AN ABSTRACT OF THE DISSERTATION OF

Dylan A. Jones for the degree of Doctor of Philosophy in Robotics presented on August 25, 2020.

Title: Realizable Path Planning and Execution for Robotic Systems

Abstract approved: ____

Geoffrey A. Hollinger

Performing autonomous robotic tasks in the field, such as ocean monitoring and aerial surveillance, requires planning and executing paths in dynamic environments. In these uncertain and changing environments, it is not uncommon to see a large difference between the path planned by the robotic vehicle and the path that the robotic vehicle realizes while executing that path. This difference can decrease the performance of the robotic system by introducing additional risk or forcing the vehicle to miss important survey areas. Existing systems do not consider large-scale disturbances, do not consider the differences between the planning and control models, and do not incorporate new information about disturbances found online during planning and execution. To address these shortcomings, this thesis provides algorithms that incorporate disturbances directly into planning, reason about the robot's low-level controller, and utilize information gathered during execution about both disturbances and the robot's dynamics. The impact of these improvements is a reduction in risk and improvement of the quality of robotic information collection.

This thesis provides three contributions to help reduce this difference between planned and executed trajectories. First, we introduce a stochastic optimization framework which utilizes an action-space path representation to remove the need for expensive reachability calculations. This action-space formulation allows for a more natural representation of the effects of disturbances on vehicles with low actuation, and the stochastic optimization technique allows the mapping of a state-space based reward function to the action-space while being efficient enough to be used in a sequential allocation framework for planning for multiple vehicles. We demonstrate the computational efficiency of this algorithm against other state-of-the-art planners in a simulated ocean environment of the Gulf of Mexico.

Second, we present a novel algorithm, Energy-Efficient Stochastic Trajectory Optimization (EESTO), which allows vehicles with moderate levels of actuation to plan energy-efficient trajectories thorough strong and uncertain disturbances. In addition to this algorithm, we introduce a framework which can utilize the efficiency of EESTO to account for information gathered online about the disturbances that the vehicle is moving through. We demonstrate the capabilities of the algorithm and framework in both a simulated ocean environment off the coast of California near the Channel Island as well as on hardware on a lake near Eugene, Oregon.

Lastly, we present a framework for increasing the realizability of planned paths for high-actuation vehicles, which allows the robotic system to reason about the capabilities of the on-board low-level controller. By incorporating the capabilities of the low-level controller into execution and planning, this framework is able to increase the realizability of the planned information gathering path. We demonstrate the capabilities of this framework through extensive simulation trials and on hardware on a lake near Corvallis, Oregon.

Realizable Path Planning and Execution for Robotic Systems

by
Dylan A. Jones

A DISSERTATION

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Presented August 25, 2020
Commencement June 2021

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my advisor Geoffrey Hollinger for all the help, support, and guidance he has given me over the time of my dissertation. I would also like to thank the members of my committee: Drs. Bill Smart, Kagan Tumer, and Ross Hatton for their feedback throughout my time at Oregon State.

Next, I have to thank the many members of the Robotics Decision Making Laboratory who have helped me in my time here. Specifically, thanks are especially needed for Seth McCammon and Bob DeBortoli, without whom there would be many more spelling errors throughout my work.

No dissertation is made without a village, and as such I would like to thank all the members, past and present, of the Oregon State Robotics Program for their support and insightful conversations.

I would like to acknowledge the many different agencies and organizations who have funded and assisted with this research: the Office of Naval Research (N00014-14-1-0509, N00014-17-1-2581), Department of Energy (DE-EE-0006816), USDA/NIFA (2014-67021-22174), the Naval Research Lab, and collaborators from Washington State University and the Naval Research Lab.

Lastly, I would like to thank all the friends and family not covered previously for helping keep me sane through these past five years. Special thanks to my parents, Cliff and Karen Jones, and my sister Lily are also in order.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF TABLES

LIST OF ALGORITHMS

Chapter 1: Introduction

There is currently a sizable gap between the performance of robots seen in factory [66], warehouse [113], and research [87] environments and the performance seen in practice in uncontrolled environments, such as marine [40] or aerial [2] domains. On factory floors and in warehouse workspaces, the environment is engineered to remove disturbances (e.g. by ensuring that relevant objects are always in the same place or installing fiducials on the floor to ensure localization). Research labs are also typically well controlled and instrumented environments where the criterion for robustness of execution is lower than that required for every day applications [90]. In contrast, unstructured field robotics environments do not allow for engineering the environments to simplify the robot's task because the required equipment is expensive and the environments are extensive and dynamic [106]. Instead, in field robotics applications, the robot's plan and execution must account for uncontrolled disturbances, such as ocean currents or wind gusts, which are not known with precision, and the robot must learn and adapt to these disturbances online. Additionally, field robots customarily have to deal with the restrictions their controllers and actuation abilities place on what these robots can achieve, which again is lessened in controlled environments through engineering solutions, such as over-actuation.

These factors lead to a large gap in performance by current state-of-the-art algorithms between these two classes of environments. One noticeable difference can be seen in the divergence commonly seen between the path *planned* and the path *executed* by robots in field robotic applications. When flying aerial vehicles [47] or piloting marine vehicles [114], it is common for the vehicle to execute a substantially different trajectory than the one specified by the operator (see Figure 1.1). In controlled environments the practitioner is able to engineer the environment to minimize this difference. However, in uncontrolled environments, this kind of solution is not possible and instead, the robot must account for these uncontrolled aspects of the environment both during path planning and path execution. We term this problem of minimizing the difference between the robotic vehicles planned path and the one it realizes during execution the *Realizable Path Planning and Execution Problem*.

Realizable path planning and execution is especially important in problems where the objective is not just to achieve a set of goal states, but also to optimize some objective along the path. Consider a problem where a robot must deploy a set of sensors to a number of locations in the environment. If the only requirement on the system is that the sensors are deployed to the desired location, it does not matter how the robot achieves those locations. However, once additional constraints are added to the problem, such as minimizing fuel cost during deployment or avoiding obstacles, then the path the robot takes to achieve these goal states becomes important. Additionally, when the robot plans a path to optimize these objectives (e.g. minimize fuel usage) and there is a large difference between the path planned and the path realized, the robot may not be able to achieve the desired objective (e.g. it will run out of fuel before deploying all sensors) or will result in a performance that is significantly different from the expected (e.g. information gathering tasks).

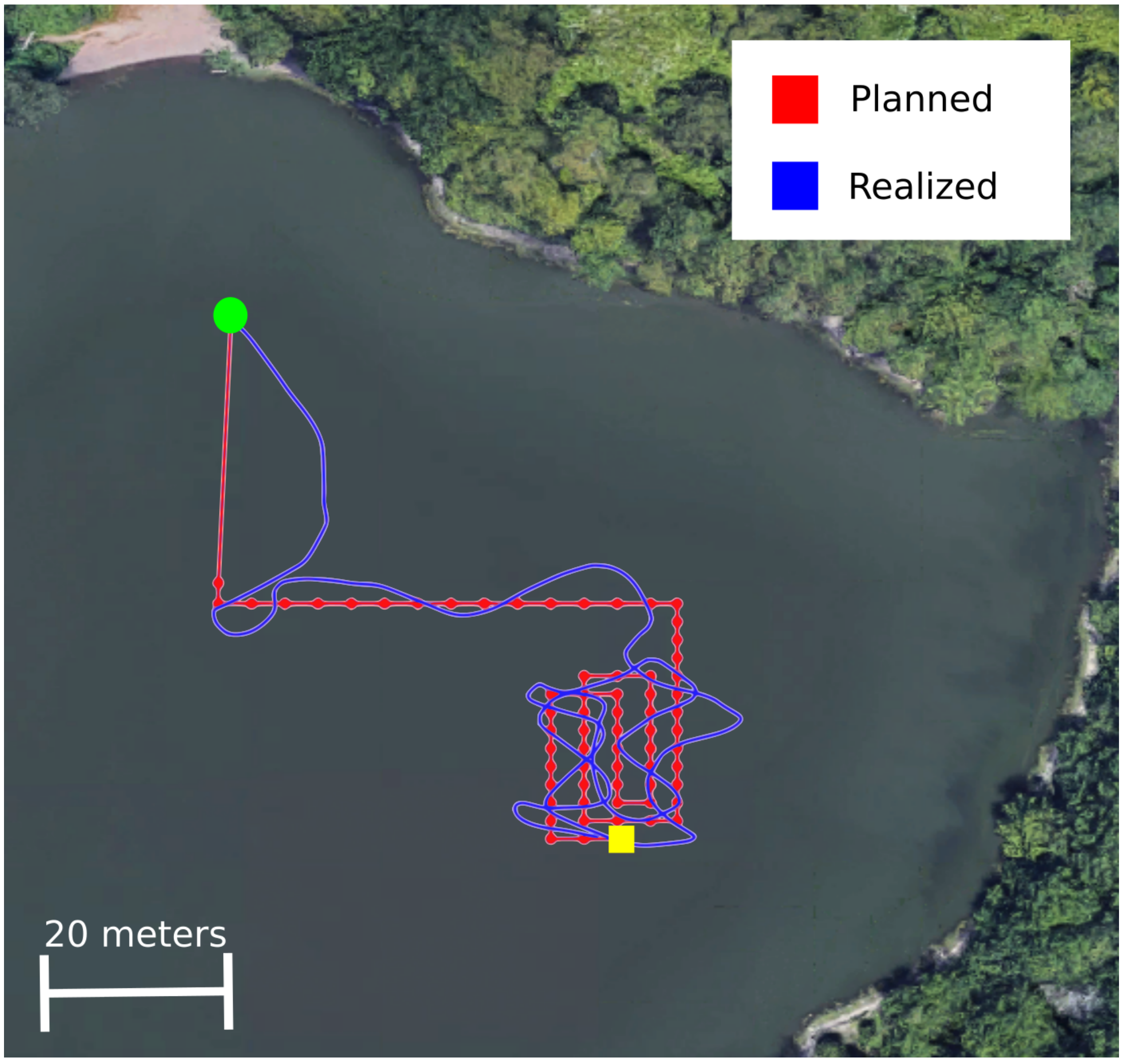


Figure 1.1: A planned (red) and realized (blue) path from a deployment of our Lutra Prop autonomous boat from Platypus LLC at Ireland Lake in Corvallis, OR (Lat: 44.563, Lon: -123.249). The vehicle's path starts in the upper left (green circle) and it travels towards the bottom right (yellow square). Notice that there is a large difference between the path that was planned and the realized path that the robot actually executed, especially on the tight turns of the last half of the plan.

We have identified three major issues that cause this disconnect between planned and realized paths:

1. During the planning process, robotic systems typically ignore environmental disturbances and assume that the low-level controller is sufficient to handle them. Built into this premise is the assumption that the robotic system is capable of actuation that is stronger than that of the disturbances. Yet, in many applications for which we would like to utilize robotic systems, such as marine monitoring [97] or aerial surveillance [83], the disturbances (ocean currents or wind gusts) are equal to or stronger than the actuation capabilities of the system. This can cause situations where the robot cannot execute its intended path and instead ends up in an undesired state.

2. Typically, field robotics deployments are in dynamic environments where it is unrealistic to assume that the system will have full knowledge of the obstacles, disturbances, and the robot's systems dynamics. Dealing with all of these uncertainties is difficult as it requires reasoning over a belief about the robotic system's states [10] as well as potentially reasoning over possible environments and disturbances [58], all of which introduce a significant computational load. However, robots are physically present in the environment and can gather information about these uncertainties in the environment. Many existing planning frameworks do not utilize this ability in a computationally efficient manner or fail altogether to consider this additional information.

3. Often the dynamics of robotic systems are highly non-linear, and many simplify-

ing assumptions on the system's motion are made during planning. One typical assumption is to make straight-line connections between states using graph search [49] or sampling-based [48] planning algorithms. Another common approach is to linearize the non-linear dynamics of the vehicle around a reference point [17]. These assumptions about the robot's system dynamics cause planners to output unrealistic paths for the robotic vehicles to execute and can require additional computation afterward to smooth the path. Additionally, trajectory smoothing can cause the trajectory to deviate from the original goals of the planner (e.g. information gain or energy efficiency).

Taking these issues into consideration, we can visualize the capabilities of robotic systems based on two major axes as shown in Figure 1.2. On the x-axis is the knowledge of the environmental disturbances that the system has, while on the y-axis is the relative actuation strength the system has relative to the environmental disturbances, which we have divided into three groups. In low relative actuation domains, the robot's actuation (u_r) is dominated by the environmental disturbances (u_e), so $u_r \ll u_e$. In moderate relative actuation domains, the robot's actuation is approximately equal to the environmental disturbances, so $u_r \approx u_e$. In high relative actuation domains, the robot's actuation is much stronger than the environmental disturbances, so $u_r \gg u_e$. Previous work (discussed in detail in Chapter 2) has primarily focused in two regions. First, the body of literature on planning has primarily focused on the upper right of this diagram where disturbances are well known and relative strength of actuation is strong. Second, the body of literature on control has mainly focused on reactive control strategies where the disturbances are not known and relative actuation strength is high.

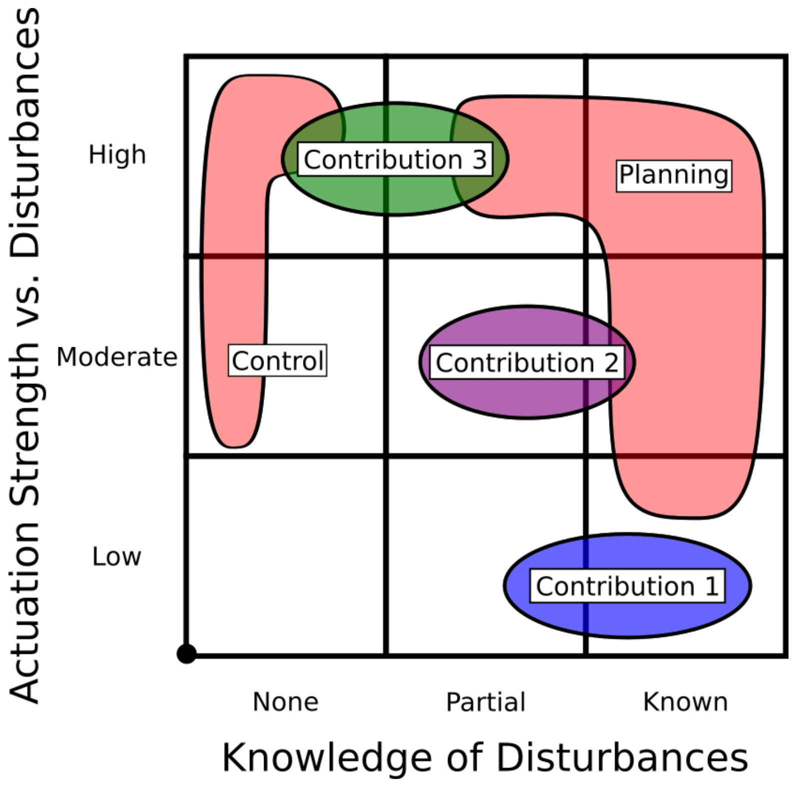


Figure 1.2: Our contribution relative to previous work on the axes of the knowledge of environmental disturbances and the strength of the system's actuation versus those disturbances. Previous work has focused on the upper right and left of this space. The contributions within this thesis have moved previous work further left and down in this space across all levels of relative actuation strength.

The research proposed here seeks to improve the ability of robotic systems to fully realize their planned paths by accounting for these three separations between the theory and the practice of using robotic systems. This leads us to our thesis statement:

Algorithms that reason about large-scale environmental disturbances and uncertain vehicle dynamics, while also incorporating new information about the environment gathered during execution, can improve the realizability of robotic path planning.

In this thesis we propose three primary contributions towards this thesis statement as follows:

1. A stochastic optimization algorithm that utilizes an action-space planning representation for low-actuation vehicles in known disturbances with a state-space based reward function. By planning in the action-space of these low-actuation vehicles, we can ensure that they realize the planned path. This primarily addresses the the first major issue discussed above by more explicitly incorporating disturbances into planning.

2. An algorithm, Energy-Efficient Stochastic Trajectory Optimization, which allows for moderate-actuation vehicles to exploit environmental disturbances for energy-efficient plans. Additionally, we introduce a framework for intelligently incorporating information gathered about these disturbances during execution. By accounting for disturbances and incorporating knowledge of disturbances back into the plans, this algorithm is able to increase the realizability of the planned path. This addresses the first and second major issue discussed above by rea-

soning about the effects of the disturbances and adapting the plan online to the disturbances measured.

3. A framework for increasing the realizability of planned paths for high actuation vehicles which allows the robotic system to reason about the capabilities of the on-board low-level controller. By incorporating the capabilities of the low-level controller into execution and planning, this framework is able to increase the realizability of the planned information gathering path. This addresses the first and third major issues by folding back in the knowledge of the low-level controller into the planning and accounting for the goals of the planner during execution.

Thesis Roadmap

In Chapter 2 we provide a background for motion planning, control, and field robotics to provide context for the contributions of this thesis.

In Chapter 3 we introduce a stochastic optimization algorithm for planning for multiple low-actuation robots in strong disturbances for an information gathering task. We demonstrate how an action-space representation of the state allows for an efficient incorporation of disturbances and that this optimization framework is computationally efficient.

In Chapter 4 we introduce our Energy-Efficient Stochastic Trajectory Optimization algorithm, which incorporates reasoning about environmental disturbances for moderate-actuation vehicles. Additionally, we introduce a framework which takes advantage of the computational efficiency of this optimization method to incorporate information gathered about the environment during execution for replanning of paths.

In Chapter 5 we introduce our framework for allowing a robotic system with high-actuation to reason about the capabilities of its low-level controller. By utilizing a reinforcement learning based approach the system is able to learn a policy capable of reducing the distance between the planned and realized path. Additionally, using a stochastic optimization algorithm, the framework is capable of increasing the realizability of the planned path.

In Chapter 6 we summarize the contributions of this thesis and re-emphasise the major theme and provide potential future research directions

Chapter 2: Background

As presented in the introduction, the bodies of literature in planning and controls have looked at different aspects of the realizable path planning and execution problem. The planning literature (Section 2.1) has mostly focused on vehicles where the disturbances are well known (or can be ignored) and on vehicles with moderate to high relative levels of actuation. This prior work can be broadly grouped into three major approaches: search, optimization, and sampling. Search-based planners (Section 2.1.1) were some of the first planning algorithms and rely on splitting the environment into a large number of discrete states and searching through the resulting graph formed by connecting those states. Optimization techniques (Section 2.1.2) rely on formulating the problem as either a maximization or minimization problem on the path. Sampling-based planners (Section 2.1.3) seek to find a path by probabilistically building a graph over the environment and then planning through that graph.

On the other hand, the controls literature (Section 2.2) has focused on domains where the disturbances are not well known and the relative actuation is high. Classical approaches (Section 2.2.1) have typically framed the problem as an optimization problem based upon the Bellman Equation [41]. Recently, there has been interest in using learning-based approaches (Section 2.2.2) in control.

In this chapter we will also provide a background on field robotics and the applications (Section 2.3) we will be considering in this dissertation. One common application for field robots is the Informative Path Planning Problem [93] (Section 2.3.2) where the goal is compute a path through the environments that gathers the most information. Another important task for field robotics is energy-efficient planning [77] (Section 2.3.3) where the goal is to compute a path from the start to the goal which minimizes the amount of energy the robot consumes.

2.1 Planning

A typical motion planning problem consists of five major elements [49]. The first is a *state*, which encodes information about the robot such as its position and orientation in space or the configuration of its arms. These states can be represented either continuously or discretely. Second, planning has at the very least an implicit representation of *time*, which indicates the order in which actions are to be performed. Third, planning has *actions*, which represent how the robot can change its state. Additionally, more complicated planning problems also deal with actions performed by the environment (disturbances) which change the state of the robot without the robot taking an action. Fourth, planning has *initial* and *goal* states, which informs the planner where the robot begins and where the robot desires to end. Lastly, there is a *criterion*, which informs how the robot is to plan. Generally there are two major different criterion considered, *feasibility* and *optimality*. *Feasibility* tells the planner to ensure achieving the goal state, regardless of efficiency. Feasibility is a binary function of the path, either it is feasible or not consider criterion based on some constraints either on the state transitions (such as turning constraints) or on the states themselves (such as no collision constraints). *Op-*

timality tells the planner to achieve the goal state with respect to some underlying cost or reward function. One note to make here is that when using a cost function the planner typically attempts to minimize cost, while when using a reward function the planner typically attempts to maximize reward.

Another way to describe planning algorithms is as complete and optimal [98]. An algorithm is called complete if, when at least one feasible path exists, the algorithm will return a feasible path. One caveat here is that this feasibility requirement is tied to the assumption made by the planning algorithm. One of the clearest examples of this is that of discrete planners, which partition the environment into cells, and are only resolution complete, meaning that these algorithms only return feasible paths which respect the resolution of the cells used. An algorithm is considered optimal when it returns the path which is optimal with respect to the specified cost function.

2.1.1 Search

The classic example of a search algorithms is the graph search algorithm A* [25] and its descendants such as Dynamic A* (D*) [98], which was designed to work in partially known environments, and Anytime Repairing A* (ARA*) [55], which was designed to give A* anytime properties and would continue to improve the solution as computation time was extended. While there has been a large amount of success using these algorithms, these algorithms tend to suffer from the curse of dimensionality [9], meaning that as the number of dimension increases linearly, the number of states increases exponentially. Additionally, due to the discretization of the state-space, many times these graph search algorithms need to make assumptions either about the environment or the robot that do not hold in increasingly complex environments, such as restricting movement to grid squares.

Some of the first attempts to remove these unrealistic assumption on motion used the idea of a *lattice graph* [76], which is a graph built over the space using a set of kinodynamically feasible motion primitives. This framework was used successfully as the local planner for Stanley [107], the robotic car from Stanford that won the DARPA Grand Challenge. In Stanley, the motion primitives were tailored around the speed and so allowed Stanley to plan high speed maneuvers. Recent work using lattice planning has looked at methods for using lattices constructed in one environment to plan in another because constructing these lattices is computationally expensive. In [85], the authors look at using a state lattice constructed in a regular grid to plan in a curved deformed environment. This allows the robot to reuse lattices, rather than needing to re-create one for every different environment. In [67], the authors leveraged local optimization of lattice actions to improve the underlying lattice graph. To keep this method computationally feasible, the authors developed a learning method which learns what areas of the lattice to refine.

Another discrete approach is the maneuver automaton, introduced in [21]. The authors construct a state machine which encodes actions and their transitions. The authors then plan over this state machine, choosing maneuvers and the amount of time spent executing them as well as the transitions between these maneuvers. However, properly designing this set of maneuvers is non-trivial and has a large effect on the solution quality. In [88], this idea is expanded by using controllable linear modes, which are more expressive than maneuvers. The authors formulate this planning problem as a mixed integer linear program which limits the length of the path that it can consider.

Existing search based planning methods fail to adequately solve the realizable path planning and execution problem because the discritization that they impose to be computationally efficient enough typically place too many restrictions on the robot. Additionally, when vehicles have low to medium relative levels of actuation the needed checks on the connections between states can significantly increase the computation time. Lastly, these methods tend to not naturally allow for the incorporation of disturbances and so make reasoning over them difficult.

2.1.2 Optimization

Another approach to motion planning is to use optimization-based planning methods. These methods work by taking an initial solution to the planning problem generated by a naive planner and iteratively improving these solutions to determine the best solution. There are both deterministic and stochastic methods for path optimization. In [116], the authors introduce Covariant Hamiltonian Optimization for Motion Planning (CHOMP), which calculates the analytical gradient of a cost function based upon smoothness and obstacle cost functions to iteratively improve a path. In [89], the authors present an algorithm called TrajOpt, which uses sequential convex optimization and a novel collision checking formulation to reduce the number of iterations the algorithm requires to converge. In [65], trajectories are represented as a Gaussian Process, and a gradient-based optimization technique is used to optimize the path. One of the biggest issues with these planning methods is that they are highly susceptible to finding local optima instead of the global optimum of the cost function. The first stochastic optimization techniques attempted to solve this problem by either using random restarts (i.e. reinitializing the problem) or using a large number of initial solutions (particle swarms) to make the algorithm less sensitive to these local optima. CHOMP uses random restarts in an attempt to move the solution out of the basin of attraction that it starts in to attempt to find a more cost efficient solution.

Particle swarm algorithm keep a population of solutions and refines this population by improving each solution, which allows the algorithm to explore the space of solutions. In [112], the authors use a particle swarm optimization algorithm to plan energy efficient paths in ocean currents for autonomous underwater vehicles (AUVs). This particle swarm formulation allows the planner to search many different basins of attraction which are present in the highly non-linear cost function of energy usage by the AUV in ocean currents. Recent work [102], has used shooting methods in combination with random restarts and a trajectory database to efficiently calculate motions for humanoid robots. Shooting methods work by taking an initial condition and plan and simulating it forward using a simulator to find the final configuration. These methods suffer from needing many calculations to find an acceptable solution. The authors in [102] use a pre-computed trajectory database to speed up this search by providing the shooting method with high quality starting trajectories.

Another approach to stochastic optimization motion planning is to use sampling to estimate the gradient of a non-differentiable cost function. Borrowing from the stochastic optimal control literature [105], the authors in [36] present Stochastic Optimization for Motion Planning (STOMP). STOMP calculates trajectories by starting with an initial trajectory guess and iteratively updating this trajectory. These updates are calculated by sampling trajectories around the current trajectory and using these samples to approximate the gradient of the cost function, which can then be used to update the path. One advantage of STOMP is that the paths produced by the algorithm are smooth and so are more easily executed by the robot. STOMP was originally designed to be used for a humanoid robot in grasping tasks trying to minimize torque. STOMP has also been used in the marine domain [23], where it was used to update local coverage plans when obstacles differed from their initially mapped positions. In this case, the method was initialized with the originally planned path and [23] utilized the smooth updates offered by STOMP to keep the calculated solution smooth, which is important for their application of providing high quality coverage plans.

Optimization-based planning methods offer an interesting avenue for realizable path planning. However, existing methods are incapable of fully considering the effects of disturbances on the robot due to the restrictions on the cost functions they can consider.

2.1.3 Sampling

There are two main algorithms which utilize sampling techniques: Probabilistic Road Maps (PRM) [39] and Rapidly-exploring Random Trees (RRT) [51]. Both these algorithms use sampling to build an underlying data structure that is then used during planning.

PRMs build a graph over the environment which a standard graph-search based planner can then plan over. To do this, the PRM algorithm samples a number of states from the free space and then attempts to find collision free connections between these states using a number of connection criterion depending on implementation. Some common connection criteria are k-nearest neighbors and connecting to all neighbors within some radius r [64]. One of the most difficult and computationally expensive steps in constructing a PRM graph is determining the feasibility of these connections. This is due to the difficulty in performing collision checks along the entirety of the path and potentially considering dynamic constraints on the robot's motion. There have been a number of extensions to this basic PRM formulation. In [7], the authors introduce Lazy PRM which attempts to speed up computation by only checking for collision on edge connections when that edge is included in the solution. The authors in [92] propose an updated sampling technique that biases samples to be near obstacles. By doing this the authors attempt to alleviate the problem of finding narrow passageways. In [50], the authors in part examine the relationship between sampling strategies and solution quality. The authors find that using more informed sampling techniques produces shorter solutions faster. Another interesting extension to PRM was introduced in [37], where the authors introduced PRM*. The main contribution was analytically deriving an optimal connection radius, r^*, which is based upon the number of samples already in the graph and the dimensionality of the problem. This allows PRM* to intelligently consider connections and minimize the number of expensive connect states computations. One problem that PRMs face is that building this underlying map can be expensive and requires a map of the environment to be known a priori. This can be a problem when the environment is uncertain and the robot may require a large amount of replanning. Additionally, the path returned by searching on a PRM may be suboptimal.

RRTs [48] build a tree structure that is rooted at the starting state and explore the environment until the goal state is found. Then the robot executes the plan by traversing the branch of the tree which leads the robot from the start state to the goal state. To build this tree structure, the RRT algorithm samples a state randomly from the environment or configuration space, finds the nearest state currently in the tree to that sample and then grows the tree from the nearest node in the tree in the direction of that sample using a predefined steering function. Traditionally the steering function used has been to grow the tree in a straight line, which suffers from problems when trying to connect near obstacles and does not accurately reflect how the robot will move.

In [103], the authors develop a steering function which utilizes obstacle information to help grow the tree through narrow passages. More complicated steering functions that attempt to more accurately reflect the dynamics of the robot have been considered. In [73], the authors propose a non-holonomic distance function that considers constraints on the vehicles motion. They also introduce a steering function using this cost function for a unicycle-type vehicle. In [72], the authors use an optimal controller to locally extend the tree built by the RRT algorithm. However, this is based only on a wheeled robot and makes the assumption that the velocity throughout the extension is decaying, which limits its usefulness when considering a string of high speed maneuvers. Another approach introduced in [110] solves this problem by assuming controllable linear dynamics and then solving the two-point boundary value problem. The assumptions introduced by linearizing the dynamics limits the systems for which this method can be used and, as noted by the authors, imposes limits in some cases on the radius of connection that can be considered. Additionally, this connection method is computationally expensive when the system cannot use a derived special case closed form solution and must instead use a numerical approximation algorithm.

Similarly to PRMs, there has been work looking at different sampling strategies for RRTs. In [109], the authors present a number of different methods, such as biasing samples towards the goal state, that help to improve the performance of the RRT algorithm. This was done to help increase the computational efficiency of the algorithm. While RRTs provide fast single query searches, many times the found path is sub-optimal and contains many sharp motion transitions.

In [37], the authors expand the RRT algorithm to RRT*, which is an asymptotically optimal algorithm, meaning that as the number of samples goes to infinity the solution converges to the optimal solution. To extend RRT to RRT*, the authors developed an edge rewiring technique that recomputes state connections in the tree as new samples are added. Additionally, RRT* is a probabilistically complete algorithm, meaning that as the number of samples goes to infinity the algorithm will find a solution if one exists. The authors in [68] introduce RRT*-Smart, which helps to improve the convergence rate of RRT* by biasing the sampling towards the first solution that is found. By doing this, the authors are able to increase the computational efficiency of RRT*. While RRT* has been used in a number of robotics research applications, in practice its use has been limited as the number of samples needed to significantly improve the solutions is quite high.

This is similar to recent work in Monte Carlo Tree Search (MCTS), which uses intelligent sampling to quickly explore a large amount of the state-space. In [4], the authors present Decentralized MCTS (Dec-MCTS) which is used to plan for multiple robots in an active perception problem. In Dec-MCTS, the robots keep a probability distribution over the plans of other robots. This distribution helps to simplify planning and offers a principled way to deal with uncertainty about other robots.

Another approach to uncertainty planning was introduced in [10], where the authors introduce Rapidly-exploring Random Belief Trees (RRBT). RRBT works very similarly to RRTs discussed above, however it introduces the idea of comparing partial paths that arrive at nodes which account for the uncertainty in the state of the robot. The authors then compare partial paths that lead to the node and impose an ordering based upon the uncertainty. By doing this, the authors are able to plan a path where the robot is more certain about its location and so able to avoid collision. In [57], the authors extend upon RRT* to develop Chance Constraint RRT* (CC-RRT*) which checks probabilistic feasibility using chance constraints. Another approach explored in [95] constructs an underlying PRM graph of the space with uncertainty on the edge weights. The authors present Risk-Aware Graph Search (RAGS) which plans an initial path and then continues to update that path as edge weights are discovered. This approach demonstrates the importance of replanning when dealing with uncertainty, as it allows the robot to account for newly gathered information during execution. In [13], the authors present Online RRT* (ORRT*), which dynamically adds samples and rewires the tree during execution. This allows the planner to account for uncertainty by efficiently replanning when unexpected obstacles are discovered.

The common assumption made by these works is that these disturbances will be small compared to the actuation of the vehicle. There has been some limited work using

RRTs with large environmental disturbances. In [81], the authors use an RRT-based method to plan paths for ocean gliders in ocean currents. The authors introduce a steer function which accounts for the ocean currents when expanding the tree. However, the ocean currents introduce a bias into the tree growth, which causes the planner to perform poorly when the optimal paths are against this bias introduced by the ocean currents. In [101], the authors introduce an RRT-based algorithm which uses a reconnection scheme to help deal with dynamic constraints. The authors continually prune the tree back after a feasible trajectory is found and use this to both speed up replanning and bias tree growth around existing solutions.

Sampling-based planners are a powerful set of algorithms for solving planning problems. However, most of the work using these planners assumes simple steering functions. Even when more computationally expensive steering functions are used these planners ignore large-scale disturbances. Lastly, incorporating disturbances into these frameworks can cause unforeseen problems, such as those seen in [81].

2.2 Control

A simple way to think about a control system is that it is a system that takes in an input and converts that input into a desired output to achieve a desired performance [70]. A control system can be evaluated on a number of different performance metrics but some common metrics are transient response (the system's performance as it approaches the solution), steady-state response (the system's performance as it attempts to hold a state), stability (does the system eventually settle down to near the desired state), and robustness (how does performance change as the parameters change). Classical controls methods are reviewed in 2.2.1. Recently there has been interest in using learning-based control methods [20], which are discussed in 2.2.2.

2.2.1 Classical

In [104], the author introduces the algorithm LQR-Trees, which use linear quadratic regulator (LQR) controllers to compute locally valid controllers, and combines them into non-linear combinations which can cover the space of the trajectory and be used in planning. This work borrows from the sampling-based motion planning community by growing a tree of states and controllers by using sampling to add nodes to the tree and then designing the local linear controller around that state if it is required. In [59], the authors introduce a related idea of planning with a funnel library. These funnels represent an outer limit on the area reached during execution of the trajectory and are calculated using sum-of-squares programming. By stringing these funnels together, the authors can compute both a trajectory which respects the dynamics of the vehicle and a region around the trajectory within which the robot will stay.

Another common controls approach is to use Model Predictive Control (MPC) [12], which utilizes a predictive model and objective function to compute a control law. MPC methods forward simulate the system with the planned controls to calculate the future controls that will be needed. This offers an advantage over controllers which do not consider the future by analyzing the impact controls executed now will have on the needed control signals in the future. Additionally by optimizing the set of controls the robot will use, MPC methods are able to plan paths which respect the kinodynamic constraint of the robot. In [19], the authors use MPC to stabilize a ROV in ocean disturbances in a station-keeping task. This work illustrate one of the major drawbacks to these MPC formulations. The approach is only able to plan over a time horizon of 0.8 seconds, which is quite limited when trying to plan longer term plans for the robot. One way to increase this time horizon is to consider a more limited set of actions and make more assumptions on the vehicles motion. This approach is taken in [29], where the authors use MPC to compute paths for AUVs in strong and uncertain ocean currents. The controls are discretized into 16 different thrust directions and 24 different depths. Using this discretization the authors are able to utilize an A* solver which allows them to consider long term path through the ocean. While their method is able to offer improvements in path realization by these vehicles in simulation, the authors still make assumptions on the vehicles motion, namely that depth transition can happen instantaneously, which is not realizable in the physical world. Another method of speeding up calculations is to use a learned model for vehicle dynamics. The authors in [111] present an MPC based method which uses a neural-network based model of the vehicle dynamics alongside a reinforcement learning based control strategy on an aggressively driving model car. While this provides strong results, it does require an amount of learning on the real system, something which may not always be available. Additionally while they are able to consider further ahead, the time horizon is still limited to approximately 2.5 seconds.

Classical control solutions are a powerful set of tools for realizable path planning. However, these methods are unable to plan over long enough horizons for full realizable path planning and are not suitable for exploiting large-scale disturbances for vehicles with low or medium relative levels of actuation.

2.2.2 Learning

There has recently been a great interest in using data-driven learning methods for robot control. In [53], the authors developed a technique using deep neural networks and reinforcement learning to train a robot to perform manipulation tasks. The authors trained the robot to map raw images directly to desired motor torques. Similarly in [54], the authors utilized training examples directly from a number of robots to train a policy for grasping tasks. In [56], the authors demonstrate a deep learning method for continuous actions in 20 different simulated physics tasks. In [115], the authors propose a method for learning stacking and pushing actions directly from camera images. However, in this and similar domains the environmental disturbances are much smaller than are typically experienced in field robotics. In [54], the authors developed a deep neural network approach for robotic grasping that was able to achieve effective real time control while also successfully grasping novel objects. However, like many deep networks, their system required a large amount of training data, approximately 800,000 grasp attempts on real robots. This is not feasible for field robotics, especially across a large number of potential disturbances. [111] demonstrated a learning-based dynamics model within an information theoretic MPC scheme for handling complex cost functions and nonlinear dynamics. However, the authors' method depends upon executing and learning in the same environment and is not easily generalizable. [71] similarly develop a path tracking method which uses a Gaussian Process (GP) to model disturbances and is trained from previous path executions. This GP model can then be used by a nonlinear MPC algorithm to accurately track a desired path. However, again the state of the robot is defined in global coordinates and the disturbances that the algorithm is modeling are dependent on the desired path, requiring the path to be executed multiple times for reliable path tracking.

Additionally, many learning methods have only been applied in simulation and though there is some work on transferring work from simulation to the real world [86], it is limited and presents a number of unique challenges. Another approach to tackle this sim-to-real gap is presented in [74], where the authors use both the common technique of domain randomization during training, as well as an imitation learning based scheme to help increase the performance of the real system. However, this technique requires a large number of samples from the real system, which is not always feasible for a field robotic system. When these systems fail it is often quite hard to understand why and as such, present difficulties in utilizing these system. Lastly, many of these systems do not consider problem with large amounts of noise in the motion models (either through disturbances or noisy vehicle dynamics).

2.3 Field Robotics

There has been significant interest in using robots for a number of field robotics applications [106]. There have been many varied applications for field robotics but in this dissertation we will mainly focus on three thrusts. First, in 2.3.1 we will talk about work in planning for disturbances. Next, we will discuss the Information Gathering Problem in 2.3.2. Last, energy-efficient path planning is discussed in 2.3.3.

2.3.1 Disturbance Planning

To address the problem of planning achievable paths in disturbances, [96] creates a controllability map over the ocean environment. However, they use this controllability map to plan trajectories for patrolling tasks and do not account for the situation where the agent may want to enter a region of low controllability if the direction of controllability is favorable. Additionally, the authors in [96] utilize an A* based path planner, which is ill-suited to many problems as it is quite difficult to design an informative heuristic for many field robotic tasks. In [63], the authors describe a high-level controller design for spreading and attracting Lagrangian drifters. However, they only consider final destinations and do not account for path dependent rewards such as information.

In [111], the authors introduce an information theoretic solution method for model predictive path integral (MPPI) control. To calculate updates to the control, the authors use a large number of samples to approximate the KL-Divergence between the current control distribution and an optimal control distribution, which is used to derive an iterative control update law. In this chapter, we use sampling to approximate a gradient, which allows our method to use a comparatively smaller number of samples. In [38], the authors propose a framework for robotic operations in adversarial forces. Through a waypoint augmentation algorithm, the authors are able to improve the path tracking performance for an autonomous boat subject to wind and current disturbances. However, this system only considers a single waypoint in the future and can result in large deviations at sharp turns.

2.3.2 Information Gathering

The Informative Path Planning problem [93] is a heavily researched problem in robotics and is classified as a NP-Hard problem [27]. Some of the major challenges in this problem are that it requires searching over the space of all possible paths and that the reward function is nonconvex in all but the simplest of domains. To cope with these difficulties, a number of different solutions have been proposed.

One paradigm has been to approach this problem as a sequential decision making problem. Researchers proposed algorithms such as Greedy, Recursive Greedy [93], and Branch and Bound [6] which iteratively add new local actions to the path. Because the Informative Planning Problem has a path dependent reward structure, this requires either a large number of path evaluations or strong assumptions about the structure of the information field. Additionally, these methods require discritization of the environment, which restricts the range of solutions they can consider. Other solution methods utilize sampling-based methods to effectively explore this large search space, such as Rapidly-exploring Information Gathering [27], Monte Carlo Tree Search (MCTS) [69] [5], and Bayesian Optimization [61]. Again, these methods require a large number of path evaluations and in their current formulation consider deterministic paths. Accounting for the uncertainty in path execution would be potentially computationally expensive as estimating the distribution of paths can be quite difficult and these sampling methods typically need a large number of samples to compute their plan.

Another approach to the information gathering problem is to frame it as an optimization problem where a preexisting or naïve trajectory is refined. Optimization techniques can be much more efficient than sampling based methods, but require additional assumptions about the problem. In [15], the authors formulated the problem as a Sequential Quadratic Programming problem which requires them to treat each measurement as independent. This does not allow for path dependent rewards, which is a key structure in the information gathering problem. The authors in [79] utilize an evolutionary algorithm to optimize the robot's path. While evolutionary algorithms allow for a wide range of different objective functions, these methods are computationally expensive and do not scale well. In [34], the authors utilize Stochastic Gradient Ascent (SGA) to plan informative paths for a team of robots but do not consider the realizability of the paths.

Our proposed method uses a similar structure to [34] to optimize the information gathering path for realizable information gain. While other existing methods could use a similar structure to that presented here, the SGA formulation we use here is relatively sample efficient. This allows the method to effectively use the information from these samples to calculate the most informative path.

2.3.3 Energy-Efficient Planning

Previous research on planning energy-efficient paths in ocean currents has focused primarily in two directions. A number of different works, such as [52], [29] and [46], have used A*-based algorithms to plan energy-efficient paths due to the limited availability of coarse data and the relative tractability of discrete approaches. However, the necessary discretization limits the paths that the planner can find and can produce infeasible transitions for the vehicle to perform. In [81], a RRT-based method is used to compute energy-efficient paths. Due to the bias in tree growth due to the ocean currents, the method was not always able to outperform A* based methods and final path quality could fluctuate widely. Other work has examined different optimization methods. In [44], a gradient optimization based method is used to plan energy-efficient paths in an estuary environment. To calculate these gradient updates, the authors assume a smooth water current distribution in addition to a smooth path cost function, which cannot always be assumed. In [112], a particle swarm optimization approach is used to find energy-efficient paths. This approach is computationally expensive and not suited for real-time planning. In [99], a level-set expansion based method is employed given a desired start and end position for the vehicle. This strategy requires both full knowledge of the flow field and high computation, restrictions that make it unsuitable for real-time planning in uncertain ocean currents. In our work we account for the uncertainty present in ocean current forecasts by replanning throughout the trajectory as more information about the ocean current environment is gathered, rather than assuming that ocean currents are known perfectly.

2.4 Comparison of Related Work

Now that we have gone through the background work in planning, control, and field robotics we can place this dissertation in more context. In Figure 2.1, we can see the same two axes from the introduction (Figure 1.2; relative actuation strength and knowledge of disturbances, with related work filled in and the chapters in bold. As we can see, previous work has focused on the upper right and left of these axes. In Chapter 3, we provide new algorithms for planning for low relative actuation vehicles in known disturbances, which are computationally efficient enough to consider being used in partially known disturbances domains. Chapter 4 helps to bridge the gap between the work in planning and control by presenting algorithms which allow moderate relative actuation vehicles planning in partially known disturbances. Finally, in Chapter 5 we leverage solutions from both planning and control to provide a new algorithm and framework for realizable path planning with high relative actuation strength and partially known disturbances. One aspect here to note is that the lower left of this diagram is still left blank. This domain is especially difficult because knowing very little about the disturbances and having a low relative actuation strength greatly limits the impact of any choice the robotic vehicle could make. We leave this challenging region of the problem domain as an avenue for future work.

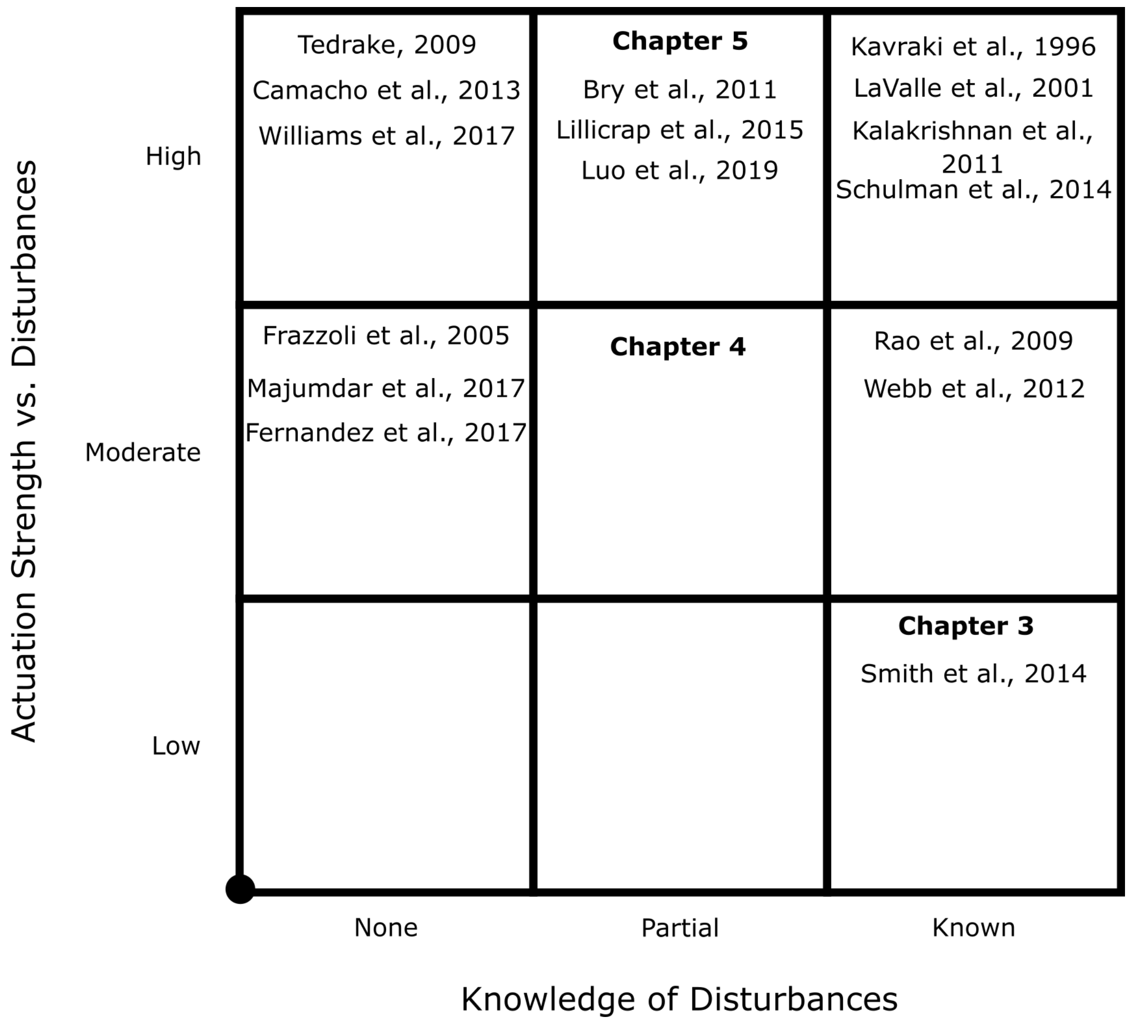


Figure 2.1: A representation of where previous work has focused (upper left and right) and where this thesis fits into the existing work. Chapters in this thesis are in bold.

Chapter 3: Action Space Representation for Low Actuation Vehicles

In this chapter we present a method for realizable path planning for low actuation vehicles in known disturbances. Rather than planning in the state-space of the vehicle, this method utilizes a novel action sequence path representation. This path formulation allows for a more natural representation of disturbances into the planning problem and removes the need for expensive reachability calculations that previous work had to perform for realizable path planning. Specifically, our proposed technique bridges the gap between previous work in information gathering and previous work considering limits to control authority. Versions of this work have been previously published in [35] and are under review in [45].

We present a stochastic optimization algorithm for information gathering that:

- Allows for different levels of control authority by using a novel action sequence path representation
- Approximates the gradient of the path dependent state space reward function with respect to the action sequence path using random roll-outs
- Uses a Sequential Greedy Allocation scheme that allows the algorithm to be scalable for multi-vehicle implementations

We validate this algorithm on data from a Navy Coastal Ocean Model (NCOM) of the Gulf of Mexico, where the dataset is constructed similarly to [30] and can be seen in Figure 3.1. This data set includes ocean currents, which affect the control authority of the vehicles, and ocean temperatures, which we use to construct an information function. We assume that there is some desired temperature that could correlate with an oceanographic process of interest, such as algae blooms, and assign each location an amount of information corresponding to how close it is to that desired temperature such as in [11].

For this chapter it may be helpful to review the background on optimization (Section 2.1.2) and sampling (Section 2.1.3) based planning and on information gathering (Section 2.3.2).

3.1 Problem Formulation

We seek to find the set of feasible paths for a team of vehicles which maximizes a given reward function. This is formulated as the following optimization problem:

$$\mathbb{X}^* = \max_{\mathbb{X}\in\Omega} \; R(\mathbb{X}), \text{ s.t.} \,\forall\, \mathbf{x}_i \in \mathbb{X}, \mathbf{x}_i \in \Psi, \tag{3.1}$$

$$\mathbb{X} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n\},$$

where $\mathbf{x}_i$ is the path of an individual agent, $\mathbb{X}$ is a set of paths, $R(\mathbb{X})$ is a user defined reward function, Ω is the set of all paths, Ψ is the set of all feasible paths, and n is the number of agents being considered. In previous work, such as [96] and [63], these paths were specified through a set of state-space coordinates defined by latitude-longitude-depth coordinates. However, optimizing the state-space coordinates directly

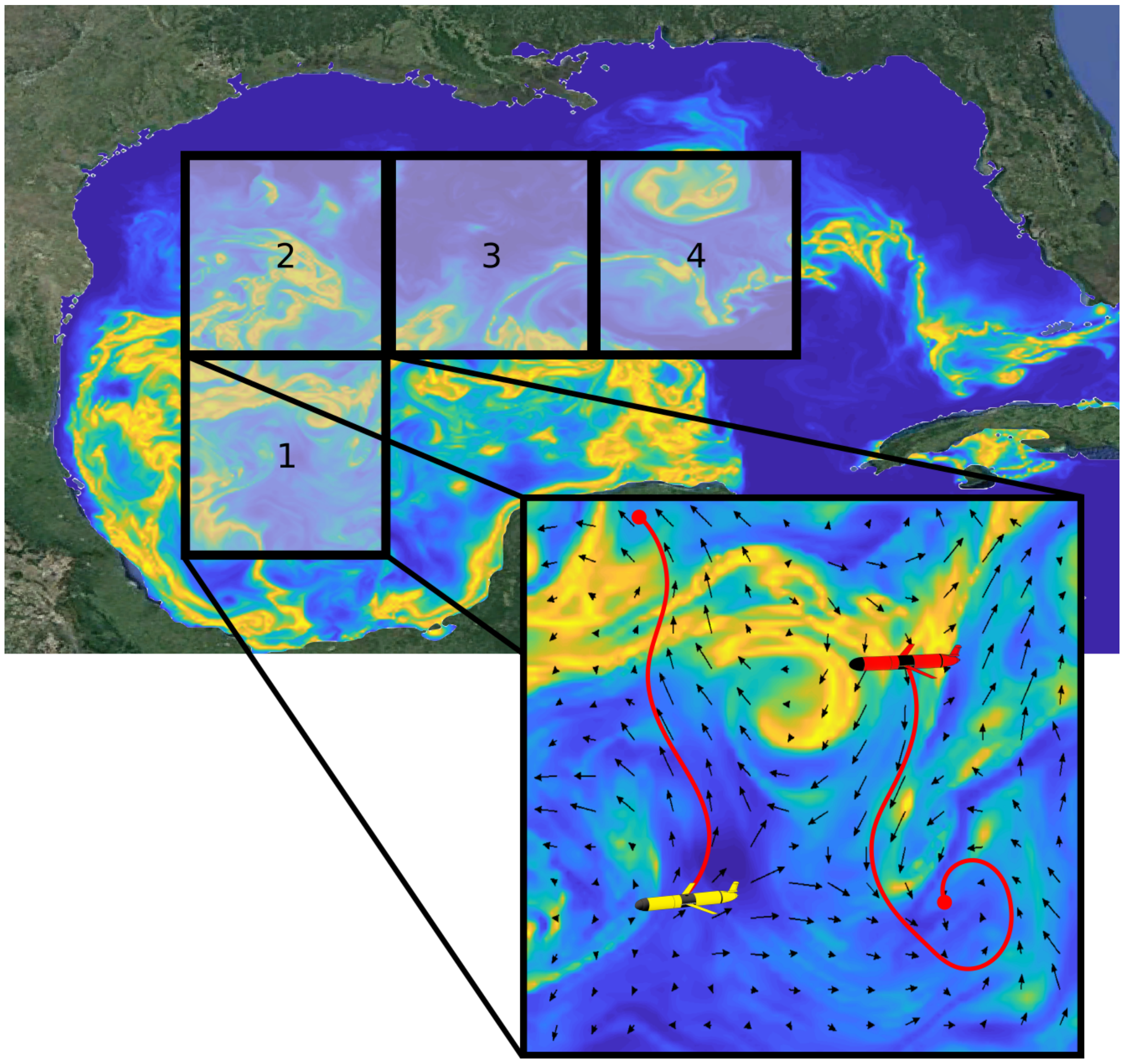


Figure 3.1: Representative information field over the Gulf of Mexico where brighter colors indicate more information. The inset shows a zoomed in view of one of the four designated test regions used in the results. Two vehicles' paths are illustrated in the inset. The vehicle on the left (yellow) is able to utilize the ocean currents to move from a location of low information to one of high information. In contrast, the vehicle on the right (red) starts in a region of high information but is unable to stay there due to the limited control authority introduced by the ocean currents.

requires nontrivial calculations to ensure that the trajectory is feasible due to control limits compared to environmental disturbances. We choose to plan over sequences of actions available to the vehicle, which removes the need to perform these feasibility calculations. Instead, we assume that we have a function $\Phi : \mathbb{A}, x_0 \mapsto \mathbb{X}$ where $\mathbb{A}$ is a set of action sequences and x_0 is a defined state-space starting location. This redefines our optimization problem as:

$$\mathbb{A}^* = \max_{\mathbb{A} \in \Lambda} R(\Phi(\mathbb{A}, x_0)), \tag{3.2}$$

where we are trying to find the optimal set of action sequences in Λ, the set of all possible action sequences.

We define an individual action sequence as:

$$\mathbf{a} = \{(d_1, t_1, v_1), (d_2, t_2, v_2), \cdots, (d_m, t_m, v_m)\},$$

where (d_i, t_i, v_i) defines a single action of diving to depth d_i, thrusting at velocity v_i, and maintaining that depth and thrust for an amount of time t_i. We assume that the agents have the ability to achieve and maintain a range of depths and provide a limited amount thrust. Using this definition, $\mathbb{A} = \{\mathbf{a}_1, \mathbf{a}_2, \cdots, \mathbf{a}_n\}$.

We use a generalized autonomous underwater vehicle (AUV) model to demonstrate our algorithm, but our formulation is general and a number of different sets of actions, a, can be used in the action sequence, $\mathbf{a}$. We make two assumptions about the actions that can be included in the actions sequence. First, we assume that the action can be represented by a function $f : x_0, a \mapsto x_1$, which maps actions to states with a defined on a range:

$$a \in \lambda = [\beta, \alpha],$$

where α defines an upper bound and β defines a lower bound. This means that the action can be represented as a continuous real number which can be used to map from one location to another. Second, we assume that it is simple to check both that $a \in \lambda$ and that x_1 is a valid state for the vehicle.

In this chapter, we are interested in information gathering tasks defined by an information function, $R(\mathbb{X}) = I(\mathbb{X})$, which maps a set of state-space paths to an amount of information gathered by the team of agents. Again, we can transform this into our action sequence representation as $I(\Phi(\mathbb{A}, x_0))$.

3.2 Stochastic Gradient Ascent Algorithm

We use the Stochastic Gradient Ascent (SGA) algorithm, shown in Algorithm I, to approximate $\mathbb{A}^*$ by iteratively updating an initial guess. The first step is to initialize the set $\mathbb{A}$ using a simple default policy. Next, we optimize the action sequences using a sequential greedy allocation method by optimizing one vehicle while holding all others fixed. This was shown in [28] to produce high-quality results in similar domains. For each vehicle, we iteratively improve the action sequence by calculating a number of perturbed sequences, scoring them, and then doing a weighted recombination of the perturbations to estimate the gradient which is used to update the sequence for the next iteration.

We will now go through each of the subroutines:

- **get_perturbed_paths**: We calculate the set of perturbations, E, in the following

Algorithm 1 Stochastic Gradient Ascent (SGA)

```
1: Initialize $\mathbb{A}$
2: for all $\mathbf{a}_i$ in $\mathbb{A}$ do                          ▷ Loop through vehicles
3:     $c \leftarrow 0$
4:     for $c < num\_its$ do
5:         $\Theta, E \leftarrow$ get_perturbed_paths($\mathbf{a}_i$)
6:         $S \leftarrow$ get_scores($\Theta, \mathbb{A}$)
7:         $\Delta \leftarrow$ estimate_grad($S, E$)
8:         $\mathbf{a}_i \leftarrow \mathbf{a}_i + \Delta * \eta(c)$
9:         $c \leftarrow c + 1$
```

way:

$$E = \epsilon_1, \epsilon_2, \cdots, \epsilon_K,$$

$$\epsilon_k = \{(\hat{d_1}, \hat{t_1}, \hat{v_1}), (\hat{d_2}, \hat{t_2}, \hat{v_2}), \cdots, (\hat{d_m}, \hat{t_m}, \hat{v_m})\},$$

$$\hat{d_i} = \mathcal{N}(0, \sigma_d),\ \hat{t_i} = \mathcal{N}(0, \sigma_t),\ \hat{v_i} = \mathcal{N}(0, \sigma_v),$$

where K is the desired number of perturbed sequences and individual waypoint perturbations $\epsilon_k^j = (\hat{d_j}, \hat{t_j}, \hat{v_j})$ are sampled from zero-mean normal distributions with standard deviations σ_d, σ_t and σ_v, respectively.

We then calculate the set of perturbed sequences, Θ, as:

$$\Theta = \theta_1, \theta_2, \cdots, \theta_K,$$

$$\theta_k = \mathbf{a}_i + \epsilon_k,$$

where θ_k is perturbed action sequence k produced by adding perturbation vector ϵ_k to $\mathbf{a}_i$.

- **get_scores**: We calculate the score of each individual action of all perturbed action sequences in Θ using a method inspired from difference learning, presented in [1]. This calculation approximates the contribution of action θ_k^j by estimating how much it improves over the existing solution. The score, $s_{k,j}$, is the score of action j of θ_k and is calculated as:

$$s_{k,j} = \mathcal{I}(\bar{\mathbb{A}}) - \mathcal{I}(\tilde{\mathbb{A}}),$$

where $\bar{\mathbb{A}}$ is $\mathbb{A}$ except $\mathbf{a}_i$ is replaced with θ_k and $\tilde{\mathbb{A}}$ is $\mathbb{A}$ with $\mathbf{a}_i$ replaced with θ_k but action θ_k^j replaced by a_i^j. By doing this, we also enable our gradient calculation to be more efficient by containing information specifically about the improvement offered by the waypoint, rather than containing a large amount of noise about the environment and other waypoints.

- **estimate_grad**: To calculate the gradient we need to convert the scores into costs by taking the inverse, which gives a cost matrix $C = \frac{1}{S}$. We then use the following calculation to approximate the gradient:

$$\Delta_j = \frac{1}{K} \sum_{k=1}^{K} w(\theta_k^j) * \epsilon_k^j,$$

$$w(\theta_k^j) = e^{-h\left(\frac{C_k^j - \min C_k^j}{\max C_k^j - \min C_k^j}\right)},$$

where the function $w(\cdot)$ is a weighting function and Δ_j is gradient at action j in the action sequence. The variable h serves as a weighting term and is set equal to 10 in this work. The $\max$ and $\min$ functions are taken over all K perturbed sequences at action j. This compares the cost of the action under consideration to that of all other actions at the same point in the path. By scaling the weighting by the maximum and minimum seen in that iteration, the algorithm is able to more accurately account for large improvements offered by a small number of samples. Additionally, during implementation we include the original path as one of the samples which helps stabilize the algorithm and allows for the gradient calculation to account for the existing solution and scale the contribution of each individual waypoint based upon how much the waypoint improves over the existing solution.

Using the calculated Δ_j values, we can compute Δ as $\{\Delta_1, \Delta_2, \cdots, \Delta_m\}$. Using this Δ, we can then update the sequence using $\eta(c)$ which serves as a learning rate, which can be a function of the iteration number and in this work is calculated as $\eta(c) = 0.99^c$.

3.3 Benchmark Algorithms

We now discuss the various algorithms we benchmark against our SGA algorithm. We assume an action sequence consists of seven actions, each of which lasts for a duration of 24 hours, which, unless otherwise noted, randomly selects a velocity which is executed for the duration of the mission. Below we list our five benchmark algorithms.

- **Const Depth**: The first default policy is to have all vehicles maintain the same constant depth for the duration of the mission. In this work, we have the vehicle maintain a constant depth of zero (i.e. floating on the surface).

- **Diff Depth**: The second default policy is to equally distribute the vehicles throughout the water column. By doing this, the vehicles hope to find different ocean currents which will cause them to spread out. In this work, we have the vehicles spread out every 20 meters in the depth column from zero to 180 meters.

- **Greedy Depth**: The third default policy is iteratively choosing a constant depth and thrust for a new vehicle as it is added to the team that maximizes the amount of information gathered by the team. To do this, we discretize the possible depths and thrusts and then forward simulate the team with the new vehicle at all these possible depths and thrusts. We then select the depth and thrust that maximizes the information gathered.

- **Rand Policy**: The final default policy is a uniform random policy. Each action is chosen randomly, with the vehicle choosing from a discrete set of depths and thrusting in one of the four cardinal directions.

- **Monte Carlo Tree Search (MCTS)**: MCTS is a natural extension of the Rand Policy, iteratively improving the policy over time. This state-of-the-art approach is outlined below.

3.3.1 Monte Carlo Tree Search

Instead of approximately solving the optimization problem by solving for the best action sequences, consider instead the task of finding a stochastic policy, which maps states in the ocean to distributions over control actions to maximize the total reward. This is similar to finding policies for Markov Decision Processes but has a different reward structure. Techniques for solving MDPs that rely upon value functions, such as dynamic programming [80] or reinforcement learning approaches [100], are not suitable. We instead focus on techniques based on Monte Carlo rollouts to evaluate the efficacy of a given policy.

One technique to find the best rollout is to start with a random stochastic policy, and, over many crude Monte Carlo rollouts, store the rollout with the highest reward. However, the optimal reward is a low probability event, and it will take prohibitively many rollouts to find a decent solution. Monte Carlo Tree Search (MCTS), on the other hand, combines multi-armed bandit techniques with graph based search to efficiently guide Monte Carlo rollouts to maximize expected reward, and is often used in playing adversarial games such as Go [16]. MCTS is readily extensible to MDP-like problems [42, 8]. When implementing MCTS, we use Upper Confidence Bound (UCB1) for the selection procedure [42], shown here:

$$a^* = \underset{a \in A}{\operatorname{argmax}} \left(\bar{a}_j + C\sqrt{\frac{\ln(n)}{n_j}} \right),$$

where a^* is the selected action, A are all possible actions from a state, $\bar{a}_j$ is the average reward from action a_j, C is a weighting parameter, n is the number of times this state has been visited, and n_j is the number of times action a_j has been selected. We also use a graph-based structure to store search nodes to save on memory requirements [8]. Use of UCB1 as a selection algorithm requires that rewards be upper bounded by a constant, which is unknown. The algorithm stores the maximum reward discovered and scales all rewards by this constant at each selection phase. However, the selection algorithm still requires experimental tuning of the parameter C which establishes tradeoffs between exploration and exploitation.

One can generate a discrete stochastic ocean motion model by selecting a discrete set of actions (depth and thrust velocities) and dividing the ocean into rectangular cells. Placing particles evenly spaced in each cell, one can track the motion of the particles according to the control action and ocean dynamics approximating the discrete transition probabilities of the system. While discrete rollouts may not be feasible trajectories, they approximate the continuous system's performance well enough, and the discrete stochastic policies can be used to control the continuous system, generating feasible paths.

3.4 SGA Simulation Results

To evaluate the performance of SGA, we used a NCOM model of the Gulf of Mexico. This model gives an ocean current forecast at a two kilometer resolution, every three hours for a week (168 hours) as a vector field of ocean currents. In this work, we assume that the vehicle can maintain its depth and so assume that the vertical current is zero. Additionally, we assume that the vehicle can provide a maximum thrust of 0.1 meters per second. This NCOM model also contains ocean temperature predictions at the same 2 km resolution. We use this temperature to generate a simulated information field over the ocean (see Figure 4.2) which can correlate with ocean phenomena such as algae blooms. To do this we assume that there is some desired temperature T_0 and use the equation:

$$I(x) = \begin{cases} e^{-a_p(T-T_0)} & \text{if } T \geq T_0 \\ e^{-a_n(T_0-T)} & \text{if } T < T_0 \end{cases}$$

where a_p is a scale factor for the positive case and a_n is a scale factor for the negative case. We calculate $T_0, a_p,$ and a_n so that 20 percent of the information lies above a threshold b, which was selected to be 0.5.

To generate starting regions for the vehicles we split the gulf into seven different regions, three degrees of latitude and longitude on a side. Three of these regions were used for parameter tuning, and four were used for testing our algorithms. The four regions used for testing are shown in Figure 4.2. In each of these regions we selected a square roughly 20 kilometers a side in the center as the possible starting location area.

From the 3 regions used for parameter tuning, we selected the maximum number of iterations as 50, a $\sigma_d = 100$ (represented in meters), $\sigma_t = 5$ (represented in hours), and a $\sigma_v = 0.02$ (represented in meters per second). Additionally, we found the number of noisy paths $K = 20$ to provide good results. A relatively small number of noisy paths gives better results by allowing the information from more informative paths to more significantly influence the gradient. Due to the randomness inherent in the gradient estimations of the information function, we optimized each action sequence five times and selected the highest scoring action sequence from those.

For tuning MCTS exploration/exploitation parameter C, we tested various values of $C \in [10^{-3}, 10^1]$ over 21 logarithmically spaced points in the training dataset. Note that MCTS's performance is sensitive to selection of C and the best vs. worst performance varied by 20% over the parameter space. Setting $C = 0.025$ yielded the best performance and was used in testing.

To test the multi-vehicle coordination aspect of the algorithm, we used teams of size 10. Inside the $\sim$20km square starting regions we generate four different locations with two locations containing four vehicles each and two locations containing one vehicle each. This was done to ensure that there was ample opportunity for the vehicles to interact while allowing the vehicles to start slightly dispersed. When all the vehicles started from the same location, there was not enough time over the week deployment to see diversity in paths.

The results from our tests can be seen in Figures 3.2 and 3.3 and Table 3.1. These show the percent improvement offered by SGA with respect to the other methods. Unsurprisingly, a default policy of having all vehicles stay at the same depth performs poorly, especially when some vehicles are started at the same location. Our optimized solution is able to offer a 104% improvement on average and can offer much larger improvements in environments with larger differences in ocean current magnitudes and directions.

A default policy of equally distributing the vehicles through the water column performs reasonably well. During testing, it was observed that this policy allows the vehicles to spread out spatially reasonably well but did not allow the team to prioritize information rich sections of the ocean. Our optimized solution offered an average of 61.00% improvement over equally distributing the vehicles.

A greedy depth selection method is effective at gathering information in these environments. However, our approach is still able to offer an average improvement of

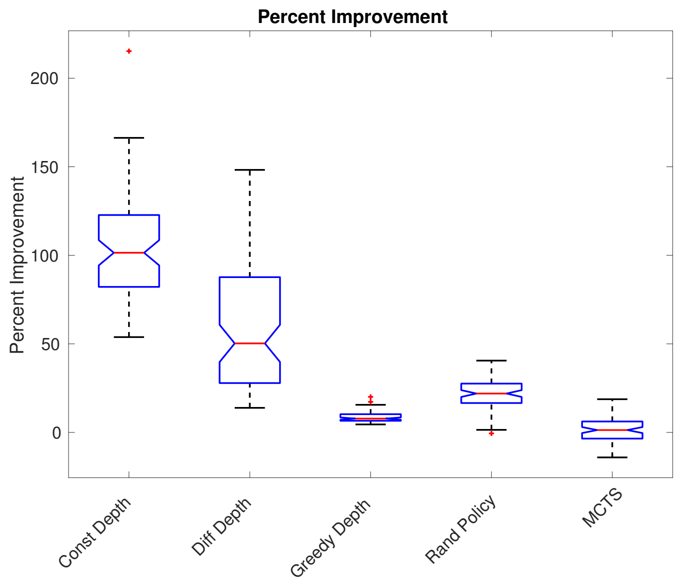


Figure 3.2: Percent improvement demonstrated by SGA with respect to all comparison algorithms. We are able to offer a large improvement over the simple baselines. MCTS is able to perform comparably to our algorithm, however it requires approximately 5.2x the computation. See Figure 3.3 for a more detailed look at the results for the final three methods.

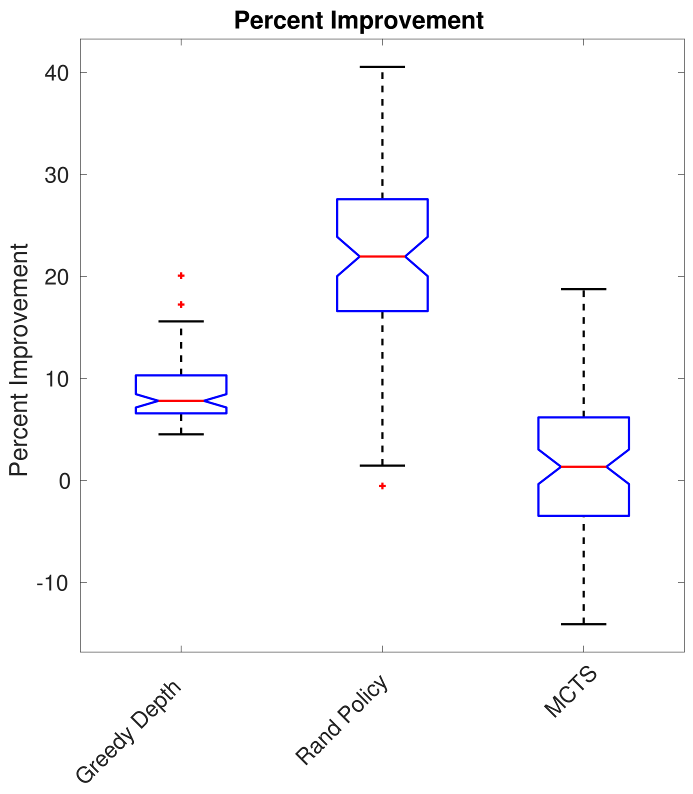


Figure 3.3: Percent improvement demonstrated by SGA over the three best performing comparison algorithms from Figure 3.2. SGA is able to improve over greedy depth selection (8.63%), Random Policy (21.58%) and MCTS (1.50%).

8.63% over this greedy selection method. SGA is able to intelligently consider interactions between different actions within the sequence. This would be difficult for the greedy planner to consider as the problem grows exponentially in the number of lookahead steps. While greedy depth selection is able to compute these plans very quickly (on the order of a second), there is a significant financial trade-off for any performance improvement. Daily deployment costs from a crewed surface ship can be upward of $30,000 dollars (see [84]) while a minute of cloud computing time costs less than $1. Even if there is only one day of deployment for a 30 day mission using an autonomous vehicle, the amortized cost is still $1000 per day for the mission. Put another way, to gather the same amount of information required for a mission, 10 vehicles using the proposed approach gather the same amount of information on average that about 10.9 vehicles gather using the greedy method. The effective deployment costs of each mission therefore increases on average by 9% ($90 per day) which dramatically outweighs the additional 1-2 minutes of compute time ($1-$2).

Using a random policy results in surprisingly good performance. The random policy has an advantage over some of the other simple policies by being able to choose a different depth at each action. However, this selection is not made in an intelligent way. Thus, the greedy depth selection, despite being only able to select a single depth for a vehicle, outperforms the random policy. Our optimized solution is able to offer an average of 21.58% improvement.

The comparison method most competitive with SGA is MCTS, where we are only able to offer a 1.5% improvement. However, this is done with an average computational time of 68.09 seconds in comparison with the 354.21 seconds average computational time required by MCTS, which can be seen in Figure 3.4. This represents a 5.2-fold increase in computational time required by MCTS in comparison with SGA. Future work will look at the effects of parallelizing the sampling process has on computational time.

Additionally, this computational reduction represents a significant savings for traditional vehicle operations where surface times are typically minimized to reduce the risk of collision with shipping [75]. From [3], we can see that a typical surfacing interval for a data-intensive task is approximately 15 minutes. As such, the planning time for MCTS represents approximately 33 percent increase in surface time while SGA represents approximately a 7 percent increase in surfacing time.

An example of the paths found by MCTS (green, x's) and SGA (magenta, o's) is shown in Figure 3.5. SGA is able to plan trajectories which collect information from the small field seen at the end of the path at approximately 21.85 latitude and 265.85 longitude while MCTS has trouble finding this extra information through its random rollouts. The vehicles are able to intelligently balance between spreading out to find information and grouping up to exploit high information regions.

One interesting observation from our testing is that this percent improvement is highly environment dependent. In two environments (1 and 4), SGA offers around the expected 1.5% improvement. However, in environment 2, SGA offers about a 10% improvement over MCTS, while in environment 3, MCTS offers a 3% improvement over the optimizer. Future work will look more at identifying what features of these environments lead to these disparities and what situations each method is better suited for.

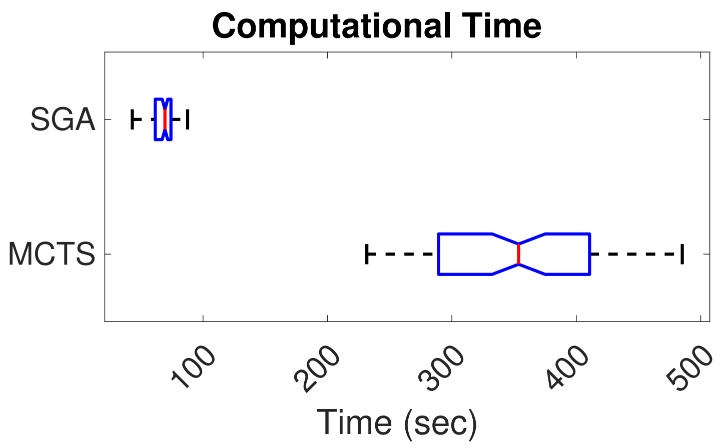


Figure 3.4: Computational time taken by the MCTS and SGA. Note that SGA takes 68.09 second on average while MCTS takes 354.21 seconds which represents a 5.2 fold increase.

Table 3.1: Percent Improvement by SGA versus competing algorithms

Const Depth	Diff Depth	Greedy Depth	Rand Policy	MCTS
104.10	61.00	8.63	21.58	1.50

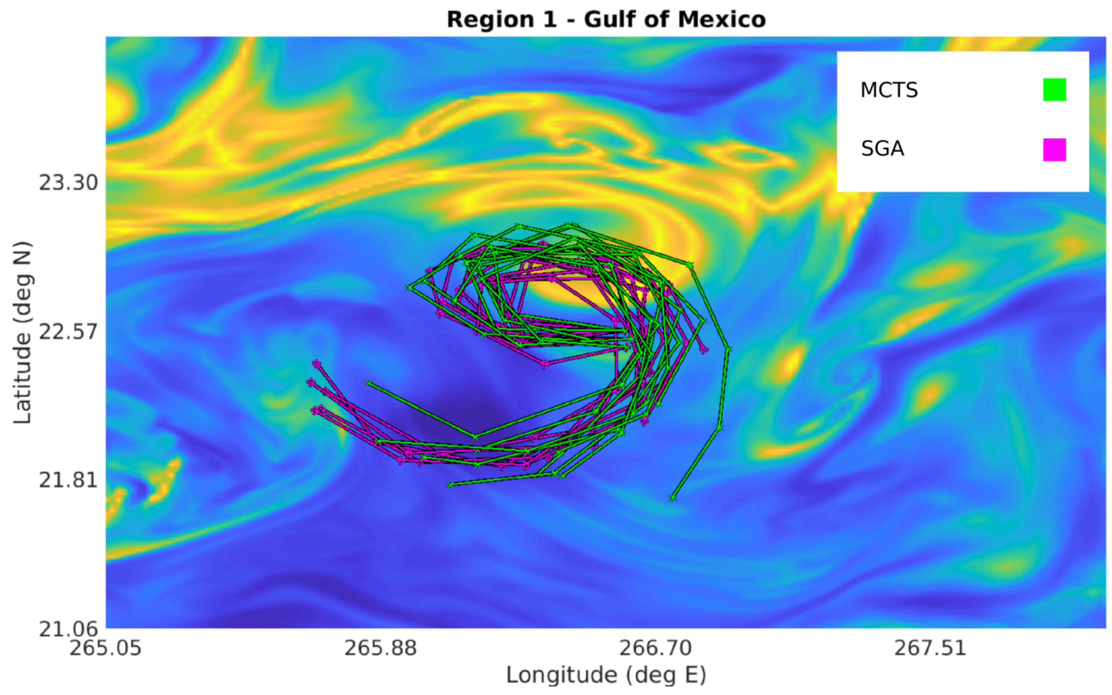


Figure 3.5: Paths from MCTS and SGA for one deployment in Region 1 from Figure 4.2. MCTS is in green and x's and SGA is in magenta and o's. Lighter indicates more information and vehicles start close to the center of the figure. SGA is able to utilize the ocean currents to find the information at the end of the path (left side of the figure) while MCTS is unable to find this path using its random rollouts.

3.5 Conclusion

In this chapter we showed that by using a novel action sequence representation and our SGA algorithm, we are able to improve upon a greedy baseline. We also compared to a computationally intensive Monte Carlo Tree Search Method, which performs comparably with SGA, but requires 5.2x the amount of computation on average. By approximating the gradient with random rollouts we were able to efficiently search the space and quickly improve the path. We presented results for a NCOM model, which closely approximates what the vehicle would experience in the ocean.

When a robot's motion is dominated by the environmental disturbances, planning in the action-space of the robot allows for a very natural representation of disturbances into the problem and therefore more realizable paths. However, when vehicles have a larger amount of actuation relative to the environmental disturbances, the actions space might be prohibitively large in both size and number of actions to allow for computationally efficient realizable path planning. In the next chapter we will discuss our contributions developing algorithms for vehicles with moderate relative levels of actuation planning in strong disturbances.

Another key assumption made in this chapter was that the disturbances were fully known. While there are a number of high quality ocean prediction models, these do not always exists for all disturbances and even in environments where these high quality model exists, these models can contain significant errors. One advantage that robots have is that they are physically present in the environment, and as such they can gather information about the environment during execution. In the next chapter we will discuss our contributions developing replanning frameworks to account for partially known disturbances.

Chapter 4: Energy-Efficient Stochastic Trajectory Optimization

In this chapter we present a realizable path planning algorithm, Energy-Efficient Stochastic Trajectory Optimization (EESTO), which allows moderate actuation vehicles to plan energy-efficient paths in strong disturbances. We do this by removing assumptions on the temporal spacing of waypoints in previous planning techniques, which allows for a larger range of cost functions to be considered and the effects of disturbances to be accounted for. Additionally, in contrast to the previous chapter, the environmental disturbance predictions are not assumed to be correct and we present a method for dynamically replanning path during execution. This method is able to leverage the computational efficiency of EESTO to generate plans online as information about the environment is gathered through execution allowing the robot to plan more realizable paths. Versions of this work have been published in [32] and [33].

In this work we generalize previous trajectory optimization techniques, such as STOMP [36], to perform energy-efficient path planning. Our algorithm, Energy-Efficient Stochastic Trajectory Optimization (EESTO), removes the assumption that the trajectory waypoints are equally spaced in time. Removing this assumption allows the algorithm to use a variety of different cost functions and improves upon the state of the art in energy-efficient ocean current path planning.

We show that naïvely replanning by always executing the newly planned path causes the vehicle to execute higher energy cost paths. We introduce five path comparison metrics, three of which can be leveraged to plan paths more energy-efficient than those generated from simply planning on the forecasted ocean currents.

For this chapter, it may be helpful to review the background on search (Section 2.1.1) and optimization (Section 2.1.2) based planning as well as energy efficient planning (Section 2.3.3).

4.1 EESTO Problem Formulation

EESTO builds upon previous trajectory optimization algorithms, such as STOMP [36], and follows the notation used there. We define our motion planning problem as finding a path from a given starting location to a desired goal location. The trajectory $\tilde{\boldsymbol{\theta}}$ is discretized into N waypoints, $\tilde{\boldsymbol{\theta}}_1, \ldots, \tilde{\boldsymbol{\theta}}_N$. EESTO then seeks to iteratively optimize the equation

$$\boldsymbol{\theta}^* = \min_{\tilde{\boldsymbol{\theta}}} \mathbb{E}\left[\sum_{i=1}^{N} q(\tilde{\boldsymbol{\theta}}_i) + \frac{1}{2}\tilde{\boldsymbol{\theta}}^\intercal \mathbf{R}\tilde{\boldsymbol{\theta}}\right], \tag{4.1}$$

where $\tilde{\boldsymbol{\theta}}$ represents a noisy trajectory with mean $\boldsymbol{\theta}$ and variance σ, $q(\tilde{\boldsymbol{\theta}}_i)$ is a state dependent cost function, $\mathbf{R}$ is a matrix where $\mathbf{R} = \mathbf{A}^\intercal\mathbf{A}$, and $\mathbf{A}$ is a constant finite differencing matrix such that $\ddot{\boldsymbol{\theta}} = \mathbf{A}\boldsymbol{\theta}$. The results of this definition of $\mathbf{R}$ is that the second term in (4.1) approximates the sum of squared accelerations and represents the control costs needed to perform the trajectory.

In previous work [36] the matrix $\mathbf{A}$ is constant because the step size considered in the finite differencing is kept constant. However, in our work we generalize this formulation to allow the step size to change, removing the assumption that the waypoints are equally spaced in time. To do this, we decompose matrix $\mathbf{A}$ into a constant center finite differencing matrix, $\mathbf{D}$, and a varying matrix, $\mathbf{T}$, which is the step sizes and is updated every iteration. This decomposition is

$$\mathbf{A} = \mathbf{TD} = \begin{bmatrix} 1/t_1^2 & 0 & \cdots \\ 0 & 1/t_2^2 & \\ \vdots & & \ddots \end{bmatrix} \begin{bmatrix} -2 & 1 & 0 & \\ 1 & -2 & 1 & \cdots \\ 0 & 1 & -2 & \\ & \vdots & & \ddots \end{bmatrix}.$$

Each iteration, t_i is updated to the average of the travel times from the waypoints $\boldsymbol{\theta}_{i-1}$ to $\boldsymbol{\theta}_i$ and from $\boldsymbol{\theta}_i$ to $\boldsymbol{\theta}_{i+1}$. We generalize the normal waypoint formulation used in STOMP to accommodate this extra information. To do this, we augment each waypoint with the corresponding travel time, t_i, which is the travel time to the next waypoint, θ_{i+1}. In the 2D case, with $\boldsymbol{\theta}_i = (x_i, y_i)$ originally, this gives $\boldsymbol{\theta}_i = (x_i, y_i, t_i)$. It is important to note that the temporal component, t_i is already incorporated into $\mathbf{R}$, and so when calculating the second term of (4.1), $\tilde{\boldsymbol{\theta}}^{\mathsf{T}}\mathbf{R}\tilde{\boldsymbol{\theta}}$, only the spatial dimensions are used.

The optimization problem presented in (4.1) is solved by approximating the gradient of the cost function $q(\tilde{\boldsymbol{\theta}}_i)$ with a weighted combination of K explored noisy paths in the same manner as [36] where a more detailed description is contained. These noisy paths are generated by adding sampled perturbations ϵ_k to the current path estimate so that noisy path $\hat{\boldsymbol{\theta}}_k = \boldsymbol{\theta} + \epsilon_k$. These perturbations are sampled from a zero-mean normal distribution $\epsilon_k = \mathcal{N}(0, \mathbf{R}^{-1})$. We sample from $\mathbf{R}^{-1}$ so that perturbations produce smooth deformations; however doing so introduces a large computational overhead each iteration due to the expensive matrix inversion calculation not present in previous work.

EESTO calculates this inversion in a computationally tractable way by factoring $\mathbf{R}^{-1}$ as:

$$\mathbf{R}^{-1} = (\mathbf{A}^{\intercal}\mathbf{A})^{-1} = \mathbf{D}^{-1}(\mathbf{T}^{2})^{-1}(\mathbf{D}^{\intercal})^{-1}. \tag{4.2}$$

Both $\mathbf{D}^{-1}$ and $(\mathbf{D}^{\intercal})^{-1}$ can be precomputed, and $\mathbf{T}^2$ is a diagonal matrix whose inverse can be quickly calculated. This step is fundamentally different from the factorization of $\mathbf{R}$ found in the publicly available implementation of STOMP, since in EESTO we are forced to recalculate $\mathbf{R}^{-1}$ every iteration to account for the temporal variation of waypoints. We rely on our factorization to compute this inversion efficiently. In contrast, STOMP's Lower-Upper (LU) factorization of $\mathbf{R}$ is used to speed up rollout calculations, and is not strictly necessary to the functionality of the algorithm.

To sample the deformations to the travel time between waypoints, a similar strategy is employed. We sample these deformations, ϵ_t, from $\mathcal{N}(0, \mathbf{T}^{-1})$. After scaling these perturbations with a user selected scaling factor, which is equivalent to choice of a step size in gradient descent methods, the time perturbations are added to the spacial perturbations ϵ_k so that $\epsilon_k^{'} = (\epsilon_k, \epsilon_t)$.

4.1.1 Cost Function

By removing the assumption that waypoints are equally spaced in time, EESTO can perform optimization over a range of cost functions. In this work we define a cost function that seeks to perform energy-efficient planning. Our cost function balances travel time and energy expenditure by rewarding both energy-efficient paths and traveling at speeds close to or greater than the vehicle's maximum velocity by riding the ocean currents.

These two goals often oppose each other, as traveling faster generally requires more energy. Both are included into the cost function so that the trade-off between these two factors can be explicitly acknowledged and defined. The cost function is defined as:

$$q(\boldsymbol{\theta}_i) = w_0 C_e(\boldsymbol{\theta}_i) + w_1 C_s(\boldsymbol{\theta}_i) + C_o(\boldsymbol{\theta}_i), \tag{4.3}$$

where C_e is the energy cost, C_s is the speed cost, C_o is the obstacle cost, and w_0, w_1 are weighting factors between the different terms of the cost function.

4.1.1.1 Energy Cost

To calculate the energy cost of a single waypoint, we compute the difference in energy expenditure between the path through $\boldsymbol{\theta}_i$ from $\boldsymbol{\theta}_{i-1}$ to $\boldsymbol{\theta}_{i+1}$, E_{with}, and the direct path from $\boldsymbol{\theta}_{i-1}$ to $\boldsymbol{\theta}_{i+1}$ which bypasses $\boldsymbol{\theta}_i$, $E_{without}$. This formulation is inspired by difference learning [1] from the multi-agent literature. The energy cost is calculated as

$$C_e(\boldsymbol{\theta}_i) = \exp(E_{with}(\boldsymbol{\theta}_i) - E_{without}(\boldsymbol{\theta}_i)). \tag{4.4}$$

The energy expenditure of these path segments is calculated in the same manner as in [112], where the drag force is assumed to dominate the inertial forces. As such, the energy expenditure is

$$E = c_d V_r^3 t, \tag{4.5}$$

where c_d is the vehicles drag coefficient, V_r is the required velocity for the motors to provide, and t is the travel time for the relevant section of the path.

4.1.1.2 Speed Cost

Using just the energy cost, the vehicle will seek to float with the ocean current to the goal rather than attempting to reach the goal destination promptly. To penalize this behavior we define a speed cost function:

$$C_s(\boldsymbol{\theta}_i) = \begin{cases} 0, & \text{if } V_r \leq V_{max} \text{ and} \\ & \quad V_{abs} > V_{max} \\ \exp(V_{max} - V_r), & \text{if } V_r \leq V_{max} \\ l + (V_r - V_{max})l^2, & \text{if } V_r > V_{max} \end{cases}, \tag{4.6}$$

where V_{max} is the maximum velocity the motors can provide, V_{abs} is the absolute velocity that the vehicle is traveling, and l is an arbitrarily large number selected to introduce a step cost when the motors are required to provide a speed that they cannot achieve. In this work, l was set to 1000.

This formulation encourages the vehicle to travel close to its maximum speed and to utilize the ocean currents to propel the vehicle forward. We use a speed-based cost function, as opposed to a travel-time based one, for two reasons. First, since the waypoints are not evenly spaced temporally, travel time is a less meaningful metric for comparing two waypoints, as it will tend to select the closer one. Second, a speed-based metric can explicitly penalize paths which require the vehicle to travel at speeds greater than the motors can attain, even with assistance from disturbances.

4.1.1.3 Obstacle Cost

Our obstacle cost formulation is a simplified form of the normal signed distance function used in previous work [36] [112]. Due to the open nature of most ocean environments, the obstacles' cost is simplified from a function of distance inside the obstacle to just a step cost. As such, C_o is calculated as:

$$C_o(\boldsymbol{\theta}_i) = \begin{cases} 0, & \text{if } \boldsymbol{\theta}_i \in \chi_{free} \\ B, & \text{if } \boldsymbol{\theta}_i \notin \chi_{free} \end{cases}, \tag{4.7}$$

where χ_{free} represents the free space of the environment, and B is some arbitrarily large number selected to introduce a large step cost when inside an obstacle.

4.2 Replanning

One of the major challenges of working in underwater environments is the high level of uncertainty present. While there are many systems that provide estimates of ocean currents [91], these ocean current estimations are both approximate and tend to be quite coarse. To manage this uncertainty, we have developed a replanning method that learns about the world as a trajectory is executed and replans more energy-efficient paths using this information. It is important to note that we do not plan paths to deliberately gather additional information about the environment. Our replanning framework uses only the information the vehicle encounters as it executes its planned trajectory. Extending this into the domain of informative path planning [94] is beyond the scope of this work presented in this chapter but does present an interesting avenue for future work.

We assume that there is a true function, $f(\mathbf{x})$, which gives the difference between the forecast disturbance, $FD(\mathbf{x})$, and the true disturbance, $TD(\mathbf{x})$, at a location $\mathbf{x} \in \mathbb{R}^2$ (i.e. $f(\mathbf{x}) = TD(\mathbf{x}) - FD(\mathbf{x})$). We approximate $f(\mathbf{x})$ using a zero-mean Gaussian Process (GP):

$$f(\mathbf{x}) \sim \mathcal{GP}(0, k(\mathbf{x}, \mathbf{x}')),$$

where k is the squared exponential radial basis kernel. This kernel was chosen because it captures the expected correlation behavior. The hyperparameters for this function were calculated offline prior to execution on a different ROMS data set and not retrained during path execution. We use a GP because it can handle noisy observations and provides both smooth approximations of the underlying function as well as variance estimates. A more detailed discussion of GPs is beyond the scope of this work but interested readers are directed to [82].

4.2.1 Replanning Method

Our replanning method operates by regularly replanning new potential paths and comparing these paths with the existing path. When a new path is planned, the equation:

$$P[EC(\boldsymbol{\theta}_{re}) < EC(\boldsymbol{\theta}_{ex})] > \lambda \tag{4.8}$$

is evaluated, where $EC(\cdot)$ is a function that calculates the energy cost of a path, $\boldsymbol{\theta}_{re}$ is the replanned path, $\boldsymbol{\theta}_{ex}$ is the existing path, and λ is a user selected threshold for the comparison. If this probability is greater than λ, then the vehicle begins to execute the newly planned path; if not, then the vehicle continues to execute the existing plan.

The replanning method we developed can be seen in Algorithm 2. An initial path, $\boldsymbol{\theta}_{curr}$, is planned on the forecast of the disturbances (in this application a vector field of ocean currents) which represents the best path from the existing information. Then, while the vehicle has not reached the end of the path, the following steps are performed. First, the vehicle moves to the next waypoint and adds the difference between the actual measured disturbance and the forecast disturbance along the executed path to the data set being modeled by the GP.

Next, with this new information about the world, the vehicle replans a path on the forecast with the predictions from the GP regression added to it. Lastly, we want to compute (4.8) for this newly planned path. However, (4.8) is difficult to compute analytically because the variance of the energy cost passes through a number of complex functions for which there are no analytical solutions. Thus, we developed a number of metrics to approximate the comparison which are detailed below.

Algorithm 2 Replanning Framework for Energy-Efficient Trajectories

```
1: $\boldsymbol{\theta}_{curr} \leftarrow EESTO(forecast)$
2: **while** $\neg end(\boldsymbol{\theta}_{curr})$ **do**
3:     $Move\_to\_next\_waypoint(\boldsymbol{\theta}_{curr})$
4:     $disturbance \leftarrow Measure\_Disturbances()$
5:     $differences.add(location, disturbance)$
6:     $GP \leftarrow GP.model(differences)$
7:     $estimate \leftarrow Model\_Disturbances(forecast, GP)$
8:     $\boldsymbol{\theta}_{re} \leftarrow EESTO(estimate)$
9:     **if** $P[EC(\boldsymbol{\theta}_{re}) < EC(\boldsymbol{\theta}_{ex})] > \lambda$ **then**
10:        $\boldsymbol{\theta}_{curr} \leftarrow \boldsymbol{\theta}_{re}$
```

4.2.2 Path Comparison Metrics

4.2.2.1 Always Accept

The first and simplest metric of path comparison is to always assume that the newly generated path will be more efficient than the old path. In this metric whenever a new path is generated it is also accepted. This is equivalent to assuming that $P[EC(\boldsymbol{\theta}_{re}) < EC(\boldsymbol{\theta}_{curr})] = 1$. This metric suffers from two distinct problems. First, the planning method being used, EESTO, is a stochastic planning method and as such has the possibility of producing a poor path. Second, the information is noisy and causes the algorithm to replan on the noise rather than the true difference function the vehicle is trying to approximate.

4.2.2.2 Difference

A second, and more informed comparison metric, is to accept a new path only if it is sufficiently different from the currently accepted trajectory. The intuition behind this comparison metric is that similar paths will have similar energy costs, and switching to a newly planned path will be more energy-efficient only if a sufficiently different path is found. In this work, the difference between two trajectories is calculated by taking the maximum Hellinger Distance (HD) [78] between the ocean current distributions at corresponding waypoints in each trajectory and comparing this calculated maximum difference to a user specified threshold, which corresponds to λ above. In this work, a relatively small threshold of $\lambda = .05$ was found to give the best results. The Hellinger Distance was selected because it is a commonly used measurement for the distance between two distributions and provides a bounded metric on the distance between two probability distributions. This is equivalent to assuming that $P[EC(\boldsymbol{\theta}_{re}) < EC(\boldsymbol{\theta}_{ex})] \propto HD(\boldsymbol{\theta}_{re}, \boldsymbol{\theta}_{ex})$.

4.2.2.3 Projected

One simple approximation of the probability $P[EC(\theta_{re}) < EC(\theta_{ex})]$ is to compare the projected energy cost that each path will require. Each path's projected energy cost is calculated on the estimated disturbances found by adding the outputs from the GP regression to the forecast to give an approximation of the energy cost of θ_{re} and θ_{ex}. If θ_{re} has a lower projected energy cost, it replaces θ_{ex}. This reflects the assumption that $P > \lambda$ is true if $EC(\theta_{re}) > EC(\theta_{ex})$ and false otherwise. This metric can fail for two different reasons. First, the projected energy can be inaccurate when the uncertainty on the ocean currents is high. Second, because this metric only compares the energy cost, it can accept a new path that is similar to the old path and incur an increase in energy cost due to frequent shifting.

4.2.2.4 Combined

The combined metric uses a heuristic to approximate the probability that one path is more energy-efficient than another. If the newly planned path satisfies conditions for both the difference and projected comparison metrics, the newly planned path replaces the current path. The projected energy approximates the mean of the energy cost distribution while the difference is used to approximate the distance between the two distributions. In this metric λ is selected in the same manner as in the two combined methods. The projected energy cost of the new path must be less than the energy cost of the existing path and the calculated difference between the ocean currents along the two paths is greater than the λ value above.

4.2.2.5 Sampled

The last metric approximates the energy distributions by performing rollouts on sampled disturbance environments created by drawing samples from the GP and adding those sampled outputs to the forecast. In this work we found that performing 100 rollouts gave a good trade-off between approximating the distribution and computation time.

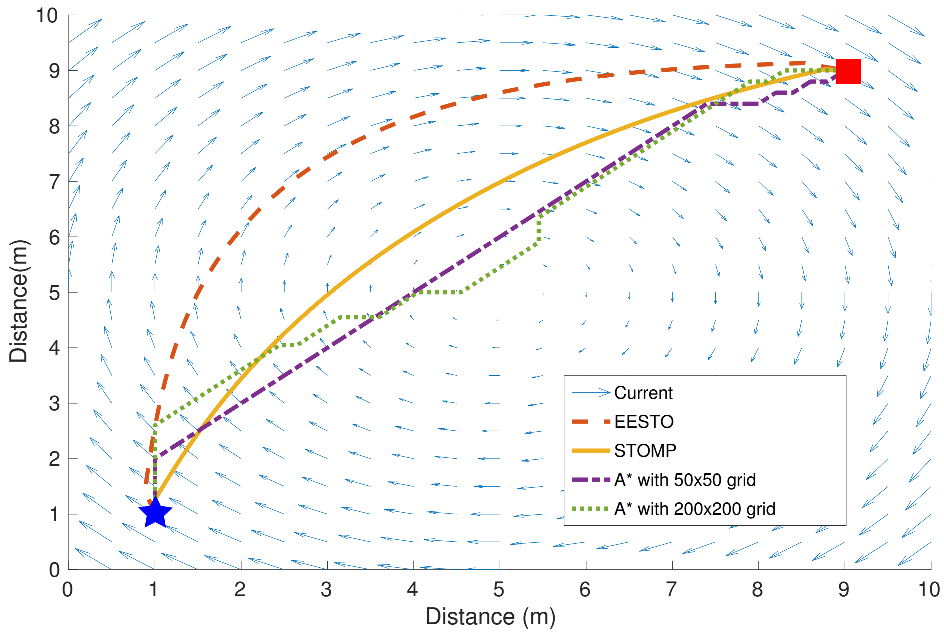
We make the simplifying assumption that the expected cost of the existing and replanned paths (EC_{ex}, EC_{new}, respectively) are normal distributions with calculated mean, μ, and variance, σ. Rather than computing $P[EC_{re} < EC_{ex}]$, we instead compute $P[(EC_{re} - EC_{ex}) < 0]$. Using this formulation, the z-score can be used as our decision threshold, which in this work corresponded to accepting a new path if a z-score of -1 or lower was computed.

4.3 EESTO Simulation Results

Results for both our trajectory optimization algorithm, EESTO, and the proposed replanning framework are presented in this section. In all instances of the EESTO algorithm we use $K = 30$ noisy paths. Note that EESTO scales linearly with K.

4.3.1 EESTO Without Replanning

We tested EESTO in both a synthetic ocean environment and one built from historical ROMS ocean current data taken from [14], which provides a large ocean current dataset for the Southern California coast. The synthetic ocean current environment can be seen in Fig. 4.1 (a). The ocean current velocities range from $0\ m/s$ at (5,5) to $1\ m/s$ along the edges, representing half of the vehicle's maximum speed of $2\ m/s$. Figure 4.1 (b) shows the results for 100 runs for both EESTO and STOMP as well as the energy cost for the path produced by a simple A* search with energy as the travel cost [52], [29], [46]. We compare to A* as representative of these discrete approaches to show the advantage



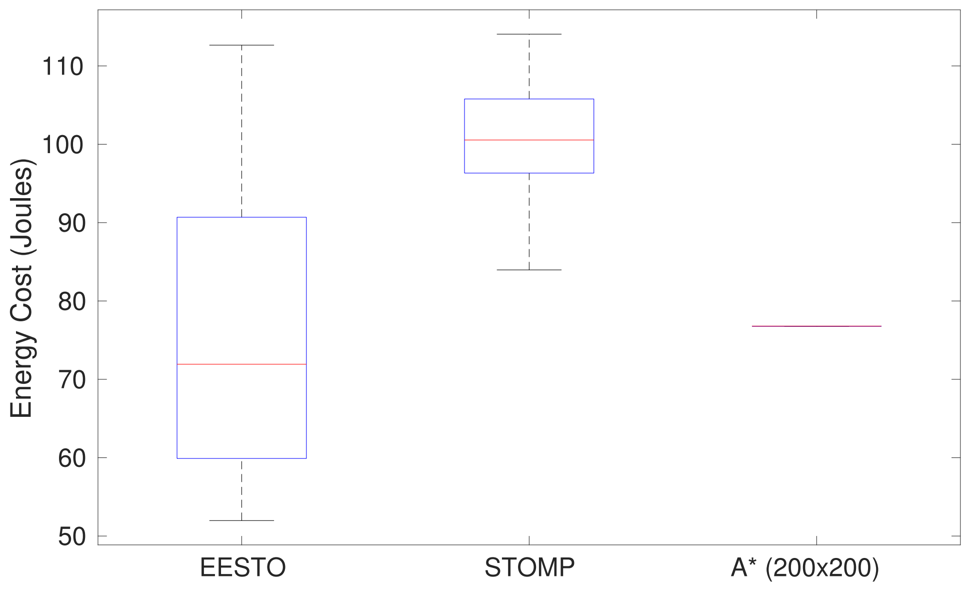


Figure 4.1: Shown above is the synthetic environment with paths starting at the blue star and ending at the red square. Shown below is the energy used by the paths produced by the three planning methods across 100 runs. EESTO is more computationally efficient than A*-based methods, meaning that multiple instance of EESTO can be run and the most energy-efficient path selected. This gives EESTO a high probability of planning a more energy-efficient path than A* in the same amount of time.

EESTO offers by allowing a tractable was of operating in continuous environments.

There are a number of observations to make from this test. First, while the approximated energy costs for A* and EESTO are quite close, the paths produced by EESTO (seen in Fig. 4.1 (a) starting at the blue star and ending at the red square) do not require sharp changes in direction. Due to finding smooth deformations that respect the kinematics of the vehicle, EESTO is able to produce paths that more accurately represent those that can be executed on an AUV. EESTO also produces more energy-efficient paths than STOMP using the same cost functions because EESTO can vary the time between waypoints giving it more freedom in the optimization. We note that increasing K does not significantly affect these results. While it can provide additional information, this additional information does not significantly improve the gradient estimate.

Second, the two A* paths are at different levels of discretization, and thus produce slightly different paths. As in other environments, A* is only optimal to its spatial resolution, but the added discretization of time amplifies this resolution dependence. Lastly, as can be seen in Table 4.1, EESTO's average run time is substantially lower than the time required for A*. This allows for multiple instances of the algorithm to be run and the lowest energy path selected when initially planning, making EESTO more likely to find a path that is lower energy than the one produced by A*.

To validate these results in a more realistic environment, EESTO was also used to plan a path through historical ocean current data from ROMS [14]. A 30 kilometer path was planned off the coast of California near the Channel Islands. The path evolution and the final path found can be seen in Fig. 4.2. The path in Fig. 4.2 represents an example path around the island produced by EESTO. The algorithm is able to both avoid the

Table 4.1: Calculation Times

	Calculation Time (sec)
EESTO	0.77
STOMP	0.2
A* 50x50	3.54
A* 200x200	348.33

island and correctly identify that the vehicle can leverage the stronger currents north of the island to use less energy when traveling than if it was to travel south of the island.

4.3.2 EESTO With Replanning

To test the replanning framework, we used a historical ROMS ocean current dataset from February 2016 [14]. We used a ROMS forecast as the forecast ocean current dataset and the corresponding ROMS nowcast as our simulated ground truth ocean current dataset. Results from 100 runs can be seen in Fig. 4.3. Our five replanning methods were compared with the best path from 20 runs of EESTO on both the nowcast and forecast which do not replan and represent planning with no additional information (forecast) and planning with perfect information (nowcast). This was done to remove some of the randomness from the results and allows planning on the forecast and nowcast to plan in a comparable amount of time to the replanning methods, which replan at every step along the trajectory.

As can be seen in Fig. 4.3, not all the replanning metrics end up helping the vehicle. The *always accept* and *difference* metrics preformed worse than planning on the forecast, giving an increase in mean energy cost of 50.00% and 55.88%, respectively.

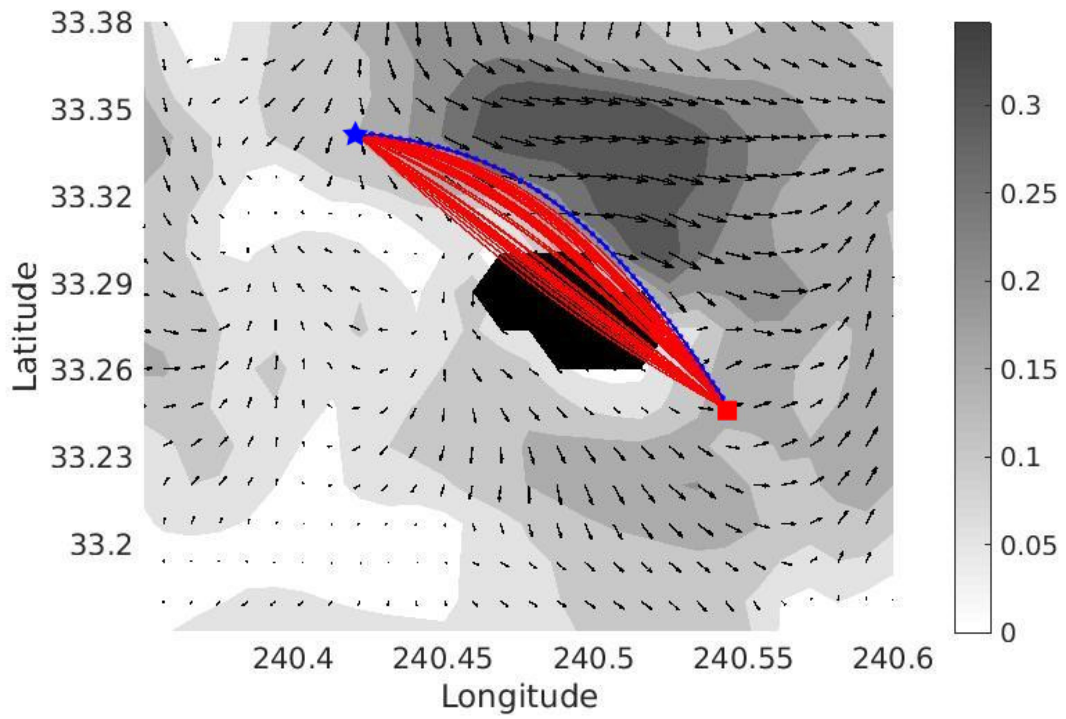
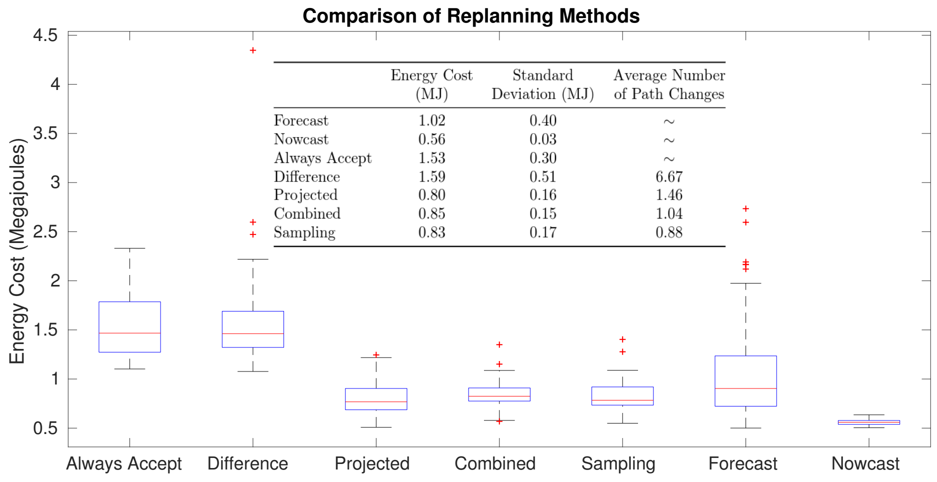


Figure 4.2: Paths planned by EESTO using historical ocean current data from January 21, 2013. Red paths are the solutions found at the end of each iteration. The final path is represented in blue. The path starts at the blue star and ends at the red square.



	Energy Cost (MJ)	Standard Deviation (MJ)	Average Number of Path Changes
Forecast	1.02	0.40	∼
Nowcast	0.56	0.03	∼
Always Accept	1.53	0.30	∼
Difference	1.59	0.51	6.67
Projected	0.80	0.16	1.46
Combined	0.85	0.15	1.04
Sampling	0.83	0.17	0.88

Figure 4.3: Energy cost using different replanning methods. The projected, combined and sampling method are able to improve the mean energy cost by 3.57, 6.43 and 8.57 percent respectively. Additionally, projected, combined and sampling are able to reduce the maximum outlier by over 25 percent. Note that both the Forecast and Nowcast methods do not replan and represent planning with no additional information (Forecast) and perfect information (Nowcast).

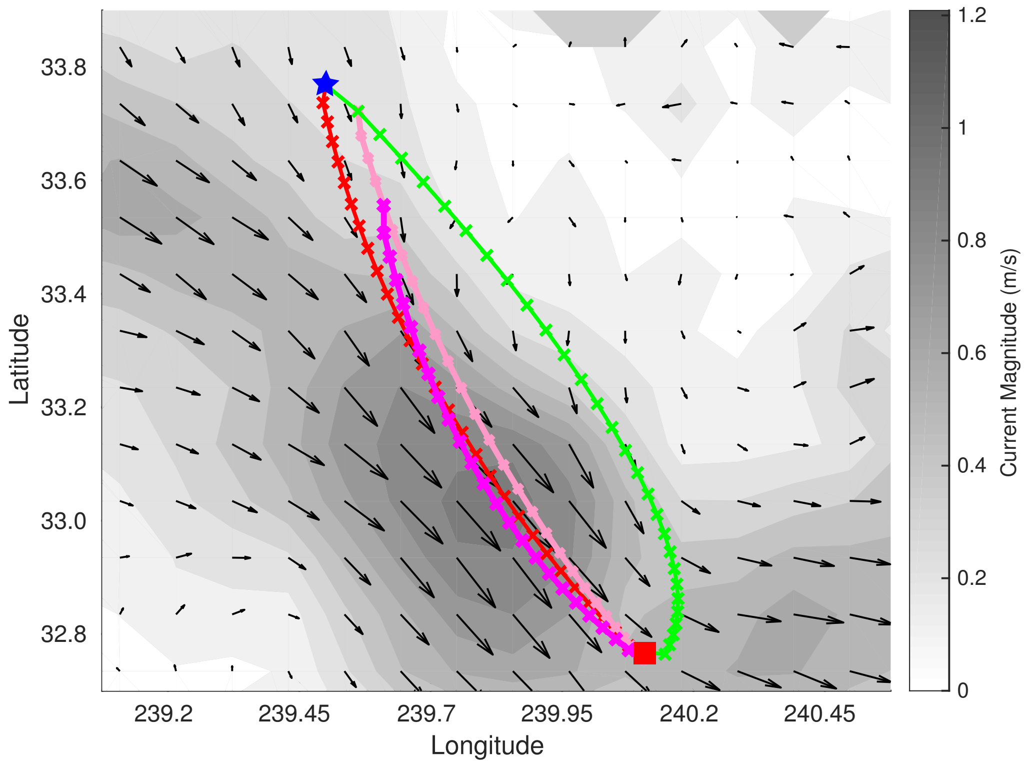


Figure 4.4: Paths planned using the replanning framework starting at the blue star and ending at the red square. The green path is planned on the forecast (1.74 MJ). The red path is planned on the nowcast (0.54 MJ). The two purple paths are the replanned paths with the darker purple path being the final path (0.86 MJ).

In the case of *always accept* the naïve assumption that more information produces more energy-efficient paths turns out to be false. The planner is instead planning on the noise in the estimates and accepts largely different paths at each step. This introduces a large energy cost as the vehicle changes back and forth between these different paths. In the case of the *difference* metric the fact that a path is different does not always correspond to the path being more energy-efficient due to the lack of information about distant ocean currents and the stochastic nature of the algorithm. Additionally, both metrics change paths more times than the other metrics which, as expected, leads to an increase in energy cost.

The final three replanning metrics do improve upon planning on the forecast. The *projected* metric gives a 21.57% improvement in the mean energy cost over planning on the forecast. This is the largest improvement, which switches and accepts new paths on average 0.42 more times than the *combined* metric and 0.58 more times than the *sampling* metric. This switching behavior is optimistic about the ocean current estimations and results in a larger variance on the energy cost.

Using the *combined* metric, the replanning framework is able to obtain a 16.67% improvement over planning on the forecast. The *combined* metric is able to reduce the deviation of the *projected* metric and performs comparably to the *sampling* metric, which gives a 18.63% improvement in mean energy cost over planning on the forecast. While the *sampling* metric gives a more accurate calculation of $P[EC(\theta_{re}) < EC(\theta_{curr})]$, it does so at an increased computational cost due to the large number of trajectory roll-outs required to approximate the energy cost distribution. When comparing paths, the *sampling* metric took 2.81 seconds on average to perform this calculation while the *combined* metric took 0.05 seconds on average.

All three of the final metrics also improve over planning on the forecast by decreasing the worst case cost. The maximum cost of planning on the forecast was 2.73 MJ, which is over twice the forecast mean cost and could be potentially disastrous for a vehicle running out of battery in the ocean. All three final metrics have a maximal cost below 2 MJ and give 54.21%, 50.55% and 48.72% reduction in maximum energy cost for the *projected*, *combined* and *sampling* metrics, respectively.

Figure 4.4 shows what a typical execution of this framework looks like. The vehicle initially is executing the green path but accepts a new path (pink) at the second waypoint. After executing that path for four waypoints the framework again accepts a new path (purple) and executes that path to the end. The red path shown represents planning if the current field is perfectly known. The path planned on the forecast took 1.74 MJ, the path planned with perfect information took 0.54 MJ and the replanned path took 0.86 MJ.

4.3.3 EESTO Field Trials

To further validate our simulated performance, we conducted a series of field trials to verify our algorithm on an autonomous vehicle. As an analog to ocean currents, we planned paths for an autonomous boat across the windy surface of a lake. These winds can significantly affect the ability of autonomous boats to travel efficiently by creating strong surface currents.

We preformed a series of field trials with the Lutra Prop autonomous boat from

Platypus LLC equipped with a Lithium-ion Polymer (LiPo) battery. The Lutra has a number of environmental monitoring sensors and has a maximum speed of approximately 4.7 m/s. Note that the average wind magnitude, as seen in Table 4.2, is approximately 65% of the vehicle's maximum speed. The Lutra system also has ROS integration, WiFi connectivity for connection to a base station, GPS, and basic waypoint following using the provided controller. In addition to the normal environment sensor suite, a Decagon Devices DS-2 sonic anemometer was mounted to the Lutra to allow the system to measure the winds across the surface of the lake. A Getac laptop running Ubuntu 14.04 with an Intel i7-4600M CPU @ 2.90GHz and 8 GB of RAM served as the base station where calculations were performed. These tests were preformed at West Kirk Park in Eugene, OR (Lat: 44.12, Lon: -123.30), (Figure 4.5).

We compared three different path planning methods. First, we considered a **naïve** method which ignores the wind and plans straight line paths from the start to the goal, which is the planning method currently used in practice. Second, we consider a **replanning** method which plans using EESTO and starts with no knowledge of the wind and starts by assuming zero wind on the surface. This method builds a GP and replans during execution using the combined method from above to compare paths. Lastly, a more accurate estimate of the wind was gathered across the surface of the lake and then EESTO planned paths using this information which simulates planning with full knowledge (**oracle**) and serves as an approximation of the ideal performance. We compare to the **naïve** method to illustrate the benefits of accounting for wind disturbances in planning. The comparison to the **oracle** method serves to show that the **replanning** method is able to perform competitively with a method given near-perfect information.

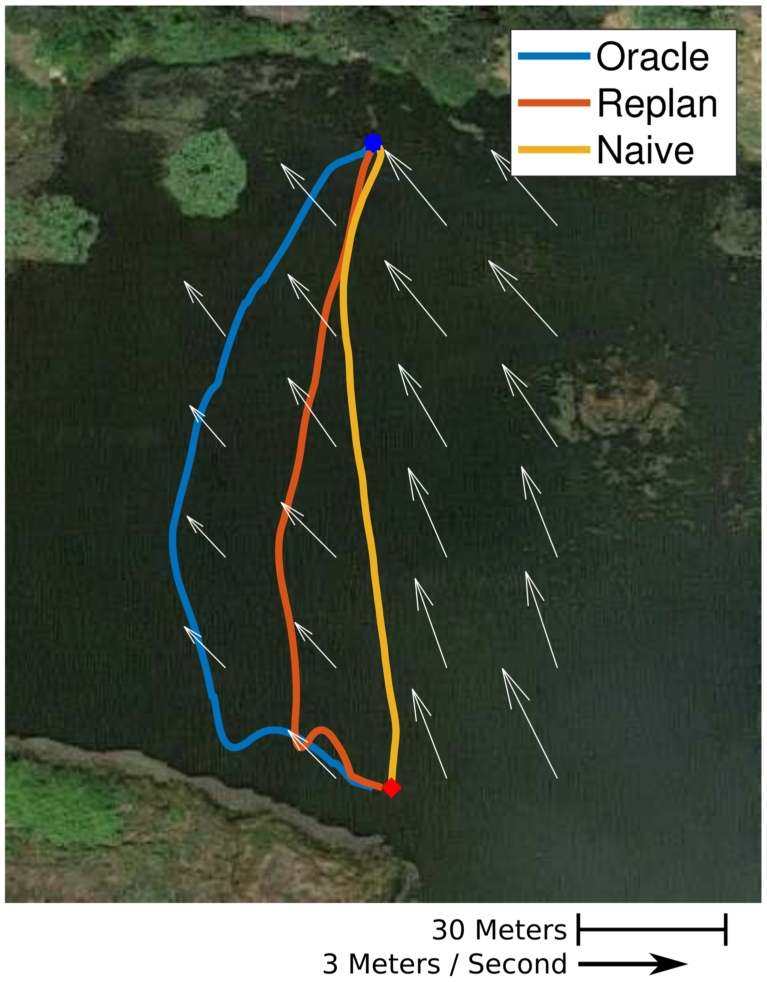


Figure 4.5: Representative planned paths on Kirk Lake starting from the blue circle (top) and ending at the red square (bottom). While the oracle method has access to the wind map seen in (a), the replanning method starts without any estimate of the wind and builds this map as it executes the path. Notice that the naïve method plans through the strong winds (white arrows) above the end point while our proposed methods come at the goal from the left where the wind is weaker.

Table 4.2: Average wind speed and direction for field trials

	Average Wind Magnitude (m/s)	Average Wind Direction (deg)
Naive	3.07	324.23
Replanning	2.96	335.65
Oracle	3.01	315.86

To evaluate the energy expenditure of the vehicle on different paths, we measured the voltage level of the vehicle's batteries before and after execution. While in general a measure of voltage drop does not provide an accurate measure of energy expenditure for a LiPo battery due to non-linearities at either end of the discharge curve, the relation is roughly linear in the middle of the discharge curve [43]. As such, tests were only performed on batteries that were measured to be in the middle of their discharge curve. In this case the voltage drop directly correlates with the energy used. To remove variations caused by slight changes in the wind speed and direction and to ensure a measurable voltage drop, each method was executed five times. As can be seen in Table 4.2, the difference in average wind speed was a maximum of 3%, and the difference in wind direction was a maximum of 6%. Both are relatively small and thus unlikely to affect the conclusions drawn.

Results can be seen in Table 4.3 and Figure 4.5. When compared with the naïve method, replanning starting with no information is able to produce a reduction in energy used by 13%, and planning with prior knowledge produces a 20% reduction in energy usage. Both the replanning and oracle methods plan paths that pass through the lighter winds to the left of the goal rather than approaching it from above where the wind is

Table 4.3: Voltage drop and percent improvement for field trials.

	Total Voltage Drop (5 Trials)	Average Voltage Drop	Percent Improvement
Naive	1.5	0.3	$\sim$
Replanning	1.3	0.26	13.3
Oracle	1.2	0.24	20.0

stronger (Figure 4.5). These results shows that our algorithm is robust enough for use in field environments and can work when limited amounts of information about the environment exist.

4.4 Conclusion

We have presented a generalized trajectory optimization algorithm which, by allowing the travel time between waypoints to vary, can consider a much larger set of cost functions. In this dissertation this increased capability was applied to the problem of finding energy-efficient paths through ocean current fields. The algorithm is validated in both simulated and real world scenarios, where it produced both more energy-efficient and more feasible paths in a computationally efficient method. Additionally, we presented a framework for replanning to account for the uncertainty present in the disturbances. Five different path comparison metrics were compared, and results were presented on historical ocean current data. We showed that always replanning does not produce more energy-efficient paths, but strategically accepting new paths based upon gathered information can improve energy efficiency when the disturbances are uncertain. Finally, we presented the results for field trials. From these trials we showed that our algorithm is capable of planning in environments with limited amounts of data about the environmental disturbances.

In this and the previous chapter, we have assumed that the interactions between a robot's plan and the low-level controller are negligible, which has been a reasonable assumption given the length of these paths and the relative amount of actuation these vehicles have. However, when considering vehicle with high relative actuation or vehicles performing much shorter paths, this interaction must be accounted for in realizable path planning and execution. In the next chapter, we present a waypoint judging policy method which reasons about this interaction between the planner and the low-level controller during execution. Additionally, we present a realization aware optimization framework which utilizes our previous work in stochastic optimization to account for the path that the robotic vehicle will realize using its onboard controller during planning.

Chapter 5: Reinforcement Learning for Integrating Control and Planning

In the previous two chapters, we assumed that the low-level controller onboard the robotic vehicle was sufficient to realize the planned path and that the difference between the planned path and executed path was small. However, this assumption does not always, or even typically, hold. In this chapter, we propose two methods for helping to reduce the distance between the planned and executed path and demonstrate their applicability to the information gathering problem. The two major contributions of this chapter are:

1. A Waypoint Judging Policy (WJP) which maps an egocentric polar state-space to decisions about whether to consider a waypoint achieved or not. By intelligently considering which waypoint to achieve, this policy allows an autonomous robot to more accurately track its planned path. This policy is computed using an off-policy Reinforcement Learning (RL) based framework with Upper Confidence Bound (UCB) exploration strategy which is shown to compute a policy which is better able to track the planned path than baseline training methods in this domain.
2. A Realization Aware Optimization (RAO) which considers the actual amount of information that the robot will gather during execution. By using stochastic optimization and Monte Carlo (MC) simulations, this framework is able to estimate the gradient of the actual information gathered by the robot and produce paths

where the amount of information gathered is closer to that reported at planning time.

In addition to these contributions, we also present results from a series of field trials at a lake in Corvallis, Oregon which demonstrate the ability of the learned policy to easily transfer from simulation to the real world with no training on the vehicle and of RAO alongside the WJP to improve the realizability of the planned paths for the information gathering task. Versions of this work are under review in [31].

For this chapter it may be helpful to review the background on optimization (Section 2.1.2) based planning as well as classical (Section 2.2.1) and learning based (Section 2.2.2) control and on information gathering (Section 2.3.2).

5.1 Problem Formulation

In this chapter, we start by considering the standard Informative Path Planning problem as presented in [93]. We seek to find the optimal path for a robotic vehicle, which satisfies the following equation:

$$\mathcal{P}^* = \underset{\mathcal{P}\in\Phi}{\operatorname{argmax}}\{I(\mathcal{P})\}\ s.t.\ C(\mathcal{P}) \leq B, \tag{5.1}$$

where $\mathcal{P}$ is a path defined as a series of waypoints $(x_0, x_1, ..., x_n) \in \mathbb{R}^2$. The optimal path, $\mathcal{P}^*$, is the path from the space of all possible paths Φ which maximizes the information function $I(\mathcal{P})$ and respects the cost constraint that the cost of the path, $C(\mathcal{P})$, does not exceed the mission budget, B. Any number of different cost functions can be used, such as energy, time, or distance travelled. In this chapter we consider budgets on the path travelled by the robotic vehicle, as we assume that the robotic vehicle will be travelling at a near constant speed. It is worth noting that the proposed methods are agnostic to the particulars of the cost function used, and it is straightforward to extend to other alternatives.

In reality, the path realized by the robotic vehicle will be different from the optimal planned path. We denote this realized path as $\bar{\mathcal{P}}$. As such, the common assumption that $\bar{\mathcal{P}} = \mathcal{P}^*$ does not hold for most field robotic applications. This can lead to a large discrepancies between the amount of information the robotic vehicle planned to gather and the amount of information the robotic vehicle actually gathered especially in budget limited missions. Below we will discuss the two non-exclusive formulations which we will use to attempt to solve this problem.

5.1.1 Path Tracking

First, and perhaps most obviously, one formulation to minimize the distance between the planned and executed path is to explicitly compute a policy which provides goals for the low-level controller to reduce the distance between the planned and executed paths. This can be formulated as the following expectation minimization problem:

$$\pi^* = \underset{\pi \in \Pi}{\operatorname{argmin}}\, \mathbb{E}\Big(\textbf{dist}(\mathcal{P}^*, \pi(\mathcal{P}^*))\Big), \tag{5.2}$$

where $\mathcal{P}^*$ is the optimal path as calculated above, Π is the set of all possible policies mapping the desired path to a realized path, $\pi(\mathcal{P}^*)$ is the path that the robotic vehicle will realize given a desired path, and the function **dist**$(\cdot, \cdot)$ is a distance metric between paths, such as the Hausdorff or Fréchet distance [26]. This formulation of π is fairly general and reflects the fact that in many field robotics applications, the control of the robotic vehicle is treated as a black-box. In this work we consider controllers in the form of a waypoint controller, but the methods presented here are general enough to be used by other popular controllers, such as a Pure Pursuit controller.

While the true distance between the planned path and realized path can only be known after execution, during planning and execution we can estimate $\pi(\mathcal{P}^*)$ through forward simulation. It is for this reason that we take the expectation of the distance over a distribution of possible paths. The robotic vehicle realizes path $\pi(\mathcal{P}^*) = \{x_0, x_1, \ldots, x_T\}$ where x_0 is the starting location and

$$x_t = f(x_{t-1}, u_{t-1}, \omega) \; \forall \, t > 0, \tag{5.3}$$

$$u_t = \rho(x_t, g_t),$$

where $f(\cdot)$ encodes the dynamics of the vehicle when moving from a state x_t under control u_t and subject to disturbances ω. These disturbances are assumed to be drawn from a normal distribution with mean μ and standard deviation σ. The control action at time t, u_t, is calculated from a policy $\rho(\cdot)$ which is induced by the onboard controller of the vehicle, which is assumed to be known or at least well approximated. Again, the form of controller we are considering is one where the control is calculated using the current state, x_t, and a goal position, g_t, which is typically a point along the desired path.

We can formalize this as a Markov decision process (MDP), which is given by the tuple $M = (S, A, P_a, R_a)$. The state, S, of the system is typically represented as a vector, $\mathbf{x}$, of positions and velocities. However this restricts the policy learned to only apply to a single goal configuration. In Section 5.2, we will discuss in detail our state-space representation to overcome this restriction. The actions, A, that the system can take are providing goal points to the low-level controller. The transition probabilities between states, P_a, are unknown, but we can approximate the true dynamics function $f(\cdot)$ with $\bar{f}(x_{t-1}, u_{t-1}, \omega) \rightarrow x_t$, which can serve to provide rollouts.

One of the more difficult aspects of this problem is to design an appropriate reward function, since the reward for a state depends upon the entire rollout, not just a single state-action pair, making the reward non-Markovian. As an example, one simple reward function would be to reward all states in the rollout with the same score based upon the distance between that rollout and the desired path.

More details on the exact methods used to solve Eq. 5.2 are discussed in Section 5.2.1.

5.1.2 Path Expectation

A second formulation is to instead treat the original problem presented in Eq. 5.1 as an expectation minimization problem over possible paths. This results in the following problem:

$$\mathcal{P}^* = \underset{\mathcal{P} \in \Phi}{\operatorname{argmax}} \, I_{\mathbb{E}}(\Psi(\mathcal{P}, \pi, \omega)) \;\; s.t. \; C(\mathcal{P}) \leq B, \tag{5.4}$$

where $\mathcal{P}^*$, $C(\mathcal{P})$, and B are as defined above in Eq. 5.2, $I_{\mathbb{E}}(\cdot)$ is the expected information of a distribution of possible paths, and $\Psi(\mathcal{P}, \pi, \omega)$ represents the distribution of paths which the robotic vehicle could realize during execution given path $\mathcal{P}$, policy π, and disturbances ω. As above, in this chapter we assume that ω is a normal distribution, but the method presented here does not depend on this assumption and could be generalized to other distributions.

The main difficulty in solving Eq. 5.4 is computing the distribution $\Psi(\mathcal{P}, \pi, \omega)$ and the gradient of the information with respect to $\mathcal{P}$. In this chapter, we will seek to solve this problem using a similar Stochastic Gradient Ascent algorithm as in Chapter 3 due to its sample efficient approach to approximate hard-to-calculate gradients. We will use Monte Carlo simulations to approximate $\Psi(\mathcal{P}, \pi, \omega)$.

More details on how we use these algorithms to solve Eq. 5.4 are discussed in Section 5.2.2.

5.2 Realizable Information Gathering Methods

Solving the information gathering problem in Equation 5.1 is typically a three step process.

1. First, the practitioner will develop a model of the environment that will be used to construct the information function $I(\cdot)$.

2. The second step is to then use the desired information gathering algorithm to plan an information gathering path over the environment.

3. The third step is to then hand the path off to the robotic vehicle's controller and the robotic vehicle will attempt to execute that path to the best of its abilities.

During the second step, existing methods fail to account for the myriad of factors which can effect the performance of the system, such as environmental disturbances or robotic vehicle limitations. In this work, we plan while accounting for these factors by using Monte Carlo simulation within a stochastic optimization framework. Additionally, during execution in step three, the robotic vehicle's controller typically acts in a myopic way, which can cause it to take sub-optimal actions such as circling waypoints. Our solution reasons not only about the current waypoint but also the following waypoint to help improve the realizability of the execution of the planned path.

To solve the minimization problem presented in Eq. 5.2, we developed a Waypoint Judging Policy (WJP) which utilizes a Reinforcement Learning (RL) based approach, which is presented below in 5.2.1. This serves as a bridge between the planner and the robotic vehicle's controller (between steps two and three) by passing on waypoint goals to the low-level controller and deciding when to consider the current waypoint achieved.

To solve the maximization problem in Eq. 5.4, we developed a Realization Aware Optimization (RAO) which leverages a Stochastic Gradient Ascent (SGA) framework, which is presented below in 5.2.2. This can be thought of as either and additional step done between the planning of the path and the controller or the waypoint policy receiving the path or as a part of the planner itself. One thing to note is that utilizing RAO requires additional information from the environmental modeling step where a model of the vehicle to be used must be provided.

5.2.1 Waypoint Judging Policy

As presented above, the minimization problem in Eq. 5.2 can be solved using a RL approach. This requires implementing the state-space, reward function, policy training algorithm, and the method for policy representation. Each of these is discussed below.

5.2.1.1 State Space

A typical representation of the state space as a vector of positions and velocities in a global frame would be quite difficult to transfer between path configurations because the policy would not be independent of the goal configuration. Instead we propose using a egocentric polar state space seen in Figure 5.1. In this representation the state is a vector, $\mathbf{x} = [d_{cur}, \theta_{cur}, d_{next}, \theta_{next}, \dot{\theta}]$, composed of the distance to the current goal (d_{cur}), the difference in heading to the current goal (θ_{cur}), the distance to the next goal (d_{next}), the difference in heading to the next goal (θ_{next}), and the turning velocity ($\dot{\theta}$). When the robotic vehicle's current goal is the final goal, we assign $d_{cur} = d_{next}$ and $\theta_{cur} = \theta_{next}$. We use an egocentric representation as it allows for learned policies to be independent of the goal configuration in the global frame, allowing the policies to be generalized across a large variety of goal configurations. Additionally, the polar representation more closely reflects the influential parameters when attempting to execute a path.

A common representation in reinforcement learning is a state-action pair, so we must also define the actions that the robotic vehicle can take in a state. Here, we assume that an ordered list of goals, $G = \{g_0, g_1, \ldots, g_N\}$, is used to define the path that the robotic vehicle has planned and that the low-level controller for the robotic vehicle is a waypoint

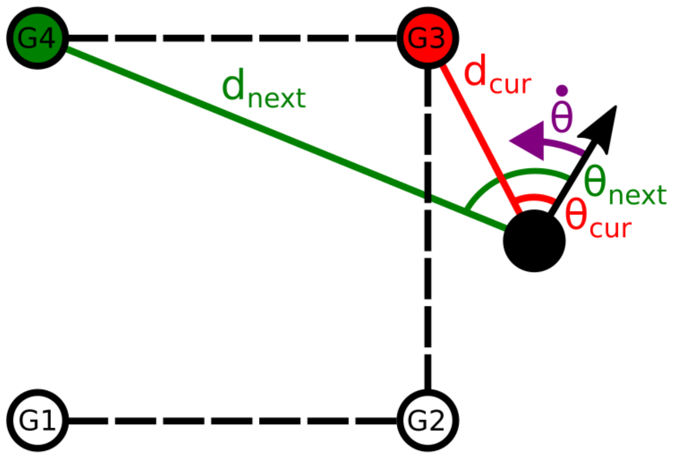


Figure 5.1: Our proposed path independent state representation, $x = (d_{cur}, \theta_{cur}, d_{next}, \theta_{next}, \dot{\theta})$, where the current state and heading are the black circle and arrow respectively and d_{cur} is the distance to the current goal, θ_{cur} is the difference in heading to the current goal, d_{next} is the distance to the next goal, θ_{next} is the difference in heading to the next goal, and $\dot{\theta}$ is the turning velocity.

based controller. This leads to a natural choice for the actions: either provide the current goal, g_n, to the low-level controller or provide the next goal, g_{n+1}.

5.2.1.2 Cost Function

In this work, the reward signal for training our policy, π, is derived from the distance between the planned and executed paths. We found that the Fréchet distance [26] matched our intuition for the distance between two paths due to the fact that it accounts for both the ordering, which the Hausdorff distance does not, as well as the location of the points along the paths. Additionally the Fréchet distance is computationally efficient to calculate in a discrete setting, taking $O(pq)$ where p and q are the number of discrete points in the two paths being compared [18]. As such, for the remainder of the chapter, when referring to the distance between two paths we use the Fréchet distance.

Simply using the Fréchet distance as the reward signal for training our policies fails to compute a useful policy (see Section 5.3.1.1 for results). One reason for this is that assigning the same reward (the Fréchet distance between the executed and desired path) to all state-action pairs along the path does not properly assign credit among these state-action pairs. When the policy decides to switch from a waypoint too early, it unfairly hurts following decisions made by the policy. Another reason for this failure to compute a useful policy is that this cost function is non-Markovian due to the dependence on both past and future states.

In addition to this naïve cost function, we present three additional cost functions which attempt to give a clearer signal of the cost of taking an action in a given state and are inspired by the work in reward shaping [108]. Additionally, these three additional cost functions attempt to address the non-Markovian aspect of this reward in different ways. These four cost functions can be seen in Figure 5.2.

1. *EntirePath*: The EntirePath (EP) cost function is the naïve reward function which computes the score for a state-action pair as the distance between the full rollout path to the full planned path. As such, every state-action pair in a rollout receives the same score. The non-Markovian nature of the problem is on full display here as this cost function depends entirely on the past and future states.

2. *FuturePath*: The FuturePath (FP) cost function compares the "perfect" future path to the rollout from the current state. The "perfect" future path is calculated as a straight line from the current state to the current goal and then follows the planned

path afterwards. The intuition behind this cost function is that once the robotic vehicle has reached a state, all that matters is what it can do to improve its future performance, and that its previous actions do not matter. Additionally, this intuition helps to reduce the non-Markovian nature of the problem by removing the dependence on history for the problem. Thus, the action that this cost updates has the largest effect on the states used in the cost function calculation as it is able to exert a slight effect on future actions.

3. *PastPath*: The PastPath (PP) cost function compares the “perfect” past plus future rollout to the full planned path. The “perfect” past is computed as the planned path up to the previous goal and then a straight line from the previous goal to the current state. The goal of this cost function is to not overly penalize a state for previous bad actions while still reflecting the true distance between the state and desired path. This cost function is less Markovian in nature than FuturePath because it still depends on the past states. However, this dependence is lessened from the EntirePath cost function by comparing to the ”perfect” past.

4. *FuturePastPath*: The FuturePastPath (FPP) cost function compares the full rollout to the two “perfect” sections calculated in the previous two cost functions. This cost function captures the positives from both FuturePath and PastPath by not overly penalizing the state due to previous actions but accurately reflecting the cost of the path that passes through that state. This cost function attempts to lessen the non-Markovian aspects of the cost function present in PastPath by comparing to the ”perfect” future instead of the desired path.

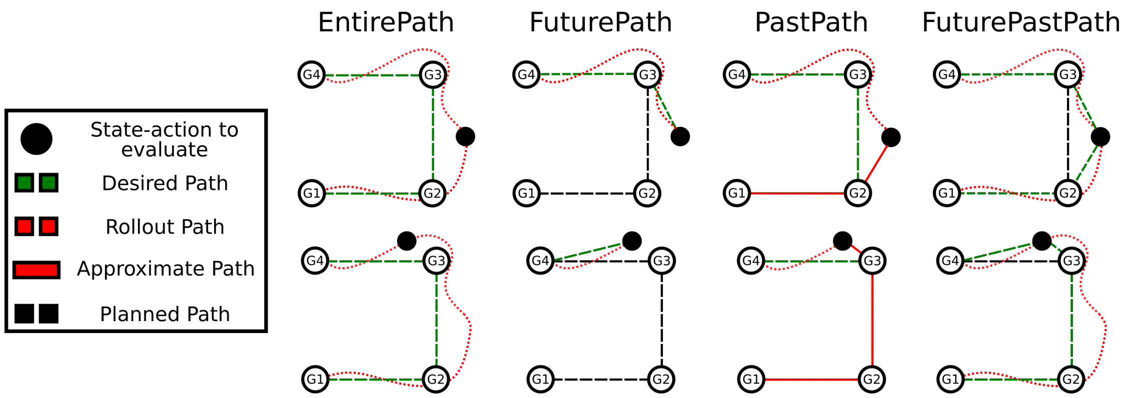


Figure 5.2: Our proposed four different cost functions for evaluating the value of a state-action pair which computes the distance between the red and green paths. In the EntirePath cost function each state-action pair receives the same score. In the FuturePath cost function the rollout from the state-action pair is compared to the "perfect" path from that state. In the PastPath cost function we compare the desired path to the rollout from that state-action pair combined with the "perfect" past up to that point. In the FuturePastPath we compare the full rollout to the "perfect" future plus the "perfect" past calculated in the previous two cost functions.

The non-Markovian nature of these cost functions also requires adjustments to the policy training procedure and is discussed in more detail in the next section.

5.2.1.3 Policy Training Algorithm

We use a reinforcement learning structure for training our policy as can be seen in Algorithm 3. We initialize a robotic vehicle through specifying the $\bar{f}(\cdot)$ function which approximates the true dynamics of the robotic vehicle, as well as dynamic properties such as maximum speed and turning rate. Additionally, any parameters (e.g. the radius to consider a waypoint achieved) needed for specifying the low-level controller onboard the robotic vehicle must be supplied. A world is initialized through specifying the goal configuration as well as the level of the disturbances, $\omega = \mathcal{N}(0, \sigma)$. Then, until an end condition is met, the algorithm resets the robotic vehicle to its initial position, and generates a state-action pair rollout using the Rollout function (shown in Algorithm 4) and current policy. Each state-action pair is then scored according to the chosen cost function from above. Finally, these scores are used to update the policy for the next iteration which is described in detail in Section 5.2.1.4. In this work, we use a fixed number of iterations as our end condition, but other conditions, such as policy convergence, could be used.

The rollout algorithm shown in Algorithm 4 is straightforward. While the robotic vehicle has not reached the final goal, we get the state from the robotic vehicle and then compute the action with the smallest score according to the equation in line 4. This action selection method is inspired by the Upper Confidence Bound (UCB) [4] and seeks to balance between exploring actions which have not been taken a large number of times and taking actions which minimize the cost, using c as a weighting parameter between these two considerations. After an action is selected the robotic vehicle then takes a step using that action with the specified $\bar{f}(\cdot)$ function which takes from the world the estimate of the disturbance levels. In contrast to classical policy training methods like Q-Learning and Monte Carlo control [100], the bias towards higher performing actions in UCB over random sampling helps the algorithm converge to better policies in these non-Markovian reward domains.

Algorithm 3 Policy Training Algorithm

```
1: Given: Robot, World, $\pi$
2: while $\neg end\_condition$ do
3:     Robot $\leftarrow InitRobot(\text{World})$
4:     $\mathbf{r} \leftarrow Rollout(\text{Robot}, \text{World}, \pi)$
5:     $\mathbf{s} \leftarrow ScoreRollout(\mathbf{r}, \text{World})$
6:     $\pi \leftarrow UpdatePolicy(\mathbf{s}, \pi)$
```

Algorithm 4 Rollout Algorithm

```
1: Given: Robot, World, $\pi$
2: while $\neg end\_not\_reached$ do
3:     $s \leftarrow \text{Robot}.getState()$
4:     $a \leftarrow \text{argmin}_{a \in A}\left(\pi.cost(s,a) - c * \sqrt{\frac{\ln(\sum_{a \in A} \pi.numPlayed(s,a))}{\pi.numPlayed(s,a)}}\right)$
5:     $rollout.append(s, a)$
6:     $\text{Robot}.step(a, \text{World})$
   return $rollout$
```

5.2.1.4 Policy Representation

We use a nearest neighbor policy representation similar to [60]. Our policy is stored as a lookup table for state-action pair values. The cost for performing an action at a given state is the average of the previous 10 scores received from that state-action pair. When updating this policy through the $UpdatePolicy(\mathbf{s}, \pi)$ method in Algorithm 3, the new score for the state-action pair is added to the list and the oldest score removed. While this policy representation is straightforward, as in [62], selective use of discretization in RL problems can lead to both faster training and better performance. However, utilizing a more complex policy function approximation method (such as a neural network) is a possible direction of future work.

5.2.2 Realization Aware Optimization

While better path tracking is one approach to improve the realizability of information gathering path, there are still potential problems. Most notably, there is a quite strong chance that a path will require turns which are sharper than the vehicle can perform, no matter what policy is controlling the vehicle. As such, during the construction of the path the realizability needs to be accounted for as well. To do this, we propose Realization Aware Optimization (RAO) algorithm. At a high-level, RAO operates by estimating the gradient by sampling perturbations and recombining them using a weighting based upon the objective function. Psuedocode for the RAO algorithm is presented in Algorithm 5

RAO requires an initial path, $\mathcal{P}$, and an information objective function $I(\cdot)$, which computes the path dependent reward for executing the $\mathcal{P}$ in the environment. Then RAO iterates through each of the waypoints in $\mathcal{P}$, and for each $x_i \in \mathcal{P}$ a set of K perturbations is generated. Each perturbation is generated by drawing from a distribution $\mathcal{D}$. This distribution, $\mathcal{D}$, can take on many different forms but is typically a zero-mean normal distribution. In this work we define $\mathcal{D}$ as a multivariate normal distribution:

$$\mathcal{D} = \mathcal{N}(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} \sigma_x & 0 \\ 0 & \sigma_y \end{bmatrix})$$

with zero-mean and covariance matrix defined by σ_x and σ_y, which are the variation in the x and y directions respectively. Note that here we have defined the perturbations as independent, but this is not required. On each iteration through $\mathcal{P}$, we consider the vertices in a random order to avoid undesirable effects of a particular ordering of the path.

After the set of perturbations, $\boldsymbol{\epsilon}$, is generated, each of these perturbations needs to be scored using the information function, $I(\cdot)$. Each of the perturbations, ϵ_i in $\boldsymbol{\epsilon}$ is independently applied to $\mathcal{P}$ at the given index to generate perturbed path $\hat{\mathcal{P}}_k$. Each of these $\hat{\mathcal{P}}_k$ is then scored using $I(\cdot)$ to generate a score vector s. This score vector, s, is then used in conjunction with $\boldsymbol{\epsilon}$ to calculate the update to that waypoint as:

$$\Delta = \frac{1}{K} \sum_{k=1}^{|K|} w_k \times \epsilon_k,$$

where

$$w_k = e^{-h\left(\frac{s_k - \min \mathbf{s}}{\max \mathbf{s} - \min \mathbf{s}}\right)},$$

is the weighting factor for perturbation ϵ_k comparing the score for ϵ_k to the maximum and minimum scores calculated and h is a weighting factor set to 1 in this work. A discount factor, λ, is used to facilitate convergence.

Algorithm 5 Realization Aware Optimization (RAO)

```
1: Given: $\mathcal{P}$, $I(\cdot)$, $\pi$, $\omega$
2: **while** $\neg$ stopping **do**
3:     **for** $p \in \mathcal{P}$ **do**
4:         $\boldsymbol{\epsilon} \longleftarrow$ genPertubations($\mathcal{D}, K$)
5:         s $\longleftarrow$ getScores($\mathcal{P}, \boldsymbol{\epsilon}, p, I(\cdot), \pi, \omega$)
6:         $\Delta \longleftarrow$ calcGrad(s, $\boldsymbol{\epsilon}$)
7:         $p \longleftarrow p + \Delta \times \lambda$
   **return** $\mathcal{P}$
```

To account for the realizability of these paths, during the computation of the score in line 5, the policy is passed to the function and a number of Monte Carlo simulations are performed. The average score of these simulations is then used as the score for the path. This allows for the system to more accurately reflect what the expected score for the path is. Additionally to note is that this means that the algorithm is also not necessarily expecting the vehicle to follow a straight line path between the waypoints and so when computing the expected score for this plan, Monte Carlo simulation should be used as well.

To compute the initial path for RAO, we use a method inspired by Rapidly Exploring Random Trees (RRT) [51]. The RRT algorithm is used to quickly generate a tree from which a large number of paths can be constructed using the leaf nodes. The five of these paths which gather the most amount of information are used to seed the RAO optimization.

5.3 WJP and RAO Results

To validate the proposed methods we performed a series of trials both in simulation and in the field using a Platypus Lutra autonomous boat. Through simulation for the path tracking policy algorithm we wanted to investigate the ability of each of the cost functions to compute a policy capable of reducing the distance between the planned and realized paths, the effectiveness of the different policy training algorithms, and the robustness of the trained policy across changes to the environment and vehicle model. For the path optimization framework, in simulation we wanted to look at the framework's ability to improve path generated by different information gathering algorithms as well as the ability of the framework to work with different policies, such as our trained policy and a default waypoint control policy, to increase the realizability of the information gathered. The two field trials demonstrate the capabilities of our framework to work on real vehicles and demonstrate the benefit of our high-level policy representation in allowing for easy sim-to-real transfer. These trials are discussed in more detail below.

5.3.1 Simulation Results

For our robotic vehicle model we used a model similar to [24]:

$$\dot{x} = v\text{cos}(\theta) + \omega_v$$

$$\dot{y} = v\text{sin}(\theta) + \omega_v$$

$$\dot{\theta} = u + \omega_t,$$

where v is the vehicle's velocity and was set to $4\frac{m}{s}$, θ is the vehicle's heading, u the commanded turning rate limited to be $\frac{-\pi}{8} \leq u \leq \frac{\pi}{8}$, the movement noise $\omega_v = \mathcal{N}(0, 0.4)\frac{m}{s}$, and the turning noise $\omega_t = \mathcal{N}(0, 0.1)\frac{rad}{s}$. The default controller for the robotic vehicle is a waypoint controller that considered a waypoint reached when the vehicle was 0.3 m from the desired waypoint. These parameters were selected to reflect that capabilities of our autonomous vehicle used in our field trials in Section 5.3.2. Note that this configuration using the default controller with radius of 0.3 is denoted **Default** throughout the following results. For the nearest neighbor policy the state-space was discretized at 0.5 m for d_{cur} and d_{next} and at 0.2 rad for θ_{cur}, θ_{next}, and $\dot{\theta}$. Additionally, we restrict the maximum allowed distance for d_{cur} and d_{next} to be less than 8 m.

5.3.1.1 Policy Simulations

We performed three different sets of tests for the policy in simulation. These three test are as follows:

1. Cost Function: The first set of experiments we performed examined the performance of the four different cost functions discussed in Section 5.2.1.2.
2. Training Algorithm: The second set of experiments examined the performance of the UCB training algorithm presented in Algorithm 3 against two other classical policy training algorithms.
3. Policy Robustness: The last set of simulation experiments we performed examined the ability of the trained policies to generalize when the robot and environment changed.

Cost Function Experiments

For our simulated policy training results, we generated two different sets of paths, one for training and one for testing, examples of which are shown in Figure 5.3. The training set consisted of two different systematic paths: a lawnmower pattern 30 m in length and a zig-zag pattern 25.4 m in length. The test set was generated as a set of 25 path configurations of between 5 and 10 waypoints by semi-randomly choosing a heading and distance from the previous waypoint. These paths include a number of different shapes including sharp turns and overlaps and varied in length from approximately 30 m to 175 m. Each policy was trained for 500,000 epochs. One epoch consists of one

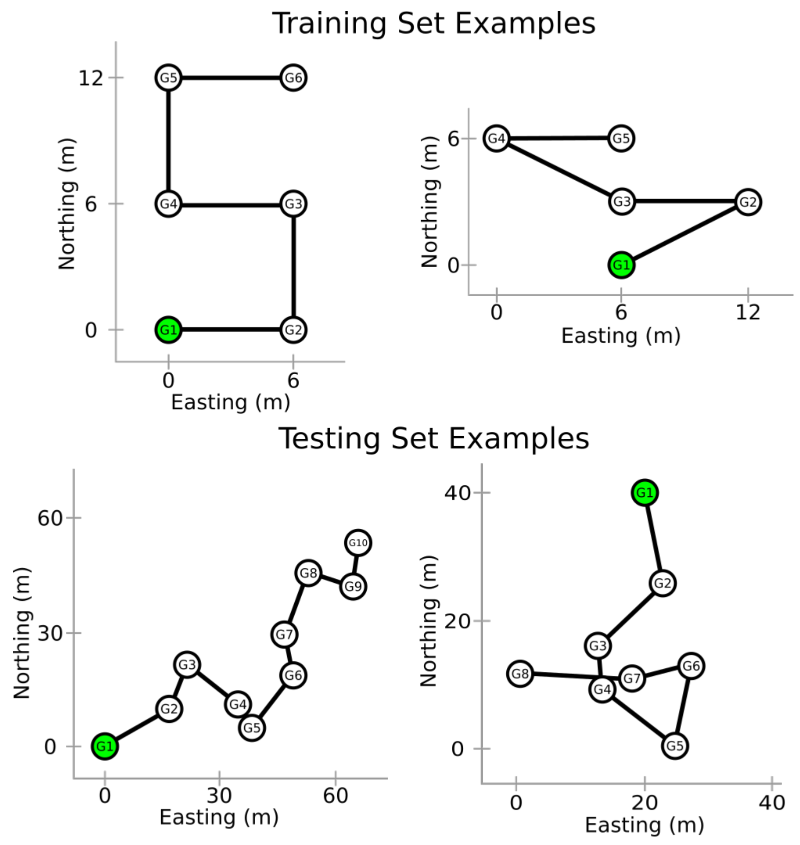


Figure 5.3: Representative paths from the training and testing set which start at G1 (green).

execution on each goal configuration in the training set. To obtain statistical results, we trained 20 different policies for each cost function.

Our first experiment examined the ability of each of the four cost functions to learn a policy capable of decreasing the distance between the planned and executed path. In Figure 5.4, we compare the learning curves of the four cost functions against the default policy with a set waypoint radius of 0.3 m at different training epochs. FuturePath, PastPath and FuturePastPath were all able to learn on the training set in under 1,000 epochs and reduced the average distance by 32.35, 37.18, and 31.09 percent respectively.

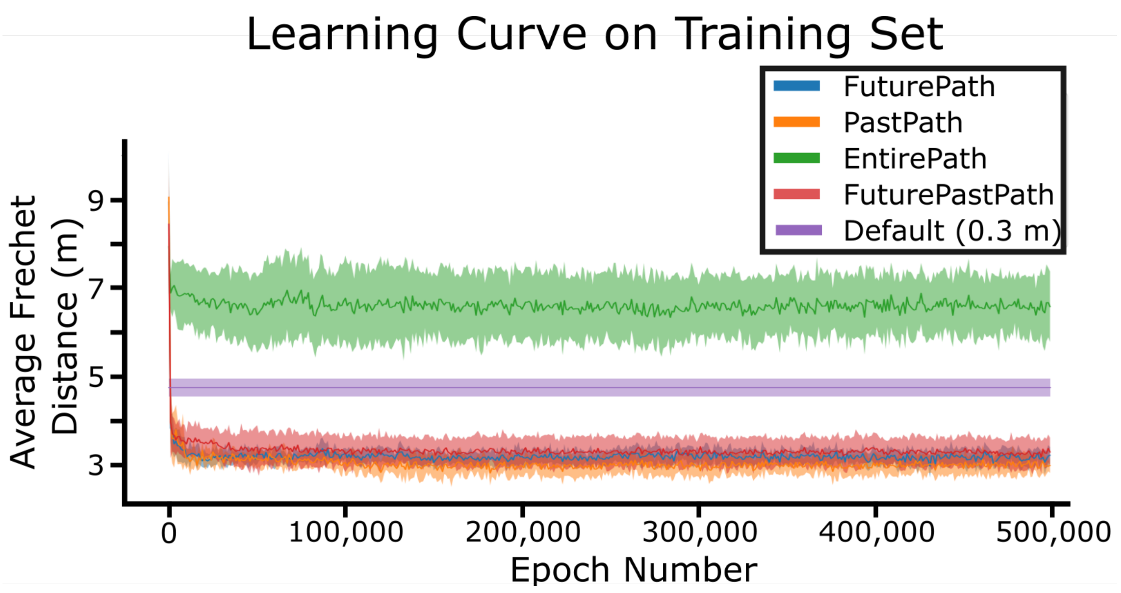


Figure 5.4: Average Fréchet distance of all 20 policies on the training set across all 500,000 training epochs. FuturePath, PastPath and FuturePastPath were all able to learn policies on the training set to reduced the average distance by 32.35, 37.18, and 31.09 percent respectively over the default while the EntirePath cost function was unable to learn to outperform the default.

In contrast, the EntirePath was unable to learn a policy which improved over the default waypoint radius controller. This matched our intuition that the reward signal from the EntirePath cost function is too noisy to effectively handle the credit assignment problem between the state-action pairs which comprise the path.

To determine if the policies being learned on the training set were not overfitting to features present there and that the state representation allows for the training of policies which are path independent, we also present the learning curves on the testing set. Note that we omit displaying the EntirePath cost function, as its inability to learn a policy on the training set led to policies that also performed poorly on the testing set with an average distance of 14.18 m. From Figure 5.5 it can be seen that the FuturePath cost function is able to learn a generalizable policy the fastest, reducing the average distance by 21.59 percent after 2,000 epochs and by 44.55 percent after training to completion. The PastPath cost function is able to significantly outperform the default after 25,000 epochs, offering a reduction of 18.64 percent after 25,000 epochs and 42.27 percent after training to completion. The FuturePastPath is able to significantly outperform the default after 70,000 epochs, offering a reduction of 25.91 percent after 70,000 epochs and 35.45 percent after training to completion. However this does not tell the full story, as after training to completion the FuturePath and PastPath have a standard deviation of 0.21 m and 0.35 m respectively, while the FuturePastPath has a standard deviation of 0.63 m. We suspect that the reason for the larger variance in the performance of the FuturePastPath is that using both approximations and comparing them to the full rollout causes the cost signal to be noisier due to the larger number of state-action pairs which affect the score for a single state-action pair. In general, it appears that the smaller dependence that a cost function has on other state-action pairs and the more Markovian the cost function is, the better that the policies that it trained performed.

Training Algorithm Experiments

After identifying the best performing cost function we wanted to compare the performance of our policy training algorithm with other classical training algorithms to see if the UCB exploration policy offered benefits for these non-Markvoian problem domains. We compared against Q-Learning and Monte Carlo Control [100], which are two classical policy training methods. Again, the same training procedure as above was taken,

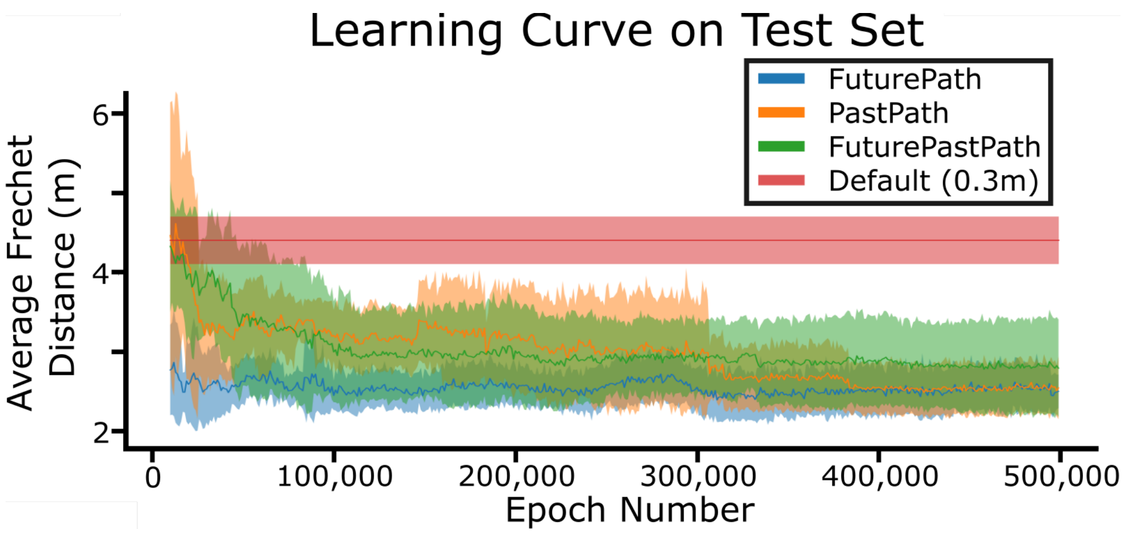


Figure 5.5: Average Fréchet distance of all 20 policies on the testing set across all 500,000 training epochs. FuturePath, PastPath and FuturePastPath were all able to learn policies on the training set to reduced the average distance by 44.55, 42.27, and 25.91 percent respectively over the default. The FuturePath is able to learn a policy that out performs the default in 2,000 epochs while it takes the PastPath 25,000 epochs and FuturePastPath 70,000 epochs.

with 20 different policies being trained for each training algorithm. The performance of the algorithms on the training set can be seen in Figure 5.6 and the performance on the testing set can be seen in Figure 5.7. There are a number of interesting trends that can be observed here. First, the proposed UCB training method outperforms the two comparison methods both on the training and testing set. Second, it is interesting to note that Q-Learning converges quickly to the default policies performance but is unable to learn a better policy than the default. The Monte Carlo algorithm is able to very slowly converge towards the default policy on the training set, but the learned policy does not generalize to the testing set. Lastly, it is interesting to note that on the training set UCB and Q-Learning converge to the default at approximately the same rate, while on the testing set the UCB converges slower but to a lower average Fréchet distance.

We were also interested in investigating how the performance of the policies trained on each cost function compared against a number of different waypoint controllers with different radii. A violin plot of the mean performance of the 20 policies across each goal configuration in the test set is presented in Figure 5.8. The average distance of both FuturePath and PastPath are comparable to that of the best default radius, 2.47 m and 2.5 m respectively to 2.03 m for a 2.5 m radius. However, when looking at the minimum distance FuturePath has a minimum of 1.65 m and PastPath of 1.58 m while the 2.5 m radius has a minimum distance of 1.94 m suggesting that when selecting the best performing policy it can outperform the best performing default radius. Additionally, through our method the policy could automatically be updated and adjusted online to account for changes in dynamics of environmental disturbances while a fixed radius controller cannot as easily be changed without human input. Another interesting ob-

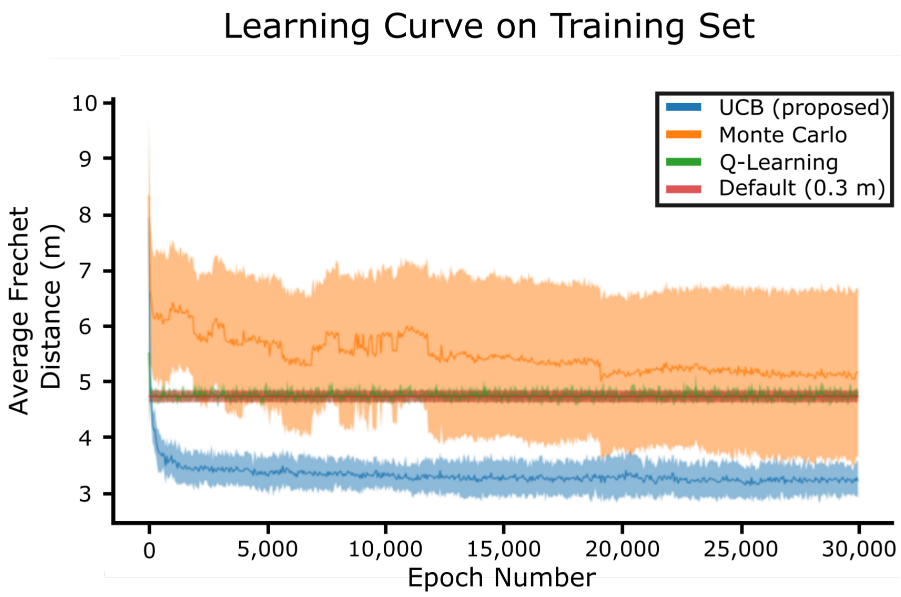


Figure 5.6: Average Fréchet distance of 20 policies trained using three different policy training algorithms. On the training set the proposed method is able to outperform both Q-Learning and Monte Carlo significantly while also converging quickly. Q-Learning quickly converges to the default policy while Monte Carlo is eventually able to approach the performance of the default policy but has a very wide standard deviation in performance.

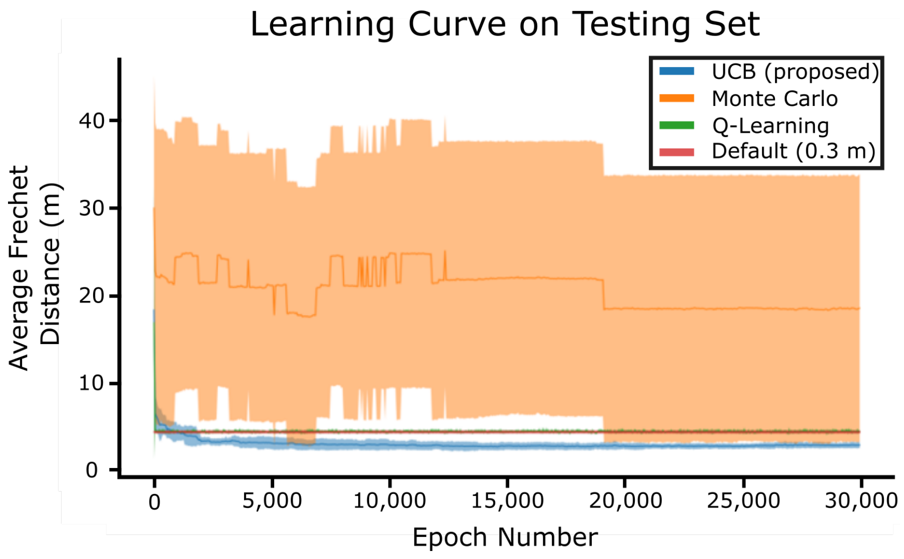


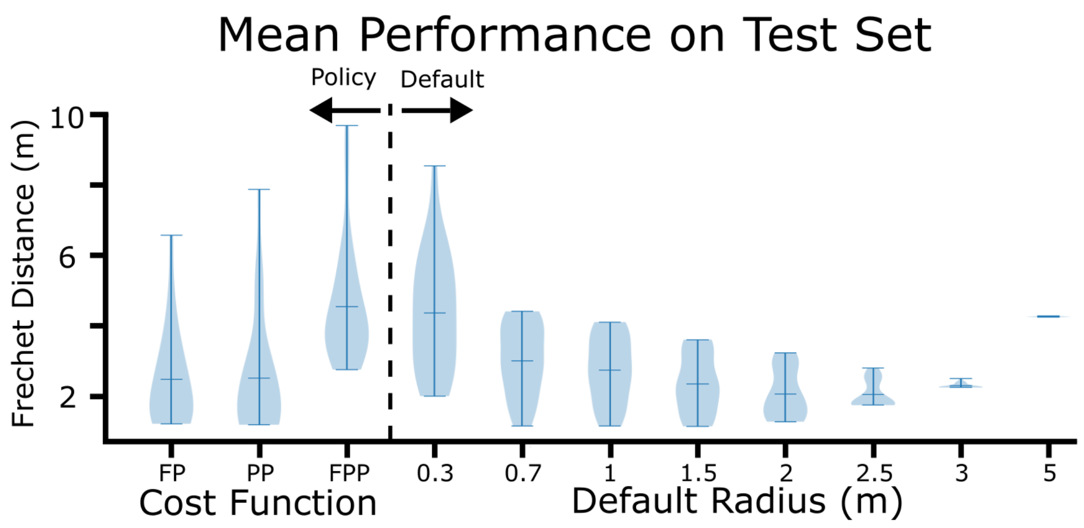
Figure 5.7: Average Fréchet distance of 20 policies trained using three different policy training algorithms. On the testing set the proposed method outperforms all methods after approximately 1000 epochs. One interesting difference between the test and training set is that the Q-Learning method quickly converges to the default policy on both while the proposed method is able to learn to outperform it. While the Monte Carlo policies are able to approach the performance of the default on the training set, they are unable to generalize that performance to the testing set.

servation to make from Figure 5.8 is the difference between the average and minimum distance of FuturePastPath. It appears that while the training signal from that cost function can be noisy, when it is able to find a good policy it is able to refine that policy to perform quite well. This suggests an interesting direction for future investigation of using the FuturePath cost function to quickly learn an acceptable policy and then refine it using the FuturePastPath cost function.

Policy Robustness Experiments

We also wanted to thoroughly investigate the performance of the different cost functions' generalizability as well as compare their performance against a large number of different default radii. The mean performance of all 20 different policies across the 25 different goal configurations from the test set trained using each cost function as well as that of a number of different default radii is shown in Figure 5.8. As expected, the performance of the FuturePath and PastPath cost functions perform quite similarly and the FuturePastPath performs slightly worse than both. Additionally the mean performance of both the FuturePath and PastPath cost function is similar to that of the best performing default radii. However, the average minimum performance from policies learned on all three cost functions outperformed those for all radii implying that if the best performing policy is chosen, it can outperform any set radius.

Lastly, we wanted to investigate the robustness of the trained policies and the ability of the algorithm to train policies across a number of different environments. We tested training the policies in five additional environments and then tested the policies trained



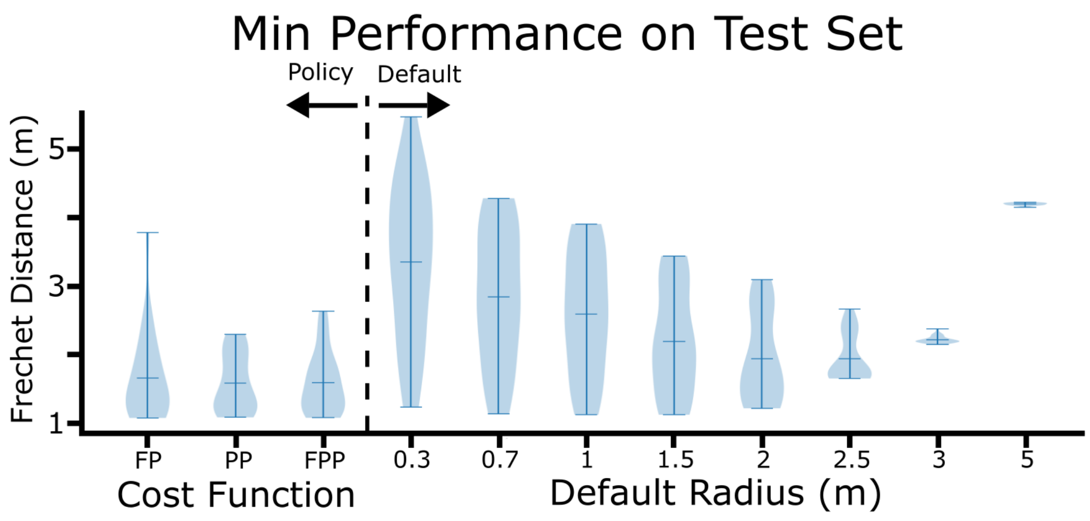


Figure 5.8: Mean and minimum Fréchet distance from all 20 trained policies across all 25 waypoint configurations in the test set. The FuturePath (FP) and PastPath (PP) are able to achieve similar average performance as that of the best performing default radius while having a lower average minimum value. Another interesting note is the difference in comparable performance between the average mean and minimum performance of the FuturePastPath (FPP) which indicates that when it is able to compute a well performing policy it is able to refine that policy.

in these five environments plus the policies trained in the normal environment across each of these environments. The additional environments were:

1. No Disturbances: there were no disturbances used during testing (i.e. the world was deterministic).

2. Low Disturbances: the disturbances were set to movement noise $\omega_v = \mathcal{N}(0, 0.2)\frac{m}{s}$, and the turning noise $\omega_t = \mathcal{N}(0, 0.05)\frac{rad}{s}$.

3. Medium Disturbances: the disturbances were set to movement noise $\omega_v = \mathcal{N}(0, 0.6)\frac{m}{s}$, and the turning noise $\omega_t = \mathcal{N}(0, 0.15)\frac{rad}{s}$.

4. High Disturbances: the disturbances were set to movement noise $\omega_v = \mathcal{N}(0, 0.8)\frac{m}{s}$, and the turning noise $\omega_t = \mathcal{N}(0, 0.2)\frac{rad}{s}$.

5. Different Model: we also attempted changing the boat model so that the top speed and turning rate was 80 percent of that of the normal boat while holding the disturbance level at that described above.

The results for the training and executing on different environments can be seen in Figure 5.9. As expected the different policies all outperform the default waypoint controller. The most interesting trend that can be seen in these graphs is that changing the vehicle model has a larger effect on the performance than changing the levels of disturbances. Another interesting trend is that lower levels of disturbance do better, but when training with no disturbances the performance is worse. This may be because the deterministic behavior when there are no disturbances causes the policy to not fully

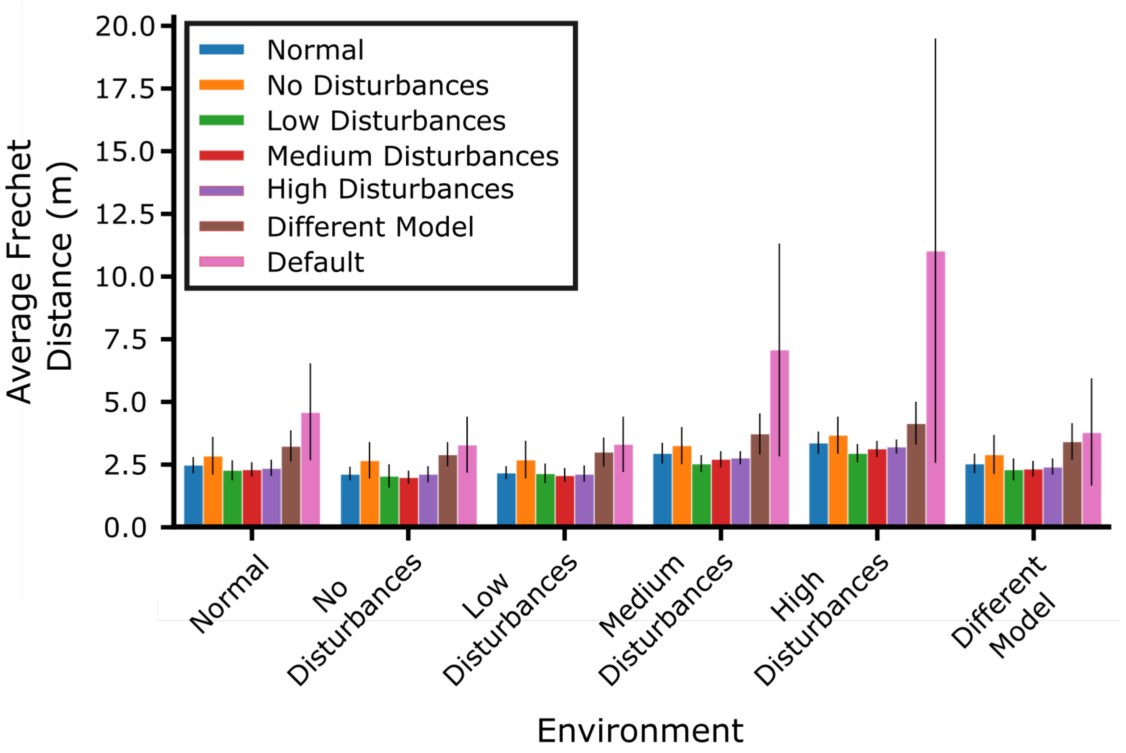


Figure 5.9: Average Fréchet distance of policies trained on the six different environments compared against each other on those same six environments. As expected the default has the highest average Fréchet distance on all environments. One interesting thing to note is that changing the vehicle model seems to have a much larger effect on the performance than changing the disturbance levels. The other interesting thing to note here is that training at lower disturbances seems to have a lower average Fréchet distance, but it is not a significant difference.

explore the state-space and therefore perform worse when these unknown states are encountered.

Lastly, we investigated the effects of varying the disturbances from 0 to 100 percent of the actuation levels. The average Fréchet distance can be seen in Figure 5.10. The performance of all the policy based methods slowly degrades as the amount of disturbances increases while the default waypoint controller starts to fail after the disturbances

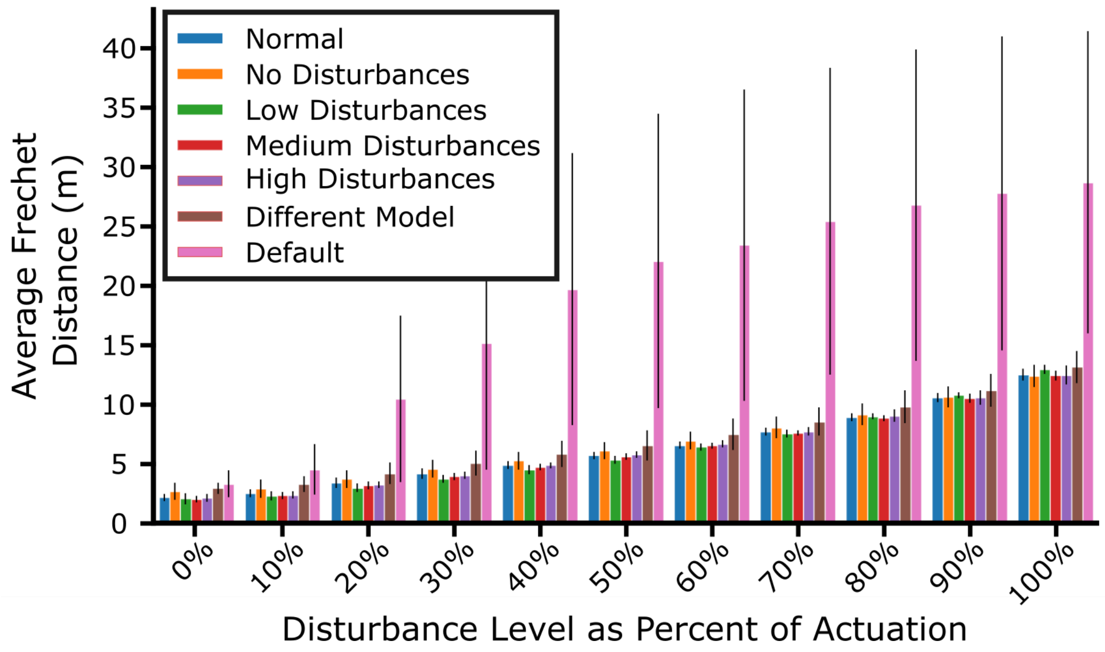


Figure 5.10: Average Fréchet distance of policies trained on the six different training environments. 20 different policies were trained for each training environment and were tested over 25 different worlds. The policy based methods all fail gracefully as the level of disturbance rises from 0 to 100 percent of the actuation level of the robotic vehicle. However, the default waypoint controller very quickly fails when the level of disturbance passes 20 percent of total actuation.

get to be 20 percent of the total actuation.

5.3.1.2 Stochastic Optimization Simulations

To test the ability of both RAO and the WJP to improve the realizability of an information gathering task, we performed a set of simulated trials. We constructed a set of 30 different random worlds which were generated using a sum-of-Gaussians method to randomly distribute information hotspots throughout each world. These worlds were 50 m by 50 m and budgets of 100 m were considered. We compared four different configurations:

1. Baseline: The baseline represents the state of practice in information gathering. We used an RRT based stochastic optimization information gathering algorithm which uses an RRT inspired algorithm to generate a number of paths through an environment, which are then optimized using SGA [34]. The path is then executed using a waypoint radius controller.

2. WJP Only: In this configuration we use the same information gathering algorithm as the Baseline but instead use the WJP policy to execute the planned path. The WJP policy used is the best performing policy calculated using the FuturePath cost function.

3. RAO Only: In this configuration we use the RAO algorithm to compute the information gathering path and provide it with the waypoint radius controller as the policy, which is also used to execute the path.

4. RAO + WJP: In this final configuration we use both the RAO algorithm to compute the information gathering path with the calculated WJP as the policy and then use the same policy to execute the path. Again, the WJP policy used is the best performing policy calculated using the FuturePath cost function.

A violin plot of the percent difference between the planned information gathered and the actual information gathered can be seen in Figure 5.11 and in Table 5.1. There are a number of interesting trends to observe from these results. First, the Baseline does quite poorly at gathering the expected amount of information due to the assumptions it is making about the capabilities of the vehicle. On average the Baseline gathers -12.98 percent of the planned information with a standard deviation of 8.72 percent. When just using the WJP, the vehicle is capable of on average gathering -1.75 percent of the expected information with a standard deviation of 7.75 percent. However, this distribution seems to have two peaks, corresponding to when deviations from the expected path result in the execution either gathering more of less information than expected. This is also reflected in the large standard deviation for this configuration. The third configuration where only RAO is used results in an average of -4.70 percent of the expected information being gathered with a standard deviation of 4.96 percent. It appears that when using RAO the system is able to account for a number of the deviations caused by the waypoint radius controller, but when that controller causes deviations it results in poor performances. Lastly, using RAO + WJP results in the best performance with an average percent difference in information gathered of -0.19 percent with a standard deviation of 2.20 percent. Here the system is able to quite accurately gather the expected amount of information.

5.3.2 WJP and RAO Field Results

We also performed two different field trials to verify the efficacy of the algorithms and frameworks presented here. The first set of trials was focused on testing the performance of the RL policies and that the high-level discrete policy representation was able to ease the sim-to-real transfer. The second set of field trials was testing the entire framework to ensure that the proposed methods were able to increase the realizability of the infor-

Table 5.1: Average percent of planned information gathered during execution for 30 random worlds in simulation. An average percent of information of zero would mean that the system gathered the expected amount of information, while a negative percent means the system gathered less than expected and a positive percent means it gathered more than expected.

Method	Average Percent of Expected Information (std)
Baseline	-12.98 (8.72)
WJP Only	-1.75 (7.75)
RAO Only	-4.70 (4.96)
RAO + WJP	-0.19 (2.20)

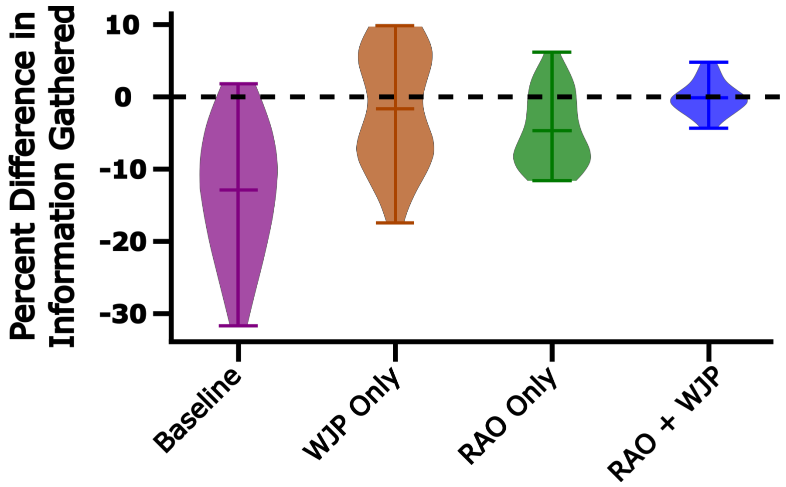


Figure 5.11: Violin plot of the percent difference in information gathered between the planned and actual information gathered. Using the Baseline the vehicle is almost never able to gather the expect amount of information. Using only the WJP, the vehicle on average collects close to the expected information but has a wide standard deviation. Using only RAO results in a much better performance than just the baseline but it struggles due to the limitations of the waypoint radius controller. Using RAO + WJP results in the best performance and the system is capable of quite accurately gathering the planned amount of information.

mation gathered in the field.

The vehicle used was a Lutra Prop autonomous boat from Platypus LCC, which is equipped with a number of environmental sensors and maximum speed of approximately $4.7\,\frac{m}{s}$. The Lutra system also has ROS integration, WiFi connectivity for connection to a base station, GPS, and basic waypoint following using the onboard controller which was the default for the field trials. A Getac laptop running Ubuntu 14.04 with an Intel i7-4600M CPU @ 2.90GHZ and 8 GB of RAM served as the base station where policy calculations were performed. These test were performed at Ireland Lake in Corvallis, OR (Lat: 44.563, Lon: -123.249) on two separate days.

5.3.2.1 Policy Field Trials

We performed three different path configurations: (1) systematic lawnmower and (2) zig-zag paths, as well as a (3) path inspired by an information gathering function. This information gathering path is modeling a path generated by attempting to follow a depth contour, which would have a path-dependent reward structure. The waypoints for the three paths can be seen in Figure 5.13 in dashed red, with a representative path for the default using the onboard controller shown in dark blue and a representative path from the policy shown in light blue. The policy used was the best performing policy trained with the Future cost function and no additional training was performed between simulation and deployment on the vehicle.

The average performance for the default and policy methods are shown in Table 5.2 and the individual scores for each run can be seen in Figure 5.12. Each path was

Table 5.2: Average Fréchet distance (in meters) from field deployment.

	Lawnmower	Diagonal	Information
Default	6.55 ± 0.89	5.23 ± 0.32	6.30 ± 0.01
Policy	4.23 ± 0.33	4.11 ± 1.65	5.84 ± 0.22

performed three times by both the default and the policy, except for the default on the information path where only two trials could be performed due to battery constraints. The policy offered on average a reduction in Fréchet distance of 35.42, 21.41, and 7.30 percent on the Lawnmower, Diagonal, and Information paths respectively. Qualitatively, we observed that our policy method was able to correctly choose when to consider moving onto the next waypoint. Additionally, it was able to avoid the behavior seen on the Information path in Figure 5.13, where the default had to circle back for a waypoint. Our trained policy was taken directly from simulation and deployed on a real vehicle showing that through the state-action representation we are able to successfully and easily achieve sim-to-real transfer.

5.3.2.2 Optimization Field Results

The second set of field trials investigated the ability of the framework to increase the realizability of the information gained. As such, four different configurations were tested during this deployment. These four were the same four configurations used in the optimization simulations in Section 5.3.1.2, Baseline, WJP Only, RAO Only, and RAO + WJP.

The different planned paths as well as the executed paths for the three trials can

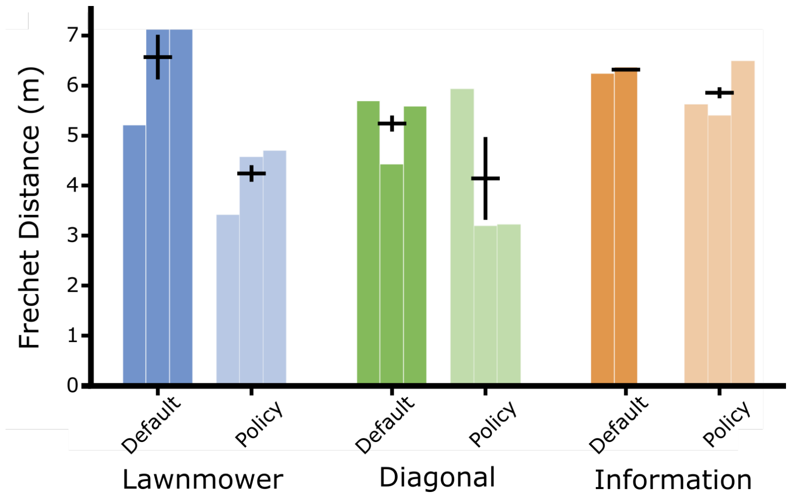


Figure 5.12: All distance results from field deployment across the three different goal configurations. On average the policy offered a reduction of Fréchet distance of 35.42, 21.41, and 7.30 percent over the default across the three goal configurations. Note that there are only two trials for the default on the Information path due to battery constraints.

be seen in Figure 5.15. The behaviors exhibited by the system in each of these four scenarios is consistent with those seen in simulation and the previous field trials. As can be seen in Figure 5.15(a), without the policy helping control the vehicle and the path not being optimized to reflect the abilities of the system, the vehicle does not track the first sharp turn well and as such fails to gather the expected amount of information. Figure 5.15(b) shows that just using the policy can increase the realizability of the path and information gathered. However, it does this by cutting off the first sharp turn, which might have been favorable in this circumstance is not always the best action. Figure 5.15(c) demonstrates that by using the realizability optimization framework, the system is able to greatly increase the realizability of the path. Lastly, we can see in Figure 5.15(d), the system is able to closely realize the expected amount of information.

As can be seen in Figure 5.14 and Table 5.3, the average Fréchet distance is in line

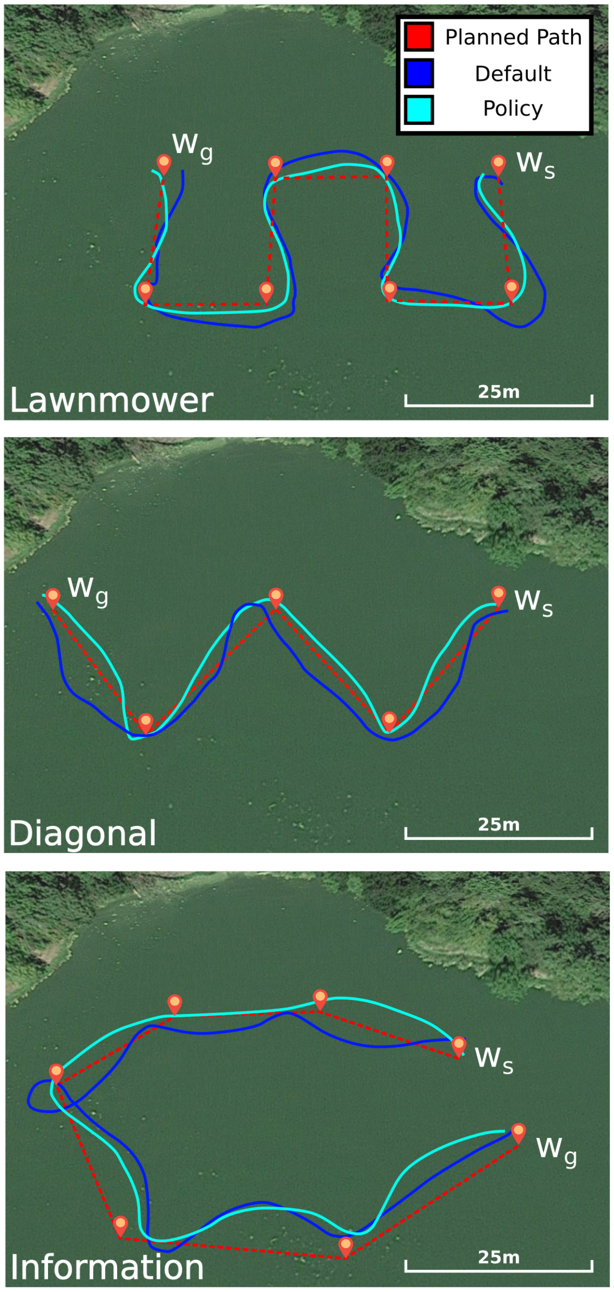


Figure 5.13: Representative paths taken by the vehicle during field trials starting at W$_s$ and ending at W$_g$. It can be observed that through intelligently choosing when to change to the next goal the policy is able to reduce overshoot on sharp turns and is able to avoid the behavior seen in the information path of looping back to achieve a waypoint.

Table 5.3: Average Fréchet distance between the planned path and executed paths on Platypus Lutra boat

Method	Average Fréchet Distance (std)
Baseline	21.21 (1.61)
WJP Only	7.28 (2.43)
RAO Only	6.96 (2.71)
RAO + WJP	5.54 (0.86)

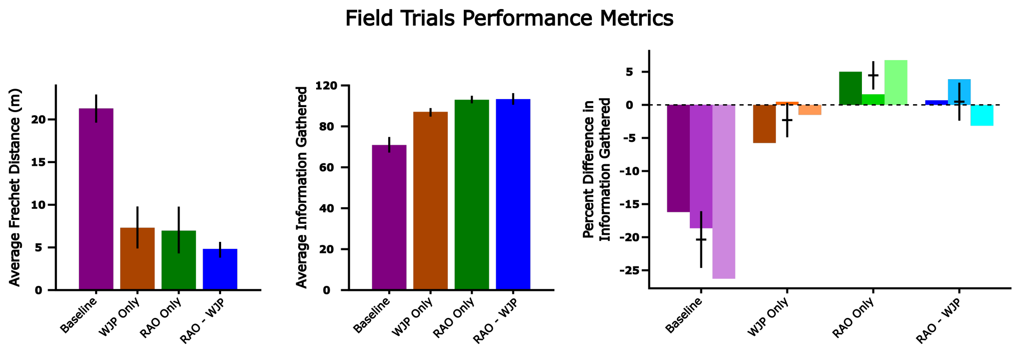


Figure 5.14: Three field trial performance metrics. First, using both the policy and optimization is able to significantly reduce the average Fréchet distance. On average both using RAO and RAO + WJP are able to gather the most amount of information. Lastly, using both RAO + WJP has the smallest percent difference in information gathered from the expected.

with the results seen above. When not using the policy or the optimization framework the system has quite a high average Fréchet distance. One interesting thing to note here is that both the policy and the optimization framework are able to independently significantly reduce the average Fréchet distance. Unsurprisingly, when using the policy and the optimization framework, the average Fréchet distance is the smallest.

In Figure 5.14 and Table 5.4 we can see the average information gathered by the four different configurations. As expected, when not using the policy or the realizability

Table 5.4: Average information gathered during execution

Method	Average Information Gathered (std)
Baseline	84.53 (4.53)
WJP Only	103.66 (2.74)
RAO Only	110.84 (2.29)
RAO + WJP	111.19 (3.17)

optimization framework the system gathers significantly less information. When just adding in the policy the system is able to gather a much larger amount of information but still fails to gather as much as the methods which use the realizability optimization framework. When using the optimization framework the system is able to gather more information as the paths account for the actual abilities of the system.

Lastly, we looked at the average percent of the expected information gathered by each of the configurations, which can be seen in Figure 5.14. When using the policy and the optimization framework the system is able to on average only gather a difference of 0.48 percent of the expected information. When using just the optimization framework the system gathers on average a difference of 5.54 percent of the expected information. Just using the policy causes the system to gather on average a difference of 2.25 percent of the expected information. One thing to note here thought is that by looking at Figure 5.15 (b), we can see that some of this is due to the environmental configuration, and the large average Fréchet distance indicates that in different path configurations this number could be larger.

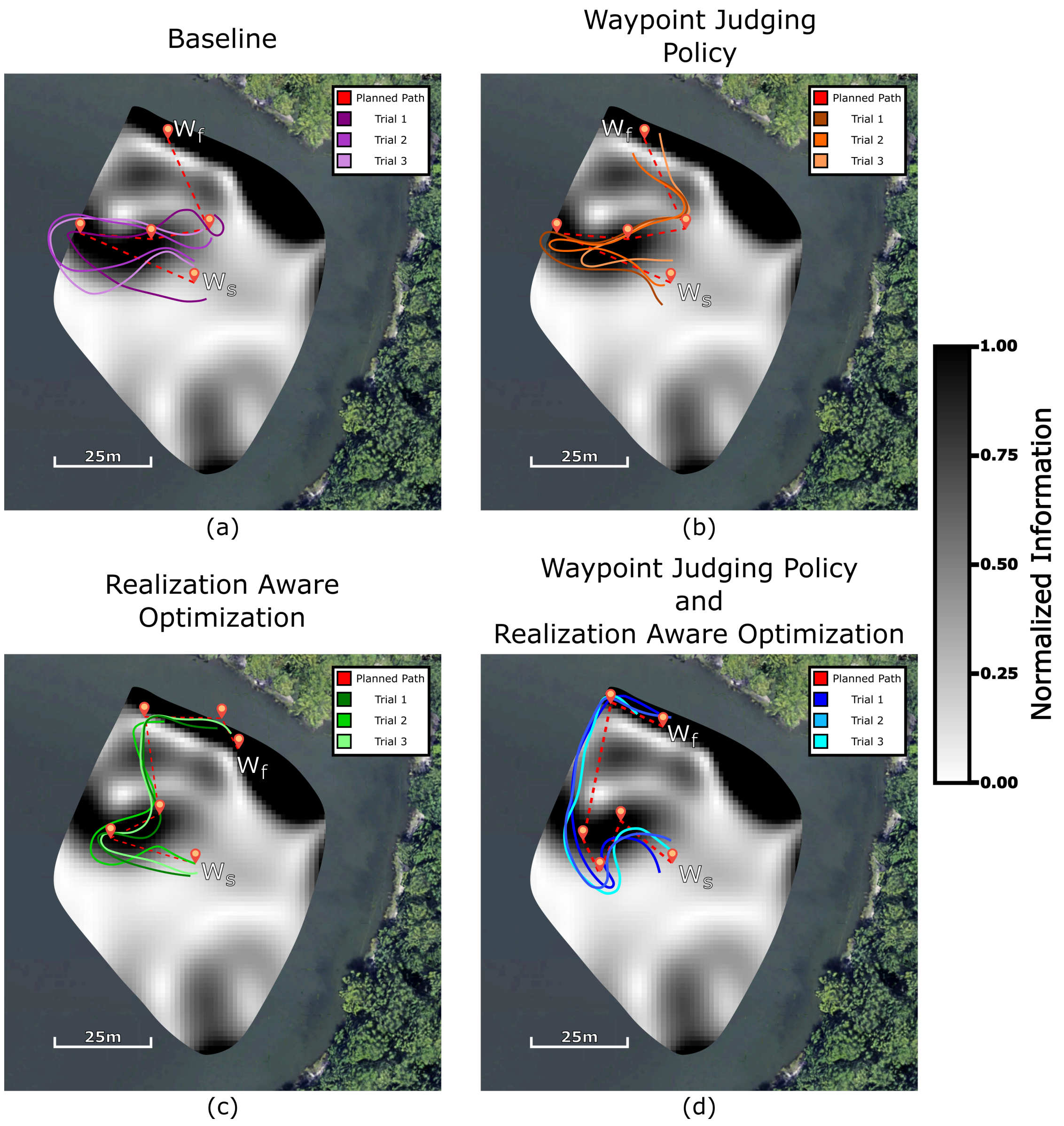


Figure 5.15: Representative paths taken by the vehicle during field trials for the four different system configurations. When using both the Waypoint Judging Policy and the Realization Aware Optimization, the system is able to on average track the planned path the best and gather the planned amount of information.

Table 5.5: Average percent of the planned information gathered

Method	Average Percent of Information (std)
Baseline	-20.30 (4.27)
WJP Only	-2.25 (2.58)
RAO Only	5.54 (2.16)
RAO + WJP	0.48 (2.86)

5.4 Conclusion

We have presented a reinforcement learning method that improves the realization of paths for autonomous vehicles by generating a policy that intelligently chooses goals for the low-level onboard controller. Through cost function design and state space selection, we are able to learn a policy which reduces the distance between the planned and realized paths while also easily generalizing to many different path configurations. Additionally, in field trials we were able to demonstrate successful sim-to-real transfer and showed that our approach is capable of reducing the distance between planned and realized paths in real-world field robotics conditions.

This chapter, as well as those before it, have presented a number of different algorithms and frameworks for solving the realizable path planning and execution problem. While we have taken a number of steps towards providing a solution to this problem, there are a number of directions for future work. In the next chapter we will provide a summary of this dissertation, as well as discuss directions for future research.

Chapter 6: Conclusion and Discussion

This dissertation has provided a number of solutions towards realizable path planning and execution across different levels of knowledge of the environment and relative levels of actuation. By leveraging stochastic optimization to compute gradients in domains where gradients are hard to compute due to non-smooth cost functions, we can use advantageous representations of the planning space to improve the realizability of the paths planned by robotic vehicles. These advancements help reduce the gap between the performance of robotic systems in heavily controlled environments with that in uncontrolled environments. The contributions presented in this dissertation are:

- A stochastic optimization algorithm that allows for an action-space representation while still allowing the reward function to be specified in the state-space. This representation more naturally allows disturbances to be accounted for when planning for low actuation vehicles, which in turn allows these planned paths to more closely match those that would be executed.

- An algorithm, Energy-Efficient Stochastic Trajectory Optimization, which allows vehicles with moderate actuation to plan energy-efficient paths in disturbances and more accurately realize the expected amount of energy expended.

- A replanning framework, which allows robotic vehicles to intelligently adapt their path in response to new information gathered about world during operation, which improves realizability during execution.

- A waypoint judging policy, which accounts for the expected disturbances in the environment, vehicle dynamics, as well as reasoning less myopically than current methods, all of which help improve realizability during execution.
- A realization aware optimization algorithm that accounts for the policy that the robotic vehicle will use during execution to improve the realizability of the planned path.

The remainder of this chapter provides a summary of the previous chapter, synthesizes this dissertation, and then describes potential avenues for future work.

In Chapter 3 we described a stochastic optimization algorithm for planning for a team of low-actuation vehicles in an information gathering task. By planning in the action-space of the vehicles, the algorithm can easily incorporate environmental disturbances into the planning process. By using a stochastic optimization framework, we are able to map gradient information from a state-space reward function to this action-space to efficiently compute action plans for the team of vehicles. The proposed method is able to outperform a greedy baseline by 8.63 percent and perform comparably to a Monte Carlo Tree Search (MCTS) based algorithm. While it performs comparably to the MCTS solution, MCTS requires 5.2 times the amount of computational time. This was shown through simulation on a number of realistic environmental models.

In Chapter 4 we presented our algorithm, Energy-Efficient Stochastic Gradient Descent (EESTO), which removes the assumption that waypoints are equally spaced in time from previous techniques by intelligently factoring the varying and non-varying aspects of the optimization. Using EESTO, we also present a framework for intelligently replanning when new information about the environment is gathered during execution. We are able to show up to a 21.57 percent reduction in energy usage over not replanning in simulation on a representative real world ocean current data set. Additionally, through field trials we demonstrate a 20 percent reduction in energy usage by an autonomous boat in wind field and associated surface currents in Eugene, Oregon.

In Chapter 5 we described a waypoint judging policy and associated reinforcement learning based training algorithm for non-Markovian cost functions. By accounting for the expected disturbances, the robot model, and reasoning over future goals, the waypoint judging policy is able to decrease the distance between the planned and executed path by up to 44.55 percent in simulation. By representing this policy at a high-level, we achieve impressive sim-to-real performance with no additional training on the vehicle, which was shown through field trials. In addition to this waypoint judging policy, we presented a realization aware optimization algorithm that used stochastic optimization to account for the path that the robotic vehicle would actually execute. This algorithm had on average a -0.19 percent difference in information gathered from the planned path in comparison to the state-of-practice which had an average of -12.98 percent difference in information gathered. We also demonstrated the benefits of this realization aware optimization through field trials in Corvallis, Oregon.

The fundamental goal of this research was to increase the realizability of path planning and execution for robotic vehicles. By using favorable representations, such as an action-space planning representation for low actuation vehicles and an egocentric state-space for waypoint following, disturbances can be naturally incorporated into planning and execution. The stochastic optimization methods presented in this dissertation allow these representations to be used for planning in an efficient manner, which was a major shortcoming of prior methods. The algorithms and frameworks presented in this dissertation allow for safer and more reliable operation of robotic vehicles in a number of challenging domains.

6.1 Future Work

While the methods presented here can help to increase the realizability of path planning and execution, there are a number of potential avenues for future work which we will now discuss.

6.1.1 Efficient Approximations of Uncertainty

Throughout this work, we have dealt with uncertainty by leveraging Monte Carlo simulations to construct approximations of the distributions of interest. While these simulation were relatively computationally efficient when compared to the state-of-the-art, the amount of computation time might be limiting in other domains. As such, one interesting and promising direction for future work would be to investigate more efficient methods for accounting for disturbances or calculating distributions of interest, such as those presented in [22]. This would also potentially allow for a more nuanced handling of the uncertainty, rather than just relying on the expectation.

6.1.2 Online Learning During Execution

One interesting avenue of future work would be to look more closely at how to use online learning to improve the realizability of paths during execution. This includes both methods for intelligently modeling and updating models of the environment and vehicle as well as how to incorporate this learning into the planning and execution. Additionally, it would potentially be useful to incorporate planning about the potential information that could be gathered during execution into the realizability planning framework to allow the system to account for the potential of the information it might gather.

6.1.3 Multi-Robot Realizability

In the multi-robot domain, it would be interesting to investigate how multiple robots could be used to increase the realizability across the team. One could imagine a situation where a single vehicle could be used to initially explore the environment and send back information which would then allow the rest of the team to plan more realizable paths. Another interesting question in this domain would be how and when the robot should share information about the environment accounting for the potential for it to affect the realizability for other vehicles.

Bibliography

[1] Adrian K Agogino and Kagan Tumer. Analyzing and visualizing multiagent rewards in dynamic and stochastic domains. *Autonomous Agents and Multi-Agent Systems*, 17(2):320–338, 2008.

[2] Tomas Baca, Petr Stepan, Vojtech Spurny, Daniel Hert, Robert Penicka, Martin Saska, Justin Thomas, Giuseppe Loianno, and Vijay Kumar. Autonomous landing on a moving vehicle with an unmanned aerial vehicle. *Journal of Field Robotics*, 36(5):874–891, 2019.

[3] KJ Benoit-Bird, T Patrick Welch, CM Waluk, JA Barth, I Wangen, P McGill, C Okuda, GA Hollinger, M Sato, and S McCammon. Equipping an underwater glider with a new echosounder to explore ocean ecosystems. *Limnology and Oceanography: Methods*, 16(11):734–749, 2018.

[4] Graeme Best, Oliver M Cliff, Timothy Patten, Ramgopal R Mettu, and Robert Fitch. Decentralised monte carlo tree search for active perception. In *Workshop on the Algorithmic Foundations of Robotics (WAFR)*, San Francisco, California, December 2016.

[5] Graeme Best, Oliver M Cliff, Timothy Patten, Ramgopal R Mettu, and Robert Fitch. Dec-MCTS: Decentralized planning for multi-robot active perception. *The International Journal of Robotics Research (IJRR)*, 38(2-3):316–337, 2019.

[6] Jonathan Binney and Gaurav S Sukhatme. Branch and bound for informative path planning. In *International Conference on Robotics and Automation (ICRA)*, pages 2147–2154, Saint Paul, Minnesota, May 2012.

[7] Robert Bohlin and Lydia E Kavraki. Path planning using lazy prm. In *International Conference on Robotics and Automation (ICRA)*, pages 521–528, San Francisco, California, April 2000.

[8] Blai Bonet and Hector Geffner. Action selection for mdps: Anytime AO* versus UCT. In *AAAI Conference on Artificial Intelligence*, Toronto, Canada, July 2012.

[9] Scott A Bortoff. Path planning for UAVs. In *IEEE American Control Conference (ACC)*, pages 364–368, Chicago, Illinois, August 2000.

[10] Adam Bry and Nicholas Roy. Rapidly-exploring random belief trees for motion planning under uncertainty. In *International Conference on Robotics and Automation (ICRA)*, pages 723–730, Shanghai, China, May 2011.

[11] Jeffrey A Caley and Geoffrey A Hollinger. Data-driven comparison of spatio-temporal monitoring techniques. In *OCEANS MTS/IEEE*, Washington DC., October 2015.

[12] Eduardo F Camacho and Carlos Bordons Alba. *Model Predictive Control*. Springer Science & Business Media, 2013.

[13] Bryant Chandler and Michael A Goodrich. Online RRT* and online FMT*: Rapid replanning with dynamic cost. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Vancouver, Canada, September 2017.

[14] Yi Chao and John D. Farrara. Regional ocean modeling and prediction group, http://west.rssoffice.com (visited: Jan - 2017).

[15] Benjamin Charrow, Gregory Kahn, Sachin Patil, Sikang Liu, Ken Goldberg, Pieter Abbeel, Nathan Michael, and Vijay Kumar. Information-theoretic planning with trajectory optimization for dense 3D mapping. In *Robotics: Science and Systems (RSS)*, Rome, Italy, July 2015.

[16] Guillaume Chaslot. *Monte-Carlo Tree Search*. PhD thesis, Maastricht: Universiteit Maastricht, Maastricht, Netherlands, 2010.

[17] Yvain de Viragh, Marko Bjelonic, C Dario Bellicoso, Fabian Jenelten, and Marco Hutter. Trajectory optimization for wheeled-legged quadrupedal robots using linearized zmp constraints. *IEEE Robotics and Automation Letters*, 4(2):1633–1640, 2019.

[18] Thomas Eiter and Heikki Mannila. Computing discrete fréchet distance cd-tr 94/64. Technical report, Citeseer, 1994.

[19] Daniel C Fernández and Geoffrey A Hollinger. Model predictive control for underwater robots in ocean waves. *IEEE Robotics and Automation Letters*, 2(1):88–95, 2017.

[20] Jaime F Fisac, Anayo K Akametalu, Melanie N Zeilinger, Shahab Kaynama, Jeremy Gillula, and Claire J Tomlin. A general safety framework for learning-based control in uncertain robotic systems. *IEEE Transactions on Automatic Control*, 64(7):2737–2752, 2018.

[21] Emilio Frazzoli, Munther A Dahleh, and Eric Feron. Maneuver-based motion planning for nonlinear systems with symmetries. *IEEE Transactions on Robotics*, 21(6):1077–1091, 2005.

[22] Kristoffer M Frey, Ted J Steiner, and Jonathan P How. Collision probabilities for continuous-time systems without sampling. In *Robotics: Science and Systems (RSS)*, Corvallis, Oregon, July 2020.

[23] Enric Galceran, Ricard Campos, Narcís Palomeras, David Ribas, Marc Carreras, and Pere Ridao. Coverage path planning with real-time replanning and surface reconstruction for inspection of three-dimensional underwater structures using autonomous underwater vehicles. *Journal of Field Robotics (JFR)*, 32(7):952–983, 2015.

[24] Bilal Hammoud and Elie Shammas. Planar time optimal paths for non-symmetric vehicles in constant flows. In *Robotics: Science and Systems (RSS)*, Ann Arbor, Michigan, June 2016.

[25] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, July 1968.

[26] Rachel M Holladay and Siddhartha S Srinivasa. Distance metrics and algorithms for task space path optimization. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5533–5540, Daejeon, Kores, October 2016.

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

[28] Geoffrey A Hollinger, Srinivas Yerramalli, Sanjiv Singh, Urbashi Mitra, and Gaurav S Sukhatme. Distributed data fusion for multirobot search. *IEEE Transactions on Robotics*, 31(1):55–66, 2015.

[29] V. T. Huynh, M. Dunbabin, and R. N. Smith. Predictive motion planning for auvs subject to strong time-varying currents and forecasting uncertainties. In *International Conference on Robotics and Automation (ICRA)*, pages 1144–1151, Seattle, Washington, May 2015.

[30] Gregg A Jacobs, Helga S Huntley, AD Kirwan, Bruce L Lipphardt, Timothy Campbell, Travis Smith, Kacey Edwards, and Brent Bartels. Ocean processes underlying surface clustering. *Journal of Geophysical Research: Oceans*, 121(1):180–197, 2016.

[31] Dylan Jones and Geoffrey A. Hollinger. Realizable robotic information gathering using reinforcement learning and stochastic optimization. *Under Review in Journal of Field Robotics (JFR)*.

[32] Dylan Jones and Geoffrey A Hollinger. Planning energy-efficient trajectories in strong disturbances. *IEEE Robotics and Automation Letters*, 2(4):2080–2087, 2017.

[33] Dylan Jones and Geoffrey A Hollinger. Real-time stochastic optimization for energy-efficient trajectories. *Robotics: Science and Systems Conference Workshop on Robot-Environment Interaction for Perception and Manipulation*, Ann Arbor. June, 2016.

[34] Dylan Jones, Michael J. Kuhlman, Don A. Sofge, Satyandra K. Gupta, and Geoffrey A. Hollinger. Stochastic optimization for autonomous vehicles with limited control authority. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Madrid, Spain, October 2018.

[35] Dylan Jones, Michael J. Kuhlman, Don A. Sofge, Satyandra K. Gupta, and Geoffrey A. Hollinger. Stochastic optimization for autonomous vehicles with limited control authority. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Madrid, Spain, October 2018.

[36] Mrinal Kalakrishnan, Sachin Chitta, Evangelos Theodorou, Peter Pastor, and Stefan Schaal. STOMP: Stochastic trajectory optimization for motion planning. In *International Conference on Robotics and Automation (ICRA)*, pages 4569–4574, Shanghai, China, May 2011.

[37] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research (IJRR)*, 30(7):846–894, 2011.

[38] Nare Karapetyan, Jason Moulton, and Ioannis Rekleitis. Dynamic autonomous surface vehicle control and applications in environmental monitoring. In *OCEANS MTS/IEEE*, Seattle, Washington, October 2019.

[39] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[40] Peter W Kimball, Evan B Clark, Mark Scully, Kristof Richmond, Chris Flesher, Laura E Lindzey, John Harman, Keith Huffstutler, Justin Lawrence, Scott Lelievre, et al. The ARTEMIS under-ice AUV docking system. *Journal of Field Robotics*, 35(2):299–308, 2018.

[41] Donald E Kirk. *Optimal Control Theory: An Introduction*. Courier Corporation, 2004.

[42] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European Conference on Machine Learning*, volume 6, pages 282–293. Springer, 2006.

[43] Marc Kramer and Alexander Geraldy. Energy measurements for micaz node. *University of Kaiserslautern, Kaiserslautern, Germany, Technical Report KrGe06*, 2006.

[44] Dov Kruger, Rustam Stolkin, Aaron Blum, and Joseph Briganti. Optimal AUV path planning for extended missions in complex, fast-flowing estuarine environments. In *International Conference on Robotics and Automation (ICRA)*, pages 4265–4270, Roma, Italy, April 2007.

[45] Michael J. Kuhlman, Dylan Jones, Don A. Sofge, Geoffrey A. Hollinger, and Satyandra K. Gupta. Coordinating underwater vehicle teams to conduct large-scale geospatial tasks. *Under Review in Journal of Ocean Engineering*.

[46] Dhanushka Kularatne, Subhrajit Bhattacharya, and M. Ani Hsieh. Time and energy optimal path planning in general flows. In *Robotics: Science and Systems (RSS)*, Ann Arbor, Michigan, June 2016.

[47] Vijay Kumar and Nathan Michael. Opportunities and challenges with autonomous micro aerial vehicles. *Robotics Research*, pages 41–58, 2017.

[48] Steven M LaValle. Rapidly-exploring random trees: A new tool for path planning, Technical Report. Computer Science Department, Iowa State University (TR 98–11). 1998.

[49] Steven M LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[50] Steven M LaValle, Michael S Branicky, and Stephen R Lindemann. On the relationship between classical grid search and probabilistic roadmaps. *The International Journal of Robotics Research (IJRR)*, 23(7-8):673–692, 2004.

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

[52] Taehwan Lee, Hanguen Kim, Hyun Chung, Yuseok Bang, and Hyun Myung. Energy efficient path planning for a marine surface vehicle considering heading angle. *Ocean Engineering*, 107:118–131, 2015.

[53] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[54] Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research (IJRR)*, 37(4-5):421–436, 2018.

[55] Maxim Likhachev, Geoffrey J Gordon, and Sebastian Thrun. ARA*: Anytime A* with provable bounds on sub-optimality. In *Advances in Neural Information Processing Systems*, pages 767–774, Vancouver, Canada, December 2004.

[56] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[57] Brandon D Luders, Sertac Karaman, and Jonathan P How. Robust sampling-based motion planning with asymptotic optimality guarantees. In *AIAA Guidance, Navigation, and Control (GNC) Conference*, pages 5097–5122, Boston, Massachusetts, August 2013.

[58] Yuanfu Luo, Haoyu Bai, David Hsu, and Wee Sun Lee. Importance sampling for online planning under uncertainty. *The International Journal of Robotics Research (IJRR)*, 38(2-3):162–181, 2019.

[59] Anirudha Majumdar and Russ Tedrake. Funnel libraries for real-time robust feedback motion planning. *The International Journal of Robotics Research (IJRR)*, 36(8):947–982, 2017.

[60] Elman Mansimov and Kyunghyun Cho. Simple nearest neighbor policy method for continuous control tasks. *Advances in Neural Information Processing Systems (NIPS) Deep Reinforcement Learning Symposium*, 2017.

[61] Roman Marchant and Fabio Ramos. Bayesian optimisation for informative continuous path planning. In *International Conference on Robotics and Automation (ICRA)*, pages 6136–6143, Hong Kong, China, May 2014.

[62] Enrico Marchesini and Alessandro Farinelli. Discrete deep reinforcement learning for mapless navigation. In *International Conference on Robotics and Automation (ICRA)*, Paris, France, August 2020.

[63] Artem Molchanov, Andreas Breitenmoser, and Gaurav S Sukhatme. Active drifters: Towards a practical multi-robot system for ocean monitoring. In *International Conference on Robotics and Automation (ICRA)*, pages 545–552, Seattle, Washington, May 2015.

[64] Marco Morales, Samuel Rodriguez, and Nancy M Amato. Improving the connectivity of PRM roadmaps. In *International Conference on Robotics and Automation (ICRA)*, pages 4427–4432, Taipei, Taiwan, September 2003.

[65] Mustafa Mukadam, Xinyan Yan, and Byron Boots. Gaussian process motion planning. In *International Conference on Robotics and Automation (ICRA)*, Stockholm, Sweden, May 2016.

[66] Sean Murray, Will Floyd-Jones, Ying Qi, Daniel J Sorin, and George Dimitri Konidaris. Robot motion planning on a chip. In *Robotics: Science and Systems (RSS)*, Ann Arbor, Michigan, July 2016.

[67] Michael E Napoli, Harel Biggie, and Thomas M Howard. Learning models for predictive adaptation in state lattices. In *Field and Service Robotics (FSR)*, pages 285–300. Springer, 2018.

[68] Jauwairia Nasir, Fahad Islam, Usman Malik, Yasar Ayaz, Osman Hasan, Mushtaq Khan, and Mannan Saeed Muhammad. RRT*-SMART: A rapid convergence implementation of RRT. *International Journal of Advanced Robotic Systems*, 10(7):299, 2013.

[69] Joseph L. Nguyen, Nicholas R. J. Lawrance, Robert Fitch, and Salah Sukkarieh. Real-time path planning for long-term information gathering with an aerial glider. *Autonomous Robots*, 40(6):1017–1039, August 2016.

[70] Norman S Nise. *Control Systems Engineering*. John Wiley & Sons, 2007.

[71] Chris J Ostafew, Angela P Schoellig, Timothy D Barfoot, and Jack Collier. Learning-based nonlinear model predictive control to improve vision-based mobile robot path tracking. *Journal of Field Robotics (JFR)*, 33(1):133–152, 2016.

[72] Luigi Palmieri and Kai O Arras. Efficient and smooth rrt motion planning using a novel extend function for wheeled mobile robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 205–211, Chicago, Illinois, September 2014.

[73] Jong Jin Park and Benjamin Kuipers. Feedback motion planning via non-holonomic RRT* for mobile robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4035–4040, Hamburg, Germany, September 2015.

[74] Xue Bin Peng, Erwin Coumans, Tingnan Zhang, Tsang-Wei Lee, Jie Tan, and Sergey Levine. Learning agile robotic locomotion skills by imitating animals. In *Robotics: Science and Systems (RSS)*, Corvallis, Oregon, July 2020.

[75] Arvind A Pereira, Jonathan Binney, Geoffrey A Hollinger, and Gaurav S Sukhatme. Risk-aware path planning for autonomous underwater vehicles using predictive ocean models. *Journal of Field Robotics (JFR)*, 30(5):741–762, 2013.

[76] Mihail Pivtoraiko and Alonzo Kelly. Efficient constrained path planning via search in state lattices. In *International Symposium on Artificial Intelligence, Robotics, and Automation in Space*, pages 1–7, 2005.

[77] Patrick A Plonski, Pratap Tokekar, and Volkan Isler. Energy-efficient path planning for solar-powered mobile robots. *Journal of Field Robotics (JFR)*, 30(4):583–601, 2013.

[78] David Pollard. *A User's Guide to Measure Theoretic Probability*. Cambridge University Press, 2002.

[79] Marija Popović, Gregory Hitz, Juan Nieto, Inkyu Sa, Roland Siegwart, and Enric Galceran. Online informative path planning for active classification using uavs. In *International Conference on Robotics and Automation (ICRA)*, pages 5753–5758, Singapore, May 2017.

[80] Martin L Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., 1994.

[81] Dushyant Rao and Stefan B Williams. Large-scale path planning for underwater gliders in ocean currents. In *Australasian Conference on Robotics and Automation (ACRA)*, 2009.

[82] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

[83] Christopher Reardon and Jonathan Fink. Air-ground robot team surveillance of complex 3d environments. In *IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pages 320–327. IEEE, 2016.

[84] National Research Council. *Science at Sea: Meeting Future Oceanographic Goals with a Robust Academic Research Fleet*. The National Academies Press, Washington, DC, 2009.

[85] Zhongqiang Ren, Chaohui Gong, and Howie Choset. Deformed state lattice planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Vancouver, Canada, September 2017.

[86] Andrei A Rusu, Mel Vecerik, Thomas Rothörl, Nicolas Heess, Razvan Pascanu, and Raia Hadsell. Sim-to-real robot learning from pixels with progressive nets. *arXiv preprint arXiv:1610.04286*, 2016.

[87] Gilhyun Ryou, Ezra Tal, and Sertac Karaman. Multi-fidelity black-box optimization for time-optimal quadrotor maneuvers. In *Robotics: Science and Systems (RSS)*, Corvallis, Oregon, July 2020.

[88] Tom Schouwenaars, Bernard Mettler, Eric Feron, and Jonathan How. Hybrid model for trajectory planning of agile autonomous vehicles. *Journal of Aerospace Computing, Information, and Communication*, 1(12):629–651, 2004.

[89] John Schulman, Yan Duan, Jonathan Ho, Alex Lee, Ibrahim Awwal, Henry Bradlow, Jia Pan, Sachin Patil, Ken Goldberg, and Pieter Abbeel. Motion planning with sequential convex optimization and convex collision checking. *The International Journal of Robotics Research (IJRR)*, 33(9):1251–1270, 2014.

[90] Archit Sharma, Michael Ahn, Sergey Levine, Vikash Kumar, Karol Hausman, and Shixiang Gu. Emergent real-world robotic skills via unsupervised off-policy

reinforcement learning. In *Robotics: Science and Systems (RSS)*, Corvallis, Oregon, July 2020.

[91] Alexander F Shchepetkin and James C McWilliams. The regional oceanic modeling system (ROMS): a split-explicit, free-surface, topography-following-coordinate oceanic model. *Ocean Modelling*, 9(4):347–404, 2005.

[92] Thierry Siméon, J-P Laumond, and Carole Nissoux. Visibility-based probabilistic roadmaps for motion planning. *Advanced Robotics*, 14(6):477–493, 2000.

[93] Amarjeet Singh, Andreas Krause, Carlos Guestrin, William J Kaiser, and Maxim A Batalin. Efficient planning of informative paths for multiple robots. In *International Joint Conference on Artificial Intelligence (IJCAI)*, volume 7, pages 2204–2211, Hyderabad, India, January 2007.

[94] Amarjeet Singh, Andreas Krause, and William J. Kaiser. Nonmyopic adaptive informative path planning for multiple robots. *21st International Joint Conference on Artificial Intelligence*, pages 1843–1850, July 2009.

[95] Ryan Skeele, Jen Jen Chung, and Geoffrey A Hollinger. Risk-aware graph search with dynamic edge cost discovery. In *Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2016.

[96] Ryan N Smith and Van T Huynh. Controlling buoyancy-driven profiling floats for applications in ocean observation. *IEEE Journal of Oceanic Engineering*, 39(3):571–586, 2014.

[97] Thane Somers and Geoffrey A Hollinger. Human–robot planning and learning for marine data collection. *Autonomous Robots*, 40(7):1123–1137, 2016.

[98] Anthony Stentz. Optimal and efficient path planning for partially-known environments. In *International Conference on Robotics and Automation (ICRA)*, pages 3310–3317, San Diego, California, May 1994.

[99] Deepak N Subramani and Pierre FJ Lermusiaux. Energy-optimal path planning by stochastic dynamically orthogonal level-set optimization. *Ocean Modelling*, 100:57–77, 2016.

[100] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[101] Chiew Seon Tan, Robert Sutton, and John Chudley. An incremental stochastic motion planning technique for autonomous underwater vehicles. *IFAC Proceedings Volumes*, 37(10):483–488, 2004.

[102] Gao Tang and Kris Hauser. A data-driven indirect method for nonlinear optimal control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4854–4861, Vancouver, Canada, September 2017.

[103] Xinyu Tang, Jyh-Ming Lien, NM Amato, et al. An obstacle-based rapidly-exploring random tree. In *International Conference on Robotics and Automation (ICRA)*, pages 895–900, Orlando, Florida, May 2006.

[104] Russ Tedrake. LQR-trees: Feedback motion planning on sparse randomized trees. In *Robotics: Science and Systems (RSS)*, Seattle, USA, June 2009.

[105] Evangelos Theodorou, Jonas Buchli, and Stefan Schaal. Learning policy improvements with path integrals. In *Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 828–835, Sardinia, Italy, May 2010.

[106] Chuck Thorpe and Hugh F Durrant-Whyte. Field robots. In *International Symposium on Robotics Research (ISRR)*, pages 329–340, Lorne, Australia, November 2001.

[107] Sebastian Thrun, Mike Montemerlo, Hendrik Dahlkamp, David Stavens, Andrei Aron, James Diebel, Philip Fong, John Gale, Morgan Halpenny, Gabriel Hoffmann, et al. Stanley: The robot that won the darpa grand challenge. *Journal of Field Robotics (JFR)*, 23(9):661–692, 2006.

[108] Alexander Trott, Stephan Zheng, Caiming Xiong, and Richard Socher. Keeping your distance: Solving sparse reward tasks using self-balancing shaped rewards. In *Advances in Neural Information Processing Systems (NIPS)*, pages 10376–10386, Vancouver, Canada, December 2019.

[109] Chris Urmson and Reid Simmons. Approaches for heuristically biasing RRT growth. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1178–1183, Las Vegas, Nevada, October 2003.

[110] Dustin J Webb and Jur van den Berg. Kinodynamic RRT*: Optimal motion planning for systems with linear differential constraints. *arXiv preprint arXiv:1205.5088*, 2012.

[111] Grady Williams, Nolan Wagener, Brian Goldfain, Paul Drews, James M Rehg, Byron Boots, and Evangelos A Theodorou. Information theoretic MPC for model-based reinforcement learning. In *International Conference on Robotics and Automation (ICRA)*, pages 1714–1721, Singapore, May 2017.

[112] Jonas Witt and Matthew Dunbabin. Go with the flow: Optimal path planning in coastal environments. *Australasian Conference on Robotics and Automation (ACRA)*, pages 1–9, 2008.

[113] Peter R Wurman, Raffaello D'Andrea, and Mick Mountz. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI magazine*, 29(1):9–9, 2008.

[114] Enrica Zereik, Marco Bibuli, Nikola Mišković, Pere Ridao, and António Pascoal. Challenges and future trends in marine robotics. *Annual Reviews in Control*, 46:350–368, 2018.

[115] Marvin Zhang, Sharad Vikram, Laura Smith, Pieter Abbeel, Matthew Johnson, and Sergey Levine. SOLAR: Deep structured representations for model-based reinforcement learning. In *International Conference on Machine Learning (ICML)*, pages 7444–7453, Long Beach California, June 2019.

[116] Matt Zucker, Nathan Ratliff, Anca D Dragan, Mihail Pivtoraiko, Matthew Klingensmith, Christopher M Dellin, J Andrew Bagnell, and Siddhartha S Srinivasa. CHOMP: Covariant hamiltonian optimization for motion planning. *The International Journal of Robotics Research (IJRR)*, 32(9-10):1164–1193, 2013.