue to locate survivors [58], ocean exploration to map underwater terrain and monitor water quality parameters such as pH and chlorophyll [85], and data gathering in agricultural settings for pesticide spraying, seed sowing, as well as farm monitoring and sampling [22]. Figure 1.1 displays the Husky robot platform gathering leaf temperature data from a pistachio orchard using a multi-spectral camera. The question arises as to why we need to collect samples from farms and orchards?

In light of the expected rise in world population, food supply must be ensured in an urgent manner. In order to accomplish this, agricultural yields and water productivity must be increased worldwide. In spite of this, water is a limited resource, and global population growth and climate change are putting unprecedented pressure on it. It is therefore necessary to implement precision automatic irrigation technologies that take advantage of control theory in order to ensure sustainable and rational use of water in irrigated crops.

A few decades ago, irrigation was done in so many ways, such as basin irrigation system or flood irrigation system. In basin method, water is ponded over the entire surface of the soil [104]. These days, there is a need for a more rational approach to irrigation optimization. For instance, an automatic irrigation system in which water is applied directly to plants' roots by means of applicators (orifices, emitters, porous tubing, perforated pipes, etc). A sprinkler irrigation system applies water under pressure through perforated pipes or nozzles to form a spray pattern [104]. In automatic irrigation, the key concept is feedback which is a mechanism, process, or signal that controls the irrigation system. In the field of automatic irrigation, measurements of soil (soil water potential), plant (stem water potential) and atmosphere variables are used to determine the next irrigation dosage based on the consequences of previous actions [66]. Physical sampling can be quite laborious and often vary among different types of crops. Given the growing agricultural workforce shortages, the labor-intensive nature of these measurements poses a severe limitation on how many samples can be collected and analyzed, and restricts how growers and farm consultants can assess local conditions and optimize operations in support of sustain-

able crop production. This is particularly critical in high-value perennial crops, such as avocados, citrus, almonds, nuts, and vines. Hence, robotics and automation technology can be employed to support physical sampling. In this context, the capability of deploying a coordinated team of robots to collect data is instrumental, as a team of robots can collect more data per time unit, and also offers increased robustness to individual failures.

For this approach to be efficient, it is necessary for robots to coordinate their efforts to avoid unnecessary duplicate work or negative interference. As part of the measurement collection task, sample locations can be provided in advance chosen by experts, or randomly, or selected along the way in response to collected data. Central to our work is the necessity to perform task allocation being aware of the distance a robot can travel before its battery is depleted. In particular, we focus on the implementation of budget-aware algorithms, therefore casting our problem as an instance of constrained optimization, whereby the goal is to collect *good data* (in a sense to be formally defined later), while being subject to constraints on the available energy limiting the distance a robot can travel while fulfilling its mission.

Figure 1.1: Robot moving autonomously at pistachio orchard.

1.2 Overview of Contributions

The purpose of this dissertation is to address the challenges of decision-making under uncertainty in robotics systems tasked with reconstructing a scalar field through sensing. A multi-robot informative path planning (IPP) method aims to select paths and sequence of sampling locations for each robot that maximize the quality of the reconstructed scalar field while do not exceed the travel budget. In this setting, each robot must determine its next decision considering surrounding uncertainties, and environmental and physical constraints. The complexity escalates in a multi-agent scenario, as each robot must not only assess its course of action but also predict and consider other robots' movements and plans. This is crucial to avoid collisions and prevent repetitive tasks among interacting agents. This dissertation is motivated by improving agricultural efficiency, but the algorithms presented can also be applied to a wide range of scenarios, such as warehouse robot navigation.

This dissertation and its related publications make five major contributions. First, informative path planning (IPP) within a predefined budget in a stochastic environment for a single robot is discussed. Second, three offline and online learning based algorithms are presented, which are capable of solving these task planning problems efficiently. These algorithms describe how to select the sequence of sampling locations to yield an accurate Gaussian Process (GP) reconstruction as well as to determine when the exploration should be concluded and move to final docking point. Third, an extension to multi robot IPP within a predetermined budget is discussed, and a set of heuristic learning based algorithms are presented which provide distributed online policies for task planning in different environments. Fourth, a strategy is proposed to update sampling locations on the fly to leverage data collected during the mission and predict the next decision of other robots in the team to prevent multiple robots from visiting and collecting data samples at the same location at the same time. Fifth, a real-world experiments with a Husky robot are presented to verify the proposed method.

In the remaining six chapters of this dissertation, different aspects of efficient task planning solutions are discussed to solve the IPP problem. Chapter 2 provides a summary of related literature for the specific types of problems discussed throughout this dissertation. Chapter 3 describes an offline Q-Learning solution to solve the single robot IPP problem. Chapter 4 discusses a budget-aware, online method based on Monte Carlo Tree search (MCTS) for solving the single robot IPP problem. In Chapter 5, the IPP problem is extended to multi-agent scenario focusing on avoiding collisions, revisiting the same locations while managing budgets. A real-world experiments with a Husky robot are presented in Chapter 6. Finally, Chapter 7 offers concluding remarks on the methods presented in this dissertation and outlines a few areas for future research.

Chapter 2

Related Work

2.1 Robots Application in Agriculture

There has been a labor crisis in American agriculture for decades, resulting in the loss of $3.1 billion worth of food over the past decade [78]. According to the USDA, farm labor employment has declined by 75 percent over the past 70 years [78]. In addition to being laid off by the pandemic, employees are likely to migrate to industries that are highly attractive to them, such as technology or health care [26]. With this introduction, it is imperative that the agricultural industry be able to address this issue in a more efficient manner.

Autonomous robots are finding more and more applications in agricultural industry [99, 30]. Perhaps the most obvious tasks that would require ground robots for automation are tasks that involve direct interaction with the plants and soil. One such task that could be enhanced by robotics is the harvesting of fruit by robots [16]. Another application is pesticide spraying, which is a recurring task in nearly every growing operation. Aside from spraying for pests, robots have been developed that are capable of autonomously weeding orchards and vineyards where the robot can remove weeds that are growing between vines and within rows without harming the vine trunks [59]. Ground robots can also be used by farmers to sow seeds into the ground [99]. Finally, ground robots can be used as part of a farm monitoring and sampling system. Figure 2.1 shows the UC Merced, Robotics lab Husky robot which has been retrofit with a soil moisture sensor that can be inserted in the soil to collect data at preassigned locations, or at locations decided by the robot on-the-fly. These samples must be collected over vast spatial domains and the utilization of robots allows better use of the dwindling agricultural workforce. In these applications it is often necessary to frequently collect data to monitor parameters such as water, nitrates, nutrients, or to determine soil quality, or crop estimates [74].

Other non-agricultural information gathering can also be achieved with multi-robots. Examples of applications include search and rescue to locate survivors [58], ocean exploration to map underwater terrain [85].

Agricultural robotics and automation technology play an increasingly critical role across several crop production stages to improve sustainability (e.g., water use optimization). Robotics has been applied for remote and proximal sensing, as well as physical sampling [22]. The procedure of physical sampling and follow-on analysis of plant specimens such as leaves or shoots, often constitutes the only accurate way to

Figure 2.1: The Husky robot has been retrofit with a soil moisture sensor that can be inserted in the soil to collect data (the soil moisture probe is visible on the left).

measure some essential parameters that affect crop production, such as stem water potential (SWP) and nitrogen content which help determine a crop's need for watering and can only be accurately measured through the physical collection and analysis of specimens such as leaves or shoots.

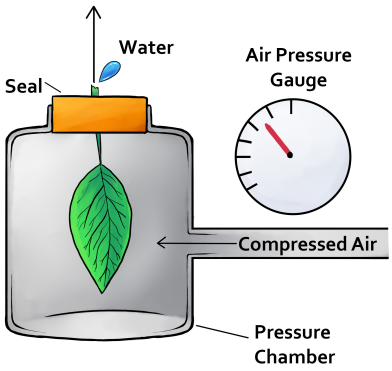
However, both physical sampling and specimen analysis can be quite laborious and often vary among different types of crops. Given the growing agricultural workforce shortages [12], the labor-intensive nature of these measurements poses a severe limitation on how many samples can be collected and analyzed, and restricts how growers and farm consultants can assess local conditions and optimize operations in support of sustainable crop production. This is particularly critical in high-value perennial crops, such as avocados, citrus, almonds, nuts, and vines. Hence, robotics and automation technology can be employed to support physical sampling and specimen analysis.

SWP is a metric frequently used by agronomists and growers to optimize irrigation schedules for crops [28], with the intent of reducing water waste and maximizing profits [97, 70]. The current industry standard for obtaining SWP measurements is the Scholander pressure chamber method [71], which involves the insertion of a leaf sample into a pressure chamber with its stem's excised end exposed (Figure 2.2 (a)).

After the sample is secured, a human operator slowly activates a valve to pressurize the chamber while also observing the water expression at the end of the exposed stem through a magnifying glass (Figure 2.2 (b)) [22].

The pressure necessary to force water out of the stem determines the SWP. The process measures the capacity of the cells to retain water by pressurizing the leaf. The less free water there is in the plant, the greater the pressure required to cause the leaf to exude water.

When taken in pre-dawn conditions (i.e. performed before sunrise) and plant



(a)

(b)

Figure 2.2: (a) Working principle of the pressure chamber. (b) Manual visual inspection is currently performed in the field for SWP analysis [22].

stomata are closed, the measurement is at equilibrium with soil moisture conditions. Subsequent measurements can then precisely determine water deficit, and thus irrigation demand to meet evapotranspiration loss. A precise measurement of the SWP is essential to assess the water deficit of the plant and to therefore adjust irrigation. Given that 80% of managed freshwater in the US is consumed by agriculture [100] and that evapotranspiration model estimates widely diverge [36], even modest improvements in irrigation practices can have huge impacts, especially in the semi-arid southwestern US that periodically undergoes drought periods while providing a large fraction of fruits and vegetables.

Some alternative methods have been proposed in recognition of the labor-intensive steps involved in measuring SWP using the pressure chamber method. Some rely on remote sensing using spectral reflectance or multi-spectral imaging to determine water potential [105, 98]. In addition to their non-invasive characteristics, these methods are scalable since they are intended to replace the pressure chamber for mass assessment in a faster manner. Despite their initial promise, these methods are highly sensitive to an intractable external factor, namely light variability due to weather and solar movement. Zhao et al. [105] mounted multi-spectral cameras on a small UAV to take high-resolution multi-spectral images of orchards for SWP prediction using the canopy Normalized Difference Vegetation Index (NDVI), but mentioned the high variability in data collected from different flights within the same day due to solar motion. Vila et al. [98] used remote sensors and spectral reflectance as a proxy for SWP measurements but had a low correlation coefficient and thus concluded this method cannot replace a pressure chamber method. To minimize water loss through transpiration, it is recommended to bag leaf samples with reflective foil bags for at least ten minutes prior to excision [55]. While earlier literature was recommending making the measurements on the spot and avoid transportation of samples, the work of [65] suggested immediate storage of excised leaf samples in a cold and moist environment helps stabilize the

sample's water potential and preserve the condition for hours or even days, depending on the plant species [65].

Since the concepts of spectral reflectance and imaging may not be mature yet for this application, we must seek to directly automate the pressure chamber method for assessing SWP. It is necessary to develop robotics and automation technology tools that can help assess watering needs for tree crops. The primary focus lies in automating the process of measuring SWP.

In one of our works [22], a mobile robot autonomously selects multiple measurement locations for sample collection. The in-house designed end effector mounted on the robot's arm will then be used to retrieve the leaves. In addition, we used RGBD and lidar points clouds to determine the potential leaf. When the robot has reached the SWP analysis station, a human operator loads the samples into a machine-vision-assisted pressure chamber to determine the exposed leaf stem's wetness. Figure 2.3 outlines the overall idea.

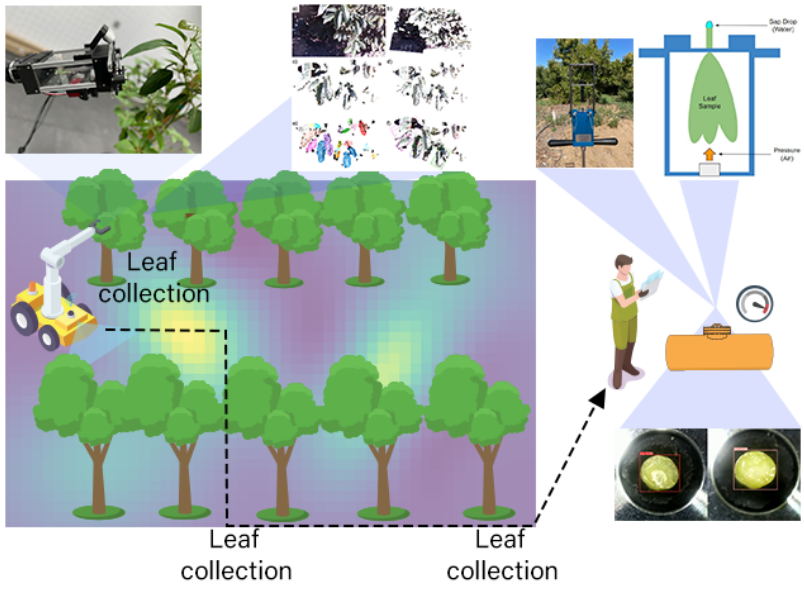


Figure 2.3: A mobile robot autonomously selects multiple measurement locations for sample collection and retrieves leaves. These are later on conveyed to a human operator who uses a pressure chamber to determine the pressure defining the SWP [22].

There are several factors that need to be considered in order for these automated sample collection systems to be efficient and practical. In [42], the authors provided a graphical user interface (GUI) for selecting waypoints and viewing samples. For selecting informative waypoints, they compared human performance with adaptive informative sampling algorithms. For selecting waypoints, humans and simulated robots receive the same information. Using two entropy-based optimization criteria, the simulated method iteratively selected waypoints using Gaussian Process regression.

Among the challenges for utilizing robots in precision agriculture are the need to

interact with human workers and machinery, the size of farms, and the limited battery life or fuel sources of robots. It can also be challenging to navigate in these environments. The survey [5] outlines some techniques developed specifically for farmland localization and mapping and provides an overview of the challenges associated with them. This recent review [69] provides a brief overview of robot path planning and routing algorithms used in agriculture.

2.2 Kriging and Gaussian Process Model

There are many physical, chemical, and biological processes that interact to produce environmental features, such as soil moisture. The interactions between these processes are so complex that the variation may seem random. Because of the complexity and incomplete understanding of the processes, deterministic or mathematical solutions to quantify the variation are not feasible. Consequently, random variables and random processes have been introduced to suitably model these uncertainties.. For example, soil moisture, element concentrations in soil, and air temperatures are considered as spatial random variables [27, 15].

A Gaussian Process (GP) is defined as a collection of random variables with joint Gaussian distribution [57]. Let us consider y is the observed target value and modeled as follows:

$$\begin{aligned} y =& f(x) + \epsilon \\ \text{where } \epsilon \sim& \mathcal{N}(0,\, \sigma_n^2) \end{aligned} \tag{2.1}$$

where x is the input vector and and f is the function value. We assume that the observation value y differ from function value $f(x)$ by additive Gaussian distribution noise with zero mean and variance σ_n^2. According to this modeling approach, $f(x)$ is modeled as Gaussian Process, $f(x) \sim \mathcal{GP}(m(x), k(x, x'))$ which is specified by its mean function $m(x)$ and covariance function $k(x, x')$ [77].

GP assumes that $p(f(x_1), ..., f(x_n))$ is jointly Gaussian with mean $\mu(X)$ and covariance $\Sigma_{ij} = k(x_i, x_j)$. For any data points, this process defines a joint Gaussian distribution $p(f|X) = \mathcal{N}(f|\mu, \Sigma)$. Now, the goal would be to predict the $f(x)$ for a value at locations that has not been visited [82].

Figure 2.4 shows the 2D and 3D mesh plot of the Gaussian Process. As it can be seen, it is a mixture of four Gaussians with different centers and covariance matrices. To reconstruct the GP accurately, the robot must visit all peaks (high values) as well as valleys (low values).

Prediction with Noise-free Observations

In this case, the observations are $\{(x_i, f_i)|i = 1, 2, ..., n\}$. The ϵ in equation 2.1 is considered as zero. The joint distribution of the training outputs f and test outputs

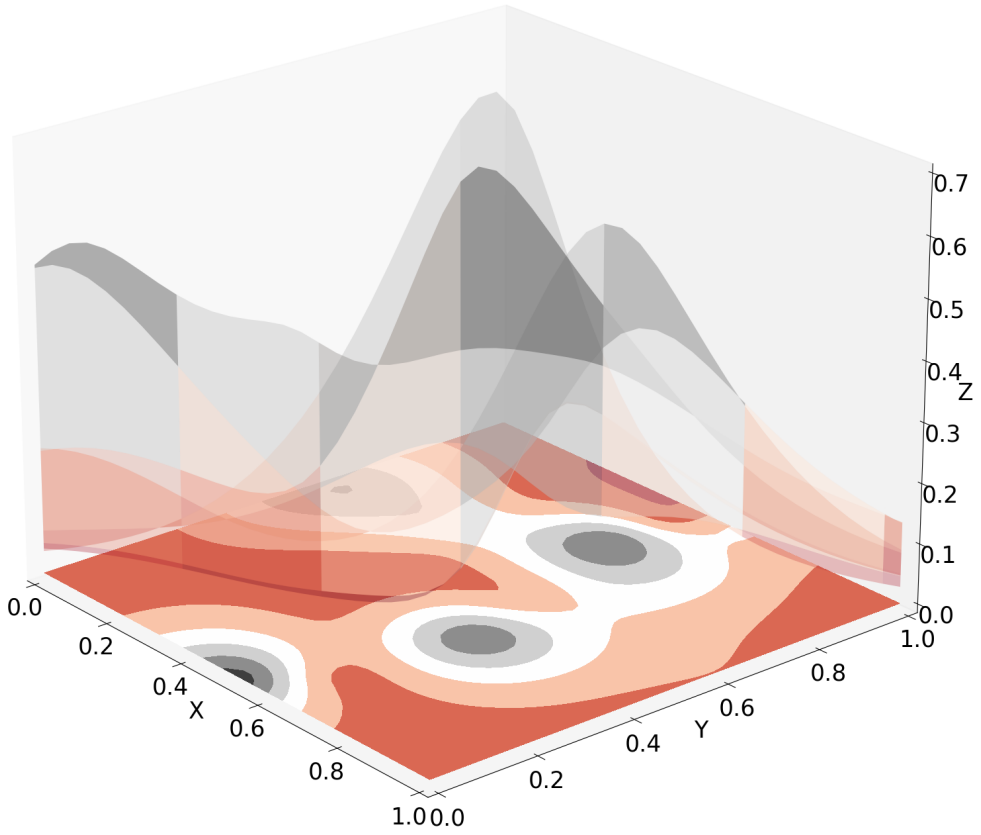


Figure 2.4: 2D and 3D mesh plot of the Gaussian Process.

f_* according to the prior is defined as follows:

$$\begin{bmatrix} f \\ f_* \end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} K(X,X) & K(X,X_*) \\ K(X_*,X) & K(X_*,X_*) \end{bmatrix}\right)$$

where X and X_* are training and test inputs. If there are n training points and n_* test points then $K(X, X_*)$ denotes the n x n_* matrix of the covariances evaluated at all pairs of training and test points, and similarly for the other entries $K(X, X)$, $K(X_*, X_*)$ and $K(X_*, X)$. So, we can define the noise-free predictive posterior as follows [102]:

$$\begin{aligned} p(f_*|X_*, X, f) = \mathcal{N}(&K(X_*,X)K(X,X)^{-1}f, \\ &K(X_*,X_*) - K(X_*,X)K(X,X)^{-1}K(X,X_*)) \end{aligned} \tag{2.2}$$

Prediction with Noisy Observations

In this case, Equation 2.1 would become as follows:

$$\begin{aligned} y &= f(x) + \epsilon \\ \epsilon &\sim \mathcal{N}(0,\, \sigma_n^2) \\ cov(y) &= K(X, X) + \sigma_n^2 I \end{aligned} \tag{2.3}$$

The joint distribution of the training outputs f and test outputs f_* according to the prior is defined as follows:

$$\begin{bmatrix} f \\ f_* \end{bmatrix} \sim \mathcal{N}\left(0,\, \begin{bmatrix} K(X,X) + \sigma_n^2 I & K(X, X_*) \\ K(X_*, X) & K(X_*, X_*) \end{bmatrix}\right)$$

where X and X_* are training and test inputs. If there are n training points and n_* test points then $K(X, X_*)$ denotes the n x n_* matrix of the covariances evaluated at all pairs of training and test points, and similarly for the other entries $K(X, X)$, $K(X_*, X_*)$ and $K(X_*, X)$. Once again, we can define the noisy predictive posterior as follows [102]:

$$\begin{aligned} p(f_*|X_*, X, y) = \mathcal{N}(&K(X_*, X)([K(X, X) + \sigma_n^2 I])^{-1} y, \\ &K(X_*, X_*) - K(X_*, X)[K(X, X) + \sigma_n^2 I]^{-1} K(X, X_*)) \end{aligned} \tag{2.4}$$

Modeling environmental phenomena with these predictions has proven highly effective [21].

2.3 Single-Robot Informative Path Planning

With an increasing population, decreasing arable land, climate change, and a declining skilled workforce, supplying food on a global scale is becoming increasingly difficult. As a result, reducing the use of water and agrichemicals while increasing productivity becomes increasingly important. Precision agriculture can be defined as "the matching of agronomic inputs and practices to localized conditions within a field and the improvement of the accuracy of their application" [25]. Essential to this paradigm is accurate assessment of multiple quantities with spatiotemporal variations, like evapotranspiration, nitrogen concentration, and others.

A key problem in IPP is how to handle the tradeoff originating by the necessity to collect numerous samples while being constrained by how far a robot can travel before its battery is depleted and needs recharging. In this context, planning a path to ideally collect *the best samples* while not running out of fuel is of fundamental importance. Informally speaking, the goal of IPP is to define a path through a set of sampling locations allowing for the best estimate of the parameter being measured. Depending on the application, sampling locations may be preassigned (and then the robot has to select a subset of them), or may be determined on the fly. Owing to the

intrinsic computational complexity of the problem, one has to settle for collecting a sub-optimal subset of samples.

A distribution of spatial data can be modeled by a Gaussian Process (GP) described in Section 2.2, and this approach is often used in agriculture [84, 81]. Combined with this model, metrics such as mutual information (MI), entropy, or variance can be used to guide the process of collecting samples to estimate a GP of the field. While the IPP problem has been extensively studied in the past, we consider a harder version where we also manage an assigned energy budget that is influenced by stochastic travel costs. This makes this problem different and harder than the orienteering problem that have been recently studied in a similar, yet simpler setting emerging in precision agriculture [92]. This problem is harder because we investigated instances where rewards are either incrementally discovered or reassigned comparing to orienteering where the reward is known upfront. The reason that we consider the stochastic environment is because agricultural fields may change from time to time, which will affect the cost of travel.

As path planning is an integral part of this process, in this section we review selected works addressing some of the issues related to single robot IPP and path planning.

2.3.1 Single-Robot IPP and Learning Based Methods

IPP can be solved in several ways, one of which is learning based methods. In [101], the authors used Reinforcement Learning (RL) to solve the IPP problem. The problem is defined as visiting a set of sample locations with a robot with a travel budget. The nearest sample location with shortest path is selected and the reward is assigned considering the budget. However, the authors did not consider a stochastic and dynamic environment and exclusively choose the shortest path for the next sampling location. In [44], the authors developed a constraint double Q-learning algorithm to find the path for an aerial vehicle. To prevent the divergence issue of Q-learning, two utility functions Q_A and Q_B were developed for decoupling action selection from action evaluation and updating each of these Q-functions using the value of the other one and choosing the action based on the argmax of the other Q-function.

Various approaches can be used to solve reinforcement learning problems in which constraints must be met. Lagrangian methods include constraints in the loss function, while another method restricts the action space directly before choosing the next action. Our methods use a reward shaping technique. In [29], the authors used the Lagrange multiplier method to combine the cost function with constraint in order to determine the next action and analyze constrained Q-learning for robot path planning in a grid environment to avoid the trap. To demonstrate that their proposed method was able to find the better path, resulting in better return values in each episode, they compared their method with baseline SARSA and Q-learning. Using this method, there is no guarantee that the agent will satisfy the constraints

mentioned in this problem at all points, which may result in undesirable behavior. In [37], the authors proposed an off-policy reinforcement learning framework in which the action space is limited in the Q-update in order to learn the optimal Q-function for the constrained Markov Decision Process (CMDP). First, the safe set of states is calculated for all constraints, then the deep Q-learning algorithm is applied. In on-policy learning, the $Q(s, a)$ function is learned from the actions the agent takes using its current policy $\pi(a|s)$. In off-policy learning, the $Q(s, a)$ function is acquired by taking different actions (for example, random actions). In [88], the authors integrate the GP upper confidence bound (GP-UCB) with the Cross-Entropy method for adaptive sampling in environmental sensing. This approach is designed to minimize the number of sampling points needed while maximizing the accuracy of information and environmental monitoring.

For single-robot and multi-robot informative path planning, Monte Carlo tree search (MCTS) has recently gained a great deal of attention. The goal of MCTS is to approximate the value of the actions that could be taken from the current state and this is accomplished through iteratively building a search tree. Afterwards, the best action will be chosen based on the tree policy [14]. In [19], a multi-objective informative planning method referred to as Pareto Monte Carlo tree search was introduced, which allows the robot to handle exploration versus exploitation based on a multi-objective tree search model. The Pareto MCTS finds the optimal actions for the current state until a given time budget is exhausted. The node selection problem is treated as a multi-objective multi-armed bandit problem. For each child node, the Pareto Upper Confidence Bound (Pareto UCB) is computed. An approximate Pareto optimal set is then constructed. In the end, the best child node is randomly selected from the Pareto optimal set. Their action selection process does not take the budget into account.

In [67], deep reinforcement learning (DRL) was used to train an unmanned aerial vehicle (UAV) to make decisions based on sensory data in order to improve information collection rates and reduce mission time. They combined MCTS with a convolutional neural network (CNN) to learn information-rich actions in data collection operations. The implementation of DRL for real-time UAV path planning can be computationally expensive, and the trained model may not generalize well in real situations. It is easy to generalize our MCTS-based model to any environment because it does not require training. In our problem, we address the selection of next sampling location with learning methods where the robot itself learns to choose the next sample point based on assigned budget, consumed and remaining energy, the information and reward it gets by visiting each location and the stochastic noise of the environment.

2.3.2 Single-Robot IPP and Other Methods

Sampling based path planning methods such as rapidly exploring random trees (RRT-RRT*), rapidly exploring random graphs (RRG*), rapidly exploring informa-

tion gathering (RIG*) and probabilistic roadmaps (PRM*) have been adapted to tackle the IPP problem [20]. In [31], authors proposed a RIG-based algorithm for incremental information gathering. They used mutual information as an information criteria to maximize the information gathering with respect to the budget constraint. In [34], rapidly exploring information gathering (RIG) algorithms are introduced to solve the informative motion planning problem using iterative sampling. The authors proposed three different RIG algorithm: RIG roadmaps to build up a roadmap of possible trajectories; RIG tree to extend a tree of possible trajectories with a fixed start point; and RIG graph to build a graph of solution trajectories with a fixed start point. One of the limitations of these methods is the assumption of the availability of a suitable near point for graph expansion which in some cases may result in task failure. On the other hand, sampling-based methods have the advantage that they do not require to discretize the environment as they work on continuous space. In [41], probabilistic roadmaps (PRM), rapidly exploring random graphs (RRG) and rapidly exploring random trees (RRT) algorithms are adapted as new algorithms. The nearest neighbour location will always be selected according to various criteria. For instance, based on k-nearest algorithm or among k random samples within a circle of radius r. The robot moves to the selected location if it is feasible and not blocked by obstacles. As in the previous papers, they also did not take the robot's budget into account.

In [10], graph-based IPP for a single robot is discussed. The problem is finding a path in a graph G that maximizes a submodular function f of the nodes visited by a path from a node s to a node t. In the recursive greedy algorithm, the gain obtained by adding a new partial path to the current set is recursively optimized. At each step of recursion, all possible nodes for the path are tried, as well as all possible ways to split the budget between the halves of the path. The algorithm calls itself recursively to find the best path for the first half, then fixes that half and solves for the best path for the second half.

An alternative approach to solve the problem of IPP is to use the model of the environment to place the sensors and plan a path for a robot. This approach is more similar to what we studied in [74], where we assumed that a set of potential sampling locations is given upfront. GP is a powerful Bayesian approach used to model different phenomena in natural environments [84]. Different kernels can be used to model prior assumptions about the underlying function being estimated [72]. After modeling the spatial field, one can then plan the path based on the model. In [48, 33] the authors proposed a method for strategically placing sensors to maximize the amount of useful information collected about the environment while considering limitations on communication bandwidth or energy consumption. The greedy improvement algorithm continues to add sensors until it reaches the near-optimal level of information. In [49], they studied the problem of sensor placement based on maximizing mutual information (MI) and it was shown that using MI criteria leads to a lower number of sensors readings. In [47], the authors extensively discussed the submodularity property of IPPs. Information gathering problems often involve submodularity which implies as diminishing returns—meaning as a subset of sensors grows, the marginal gain from

adding an element decreases. Therefore, there must be a trade-off between information gathered and number of sensors. In IPP, submodular functions are utilized to select a subset of items (such as locations or data points) to maximize information. There is a limit to the number of sensors they can place, but including failure-tolerant methods would be a good addition.

In [84], the authors study how to balance between the amount of sensing resources (e.g., number of deployed robots, energy consumption, mission time) and the quality of data collected. To this end, they formulated a constrained optimization problem imposing a bound on the variance of the estimated field. The problem is then solved finding measurement locations, planning a tour for a single robot to visit those measurement locations and finally planning tours for multiple mobile robots. GP is used to model the spatial parameter being estimated and later the traveling salesperson with neighborhoods (TSPN) is used to find the best path for a single robot to visit the location of interest. Finally, the multi robot version of the algorithm is proposed to save operating time. This solution however, does not consider energy constraints.

In [62], an online IPP approach that adjusts the UAV's course according to feedback from active classifications is presented. During this process, evolutionary optimization and IPP techniques are combined to prioritize regions that are most informative for classification in continuous 3D space. In [63], the authors introduced a 3D path planning framework that prioritizes informative portions of the terrain during terrain monitoring with UAV. The proposed solution maximizes the information collected about the terrain by optimizing the path. Their model has been extensively tested in simulations and real-world experiments. Even though they considered a time as a constraint in their method, it is not explicitly used in training the UAV for action selection. In [11], the authors explored the problem of reconstructing an unknown environment with a UAV based on Bayesian optimization measurements. The UAV is provided with a mechanism that integrates prior beliefs it may have regarding anomaly locations while allowing it to make corrections and refine its beliefs on the fly as more information is gathered. The energy constraint is not taken into account in this method. In [7], an information-based controller guides a mobile robot through an a priori unknown environment in an autonomous exploration problem. Based on its estimation of which location will provide the most useful information, the robot determines where to measure next in its current field of view using MI. However, this method does not take budgets into account when selecting candidates.

2.4 Multi-Robot Informative Path Planning

An important variation of informative path problems and environmental map learning is multi robot IPP, which has wide applicability in many real-life situations.

As we discussed in Section 2.1, robots are expected to play a vital role in the implementation of farm monitoring systems in support of *precision agriculture*. Key to precision agriculture is the ability to perform scalable data collection on demand. In this context, the capability of deploying a coordinated team of robots to collect

data is instrumental, as a team of robots can collect more data per time unit, and also offers increased robustness to individual failures [103]. For this approach to be efficient, it is necessary for robots to coordinate their efforts to avoid unnecessary duplicate work or negative interferences. As part of the measurement collection task, sample locations can be provided in advance chosen by experts, or randomly, or selected along the way in response to collected data.

2.4.1 Multi-Robot IPP and Learning Based Methods

For multi-robot informative path planning, Monte Carlo tree search (MCTS) has recently gained a lot of attention. In [35, 9], the authors proposed a method that combines Gaussian Processes (GPs) and MCTS to monitor the environment, while in [73], the authors study the 2D area exploration with modifying MCTS. The goal is to minimize the time required to cover the areas of interest by first creating a tree of possible actions for the robots to take, and then choosing the one that maximizes exploration gain. To adapt to changes in the environment as it is explored, the underlying tree structure is continuously adjusted by changing the feasible actions for each node. The authors of [73] also explored the decentralized multi-robot scenario. Each robot shares its plan with another and each robot stores the shared information in a buffer. At the next round of MCTS computation, the robot will choose one of each robot's shared information from the buffer and it will update the map (reward function) based on that.

In [85], the goal is to plan the paths to identify the hotspots in unknown 2D environments with single as well as multiple mobile robots. The spatial field of the environment is modeled using Gaussian Processes whose covariance function has unknown hyperparameters. Their adaptive GP-MCTS monotonically changes the hyperparameters of the GP model to capture more complex function candidates. In the multi-robot version, they divide the environment using Voronoi partitioning whose center has been computed based on the GP model. Our method differs from these because we do not aim to only identify hotspots, but we rather aim at estimating an unknown scalar field over its entire domain. The term hotspots refers to regions in the 2D space where the predicted mean of the GP is particularly high. This, in turn, leads to the definition of a different reward function guiding the selection of sampling locations.

In [52, 53], a decentralized approach is proposed using a policy gradient method for multirobot environmental monitoring and sampling. To encourage robots to spread away from each other, they are rewarded based on their distance from each other. Furthermore, the method considers a communication range between robots to exchange locations and the history of previous locations with co-working robots. In [60], a distributed adaptive sampling method for multi-agent scenarios was proposed which is robust to the failure of robots and communication. To estimate the optimal policy, deep neural networks and policy gradient methods are used. These works, however, do not explicitly incorporate limits on the distance traveled by robots. In [13],

they considered the problem of reconstructing a spatial field using multiple robots, Gaussian Processes, and MCTS. As part of this work, robots communicate with one another and send their current location and observations to other robots and explore the spatial properties of GPs to attempt to spread the robots in the environment.

In [24], researchers proposed a decentralized Markov Decision Process (Dec-MDP) for robots to visit a subset of locations to maximize information collection. In order to solve this problem, they used GP and the value iteration algorithm. Initially, robots have the same GP learned from the initial training set, then they update it with new measurements. Rather than forcing robots to stay in continuous communication, their strategy relies on robots' opportunistic communication patterns. Choosing actions is not influenced by budget constraints; instead, more sample points are added until the budget runs out.

In many practical cases, the state space and the action space are enormous. In that case, finding the optimal policy is not possible, so the goal instead would be to find a proper approximate solution using limited computational resources [86]. In [60], they used deep neural networks and policy gradient methods in order to estimate the optimal policy, however, they did not explicitly incorporate budgetary considerations. We would use deep neural networks to find the best policy considering the budget limit.

2.4.2 Multi-Robot IPP and Other Methods

Having discussed learning-based methods to solve the multi-robot IPP in Section 2.4.1, this section explores other methods. In [43], the authors propose a dynamic Voronoi approach, where robots repeatedly compute Voronoi partitions and each robot performs sampling within its partition. Although this method shares the exploration task efficiently between robots, the iterative recomputation of Voronoi regions may lead to many unnecessary motions that are problematic in our application where robots are subject to a limited travel budget. In [23], a method was described for mapping unknown environments collaboratively and efficiently by multi-robot systems. Map knowledge is used to guide robots to allocated regions. Based on the information collected, robots can modify their exploration strategies in real-time. In order to maximize coverage and accuracy of the generated map, it is important to decide when and where to move next. To reduce redundancy and improve overall exploration efficiency, they share information and coordinate actions. It is likely that region-based sampling strategies will require significant computational resources since these regions must be recomputed at every stage. In [54], a distributed approach for multi-robot sampling based on Voronoi partitioning and GP was proposed. The goal of the paper is choosing the next sampling point in a way to reduce overall uncertainty about the knowledge of a given field. To distribute the objective among the robots, Voronoi partitioning is used. In [79], the authors proposed a method based on MI criteria and GP. In the proposed method, the sensing domain is decomposed into cells representing clusters of sensing locations, and the recursive greedy algorithm is run on

these cells instead of the actual sensing locations. The reason for cell decomposition is the locality property of the mutual information criterion. Based on this property, two sets of sensing locations which are sufficiently far apart are roughly independent. Therefore, to obtain a large amount of information, a robot will need to visit a number of locations that are far apart. In [38], the authors used hexagonal decomposition to partition the environment into manageable regions and then developed strategies to explore the field efficiently by moving robots throughout the regions considering energy constraints. There are specific movement constraints on the Dubins vehicles, and the paper discusses how this affects the exploration efficiency and effectiveness. [39] extends previous work in environments with cluttered obstacles. In [95], AUV and UGV collaborate to gather comprehensive data for precision agriculture tasks. Initially, the environment is modeled with a set of possibly overlapping disks. The objective is to choose a sampling location in each disk, and to plan a tour based on Orienteering to visit the sampling locations in order to minimize the sum of the travel time and measurement time. In the algorithm, the UAV can land on the UGV and be transported between points without consuming energy. This method is not online, meaning the map is not updated after each measurement.

In [40], task planning under uncertainty was discussed for precision agriculture using multiple robots. As part of the three phases approach, feasible vertices are first sampled subject to resource budgets (for example, irrigation water limits), then feasible vertices are then sampled subject to energy budgets (for example, time), then a row is selected and the corresponding paths are planned.

In [94], they studied multirobot routing in a vineyard using a formulation based on the team orienteering problem. In the orienteering problem (OP) an agent is required to traverse a graph in which each vertex has a predetermined reward and each edge has a fixed cost. A path should be computed to maximize the sum of collected rewards while ensuring that the sum of the costs of traversed edges does not exceed a preassigned budget. Team Orienteering Problem (TOP) is a version of OP in which a team of robots works together to collect rewards. Various variations of this problem have been extensively discussed in [89, 90, 91]. The main limitation of this line of research is that it assumes that rewards are predetermined in advance. However, in many situations this is not realistic, as data collected on the fly may lead to revised estimates about the value of reaching a certain location.

In [46], the authors discussed the centralized and decentralized multi-robot Gaussian Process (GP) training and prediction in multi-agent system. This paper introduces different methods for GP prediction on graph-based topologies, including covariance-based nearest neighbors that select agents for GP prediction on locations of interest. Their decentralized method is based on the fact that GPs behave poorly with increasing observation numbers ($O(n^2)$). They plan to provide the more computationally efficient algorithm with a multirobot system.

Chapter 3

Single-Robot IPP, Q-Learning Solution

In this chapter, the concept of IPP with a single robot is explored with applications in environmental map learning, and a solution based on Q-Learning method is developed and discussed. The work shown here was originally presented in [74].

3.1 Informative Path Planning

As discussed in Section 2.1, ground robots can be used as part of a farm monitoring and sampling system . These samples must be collected over vast spatial domains and the use of robots allows better handling of dwindling agricultural workforce. In these applications it is often necessary to frequently collect data to monitor parameters such as water, nitrates, nutrients, or to determine soil quality, or crop estimates.

A key problem in this area is how to handle the tradeoff originating by the necessity to collect numerous samples while being constrained by how far a robot can travel before its battery is depleted and needs recharging. In this context, planning a path to ideally collect *the best samples* while not running out of fuel is of fundamental importance.

This family of problems is known as informative path planning (IPP). Informally speaking, the goal of IPP is to define a path through a set of sampling locations allowing for the best estimate of the parameter being measured. Depending on the application, sampling locations may be preassigned (and then the robot has to select a subset of them), or may be determined on the fly. Owing to the intrinsic computational complexity of the problem, one has to settle for collecting a sub-optimal subset of samples. A distribution of spatial data can be modeled by a Gaussian Process (GP), and this approach is often used in agriculture [84, 81]. Combined with this model, metrics such as mutual information (MI), entropy, or variance can be used to guide the process of collecting samples to estimate a GP of the field. While the IPP problem has been extensively studied in the past, we consider a harder version where we also manage an assigned energy budget that is influenced by stochastic travel costs. This makes this problem different and harder than the orienteering problem that has been recently studied in a similar, yet simpler setting emerging in precision agriculture [92, 89, 90, 91]. We consider the stochastic environment because agricultural

fields may change from time and this may affect the cost of travel.

Let us now define the IPP in mathematical terms. Let $\mathcal{U} \subset \Re^2$ be the environment of interest. We hypothesize the presence of random distribution characterized by a time-invariant, Markovian transition kernel that depends on the position, but not on the underlying field being sampled. The current location of the robot is defined by $s_s = (s^x, s^y)$. We assume the robot starts from a pre-assigned start location s_{init} (e.g., the point where the robot is deployed), and must terminate its mission at location s_f, where it will either be retrieved or recharged. The robot is provided with n different sample locations of interest denoted by the set $\mathcal{V} = \{s_1, s_2, \ldots, s_n\}$. The robot is subject to a travel budget B limiting the set of sample locations it can visit. This constraint models, for example, the limited energy provided by the battery onboard the robot. We denote with x_g the scalar reading collected by the robot when sampling location $s_g \in \mathcal{V}$, and will denote with χ_g the random variable modeling x_g. The goal is to visit a subset of locations of $\mathcal{V}$ such that the overall travel budget is not exceeded and the accuracy of field reconstructed using the collected samples is maximized. More formally, assuming ρ is a path starting at s_{init} visiting a subset of locations of $\mathcal{V}$ and ending in s_f, let $f(\rho)$ be a generic function measuring the quality of the reconstructed field and $C(\rho)$ be the cost associated with traversing ρ. The informative path planning problem (IPP) can then be expressed as the problem of solving the following constrained optimization problem

$$\begin{aligned} \rho^* &= \underset{\rho \in \psi}{\operatorname{argmax}}\, f(\rho) \\ s.t.\, &C(\rho) \leq B \end{aligned} \tag{3.1}$$

where ψ is the set of all paths from s_{init} (start point) to s_f (final point). It is immediate to note that the orienteering problem is a special instance of IPP, and therefore IPP is NP-hard [84].

3.2 Q-learning based Solution

A Q-Learning-based algorithm is described in this section.

3.2.1 Markov Decision Process

Markov Decision Process (MDP) is a standard modeling tool for tackling planning problems when the state is fully observable, the environment is stochastic, and rewards are additive. A finite MDP is defined by set of states $s \in S$, set of actions $a \in A$, a transition function $p(s'|s, a)$ and a reward function $R(s, a, s')$. The transition function, describes the probability of moving to a new state s' given the current state s and action a. The reward function, specifies the immediate reward received after taking action a in state s and transiting to state s'.

The goal is to find the optimal policy $\pi^* : S \to A$ to maximize the expected return. We consider finite MDPs with finite number of states and actions. Different reward functions have been proposed in literature. In our work we deal with an *episodic* task, i.e., the task always ends after a finite number of steps, either because the robot reaches the final location, or because it runs out of fuel. In this case it is typical to define the expected return, G_t, as a function of reward sequence

$$G_t = r_t + \lambda r_{t+1} + ... + \lambda^{T-t} r_T;\ 0 \leq \lambda \leq 1 \tag{3.2}$$

where λ is discount factor and shows the rewards in next steps have less effect on the problem and T is the time of the last action. The objective is then to produce realizations that in expectation maximize G_t. When the model of the environment is known, classic methods such as the Bellman equation and dynamic programming can be used to find the optimal policy. It is worth noting that this problem is unconstrained, i.e., the objective is to maximize G_t only, while in our problem, we are also assigned a constraint on the traveled budget. In constrained MDP (CMDP), the optimal policy must satisfy a set of constraint which is represented by penalty function [6].

If the problem is episodic, Monte Carlo (MC) methods can be used. An MC method only needs experience (sample sequence of starts, actions and rewards) from actual or simulated environment. Then, it solves the problem based on averaging sample returns. MC methods use the value function to find the policy where this is the expected return when the start state is s following policy π.

If the problem can not be defined in episodic way and it needs update at every time step, the Temporal Difference (TD) methods can be used; A Sarsa and Q-learning are on-policy and off-policy method of model free RL. A system that considers n-step rewards instead of one-step rewards is called n-step bootstrapping, (n-step Sarsa and Q-learning) [86].

3.2.2 Q-Learning

In model-free reinforcement learning, complete knowledge of the environment is not assumed. In our case, while we assume that the transition probabilities are known, the rewards are instead not known upfront (their specific formulation is described later in Section 3.2.3). In such scenarios, methods such as Q-learning, SARSA, and others can be used to determine the policy [86]. Q-learning is often used because of its off-policy nature, i.e., the ability to estimate the value of taking a certain action a from state s while the system is following a preassigned policy that may be different from the optimal one. In on-policy learning (SARSA), the $Q(s, a)$ function is learned from actions the agent took using its current policy $\pi(a|s)$ [86]. In Q-learning, for each state s we estimate $Q(s, a)$, i.e., the expected return of executing action a while in state s at time-step t and following the policy associated afterwards. Key to Q-learning is the following update equation

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[R_{t+1} + \lambda \operatorname*{argmax}_a Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{3.3}$$

where α is the learning parameter and s_t and a_t are the state and action at time t. In this case, the learned action value function, directly approximates the optimal action value function Q_* independent of the policy being followed [86]. In our work, we implement a similar update equation to estimate the value of visiting a certain location.

3.2.3 Reward Function

For this section, we assume that $\mathcal{V}$, the subset of coordinates in the domain where the robot can collect a sample, is given, e.g., it may be provided by a human with expertise about the domain of operation. As we mentioned before we associate a random variable χ_g to each location $s_g \in \mathcal{V}$. χ_g models the sample collected at s_g. Let $\mathcal{A} \subset \mathcal{V}$ be the subset of already visited locations and let $\chi_\mathcal{A}$ denote the set of random variables associated with the elements of $\mathcal{A}$. Initially $\mathcal{A} = \{s_{init}\}$ (the starting location), and the role of the planning algorithm is to iteratively add locations to $\mathcal{A}$ so that eventually the underlying spatial field can be accurately estimated, while at the same time ensuring that the robot does not overrrun its assigned travel budget. To avoid trivial instances, we assume that with the assigned budget the robot cannot visit all elements $\mathcal{V}$.

To asses the quality of the information gathered at a given location and drive the sampling selection process we use mutual information (MI). MI expresses the expected reduction of entropy of the sample locations in set $\mathcal{V} \setminus \mathcal{A}$, which the robot has not visited yet, after considering the measurements obtained at visited locations. This is an instance of the regression problem, where we use the measured data from the visited sample locations to predict values at yet to be visited sample locations. MI can be defined as follows by using the concepts of entropy and conditional entropy. The classical definition of continuous (or differential) entropy of a continuous random variable χ_g according to [64] is

$$H(\chi_g) = -\int P_{\chi_g} \log(P_{\chi_g})\, dx$$

This definition can be naturally extended to the case of a set of variables $\chi_\mathcal{A}$. For a sample location $s_g \notin \mathcal{A}$ and a set $\mathcal{A} \subset \mathcal{V}$ we define the conditional entropy of s_g with respect to $\mathcal{A}$ as:

$$H(\chi_g|\chi_\mathcal{A}) = -\int P(\chi_g, \chi_\mathcal{A}) \log(P(\chi_g|\chi_\mathcal{A}))dx_g dx_\mathcal{A}$$

MI is then defined as

$$\mathcal{MI}(\chi_\mathcal{A}; \chi_{\mathcal{V}\setminus\mathcal{A}}) = H(\chi_{\mathcal{V}\setminus\mathcal{A}}) - H(\chi_{\mathcal{V}\setminus\mathcal{A}}|\chi_\mathcal{A})$$

Therefore, the problem of selecting the best set of sample locations $\mathcal{A}$ is equivalent to solving the following optimization problem

$$\underset{\mathcal{A}\subset\mathcal{V}}{\operatorname{argmax}}[H(\chi_{\mathcal{V}\setminus\mathcal{A}}) - H(\chi_{\mathcal{V}\setminus\mathcal{A}}|\chi_{\mathcal{A}})]$$

while ensuring that the robot keeps the consuming energy under the travel budget as it visits the locations in $\mathcal{A}$. In our work, $\mathcal{A}$ is not determined *globally*, but rather incrementally, i.e., by adding one sample location at the time we aim to maximally increase MI. More formally, each candidate sample location s_g will give the following contribution [33]:

$$\mathcal{MI}(\chi_{\mathcal{A}} \cup \chi_g) - \mathcal{MI}(\chi_{\mathcal{A}}) = H(\chi_g|\chi_{\mathcal{A}}) - H(\chi_g|\chi_{\mathcal{V}\setminus\{\mathcal{A}\cup g\}}) \tag{3.4}$$

It is well known that this approach yields suboptimal results, but this is inevitable in the general case due to the intrinsic computational complexity of the problem. To formally define how we compute Equation 3.4, we resort to the assumption that the underlying field can be modeled using a GP. To make predictions at location s_g, we consider the conditional distribution $p(\chi_g = x_g|\chi_{\mathcal{A}} = x_{\mathcal{A}})$, where we condition on all sample locations $x_{\mathcal{A}}$ visited by the robot [48].

The multivariate normal distribution over a set $\chi_{\mathcal{V}}$ of random variables associated with n locations in $\mathcal{V}$ is defined as:

$$P(\chi_{\mathcal{V}} = x_{\mathcal{V}}) = \frac{1}{(2\pi)^{n/2}|\Sigma|} e^{-\frac{1}{2}(x_{\mathcal{V}}-\mu)^T\Sigma^{-1}(x_{\mathcal{V}}-\mu)} \tag{3.5}$$

where every location in $\mathcal{V}$ corresponds to one particular sampling location. The multivariate normal distribution is fully specified by providing a mean vector μ and a covariance matrix Σ. If we know the values collected at $\mathcal{A} \subset \mathcal{V}$, we find that for the sample location $s_g \in \mathcal{V} \setminus A$ the conditional distribution $p(\chi_g = x_g|\chi_{\mathcal{A}} = x_{\mathcal{A}})$ is a normal distribution, where mean $\mu_{g|\mathcal{A}}$ and variance $\sigma^2_{g|\mathcal{A}}$ are given by [64]:

$$\begin{aligned} \mu_{g|\mathcal{A}} &= \mu_g + \Sigma_{g\mathcal{A}}\Sigma^{-1}_{\mathcal{A}\mathcal{A}}(x_{\mathcal{A}} - \mu_{\mathcal{A}}), \\ \sigma^2_{g|\mathcal{A}} &= \sigma^2_g - \Sigma_{g\mathcal{A}}\Sigma^{-1}_{\mathcal{A}\mathcal{A}}\Sigma_{\mathcal{A}g} \end{aligned} \tag{3.6}$$

where $\Sigma_{g\mathcal{A}} = \Sigma^T_{\mathcal{A}g}$ is a row vector of the variances χ_g with all variables in $\chi_{\mathcal{A}}$, similarly for $\Sigma_{\mathcal{A}\mathcal{A}}$. μ_g and σ^2_g are the mean and variance of χ_g. A general formula and detailed explanation can be found in Section 2.2.

For any finite subset $\mathcal{A} = \{s_1, s_2, ..., s_m\}$, $\mathcal{A} \subset \mathcal{V}$ of location variables, the covariance matrix $\Sigma_{\mathcal{A}\mathcal{A}}$ of the variables χ_A is obtained by

$$\Sigma_{\mathcal{A}\mathcal{A}} = \begin{bmatrix} \mathcal{K}(s_1, s_1) & \mathcal{K}(s_1, s_2) & \cdots & \mathcal{K}(s_1, s_m) \\ \vdots & \vdots & & \vdots \\ \mathcal{K}(s_m, s_1) & \mathcal{K}(s_m, s_2) & ... & \mathcal{K}(s_m, s_m) \end{bmatrix} \tag{3.7}$$

where $\mathcal{K}$ is the Mattern kernel, which is the generalization of RBF Kernel and is parameterized by length scale $l \geq 0$, smoothness scale ν and Euclidean distance $d(.,.)$ which assumes it only depends on the distance between two random variables (isotropy) and is independent of their locations (stationary). Once all the mean and covariance functions has been estimated, we can evaluate the MI criterion in Equation 3.4. Using Equation 3.6 and the fact that the differential entropy of a Gaussian random variable χ_g conditioned on some set of variables $\mathcal{A}$ is a monotonic function of its variance

$$\begin{aligned} H(\chi_g|\chi_\mathcal{A}) &= \frac{1}{2}\log(2\pi e\sigma^2_{g|\mathcal{A}}) \\ &= \frac{1}{2}\log(\sigma^2_{g|\mathcal{A}}) + \frac{1}{2}(\log(2\pi)+1) \end{aligned} \tag{3.8}$$

we can define the value information of each candidate sample location as

$$r_g = \frac{\sigma^2_{g|\mathcal{A}}}{\sigma^2_{g|\bar{\mathcal{A}}}} = \frac{\sigma^2_g - \Sigma_{g\mathcal{A}}\Sigma^{-1}_{\mathcal{A}\mathcal{A}}\Sigma_{\mathcal{A}g}}{\sigma^2_g - \Sigma_{g\bar{\mathcal{A}}}\Sigma^{-1}_{\bar{\mathcal{A}}\bar{\mathcal{A}}}\Sigma_{\bar{\mathcal{A}}g}} \tag{3.9}$$

where $\bar{\mathcal{A}} = \mathcal{V} \setminus \{\mathcal{A} \cup \{g\}\}$. We have to make sure $\sigma^2_g - \Sigma_{g\bar{\mathcal{A}}}\Sigma^{-1}_{\bar{\mathcal{A}}\bar{\mathcal{A}}}\Sigma_{\bar{\mathcal{A}}g}$ is always nonzero to guarantee the r_g is the finite number.

Proof of Equation 3.9 The reward function is calculated as below.

$$\begin{aligned} &\mathcal{MI}(\chi_\mathcal{A} \cup \chi_g) - \mathcal{MI}(\chi_\mathcal{A}) = H(\chi_g|\chi_\mathcal{A}) - H(\chi_g|\chi_{\mathcal{V}\setminus\{\chi_\mathcal{A}\cup\chi_g\}}) \\ &= \frac{1}{2}\log(\sigma^2_{g|\mathcal{A}}) + \frac{1}{2}(\log(2\pi)+1) - (\frac{1}{2}\log(\sigma^2_{g|\bar{\mathcal{A}}}) + \frac{1}{2}(\log(2\pi)+1)) \\ &= \frac{1}{2}\log\frac{\sigma^2_{g|\mathcal{A}}}{\sigma^2_{g|\bar{\mathcal{A}}}} \end{aligned} \tag{3.10}$$

In order to simplify the calculation, we remove the constant value 1/2 and log in Equation 3.9.

3.2.4 Proposed Algorithm

Starting from the framework established in the previous subsection, to select the subset of sample locations $\mathcal{A}$ while considering the travel budget, we propose a two stages algorithm. This problem is related to orienteering problem, but it is however different because in the orienteering problem one must know the value of each vertex before, while in this scenario this is not the case. This aspect will be further discussed in the experimental section.

The outer planning algorithm determines the next location to add to the path, while the inner algorithm plans a collision-free path to the designated location and estimates the travel cost while accounting for the stochastic nature of the environment. Algorithm 3.1 shows the outer planning loop and works as follows.

Algorithm 3.1 Q-Learning based Planner for robot R with limited budget B

Input: $\mathcal{V}$, s_{init}, s_f, B, N, obstacles-list, r, γ, ε, α
Output: $R_T, E_T, \mathcal{A}$
Initialize matrix U with zeros
for N **iterations do**
    $R_T \leftarrow 0$
    $E_T \leftarrow 0$
    $\mathcal{A} \leftarrow \{s_{init}\}$
    $s_s \leftarrow s_{init}$
    for each episode **do**
        $s_g \leftarrow \begin{cases} \text{random element in } \mathcal{V} \setminus \mathcal{A} & \text{within r with prob } \varepsilon \\ \text{element as per Eq. 3.11} & \text{with prob } 1-\varepsilon \end{cases}$
        $e_T \leftarrow \text{RRT}(env, s, s_g, \text{obstacles-list})$
        $E_T \leftarrow E_T + e_T$
        if $E_T > B$ **then**
            $R_T \leftarrow R_T - \Delta$
        else if $E_T \leq B$ & $s_g = s_f$ **then**
            $R_T \leftarrow R_T - (B - E_T)/K$
        else
            $R_T \leftarrow R_T + r_g$ (see Eq. 3.9) /dist (s_s, s_g)
        end if
        update U based on Eq. 3.12
        $\mathcal{A} \leftarrow \mathcal{A} \cup \{s_g\}$
        $s_s \leftarrow s_g$
    end for
end for
return $\mathcal{A}$ and navigation path to travel along sample points in $\mathcal{A}$.

The algorithm takes as input the set of candidate sampling locations $\mathcal{V}$, the start and final locations s_{init} and s_f, the budget B, a number of iterations N, the locations of obstacles, and parameter r (radius), ε, γ, and α.

The idea behind the algorithm is to develop a plan using an approach inspired by Q-learning, whereby for each state s we estimate $Q(s, a)$, i.e., the expected return of executing action a while in state s. In our problem, s is a robot location, i.e., one of the elements of $\mathcal{A}$, and action a represents a possible next sampling location in $\mathcal{V} \setminus \mathcal{A}$, i.e., a is an unvisited location. The reward associated with a potential sampling location (i.e., an action) is given by r_g defined in Equation (3.9). These values are stored in an $n \times n$ matrix U such that $U[i, j]$ represents the estimated value of moving from s_i to s_j and collecting a sample at s_j. Assuming the robot is currently positioned at location s_s, it will pick the next location randomly among the locations in $\mathcal{V} \setminus \mathcal{A}$ within a radius r with probability ε, and with probability $1 - \varepsilon$ it will pick location

s_g with g defined as

$$g = \arg \max_{j \in \mathcal{V} \setminus \mathcal{A}} U[s, j]. \tag{3.11}$$

Constraining the choice of the next location to be within radius r ensures that the robot does not switch too frequently between far away locations, thus helping to make a more efficient use of the available budget. The parameter ε instead balances exploration and exploitation and is decreased during the execution of the algorithm to favor exploration in the beginning and then exploitation. After a location s_g is selected, in line 11 the second step of the algorithm (inner algorithm) is executed. Based on the current location s_s and the selected next location s_g, the RRT algorithm [50] computes an obstacle free path and estimates the energy consumed (e_T). In our implementation we use RRT, but other motion planners could be used as well. To estimate the consumed energy e_t, the algorithms uses the motion model for the robot to simulate multiple times the execution of the returned path and obtain an estimate for e_T. The concatenation of the routes returned by the successive calls of the inner planning loop are eventually returned by the overall planner to define how to navigate through the sequence of selected locations $\mathcal{A}$.

In the next step, the matrix U is updated as follows. The dimension of the U function depends on the number of sampling points. First, for location s_g, we will find the best next goal based on Equation 3.11 and then U is updated as follows:

$$U[i, g] = (1 - \alpha)U[i, g] + \alpha(R_T + \gamma \operatorname*{argmax}_{j \in \mathcal{V} \setminus \mathcal{A}} U[g, j]) \tag{3.12}$$

where i is the index fo the current location and g varies over the set of all unvisited locations. The idea behind this update is to consider not only the immediate reward, but also a one-step lookahead. In Equation (3.12) the current return value R_T is used. As seen in the pseudocode, R_T accumulates the rewards, but incurs a penalty when the consumed energy E_T exceeds the budget. The algorithm also penalizes runs where the robot ends with too much unused budget (line 18). One could of course use a deeper lookahead, but this would come at an increased computational cost. Studying the impact of the lookahead horizon is part of our future work. In each iteration, the algorithm assumes the robot starts from beginning location s_{init} and in each episode, it will add one sample location s_g. Each iteration consists of number of episodes and will end if the consumed energy goes beyond the Budget or the robot visits the final location s_f with in a budget. Through repeated iterations, the contents of the matrix U are updated and model the value of visiting a certain location while averaging all past executions, aligned with the spirit of Q-learning.

3.3 Results and Discussion

To investigate the effectiveness of the proposed method, we simulate it in different scenarios with various underlying distributions of the field being measured, number of sample locations and obstacles.

3.3.1 Methods

In this section, comparisons are made with three alternatives; heuristic greedy-random point selection (M_{GRRRT}); a Q-learning method (M_{LQL}); and an orienteering method (M_{Or}).

Heuristic Greedy-Random

The heuristic greedy-random strategy selects at each stage either a random unvisited location or the closest unvisited location. More precisely, with probability $0 \leq \beta \leq 1$ it selects a location in $\mathcal{V} \setminus \mathcal{A}$ using a uniform distribution (random option), and with probability $1 - \beta$ it selects the location in $\mathcal{V} \setminus \mathcal{A}$ that is closest to the current robot location (greedy option). To account for the budget constraint, in both instances the next sample is rejected if there is not enough budget left to reach it and then move to the final location s_f. When a location is discarded, the selection method is iterated until either a valid location is found or no more candidate locations are left. In this last case, the algorithm selects s_f and tries to reach the final location. After the sample selection process, the RRT method finds an obstacle free path to the next location.

Q-Learning

The Q-learning method chooses the next location based on Algorithm 3.1, and afterwards runs an offline Q-learning method to find the best obstacle free path to reach the location of choice. Note that our proposed method uses the RRT algorithm rather than Q-Learning.

Orienteering method

The orienteering method (Or) determines the path that visits the greatest number of points in $\mathcal{V}$ without exceeding the preassigned budget. It involves maximizing a reward function while keeping a constraint on the cost of traveling between the places visited by the route. Note that in this case it is necessary to assign a value to each element of $\mathcal{V}$ in advance, consistently with the fact that in orienteering one must know the rewards of the vertices beforehand. Each vertex has a reward associated with it, and each edge between two vertices has a cost as well. Vertices can be visited more than once if necessary, but rewards can only be collected once. The starting and ending vertices are fixed. In many cases, the start and end vertices coincide such that the resulting path begins and ends in the same place.

3.3.2 Metrics

In order to compare the quality of the solutions produced by the different methods, we use two criteria.

MSE

We use the MSE error between the predicted model and the underlying model (ground truth, unknown to each of the algorithms). To predict the model using the values collected at the sample locations we use the *scikit-learn* Python library and its GP regression library with Mattern Kernel with length scale of 1 and smoothness parameter of 1.5. The choice of the kernel and of the parameters was made after having experimentally evaluated different alternatives. The same kernel and parameters were used for all algorithms.

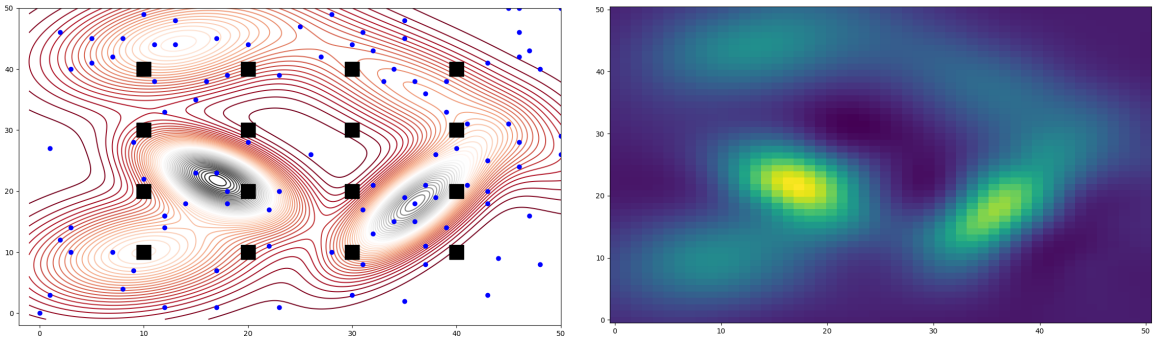
Failure Rate

The second criterion to evaluate the effectiveness of the algorithm is the ability to remain within the assigned budget B. Indeed, from a practical perspective it is important for the robot to reach the final docking station before running out of energy.

3.3.3 Results

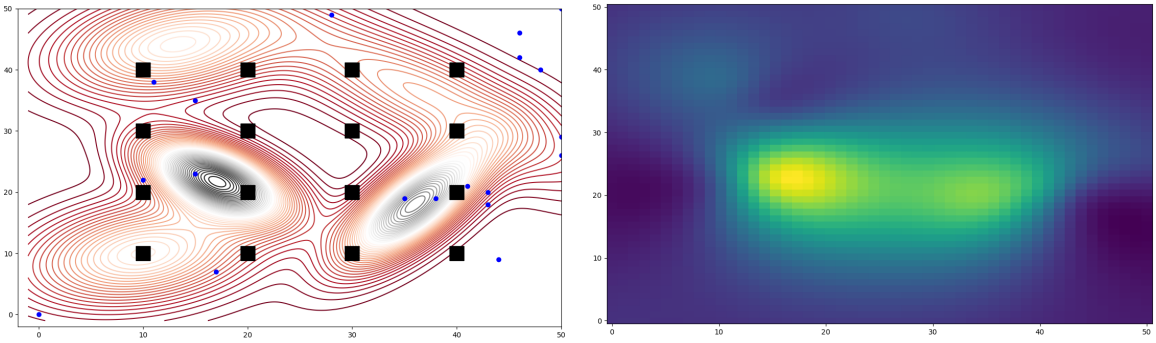
In the first scenario, we consider a 2D stochastic environment with obstacles spread uniformly on a lattice pattern. This choice is informed by our practical problem, i.e., collecting samples in an orchard with equally spaced fruit trees. In all cases, the start location s_{init} is on the bottom left (0,0) and the final location s_f is on the top right, (50,50). Four different budgets were used, namely 500, 750, 1000 and 2000. For this scenario, the set $\mathcal{V}$ consists of 100 locations, half of which are randomly dispersed in the environment, while the other half are chosen by hand in proximity of the peaks of the underlying ground truth distribution (see Figure 3.2 (a)-(b)). The random locations are used to test robustness to inaccurate information, while the locations placed by hand are consistent with our initial assumption that a human with domain expertise would select *interesting* places to sample. With reference to Algorithm 3.1, we set Δ to 1, K to 100, γ to 0.9, α to 0.5, r to 10 and ε starts from 0.9 and decreases to 0.2. Also for M_{LQL} and M_{GRRRT}, we set α to 0.5, λ to 0.9 and β to 0.5.

Table 3.1 summarizes the results of the three algorithms for the different budgets. For this first scenario, we do not consider the orienteering method. For each test case we present the average results produced by each of the algorithms (M_{LRRT} is the algorithm discussed in this paper, M_{GRRRT} is the heuristic greedy-random algorithm and M_{LQL} is the learning sampling selection method following by Q-learning). The values displayed for N_A, E_T, R_T and MSE are limited to the cases where the robot does not exceed the assigned budget B. N_f shows how often the algorithms violate the budget constraint B (out of 25 runs). The trial counts as a failure when the robot runs out of the energy while it has not visited the final location s_g. $N_{\mathcal{A}}$ is the number of locations visited by each algorithm, E_T is the consumed energy, R_T is total reward and MSE is the mean square error. The last column provides the running time of each algorithms. The results illustrate that our proposed algorithm M_{LRRT} is generally faster and achieves better MSE comparing to M_{GRRRT} despite



(a) (b)

Figure 3.1: (a) The underlying distribution with all sample locations and (b) predicted distribution with all locations.

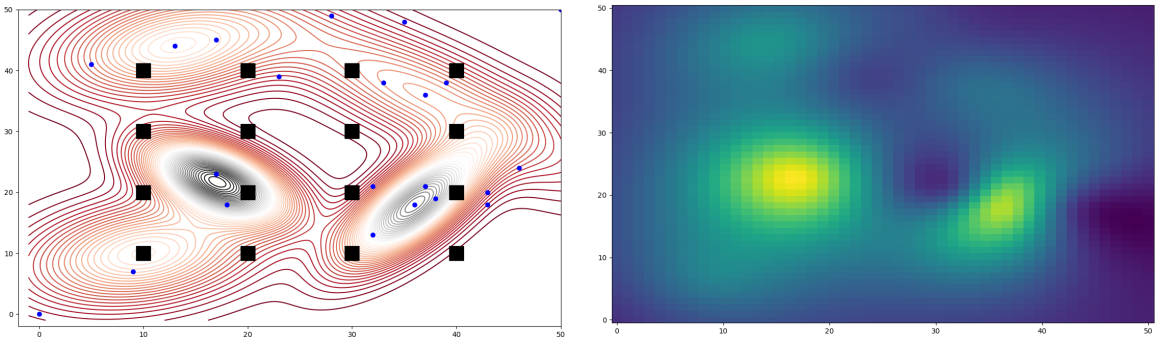


(a) (b)

Figure 3.2: (a)-(b) Selected sample locations and reconstructed underlying distribution with M_{GRRRT} ($B = 750$).

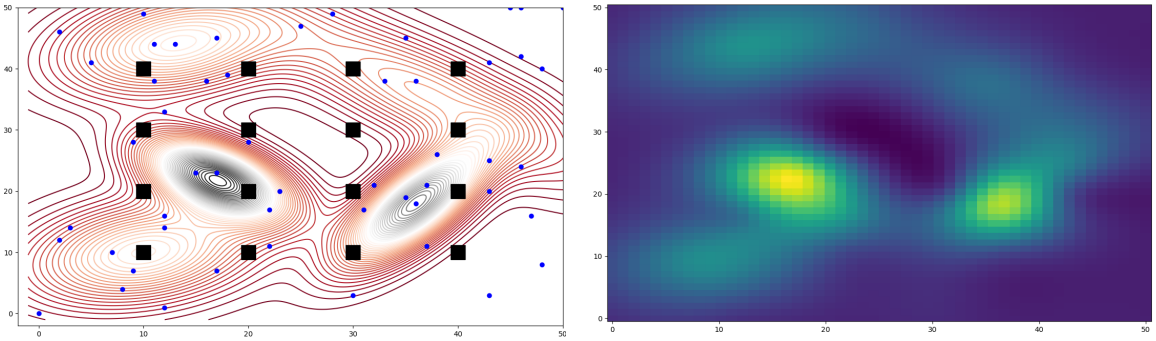
visiting a smaller number of locations in average. Importantly, M_{LRRT} consistently manages the assigned budget, while the other two algorithms have a higher failure rate. Overall, the reward column indicates the M_{LRRT} method visits the locations providing better information (in term of GP regression) which is another crucial factor in IPP problem. In cases with tight budgets ($B = 500$), this issue becomes increasingly important where the M_{LRRT} method would be able to reach better MSE and reward with fewer failures.

Figures 3.1 (a)-(b) show the underlying model of the spatial data in the environment with all sample locations and the predicted model computed with all locations. Figures 3.2 (a)-(b) show the selected points with the heuristic greedy-random method. Despite the robot visiting more locations, the reconstructed scalar field is less accurate. Furthermore, it runs out of energy more frequently. Figures 3.3 (a)-(b) show the selected and visited locations with our proposed method with budget 750. It can be seen that the proposed approach selects the points widely to cover the whole area and predicts the model more accurately by visiting a sufficient number of sample



(a) (b)

Figure 3.3: (a)-(b) Selected sample locations and reconstructed underlying distribution with M_{LRRT} ($B = 750$).



(a) (b)

Figure 3.4: (a)-(b) Selected sample locations and reconstructed underlying distribution with M_{LRRT} and ($B = 2000$).

locations while maintaining the budget. Figures 3.4 (a)-(b) show the predicted model with the visited locations in M_{LRRT} method with budget 2000. It is clear from the picture that the predicted models resemble the underlying model especially it has the ability of predicting each peaks of Gaussian distribution despite the fact that it will only visit half of the sample locations.

The path generated by the M_{LRRT} algorithm with a budget 750 is shown in Figure 3.5, while Figure 3.6 shows the path generated by M_{GRRRT} under the same conditions. The figures show that the path produced in the first case more effectively balances between covering the entire space and focusing on the peaks of the underlying distribution. Moreover, the path is more regular with less switches between different sides of the environment.

As a second scenario, we consider the same environment, but this time without obstacles and with a different underlying distribution (see Figure 3.7).

In this case our method is compared with the orienteering method. M_{LRRT} represents the proposed method, M_{Or} is the orienteering method with setting all rewards

Budget	methods	$N_{\mathcal{A}}$	E_T	R_T	MSE	N_f	time
500	M_{LRRT}	14.4	437.93	1.53	**0.001734**	1	0:17.23
	M_{LQL}	15.2	456.12	0.54	0.001803	1	7:62.13
	M_{GRRRT}	17.4	468.95	0.44	0.001748	3	0:28.74
750	M_{LRRT}	16.4	645.14	2.74	**0.001494**	0	0:23.38
	M_{LQL}	22.25	690.22	2.28	0.001671	1	10:45.12
	M_{GRRRT}	22.6	686.35	1.14	0.001621	3	0:35.42
1000	M_{LRRT}	25	801.13	2.69	**0.001144**	1	0:35.55
	M_{LQL}	30.7	847.24	2.47	0.001372	0	14:25.18
	M_{GRRRT}	34.5	780.90	1.40	0.001468	2	0:45.87
2000	M_{LRRT}	45.9	1764.85	3.14	0.0007221	0	01:34.69
	M_{LQL}	55.3	1842.84	3.63	**0.0005320**	0	22:38.10
	M_{GRRRT}	53.2	1743.23	2.17	0.0007980	0	01:40.74

Table 3.1: Summary of results for 25 runs with different budgets. The iteration number is set to 20.

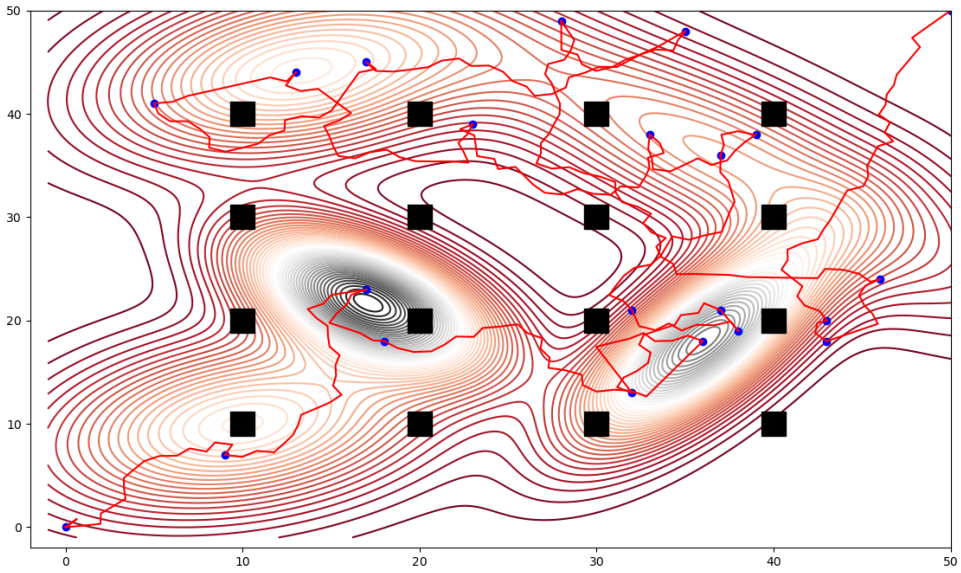


Figure 3.5: Path followed by the robot running the M_{LRRT} algorithm with budget of 750.

for all locations to 1 and M_{OrwP} represents the Orienteering method with rewards set equal to the expectation of the GP prior at the point. The orienteering problem is solved using a general purpose heuristic known as S-algorithm [96] (the exact solution based on mixed integer linear programming is too time consuming when considering

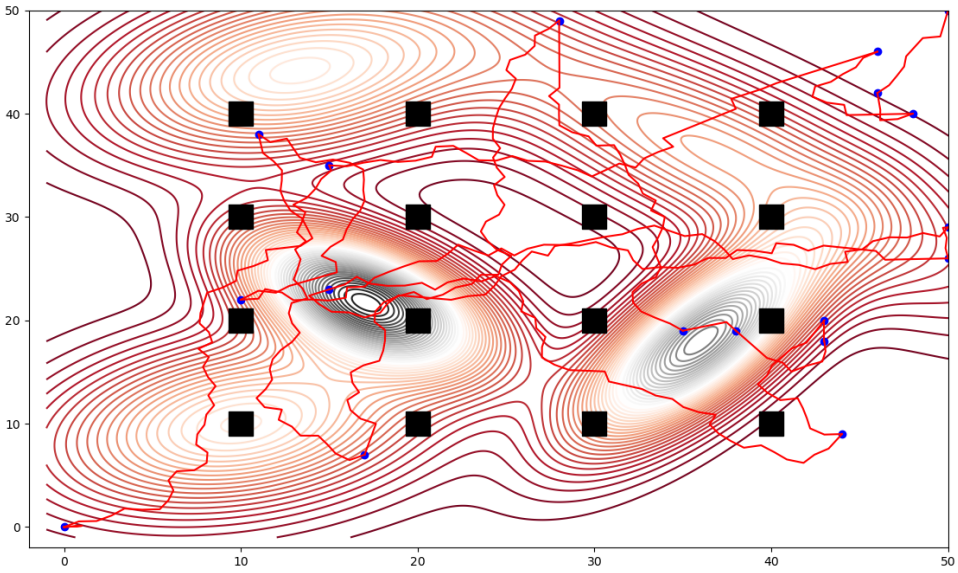


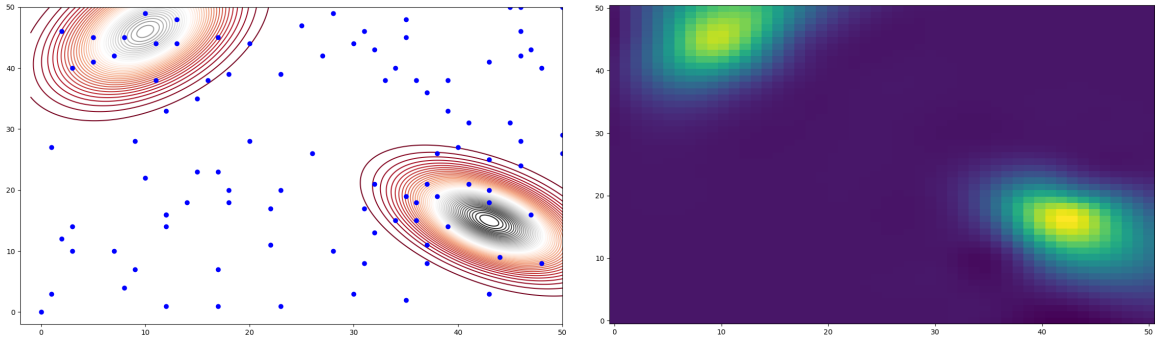
Figure 3.6: Path followed by the robot running the M_{GRRRT} algorithm with budget of 750.

our problem instances). While the orienteering method manages to visit a greater number of locations in $\mathcal{V}$ it suffers from two major setbacks. The first is that its number of failures is much higher. This is explained by the fact that orienteering assumes deterministic travel costs, while in our scenario these are stochastic. The second is that the subset of selected points still renders a less accurate reconstruction, as attested by the MSE column. Note that in this case our algorithm is slower because we have increased the number of iterations to 200.

Figures 3.7 (a) and (b) show all the sample locations in the environment and predicted model using them. Figures 3.8 (a) and (b) show the sample locations selected by learning based selection method M_{LRRT} and the reconstructed underlying model with budget 200. M_{LRRT} can more precisely reconstruct the underlying field even with half the locations visited compared to both orienteering methods. Figures 3.9 (a) and (b) show the sample locations selected by the M_{Or} method and the reconstructed underlying model. Since the robot does not reach the second peak, the reconstructed field is incomplete. Figures 3.10 (a) and (b) show the sample locations selected by M_{OrwP} method. In this case, the robot attempts to visit both peaks, but is unsuccessful in rebuilding the underlying field correctly which in result leads to higher MSE.

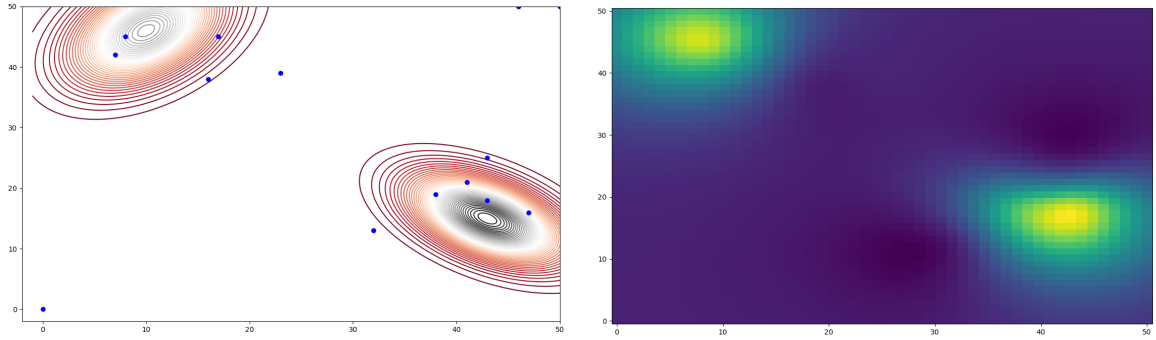
3.4 Conclusions

In this chapter, we study the IPP using a single robot, where path planning was required on a set of sampling locations within a limited budget in a stochastic dynamic environment with stationary obstacles. The objective is to provide the best



(a) (b)

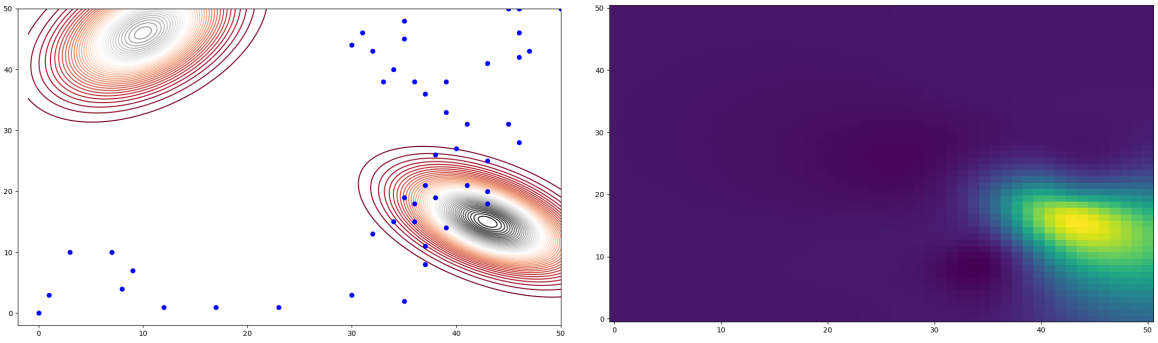
Figure 3.7: (a) The underlying distribution with all sample locations and (b) predicted distribution with all locations.



(a) (b)

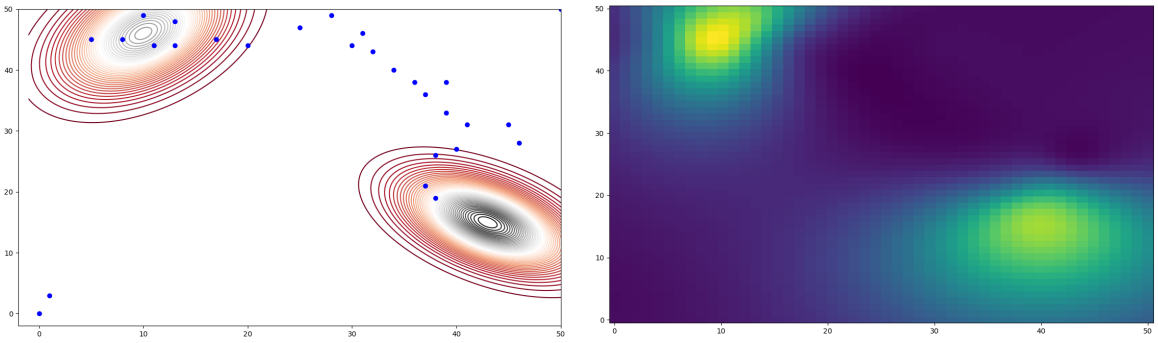
Figure 3.8: (a) The selected sample locations and (b) reconstructed underlying distribution with M_{LRRT} ($B = 200$).

estimation of an unknown scalar field while subject to a travel budget. Agricultural robots in constrained environments are likely to encounter this problem when collecting samples. The algorithm determines the next sample location based on the consumed energy, budget and mutual information criteria, and then a second loop finds the best obstacle free path using the RRT algorithm to visit the designated sample location. The Or method and heuristic greedy-random algorithm are compared with Q-Learning-based methods. The simulation result showed the effectiveness of the proposed method. According to MSE criteria, our proposed learning method predicts the underlying field better than heuristic greedy-random and orienteering methods in complex environments.



(a) (b)

Figure 3.9: (a) The selected sample locations and (b) reconstructed underlying distribution with M_{Or} ($B = 200$).



(a) (b)

Figure 3.10: (a) The selected sample locations and (b) reconstructed underlying distribution with M_{OrwP} ($B = 200$).

Budget	methods	N_f	$N_{\mathcal{A}}$	E_T	MSE	time
100	M_{LRRT}	3	7.5	89.22	**0.00121**	3:20.78
	M_{Or}	12	23	99.54	0.00137	0:11.09
	M_{OrwP}	9	17.8	98.80	0.00128	0:10.33
200	M_{LRRT}	2	13.2	187.84	**0.00072**	5:14.08
	M_{Or}	8	38	199.03	0.00126	0:32.39
	M_{OrwP}	10	32.8	198.47	0.00111	0:29.12

Table 3.2: Summary of results for 25 runs with $B = 100$ and $B = 200$. The iteration number for M_{LRRT} is set to 200.

Chapter 4

Single-Robot IPP, MCTS Solution

In this chapter, we address the problem of informative path planning with a single robot with applications to environmental monitoring. We present one solution based on MCTS and resampling (RMCTS) and anther based on MCTS and heuristicm method (All-MCTS). The heuristic and Or methods introduced in Section 3.3.1 are used as a benchmark for comparing performance against the other methods. As we discussed the IPP problem in Section. 3.1, we begin this chapter by reviewing the MCTS basics. The work shown here was originally presented in [76].

4.1 MCTS based Algorithms

We start proposing two methods for single robot information acquisition that in the next chapter will be extended for the multi-robot case. In both methods the 2D area of interest $\mathcal{U}$ is discretized by partitioning it into a uniform grid where robot occupies one grid cell. In this section, we introduce two methods, i.e., the all grid MCTS algorithm (All-MCTS) and the sampling-based MCTS (RMCTS).

4.1.1 Monte Carlo tree search (MCTS)

MCTS is an online method for solving sequential, stochastic decision making problems. MCTS builds a tree with a root node representing the current state and edges connecting states that can be reached by executing a single action. Nodes subsequently added to the tree represent states that can be reached through a sequence of actions originating at the root. Each action is assigned a Q-value representing *how good* the action is, which is an estimate of the value that will be obtained through a complete execution starting with that action. Once the tree has been constructed, an action is selected from those available at the root. Upon execution of the selected action, the tree is discarded and rebuilt with the next state as its root. A basic version of MCTS consists of the following four steps [86, 17] as illustrated in Figure 4.1.

- **Selection:** Using the so-called *tree policy*, a path from the root to a leaf node is selected.
- **Expansion:** From the selected leaf node, one or more child nodes are added to the tree.

- **Rollout**: A complete episode is simulated from the selected leaf node, or from one of its newly added child nodes (if any). During this simulation, a simple, suboptimal policy is used to decide the actions.
- **Backup:** Based on the return generated by the simulated episode, the action values attached to the tree edges traversed by the tree policy are updated, or initialized.

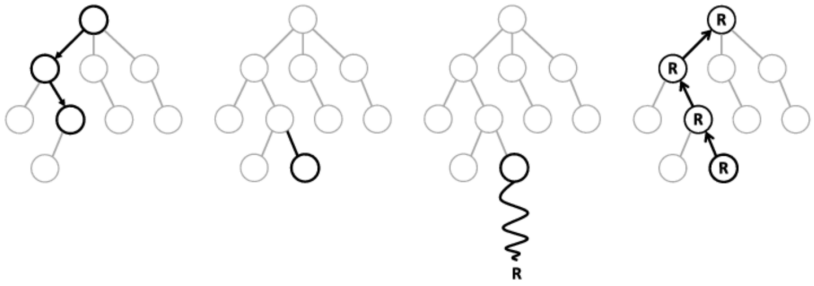


Figure 4.1: Phases of the Monte Carlo tree search algorithm [86].

A critical component is the tree policy for action selection ("selection" step in the list above). One popular criterion for action selection is the Upper Confidence Bounds, UCT rule defined in Equation (4.1) and first introduced in [45]. Each candidate action a is assigned a $UCT(a)$ value defined by,

$$UCT(a) = Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \tag{4.1}$$

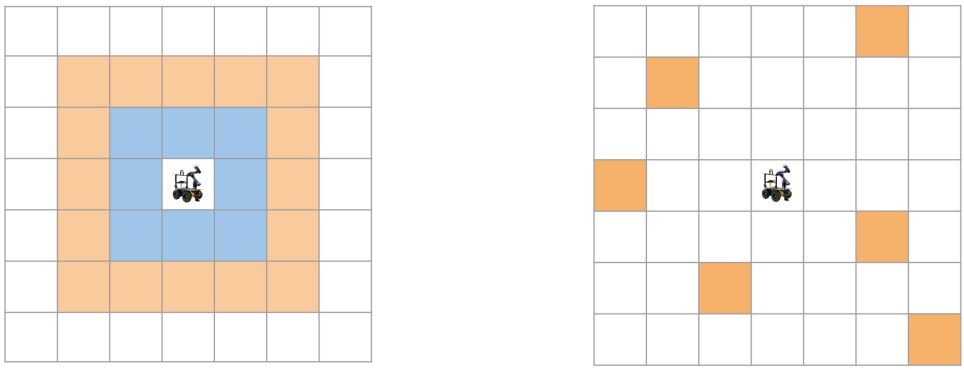
and eventually the action with the highest UCT value is selected for execution. In Equation (4.1), $Q_t(a)$ denotes the action value estimate, $N_t(a)$ is the number of times that action a has been selected prior to time t, and c is a constant controlling the exploration. Initially, $N_t(a)$ is zero for all actions and $UCT(a)$ is assumed to be ∞ when $N_t(a) = 0$, thus forcing exploration. Every time a is selected, t and $N_t(a)$ increase, and every time a is not selected, t increases but not $N_t(a)$, ensuring that all actions will eventually be selected, but actions with lower value estimates or those that have already been selected frequently will be selected less frequently. This criterion balances exploration and exploitation, with the balance determined by the constant parameter c.

4.1.2 All Grid MCTS Algorithm (All-MCTS)

The All-MCTS method defined in Algorithm 4.2 has been built upon the AdaptGP-MCTS discussed in [85]. The main difference is that the method presented in [85] operates on a continuous domain and correspondingly assumes that robots can move to any point by executing a set of preassigned motion primitives tailored to the robot motion model. In addition, [85] does not consider a finite travel budget. The method

presented in this section, instead, operates on a grid and aims at reaching the preassigned goal location before the robot runs out of energy.

Each robot's next sampling location is selected using the MCTS algorithm where the current location of the robot s_s is considered as the root node of the MCTS tree. MCTS expands the tree by adding some, or all, of the node's children to the tree. A children set contains the locations that can be reached from each sample location. Each grid cell (and therefore each robot position) has an associated children set $\Psi[s_s]$ including all locations that can be reached through the execution of a single motion action (north, south, east, west, north-east, north-west, south-east, south-west). The children set $\Psi[s_s]$ includes all grid neighbors located one or two hops away from the current robot location (see Figure 4.2(a).) The highlighted area represents the children; one step neighbors are shown in blue and two step neighbors are in orange color. Due to the fact that the children set includes all possible neighbors surrounding the robot's current location, we call this algorithm All-MCTS.



(a) (b)

Figure 4.2: (a) Robot's children set $\Psi[s_s]$ in the All-MCTS method. One step neighbors are shown in blue and two step neighbors are in orange color. (b) Candidate locations in the RMCTS method consisting of a set of locations $\mathcal{V}$ scattered in the environment.

In addition to one-step and two-step neighbours, the final location s_f is always added to the children set. The addition of the final location ensures that from any location robot R can always consider moving to the final goal location. This is useful when the travel budget is about to expire. The reward r_g associated with each neighbor potential sampling location s_g considered by robot R is defined as (as per the original method in [85]):

$$r_g = \sigma + \beta^{1/2}\mu \tag{4.2}$$

where σ is the variance of the candidate location, μ is mean, and β is the number of measurements collected along the way. The second term in equation 4.2 encourages

the robot to visit locations with high values for the underlying field before exploring locations with higher variance. The trade-off is mediated by the factor β.

The selection of the next location is performed online, i.e., the reward associated with each location is not predetermined, but re-estimated iteratively based on the locations already visited and data previously collected using standard GP regression algorithms. That is to say that in Equation (4.2), σ and μ are the values estimated by the GP regression algorithm based on the samples collected thus far.

The MCTS is expanded for a fixed number of iterations, and at each iteration, the path and leaf are chosen based on UTC formula introduced in Equation (4.1). Once a leaf is reached, as per the MCTS framework a rollout is executed randomly to estimate the quality of the leaf, i.e., its Q value. In our implementation we use a simple random rollout, i.e., the planner continues to select additional random locations from the unvisited children set until it either reaches the final destination or runs out of energy. In this phase of running the rollout, the GP distribution is not updated. During the MCTS expansion and rollout, every time a candidate location is included in the tree, a generative model is used to estimate how much energy would be consumed. c_s^g is an estimate of energy consumed by the robot when it moves from its current location s_s to the designated candidate location s_g and is given by the formula defined in Equation (4.3)

$$c_s^g = \alpha d(s_s, s_g) + k \cdot \varepsilon(1, d) \tag{4.3}$$

where $d(s_s, s_g)$ is the Euclidean distance between the current location, s_s and candidate location s_g, k is a constant number and ε is a random sample from a uniform distribution over the interval $[1, d]$. We use the uniform distribution over the interval $[1, d]$ because added noise must be directly proportional to distance traveled. Therefore, when robot travels long distances, it is more likely to consume more energy.

After the tree $\mathcal{T}$ has been built, the next location s_g is selected based on the UCT formula defined in Equation (4.1). The robot moves to s_g and collects a sample. Then, the budget of the robot is updated by considering the amount of energy consumed during the motion and the GP is updated based on the value read at s_g. The process continues until the robot reaches the final destination, which is a success, or it runs out of energy, which is a failure. Algorithm 4.2 sketches the process.

Algorithm inputs are initial location, s_{init}, final location, s_f, and assigned budget, B. In the beginning, the visited locations set contains the initial location, and the current location is also the initial location. In order to avoid revisiting the same locations, the input of the MCTS planner should be the *cand* set instead of the children set itself, $\Psi[s_s]$. The *cand* set is the children set with the visited locations removed. This algorithm returns the remaining budget and the set of visited locations.

4.1.3 Sample Location MCTS Algorithm (RMCTS)

The RMCTS method differs from All-MCTS in two ways, i.e., first in how the children set is defined, and second in how the reward is assigned to candidate locations. The children set in All-MCTS method described in Section 4.1.2 included all grids

Algorithm 4.2 Online All-MCTS planner for robot R with limited Budget B

Input: s_{init}, s_f, B
$\mathcal{A} \leftarrow \{s_{init}\}$
$s_s \leftarrow s_{init}$
while $B > 0$ **and** $s_s \neq s_f$ **do**
    $cand \leftarrow \Psi[s_s] \setminus \mathcal{A}$
    $\mathcal{T}, s_g \leftarrow \text{MCTS}(s_s, cand)$
    Move to s_g, collect reading x_g, and compute consumed energy c_s^g
    $\sigma_g, \mu_g \leftarrow$ update GP with new observation x_g
    $B \leftarrow B - c_s^g$ (defined in Eq. (4.3))
    $\mathcal{A} \leftarrow \mathcal{A} \cup \{s_g\}$
    $s_s \leftarrow s_g$
end while
return B, $\mathcal{A}$

surrounding the current location for sampling, but children set in this method consists of n preassigned sample locations of interest scattered in the environment that are identified a-priori by domain experts based on past experience. In this case, we have one more input to the Algorithm 4.3, i.e., the set $\mathcal{V} = \{s_1, s_2, \ldots, s_n\}$ of candidate locations. $\mathcal{V}$ includes the number of locations and their placement in $\mathcal{U}$ in such a way that no robot has sufficient budget to visit all of them, otherwise the problem becomes trivial (see Figure 4.2(b)). This setup is similar to what we considered in our former works described in Section 3.2, and is informed by practices implemented in precision agriculture (e.g., the definition of sentinel locations to be monitored through the growing season). For RMCTS, the children set $\Psi[s_g]$ contains the locations that can be reached from s_s. Problem instances may have tens or hundreds of possible locations, so considering them all would result in search trees with extremely high branching factors, and this would be unmanageable. To minimize planning time, we limit the locations that are considered from each location in children set, which is returned by Ψ. If M (an even number) is considered for the branching factor, $M/2$ elements in Ψ are the nearest elements in $\mathcal{V}$, while $M/2-1$ are chosen randomly from the rest. This selection balances global exploration and local exploitation. Additionally, the final location s_f is always added to children set Ψ. As in the previous method, the addition of s_f to Ψ ensures that from any location robot R can always consider moving to the final location. This is useful when the travel budget is about to expire. In RMCTS, each candidate location is assigned a reward r_g as per the following formula which favors locations with high uncertainty (σ_g) and small distance:

$$r_g = \frac{\sigma_g^2}{d(s_s, s_g) + k \cdot \varepsilon(1, d)} \tag{4.4}$$

where $d(s_s, s_g)$ is the Euclidean distance between the current location, s_s and candidate location s_g, k is a constant number and ε is a random sample from a uniform

distribution over the interval $[1, d]$. The estimate of the cost in this method is given by the formula defined in Equation 4.3.

Algorithm 4.3 Online RMCTS planner for robot R with limited Budget B

1: **Input**: s_{init}, s_f, B, $\mathcal{V}$
2: $\mathcal{A} \leftarrow \{s_{init}\}$
3: $s_s \leftarrow s_{init}$
4: **while** $B > 0$ **and** $s_s \neq s_f$ **do**
5: $\quad cand \leftarrow \Psi[s_s] \setminus \mathcal{A}$
6: $\quad \mathcal{T}, s_g \leftarrow$ MCTS(s_s, $cand$)
7: $\quad$ Move to s_g, collect reading x_g, and compute consumed energy c_s^g
8: $\quad \sigma_g^2, \mu_g \leftarrow$ update GP with new observation x_g
9: $\quad \mathcal{V} \leftarrow$ Resampling based on new σ_g^2
10: $\quad B \leftarrow B - c_s^g$ (defined in Eq. (4.3))
11: $\quad \mathcal{A} \leftarrow \mathcal{A} \cup \{s_g\}$
12: $\quad s_s \leftarrow s_g$
13: **end while**
14: **return** B, $\mathcal{A}$

Algorithm inputs are the initial location, s_{init}, the final location, s_f, the assigned budget, B and the set of sample locations, $\mathcal{V}$. In the beginning, the visited locations set contains the initial location, and the current location is also the initial location. In order to avoid revisiting the same locations, the input of the MCTS planner should be the *cand* set instead of the children set itself, $\Psi[s_s]$. The *cand* set is the children set with the visited locations removed.

The current location of the robot, s_s, is considered as a root node of the MCTS tree (line 6 in Algoeithm 4.3). The MCTS is expanded for a fixed number of iterations. Each time, the path and leaf are chosen using UCT, as per Equation (4.1). When a leaf is reached, a children is chosen randomly from unvisited ones and a rollout is executed. During rollout, the planner continues to select additional random locations until it either reaches the final destination or runs out of energy. During the MCTS expansion and rollout, every time a candidate location is included in the tree, the same generative model used in previous section based on Equation 4.3 is used to estimate how much energy would be consumed.

After the tree $\mathcal{T}$ has been built and next location, s_g, is selected, the budget for robot is updated. Based on new sample reading, the GP is updated and based on the updated GP, it will generates a new set of random sample locations with the normalized variance as probabilities associated with each location. This algorithm returns the remaining budget and the set of visited locations.

4.2 MCTS based Algorithms Results and Discussion

In this section, we evaluate the two proposed methods in different environments under different constraints. As the RMCTS selects sample locations that are potentially far away while All-grid MCTS always selects nearby locations, we allow RMCTS to also collect samples along the way towards the sample location. This ensures that in both cases the number of collected samples is comparable.

4.2.1 Methods

We evaluate the two proposed methods (All-MCTS and RMCTS) with Orienteering method (called Or in the following) explained in Section 3.3.1 and a policy gradient method (MRS) [52] using four different datasets. As we explained the Or method in Section 3.3.1, here we briefly discuss the policy gradient method.

Policy Gradient Method (MRS)

The MRS algorithm and its implementation are described in greater detail in [52]. The objective of a Reinforcement learning is to maximize the expected reward when following a policy π. Dynamic programming or other iterative techniques may be used to determine a sequence of actions (in our case sample locations) that maximizes long-term rewards [52]. However, in some cases, the state space (number of sample locations in agricultural fields) and the action space (in our case number of sample locations in agricultural fields) can be enormous and finding the optimal policy is not possible. So, rather than relying on action-value methods, it is possible to use methods that directly optimize the policy parameters based on (simulated) experiences.

By learning the values of actions, action-value methods select the appropriate action based on those values. Without those estimates, the policy would not exist. Rather than learning action-value methods, MRS introduces a method that learns a parameterized policy which allows actions to be selected without consulting a value function [87]. So, the goal would be learning the policy parameter θ, for $\pi(a|s,\theta)$, based on the gradient of the performance cost function $J(\theta)$ with respect to the policy parameter. One method of policy parameterization is soft-max action preferences which is according to an exponential soft-max distribution [86] and is formulated as follows:

$$\pi(a|s,\theta) = \frac{e^{h(s,a,\theta)}}{\sum_b e^{h(s,b,\theta)}} \tag{4.5}$$

where $h(s,a,\theta)$ is the parameterization for each pair of state-action. As an example, they might be calculated by a deep artificial neural network (ANN), where θ is the vector of all the network's connection weights.

In its original formulation, this method does not consider budget constraints during training. To account for it, during testing the next location proposed by the

algorithm is rejected if there is not enough budget left to reach the designated location and then the robot moves to the final location $s_f^{R_i}$. In this case, the algorithm selects $s_f^{R_i}$ and tries to reach the final location.

4.2.2 Data Sets

Synthetic Dataset (SYNT1)

Here we consider bidimensional grid of size 30×30 where the scalar field h is defined as a mixture of Gaussian distributions. Figure 4.3 (a) displays the synthetic data set used in these experiments.

California Central Valley soil moisture dataset (CAL-SOIL)

The California Central Valley soil moisture dataset is a self developed dataset featuring soil moisture data manually collected and interpolated in a commercial vineyard located near Merced, California [93]. The particular vineyard examined was not rectangular, however for the purposes of this study it was assumed to be. Figure 4.3 (b) shows the scalar field modeled by the California Central Valley soil moisture dataset.

NASA Chlorophyll concentration dataset (NASA1, NASA2)

The NASA chlorophyll concentration dataset includes measures collected on Sep 1, 2023, obtained from NASA Earth Observations from a Pacific ocean subregion. The purpose of this satellite observation is to determine how much phytoplankton is growing in the ocean. The green color of phytoplankton is due to chlorophyll [3]. All two datasets, include a scalar datafield to be estimated through measurements. Figure 4.4 (a), and (b) show different patches from the data set.

4.2.3 Metrics

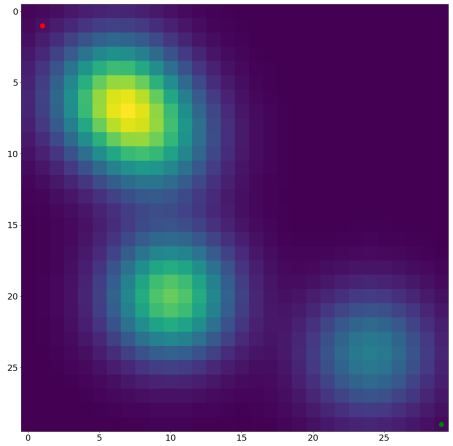
To assess and compare the performance of the various algorithms, we consider three metrics.

MSE

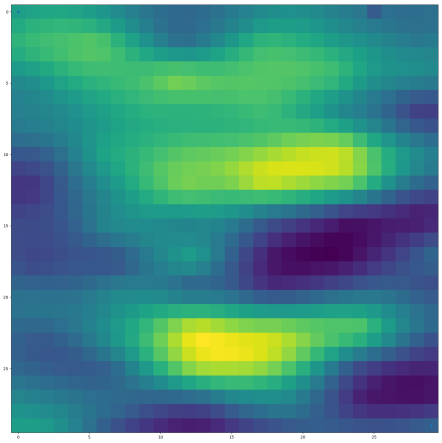
The first is the mean square error (MSE) between $\hat{h}$ (the estimate of h) and h itself.

$$MSE = \frac{1}{n}\Sigma_{i=1}^{n}(h_i - \hat{h}_i)^2 \tag{4.6}$$

where n is the number of cells in the grid. In our implementation, GP regression is computed using the *scikit-learn* Python library [61] and its GP regression module using Mattérn Kernel with length scale of 1 and smoothness parameter of $\nu = 1.5$. The choice of the kernel and of the parameters was made after having experimentally

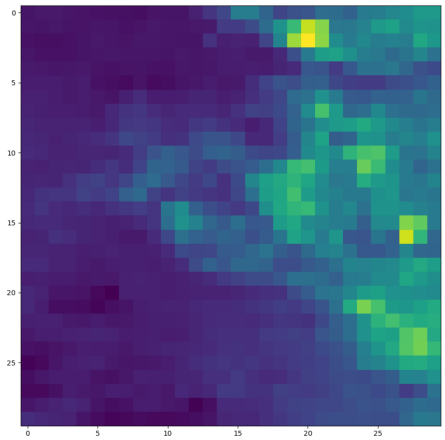


(a) SYNT1

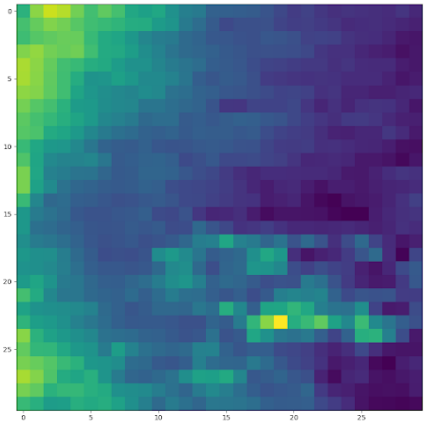


(b) CAL-SOIL

Figure 4.3: (a) The synthetic scalar field modeled by the mixture of Gaussian distributions. (b) the scalar field modeled by the California Central Valley soil moisture dataset. In both cases, warmer colors indicate higher values for the underlying scalar field h.



(a) NASA1



(b) NASA2

Figure 4.4: (a) and (b) The scalar field modeled by the NASA chlorophyll concentration dataset. In all cases, warmer colors indicate higher values for the underlying scalar field h.

evaluated different alternatives and having assessed that these are the best choices. All algorithms build the posterior for h over all grid cells based on all collected measurements.

Remaining Budget

The second metric we consider is the remaining budget, which is the amount of energy that has not been used when the mission terminates. Ideally, this value should be low to ensure robot makes the best of use of its allocated energy resource.

Number of Failures

Finally, we want to minimize the number of failures of the robot. In this context, a robot failure is defined as the event when the robot runs out of energy before reaching the final location. As a result, any time B drops below 0 is considered a failure.

4.2.4 SYNT1 Results and Discussion

In this experiment, we compare the performance of the RMCTS with MCTS, and MRS methods using a SYNT1 data-set. The MCTS method works the same as RMCTS without generating new samples at each iteration (line 9 in Algorithm 4.3). In this case, we use 100 sample locations that are distributed throughout the environment. The robot starts from (0, 0) (upper left corner), and the final location is located at (30, 30) (lower right corner). We consider different budgets (100 and 200). For each case, displayed data are averages over 100 independent runs. In all simulations, the MCTS, and RMCTS algorithms add 1000 nodes to the tree. For the function Ψ we set $M = 30$. In Equation (4.1) we set $c = 3$. To calculate the travel cost, we used Equation (5.3) with $\alpha = 0.5$ and $\Lambda = 1$ and Manhattan distance. In RMCTS, the resampling process is applied every other iteration and after each resampling, 100 new samples are computed, $|\mathcal{V}| = 100$.

During training, MRS runs 20 simulated trajectories to update and learn the parameter of policy π, and in the test phase it uses that learned policy to generate an explicit action plan.

Table 4.1 summarizes the results for 100 runs of MCTS, RMCTS, and MRS methods for one robot. It can be seen that RMCTS outperforms other methods with a tight budget, while MRS achieves better MSE with a higher budget. This is expected, since MRS takes samples along the way, whereas RMCTS and MCTS take samples at specific points.

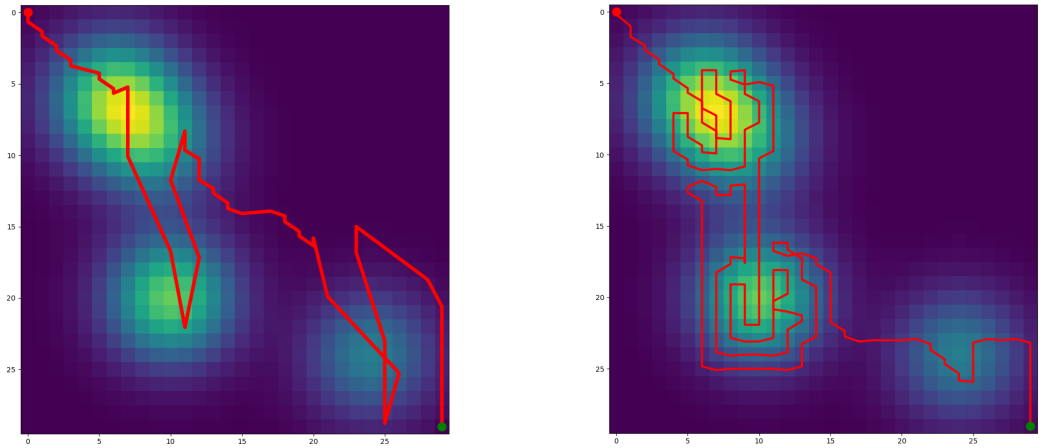
A Figure 4.5 (a) and (b) show the robot's path with $B = 200$ using MRS and RMCTS. Clearly, the MRS focuses on hotspots and drives the robot to locations with higher values. However, RMCTS encourages the robot to explore the entire environment, which results in a better MSE.

4.2.5 CAL-SOIL Results and Discussion

In the first experiment, we compare the performance of the RMCTS with MCTS, Or and MRS methods using a CAL-SOIL data-set. The MCTS method works the

B	method	MSE (std)	B_{re} (std)
100	MCTS	0.52 (0.35)	23.72 (20.05)
100	RMCTS	**0.41 (0.10)**	**21.14 (18.68)**
100	MRS	0.68 (0.01)	27.13 (0.05)
200	MCTS	0.43 (0.28)	28.62 (19.08)
200	RMCTS	0.39 (0.19)	26.17 (17.5)
200	MRS	**0.36 (0.01)**	**15.7 (0.05)**

Table 4.1: Avg. Results for 100 runs for the SYNT1 dataset (MCTS, RMCTS, and MRS methods).



(a) RMCTS (b) MRS

Figure 4.5: (a) and (b) The single robot path with $B = 200$ in SYNT1 environment.

same as RMCTS without any generating new samples at each iteration (line 9 in Algorithm 4.3). In this case, we use 100 sample locations that are distributed throughout the environment. Robot starts from (0, 0) (upper left corner), and the final location is located at (30, 30) (lower right corner). We consider different budgets (100 and 200). For each case, displayed data are averages over 100 independent runs. In all simulations, the MCTS, and RMCTS algorithms add 1000 nodes to the tree. For the function Ψ we set $M = 30$. In Equation (4.1) we set $c = 3$, in Equation (5.2) we set $\lambda = 1$, and in Equation (5.3) we set $\alpha = 0.5$ and $\Lambda = 1$ and we use Manhattan distance. In RMCTS, the resampling process is applied every other iteration and after each resampling, 100 new samples are computed, $|\mathcal{V}| = 100$.

During training, MRS runs 20 simulated trajectories to update and learn the parameter of policy π, and in the test phase it uses that learned policy to generate an explicit action plan.

Table 4.2 summarizes the results for 100 runs of MCTS, RMCTS, Or, and MRS methods for one robot. It can be seen that RMCTS outperforms other methods with a tight budget, while MRS achieves better MSE with a higher budget. The Or method consumes almost all of the budget, but it should be noted that for example in our implementation, the planning time for Or (40.21) is more than five times greater than RMCTS (7.52 s) and MCTS (6.18 s). The fact that the MSE is not significantly better in the Or method even with a higher budget is due to the fact that in orienteering, one aims at collecting the maximum additive reward, and this can be achieved by visiting many nearby locations that will lead to limited additional information to better estimate the scalar field h.

B	method	MSE	B_{re}
100	MCTS	3.16	18.45
100	RMCTS	**2.99**	15.06
100	Or	4.3	**5.78**
100	MRS	6.67	16
200	MCTS	2.99	16.83
200	RMCTS	2.17	12.03
200	Or	2.14	**10.17**
200	MRS	**1.42**	14

Table 4.2: Avg. Results for 100 runs for the CAL-SOIL dataset (MCTS, RMCTS, Or and MRS methods).

A second experiment compares RMCTS with All-MCTS. The parameters are the same as in the first experiment, except for the cost function. To carry out this experiment, we use the formula defined by equation 4.3 with euclidean distance. As the RMCTS selects sample locations that are potentially far away while All-MCTS always selects nearby locations, we allow RMCTS to also collect samples along the way towards the sample location. This ensures that in both cases the number of collected samples is comparable.

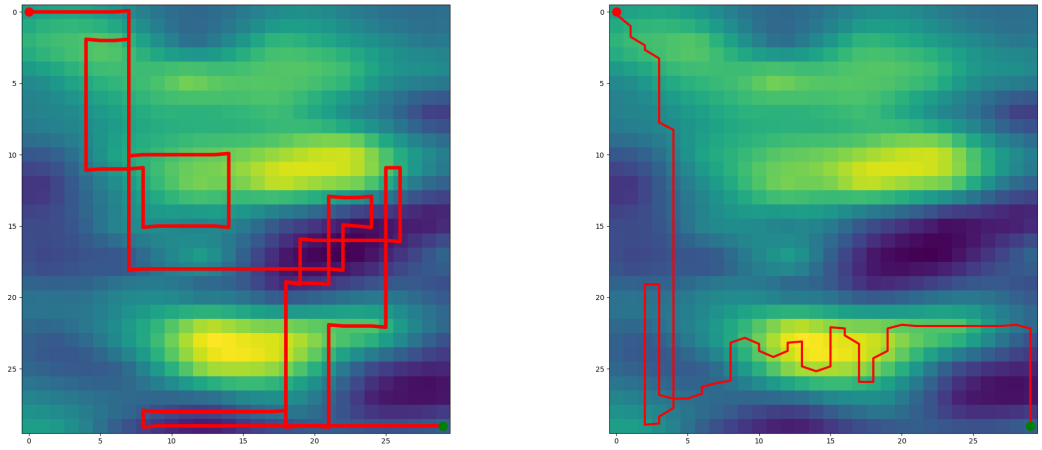
Table 4.3 summarizes the results for 100 runs of RMCTS, and All-MCTS methods for one robot. It can be seen that RMCTS outperforms other methods in all cases. Figure 4.6 illustrates the path taken by a single robot using RMCTS and MRS when $B = 100$. The MRS drives the robot to locations with high values based on hotspots. However, RMCTS encourages the robot to explore the entire environment, resulting in a better MSE.

4.2.6 NASA1 and NASA2 Results and Discussion

In the first benchmark, the RMCTS and All-MCTS methods are tested using a NASA1 dataset. For RMCTS, we have 100 sample locations that are distributed

B	method	MSE (std)	B_{re} (std)	N_f
100	ALL-MCTS	3.81 (0.94)	8.18 (2.17)	11
100	RMCTS	**3.66 (0.79)**	**7.91 (1.09)**	**7**
200	ALL-MCTS	3.61 (0.82)	**12.16 (3.04)**	8
200	RMCTS	**3.27 (0.70)**	13.09 (3.41)	**4**

Table 4.3: Results for 100 runs for CAL-SOIL dataset (RMCTS and All-MCTS methods).



(a) RMCTS (b) MRS

Figure 4.6: (a) and (b) The single robot path with $B = 100$ in CAL-SOIL environment.

throughout the environment. For the function Ψ we set $M = 30$. In Eq. (4.1) we set $c = 3$, and in Eq. (4.3) we use Euclidean distance. Table 4.4 summarizes the metrics for All-MCTS and RMCTS. It can be seen that RMCTS again outperforms All-MCTS and achieves better MSE.

In the second benchmark, RMCTS and All-MCTS methods are tested using a NASA2 dataset. For RMCTS, we have 100 sample locations that are distributed throughout the environment. For the function Ψ we set $M = 30$. In Eq. (4.1) we set $c = 3$, and in Eq. (4.3) we use Euclidean distance. Table 4.5 summarizes the metrics for All-MCTS and RMCTS. It can be seen that RMCTS again outperforms All-MCTS and achieves better MSE. All-MCTS rewards the robot for only visiting hotspot locations, and since there is a hotspot at the beginning of their paths, it spends much of its time exploring it, which means that the rest of the environment cannot be explored due to a limited budget. Alternatively, RMCTS rewards the robot for exploring the entire environment resulting in better performance.

B	method	MSE (std)	B_{re} (std)	N_f
150	ALL-MCTS	0.3 (0.09)	4.46 (3.64)	14
150	RMCTS	**0.21 (0.1)**	**4.17 (3.02)**	**6**
250	ALL-MCTS	0.28 (0.16)	5.18(3.14)	10
250	RMCTS	**0.19 (0.08)**	**4.21(3.67)**	**5**

Table 4.4: Results for 100 runs for the NASA1 chlorophyll concentration dataset (RMCTS and All-MCTS methods).

B	method	MSE (std)	B_{re} (std)	N_f
100	ALL-MCTS	1.12 (0.37)	5.6 (1.74)	14
100	RMCTS	**1.04 (0.29)**	**5.1 (1.08)**	**11**
200	ALL-MCTS	0.91 (0.23)	**9.19 (2.97)**	9
200	RMCTS	**0.74 (0.19)**	10.45 (3.61)	**5**

Table 4.5: Results for 100 runs for the NASA2 chlorophyll concentration dataset (RMCTS and All-MCTS methods).

4.3 Conclusions

An IPP using a single robot was studied in this chapter. We proposed two different online sampling solutions based on the MCTS algorithm. When a sample location is measured, a GP model of the scalar field is updated and used to generate a new set of random sample locations. The Or method and Policy-Gradient-based algorithms are compared against the learning-based methods presented in this chapter. In comparison to baseline methods, our RMCTS proposed approach is more accurate and makes better use of the limited budget. Also, our proposed methods does not require prior knowledge of the environment distribution.

Chapter 5

Multi-Robot IPP

The purpose of this chapter is to extend the problem of environmental map learning and IPP to the use case of a team of robots. In this chapter, the multi robot informative path planning (MRIPP) is formulated and three solution methods for the general case problem are given, one based on MCTS and resampling (MR-RMCTS), another based on MCTS and predicting coworkers actions (MR-PMCTS), and the last based on MCTS and heuristic method (MR-All-MCTS). In this chapter, the heuristic and Or methods are used as a benchmark for comparing performance against the other methods. The work shown here was originally presented in [76].

5.1 Multi-Robot Informative Path Planning

It is assumed that there are m robots in the team, each indicated as R_i with $i \in [1, 2, 3, ..., m]$. Without loss of generality, we assume all robots in the team are equipped with identical sensors used to estimate h. The location of each robot (x, y-position) is denoted by $s_s^{R_i} = (s_x^{R_i}, s_y^{R_i})$. Each robot starts from a preassigned position $s_{init}^{R_i}$, and must terminate at a preassigned final location $s_f^{R_i}$. The location $s_{init}^{R_i}$ models the deployment location for robot R_i (e.g., the location where the robot is activated), while $s_f^{R_i}$ represents the desired end location for the robot (e.g., where the robot can be retrieved or where its batteries can be recharged or swapped). Each robot is assigned a predetermined travel budget B^{R_i} limiting the distance it can travel. This constraint models the limited energy provided by the robot's battery. If a robot exceeds its travel budget before reaching its assigned goal location $s_f^{R_i}$, the robot stops and this is considered a failure, as from a practical standpoint this will require someone to go and retrieve or recharge the robot in the field – a costly operation. For simplicity, we assume in the following that all robots have the same budget B^{R_i}, but the algorithms we present can be easily adapted for the case when this assumption does not hold.

For this approach to be efficient, it is necessary for robots to coordinate their efforts to avoid unnecessary duplicate work or negative interferences [80]. As part of the measurement collection task, sample locations can be provided in advance, chosen by experts or randomly, or selected along the way in response to collected data. Central to our work is the necessity to perform task allocation being aware of the distance a robot can travel before its battery is depleted. Our task allocation

strategy, therefore, focuses both on avoiding duplicate work and on managing the energy constraints.

Communication among robots is another key dimension to be considered in multi-robot scenarios [8] where in some works all robots can share limited amounts of data with one another irrespective of the distance [13], whereas in other approaches more data is exchanged, but only when robots are sufficiently close to each other [52].

We present a distributed algorithm based on communication between robots to avoid having multiple robots collecting measurements at the same locations, as accordingly to our model multiple measurements at the same site do not increase the quality of the estimate.

5.2 MCTS based Algorithms

In this section, three solutions for the MRIPP problem are given, one based on MCTS and resampling (MR-RMCTS), another based on MCTS and predicting coworkers actions (MR-PMCTS), and the last based on MCTS and heuristic method (MR-All-MCTS).

5.2.1 MR-RMCTS

For each robot R_i, let $\mathcal{A}^{R_i} \subseteq \mathcal{V}$ be the set containing the locations visited by robot R_i. Initially $\mathcal{A}^{R_i} = \{s^{R_i}_{init}\}$ and, by definition of $\mathcal{A}^{R_i}$, throughout the execution of the algorithm the current location of the robot $s^{R_i}_s$ is one of the elements of $\mathcal{A}^{R_i}$. Each set $\mathcal{A}^{R_i}$ is iteratively expanded by the execution of an action a representing a possible next sampling location s_g in $\mathcal{V}$ for robot R_i. The execution of a implies that the robot will move to s_g and collect a sample at that location. Using MCTS, the goal is to select a *good* sequence of sample locations, $\mathcal{A}^{R_i}$, for robot R_i while considering the travel budget, B^{R_i}, and other robots' decisions. The meaning of *good* depends on the choice of objective function(s) and will be discussed shortly.

In our proposed method each robot R_i shares its visited locations with other robots with the objective of avoiding to have multiple robots visiting the same locations. This will lead to more distinct samples being collected and ultimately to a better estimate for h. Our communication model assumes that robots can exchange limited information (such as locations) at long range. This is in line with the current technology (e.g., LoRa [83]) used by robots in agricultural applications and previous works [13, 52].

To select the next location, each robot uses a reward function defined as follows. For each unvisited location s_g, robot R_i defines a value $r^{R_i}_g$. A possible candidate could be $r^{R_i}_g = \sigma^2_g$ where σ^2_g is the variance of the posterior estimate of h provided by GP regression based on the samples collected up to that moment. With this choice, high reward values would be assigned to locations with high uncertainty [13]. In our case, we scale the predicted variance by the distance between the robot's current location

and s_g location. The reward $r_g^{R_i}$ associated with a potential sampling location s_g considered by robot R_i is therefore defined as

$$r_g^{R_i} = \frac{\sigma_g^2}{d(s_g, s_s^{R_i})} \tag{5.1}$$

where σ_g is variance of the candidate location $s_g \in \mathcal{V}$, and $d(s_g, s_s^{R_i})$ is the distance between the current location of robot R_i and s_g. By introducing the distance into the reward function we bias the algorithm to favor closer locations if the predicted variance of two candidates is the same. The selection of the next location is performed online, i.e., the reward associated with each location is not predetermined, but re-estimated iteratively based on the locations already visited and data previously collected. The reward is then used to calculate the expected return, or the action value estimate $Q_t(a)$. In our work we deal with an *episodic* task, i.e., the task always ends after a finite amount of time, either because the robot reaches the final location, or because it runs out of energy. In this case it is typical to define $Q_t(s_g)$ as a function of reward sequence

$$Q_t(s_g) = r_g^t + \lambda r_{g'}^{t+1} + ... + \lambda^{T-t} r_{g''}^T;\ 0 \leq \lambda \leq 1 \tag{5.2}$$

where λ is a factor discounting future rewards and T is the time of the last action. $g, g', ..., g''$ are the selected sample locations and r_g is the reward associated with the sampling location g. Algorithm 5.4 shows the planning algorithm independently executed by each robot.

Algorithm 5.4 Online MR-RMCTS planner (executed by each robot R_i)

Input: $\mathcal{V}$, $s_{init}^{R_i}$, $s_f^{R_i}$, B^{R_i}
$\mathcal{A}^{R_i} \leftarrow \{s_{init}^{R_i}\}$
$s_s^{R_i} \leftarrow s_{init}^{R_i}$
while $B^{R_i} > 0$ **and** $s_s^{R_i} \neq s_f^{R_i}$ **do**
    $cand[s_s^{R_i}] \leftarrow \Psi[s_s^{R_i}] - \mathcal{A}^{R_i}$
    $cand[s_s^{R_i}] \leftarrow cand[s_s^{R_i}] - \mathcal{A}^{R_j}$ for all $j \neq i$
    $\mathcal{T}, s_g \leftarrow$ MCTS($s_s^{R_i}$, $cand[s_s^{R_i}]$)
    Move to s_g, collect reading $x_g^{R_i}$, and measure consumed energy c_s^g
    $\sigma_g^2 \leftarrow$ update GP with new observation $x_g^{R_i}$
    $\mathcal{V} \leftarrow$ Resampling based on new σ_g^2
    $B^{R_i} \leftarrow B^{R_i} - c_s^g$
    $\mathcal{A}^{R_i} \leftarrow \mathcal{A}^{R_i} \cup \{s_g\}$
    $s_s^{R_i} \leftarrow s_g$
    Brodacast($s_s^{R_i}$)
end while
return B^{R_i}, $\mathcal{A}^{R_i}$

The algorithm takes as input the set of $\mathcal{V}$, the initial and final locations $s_{init}^{R_i}$ and $s_f^{R_i}$, and the budget B^{R_i} for the each robot.

A children set $\Psi[s_s]$ contains the locations that can be reached from each location s_s. As we consider problem instances with tens or hundreds of possible locations, considering all of them would lead to search trees with extremely high branching factors. Therefore, to minimize planning time, we limit the set of locations that are considered from each locations, and this set is returned by the function Ψ. More specifically, we limit the branching factor to M (an even number). For each location s_g, $M/2$ elements in Ψ are the nearest elements in $\mathcal{V}$ and $M/2-1$ are randomly chosen from the remaining locations. Moreover the final location, $s_f^{R_i}$ is always added to Ψ. This selection balances global exploration and local exploitation. The addition of $s_f^{R_i}$ to Ψ ensures that from any location robot R_i can always consider moving to the final goal location. This is useful when the travel budget is about to expire. $cand^{R_i}$ is the set of candidate locations for the current location and is obtained by removing already visited locations $\mathcal{A}^{R_i}$ from the set of reachable locations returned by Ψ.

The current location of the robot, $s_s^{R_i}$, is considered as a root node of the MCTS tree (line 7 in Algorithm 5.4). The MCTS is expanded for a fixed number of iterations. Each time, the path and leaf are chosen using UCT, as per Equation (4.1). When a leaf is reached, a rollout is executed. During rollout, the planner continues to select additional random locations until it either reaches the final destination or runs out of energy. During the MCTS expansion and rollout, every time a candidate location is included in the tree, a generative model is used to estimate how much energy would be consumed. This estimate is given by the formula

$$c_s^g = \alpha d(s_s^{R_i}, s_g) + \varepsilon(\Lambda) \tag{5.3}$$

where $d(s_s^{R_i}, s_g)$ is the distance between the current location, $s_s^{R_i}$ and candidate location, s_g, and $U(\Lambda)$ is random sample from a uniform distribution over the interval $[0, \Lambda]$.

After the tree $\mathcal{T}$ has been built and next location, $s_g^{R_i}$, is selected, the budget for robot R_i is updated. Based on new sample reading, the GP is updated and based on the updated GP, it will generates a new set of random sample locations with the normalized variance as probabilities associated with each location. In the event that one of the robots reaches its final destination or runs out of energy, other robots will continue their sampling task.

5.2.2 MR-PMCTS

Algorithm 5.5 sketches how the single robot SMCTS algorithm explained in Section 4.1.3, Algorithm 4.3 can be extended for multiple robots. Algorithm inputs are the set of sample locations, $\mathcal{V}$, initial location, $s_{init}^{R_i}$, final location, $s_f^{R_i}$, and assigned budget, B^{R_i} for each robot. In the beginning, the visited locations set contains the initial location, and the current location is also the initial location. In order to avoid revisiting the same locations, the input of the MCTS planner should be the *cand* set instead of the children set itself, $\Psi[s_s^{R_i}]$. You can find detailed information about choosing the children set and branching factor in Section 4.1.3.

Throughout this section, we will explain how the *cand* set is generated and will be used to choose the next sampling location. The *cand* set for current location of robot R_i, $s_s^{R_i}$, is given by Equation 5.4

$$cand[s_s^{R_i}] = \Psi[s_s^{R_i}] - \mathcal{A}^{R_i} - \cup_{j=1_{j\neq i}}^{m} \mathcal{A}^{R_j} - \cup_{j=1_{j\neq i}}^{m} s_g^{R_j} \tag{5.4}$$

where $\Psi[s_s^{R_i}]$ is the children set for the current location $s_s^{R_j}$, $\mathcal{A}^{R_i}$ is the set of visited locations by the robot itself, $\mathcal{A}^{R_j}$ is the set of visited locations by other robots and $s_g^{R_j}$ is the predicted next location for other robots. Continuing our discussion, we will explain how we calculate $s_g^{R_j}$.

Our proposed method relies on predicting the next movement of other robots to prevent two robots from visiting the same location simultaneously, which is the main contribution of this section. Simultaneous visits to the same location by multiple robots are undesirable because of possible collisions, but also because duplicate efforts lead to less efficient resource use. The proposed algorithm also balances exploration and exploitation based on energy consumed, remaining energy, and rewards each robot receives after visiting different locations. This will allow each robot to determine its next destination while respecting the budget constraint.

As discussed previously, let us assume there are m robots in the team. In our proposed method, each robot R_i shares its visited locations with other robots. Even though robots share their locations at each iteration, they do not share the values of the collected samples. Using the data collected at the sample locations, a posterior of h can be estimated using standard GP regression algorithms. As this posterior was created based on local data from one robot, it remains local and is not shared. Our communication model assumes that robots can exchange limited information (such as locations) at long range. In particular, LoRa [83] permits transmitting limited data at long distances.

Using the information that robot R_i receives from other robots, it removes already visited locations by other robots and the next decision (sample location) of the other robots as well. Robot R_i predicts the next decision (the next sample location that will be visited), $s_g^{R_j}$ of other robots, R_j, using the following equation

$$r_g^{R_j} = \frac{\sigma_g^2}{d(s_s^{R_j}, s_g) + k \cdot \varepsilon(1, d)} \tag{5.5}$$

where σ_g is variance of the candidate location $s_g \in \mathcal{V}$, and $d(s_s^{R_j}, s_g)$ is the Euclidean distance between the current location of robot R_j, $s_s^{R_j}$, and selected location, s_g, k is a constant and $\varepsilon(1, d)$ is random sample from a uniform distribution over the interval $[1, d]$. If two candidates' predicted variances are the same, adding the distance to the reward function biases the algorithm toward closer locations. In Equation (5.5) as in Equation (4.4), noise is added in the predictive models to account for the uncertainty in travel costs. Robot R_j's current location is considered the last location that it has shared. In this model, σ_g is robot R_i's local variance at location s_g, since robots do not share their sensory information, even though they share the locations they have visited.

Lemma 5.2.1. *In case s_r has been predicted in step k as the next sampling location of robot j, i.e $s_g^{R_j}$, which will be eliminated from candidate locations of current location robot R_i, s_r still has a chance of being considered in candidate location of robot R_i and being chosen in step $k+m$, where $m \geq 1$. (In other words, removing the nodes from candidates in step k does not remove them permanently)*

Proof. Let us consider sample location s_r is in the children set in step k, $\Psi[s_k] = \{s_l, ..., s_r, ..., s_f\}$ and in step $k+m$, $\Psi[s_{k+m}] = \{s_t, ..., s_r, ..., s_f\}$ for location s_k and s_{k+m}. Let us assume in step k, s_r has been removed from $cand[s_k^{R_i}]$ because it has been predicted as a $s_g^{R_j}$, so $cand[s_k^{R_i}] = \{s_l, , ..., s_f\}$.

In step $k+m$, if s_r has not visited by other robots and has not predicted as a next sampling location by other robots, then it will appear in the $cand[s_{k+m}^{R_i}]$ in step $k+m$ that it has been removed from it previously in step k (that is due to the fact that in each step and for each location, the $\Psi[s_k]$ are independent from each other) and then it has a chance of being chosen based on Equation 5.5.

□

The MCTS algorithm is used to select the next location to be visited, s_g. The current location of the robot R_i, $s_s^{R_i}$ is considered as a root node of the MCTS tree.

Considering all visited locations by robot R_i and other robots $j \neq i$ that have been shared with robot R_i and all predicted locations for other robots $j \neq i$, the next location, s_g will be chosen as follows:

$$\begin{aligned} &s_g = \arg\max Q_t(s_g) \text{ for } s_g \in cand[s_s^{R_i}] \\ &\text{with} \\ &cand[s_s^{R_i}] = \Psi[s_s^{R_i}] - \mathcal{A}^{R_i} - \cup_{j=1_{j\neq i}}^{m} \mathcal{A}^{R_j} - \cup_{j=1_{j\neq i}}^{m} s_g^{R_j} \end{aligned} \tag{5.6}$$

where $\Psi[s_s^{R_i}]$ is the children set for the current location $s_s^{R_i}$, $\mathcal{A}^{R_i}$ is the set of visited locations by the robot itself, $\mathcal{A}^{R_j}$ is the set of visited locations by other robots, and $s_g^{R_j}$ is the predicted locations visited by the other robots in a team. The robot predicts the next location of other robots using Equation 5.5. In Equation 5.6, $Q_t(s_g)$ is defined as a function of the reward sequence

$$Q_t(s_g) = r_g^t + \lambda r_{g'}^{t+1} + ... + \lambda^{T-t} r_{g^T}^T;\ 0 \leq \lambda \leq 1$$

where λ is a factor discounting future rewards, T is the time of the last action, $g, g', ..., g^T$ are selected sample locations and r_g is the reward associated with the sampling location g.

For each unvisited location s_g, robot R_i defines a value $r_g^{R_i}$. A possible candidate could be $r_g^{R_i} = \sigma_g^2$ where σ_g^2 is the variance of the posterior estimate of h provided by GP regression based on the samples collected up to that moment. With this choice, high reward values would be assigned to locations with high uncertainty [13]. In our case, we scale the predicted variance by the distance between the robot's current

location and s_g location. The reward $r_g^{R_i}$ associated with a potential sampling location s_g considered by robot R_i is therefore defined as

$$r_g^{R_i} = \frac{\sigma_g^2}{d(s_s^{R_i}, s_g) + 0.1 \cdot \varepsilon(1, d)} \tag{5.7}$$

where σ_g is variance of the candidate location $s_g \in \mathcal{V}$, $d(s_s^{R_i}, s_g)$ is the Euclidean distance between the current location of robot R_i and s_g and $\varepsilon(1, d)$ is random sample from a uniform distribution over the interval $[1, d]$. By introducing the distance into the reward function we bias the algorithm to favor closer locations if the predicted variance of two candidates is the same.

Algorithm 5.5 Online MR-PMCTS planner (executed by each robot R_i)

Input: $\mathcal{V}$, $s_{init}^{R_i}$, $s_f^{R_i}$, B^{R_i}
$\mathcal{A}^{R_i} \leftarrow \{s_{init}^{R_i}\}$
$s_s^{R_i} \leftarrow s_{init}^{R_i}$
while $B^{R_i} > 0$ **and** $s_s^{R_i} \neq s_f^{R_i}$ **do**
    $cand[s_s^{R_i}] = \Psi[s_s^{R_i}] - \mathcal{A}^{R_i}$
    for all $j \neq i \,\&\, j \in m$ **do**
        estimate $s_g^{R_j}$ based on $\arg\max r_g^{R_j}$ (Eq. (5.7))
        $cand[s_s^{R_i}] \leftarrow cand[s_s^{R_i}] - s_g^{R_j}$
        $cand[s_s^{R_i}] \leftarrow cand[s_s^{R_i}] - \mathcal{A}^{R_j}$
    end for
    $\mathcal{T}, s_g \leftarrow$ MCTS($s_s^{R_i}$, $cand[s_s^{R_i}]$)
    Move to s_g, collect reading $x_g^{R_i}$, and measure consumed energy c_s^g
    $\sigma_g^2 \leftarrow$ update GP with new observation $x_g^{R_i}$
    $B^{R_i} \leftarrow B^{R_i} - c_s^g$
    $\mathcal{A}^{R_i} \leftarrow \mathcal{A}^{R_i} \cup \{s_g\}$
    $s_s^{R_i} \leftarrow s_g$
    Brodacast($s_s^{R_i}$)
end while
return B^{R_i}, $\mathcal{A}^{R_i}$

Lemma 5.2.2. *If the robots' current locations are different, but their distance to the candidate location is the same, they will not choose the same location.*

Proof. The next location is selected based on $\text{argmax}\, Q_t(s_g)$ for s_g where $Q_t(s_g)$ is defined as a function of reward sequence

$$Q_t(s_g) = r_g^t + \lambda r_{g'}^{t+1} + ... + \lambda^{T-t} r_{g^T}^T;\ 0 \leq \lambda \leq 1$$
$$Q_t(s_g) = \frac{\sigma_g^2}{d(s_s^{R_i}, s_g) + 0.1\epsilon(1, d)} + \lambda \frac{\sigma_{g'}^2}{d(s_s^{R_i}, s_{g'}) + 0.1\epsilon(1, d)} + ...$$

where λ is a factor discounting future rewards and T is the time of the last action. $g, g', ..., g''$ are selected sample locations and r_g is the reward associated with the sampling location g. As it can be seen, r_g^t depends on the distance metrics plus random sample from a uniform distribution, so even though the variance may be equal, the distance will not be similar since it uses random samples and the rewards will not be equal either. □

The planning algorithm determines the next location for R_i with the objective of avoiding to have multiple robots visiting the same locations. By avoiding to have multiple robots revisit the same places, more sampling locations can be visited and then the final reconstruction of the scalar field h will be better since it will be informed by more data. The selection of the next location is performed online, i.e., the reward associated with each location is not predetermined, but re-estimated iteratively based on the locations already visited and data previously collected.

5.2.3 MR-All-MCTS

In this section, we extend the All-MCTS method defined in Algorithm 4.2 which has been built upon the AdaptGP-MCTS discussed in [85] to multi robot scenarios. The method presented in this section, operates on a grid and aims that each robot R_i in a team reaches the preassigned goal location before the robot runs out of energy.

Each robot's next sampling location is selected using the MCTS algorithm where the current location of the robot $s_s^{R_i}$ is considered as the root node of the MCTS tree. MCTS expands the tree by adding some, or all, of the node's children to the tree. A children set contains the locations that can be reached from each sample location. Throughout this section, we consider each grid cell (and therefore each robot position) has an associated children set $\Psi[s_s^{R_i}]$ including all locations that can be reached through the execution of a single motion action (north, south, east, west, north-east, north-west, south-east, south-west). The children set $\Psi[s_s^{R_i}]$ for robot R_i includes all grid neighbors located one or two hops away from the current robot location (see Figure 4.2(a)). The highlighted area represents the children; one step neighbors are shown in blue and two step neighbors are in orange color. Due to the fact that the children set includes all possible neighbors surrounding the robot's current location, we call this algorithm MR-All-MCTS. In addition to one-step and two-step neighbours, the final location s_f is always added to the children set. The addition of the final location ensures that from any location each robot R_i can always consider moving to the final goal location. This is useful when the travel budget is about to expire. The reward r_g associated with each neighbor potential sampling location s_g considered by robot R_i is defined as (as per the original method in [85]):

$$r_g = \sigma + \beta^{1/2}\mu \tag{5.8}$$

where σ is the variance of the candidate location, μ is mean, and β is the number of measurements collected along the way. The β encourages the robot to visit the peak of the environment before exploring unknown areas (higher variance).

The selection of the next location is performed online, i.e., the reward associated with each location is not predetermined, but re-estimated iteratively based on the locations already visited and data previously collected using standard GP regression algorithms. That is to say that in Equation (5.8), σ and μ are the values estimated by the GP regression algorithm based on the samples collected thus far.

At each iteration of the MCTS, the path and leaf are chosen randomly if there are unvisited children or on the basis of the UCT formula defined in Equation (4.1). Then, a rollout is executed once a leaf has been reached to estimate its value, or Q. In our implementation we use a simple random rollout without updating the GP distribution. A generative model is used during the expansion and rollout of MCTS to estimate the energy consumption of each candidate location. c_s^g is an estimate of energy consumed by the robot R_i when it moves from its current location $s_s^{R_i}$ to the designated candidate location s_g and is given by the formula defined in Equation (5.9)

$$c_s^g = d(s_s^{R_i}, s_g) + k \cdot \varepsilon(1, d) \tag{5.9}$$

where $d(s_s^{R_i}, s_g)$ is the Euclidean distance between the current location, $s_s^{R_i}$ and candidate location s_g, k is a constant and ε is a random sample from a uniform distribution over the interval $[1, d]$. We use the uniform distribution over the interval $[1, d]$ but any generic model can be used. After the tree $\mathcal{T}$ has been built, the next location s_g is selected based on the UCT formula defined in Equation (4.1). The robot R_i moves to s_g and collects a sample. Then, the budget of the robot is updated by considering the amount of energy consumed during the motion and the GP is updated based on the value read at s_g. The process continues until the robot reaches the final destination, which is a success, or it runs out of energy, which is a failure. Algorithm 5.6 sketches the process.

Algorithm 5.6 Online MR-All-MCTS planner (executed by each robot R_i)

Input: $s_{init}^{R_i}$, $s_f^{R_i}$, B^{R_i}
$\mathcal{A}^{R_i} \leftarrow \{s_{init}^{R_i}\}$
$s_s^{R_i} \leftarrow s_{init}^{R_i}$
while $B^{R_i} > 0$ **and** $s_s^{R_i} \neq s_f^{R_i}$ **do**
    $cand[s_s^{R_i}] \leftarrow \Psi[s_s^{R_i}] - \mathcal{A}^{R_i}$
    $cand[s_s^{R_i}] \leftarrow [s_s^{R_i}] - \mathcal{A}^{R_j}$ for all $j \neq i$
    $\mathcal{T}, s_g \leftarrow$ MCTS($s_s^{R_i}$, $cand[s_s^{R_i}]$)
    Move to s_g, collect reading $x_g^{R_i}$, and measure consumed energy c_s^g
    $\sigma_g^2 \leftarrow$ update GP with new observation $x_g^{R_i}$
    $B^{R_i} \leftarrow B^{R_i} - c_s^g$
    $\mathcal{A}^{R_i} \leftarrow \mathcal{A}^{R_i} \cup \{s_g\}$
    $s_s^{R_i} \leftarrow s_g$
    Brodacast($s_s^{R_i}$)
end while
return B^{R_i}, $\mathcal{A}^{R_i}$

Algorithm inputs are initial location, $s_{init}^{R_i}$, final location, $s_f^{R_i}$, and assigned budget, B^{R_i} for each robot. In the beginning, the visited locations set contains the initial location, and the current location is also the initial location. In order to avoid revisiting the same locations, the input of the MCTS planner should be the *cand* set instead of the children set itself, $\Psi[s_s^{R_i}]$. The *cand* set is the children set with the visited locations removed. This algorithm returns the remaining budget and the set of visited locations for each robot.

5.3 Results and Discussion

5.3.1 Synthetic Dataset (SYNT1) Results

In this experiment, we compare the MR-RMCTS with three different alternatives. The first is a non-coordinated MCTS method (MR-NCMCTS), the second is MCTS without resampling (MR-MCST), and the third is Multirobot Planning for Informed Spatial Sampling (MR-MRS) [52]. MR-NCMCTS is the same as MR-RMCTS method but without any data shared between robots (line 14 in the Algorithm 5.4). MR-MCTS is also the same as MR-RMCTS method, but without the resampling step (line 10 in the Algorithm 5.4). MR-MRS is the extension of planning method described in section4.2.1 and is a decentralized sampling approach where each robot in a team performs an informed survey using a policy-gradient-based sampling strategy [52].

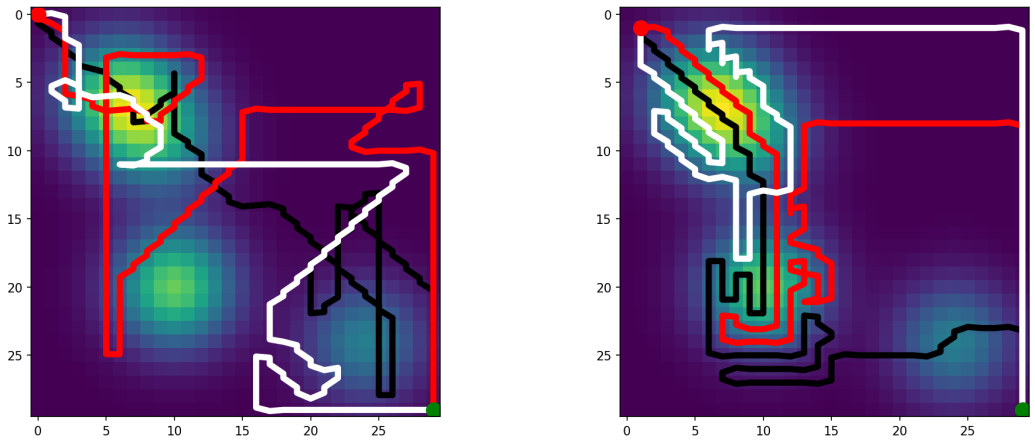
To assess the performance of the various algorithms, we consider two metrics. The first is the mean square error between $\hat{h}$ (the estimate of h) and h itself. In our implementation, GP regression is computed using the *scikit-learn* Python library and its GP regression module using Mattérn Kernel with length scale of 1 and smoothness parameter of $\nu = 1.5$. The choice of the kernel and of the parameters was made after having experimentally evaluated different alternatives and having assessed that these are the best choices. The second metric is the remaining budget, which is the amount of budget that has not been used when the mission terminates.

The synthetic dataset (SYNT1) used in this section is introduced in Section 4.2.2 with the scalar field h is defined as a mixture of Gaussian distributions. The set $\mathcal{V}$ consists of 100 locations. Each robot starts from (0, 0) (upper left corner), and the final location is located at (30, 30) (lower right corner). We consider different numbers of robots (3 and 5) and different budgets (100 and 200). For each case, displayed data are averages over 100 independent runs. In all simulations, the MR-MCTS, MR-NCMCTS, and MR-RMCTS algorithms add 1000 nodes to the tree. For the function Ψ we set $M = 30$. In Equation (4.1) we set $c = 3$, in Equation (5.2) we set $\lambda = 1$, and in Equation (5.3) we set $\alpha = 0.5$ and $\Lambda = 1$ and we use Manhattan distance. In MR-RMCTS, the resampling process is applied every other iteration and after each resampling, 30 new samples are computed, $|\mathcal{V}| = 30$. The MR-MRS method seeks to collect samples at locations with high values for the function h at earlier stages of the exploration. During training, MR-MRS runs 20 simulated trajectories to update and learn the parameter of policy π, and in the test phase it

uses that learned policy to generate an explicit action plan.

Table 5.1 summarizes the average metrics for MR-MCTS, MR-MCTS, MR-MRS and MR-NCMCTS for 100 runs. The table displays the initial budget B, the number of robots in a team N_{R_i}, the average MSE error and average remaining budget for each robot B_{re}. The numerical comparison confirms the superiority of MR-RMCTS across the board. As the budget and number of robots increase, the performance of MR-NCMCTS becomes similar to MR-MCTS. With a greater number of robots and a large budget, it is possible to visit numerous locations even without coordination. However, this is not the case when the number of robots is smaller or the budget is tight, and in such cases coordination is key. As a result of resampling, MR-RMCTS achieves a lower MSE because based on the updated GP it generates a better set of sample locations to explore. A better selection of sample points during mission can give an even better MSE when $\mathcal{V}$ was not chosen wisely at the beginning of the mission. MR-MCTS, MR-NCMCTS and MR-RMCTS do not need any prior knowledge about the prior model of the environment whereas MR-MRS needs to know the prior model as a reward map. Also, MR-MRS requires pre-training, while MR-MCTS and MR-RMCTS are online and can choose the next locations on the fly. To give an order of magnitude, training time for one robot in MR-MRS on Ubuntu machine with 32 G RAM is about 17 minutes, while MR-RMCTS and MR-MCTS do not need training and planning time for both on the same machine is below 15 seconds (cumulative time for all planning stages alternating with execution).

Figure 5.1(a) and 5.1(b) show sample paths for 3 robots with $B = 100$. With MR-MRS, robots focus on areas with high values for the underlying function being reconstructed (warmer colors), while in MR-RMCTS, robots visit all areas. As the goal is to reconstruct the underlying function h with low RMSE error, to do that robots must visit both areas with high and low values.



(a) MR-RMCTS (b) MR-MRS

Figure 5.1: (a)-(b) Three-robots sampling paths with budget $B = 100$ in synthetic environment using MR-RMCTS and MR-MRS.

B	N_{R_i}	method	MSE	B_{re}
100	3	MR-MCTS	0.52	13.72
100	3	MR-RMCTS	**0.47**	**11.23**
100	3	MR-NCMCTS	1.27	14.1
100	3	MR-MRS	1.39	12
100	5	MR-MCTS	0.38	13.21
100	5	MR-RMCTS	**0.35**	11.29
100	5	MR-NCMCTS	0.81	15.04
100	5	MR-MRS	1.19	**6**
200	3	MR-MCTS	0.48	23.72
200	3	MR-RMCTS	**0.41**	**23.13**
200	3	MR-NCMCTS	0.54	27.11
200	3	MR-MRS	1.13	34
200	5	MR-MCTS	0.35	21.32
200	5	MR-RMCTS	**0.32**	**18.71**
200	5	MR-NCMCTS	0.49	20.41
200	5	MR-MRS	0.65	20

Table 5.1: Avg. Results for 100 runs for the synthetic dataset.

5.3.2 California Central Valley Soil Moisture Dataset (CALSOIL) Experiment I Results

In this experiment, we test MR-RMCTS, MR-MCTS, MR-NCMCTS and MR-MRS methods using a data-set for soil moisture outlined in Section 4.2.2. The methods and metrics are introduced in Section 5.3.1. In this set of experiments, we use 100 sample locations that are distributed throughout the environment. The parameters are the same as Section 5.3.1.

Table 5.2 summarizes results for 100 runs of MR-MCTS, MR-RMCTS, and MR-MRS methods for teams of three, five, and ten robots. Similar to other cases, we also find that MR-RMCTS outperforms other methods when the budget is tight, while MR-MRS outperforms other methods when the budget is higher. MR-MRS performs better with higher budgets since it considers the samples' locations at each step rather than our method, which considers samples at specific locations.

Figure 5.2 (a) and (b) show the sampling path for 3 robots with $B = 100$ while Figure 5.3 (a) and (b) show the sampling path for 3 robots with $B = 200$ and Figure 5.4 (a) and (b) show the sampling path for 5 robots with $B = 100$. With MR-RMCTS the robots visit the most informative locations which leads to a more accurate reconstruction of the spatial domain and lower MSE.

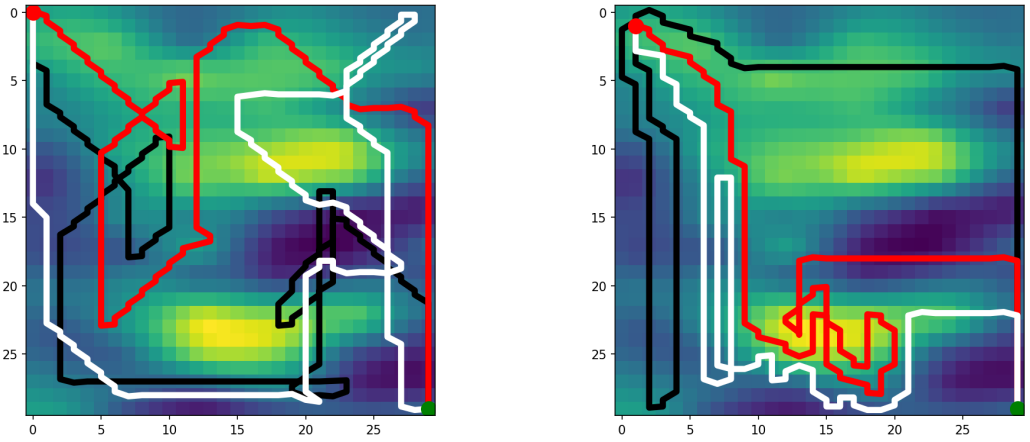
B	N_{R_i}	method	MSE	B_{re}
100	3	MR-MCTS	2.83	12.87
100	3	MR-RMCTS	**2.50**	**10.62**
100	3	MR-MRS	6.67	17
100	5	MR-MCTS	2.53	17.38
100	5	MR-RMCTS	**2.41**	**15.64**
100	5	MR-MRS	6.23	28
200	3	MR-MCTS	2.46	20.66
200	3	MR-RMCTS	2.37	**12.27**
200	3	MR-MRS	**1.08**	13
200	5	MR-MCTS	2.38	16.38
200	5	MR-RMCTS	2.14	14.37
200	5	MR-MRS	**0.10**	**6**
100	10	MR-MCTS	2.17	27.35
100	10	MR-RMCTS	1.84	25.11
100	10	MR-MRS	**0.23**	**12**

Table 5.2: Avg. Results for 100 runs for the CAL-SOIL dataset.

5.3.3 California Central Valley Soil Moisture Dataset (CAL-SOIL) Experiment II Results

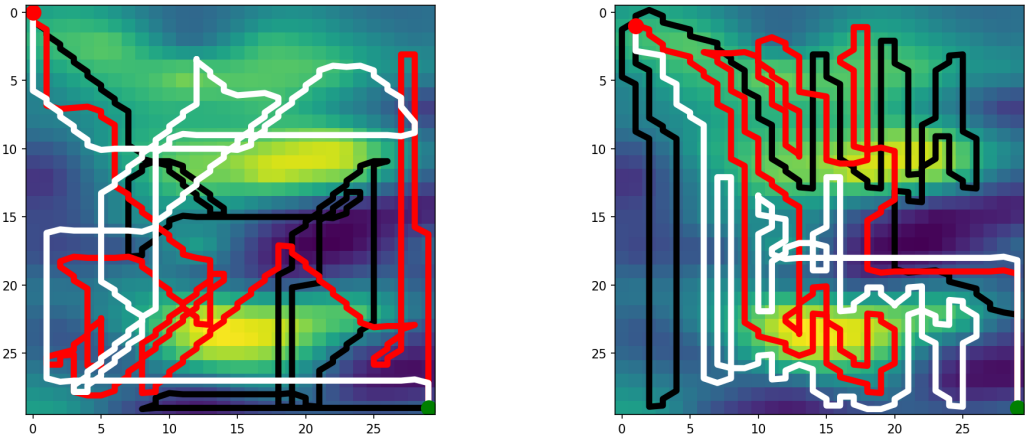
We test and compare the MR-PMCTS and MR-All-MCTS methods on the California Central Valley soil moisture dataset introduced in Section 4.2.2. Metrics are the same as those introduced in Section 4.2.3. For MR-PMCTS, we have 100 sample locations that are distributed throughout the environment. More precisely, their location was determined by sampling from a uniform distribution, i.e., the locations are not informed by the underlying unknown scalar field. For the function Ψ we set $M = 30$. In Equation (4.1) we set $c = 3$, and in Eq. (5.9) we use Euclidean distance. As the MR-PMCTS selects sample locations that are potentially far away while MR-All-MCTS always selects nearby locations, we allow MR-PMCTS to also collect samples along the way towards the sample location. This ensures that in both cases the number of collected samples is comparable.

Table 5.3 summarizes the metrics for MR-All-MCTS and MR-PMCTS for 100 runs. The table displays the budget B, the number of robots N_{R_i}, the average MSE error, average remaining budget for each robot B_{re} and number of failures of each robot N_f. The numerical comparison shows that MR-PMCTS outperforms MR-All-MCTS across the board. As the budget and number of robots increase, the performance of MR-All-MCTS becomes similar to MR-PMCTS. This happens because with a large budget and a large number of robots, it is possible to visit more locations



(a) MR-RMCTS (b) MR-MRS

Figure 5.2: (a)-(b) Three-robots sampling paths with budget $B = 100$ in vineyard environment using MR-RMCTS and MR-MRS.

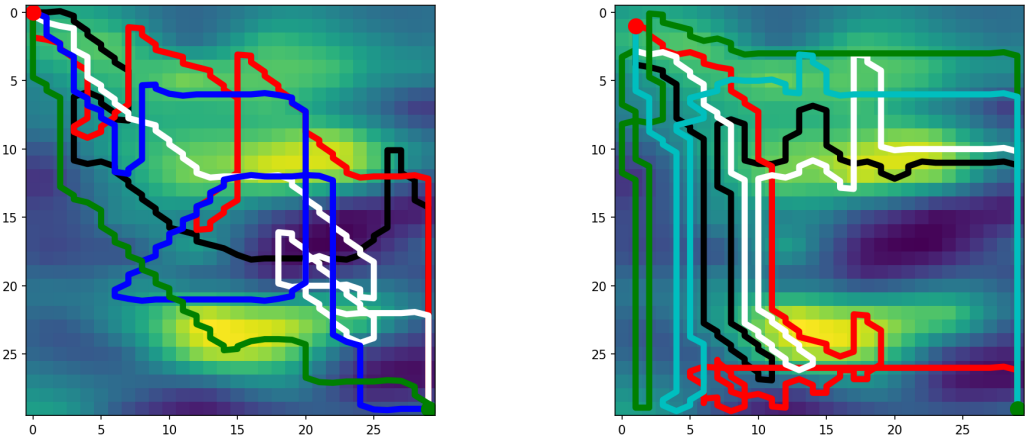


(a) MR-RMCTS (b) MR-MRS

Figure 5.3: (a)-(b) Three-robots sampling paths with budget $B = 200$ in vineyard environment using MR-RMCTS and MR-MRS.

without coordination. The opposite is true when the number of robots is smaller or the budget is tight, and in such cases coordination is essential. This is ensured by the different reward function used by MR-PMCTS. Robots are encouraged to explore the entire area by the reward function introduced in Equation 5.7 rather than focusing on the hotspots.

Figures 5.5(a) and 5.5(b) show the paths taken by 5 robots with $B = 100$ using MR-PMCTS and MR-All-MCTS, respectively. In MR-PMCTS, the reward function encourages the robots to explore the entire environment while in MR-All-MCTS, the reward function drives them to visit hotspot locations first, and thus they cannot explore the rest of the environment due to budget constraints. Therefore, with tighter budgets, MR-PMCTS perform better than MR-All-MCTS.



(a) MR-RMCTS (b) MR-MRS

Figure 5.4: (a)-(b) Five-robots sampling paths with budget $B = 100$ in vineyard environment using MR-RMCTS and MR-MRS.

5.3.4 NASA Chlorophyll Concentration Dataset (NASA3) Results

In this benchmark, MR-PMCTS and MR-All-MCTS methods are compared using a Chlorophyll concentration dataset introduced in Section 4.2.2. Metrics are the same as those introduced in Section 4.2.3. For MR-PMCTS, again, we have 100 sample locations that are distributed throughout the environment. For the function Ψ we set $M = 30$. In Equation (4.1) we set $c = 3$, and in Equation (5.9) we use Euclidean distance. Table 5.4 summarizes the metrics for MR-All-MCTS and MR-PMCTS. It can be seen that MR-PMCTS again outperforms MR-All-MCTS with a tight budget, while MR-All-MCTS achieves better MSE with a higher budget.

Figure 5.6 (c) and (d) show the paths taken by 5 robots with $B = 100$ using MR-PMCTS and MR-All-MCTS, respectively. MR-All-MCTS rewards the robots for only visiting hotspot locations, and since there is a hotspot at the beginning of their paths, they spend much of their time exploring it, which means they can not explore the rest of the environment due to a limited budget. Alternatively, MR-PMCTS rewards the robot for exploring the entire environment resulting in better performance with a tight budget.

5.4 Conclusions

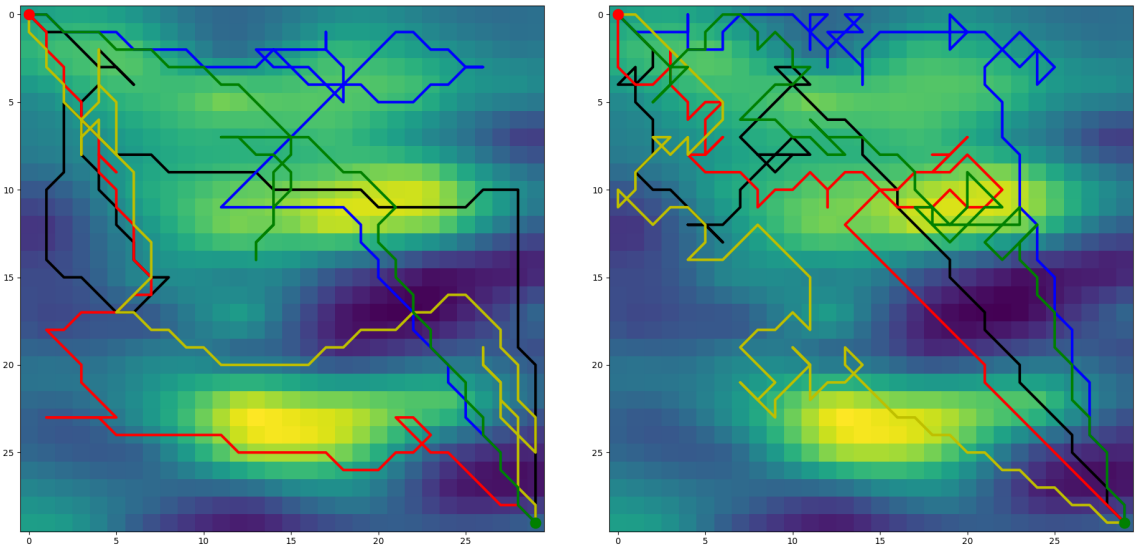
In this chapter, we proposed various online distributed multi-robot sampling solutions based on the MCTS algorithm which are scalable to the size of the team. To minimize revisiting locations, robots share their past experiences (visited sampling locations). Also, each robot estimates the next movement and decision of the others in order to minimize revisiting locations. A GP model of the scalar field being esti-

B	N_{R_i}	method	MSE (std)	$B_{re}(std)$	N_f
100	1	MR-All-MCTS	3.81 (0.94)	8.18 (2.17)	11
100	1	MR-PMCTS	**3.66 (0.79)**	**7.91 (1.09)**	**7**
100	3	MR-All-MCTS	3.52 (0.80)	9.24 (3.41)	9
100	3	MR-PMCTS	**3.21 (0.71)**	**8.22 (2.87)**	**5**
100	5	MR-All-MCTS	2.43 (0.83)	12.96 (3.27)	10
100	5	MR-PMCTS	**2.30 (0.64)**	**10.78 (2.76)**	**6**
200	1	MR-All-MCTS	3.61 (0.82)	**12.16 (3.04)**	8
200	1	MR-PMCTS	**3.27 (0.70)**	13.09 (3.41)	**4**
200	3	MR-All-MCTS	**2.31 (0.68)**	**14.11 (3.81)**	7
200	3	MR-PMCTS	2.38 (0.73)	14.17 (4.20)	**4**
200	5	MR-All-MCTS	2.35 (0.55)	13.01 (4.10)	7
200	5	MR-PMCTS	**2.21 (0.48)**	**11.90 (3.29)**	**3**

Table 5.3: Results for 100 runs for California Central Valley soil moisture dataset.

mated is updated every time a sample location is measured and is used in the process of generation of a new set of random sample locations. Also, our proposed solutions do not require prior knowledge of the environment distribution.

The different learning based methods proposed in this chapter (MR-RMCTS, MR-PMCTS and MR-All-MCTS) were compared against MR-MRS, MR-NCMCTS and MR-MCTS. The algorithms have been compared on a variety of datasets to demonstrate that the results generalize and aren not simply point solutions. Of them, the MR-PMCTS performs more accurate with less failure rate to give the faster solution.

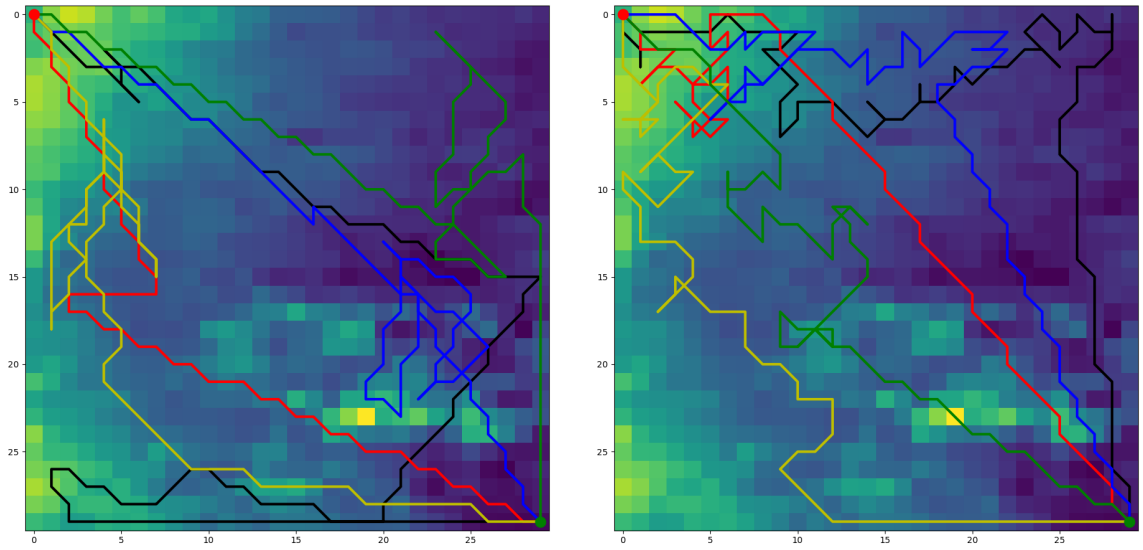


(a) MR-PMCTS (b) MR-All-MCTS

Figure 5.5: (a)-(b) Five-robots sampling paths with budget $B = 100$ in vineyard environment (CAL-SOIL) using MR-PMCTS and MR-All-MCTS.

B	N_{R_i}	method	MSE (std)	$B_{re}(std)$	N_f
100	1	MR-All-MCTS	1.12 (0.37)	5.6 (1.74)	14
100	1	MR-PMCTS	**1.04 (0.29)**	**5.1 (1.08)**	**11**
100	3	MR-All-MCTS	0.93 (0.28)	**7.18 (2.01)**	9
100	3	MR-PMCTS	**0.84 (0.23)**	8.1 (2.17)	**7**
100	5	MR-All-MCTS	0.69 (0.17)	9.80 (3.21)	9
100	5	MR-PMCTS	**0.48 (0.11)**	**8.23 (2.84)**	**4**
200	1	MR-All-MCTS	0.91 (0.23)	**9.19 (2.97)**	9
200	1	MR-PMCTS	**0.74 (0.19)**	10.45 (3.61)	**5**
200	3	MR-All-MCTS	0.42 (0.16)	8.76 (2.64)	8
200	3	MR-PMCTS	**0.37 (0.13)**	**8.31 (2.26)**	**5**
200	5	MR-All-MCTS	**0.28 (0.07)**	**11.04 (3.49)**	**6**
200	5	MR-PMCTS	0.32 (0.09)	11.27 (4.18)	6

Table 5.4: Results for 100 runs for the NASA chlorophyll concentration dataset.



(a) MR-PMCTS (b) MR-All-MCTS

Figure 5.6: (a)-(b) Five-robots sampling paths with budget $B = 100$ in ocean environment using SMCTS and All-MCTS.

Chapter 6

Real World Experiments

In this chapter, a single robot MCTS method is tested on a real robot to evaluate and verify the proposed method's feasibility and performance. The first section discusses the Husky platform, the method, and the test locations. In the second section, we provide the results and discuss the key takeaways.

6.1 Experiment Setup

The purpose of this section is to implement and test one of our proposed method on Husky robot in the field. Implementing simulated code on real robots poses many challenges. Examples include high costs associated with technology integration, software maintenance, and hardware purchases [32].

6.1.1 Platform

We use a Clearpath Husky robotic platform with an average top speed of 3.3 feet per second (1m/s). Husky is a medium-sized platform that can be customized with cameras, manipulators, and more. Detailed technical specifications can be found in the data-sheet [2]. In this experiment, the robot is equipped with a Sick LMS111 LiDAR in the front, a Bosch 9-DOF axis Orientation BNO085 IMU [1], a Ublox GNSS receiver, and an Oak-D 3D camera by Luxonis.

In current satellite navigation systems, real-time kinematic positioning (RTK) is used to correct for common errors [4]. The error can be reduced by comparing the measurements from the signals between multiple satellites. By doing this, the unit is able to calculate their relative position within millimeters [18]. As a first step, we set up the RTK base station. Figure 6.1 shows the Husky robot along with the RTK base station.

The robot is controlled by an onboard NVIDIA Jetson Orin Nano developer kit which is a compact edge AI board, built with Jetson Orin™ Nano 8GB and ROS 2 Humble. The jetson is accessible over remote access SSH or VNC.

We launch the main husky control before running the RMCTS planner after all sensors are launched (IMU and GPS). The ROS2 Navigation Stack (Nav2) provides deployment-grade and high-quality navigation system for mobile robots in complex environments [51]. It provides the capability to follow sequential waypoints. A ROS2 node provides localization using an Extended Kalman Filter based on data from these

sensors, odometry readings, and motion commands. In outdoor environments such as orchards, farms and vineyards, robots movement is limited by the presence of obstacles such as tall grass and weeds. The robot can traverse safely among these grass, but that can be perceived as obstacles by LiDAR sensors, and then force the robot to stop. In [68], the authors presented the novel algorithm based on YOLOv8 deep neural network-based object detection algorithm to detect traversable zones, overcoming the limitation that tall grass acts as an obstacle. In these experiments, we use the GPS navigation plugin which enables localization and Nav2 with the visual-based obstacle avoidance plugin introduced in [68].

Figure 6.1: The Husky robot and the RTK base station.

6.1.2 Method

In this section, we employ the RMCTS method presented in Algorithm 4.3 in Section 4.1.3 in order to test the algorithm on a real robot. Since this is a real-time method and to increase the algorithm's speed, we removed the step of generating new sample points based on the updated GP (line 9 in Algorithm 4.3). These test cases demonstrates how RMCTS works on a real platform, but other proposed methods can be applied and tested as well.

6.1.3 Field Test Locations

In the first experiment, we conduct the experiment on our campus, which serves as a controlled outdoor environment. The second one, which took place in a pistachio orchard, was designed to challenge the robot's capabilities in a more complex setting.

Carol Tomlinson-Keasey Quad, University of California, Merced

In one of the experiments, we test the method on Carol Tomlinson-Keasey Quad at University of California, Merced. Figure 6.2 shows a map of the location obtained from Google maps. In this experiment, 20 sample locations were used, 10 randomly selected and 10 manually selected. Figure 6.3 shows the location of 20 sample points.

Figure 6.2: Aerial view of the Carol Tomlinson-Keasey Quad area where the experiments took place.

Pistachio Orchard

A pistachio orchard in California's San Joaquin valley is used to model the GP for this set of experiments. This orchard consist of 35 rows with 21 trees in each row. Each tree is separated by a distance of about 6 feet 5 inches (2 meters) and each row is separated by a distance of about 9 feet 8 inches (3 meters) for a plot size of roughly 6.1 acres (2.5 hectare). 18 trees are selected as sample trees for measuring stem water potential (SWP). Figure 6.4 shows the top view of the orchard captured from

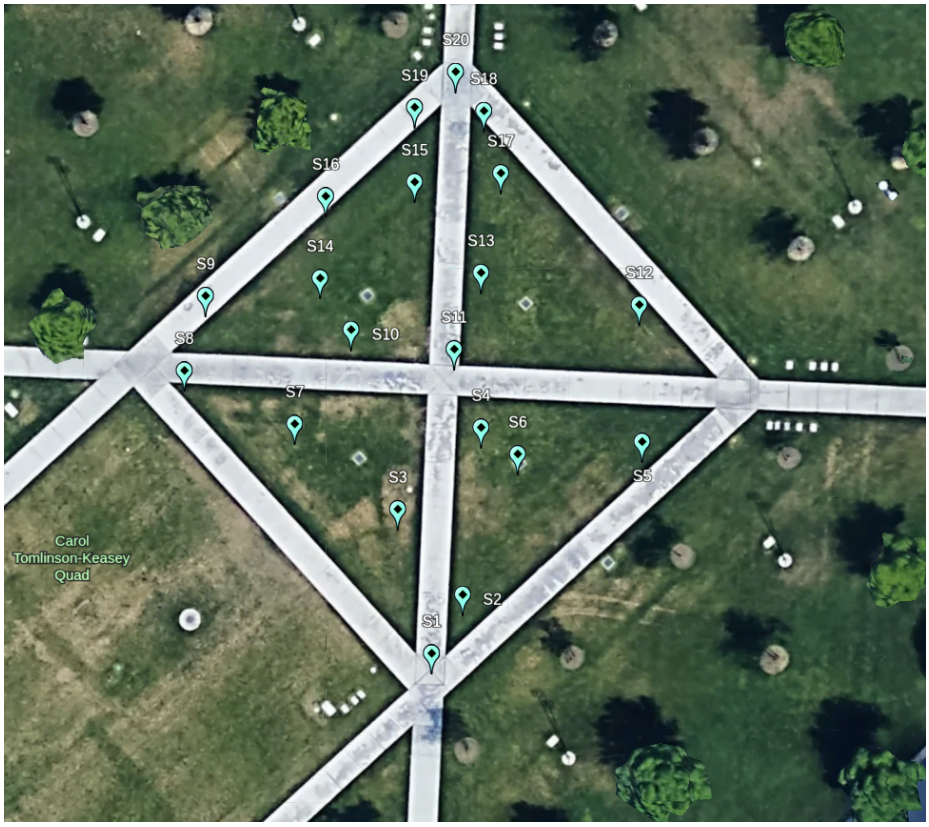


Figure 6.3: 20 sample locations spread across the Carol Tomlinson-Keasey Quad area.

google maps. Figure 6.5 shows the placement of the sensors. We utilize the average of SWP measurements collected in three different data collection in [56] to generate the underlying field. To interpolate and predict the values in the target region, we use the GP regression with the Matern kernel with length-scale equal to 1.5. There are 50 sampling locations in this experiment, 25 randomly chosen and 25 manually selected. Figure 6.6 shows the Husky robot platform gathering leaf temperature data from a pistachio orchard using a multi-spectral camera.

6.2 Results and Discussion

6.2.1 Carol Tomlinson-Keasey Quad Results

Figure 6.7 shows the Husky robot at Carol Tomlinson-Keasey Quad, where these experiments were conducted. Running the RMCTS method on a real robot is challenging due to the difference between Cartesian coordinate system *(x-y)* and geographic coordinate system (GCS) (*lat-long*). In previous chapters, all simulations were presented in Cartesian coordinates. A set of sample points, $\mathcal{V}$ is one of the inputs to the algorithm. In this case, we have 20 sample points represented in GSC. First, we need to convert locations in latitude and longitude to Cartesian coordi-

Figure 6.4: Top view of the pistachio orchard

nates. To accomplish this, we use the *python* package *utm*. Universal Transverse Mercator coordinate system (UTM) is a map projection-based global coordinate system that uses Cartesian coordinates in metric units to provide location information. The experimental dimensions used in this section have been normalized for ease of calculation and representation. s_1 is the starting point for the robot, and s_{20} is the final location. Next, the RMCTS planner is called to get the next sampling location, s_g. With $M = 20$ children in the children set, the robot can move to all other sample locations from each sample location. Before choosing the next action, the MCTS tree adds 1000 nodes in each run.

Once the s_g has been computed, we use *nav2-simple-commander* package, *BasicNavigator*, to navigate the robot to the designated location. The navigator method *followWaypoints(poses)* is used to requests the robot to follow a set of waypoints. Alternatively, we can use the *goToPose(pose)* navigator which drives the robot to the designated point. The *followWaypoints(poses)* navigator receives a list of waypoints, which in our case consists of one location, s_g. *PoseStamped* is the pose ROS2 message with reference coordinate frame and timestamp. In our case we choose the *utm* as coordinate frame.

A synthetic mixture of GPs was used throughout the experiment to model the scalar field. Figure 6.8 shows the posterior mean of the GP used to model the sensory reading. In the beginning, the robot has no knowledge of the prior, that is, it does not have access to the GP mean and standard deviation. As the robot moves through

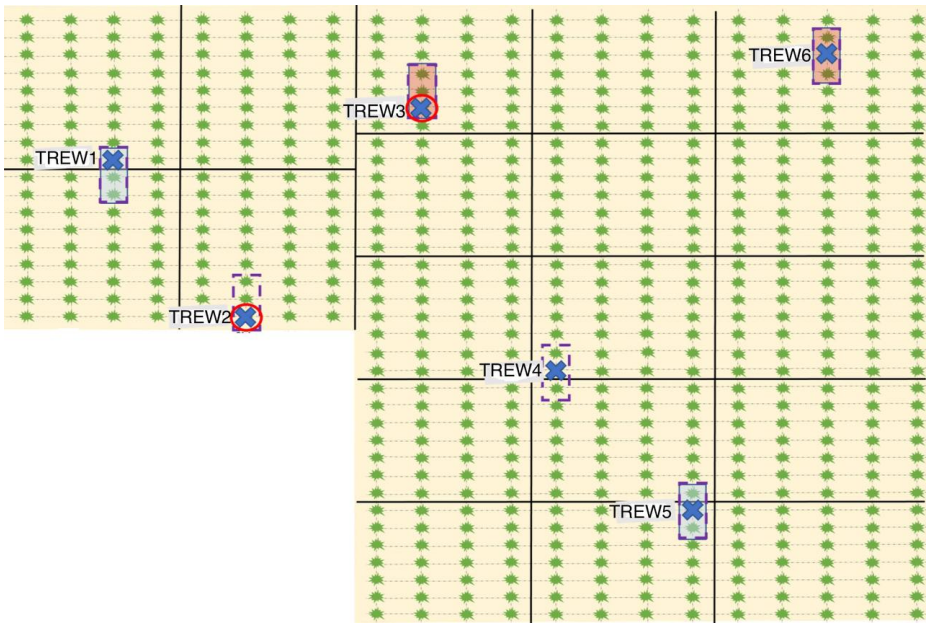


Figure 6.5: Top view of the pistachio orchard with sensor placement

the environment, it reads modeled sensory readings at each sample point and its GP model is updated based on new reading. Since there are no obstacles along the way, the travel cost of the robot is approximated by Equation 4.3. As the robot can move freely between each sample point, Euclidean distance is used here.

Figure 6.9 (a) and (b) show the reconstructed spatial field with visited sample locations with budget 5 and 7. Figure 6.10 show the reconstructed spatial field and visited sample points with B=10 and B=12. Increasing the budget allows the robot to explore the environment more and take more samples, resulting in a more accurate and similar reconstruction to the ground truth shown in Figure 6.8. Figure 6.11 shows the Husky robot at one of the sample locations in the field.

6.2.2 Pistachio Orchard Results

Figure 6.12 shows the reconstructed underlying field and 50 sample location. In this experiment, robot starts from location $(A1, R1)$, and $(B13, L34)$ is the final location. The mean and standard deviation of the GP are unknown to the robot at the beginning of the experiment. As the robot moves through the environment, it reads the modeled sensory readings at each sample point and its GP is updated based on new reading.

Since all sample locations are represented in Cartesian coordinates, we do not need to change the coordination system. With $M = 20$ children in the children set, the robot can move to all other sample locations from each sample location. Before choosing the next action, the MCTS tree adds 1000 nodes in each run. Despite the fact that the robot requires path-planning algorithm to find the obstacle-free path to the designated point, the travel cost of the robot is approximated by Equation 4.3

Figure 6.6: Robot moving autonomously at pistachio orchard.

with Euclidean distance. Rather than using a real robot, these experiments were conducted in simulation.

Figure 6.13, Figure 6.14 and Figure 6.15 show the reconstructed spatial field and visited sample locations with budget 100, 300, and 500. Increasing the budget allows the robot to perform a deeper exploration and collect more samples, resulting in a more accurate and similar reconstruction to the ground truth shown in Figure 6.12.

6.3 Conclusions

The purpose of this chapter was to implement RMCTS proposed method on the Husky robot in the field. In this chapter, we discussed the challenges we faced when implementing simulated code on a real robot. In our experiments, we demonstrated that the proposed method is feasible in real-life situations.

Figure 6.7: Husky robot at Carol Tomlinson-Keasey Quad.

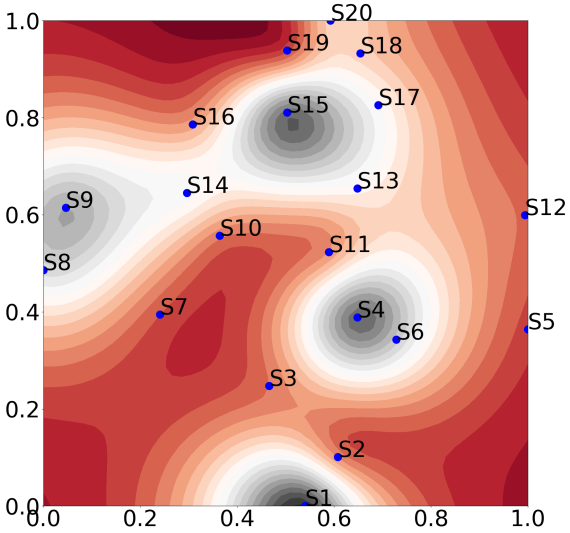
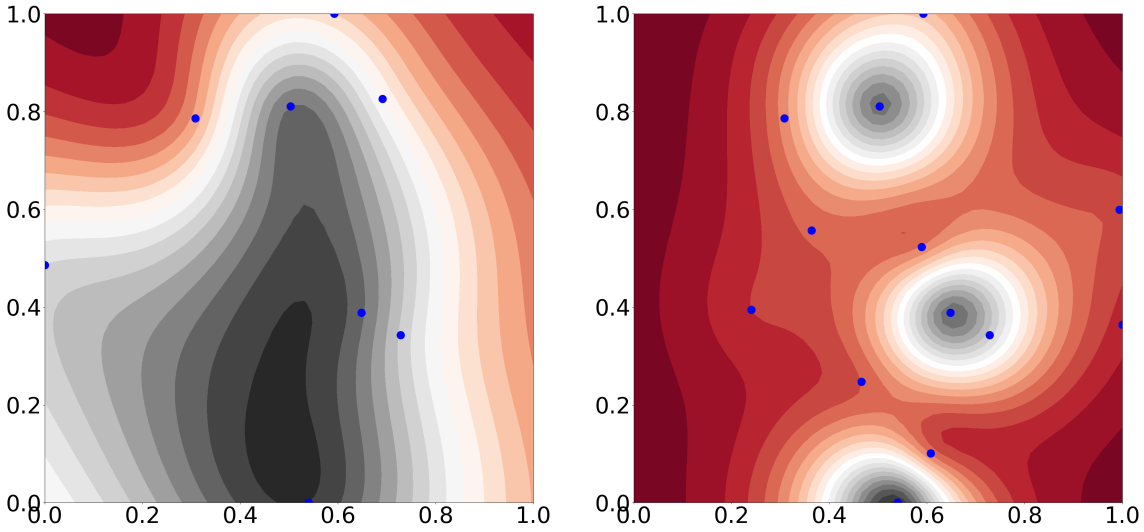
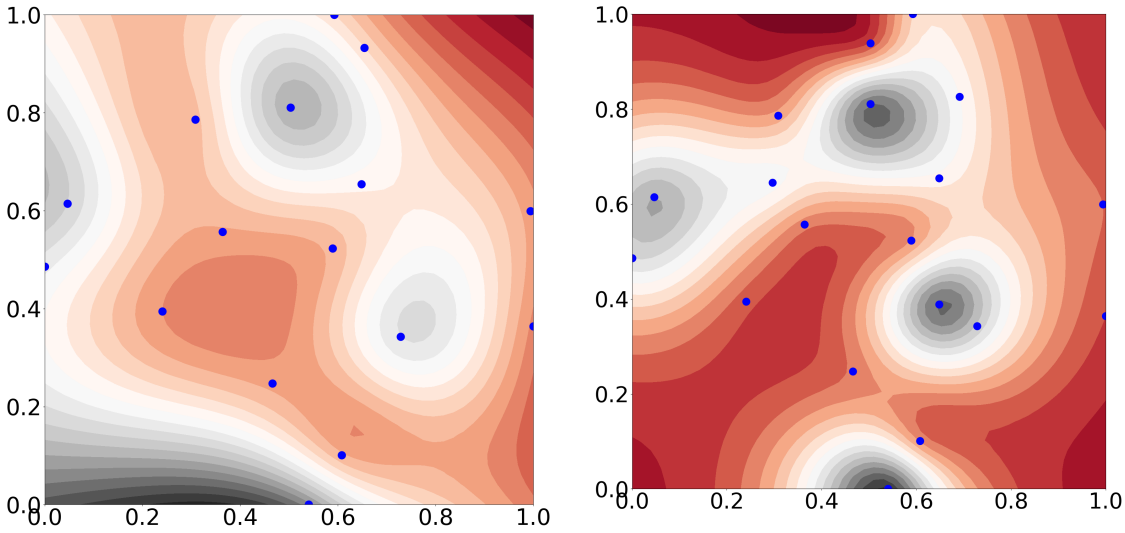


Figure 6.8: Posterior mean used as a sensory reading.



(a) B=5 (b) B=7

Figure 6.9: Reconstructed spatial field and visited sample points with B=5 and B=7.



(a) B=10 (b) B=12

Figure 6.10: Reconstructed spatial field and visited sample points with B=10 and B=12.

Figure 6.11: The Husky robot at one of the sample locations in the field.

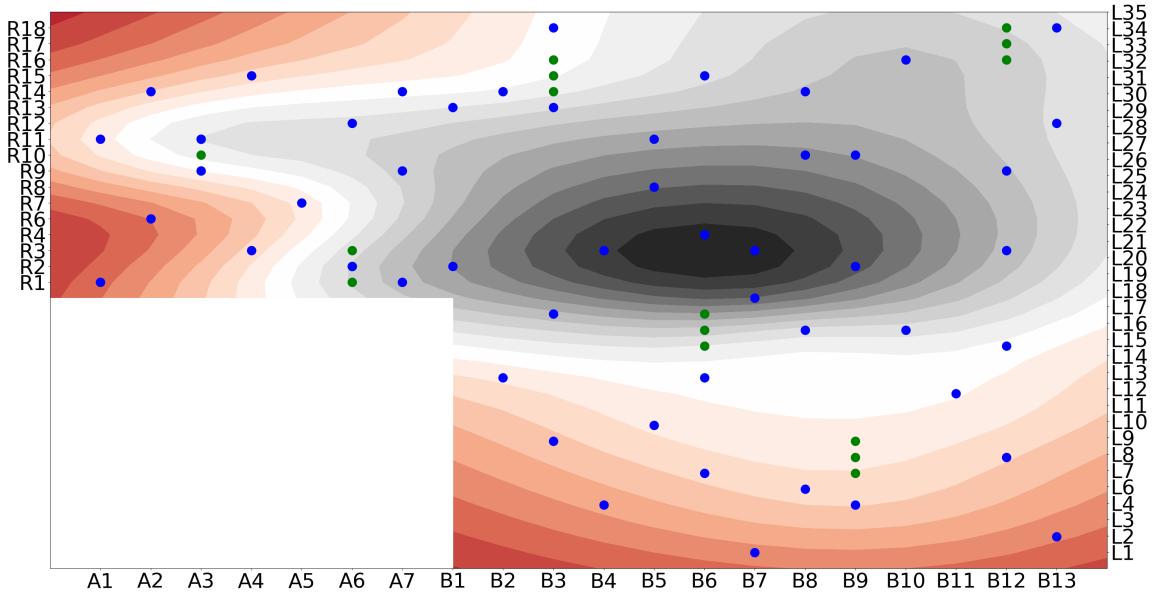


Figure 6.12: Reconstructed spatial field and all 50 sample points.

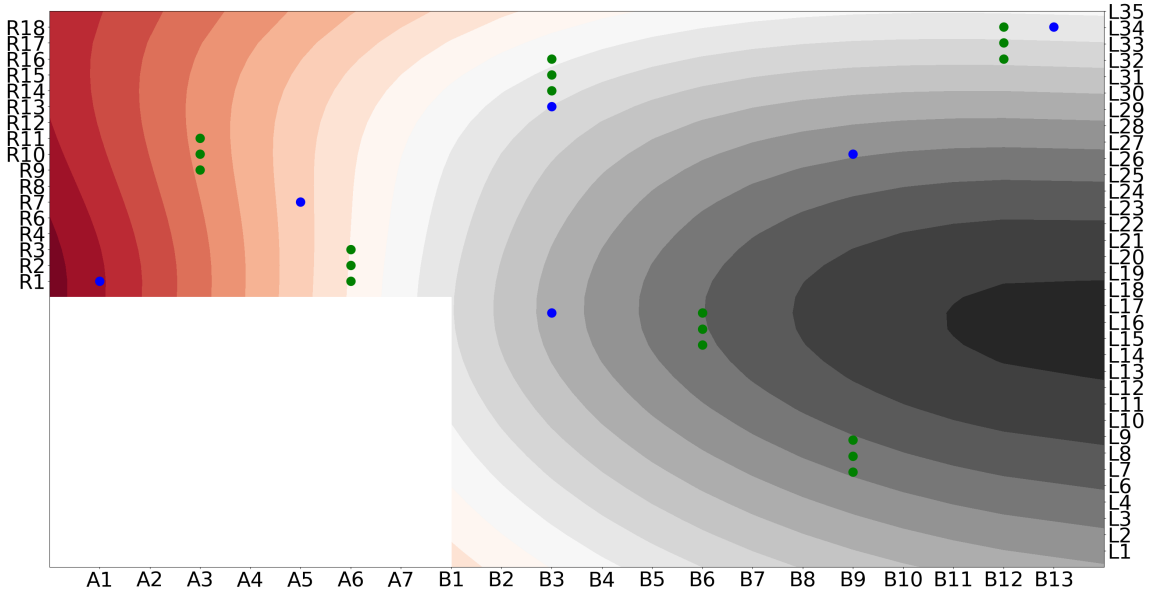


Figure 6.13: Reconstructed spatial field and visited sample points with B=100.

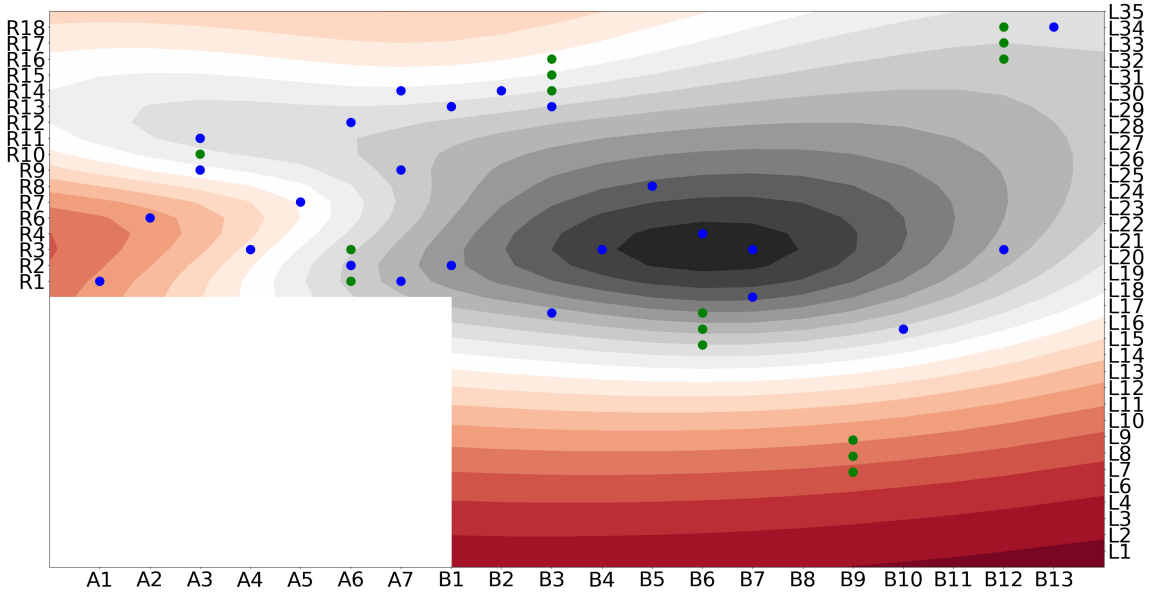


Figure 6.14: Reconstructed spatial field and visited sample points with B=300.

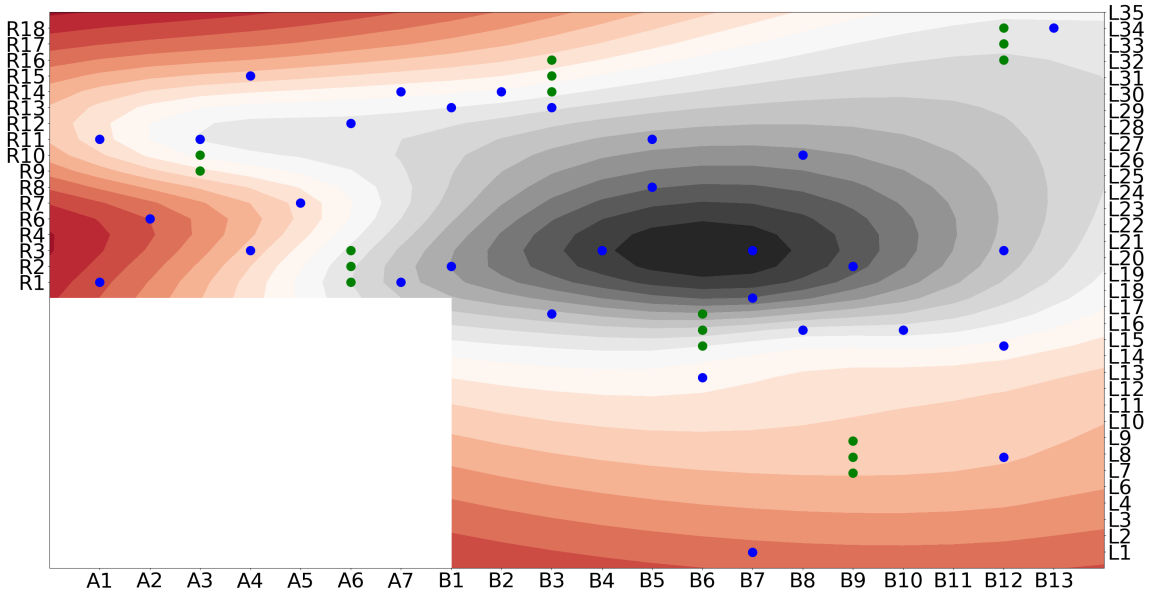


Figure 6.15: Reconstructed spatial field and visited sample points with B=500.

Chapter 7

Final Thoughts

7.1 Conclusions

Throughout this dissertation, the single robot and multi robot informative path planning (IPP) are discussed. A multi-robot IPP method aims to select paths and sequence of sampling locations for each robot that maximize the quality of the reconstructed scalar field while do not exceed the travel budget. In Chapter 1, the motivation of the study is given, which involves using robots to learn a map of the environment, specifically in agriculture for adjusting water use. Managing a robot within an agricultural field poses a number of challenges, and these challenges are highly specific to the environment, which was framed as a stochastic environment with obstacles. An overview of each of the problems discussed throughout the thesis are given in Chapter 2.

Chapter 3 discusses the Q-Learning based solution. The offline proposed method involves two algorithms; the first algorithm determines the next sample location based on the consumed energy, remaining budget and mutual information criteria as a reward, and then a second loop finds the best obstacle free path using the RRT algorithm to visit the designated sample location.

Chapter 4 proposes an online sampling algorithm based on the MCTS algorithm. It is assumed that the current location of the robot is the root node of the MCTS tree. MCTS expands the tree by adding some nodes to the tree. After the tree has been built, the next sampling location is selected based on the UCT formula. Every time a sample location is measured, the GP model of the scalar field is updated and used to generate a new set of random sample locations. The proposed approach is online and it does not require prior knowledge of the environment distribution.

In Chapter 5, the IPP problem is extended to multi-agent scenario focusing on avoiding collisions, revisiting the same locations while managing budgets. Using the MCTS algorithm, we propose different versions of online distributed multi-robot sampling methods that scales with the number of robots in the team. The robots share their past experiences (visited sampling locations), and each robot estimates the next movement and decision of other robots in order to minimize revisiting locations. Each time a sample location is measured, the GP model of the scalar field is updated. We propose methods that are more accurate, fails less frequently, and has a lower remaining budget than baseline methods. Also, our proposed methods do not require prior knowledge of the environment distribution.

Chapter 6 evaluates and verifies a single robot MCTS method on a real robot to verify the proposed method feasibility and performance.

7.2 Future Work

In the context of environmental map learning, task planning algorithms for robots require further development. The algorithms presented in this dissertation are only useful with some assumptions that may not be true in practice. This assumption should be challenged in the future so that the algorithms become more practical and achieve better results than they have thus far.

7.2.1 Improving Cost Model

The proposed models are often built on plots of land that are not flat, resulting in non-uniform costs between two locations and possibly different costs depending on the direction of travel. It is also possible that the route between two locations is not straight. Depending on the robot's age, payload, and battery level, the travel cost can also vary. The travel cost needs to be computed using a new method that takes into account all of these assumptions.

7.2.2 Further Experiments in the Field with Multiple Robots

A real robot was used in this thesis to evaluate and verify the feasibility and performance of the MCTS method. Different scenarios are examined on the Husky platform, as well as two different locations to verify the algorithm's performance. The multi-robot MCTS should be tested in real-world situations with a team of robots. The communication between robots is a crucial aspect that must be considered.

7.2.3 Non-homogeneous Agents

Another assumption made throughout this thesis by the algorithms solving the IPP problem is that all of the agents are exactly the same. That is, they all have the same sensors and moving constraint. It is possible that the agents cooperating together are heterogeneous; there may be a variety of robots that move at different speeds and have different sensors. Due to this, rewards and costs would differ depending on the agent, requiring agents to coordinate and plan in accordance with their abilities. In order for non-homogeneous agents to work together, some changes must be made to the algorithms.

7.2.4 Approximation Methods

In many practical cases, the state space (number of sample locations in fields) and the action space (in our case number of sample locations in fields) are enormous. In

that case, finding the optimal policy is not possible, so the goal instead would be to find a proper approximate solution using limited computational resources.

Deep feedforward networks are one of the most widely used deep learning models. The goal of a feedforward network is to approximate some function. In this case, with parametric approximation of the action-value function $q(s, a, w)$ where w is a finite-dimensional weight vector and the network learns the value of the parameters w that result in the best function approximation. One possible future work would be developing a method to approximate the action-value function, $q(s, a, w)$, that is represented as a parameterized functional form with weight vector w.

Bibliography

[1] Bno08x imu data sheet. Available at https://www.ceva-ip.com/wp-content/uploads/2019/10/BNO080_085-Datasheet.pdf (2024/07/15).

[2] HUSKY, unmanned ground vehicle. Available at https://clearpathrobotics.com/husky-unmanned-ground-vehicle-robot/ (2024/07/15).

[3] NASA Chlorophyll concentration data-set. Available at https://neo.gsfc.nasa.gov/view.php?datasetId=MY1DMM_CHLORA (2024/07/15).

[4] Real-time kinematic positioning (rtk). Available at https://en.wikipedia.org/wiki/Real-time_kinematic_positioning (2024/07/15).

[5] A. S. Aguiar, F. N. Dos Santos, J. B. Cunha, H. Sobreira, and A. J. Sousa. Localization and mapping for robots in agriculture and forestry: A survey. *Robotics*, 9(4):97, 2020.

[6] E. Altman. *Constrained Markov decision processes*. Routledge, 2021.

[7] Sh. Bai, J. Wang, F. Chen, and B. Englot. Information-theoretic exploration with bayesian optimization. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1816–1822. IEEE, 2016.

[8] D. Bertsekas. *Rollout, policy iteration, and distributed reinforcement learning*. Athena Scientific, 2021.

[9] G. Best, O. M Cliff, T. Patten, R. R. Mettu, and R. Fitch. Dec-mcts: Decentralized planning for multi-robot active perception. *The International Journal of Robotics Research*, 38(2-3):316–337, 2019.

[10] J. Binney, A. Krause, and G. S. Sukhatme. Optimizing waypoints for monitoring spatiotemporal phenomena. *The International Journal of Robotics Research*, 32(8):873–888, 2013.

[11] A. Blanchard and Th. Sapsis. Informative path planning for anomaly detection in environment exploration and monitoring. *Ocean Engineering*, 243:110242, 2022.

[12] D. Bochtis, L. Benos, M. Lampridi, V. Marinoudi, S. Pearson, and C.G. Sørensen. Agricultural workforce crisis in light of the covid-19 pandemic. *Sustainability*, 12(19), 2020.

[13] L. Booth and S. Carpin. Distributed estimation of scalar fields with implicit coordination. In J. Bourgeois et al., editor, *Distributed Autonomous Robotic Systems 16.*, pages 466–478. Springer, 2024.

[14] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, Ph. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

[15] J. A. Caley and G. A. Hollinger. Data-driven comparison of spatio-temporal monitoring techniques. In *OCEANS 2015-MTS/IEEE Washington*, pages 1–7. IEEE, 2015.

[16] M. Campbell, A. Dechemi, and K. Karydis. An integrated actuation-perception framework for robotic leaf retrieval: Detection, localization, and cutting. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 9210–9216. IEEE, 2022.

[17] S. Carpin and T. C. Thayer. Solving stochastic orienteering problems with chance constraints using monte carlo tree search. In *2022 IEEE 18th International Conference on Automation Science and Engineering (CASE)*, pages 1170–1177. IEEE, 2022.

[18] S. Cerrato. *Gps-based autonomous navigation of unmanned ground vehicles in precision agriculture applications*. PhD thesis, Politecnico di Torino, 2020.

[19] W. Chen and L. Liu. Pareto monte carlo tree search for multi-objective informative planning. *arXiv preprint arXiv:2111.01825*, 2021.

[20] R. Cui, Y. Li, and W. Yan. Mutual information-based multi-auv path planning for scalar field sampling using multidimensional rrt. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 46(7):993–1004, 2015.

[21] J. Das, J. Harvey, F. Py, H. Vathsangam, R. Graham, K. Rajan, and G. S. Sukhatme. Hierarchical probabilistic regression for auv-based adaptive sampling of marine phenomena. In *2013 IEEE International Conference on Robotics and Automation*, pages 5571–5578. IEEE, 2013.

[22] A. Dechemi, D. Chatziparaschis, J. Chen, M. Campbell, A. Shamshirgaran, C. Mucchiani, A. Roy-Chowdhury, S. Carpin, and K. Karydis. Robotic assessment of a crop's need for watering. *IEEE Robotics and Automation Magazine*, 30(4):52 – 67, 2023.

[23] G. A. Di Caro and A. W. Ziaullah Yousaf. Map learning via adaptive region-based sampling in multi-robot systems. In *Distributed Autonomous Robotic Systems: 15th International Symposium*, pages 335–348. Springer, 2022.

[24] A. Dutta, Patrick K., and J. M. O'Kane. Opportunistic multi-robot environmental sampling via decentralized markov decision processes. In *Distributed Autonomous Robotic Systems: 15th International Symposium*, pages 163–175. Springer, 2022.

[25] H.J.S. Finch, A.M. Samuel, and G.P.F. Lane. Precision farming. In *Lockhart & Wiseman's Crop Husbandry Including Grassland*, pages 235 – 244. Woodhead Publishing, ninth edition edition, 2014.

[26] K. Fowler. Five reasons labor shortages are impacting supply chains. `https://www.forbes.com`, 2021. [Online; accessed 1-Jan.-2023].

[27] J. N. Fuhg, A. Fau, and U. Nackenhorst. State-of-the-art and comparative review of adaptive sampling methods for kriging. *Archives of Computational Methods in Engineering*, 28:2689–2747, 2021.

[28] A. Fulton, J. Grant, R. Buchner, and J. Connell. Using the pressure chamber for irrigation management in walnut, almond and prune. *ANR*, 2014.

[29] Y. Ge, F. Zhu, X. Ling, and Q. Liu. Safe q-learning method based on constrained markov decision processes. *IEEE Access*, 7:165007–165017, 2019.

[30] D. V. Gealy, S. McKinley, M. Gou, L. Miller, S. Vougioukas, J. Viers, S. Carpin, and K. Goldberg. Co-robotic device for automated tuning of emitters to enable precision irrigation. In *Proceedings of the IEEE Conference on Automation Science and Engineering*, pages 922–927, 2016.

[31] M. Gh. Jadidi, J. V Miro, and G. Dissanayake. Sampling-based incremental information gathering with applications to robotic exploration and environmental monitoring. *The International Journal of Robotics Research*, 38(6):658–685, 2019.

[32] G. Gil, D. Emilio Casagrande, L. P. Cortés, and R. Verschae. Why the low adoption of robotics in the farms? challenges for the establishment of commercial agricultural robots. *Smart Agricultural Technology*, 3:100069, 2023.

[33] C. Guestrin, A. Krause, and A. P. Singh. Near-optimal sensor placements in gaussian processes. In *Proceedings of the 22nd international conference on Machine learning*, pages 265–272, 2005.

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

[35] D. Jang, J. Yoo, C. Y. Son, and H. J. Kim. Fully distributed informative planning for environmental learning with multi-robot systems. *arXiv preprint arXiv:2112.14433*, 2021.

[36] J.R. Jankowski. *Consumptive Water Use in California's Sacramento-San Joaquin Delta: A Comparison of Estimation Methods and Field Data, with Implications for Water Right Diversion Reporting.* University of California, Davis, 2018.

[37] G. Kalweit, M. Huegle, M. Werling, and J. Boedecker. Deep constrained q-learning. *arXiv e-prints*, pages arXiv–2003, 2020.

[38] X. Kan, H. Teng, and K. Karydis. Multi-robot field exploration in hex-decomposed environments for dubins vehicles. In *Proceedings of the IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 449–455, 2019.

[39] X. Kan, H. Teng, and K. Karydis. Online exploration and coverage planning in unknown obstacle-cluttered environments. *IEEE Robotics and Automation Letters*, 5(4):5969–5976, 2020.

[40] X. Kan, T. Thayer, S. Carpin, and K. Karydis. Task planning on stochastic aisle graphs for precision agriculture. *IEEE Robotics and Automation Letters*, 6(2):3287 – 3294, 2021.

[41] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 30(7):846–894, 2011.

[42] S. Kemna, S. Kangaslahti, O. Kroemer, and G. S Sukhatme. Adaptive sampling: Algorithmic vs. human waypoint selection. *arXiv preprint arXiv:2104.11962*, 2021.

[43] S. Kemna, J. G Rogers, C. Nieto-Granda, S. Young, and G. 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.

[44] B. Khamidehi and E. S. Sousa. A double q-learning approach for navigation of aerial vehicles with connectivity constraint. In *ICC 2020-2020 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2020.

[45] L. Kocsis and C. Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer, 2006.

[46] G. P. Kontoudis and D. J. Stilwell. Fully decentralized, scalable gaussian processes for multi-agent federated learning. *arXiv preprint arXiv:2203.02865*, 2022.

[47] A. Krause and C. Guestrin. Submodularity and its applications in optimized information gathering. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(4):1–20, 2011.

[48] A. Krause, C. Guestrin, A. Gupta, and J. Kleinberg. Near-optimal sensor placements: Maximizing information while minimizing communication cost. In *Proceedings of the 5th international conference on Information processing in sensor networks*, pages 2–10, 2006.

[49] A. Krause, A. Singh, and C. Guestrin. Near-optimal sensor placements in gaussian processes: Theory, efficient algorithms and empirical studies. *Journal of Machine Learning Research*, 9(2), 2008.

[50] S. M LaValle and J. Kuffner Jr. Randomized kinodynamic planning. *The international journal of robotics research*, 20(5):378–400, 2001.

[51] S. Macenski, F. Martín, R. White, and J. G. Clavero. The marathon 2: A navigation system. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2718–2725. IEEE, 2020.

[52] S. Manjanna, M. A. Hsieh, and G. Dudek. Scalable multirobot planning for informed spatial sampling. *Autonomous Robots*, 46(7):817–829, 2022.

[53] S. Manjanna, H. van Hoof, and G. Dudek. Reinforcement learning with non-uniform state representations for adaptive search. In *2018 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pages 1–7. IEEE, 2018.

[54] A. Marino, G. Antonelli, A. P. Aguiar, A. Pascoal, and S. Chiaverini. A decentralized strategy for multirobot sampling/patrolling: Theory and experiments. *IEEE Transactions on Control Systems Technology*, 23(1):313–322, 2014.

[55] M. Meron, D. Grimes, C. Phene, and K. Davis. Pressure chamber procedures for leaf water potential measurements of cotton. *Irrigation science*, 8:215–222, 1987.

[56] M. Mortazavi, R. Ehsani, S. Carpin, and A. Toudeshki. Predicting tree water status in pistachio and almond orchards using supervised machine learning. *Available at SSRN 4511076*, 2024.

[57] MA. Oliver and R. Webster. A tutorial guide to geostatistics: Computing and modelling variograms and kriging. *Catena*, 113:56–69, 2014.

[58] J. Orr and A. Dutta. Multi-agent deep reinforcement learning for multi-robot applications: A survey. *Sensors*, 23(7):3625, 2023.

[59] A. Owen-Hill. Top 10 robotic applications in the agricultural industry. `https://www.blog.robotiq.com`, 2022. [Online; accessed 1-Jan.-2023].

[60] L. Pan, S. Manjanna, and M. A. Hsieh. Marlas: Multi agent reinforcement learning for cooperated adaptive sampling. *arXiv preprint arXiv:2207.07751*, 2022.

[61] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[62] M. Popović, G. Hitz, J. Nieto, I. Sa, R. Siegwart, and E. Galceran. Online informative path planning for active classification using uavs. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 5753–5758. IEEE, 2017.

[63] M. Popović, T. Vidal-Calleja, G. Hitz, J. J Chung, I. Sa, R. Siegwart, and J. Nieto. An informative path planning framework for uav-based terrain monitoring. *Autonomous Robots*, 44(6):889–911, 2020.

[64] C. E. Rasmussen. Gaussian processes in machine learning. In *Summer school on machine learning*, pages 63–71. Springer, 2003.

[65] C. M. Rodriguez-Dominguez, A. Forner, S. Martorell, et al. Leaf water potential measurements using the pressure chamber: Synthetic testing of assumptions towards best practices for precision and accuracy. *Plant, Cell & Environment*, 45(7):2037–2061, 2022.

[66] R. Romero, JL. Muriel, I. García, and D. Muñoz de la Peña. Research on automatic irrigation control: State of the art and recent results. *Agricultural water management*, 114:59–66, 2012.

[67] J. Rückin, L. Jin, and M. Popović. Adaptive informative path planning using deep reinforcement learning for uav-based active sensing. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 4473–4479. IEEE, 2022.

[68] E. Sani, A. Sgorbissa, and S. Carpin. Improving the ros 2 navigation stack with real-time local costmap updates for agricultural applications. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2024 (to appear).

[69] L. C. Santos, F. N. Santos, EJ. S. Pires, A. Valente, P. Costa, and S. Magalhães. Path planning for ground robots in agriculture: A short review. In *2020 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 61–66. IEEE, 2020.

[70] G. D. Schaible and M. P. Aillery. Water conservation in irrigated agriculture: Trends and challenges in the face of emerging demands. Technical Report EIB-99, U.S. Department of Agriculture, Economic Research Service, September 2012.

[71] P. F. Scholander, H. T. Hammel, E. A. Hemmingsen, and E. D. Bradstreet. Hydrostatic pressure and osmotic potential in leaves of mangroves and some other plants. *Proceedings of the National Academy of Sciences of the United States of America*, 52(1):119, 1964.

[72] E. Schulz, M. Speekenbrink, and A. Krause. A tutorial on gaussian process regression: Modelling, exploring, and exploiting functions. *Journal of Mathematical Psychology*, 85:1–16, 2018.

[73] B. Sean, L. Bartolomei, F. Kennel-Maushart, and M. Chli. Decentralised multi-robot exploration using monte carlo tree search. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robotis and Systems*, pages 7354–7361, 2023.

[74] A. Shamshirgaran and S. Carpin. Reconstructing a spatial field with an autonomous robot under a budget constraint. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 8963–8970, 2022.

[75] A. Shamshirgaran and S. Carpin. Environmental map learning with multi-robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2025 (to be Submitted).

[76] A. Shamshirgaran, S. Manjanna, and S. Carpin. Distributed multi-robot online sampling with budget constraints. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 12658–12664, 2024.

[77] M. Shekaramiz, T. K. Moon, and J. H. Gunther. A note on kriging and gaussian processes. 2019.

[78] R. Shrivastava. Seso wants to help solve a 3.1 billion farm labor crisis through automation. `https://www.forbes.com`, 2021. [Online; accessed 1-Jan.-2023].

[79] A. Singh, A. Krause, C. Guestrin, W. Kaiser, and M. Batalin. Efficient planning of informative paths for multiple robots. In *Proceedings of the 20th international joint conference on Artifical intelligence*, pages 2204–2211, 2007.

[80] A. Sipahioglu, G. Kirlik, O. Parlaktuna, and A. Yazici. Energy constrained multi-robot sensor-based coverage path planning using capacitated arc routing approach. *Robotics and Autonomous Systems*, 58(5):529–538, 2010.

[81] M. L. Stein. *Interpolation of Spatial Data – Some Theory for Kriging*. Springer, 1999.

[82] M. L. Stein. *Interpolation of spatial data: some theory for kriging*. Springer Science & Business Media, 1999.

[83] J. S. P. Sundaram, W. Du, and Zh. Zhiwei. A survey on lora networking: Research problems, current solutions, and open issues. *IEEE Communications Surveys & Tutorials*, 22(1):371–388, 2019.

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

[85] V. Suryan and P. Tokekar. Efficiently identifying hotspots in a spatially varying field with multiple robots. *arXiv preprint arXiv:2309.07981*, 2023.

[86] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[87] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

[88] 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. IEEE, 2018.

[89] T. Thayer and S. Carpin. An adaptive method for the stochastic orienteering problem. *IEEE Robotics and Automation Letters*, 6(2):4185–4192, 2021.

[90] T. Thayer and S. Carpin. A fast algorithm for stochastic orienteering with chance constraints. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2021 (to appear).

[91] T. Thayer and S. Carpin. A resolution adaptive algorithm for the stochastic orienteering problem with chance constraints. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2021 (to appear).

[92] T. Thayer, S. Vougioukas, K. Goldberg, and S. Carpin. Routing algorithms for robot assisted precision irrigation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2221–2228, 2018.

[93] T. Thayer, S. Vougioukas, K. Goldberg, and S. Carpin. Multi-robot routing algorithms for robots operating in vineyards. *IEEE Transactions on Automation Science and Engineering*, 17(3):1184–1194, 2020.

[94] T. C Thayer, S. Vougioukas, K. Goldberg, and S. Carpin. Multi-robot routing algorithms for robots operating in vineyards. In *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*, pages 14–21. IEEE, 2018.

[95] P. Tokekar, V. J. Hook, D. Mulla, and V. Isler. Sensor planning for a symbiotic uav and ugv system for precision agriculture. *IEEE transactions on robotics*, 32(6):1498–1511, 2016.

[96] T. Tsiligiridis. Heuristic methods applied to orienteering. *Journal of the Operational Research Society*, 35:797–809, 09 1984.

[97] G. Vellidis, V. Liakos, W. Porter, M. Tucker, and X. Liang. A dynamic variable rate irrigation control system. In *Proceedings of the International Conference on Precision Agriculture*, volume 13, pages 1–9, 2016.

[98] H Vila, I Hugalde, and M Di Filippo. Estimation of leaf water potential by thermographic and spectral measurements in grapevine. *RIA: Revista de Investigaciones Agropecuarias*, 37:46–52, 2011.

[99] S. Vougioukas. Agricultural robotics. *Annual review of control, robotics, and autonomous systems*, 2:339–364, 2019.

[100] J.S. Wallace and P.J. Gregory. Water resources and their use in food production systems. *Aquatic Sciences*, 64:363–375, 2002.

[101] Y. Wei and R. Zheng. Informative path planning for mobile sensing with reinforcement learning. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, pages 864–873. IEEE, 2020.

[102] Ch. KI Williams and C. E. Rasmussen. *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA, 2006.

[103] Z. Yan, N. Jouandeau, and A. A. Cherif. A survey and analysis of multi-robot coordination. *International Journal of Advanced Robotic Systems*, 10(12):399, 2013.

[104] Q. Zaman. *Precision Agriculture: Evolution, Insights and Emerging Trends*. Elsevier, 2023.

[105] T. Zhao, B. Stark, Y. Chen, A.L. Ray, and D. Doll. Challenges in water stress quantification using small unmanned aerial system (suas): Lessons from a growing season of almond. *Journal of Intelligent & Robotic Systems*, 88(2):721–735, 2017.