UTS UNIVERSITY OF TECHNOLOGY SYDNEY

Optimal path planning for autonomous underwater gliders in time-varying flow fields

by

James Ju Heon Lee

A thesis submitted in partial fulfilment of the
requirements for the degree of Doctor of Philosophy

at the
School of Mechanical and Mechatronic Engineering
Faculty of Engineering and Information Technology
University of Technology Sydney

June 2021

Certificate of Original Authorship

I certify that the work in this thesis has not previously been submitted for a degree nor has it been submitted as part of requirements for a degree except as fully acknowledged within the text.

I also certify that the thesis has been written by me. Any help that I have received in my research work and the preparation of the thesis itself has been acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis.

This research is supported by the Australian Government Research Training Program.

Production Note:
Signed: Signature removed prior to publication.

Date: 10th June, 2021

Optimal path planning for autonomous underwater gliders in time-varying flow fields

by

James Ju Heon Lee

A thesis submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy

Abstract

Marine robots perform various oceanic missions for commercial, scientific and military purposes. Some of these tasks include resource tracking, environmental surveying and coastal surveillance. Underwater gliders are a special class of marine robots that do not use active propulsions to move forward. This property makes the gliders more energy-efficient compared to other marine robots, and thus well-suited for long-duration missions. These missions benefit from autonomous operations that are either energy-optimal or time-optimal to maximising glider operation time and minimise human interactions. Such a level of automation is difficult to achieve, however. The underwater glider operates under a high-dimensional dynamic model with non-linear control, making it difficult to model mathematically. Optimal navigation in a flow field environment, known as Zermelo's Problem, is also a century-old open problem. This research introduces a trim-based model that reduces the glider control problem to a simpler 6D kinodynamic problem. We address this simpler problem using a state-of-the-art sampling-based algorithm to demonstrate full 3D underwater glider motion planning over various static flow fields and obstacles.

For real-world applications, it is also essential to consider the dynamics of the environment. Therefore it is natural to expect planning algorithms for underwater gliders to handle variations in flow fields. As the glider's performance heavily depends on the surrounding

flow field, planning involves the time-dependent shortest path (TDSP) problem, which has been open since the original work on graph search problem in the 1960s. This research introduces a new special case of the TDSP problem for vehicles in dynamic ocean currents. An optimal policy is solved for a time-dependent discrete graph over a dynamic flow field in polynomial time. Integrating both the trim-based and TDSP work addresses the path planning problem for underwater gliders by synthesising a continuous path from the optimal policy using the trim-based model.

The significance of this research is that it introduces an increased level of autonomy in underwater robots. The theoretical work allows for more accurate glider navigation, and considering dynamic ocean currents allows the glider to exploit the environment for practical advantages. These results also improve autonomous operation so that it requires less manual intervention from humans. This thesis shows examples of these ideas, and we are currently planning a long-duration field deployment to demonstrate these results in practice.

Acknowledgements

The submission of this dissertation marks the conclusion of a major life goal, which would not have been possible without the support of many extraordinary people. First and foremost, I would like to give praise to God, for nothing would be possible without Him. I would like first to thank my supervisor, Professor Robert Fitch, who has been a fantastic supervisor during my PhD degree. Rob's leadership skills, tactics, encouragement and patience were all instrumental in providing the best PhD experience that I could receive. Another whom I want to thank is my co-supervisor, Chanyeol Yoo, for his invaluable assistance in writing papers, and offering constant support during some of the most uncertain times of my candidature. This thesis could have never been completed without his help. A special abrupt thanks also go to Felix Kong who, despite arriving near the end of my candidature, has been a tremendous help during the final writing period. I would also like to thank Stuart Anstee (DSTG) for providing me with a wonderous opportunity to work with underwater gliders, which has become one of the main topics of my thesis.

Outside my superiors, I want to express my biggest gratitude to the *Gumnut Crew*: Jeremy Chang, Kevin Hendrawan, Kelvin Hsu, and Andrew Mak, for being my anchor to sanity throughout my academic journey since high school. I would also like to thank my church group for being patient with me for the last four years, and for their prayer for my wellbeing. Thank you also to all my friends: Rocky Pang, Stephanie Gunadi, Gloria Wang, Kranthi Baddam, David Shalavin, Cadmus To, Stefan Kiss, Giuseppe Riggio, Julien Collart, Mitchell Usayiwevu, Lan Wu, Anna Lidfors Lindqvist, Hasti Hyt, Jasprabhjit Mehami, Karthick Thiyagarajan, and so many others that I can't name due to the page limit. Thank you all for sticking by me after all these years despite my many short fallings. Without you guys, I would be fumbling in my depression, probably flipping burgers at a fast-food restaurant.

Lastly, I want to dedicate this thesis to my family: my Mom, my Dad, Jacob, Grandpa, Grandma, Jin, Min, Peter, my Cousins, my Aunts and my Uncles. Thank you for all your unconditional love and support for the past 27.5 years. I would not have even attempted a PhD if not for your trust in my abilities and my decisions. I love you all.

Contents

List of Figures

List of Tables

Acronyms & Abbreviations

1D One-Dimensional

2D Two-Dimensional

3D Three-Dimensional

6D Six-Dimensional

12D Twelve-Dimensional

AO Asymptotically optimal

ASV Autonomous Surface Vessel

AUV Autonomous Underwater Vehicle

Dec-MCTS Decentralised monte carlo tree search

Dec-POMDP Decentralised partially observable markov decision process

dRRT Discrete rapidly-exploring random tree

DSLX Discrete search leading continuous exploration

DSTG Defence Science and Technology Group

EAC East Australian Current

FIFO First–in–First–Out

FMT Fast marching tree

KPIECE Kinodynamic motion planning by interior-exterior cell exploration

MCTS Monte carlo tree search

PF Piecewise-constant function

PRM Probabilistic roadmap

RRT Rapidly-exploring random tree

SBL Single-query, bi-directional, lazy in checking collision

TDSP Time-dependent shortest path

UCB Upper confidence bound

UTS University of Technology, Sydney

Chapter 1

Introduction

Many outdoor vehicles are subject to external disturbances that can be modelled as flow fields. Examples include aircraft [1–3] and thermal gliders [4–6]. Marine robots, including autonomous surface vessels (ASVs) [7], underwater gliders [8–10], autonomous underwater vehicles (AUVs) [11], and hybrid variants [12], also operate in a flow field environment to perform long-duration missions. Lucrative and/or scientific missions such as monitoring and surveying [13], oil and gas exploration [14], and underwater surveillance [15] would use the marine robot to operate over a vast ocean environment for months at a time. There is also the potential for multi-robot implementation [16] to fulfill these missions cooperatively. Typical operations have the users manually input waypoints that the robot follows to the best of its ability with local controllers [17, 18].

Ideally, the robot should operate with either energy-optimal or time-optimal paths across the flow field environments, depending on the objective. For missions with one objective, it is ideal for minimising energy expenditure, as it is difficult to extract and refuel while in operations. For missions with multiple objectives with a time budget, it is better to minimise operation time to complete as much objective as possible in one deployment. However, the standard operation method of an operator naively picking mission waypoints cannot guarantee either energy-optimal or time-optimal path solutions. More critically, this method also cannot guarantee a feasible path solution, which forces the users to monitor the marine robot for the entirety of the long mission duration.

FIGURE 1.1: A Teledyne Slocum G2 underwater glider, deployed during many field trials.

The problem of finding an energy-optimal or time-optimal paths in flow fields is a known problem called Zermelo's navigation problem [19] that has been open for more than 80 years. The vision of this thesis is to address Zermelo's problem and other aspects of glider navigation such that optimal operation in a dynamic flow field environment is possible.

1.1 Underwater glider missions

The scope of this thesis focuses on *underwater glider* operations in oceanic environments. Stommel first proposed the concept of a special class of AUV in 1989 that does not use active propulsion to move forward [20]. One version of the underwater glider, called the *Teledyne Slocum G2* glider, is shown in Fig. 1.1. In principle, underwater gliders [21, 22] operate by cycling its buoyancy using a ballast tank and pump to generate a sequence of lift and sinking actions. Then the motion is projected into a sawtooth motion by shifting the centre of mass to change the angle of attack, generating forward motion. This passive control approach allows underwater gliders to operate energy-efficiently compared to other AUV classes and thus is appealing for deployment in long-duration missions.

One detractor for deploying underwater gliders is that they operate with slow velocity relative to the ocean flow fields. Therefore, the glider must rely on exploiting or intelligently navigating the surrounding flow field environments to reach its goal, whereas other AUVs could drive through. As such, Zermelo's navigation problem is more relevant for glider navigation, as it influences the glider path more than any other AUVs. While there is plenty of research on underwater glider navigation [23–25], much of the literature does not address 3D and/or time-varying flow field environments. In this thesis, we are motivated to solve Zermelo's navigation problem for an underwater glider in various classes of oceanic environments, such as time-invariant 3D, time-variant 2D, and time-variant 3D. We consider both energy-optimal operations, to capitalise on the gliders already energy-efficient build, and time-optimal operations, to minimise the demerit from the glider's slow speed.

1.2 Optimal glider navigation

The challenge of both energy-optimal and time-optimal underwater glider operation can be broken down into three sub-problems: complex dynamics, Zermelo's problem, and navigation in a dynamic environment. We address the difficulties in addressing each sub-problem.

1.2.1 Complex glider dynamics

The underwater glider operates over complex dynamics, due to the unorthodox method of forward motion [26]. Their dynamical models are typically non-linear, and the control space is too large to find a feasible solution promptly. Consequently, both the glider's energy and time cost function is also non-linear. Therefore, any oversimplified representation of the glider states may call into question the integrity of any control solution found.

1.2.2 Zermelo's navigation problem

As addressed earlier, optimal navigation in a flow field environment has remained opened for over 80 years. Observing the real-world ocean current, such as the East Australian

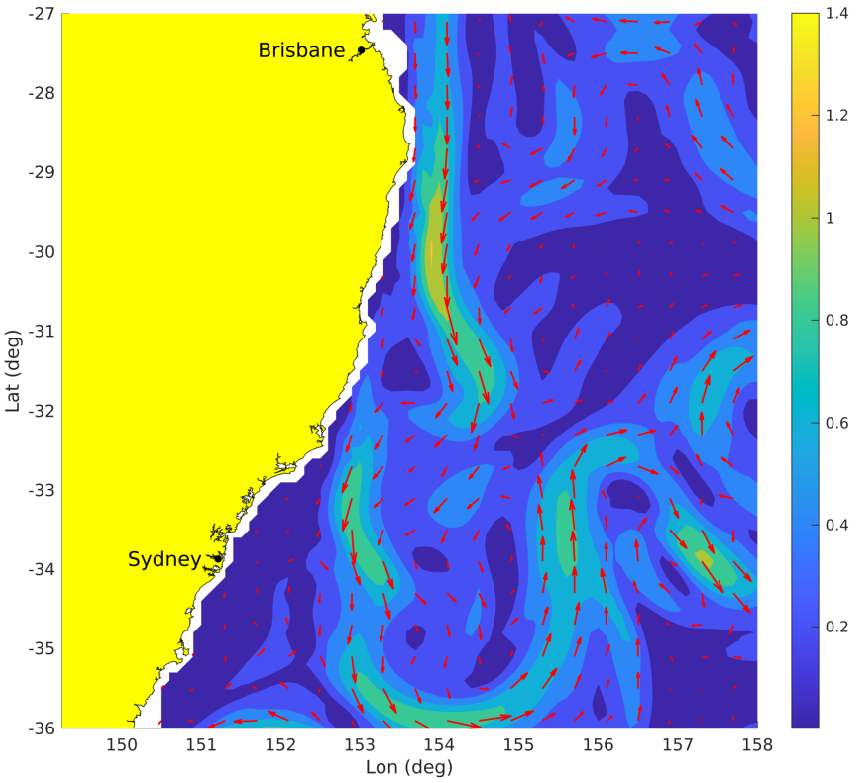


FIGURE 1.2: The ocean model of the East Australian Current. Colormap represents the current strength.

Current (EAC) shown in Fig. 1.2, the reasoning is apparent; flow field dynamics are a composite of linear and non-linear systems. That is, optimising the initial sequence of waypoints along a prolific stream will yield a different result to optimising a waypoint sequence near calmer and more uniform flow regions. When navigating across such an environment, Zermelo's problem becomes a non-linear optimisation problem. The underwater glider's operating speed is significantly slower than the average ocean current magnitude and thus more affected by it than other marine vehicles. Addressing Zermelo's problem for glider navigation is therefore critical.

1.2.3 Zermelo's problem across dynamic environment

The real-world flow field environments can change significantly in a span of a few days. For long-duration missions, this will significantly affect the quality of the solutions when compared to solutions that assumed time-invariant flow fields. The addition of the time

component to the environment adds further complexity when addressing Zermelo's problem. Furthermore, flow fields can exhibit behaviours where the flow benefits the mission objective at a later time. This is different to a First-in-First-out (FIFO) case, where earlier mission start time benefits the mission objective [27]. Optimal path planning for a non-FIFO case is known to be difficult [28, 29].

1.3 Approach to optimal glider navigation

1.3.1 Energy-optimal path planning for underwater glider in time-invariant flow fields

We address the challenge identified in Section 1.2.1 by modelling glider operations with the *trim-based model* [21]. This model reduces the complex glider dynamic to a more parsimonious 6D kinematic model without losing its dynamic properties, such as the sawtooth depth profile and a better representative cost function. We address the challenge identified in Section 1.2.2 by presenting a sampling-based planner framework that incorporates the glider trim-state model. The resulting framework plans for an energy-optimal underwater glider path across time-invariant flow field environment with realistic dynamics. We demonstrate the planning framework by simulating across analytical flow fields and the real-world EAC dataset. Both show the glider maneuvering with an irregular sawtooth profile that minimises the glider energy cost while considering the changes in the flow field and bathymetry.

1.3.2 Minimum travel time problem for discrete graph

We approach the challenge identified in Section 1.2.3 as a time-dependent shortest path (TDSP) problem. The problem is to find the time-optimal path across a time-dependent directed graph where the edge cost is a function of time. From an underwater glider navigation perspective, the edge cost reflects the glider travel time to reach a waypoint starting from different departure times as the ocean current changes with time. Our approach serves as an algorithmic framework for a special case of the non-FIFO TDSP problem,

where the edge cost is expressed as a piecewise-constant function. The time-optimal discrete path is evaluated by solving for the optimal transition policy to the goal at each graph node. We test the algorithm in a non-FIFO graph case and a non-FIFO flow field case, the latter evaluated by representing the environment as a time-dependent directed graph. The resulting policy made use of cyclic paths to delay for better transitioning conditions to reach the goal. The same policies can also query path solutions for different departure time and find the optimal departure time using the standard optimisation method.

1.3.3 Minimum travel time problem for continuous path

We extend the approach presented in Sec 1.3.2 by synthesising a continuous vehicle path under the influence of the flow field environment across the discrete path solution. The path is planned hierarchically to avoid searching through all continuous glider paths in the oceanic flow field environment. First, we evaluate the time-optimal discrete path from our algorithmic framework, then synthesise the continuous vehicle path in a flow field environment that follows the time-edge cost of the discrete path. We demonstrate synthesising the underwater glider path across the discrete solution using the trim-based model. The resulting continuous path demonstrates 3D sawtooth motion synthesised from a 2D graph solution. We also applied this planner with a real-world ocean dataset and highlighted the importance of accounting for the time-varying nature of the flow field.

1.4 Contribution

This thesis contributes novel theoretical insights and corresponding algorithms sufficient to enable efficient optimal motion planning for underwater gliders in dynamic flow field environments. The significance of this contribution is to take a step towards the maturation of underwater gliders as a reliable autonomous field robot platform. Specific contributions are as follows:

- **Energy-optimal path in a static flow field with complex underwater glider dynamics**: The proposed sampling-based framework addresses Zermelo's problem

by guaranteeing an asymptotically optimal path across 3D flow field environments. Planning with the trim-based model of an underwater glider also demonstrates the reduction of the complex glider dynamics while retaining its dynamic properties by naturally generating the required sawtooth depth profile.

- **Travel time-optimal path in a non-FIFO TDSP framework**: Presents a solution to the special case of the non-FIFO problem, under the assumption that the edge cost of the time-dependent directed graph is piecewise-constant. The new proposed framework solves for the optimal policy across the time-dependent directed graph in polynomial time. This result is the first to demonstrate this property for this special case.

- **Travel time-optimal path through dynamic flow fields with complex glider dynamics**: The proposed hierarchical planner addresses Zermelo's problem across time-dependent flow field environments. We demonstrate that this planner can solve a time-optimal path across a time-dependent flow field in polynomial time by formulating the problem as the piecewise-constant variant of the TDSP problem.

There have not yet been any field trials using our approach. However, after completing a series of field trials within our broader program of research [30] to learn how the glider is deployed in real life, we believe the thesis presents a strong argument regarding the severe limitations of current glider practices. The standard glider operation uses heuristically chosen the sawtooth profile regardless of the flow field and only deviates from it to avoid obstacles. As such, the glider control parameters are limited. Our approach presents the energy-optimal glider path as an irregular sawtooth that adapts to its flow field environment. Such control is only possible by allowing direct control of the glider's moving mass and ballast pump.

Underwater gliders perform their mission by following a sequence of waypoints chosen by a human operator. The glider is always submerged when transitioning from one waypoint to another and only resurfaces after arriving at its waypoint to receive further communications. As such, the glider effectively travels blind. Furthermore, an inspection of the real-world ocean forecast data shows that the flow field environment changes significantly

over a few days. Our method uses a hierarchical planner that solves for time-optimal glider paths across a dynamic flow field. By allowing the planner to pick the waypoints instead of a human, we can guarantee an optimal glider path across a dynamically changing flow field.

1.5 List of publications

This thesis represents the culmination of past publications on our research. Each approach listed in Section 1.3 are either published or archived work. The approach to Zermelo's problem for underwater gliders using an asymptotically optimal framework with trim states is presented as a conference paper in [31], and has since been applied in glider mission scenarios [10] and bulit upon through further algorithm development [32]. Both [31] and [10] won best student paper awards. The approach to the time-dependent flow field navigation using a special-case TDSP framework is archived in [33] for later publication and a journal version is currently under preparation. This work was then extended to consider continuous underwater glider paths and published in [34]. Journal versions of [31] and [34] are also currently in preparation.

1.6 Thesis Outline

The thesis is organised as follows:

Chapter 2: Presents related work.

Chapter 3: Provides background materials regarding underwater gliders, motion planning, and time-dependent shortest path problem. Also define the problem statements.

Chapter 4: Presents trim-based framework.

Chapter 5: Presents a special TDSP framework for navigation in non-FIFO flow fields.

Chapter 6: Presents a hierarchical planner that synthesis continuous path across the discrete TDSP solution.

Chapter 7: Presents simulations using real-world dataset of the East Australian Current. Also provides extension on the hierarchical planner for future real-world applicability.

Chapter 8: Concludes the thesis with a summary of what we addressed, and discusses potential future work.

Chapter 2

Related work

Work related to this thesis is drawn from several sub-disciplines. This chapter discusses related work in glider navigation, finding shortest paths in time-dependent graphs, and robot motion planning.

2.1 Underwater glider navigation

Underwater glider operation requires 3D manoeuvre to propel forward [18, 35]. Despite this dynamic constraint, most related work approaches the motion planning problem for gliders by assuming a 2D kinematic model with directly controllable velocities [36–38]. The objective of the glider is often to minimise time or energy. In most cases, energy-cost is modelled as a linear function of time. Therefore, both energy-optimal and time-optimal path solutions can be achieved by maximising the resultant forward velocity from both vehicle dynamics and the currents. From this standard practice, an argument can be made that all contributions regarding flow field navigation can apply to optimal glider operation. As such, this section will address all approaches to flow field navigation.

The problem of optimal underwater glider navigation in a flow field environment has been approached with various motion planning frameworks. A popular approach to flow field navigation are graph-based planners that constructs a graph that represents the flow field environment. Various graph methods include uniform discretisation [23, 25, 39], biased

discretisation using forward integration [37], adaptive discretisation [40], and building a probabilistic roadmap [41] or a graph tree [23, 42]. A well-known common property of graph-based methods is the trade-off between performance and computational cost. Other planning frameworks includes iterative optimisation [17, 18], control parametrisation [43], and parallel swarm search [24].

These frameworks can also plan for dynamic flow field environments by expanding the graph into spatial-temporal graphs [24, 37, 40] by duplicating the nodes per unit time. Recent work on optimal path policy have also formulated the dynamic flow field navigation problem as a time-varying Markov decision process [44].

There also have been various control approaches to optimal glider operations. These methods includes using PID controllers [45], minimum time feedback control using the Hamilton-Jacobi-Bellman equation [46], optimal steering policy [47], Lagrangian heuristics control [48], reduced order models [49], and streamline control [50]. The control methods, along with genetic algorithm [38] and the level set method [51–54] are computationally expensive, making them unsuitable for application in real-world mission settings.

Studies into underwater glider path planning in 3D flow fields are surprisingly rare. Two extant studies use differential evolution that assumes directly controllable turning rate [55], and the generation of a Dubins path made from a series of "sawtooth" and spiral motions without the influence of flow fields [35]. Both papers also assume that the glider always operates in a fixed "sawtooth" motion.

2.1.1 Zermelo's navigation problem

Planning optimal underwater glider paths across flow fields addresses a famous open problem known as Zermelo's navigation problem [19]. Zermelo showed that naively driving towards the goal was not optimal. Instead, the ship needs to account for the drift from the flow field and choose its heading accordingly. Aircraft navigation problem commonly addresses Zermelo's problem, focusing on handling perturbations and external disturbances [56], planning for multiple vehicles [57], accounting for obstacles [58] and balancing

between multiple objectives (time and energy) by adjusting the vehicle speed [59, 60]. Zermelo's problem was also considered to investigate the migration patterns of turtles [61].

Analytic solutions exist for simple one-directional flow field cases using vector calculus. In practical settings, flow fields behave more irregularly, and are commonly modelled using the Navier-Stokes equations [62] or the Taylor-Green equation [63]. The vehicles deployed in these environments often have sophisticated models and cost function that are non-linear, requiring a numerical solution to the optimal control problem. As such, Zermelo's problem has been left open for more than 80 years.

For underwater gliders, Zermelo's problem is further complicated as they operate with a mix of discrete variables and continuous variables; for example, the glider needs to switch between different discrete modes of ballast operation. This mixed-discrete continuous problems can be formulated as mixed-integer (non)-linear program (MILP) or mixed-integer geometric programs (MIGP) [64–67], but are computationally expensive to solve.

2.1.2 Our approach

The oversimplification of the glider dynamics is mainly due to the complexity of controlling its pose and velocity. The Slocum underwater glider [21, 68] is modelled with 12 degrees of freedom. The glider's orientation is controlled by adjusting the moving mass, and its net buoyancy is controlled by pumping water in and out of ballast tank. Both the ballast state and the glider orientation determines the net glider velocity and is expressed as a non-linear function. Other glider models share this control method where the pose of the glider is not directly controllable [69, 70].

Since energy consumption is partially based on the mechanics/hydraulics of this control system, a simple energy cost model (i.e., energy consumption is a linear function of time) may be misleading and could cause failure at the limit of the glider's energy capacity. Therefore, we consider that accurate modelling of the glider dynamics is mandatory for planning. The simplification of the glider model also reduces the flow field environment to a 2D plane. The gliders are either planned over the surface current [23, 36] or the depth-average currents [18, 37, 38]. Indeed, it is standard practice to use depth-averaged

currents for planning underwater gliders [71]. However, this remains a poor assumption that removes information the planning algorithm can use to exploit the ocean environment fully. For example, planners could compare stratified ocean currents at different depths and choose the "best" depth to operate in. [72].

In [31], we propose *trim-based kinodynamics* for underwater gliders, where a trim state is a state of equilibrium that the glider will maintain in the absence of active control. We showed that the trim-based dynamic model is an accurate representation with reduced control dimensions that preserves important physical and control properties of the glider. We also demonstrated how to generate an asymptotically optimal minimum energy path in a 3D flow field environment using this trim state assumption. Chapter 4 explains this work in more detail.

2.2 Time-dependent shortest path

This section addresses the time-dependent shortest path (TDSP) framework that evaluates time-optimal paths in a dynamic environment. A common issue with searching in the spatial-temporal graph is that the solution is resolution complete in time; paths' arrival times at nodes must align with the given time discretisation. Planning with spatial-temporal graphs increases the size of the problem representation and implies a corresponding increase in required computation time. In a dynamic flow field environment, the worst-case computation time that makes use of waiting tactics (such as cyclic motion) for better flow field conditions is non-polynomial.

A useful framework that solves for dynamic environments without using the spatial-temporal graph is the TDSP framework. Instead of extending the graph across the time dimension, the graph edge cost is evaluated with a time-dependent cost [27, 73]. One important property of the TDSP problem is whether the time-dependent graph has the *First–in–First–Out* (FIFO) property; i.e., delaying departure from any node can never result in earlier arrival. Both Djisktra [74] and A* [75, 76] can be modified to be implemented on a time-dependent graph with FIFO properties. Other algorithms that address the FIFO problem includes using precomputed heuristics [77], optimising Bellman-Ford

algorithm across the time-dependent graph [78] and bidirectional search [79–81]. A FIFO TDSP framework that solves for general cost was also presented [82]. Related work regarding non-FIFO problem is rare. One work approach the problem using a variation on the time-dependent graph called the "time-aggregated graphs" [83], where edge functions are represented as time series.

The computational complexity of FIFO problems with piecewise linear edge travel time functions is superpolynomial in the number of graph nodes [73]. There is also an existence of polynomial-time special cases where the slopes of the travel time functions are restricted. For non-FIFO problem, if waiting is allowed, then it is still solvable in polynomial time. If not, then it's NP-hard [28, 29]. We show in Chapter 5 that this bound also holds for finding minimum travel time paths in non-FIFO problems with piecewise constant edge functions, and present an algorithm with a straightforward implementation. The main distinction is that shortest paths in non-FIFO problems (that do not allow arbitrary waiting) may include cycles. Our analysis essentially bounds the length of cycles by relating the worst-case cycle time to properties of the edge functions.

2.2.1 Our approach

We propose an efficient approach to optimal planning in time-dependent flow fields expressed as time-dependent graphs, which is solved in polynomial time [33, 34]. The algorithm can correctly find cyclic time-optimal paths in pathological flow regimes. We also build on previous work that synthesises a continuous path given a sequence of discrete path states for time-invariant wind fields [1, 34].

2.3 Motion planning

This section presents an overview of general motion planning algorithms. Motion planning is a problem of determining the path a robot should take to reach some goal state. The solution path is *feasible* if the path is collision-free, that is, a path that avoids obstacles, self-collision, and satisfying certain constraints. The simplest form assumes a start and goal state, with complete knowledge of the workspace. Early theoretical work showed that

when considering an optimal path that minimises some objective function, also referred to as "shortest" path, is intractable (NP-hard) for a 3D polyhedral environment [84].

One of the first motion planning algorithms was published in 1969 [74]. Since then, there have been over five decades' worth of active research on this topic. The motion planning literature is divided into two main sub-groups: grid-based, and sampling-based approaches. This section provides an abridged overview and discussion for the most relevant robotic literature on modern motion planning methods.

2.3.1 Grid-based planning

The primary motivation behind grid-based algorithms is to discretise the continuous configuration spaces into smaller "cells," with each cell labelled as collision-free or not. This discretisation reduces the motion planning problem down to a sequencing problem between adjacent cells, with the total cost of the path derived as the sum of each cell transition. As most grid-based methods share similar traits, we will first discuss the method in general, and then introduce some of the more well-known methods individually.

The two main limitations of this method are its resolution and complexity. The quality of the solution is mainly dependent on the grid resolution (size of cells), where higher resolution is better than lower resolution. For instance, a sparse grid may fail to capture an obstacle into the configuration space, making the transition between two states unrepresentative. Alternatively, a small obstacle overlapping a cell in a sparse grid can invalidate a significant portion of the configuration space.

On complexity, the implication from the grid-based algorithm's resolution issue shows that it is resolution complete; a solution is found if and only if a solution exists given the grid abstraction. However, generating a high-resolution grid is computationally expensive and cost increases exponentially with the increase in dimension size. This "curse of dimensionality" limits the general usage of the grid-based algorithm to three dimensions or fewer in practice.

Despite these flaws, grid-based methods are an appealing, though naive, approach to the motion planning problem. These algorithms are easy to implement and offer sub-optimal

solutions that improve with grid resolution. The two most famous algorithms that use the grid-based approach are Dijkstra and A*.

Dijkstra's algorithm [74, 76] is commonly used in robot motion planning by generating a grid graph, where an edge is connected between each adjacent pairs of cells. Connection strategies include the 4-connected grid (North, South, East, West), or 8-connected (4-connected plus diagonals). Each edge is associated with a cost to traverse, which is defined by an objective function. Dijkstra then exhaustively expands the optimal path from the initial state to all other states in the graph until it reaches its goal point. The algorithm produces an optimal solution for the grid density but is computationally expensive to solve, particularly for higher dimensions.

*A** is a heuristic variant of Dijkstra's algorithm [75, 76]. Unlike Dijkstra, A* uses an estimated cost-to-arrive towards the goal as heuristics to bias the expansion order. As such, A* has the distinct advantage of performing faster than Dijkstra by leaving large areas of search space unexplored. The heuristic function must be admissible (underestimate the real cost) to guarantee an optimal solution path. A*'s computation is still affected by the "curse of dimensionality" as states still need to be partitioned. The heuristics estimate can also tune the optimality of the result. For instance, by setting the heuristic estimate to zero, A* degenerates back into Dijkstra. Overall, A* is appealing in robot motion planning due to its ease of implementation in practice.

2.3.2 Sampling-based planning

The idea of the sampling-based approach is to take random samples from a set of collision-free configuration space and use those points to generate a collision-free path. Its motivation is to avoid rigorous scanning of free configuration space, as they would do in a grid-based method, thus freeing computation to allow planning in higher-dimensional space. The sampling-based method can ultimately be categorised into the following: multi-query, and single-query. This section will discuss those two categories, followed by a discussion for some of the more recent state-of-the-art methods, such as Asymptotically Optimal (AO) algorithms.

Multi-Query

One of the first major multi-query algorithms was the probabilistic roadmap (PRM) [85]. The algorithm builds a graph (roadmap) of sample vertex and edges by having each vertices connect to its collision-free neighbour, up to some bound radius. The roadmap is then stored in some database for future offline planning. A path is generated by connecting the desired start and goal vertex to the stored roadmap, then finding a path from start vertex, through the roadmap, to the goal vertex using a local planner, like Dijkstra. Its major advantage is that generating a path for a difficult, high dimensional problem is fast as it relies on preprocessed information. Unfortunately, the algorithm generates a feasible path, not optimal. Also, the success rate is dependent on the quality of roadmap (determined by the radius), with the construction of the roadmap being computationally expensive itself.

Since the PRM algorithm was proposed, there have been many other variations to address various shortcomings. Lazy PRM [86] minimises the roadmap construction time by using lazy collision checks when building the roadmap. Dynamic PRM [87] uses the roadmap to plan a path in a dynamically changing environment with static and moving obstacles. Stochastic PRM [88] presents a PRM framework that considers uncertain vehicle motion and plans a path that maximises the probability in successfully reaching the goal.

Single-Query

The first major single-query algorithm was the rapidly-exploring random tree (RRT) [89]. The algorithm grows a tree of graphs from the start vertex as its root. Branches of the tree are made by randomly choosing a point within the free configuration space, then placing (and connecting) a vertex between the point and its nearest vertex (defined by some increment). The growth stops when one of the branches is close to the goal vertex, and a path is connected. This method of generating the tree minimises the number of collision checks, which subsequently reduces computational cost. However the path is only feasible, like PRM.

One issue with RRT-based planning, especially with its early variants, is that it showed difficulties in handling bug-trap planning environment [90, 91]. Variations like RRT-connect [92] and Triple-RRT [91] addresses this problem by growing multiple trees of graphs at different locations.

Asymptotically Optimal (AO)

The solution quality for PRM and RRT has been left as an open question for almost a decade. Many variations presented to either reduce computation time or improve the quality of the solution. However, in 2011, Frazzoli and Karaman rigorously analysed the asymptotic behaviour of PRM and RRT and showed that it does not converge to an optimal solution [93]. In response, they proposed two new variants (PRM* and RRT* respectively) that guarantee the path solution converges to an optimal solution as the sample size increases.

For PRM*, the significant change was to make the radius dependent on the sample size. This approach was intuitive since the radius did affect the quality of the roadmap. For RRT*, the significant change was that: once the new vertex is added, that vertex attempts to connect to all other nearby vertices within some radius similar to PRM*, and only accept the shortest edge. The radius function was chosen such that it will decrease the number of samples, where the decreasing rate matched sample dispersion rate. While this made the algorithms AO, the computational cost was similar to that of the original algorithms.

These algorithms have become the new standard for motion planning algorithm, and have inspired variations to address various problems. RRT$^{\#}$ [94] achieves AO solutions with faster convergence rate than RRT* through better exploration and exploitation methods when growing the tree. RRT^{x} [95] demonstrate AO solutions where the graph tree is refined and repaired to account for dynamic obstacles.

In 2015, FMT* was introduced as an improvement on PRM* and RRT* [96]. In the simplest term FMT* was a marriage between PRM* and RRT*: samples are generated in the free configuration space like PRM*, but would branch to each vertex like RRT*. However, different to RRT*, dynamic programming is used to branch, by having the vertex

with the shortest leaf-to-root path have priority in branching. Also, FMT*s collision checking is lazy and will only perform one collision check per branch, reducing overall collision-checking and subsequently reducing the computational cost. In tests that looked into various dimensional problems (2D to 7D), FMT* converged to a solution faster than PRM* or RRT* (tested over 50 trials). Higher dimensional problems saw a more significant improvement, and in situations where collision checks would have been expensive.

2.4 Edge Evaluation

As planning algorithms are increasingly used in practical systems, the computational bottleneck imposed by collision checking is starting to be addressed by researchers. Navigation in flow fields can often forgo collision checks as planning environments are uncluttered. However, an analogue of expensive collision checking is expensive edge evaluation. In order to determine the time, energy, or distance cost of an edge, the vehicle model must be invoked. For an underwater glider operating in a non-linear flow field, this model can become very complex and is often required to be solved numerically [37, 39, 42], which in turn introduces a computational bottleneck.

There are many approaches to mitigating the issue of slow edge evaluation. One common approach is lazy collision checking, as utilised in lazy PRM [86], lazy weighted A* [97], and FMT* [96], that only does edge evaluation as needed. Algorithms like KPIECE [98] and DSLX [99] reduce the number of edge evaluations using a discrete search strategy. The Euclidean workspace is discretised into regions, and each region cell provides information to evaluate exploration and appropriately bias the path search accordingly. Another approach aside from reducing the number of edge evaluations is to parallelise the state expansions, and therefore avoiding multiple edge evaluations [100].

We have recently developed an approach in mitigating slow edge evaluation for planning in flow field environments. This approach reduces the number of control instances the edge evaluation enumerates using the streamline function [41].

2.4.1 Hierarchical Planner

A different strategy to reduce the number of slow edge evaluation in path planning is by approaching the problem hierarchically. The planning strategy is typically broken into two parts and executed in sequence: *high-level* planning that generates a rough path in a smaller dimensional representation of the workspace, and *low-level* planning that generates a more specific kinematically constrained path. As a feasible solution most likely lies in the rough path solution, the low-level planners bias their path search to the high-level planner framework, thus avoiding searching the entire workspace and limiting the number of edge evaluations performed. A common approach is to bias the tree-based planning algorithm, such as RRT*, to the result of a discrete path planner. Often the high-level planner discretises the environment into lattice generated from Voronoi plots and/or grids [101–104], but a skeleton of a 2D workspace environment has also been used [105].

One detriment to using hierarchical planners is that unlike KPIECE, where the discretised workspace aids the path search, the low-level planning quality depends on the solution from the high-level planner. As such, information held by the high-level solution significantly impacts the possible solutions available to the low-level planner. A comparison between biasing the RRT* path with a geometrically feasible discrete path versus a kinodynamic discrete path showed that accounting for kinodynamic model during high-level planning yielded better results [102]. On the other hand, the planning architecture of the hierarchical planners allows low-level planners to be modular while still keeping the high-level planning solution. For example, the rough terrain planner proposed by Brunner [103] had two approaches for low-level planning that built off the same high-level framework depending on whether the environment was flat or rough.

2.5 Multi-robot coordination

There is a growing interest in the multi-robot application of gliders. It is often advantageous to deploy a team of heterogeneous gliders for monitoring and tracking mission on a massive scale [16, 106, 107]. One field trial report suggested that, for surveying 100km x 100km ocean environment, a naively deployed fleet of 8 gliders performs similar

to sampling with 10km resolution [108]. The general multi-robot coordination problem has been well-studied over many decades [109, 110] and can largely be divided into two categories: centralised and decentralised. This section introduces a few relevant examples of multi-robot algorithms, serving as a potential application of the fundamental result of this thesis and thus help motivate future work.

The centralised approach coordinates multi-robot systems by searching through the entire joint state space between each robot in the team. Basically, a central computer communicates and makes decisions for every member of the robot team. Previous literature has featured a sampling-based approach for centralised multi-robot coordination. The SBL planner [111] for instance, is a variant of the PRM algorithm that uses a single query bi-directional sampling strategy with lazy collision checks. The discrete-RRT algorithm (dRRT) [112] is a variant of the RRT algorithm that is also applicable for multi-robot coordination. It grows discrete trees using precomputed heuristics about the neighbours of visited states. For multi-robot planning formulation, both algorithms sampled across the combined configuration space between all robots in the team, which at worst grows exponentially with the number of deployed robots.

Another promising centralised approach is Monte Carlo tree search (MCTS) [113]. The MCTS algorithm works by growing a tree from a root node that represents the initial state. The tree expands by selecting a node with the largest *upper confidence bound* (UCB); UCB represents the approximate probability of finding a better plan from a node. The selected node expands by choosing an unexpanded action available and creating new nodes from it. Each expanded node then performs a process called *rollout* that generates a sequence of actions to the end of the mission (e.g., time budget) using a policy. The prefix (i.e., from root to expanded node) and suffix (i.e., from expanded node to the end of rollout) are combined to find a full action sequence. A score is evaluated for this sequence, and it *backpropagates* from the expanded to root node. The backpropagation step influences the next selection process. MCTS with a sequential allocation framework was also proposed for ocean front monitoring and tracking mission using a fleet of gliders [106].

Although MCTS is suitable for large state space problems with non-trivial objective functions, its complexity grows exponentially in the number of agents when used for multi-robot

systems. In the multi-robot case, a node in the MCTS tree is a set of all agent states. Likewise, the joint action space grows exponentially. For this reason, MCTS is not suitable for multi-robot systems with a large number of agents.

On the other hand, the decentralised approach searches state space for each individual robot in parallel and updates the search when they can communicate with each other. In other words, each robot makes decisions for itself and communicates with others without a centralised oversight. Many decentralised algorithms are an extension of previously existing algorithms for decentralised frameworks, such as Dec-POMDP [114, 115] and the decentralised sequential greedy approach [116]. Recently, an extension to the MCTS algorithm for the decentralised framework was developed, called Dec-MCTS [117]. Each robot in the team generates its own MCTS tree while considering the actions of others and updating the trees when the robots communicate. Since the computation is performed on each robot without considering the joint state and action spaces, the computational complexity is significantly reduced compared to the centralised MCTS.

2.6 Summary

This section introduced an overview of the current research in motion planning, flow field navigation in both static and dynamic flow fields, and time-dependent shortest path planning. A common limitation observed from the related work is that: 1) the path planning algorithms assume an over-simplified glider model, and 2) the approach for solving an optimal path across a time-varying flow field is either resolution-complete in time or solved in non-polynomial time. This thesis addresses these limitations and provides analysis and examples. The limitation of over-simplifying glider dynamics to plan optimal paths is addressed in Chapter 4, and the computational limitation on dynamic flow field navigation is addressed in Chapter 5 and Chapter 6.

Chapter 3

Background and problem statement

In this chapter, we present background materials for the thesis content and introduce the problem statement. Section 3.1 illustrates the underwater glider model that is considered when planning. Section 3.2 provides a summary of the various sampling-based algorithms used, including PRM, RRT and FMT*. Section 3.3 introduces the time-dependent shortest path (TDSP) problem. The section also introduces the concept of arrival and travel time, and describe the conditions and properties of FIFO and non-FIFO conditions for a TDSP problem. Sections 3.4 defines the piecewise-constant function and its evaluation. In Section 3.5, we define the overall problem for the thesis as a whole.

3.1 Underwater glider model

Underwater gliders operate by cycling their buoyancy to generate a sequence of lift and sinking forces. The glider's orientation translates this into the desired forward velocity, as shown in Fig. 3.1. Both the pitch angle θ and the angle of attack α can be derived from the desired elevation angle of the glider velocity vector, which we call glide angle γ. The equation that relates these terms can be found in Appendix A.

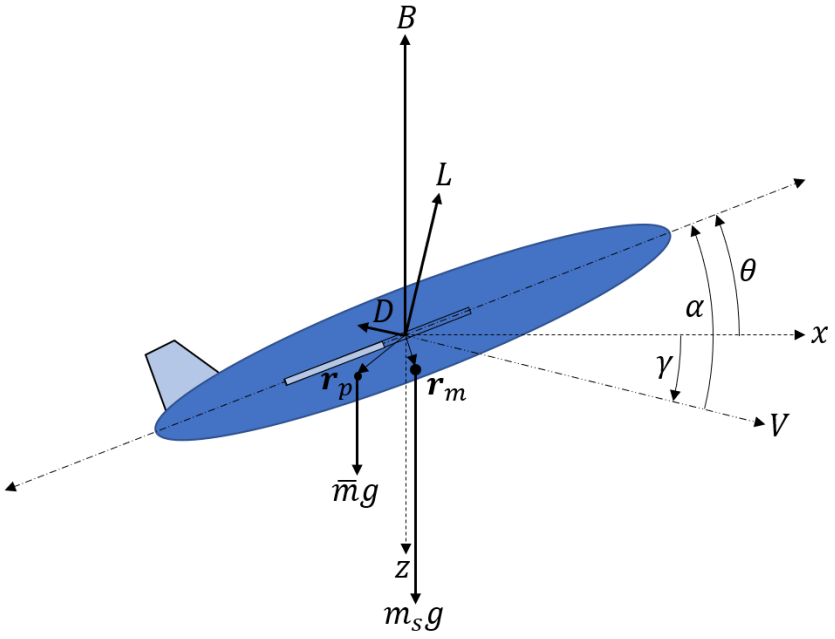


FIGURE 3.1: An approximate representation of the forces and moments acting on the glider in the inertial reference frame (i.e., centre of buoyancy). The hydrostatic forces consist of the buoyancy force B, weight due to moving mass $\bar{m}g$, weight due to stationary mass $m_s g$. The hydrodynamic forces consist of lift L, drag D and the pitching moment M_{DL}. The moving mass is offset from the nominal centre of gravity by a vector r_p. The glide angle γ, pitch angle θ and angle of attack α represent the orientation and direction of the glider with velocity V.

An underwater glider controls its pitch angle by changing internal moving mass $\mathbf{r}_p(t) = [x_p, y_p, z_p]^T$ to shift the centre of mass, as shown in Fig. 3.1. The buoyancy is controlled by filling and emptying the ballast tank with surrounding sea water, where filling it decreases the buoyancy and sinks the glider, and emptying it increases the buoyancy and raises the glider. Filling and emptying the ballast tank also increases and decreases the ballast mass, m_b. Since the centre of mass is routinely shifted, it cannot be used as the glider reference point. Instead, the centre of buoyancy derived from the external shape of the underwater glider is used as the reference point. By design, the ballast tanks are symmetrically distributed around the centre of buoyancy.

The scalar magnitude of the underwater glider velocity V_G at some instance can be defined as a non-linear function of γ and m_b:

$$V_G(\gamma, m_b) = \sqrt{\frac{(m_b - (m - m_h - \bar{m})) \cdot g}{-D(\gamma)\sin\gamma + L(\gamma)\cos\gamma}}, \tag{3.1}$$

where $D(\gamma)$ is the drag force, $L(\gamma)$ is the lift force, m is the mass of the sea water that can be displace, m_h is the hull mass, $\bar{m}$ is the movable mass, and g is the acceleration due to gravity [21]. Details on these parameters and functions can be found in Appendix A.2. The glider velocity vector can then be expressed as:

$$\begin{bmatrix} u_G \\ v_G \\ w_G \end{bmatrix} (\gamma, \delta) = \begin{bmatrix} V_G(\gamma, m_b) \cos\gamma \cos\phi \\ V_G(\gamma, m_b) \cos\gamma \sin\phi \\ V_G(\gamma, m_b) \sin\gamma \end{bmatrix}, \tag{3.2}$$

where δ is the yaw of the glider, which we call *heading angle*.

3.1.1 Dynamic model

The dynamic model of underwater gliders at time t can be defined as a 12-dimensional state $\mathbf{x}(t) = [\mathbf{p}(t), \dot{\mathbf{p}}(t), \phi, \theta, \delta, \dot{\phi}, \dot{\theta}, \dot{\delta}]^T$, where $\mathbf{p}(t) = [x, y, z]^T$ is the position vector and δ is the roll angle. The control vector for the underwater glider $\mathbf{u}(t) = [\mathbf{u}_{\mathbf{r}_p}(t), u_{m_b}(t)]^T$ consists of a force vector $\mathbf{u}_{\mathbf{r}_p}(t) = \dot{\mathbf{r}}_p(t)$ acting on a moving mass and the ballast pump $u_{m_b}(t) = \dot{m}_b$ that empties the ballast tank. The pump is not needed to fill the tank, as the surrounding oceanic pressure is sufficient to do so. We denote the continuous sequence of control vectors over time t_1 and t_2 as $\mathbf{u}(t_1 : t_2)$. Given an underwater glider position in the 3D space $\mathbf{p}(t)$, we assume that the velocity vector of the ocean current at $\mathbf{p}(t)$ is given *a priori* such that $\mathbf{V}_c(\mathbf{p}(t), t) = [u_c, v_c, w_c]^T$. If the flow field is time-invariant, it is instead denoted as $\mathbf{V}_c(\mathbf{p}(t))$. The dynamic model can then be expressed as:

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t)) + \mathbf{V}_c(\mathbf{p}(t), t), \tag{3.3}$$

where $f(\mathbf{x}(t), \mathbf{u}(t))$ represents the glider dynamics in still water.Time-invariant flow field can be obtained through ocean forecast and holds for short mission duration.

The glider's energy expenditure arises from controlling the glider (i.e., forces acting on the masses), pumping ballast fluid, and the *hotel load* (e.g., on-board processor and sensors). Given the glider dynamic model, the energy cost over a continuous control sequence between t and Δt starting from $\mathbf{x}(t)$ is stated as $cost(\mathbf{x}(t), \mathbf{u}(t : t + \Delta t))$.

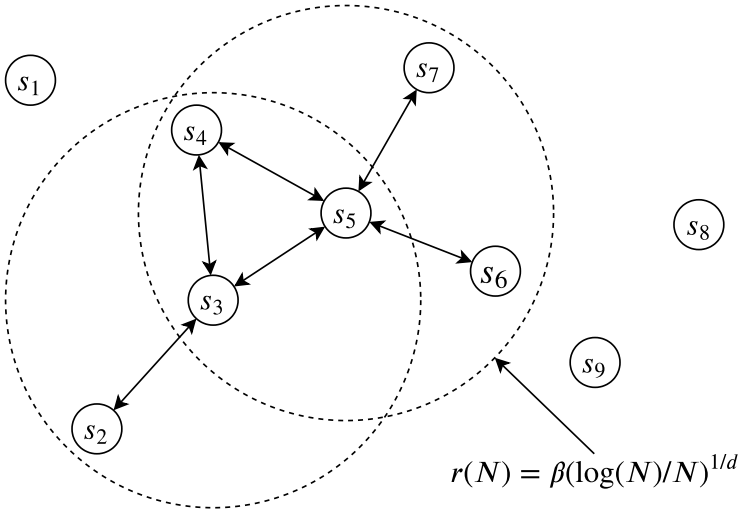


FIGURE 3.2: Partial construction of a PRM* graph. The graph samples nodes in free space and connects edges between nodes and their neighbours. The PRM* defines the neighbourhood of each node by a radius bound.

3.2 Sampling-based algorithm

As we explored in Chapter 2, sampling-based algorithm remains a popular motion planning method as they are less affected by the curse-of-dimensionality compared to grid-based methods. In this section, we provide an overview of some sampling-based methods we use throughout the thesis.

3.2.1 PRM

The PRM algorithm constructs a graph $G = (S, E)$ that consists of finite set of nodes $S \subset \mathbb{R}^d$ and edges $(s, s') \in E$ where $s, s' \in S$. The set of nodes S is populated by randomly picking $N \in \mathbb{N}$ sample points from free space χ_{free} (i.e., a collision-free region in $\mathbb{R}^d$). Edges (s, s') are defined by a pair of nodes where s' is within a user set radius r from s, as shown in Fig. 3.2. The graph allocates a cost to traverse to each edge (e.g., edge distance, time to travel, energy expenditure, etc.). The optimal path from between the chosen start and goal nodes $s_{init}, s_{goal} \in S$ through the graph G is found by implementing a local path planning algorithm, such as Dijkstra [74]. Further details can be found in [85].

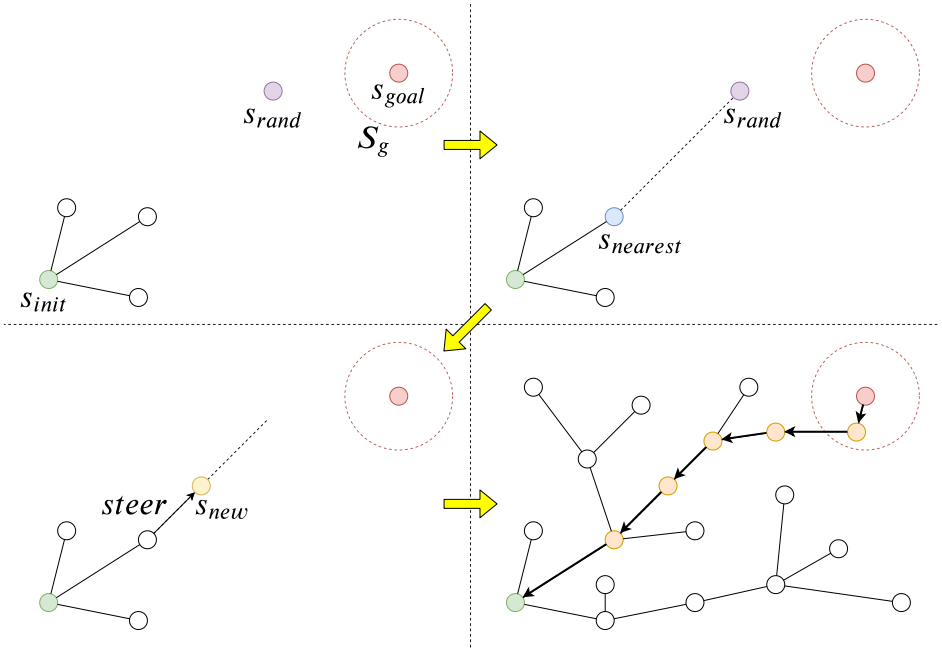


FIGURE 3.3: Steps-by-step demonstration of RRT algorithm. It generates a tree graph by iterating through steps until the graph reaches a node limit. The algorithm samples a random node in free space (top-left corner) and finds the nearest node in the existing graph (top-right). The algorithm then performs a steering function from that node and propagates a new node that is added to the graph (bottom-left). Once the tree graph is complete, a path from s_{init} to s_{goal} is found by following the source of the node, starting from the node that is within the goal region S_g (bottom-right corner).

There is also an asymptotically optimal variant called PRM* [93]. For the problem of shortest distance, the PRM graph is guaranteed to provide better path solution as $N \to \infty$ if the radius r is above the bound:

$$r > \beta \cdot (\log(N)/N)^{1/d}, \tag{3.4}$$

where β is some scaling factor to account. Partial construction of the PRM* graph is shown in Fig. 3.2. For motion planning with different objective functions, such as time or energy, the radius needs to be properly scaled with β to match the unit conversion.

3.2.2 RRT

The RRT motion planning algorithm propagates a tree toward a goal region. We now give a high level description of RRT. A visualisation is shown in Fig. 3.3. Consider a graph

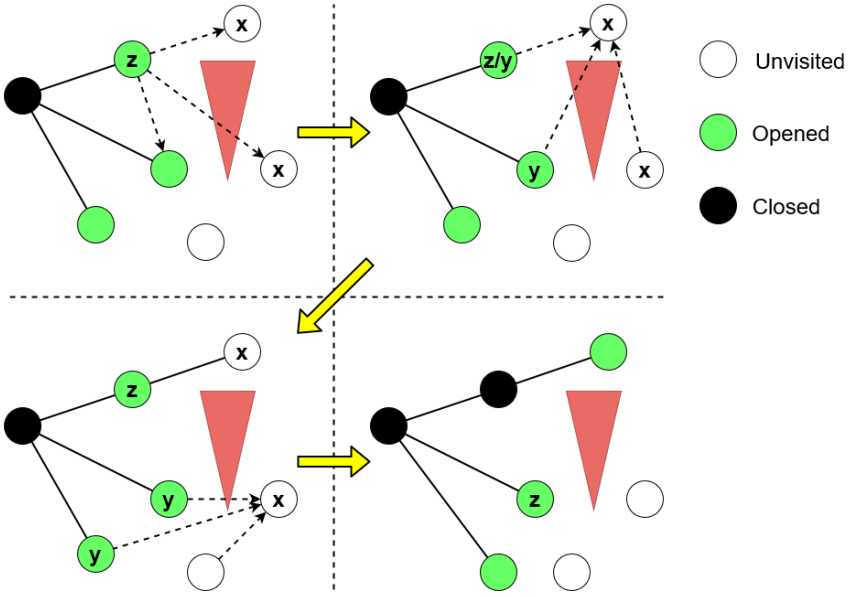


FIGURE 3.4: Partial propagation of FMT* graph. It first samples nodes in free space and propagates a tree graph from the existing sample nodes. From an *Open* node z, it finds the neighbouring *unvisited* nodes. For each of these neighbouring nodes, it finds the best connection out of its neighbouring open nodes. This connection is abandoned if there exists a collision between the two nodes. The algorithm repeats these steps after closing the current open node and allocating a new open node.

$G = (S, E)$ where initially $S = s_{init}$ and $E = \emptyset$. We define a goal node s_{goal} and goal region $S_g \subset \chi_{free}$ such that $s_{goal} \in S_g$. We also define a function $Steer(s_a, s_b)$ that, given a vehicle dynamics, defines a new node after travelling from s_a in the direction of s_b for some time step. We perform the following steps until $|S| = N$: We sample a random node s_{rand} in χ_{free}, and find its nearest node $s_{nearest} \in S$. We then sample a new node s_{new} into S that is generated from the $Steer(s_{nearest}, s_{rand})$ and add the node pair $(s_{nearest}, s_{new})$ to E as edges. The resulting graph is a tree that expands outwards from the initial node. Once N nodes have been sampled, we find the feasible trajectory by picking a node that reached S_g and follow the parent nodes back to the starting node.

3.2.3 FMT*

FMT* is a recent state-of-the-art, asymptotically optimal motion planning algorithm that is effectively a combination between PRM* and RRT*. Like PRM*, the original FMT* algorithm starts by choosing a set of sample points in the state space. Like RRT*, the algorithm incrementally grows a tree that eventually reaches the goal. A visualisation on

the construction of FMT* is shown in Fig. 3.4. The algorithm starts by first labelling the initial state as *Open* while labelling the goal and the rest of the sampled states as *Unvisited*. At a given iteration, the algorithm chooses the open state z with the lowest overall cost so far. It then finds the *Unvisited* samples within a neighbourhood of z with respect to cost, and them as $xNear$. For each $x \in xNear$, the neighbourhoods of x are found in *Open* states, and set them as $yNear$. Then the algorithm picks the state $y_{min} \in yNear$ that minimises the sum of the cost at $y \in yNear$ and the transition cost from y_{min} to x, then checking if this transition is collision-free. Once $xNear$ is exhausted, z is set to *Closed* and any newly connected states in $xNear$ become *Open*. The tree continues to grow in this manner until the goal state becomes *Open* or when no more *Open* states exist. To save computation time; neighbourhoods are cached to ensure that each edge calculation is made at most once.

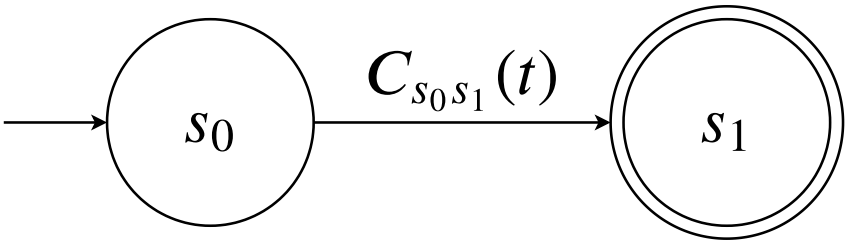
3.3 Time dependent shortest path problem (TDSP)

In the real world ocean environment, the oceanic flow field is *time-dependent* (i.e., changes with time). In this section, we introduce a method that solves for a time-dependent problem called the *time dependent shortest path problem* (TDSP) and discuss how it can be implemented in an unstructured environment like the ocean. We also define terms associated with the TDSP framework, such as arrival time and non-FIFO.

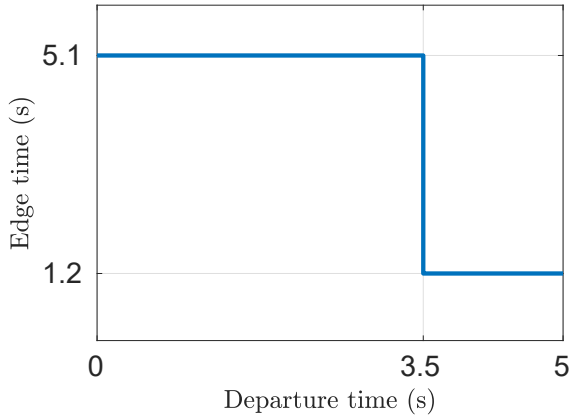
3.3.1 Time-dependent directed graph

We consider a directed graph $G = (S, E)$ that consists of a finite set of nodes S and edges $(s, s') \in E$ where $s, s' \in S$. The set of immediately reachable nodes from node s is denoted as $S_s \subseteq S$. A set of goal nodes is denoted as $S_g \subset S$ where $|S_g| \geq 1$. We restrict consideration to graphs in which goal nodes are reachable from the initial node.

We define a path Λ within G as a sequence of nodes $\Lambda = s_0 s_1 \cdots s_n$, where $s_k \in S$ and $(s_k, s_{k+1}) \in E$ for all k. We define the final node s_n to be one of the goal nodes S_g while others are not. We denote Λ_k as the prefix of Λ up to the k-th node in the path (i.e., $\Lambda_k = s_0 s_1 \cdots s_k$). Given an edge (s, s'), we define *edge travel time* $C_{ss'}(t)$, or simply



(A) Initial node s_0 and goal node s_1 with time-dependent edge time function C



(B) Edge time of $C_{s_0 s_1}(t)$

FIGURE 3.5: Two-node graph example with time-dependent edge times. The edge time $C_{s_0 s_1}$ is 5.1 seconds for the first 3.5 seconds and then reduces to 1.2 seconds. The self-transition edge time $C_{s_0 s_0}$ is 1 second for all departure time.

edge time, as the time it takes to traverse from node s to s', having departed node s at time t. Without loss of generality, path traversal begins no earlier than $t = 0$, and edge time $C_{ss'}(t)$ is ∞ for all $t \leq 0$ (i.e., we consider it to be impossible transition).

Normally the TDSP algorithm is used to solve traffic problems, where the time-dependent directed graph G is already well defined. For unstructured environment like the ocean, the graph needs to be manually generated by discretising the workspace. Discretization can be done either with grid-based sampling or random-based sampling (PRM).

3.3.2 Arrival and travel time

The term *arrival time* [27] refers to the time when the agent will arrive at its destination after departing from its initial position at some time. This is different to *travel time*, which is the time taken to arrive at the destination. Consider the following example.

Example 3.1. *Consider an example time-dependent directed graph as shown in Fig. 3.5. If the vehicle departed from s_0 to s_1 at 1 sec, then the arrival time is 6.1 sec while the travel time is 5.1 sec. If the vehicle had departed at 3.6 sec, then the arrival time is 4.8 sec while the travel time is 1.2 sec.*

We define $a_{s,s'}(t)$ as the arrival time from traversing s to s' after departing at time t and $a_\Lambda(t)$ as the arrival time of traversing path Λ at time t. Since the input and output of the function are both time, we can recursively define the arrival time for a sequence of edges. Consider a vehicle path $\Lambda = s_0 s_1 \cdots s_n$. Assuming we do not allow waiting at any nodes, the arrival time at any point in the path Λ_k can be recursively written as follows [27]:

$$\begin{aligned}
a_{\Lambda_1}(t) =& a_{s_0,s_1}(t) \\
a_{\Lambda_2}(t) =& a_{s_1,s_2}(a_{s_0,s_1}(t)) = a_{s_1,s_2}(a_{\Lambda_1}(t)) \\
a_{\Lambda_3}(t) =& a_{s_2,s_3}(a_{s_1,s_2}(a_{s_0,s_1}(t))) = a_{s_2,s_3}(a_{\Lambda_2}(t)) \\
&\vdots \\
a_{\Lambda_k}(t) =& a_{s_k-1,s_k}(a_{\Lambda_{k-1}}(t)) \\
&\vdots \\
a_\Lambda(t) =& a_{s_{n-1},s_n}(a_{\Lambda_{n-1}}(t)).
\end{aligned} \tag{3.5}$$

Similarly, *travel time* $T_\Lambda(t)$ is defined as the time it takes to complete the path Λ, having departed from the initial node s_0 at time t. Formally [27],

$$T_\Lambda(t) = a_\Lambda(t) - t. \tag{3.6}$$

Both arrival and travel times depend on the sequence of edge times, each of which depends on the individual arrival time at each edge.

A common objective of TDSP is to solve for minimal arrival time in a traffic environment, since arriving at a destination earlier is more important than shorting the travel time. However for robotic applications, minimising travel time is valued higher, as we wish to complete a mission as quickly as possible.

3.3.3 FIFO and non-FIFO properties

First-in-first-out (FIFO) is a behaviour associated with a time-dependent graph if it satisfies the following property [27, 73]:

$$t + C_{ss'}(t) \leq t' + C_{ss'}(t'), \tag{3.7}$$

for any edge $(s, s') \in E$ and $t \leq t'$. Intuitively, the arrival time $a_\Lambda(t)$ is non-decreasing with respect to departure time t. Under the FIFO condition, optimal solutions exhibit the following properties [27]:

1. waiting at any node is not beneficial at any time
2. optimal paths are acyclic (i.e., they do not revisit nodes)
3. any subset of the optimal path is also a shortest path.

Conversely, a graph with non-FIFO property means that the arrival time may not be non-decreasing with respect to departure time. For example, the time-dependent graph shown in Fig. 3.5 clearly does not exhibit FIFO behaviour, since departing later resulted in faster arrival time.

3.4 Piecewise-constant function

One method of solving TDSP is by using value iteration that converges an optimal policy. This requires a good understanding of the time-varying edge cost in our graph. However solving for continuous time has shown to be difficult [28, 29]. Furthermore, real-ocean forecast data provides discrete time information. As such, to solve the value iteration for time varying costs, we use piecewise-constant function to closely approximate for continuous time. In this section, we present travel and edge time functions in the form of *piecewise-constant functions* (PF) [4, 118]. We illustrate the form and the relevant operations required to understand the PF-based value iteration where the solution is given as a policy rather than a path.

3.4.1 Definition of piecewise-constant function (PF)

A piecewise-constant function $f : \mathbb{R} \to \mathbb{R}$ is defined as a sequence of constant v_k^f and its corresponding time interval subdomain $p_{k-1}^f \geq t \geq p_k^f$ where $p_k^f \in P^f$ for $k \in \mathbb{N}$. The subdomains are indexed backwards (i.e., $p_{k+1}^f < p_k^f,\ \forall k \in \mathbb{N}$) so that each subdomain can be defined by their lower bounds $t \geq p_k^f$. To account for the end of the forecast window, the function assumes time-invariant at the end constant value v_f^1 at the end of the forecast window. We formally write the piecewise-constant function as follows:

$$f(t) = \begin{cases} v_1^f, & \text{if } t > p_1^f \\ \vdots & \\ v_k^f, & \text{if } p_{k-1}^f \geq t > p_k^f \\ \vdots & \\ v_n^f, & \text{if } p_{n-1}^f \geq t > p_n^f \\ \infty, & \text{else} \end{cases} \equiv \begin{cases} v_1^f, & \text{if } t > p_1^f \\ \vdots & \\ v_k^f, & \text{else if } t > p_k^f \\ \vdots & \\ v_n^f, & \text{else if } t > p_n^f \\ \infty, & \text{else} \end{cases}, \tag{3.8}$$

where ∞ implies cost of impossible transition.

Example 3.2. *A piecewise-constant function of the edge cost demonstrated in Example 3.1 can be written as follows:*

$$f(t) = \begin{cases} 1.2, & \textit{if } t > 3 \\ 5.1, & \textit{else if } t > 0 \cdot \\ \infty, & \textit{else} \end{cases} \tag{3.9}$$

3.4.2 Piecewise-constant function syntax

Consider a vehicle path $\Lambda = s_0 s_1 s_2$ with no waiting allowed at any nodes and departing from s_0 at time t. Suppose we define each time segment of Λ as a PF $C_{s_0 s_1}$ and $C_{s_1 s_2}$. Then by (3.5), $a_{\Lambda_1}(t) = t + C_{s_0 s_1}(t)$ and $a_{\Lambda_2}(t) = t + C_{s_1 s_2}(t + C_{s_0 s_1}(t))$. To evaluate this arrival time, we need to define the operands for PF. Operations such as addition $+$, scalar multiplication $\cdot$, conditioning $\ominus$, merging $\oplus$ and shifting over PF are defined in [4] as follows:

- Addition: The addition between two PFs. Commutative.

$$f(t) + g(t) = \begin{cases} \vdots \\ v_k^f, & \text{else if } t > p \\ \vdots \\ \infty, & \text{else} \end{cases} \qquad \text{where } p \in (P^f \cup P^g) \tag{3.10}$$

- Multiplication: The multiplication between a PF and a constant. Commutative.

$$c \cdot f(t) = \begin{cases} c \cdot v_1^f, & \text{if } t > p_1^f \\ \vdots \\ c \cdot v_k^n, & \text{else if } t > p_n^f \\ \infty, & \text{else} \end{cases} \tag{3.11}$$

- Conditioning: Redefining the lowest bound of the subdomain. Non-commutative.

$$f(t) \ominus p = \begin{cases} f(t), & \text{if } t > p \\ \infty, & \text{else} \end{cases} \tag{3.12}$$

- Merging: Merges two PFs. Non-commutative.

$$f(t) \oplus g(t) = \begin{cases} f(t), & \text{if } t > p_n^f \\ g(t), & \text{else} \end{cases} \tag{3.13}$$

- Shifting: Shifts the subdomain.

$$f(t+p) = \begin{cases} v_1^f, & \text{if } t > p_1^f - p \\ \vdots \\ v_k^n, & \text{else if } t > p_n^f - p \\ \infty, & \text{else} \end{cases} \tag{3.14}$$

In addition, we define an operation called *recursion* where a PF is shifted by another. Suppose we have two PFs $f(t)$ and $g(t)$. A recursive PF $f(t + g(t))$ is defined by a combination of conditioning, merging and shifting operators such that

$$f(t + g(t)) = (f(t + p_1^g) \ominus p_1^g) \oplus \cdots \oplus (f(t + p_n^g) \ominus p_n^g). \tag{3.15}$$

3.5 Problem statement

This thesis addresses the problem of autonomous motion planning of underwater gliders in an energy-optimal or time-optimal manner. There are three challenges that makes this difficult to achieve:

1. Section 3.1 showed that the underwater gliders are modelled as a 12-dimensional complex dynamic system with an unorthodox method of forward propulsion (due to lacking an active propulsion hardware). Planning an optimal path with such complex model in high dimensions is known to be hard [84].

2. Time or energy optimal navigation through flow field, such as the oceanic current that the underwater glider travels through, is a well-known open problem known as the Zermelo's problem [19]. As the glider velocity is inherently slow, its motion is heavily influenced by the flow field and must ether exploit or intelligently navigate the surrounding flow fields to reach its goal in an energy or time-optimal manner.

3. Real-world oceanic current changes with respect to time (i.e., time-dependent flow field). Finding a time-optimal path though such a flow fields turns the problem into a TDSP problem, which is also difficult to solve [28, 29].

To address the first two challenges; we consider the following energy-optimal planning problem with the glider operating in time-invariants flow fields, which we later address in Chapter 4:

Problem 1 (Energy-optimal path planning for underwater glider in time-invariant flow field)**.** *Given an initial state* $\mathbf{x}(0) = \mathbf{x}_{init}$ *and a set of goal states* $\mathbf{X}_g$*, find an energy-optimal*

finite sequence of control vectors $\mathbf{U}^*$ *in time-invariant flow field, such that*

$$\begin{aligned} \mathbf{U}^* = & \underset{\mathbf{U}=\{\mathbf{u}(0),\cdots\}}{\arg\min} \ \text{cost}(\mathbf{x}(0), \mathbf{U}) \\ & s.t.\ \mathbf{x}(last) \in \mathbf{X}_g. \end{aligned} \tag{3.16}$$

To address the third challenge, we first approach the problem as a graph-based problem. We first consider two sub-problems of TDSP; one to solve for the optimal start time that minimises the total travel time, and other to solve for the minimum travel time given a set departure time t_0. Both problems are addressed in Ch. 5:

Problem 2 (Minimum travel time problem)**.** *Given a directed graph* G *with time-dependent edge time function* C*, find an optimal path* Λ^* *and initial departure time* t_0^* *that minimises the travel time* T*, such that*

$$(\Lambda^*, t_0^*) = \underset{\Lambda,t}{\arg\min}\, a_\Lambda(t) - t. \tag{3.17}$$

Problem 3 (Minimum travel time problem given initial departure time)**.** *Given a starting time* t_0 *and a directed graph* G *with time-dependent edge time function* C*, find an optimal path* Λ^* *that minimises the travel time* T*, such that*

$$\Lambda^* = \underset{\Lambda}{\arg\min}\, a_\Lambda(t_0) - t_0. \tag{3.18}$$

We extend our approach to the third challenge in Ch. 6, by incorporating the vehicle dynamics as we did with Problem 1. We consider a similar problem to our graph-based approach, except now we want to synthesis a continuous path that minimises the time to travel from initial to goal position. Let the continuous path be σ, a_σ denote the arrival time and $T_\sigma = a_\sigma(t) - t$ denote the travel time. Then the two sub-problems of TDSP can be reformulated as follows:

Problem 4 (Minimising travel time for continuous path)**.** *Given vehicle dynamics as described in Section 3.1, time-dependent flow field* $\mathbf{v}_c$*, starting state* $\mathbf{x}_{init}$*, and a goal*

region $\mathbf{X}_g$*, find the continuous path* σ^* *and starting time* t_0^* *that minimises the travel time:*

$$\begin{aligned} &(\sigma^*, t_0^*) = \arg\min_{\sigma,t} T_\sigma(t) \\ &s.t.\ \mathbf{x}(t_0^*) = \mathbf{x}_{init} \ \textit{and}\ \mathbf{x}(a_{\sigma^*}(t_0^*)) \in \mathbf{X}_g. \end{aligned} \tag{3.19}$$

Problem 5 (Minimising travel time for continuous path for given start time)**.** *Given vehicle dynamics as described in Section 3.1, time-dependent flow field* $\mathbf{v}_c$*, starting state* $\mathbf{x}_{init}$*, starting time* t_0 *and a goal region* $\mathbf{X}_g$*, find the continuous path* σ^* *that minimises the travel time:*

$$\begin{aligned} &\sigma^* = \arg\min_{\sigma} T_\sigma(t_0) \\ &s.t.\ \mathbf{x}(t_0) = \mathbf{x}_{init} \ \textit{and}\ \mathbf{x}(a_{\sigma^*}(t_0)) \in \mathbf{X}_g. \end{aligned} \tag{3.20}$$

Chapter 4

Kinodynamic planning in 3D static flow fields

In this chapter, the problem of navigating through a static oceanic flow field with underwater glider dynamics, as defined in Chapter 3, is considered. This problem addresses the low-level energy-optimal operations aspect of our underwater glider problem. The difficulties arise from both the flow field navigation and the complex dynamic model of the glider. The limitations of current methods only address one of these problems, while oversimplifying the other. Section 4.1 addresses these limitations by introducing *trim-based control method*; reducing a high-dimensional glider dynamic model to a more parsimonious 6-dimensional kinematic model. The control method achieves this reduction by using *trim state*: the state of dynamic equilibrium in which the glider continues indefinitely in the absence of active controls. Section 4.2 implements the trim-based control method onto a state-of-the-art sampling-based planning algorithm, FMT*, to develop an asymptotically energy-optimal motion planning algorithm for flow field navigation. Section 4.3 demonstrates that the trim-based implementation preserves the theoretical properties and guarantees. Section 4.4 provides some empirical examples derived from a conventional flow field model.

4.1 Trim-based control for underwater glider

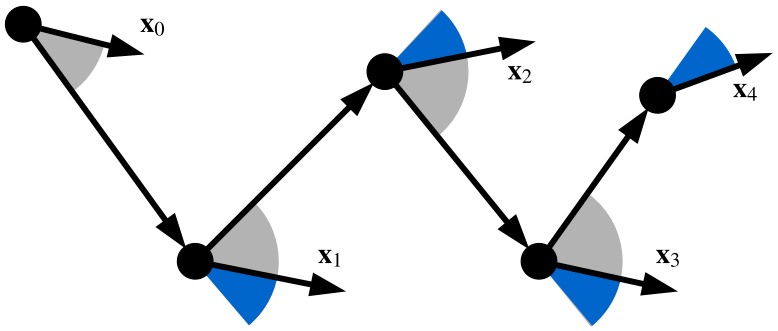
The complete dynamic model of an underwater glider is complex as its velocity is a non-linear function of γ and m_b, as discussed in Section 3.1. The glider's reliance on buoyancy for velocity control means the glider must dive up and down to operate, referred to as *saw-tooth motion*. As mentioned in Chapter 2, the general approach in modelling underwater gliders for motion planning is to approximate it as a 2D kinodynamic model in depth-average ocean currents. The non-linear velocity function is replaced with a constant glider maximum forward velocity, thus removing the sawtooth motion. Such an approach is inapplicable for real work application. The model does not consider the change in flow fields with depths that would influence the real-world sawtooth motion operation. Planning in 2D also neglects seabed obstacles from uneven bathymetry that may require a different sawtooth profile to avoid. This section introduces the concept of *trim state* and *trim-based control*. The glider dynamics is formulated with the trim state so that it can be expressed in kinodynamic model while retaining complexity from the dynamic model.

4.1.1 Trim-based model

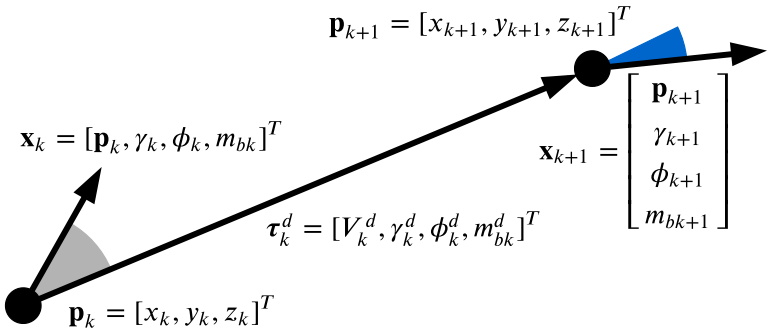
A *trim state* is a state of dynamic equilibrium in which a vehicle will continue indefinitely in the absence of disturbances or variations to control inputs [21]. A *trim-based model* uses the trim state for motion planning. This model allows for efficient underwater glider manoeuvre because the only significant proportion of energy expenditure comes from moving its internal mass and changing its net buoyancy to change the trim state. Suppose there is some sequence of state vectors $\mathbf{X} = \{\mathbf{x}_0, \mathbf{x}_1, \cdots, \mathbf{x}_k, \cdots\}$ and corresponding position vectors $\mathbf{P} = \{\mathbf{p}_0, \mathbf{p}_1, \cdots, \mathbf{p}_k, \cdots\}$ as shown in Fig. 4.1a. The k-th trim state $\boldsymbol{\tau}_k^d$ is denoted between two position vectors $\mathbf{p}_k$ and $\mathbf{p}_{k+1}$. For an underwater glider, $\boldsymbol{\tau}_k$ is defined as

$$\boldsymbol{\tau}_k = \begin{bmatrix} V_{Gk} & \gamma_k & \phi_k & m_{bk} \end{bmatrix}^T, \tag{4.1}$$

where the nominal velocity V_{Gk} is a function of control angles (γ_k, δ_k) and the ballast state m_{bk} as defined in (3.1) and (3.2). The trim-based model represents the glider state by only considering the states used to derive the trim state, reducing the 12-dimensional



(A) Sequence of state vectors and trim states



(B) Transition from state $\mathbf{x}_k$ to $\mathbf{x}_{k+1}$. A desired trim condition $\boldsymbol{\tau}_k^d$ is found between $\mathbf{p}_k$ and $\mathbf{p}_{k+1}$

FIGURE 4.1: Kinematic model for 6D glider. Gray shading represents initial control from initial glider state to trim state, and the blue represents the control from trim state to final glider state.

dynamic model defined in (3.3) to a kinematic model with 6-variable states. The reduced glider kinematic state $\mathbf{x}_k$ at timestep k is redefined as

$$\mathbf{x}_k = \begin{bmatrix} \mathbf{p}_k & \gamma_k & \phi_k & m_{bk} \end{bmatrix}^T. \tag{4.2}$$

The trim-based model represents the glider control inputs $\mathbf{u}(t)$ as *control instances*, consisting of control angles and the target depth. A sequence of control instances forms a trim-based control policy $\boldsymbol{\pi}_{\boldsymbol{\tau}}$. The k-th control instance in $\boldsymbol{\pi}_{\boldsymbol{\tau}}$ is defined as

$$\boldsymbol{\pi}_{\boldsymbol{\tau}}(k) = [\gamma_k, \phi_k, z_k]^T, \tag{4.3}$$

where z_k is the k-th target depth. The ballast tank state m_{bk} depends on whether the glider is moving up or down, as discussed in Section 3.1. For simplicity, the ballast tank is

assumed either empty or full at any time (i.e., $m_b \in [0, m_{b\max}]$). The difference between k-th and its previous depth state determines the k-th ballast control, such that

$$m_{bk} = \begin{cases} 0, & \text{if } z_k < z_{k-1} \\ m_{b\max}, & \text{otherwise.} \end{cases} \tag{4.4}$$

Given a trim-based control policy where the sequence of trim states represent the gliders manoeuvres, we approximate the dynamical path planning problem defined in Problem 1 and redefine it as follows:

Problem 6 (trim-based path planning)**.** *Given initial and goal states* $\mathbf{x}_0$ *and* $\mathbf{x}_f$*, find an energy-optimal control policy* $\boldsymbol{\pi}^*_{\boldsymbol{\tau}}$ *of the form in (4.3).*

With the trim-based model, changes to control instance are made in two cases:

1. From the initial kinematic state $\mathbf{x}_k$ to the trim state $\boldsymbol{\tau}^d_k$.
2. From the trim state to the next state $\mathbf{x}_{k+1}$.

The state transition between a kinematic state and a trim state is assumed to be instantaneous because the state transition time is negligible compared to the gliding time in a trim state. No variations to control inputs are made during the trim state, such that

$$\mathbf{u}(t) \approx \emptyset, \forall t \in \{t \in (t_k, t_{k+1}) \mid \mathbf{x}(t) \in (\mathbf{x}_k, \mathbf{x}_{k+1})\}, \tag{4.5}$$

where times at states $\mathbf{x}_k$ and $\mathbf{x}_{k+1}$ are denoted as t_k and t_{k+1}, respectively.

4.1.2 Computing a trim state

Given a glider control instance $[\gamma, \delta, z]$ and the oceanic flow fields $\mathbf{V}_c(\mathbf{p}(t)) = [u_c, v_c, w_c]^T$, the infinitesimal linear kinematic model of an underwater glider is expressed as:

$$\mathbf{p}(t + \Delta t) = \mathbf{p}(t) + \left(\begin{bmatrix} V_G(\gamma, m_b) \cos\gamma \cos\phi \\ V_G(\gamma, m_b) \cos\gamma \sin\phi \\ V_G(\gamma, m_b) \sin\gamma \end{bmatrix} + \begin{bmatrix} u_c \\ v_c \\ w_c \end{bmatrix} \right) \Delta t, \tag{4.6}$$

where the ballast state m_b is attained from the expected change in glider depth z, as defined in (4.4). Formally, we find a trim state $\boldsymbol{\tau}_k$ connecting glider positions $\mathbf{p}_k$ and $\mathbf{p}_{k+1}$, such that there exists time $t_{k+1} > t_k$ and a trim state $\boldsymbol{\tau}_k$ satisfying (4.6) for all $t \in [t_k, t_{k+1}]$ where $\mathbf{p}(t_{k+1}) = \mathbf{p}_{k+1}$ and $\mathbf{p}(t_k) = \mathbf{p}_k$.

Considering the non-linear form of the nominal glider velocity function defined in (3.1) and the complexity of oceanic flow field $\mathbf{V}_c$, solving for trim state $\boldsymbol{\tau}_k$ analytically from (4.6) is difficult. Instead, a numerical optimisation method called the *shooting method* is used to evaluate for the required trim state. The forward integrated path output of (4.6) is enumerated against a *lookup table* and derives the trim state that reaches $\mathbf{p}_{k+1}$ from $\mathbf{p}_k$. i-th instance in the lookup table is of the form $[\gamma_i, \delta_i, m_b i]$. From the dynamic equality in Appendix A.1, the set of glide angles is sampled from the following interval:

$$\left[\tan^{-1} \left(\frac{-2K_{D_0}}{K_{L_0} \mp \sqrt{K_{L_0}^2 + K_{D_0} K_L^2 / K_D}} \right), \pm 60^\circ \right]. \tag{4.7}$$

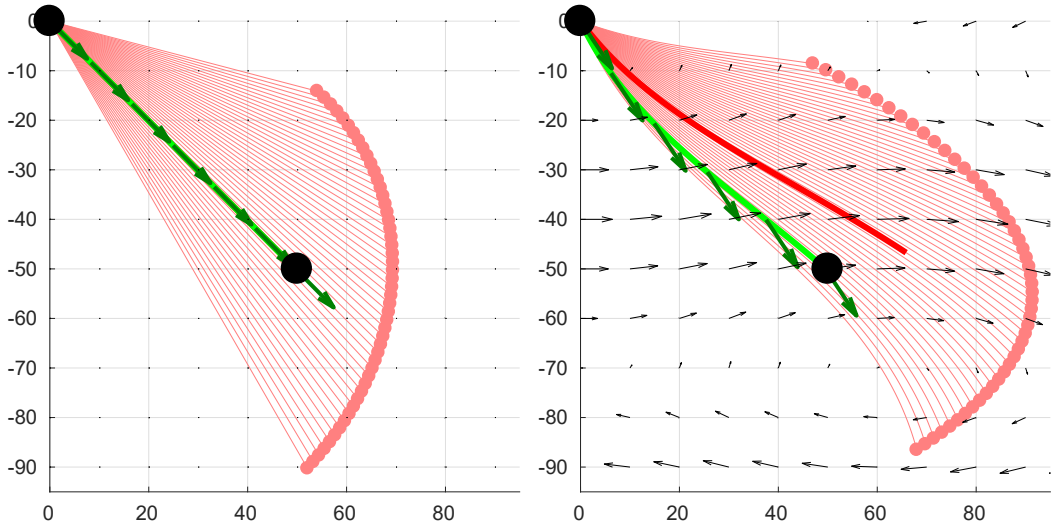
The maximum glide angle is heuristically chosen to account for safe glider operations. With the glider parameters used in Appendix A.1, the bounds on glide angles are $\Gamma_d = [-60^\circ, -3.74^\circ] \cap [3.74^\circ, 60^\circ]$.

Figure 4.2 demonstrates the shooting method case with and without flow in the xz-plane, where the glider is to traverse from $[0, 0]$ to $[50, -50]$. This example shows how the glider's velocity varies with the glide angle and illustrates reachability with respect to currents. With zero current, as shown in Fig. 4.2a, the path is aligned with the glide angle γ. Including ocean currents, as shown in Fig. 4.2b, the reachability differs significantly. The bold red line is the glide angle γ chosen from the case without currents.

4.1.3 Trim-based model energy cost

Based on the trim-based model, the *trim-based energy consumption model* is evaluated. The total glider energy expenditure E_{total} using this model is defined as follows:

$$\mathrm{E}_{total} = \mathrm{E}_{\Delta B} + \mathrm{E}_H + \mathrm{E}_{\Delta r_p} + \mathrm{E}_{\Delta \phi}, \tag{4.8}$$



(A) Path with no current reaches target in 51 s

(B) Path with current reaches target in 61 s.

FIGURE 4.2: Finding trim conditions from $[0, 0, 0]^T$ to $[50, 0, -50]^T$ (bold black circles) without and with ocean currents. The paths from the lookup table are drawn for 80 seconds (pale red lines). The satisfying path (pale green line) and the corresponding trim condition (dark arrows) are shown. The trim condition found in (a) is realised in (b) (dark red line) and the distance error is $22.54m$.

where for some constants η_p, η_m, η_δ, and L_H:

- $\mathrm{E}_{\Delta B} = -\eta_p g z \Delta m_b$ is the energy required to overcome the underwater pressure at depth z to empty the ballast tank.
- $\mathrm{E}_{\Delta \mathbf{r}_p} = \eta_m \cdot \bar{m} \cdot \Delta \mathbf{r}_p$ is the energy required to shift the moving mass to a new position $\mathbf{r}_p$ to control the glide angle γ (A.2)
- $\mathrm{E}_{\Delta\phi} = \eta_\phi \cdot \Delta\phi$ is the energy required to change the heading angle. For long edge transition, we can assume the cost is to move the rudder [119].
- $\mathrm{E}_H = L_H \cdot \Delta t$ is the hotel cost. That is, the energy drain from keeping the systems running during mission.

In contrast, most existing work do not address the energy cost in such detail and instead simplifies the energy consumption. The underwater glider energy cost to transition from states $\mathbf{x}_k$ and $\mathbf{x}_{k+1}$ is denoted as $cost(\mathbf{x}_k, \mathbf{x}_{k+1})$. Assuming the time spent in transition is negligible compared to other energy consumption terms, the glider energy cost is spent in parts across the following process:

1. Transition from state $\mathbf{x}_k$ to trim state $\boldsymbol{\tau}_k^d$. Incurs $\mathrm{E}_{\Delta \mathbf{r}_p}$ and $\mathrm{E}_{\Delta\phi}$. Can also incur $\mathrm{E}_{\Delta B}$ for the appropriate ballast change defined in (4.4).
2. During corresponding desired trim state $\boldsymbol{\tau}_k^d$. Incurs E_H.
3. Transition from trim state $\boldsymbol{\tau}_k^d$ to state $\mathbf{x}_{k+1}$. Incurs $\mathrm{E}_{\Delta \mathbf{r}_p}$ and $\mathrm{E}_{\Delta\phi}$.

4.2 Trim-based FMT*

Trim-based FMT* is an extension of the original FMT* that uses the trim-based model for path planning. The main difference is in the implementation of the neighbourhood search. Unlike the original FMT* that uses either the connection radius or the k-nearest neighbour method to find the node's neighbourhood [96], trim-based FMT* uses the following definition:

Definition 4.1 (Trim-based connection)**.** Consider a path planning problem in a flow field $\mathbf{V}_c(\mathbf{p}(t))$, and a set of glider state samples $\mathbf{X}_{smpl}$. For trim-based FMT*, state node $\mathbf{x} \in \mathbf{X}_{smpl}$ optimally connects to all other non-closed state nodes allowed by the flow field $\mathbf{V}_c(\mathbf{p}(t))$, and the trim dynamics in Section 4.1.

The motivation for this modification is to find all possible connections that are reachable with the glider trim-based model. For this reason, radius connection and k-nearest neighbour connection were dropped as they by intent limit the number of states considered for connection. Trim-based connections can potentially be applied to other motion planning algorithms, such as PRM*, to formulate its trim-based variant. As discussed in Section 4.1.2, there are no known analytical solutions to the optimal connection problem. Thus, the shooting method is used to find an approximate connection. To reduce the computation required to connect a node to all other nodes, the definition of a trim-based connection is exploited by having each *Unvisited* node attempt connections to all *Open* nodes and vice versa.

The sampling method is also modified. Instead of randomly sampling the states across the dimensions, the first N_p samples are chosen over the position space (x, y, z), and then uniformly sample N_a angle states (γ, ϕ) for each position state. From the assumption

regarding the ballast tank (i.e., $m_b \in \{0, m_{b\,\max}\}$), the overall number of samples is $N_s = N_p \cdot N_a \cdot 2$. As the resulting sampling distribution differs between position space, angular space, and ballast state, this method of sampling falls under the category of non-uniform sampling [96] for analysis purposes.

4.3 Analysis

This section presents the properties of the trim-based FMT* framework by comparing it against the original FMT*. Theoretical properties such as asymptotic optimality, convergence rate and computational complexity are shown. From Def. 4.1, consider the following:

Remark 4.2 (Connection radius for trim-based FMT*)**.** From Def. 4.1, the connection radius r_n for trim-based FMT* is large enough to contain every state node $\mathbf{x} \in \mathbf{X}_{smpl}$.

Remark 4.3 (K-nearest neighbourhood for trim-based FMT*)**.** By Def. 4.1, the connection method is equivalent to the k-nearest neighbour (KNN)-based connection where the worst-case number of neighbours is the total number of samples (i.e., $k_n \leq n$).

4.3.1 Asymptotic optimality

The main difference between the original and trim-based FMT* is the use of trim-based connections Def. 4.1 to find the nearest neighbours. We prove the asymptotic optimality (AO) of trim-based FMT* by showing that the algorithm retains properties equivalent to the original FMT* algorithm.

Remark 4.4 (The connection radius for asymptotic optimality)**.** Given a cost function that obeys the triangle inequality, the connection radius r_n that ensures the asymptotic optimality of the original FMT* algorithm is shown in [96] to be

$$r_n = \mathcal{H} \cdot \left(\frac{\log(n)}{n} \right)^{\frac{1}{d}}, \tag{4.9}$$

where $\mathcal{H} > 2\,(1+\eta)\,\left(\frac{1}{d}\right)^{\frac{1}{d}} \left(\frac{\mu\left(\chi_{free}\right)}{\zeta_d} \right)^{\frac{1}{d}}$, $\eta > 0$ and $\mathcal{H} > 0$.

Given such a radius, the expected number of connections in an obstacle-free environment is less than or equal to:

$$(n/\mu\left(\chi_{free}\right))\,\zeta_d r_n^d = 2^d(1+\eta)^d(1/d)\log(n) \tag{4.10}$$

Lemma 4.5 (Triangle inequality for trim-based energy model cost function)**.** *Assuming a negligible hotel load E_H compared to other sources depicted in (4.8), the triangular inequality holds true for (4.8), such that*

$$cost(a,c) < cost(a,b) + cost(b,c), \tag{4.11}$$

for all glider states a, b and c .

Proof. In a real-world application, the negligible hotel load assumption generally holds, as the actuation of controls (moving centre of mass, activating ballast pump) are significantly more costly than keeping the onboard computer running. If the hotel load were negligible compared to other components, then the cost would depend on the number of state transitions made in its sequence, which would always satisfy the triangle inequality. □

To get a better intuition on Lemma 4.5, consider an example glider sequence with no flow field shown in Fig. 4.3. Figure A.1 shows the nominal glider velocity V_G monotonically increasing with respect to its glide angle γ. While the Euclidean distance of glider sequence $\{a, c\}$ is shorter than $\{a, b, c\}$, it travels with a lower γ value, and thus slower. For this example, the slower velocity outweighs the shorter euclidean distance and thus the glider sequence $\{a, c\}$ takes longer to arrives at c than sequence $\{a, b, c\}$. Recall from Section 4.1.3 that the hotel load E_H is directly proportional to time and the other control costs are bounded by the glider dynamics. If the hotel load was non-negligible, then its clear that, $cost(a, c) > cost(a, b) + cost(b, c)$ for long edge transition, which contradicts the triangle inequality.

Lemma 4.6 (Size of tuning parameter η for trim-based FMT*)**.** *From the connection raidus defined in Remark, 4.4, the tuning parameter η in $\mathcal{H}$ must be large enough for the radius r_n to satisfy the criterion for asymptotic optimality.*

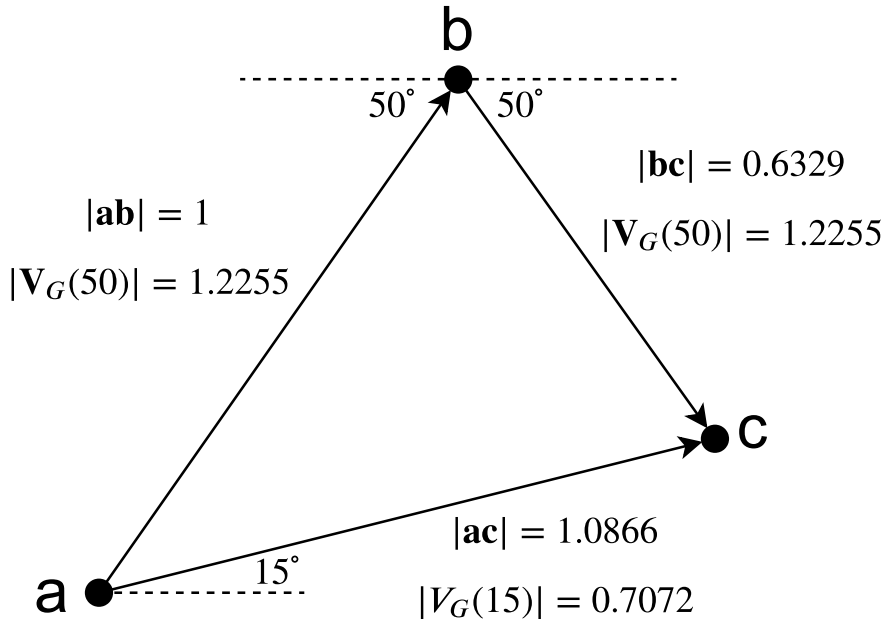


FIGURE 4.3: Counter-example of triangle inequality for hotel cost dominant cost function. We assume that all other cost components are negligible.

Proof. From Remark 4.4, the tuning parameter η can be any value greater than zero. In order to make r_n a large number as defined in Remark 4.2, parameter $\mathcal{H}$ must be a large number; this is ensured if η is a sufficiently large number. □

Theorem 4.7 (Asymptotic optimality of trim-based FMT*)**.** *As the number of samples $n \to \infty$, the solution for trim-based FMT* approaches the optimal solution in expectation, such that $\lim_{n\to\infty} \mathbb{P}(c_n > (1+\epsilon)c^*) = 0$, where c_n is the arc length with n and c^* is the optimal arc length.*

Proof. By Remark 4.4, the cost function must obey the triangle inequality to ensure AO. By Lemma 4.5, the glider energy cost function obeys the triangle inequality, assuming the hotel load is negligible compared to other expenditures. By Lemma 4.6, the tuning parameter η must be a large enough number to ensure AO. Because there exists no upper bound for the tuning parameter, η can be any number required by the configuration space to achieve AO. □

4.3.2 Convergence rate

Similar to the proof for asymptotic optimality, the proof for the convergence rate of the original FMT* is used to derive the convergence rate for trim-based FMT*

Remark 4.8 (Convergence rate of FMT*)**.** Given a d-dimensional configuration space, and a connection radius defined in Remark 4.4, the convergence rate for the original FMT* for all $\epsilon > 0$ is shown in [96] to obey

$$\mathbb{P}(c_n > (1+\epsilon)c^*) \in \begin{cases} \mathcal{O}\left((\log n)^{-\frac{1}{d}}\, n^{\frac{1}{d}\left(1-(1+\eta)^d\right)+\rho}\right), & \text{if } \eta \leq \frac{2}{\left(2^d-1\right)^{1/d}} - 1 \\ \mathcal{O}\left(n^{-\frac{1}{d}\left(\frac{1+\eta}{2+\theta}\right)^d}\right), & \text{otherwise} \end{cases}. \tag{4.12}$$

Theorem 4.9 (Convergence rate of trim-based FMT*)**.** *The convergence rate for trim-based FMT* is*

$$\mathbb{P}(c_n > (1+\epsilon)c^*) \in \mathcal{O}\left(n^{-\frac{1}{d}\left(\frac{1+\eta}{2+\theta}\right)^d}\right). \tag{4.13}$$

Proof. By Lemma 4.6, the tuning parameter η is set large enough to ensure asymptotic optimality and cover all samples. In expectation, there exist a *eta* value large enough to sure both ensure asymptotic optimality and $\eta > \frac{2}{\left(2^d-1\right)^{1/d}} - 1$. □

4.3.3 Computational complexity

The computational complexity of the trim-based FMT* is shown to be $\mathcal{O}\left(n^2\right)$. The proof follows from that for the original FMT*, starting by comparing edge cost computations for PRM* and FMT*.

Remark 4.10. As demonstrated in [96]; given the same sample size $|\mathbf{X}_{smpl}|$, the computational complexity for FMT* is less than or equal to the computational complexity for PRM*.

Theorem 4.11 (Computational complexity of trim-based FMT*)**.** *Given trim-based connections between nodes for very large r_n that connects to all other samples in $\mathbf{X}_{smpl}$, the expected computational complexity of trim-based FMT* is $\mathcal{O}(n^2)$.*

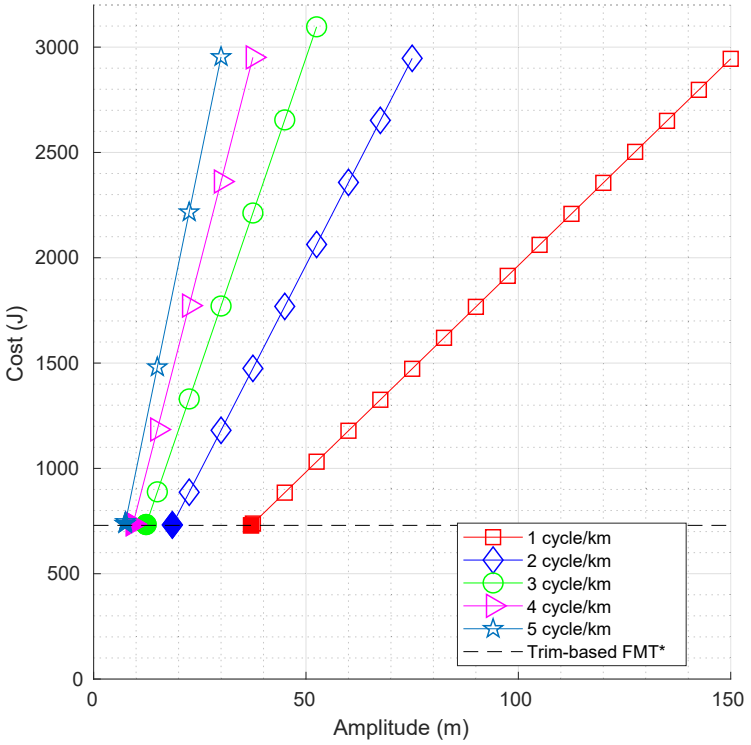
Proof. By Lemma 4.6, a sample is expected to be connected to every other sample in $\mathcal{X}_{free}$. Since the cost function in trim-based FMT* is directional, the number of edge cost computations in expectation is $\mathcal{O}(n^2)$. □

4.3.4 Comparison with standard sawtooth profiles

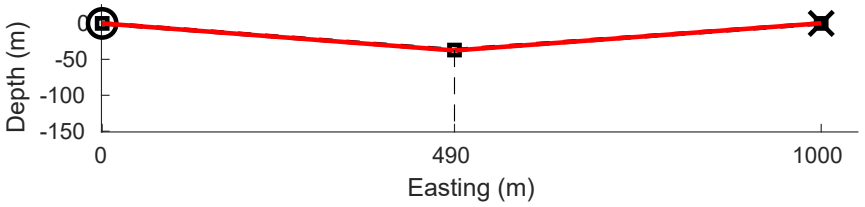
We assume that the control system of an autonomous underwater glider is a finite state machine that follows a pre-programmed mission plan. The plan can only be adjusted while the glider is on the surface. While operating, the standard mission plan causes the glider to follow a "sawtooth" profile that oscillates it between descending and ascending at the same fixed glide angle. The glider descends until it reaches a programmed maximum depth and then inverts its net buoyancy to ascend until it reaches a programmed minimum depth, at which point it inverts its net buoyancy and begins to descend again. This pattern continues for a programmed number of cycles, after which the glider ascends to the surface to update its estimated position, report measurements and optionally, receive an updated mission plan.

Standard sawtooth profiles are considered an energy-efficient method to operate an underwater glider [18]. In this section, we demonstrate that the FMT* solution, which generates an irregular depth profile, produces a solution that is at least as efficient. For convenience, we restrict the trim-based FMT* to planning over 4D space $\mathbf{x} = [x, z, \gamma, m_b]^T$ to demonstrate its effect on the depth profile.

We consider the glider travelling from inital kinematic state $\mathbf{x}_{init} = [0, 0, -0.2, 2]^T$ to goal $\mathbf{x}_{goal} = [1000, 0, -0.2, 2]^T$. To ensure the path is straight, we assume there are no obstacles or ocean currents. We compare the trim-based FMT* method against various combinations of sawtooth frequencies $f = [1, 2, 3, 4, 5]$ cycles/km and consider amplitudes at 7.5 m intervals between z_{min} and 150 m, where $z_{min} = [37.064, 18.532, 12.355, 9.266, 7.413]$ is the shallowest depth the glider can ascend to during its sawtooth pattern, given the dynamic constraints. The algorithm samples the nodes as a uniform grid to ensure the comparison remains fair. The uniform grid covers most states used by the sawtooth path. In this case, the depth is sampled every 7.413 m, and the x-axis is sampled every 10 m.



(A) Cost comparison between paths generated by the proposed framework, which outperforms standard sawtooth paths with from 1 to 5 cycles per kilometre. The minimum cost yielded by the proposed framework is $729.5J$ (dashed black line). The solid markers are the bounds imposed by the glider dynamics (i.e., limits on glide angle).



(B) The optimal path from the proposed method considered in case (a). The depth profile is similar to the 1 cycle/km sawtooth case (red square markers), but its depth profile is asymmetrical.

FIGURE 4.4: Path costs using the proposed framework and fixed sawtooth profiles

Figure 4.4a compares the energy expenditure between the sawtooth cases and the trim-based FMT* path. The sawtooth cost is linear in both its frequency and depth. This relation is due to the dominance of the cost of buoyancy inversion. The cost of buoyancy inversion is only incurred when expelling water from the ballast tank (i.e., when transitioning from diving to rising), and it is proportional to the depth at which it occurs. Surfacing from deeper depths, therefore, increases cost, as does increase the frequency at

which buoyancy is inverted. As expected, the sawtooth with a frequency of 1 cycles/km and the depth of $z_{min} = 37.064$ was the cheapest, costing only 729.5998 J. The trim-based FMT* path was slightly more efficient, costing 729.5184 J.

The path generated by the trim-based FMT* shown in Fig. 4.4b is similar to the one cycle sawtooth case with the shallowest depth at $z_{min} = 37.065$. The difference is that the depth profile is asymmetric, as a result of minimising the changes in glide angles. The resulting asymmetrical sawtooth path shows that sawtooth heuristics may not always be energy-optimal, and the trim-based FMT* will produce no worse solution than the sawtooth. The disparity between sawtooth and trim-based FMT* becomes more apparent with the addition of obstacles and currents, which will be discussed later.

4.4 Examples

In this section, glider paths generated by trim-based FMT* are demonstrated in various environments. The resulting depth profiles differ from the standard sawtooth path, and results in some are counter-intuitive trajectories. This section features how the framework finds an energy-optimal path through various flow fields and obstacles.

The underwater glider model was described in Section 3.1 with its parameter values listed in Appendix A.1. Using the trim-based FMT* presented in Section 4.2, an energy-optimal path that minimises the energy cost described in Section 4.1.3 is found. The framework is implemented using MATLAB. Unless otherwise stated, the simulation is planned in a 6-Dimensional state space, as described in Section 4.1. The kinematic states are in SI units with angles in radians.

All simulations impose additional constraints on the glide and heading angle. When transitioning from glider kinematic state to trim state and vice versa, the maximum change to the glide angle is $\pm 30°$, and the maximum change to the heading angle is $\pm 45°$. This is to ensure that planned paths are smooth.

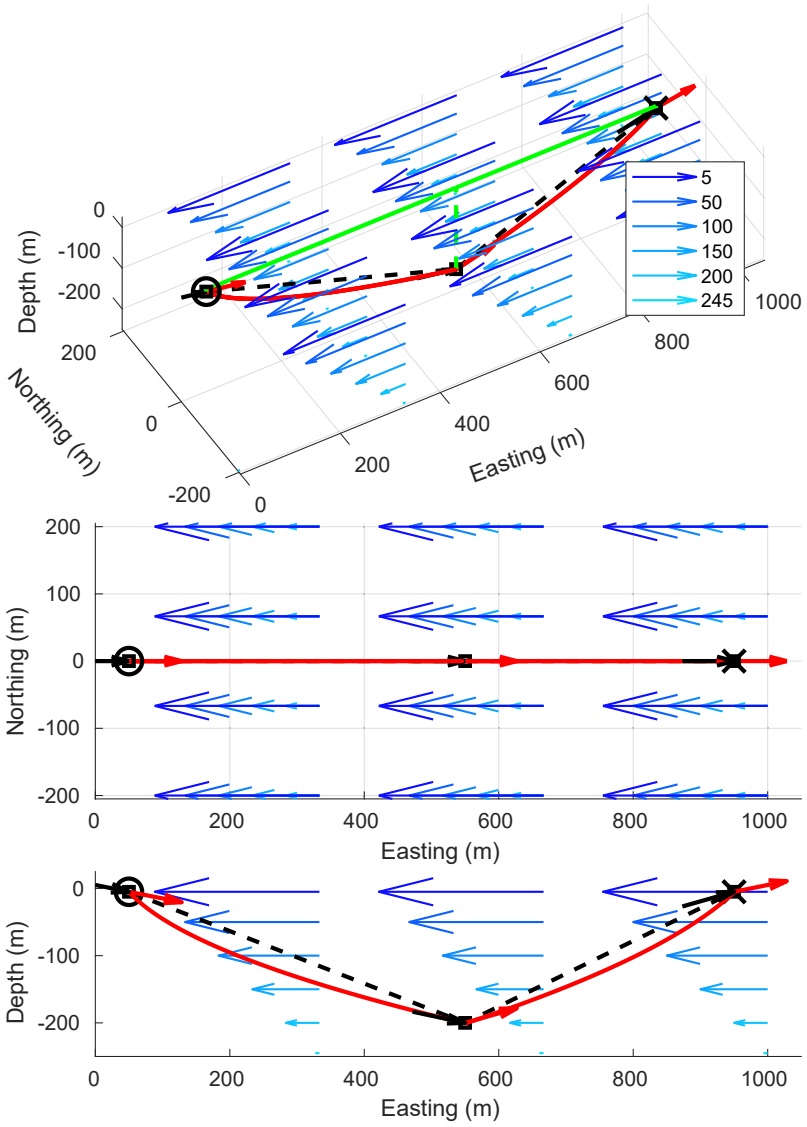
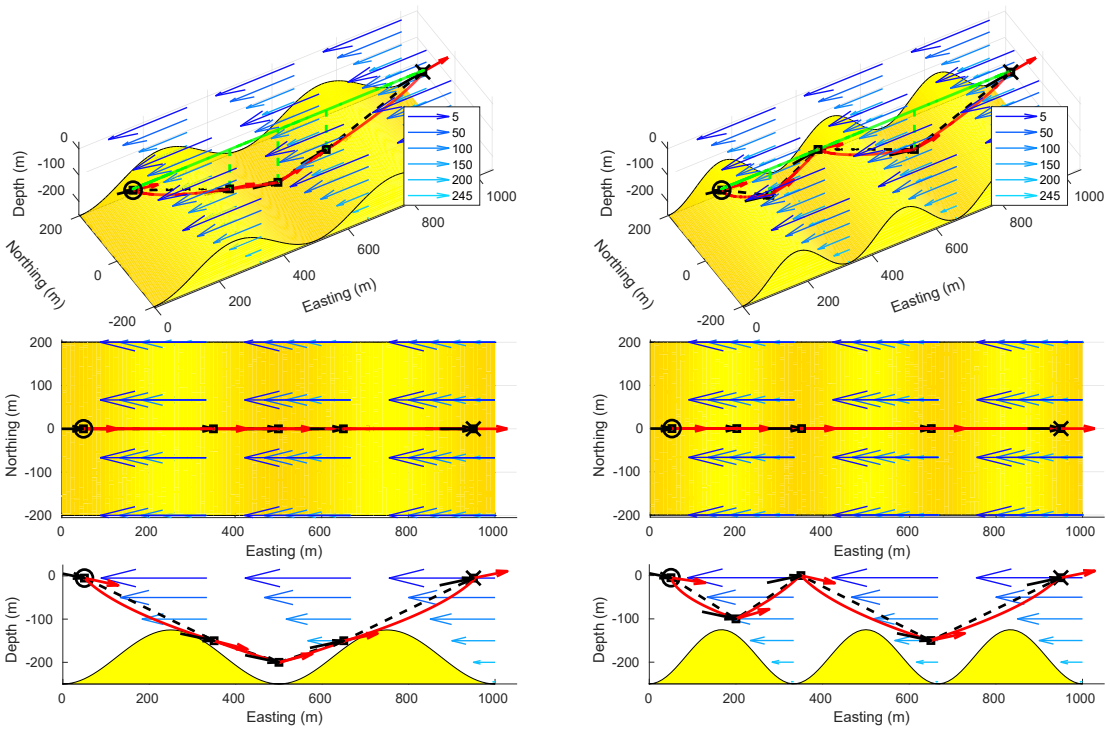


FIGURE 4.5: Optimal paths from $(50, 0, 5)$ to $(950, 0, 5)$ using the proposed method in opposing currents that diminish linearly with depth. The current-influenced trajectory and the edge connections are shown in red and dashed black lines, respectively, and the path projection to the surface is shown as a green line. The trim and glider state orientation at each node are shown as red and black and red arrows, respectively. The path cost is 3925.38J with 54434 sample nodes.

4.4.1 Opposing current with no obstacles

In this example, a glider travelling against an opposing current that decays linearly with depth is considered, shown in Fig. 4.5. The resulting path is displayed in 3D, top-down and side views. The glider starts from kinematic states $\mathbf{x}_{init} = [50, 0, 5, -0.2, 0, 2]^T$ to $\mathbf{x}_{goal} = [950, 0, 5, 0.2, 0, 0]^T$.



(A) A single cycle consists of three control variations. The path cost is 3925.4771 J with 38450 sample nodes.

(B) Two cycles consist of three control variations. The path cost is 4908.7182 J with 38450 sample nodes.

FIGURE 4.6: Optimal paths from $(50, 0, 5)$ to $(950, 0, 5)$ using trim-based method. The opposing current diminishes linearly with depth and the seafloor consists of (a) two and (b) three dune obstacles. The current-influenced trajectory and the edge connections are shown as red and dashed black lines, respectively, and the path projection to the surface is shown as a green line. The trim and glider state orientation at each node are shown as red and black arrows, respectively.

Similar to the path shown in Fig. 4.4b, the edge connections reveal an irregular sawtooth with only one surfacing to minimise the number of buoyancy inversions and glide angle changes incurring a cost. This result supports the previous claim that standard sawtooth profiles may not yield the optimal solution. The glider path also shows it dives deeper to avoid the strong opposing surface current.

Despite the sawtooth-like edge connection, the actual glider path (shown in red) follows a parabolic profile. As demonstrated in Fig. 4.2, the straight edge connection represents valid trim connection, while the actual trajectory as a result from the trim state and depth-varying flow field can be curved. In contrast, multiple nodes were needed to generate a similar path in [31].

4.4.2 Opposing current with sand dunes

In this example, the same opposing current as Section 4.4.1 is considered, but now with a sand dunes on the ocean floor acting as obstacles. Its purpose is to demonstrate the effect of obstructions on the glider's path. The dunes are represented as a cosine function with an amplitude of 125 m, considering various frequencies.

A case with two dunes is shown in Fig. 4.6a. Similar to the no-obstacle case in Fig. 4.5, the glider path features only one ballast change. However, each diving and surfacing action requires an additional trim state change for finer control to navigate past the obstacle. The glider path is slightly more expensive than the case with no obstacles, due to the extra state transitions.

A case with three dunes is shown in Fig. 4.6b. As the obstacle now obstructs the path featured in Fig. 4.5, the algorithm forces the glider to ascend twice to avoid it. The glider makes a small, inexpensive sawtooth cycle to relocate to a more favourable position, then it makes a deeper sawtooth cycle, covering the remaining distance. This solution agrees with the previous claim that the trim-based FMT* will produce no worse solution than the sawtooth heuristics. Despite ascending twice, the penalty is not severe compared to the case without an obstacle (4908.7182 J vs. 3925.3791 J), since the cost of buoyancy inversion scales only linearly with depth.

4.4.3 Irrotational flows with islands

This example shows a more horizontally oriented glider path in Fig. 4.7 with 3D, top-down and side views. Two irrotational vortices define the oceanic flow field; one outward-flowing (source), and one inward-flowing (sink). An elliptical island obstructs the centre of the planning space, splittings the stream flowing from the source to the sink. A restricted zone is set up at the eye of each vortex that is equivalent to obstacles to avoid the case where the glider gets stuck in strong eddy current. The glider moves from an initial kinematic state $\mathbf{x}_{init} = [50, -100, 5, -0.2, 0, 2]^T$, to a goal $\mathbf{x}_{goal} = [950, 100, 5, 0.2, 0, 0]^T$.

The straight edge passing through the restricted zone demonstrates the advantage of constructing edges by optimisation (shooting method) instead of straight lines. By using the

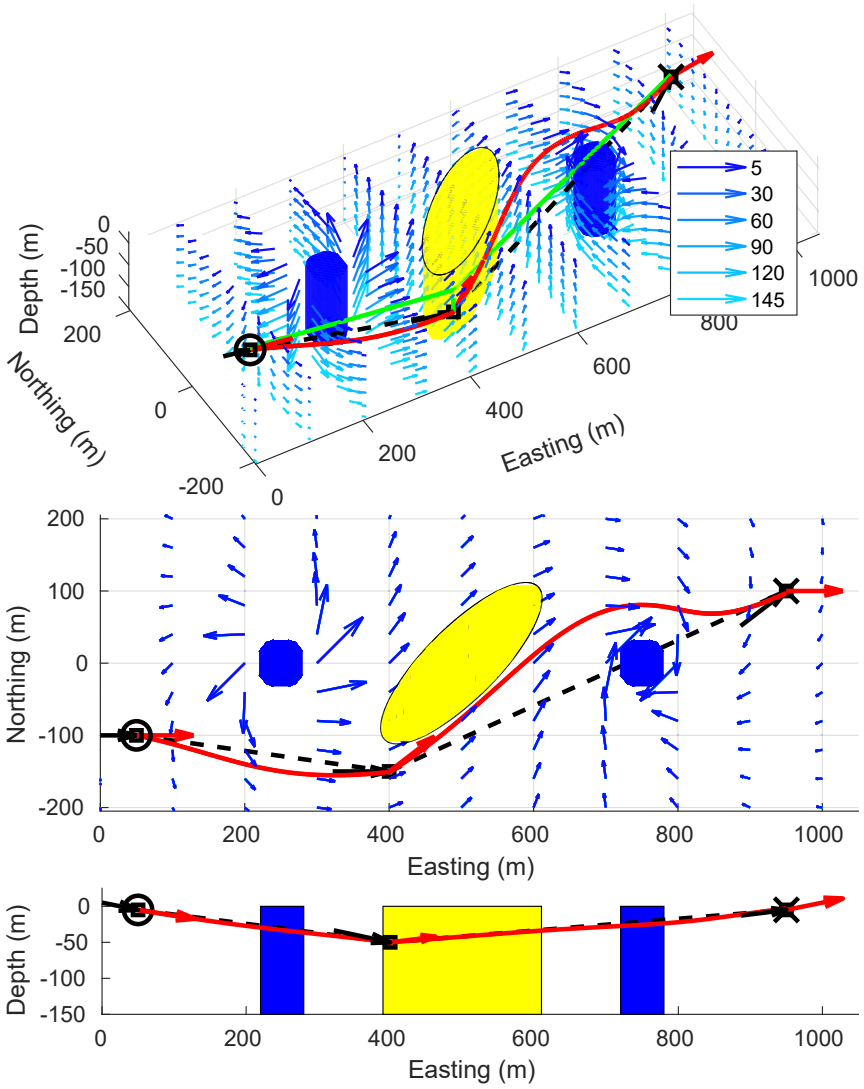
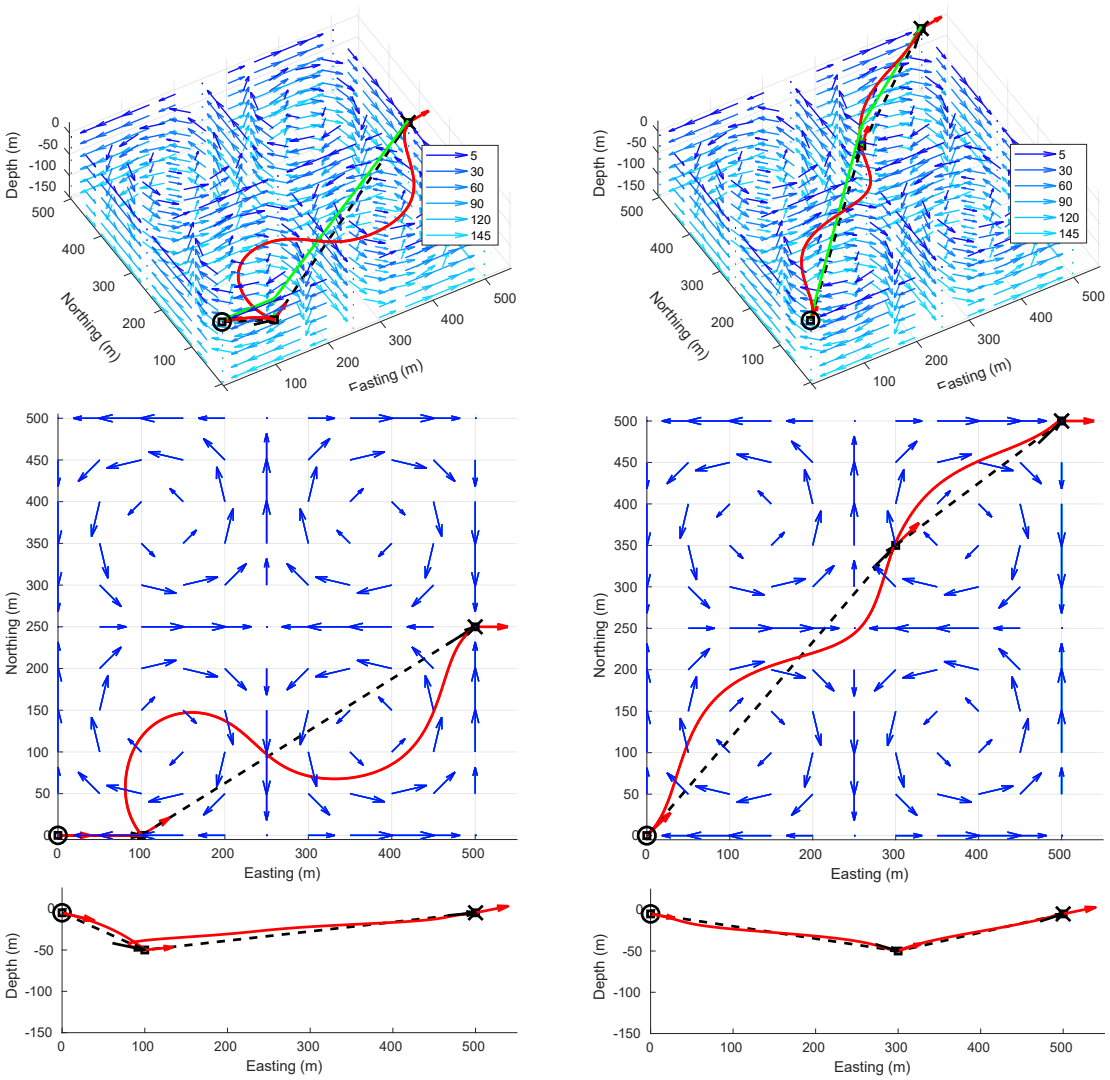


FIGURE 4.7: Optimal paths from $(50, -100, 5)$ to $(950, 100, 50)$ with two irrotational vortices. Three island obstacles are shown as yellow and blue volumes. The current-influenced trajectory and the edge connections are shown as red and dashed black lines, respectively, and the path projection to the surface is shown as a green line. The trim condition and state orientation at each node are shown as red and black arrows, respectively. The path cost is 1010.41 J with 33794 sample nodes.

oceanic flow field, the glider can navigate around the obstacle with only a single node, as opposed to multiple if being navigated with straight-edge connections.

The glider takes a single, shallow dive and surfaces to minimise the cost of buoyancy inversion. The planned path differentiates itself from a single-period sawtooth by resurfacing where the glider needs to change its bearing. The adjustment allows the glider to redirect itself into a more favourable flow field.



(A) Path from $(0, 0, 0)$ to $(500, 250, 0)$ with a cost of 1006.2012 J

(B) Path from $(0, 0, 0)$ to $(500, 500, 0)$ with a cost of 1016.1843 J

FIGURE 4.8: Optimal paths in a quad vortex featured in [25] with 23234 sample nodes. The ocean currents do not change with depth. The current-influenced trajectory and the edge connections are shown as red and dashed black lines, respectively, and the path projection on the surface is shown as a green line. The trim and glider state orientation at each node is shown as red and black arrows, respectively.

4.4.4 Quad vortices

This example considers a quad vortex case based on the Taylor-Green gyre model [63]. Previous literature [25] have used this flow field model for a planning case for a surface vehicle (e.g., boat). We extend this work by adding depth to the environment and planning using trim dynamics. Similar to the irrotational flow case, the flow across the XY-plane

is defined by the Taylor-Green equation and is uniform across depth. The horizontal path projection from the trim-based FMT* path is compared to the path in [25]. Two cases are presented, both starting from an initial kinematic state $\mathbf{x}_{init} = [0, 0, 5, -0.2, 0, 2]^T$. Figure 4.8a shows the glider moving to $\mathbf{x}_{goal} = [500, 250, 5, 0.2, 0, 0]^T$, and Fig. 4.8b shows the glider moving to $\mathbf{x}_{goal} = [500, 500, 5, 0.2, 0, 0]^T$.

In both quad vortex cases, the glider's path is defined with only a single change in trim state between initial and goal points, reducing the state transition expenditure. Similar to the island case, the edge connections in the XY-plane represent a more realistic curved path that moves smoothly with the ocean current (shown in red) and bears similarities to the time-optimal paths presented in [25]. Although negligible, the glider energy cost from Equ. 4.8 does depend on time, and the algorithm attempts to minimise it as well.

Figure. 4.8a shows a more direct path taken to reach the goal compared to the path in [25]. The glider moves slightly east, then rides the current the rest of the way to the goal point. More specifically, the trim-based path positions the glider at a favourable location that allows the glider to ride the ocean current to the goal with minimal heading changes. The existence of a solution with an edge connection that spans the majority of the glider path supports the use of trim-based connections, where the radius connection and the k-nearest connection would have only considered short edge connections.

The trim-based FMT* path also differs with the path presented in our previous paper [31]. The use of shooting method optimisation to make the connection makes the trim-based connection implementation more suited to handle exotic conditions than a straight line edge connection.

The depth profile in both Fig. 4.8a and Fig. 4.8b shows the glider taking a single dive and surfacing to minimise the buoyancy inversion cost. Similar to the no-current case and the island case, the resulting profile is an irregular sawtooth, timing the buoyancy change with the heading change to minimise overall node connections.

4.5 Summary

In this chapter, the problem of path planning in the presence of flow fields for underwater gliders with complex dynamics is addressed. A trim state-based control approach is introduced by reformulating the complex dynamical model into a kinematic model with trim states. Based on the reduced control space, a state-of-the-art sampling-based planning algorithm, FMT*, is implemented while preserving its important properties. The proposed method provides a partial solution to intelligent operation through flow field. Furthermore, it contributes to the standing problem of underwater glider operates that considers both flow fields and glider dynamics. The next chapter investigates methods for optimal operations in a time depedent oceanic flow field environment.

Chapter 5

Planning in non-FIFO time-dependent flow fields

This chapter considers the time-dependent shortest path (TDSP) problem that finds the shortest path in a time-dependent directed graphs. The work lays a foundation for planning with underwater gliders. The TDSP formulation is useful for planning a path across real-world oceanic flow fields in a way that it does not add time as another dimension for planning, thus avoiding the "curse of dimensionality". Previous literature comments on the difficulty in solving the non-FIFO TDSP problem in continuous-time [28, 29]. Furthermore, the continuous oceanic environment does not implicitly define a time-dependent directed graph, and thus the graph needs to be constructed algorithmically. Section 5.1 addresses the continuous-time non-FIFO problem by formulating the TDSP problem with piecewise-constant function (PF) [4, 118]. Section 5.2 presents an optimal PF-based policy planning for non-FIFO TDSP problems. Section 5.3 presents proof that the solution converges and can be solved in polynomial time, a first for this problem variation. Section 5.4 address the construction of a time-dependent directed graph in a flow field environment. Section 5.5 provides empirical examples from several discrete cases.

5.1 TDSP with Piecewise-constant functions

This section introduces a new PF-based TDSP framework to approximate the continuous-time problem in discrete-time. The edge cost time $C_{ss'}(t)$, and by extension the path travel time $T_\Lambda(t)$, are represented using piecewise-constant functions (PF) as defined in Section 3.8 and shown in Example 3.2. Intuitively for glider operations, we expect an optimal path might include a cyclic part that "waits" for better flow field conditions. This property violates one of the FIFO conditions, so the TDSP framework needs to be able to solve for non-FIFO cases.

Given a time-dependent directed graph G, the output of the PF-based TDSP problem is given in time-dependent *travel policy* $\pi_s(t)$ for each states s. The policy at s dictates which neighbouring state $s' \in S_s$ it should transition to at time t. The travel policies are represented as a PF defined in Section 3.4.1 with slight change in notation. Formally,

$$\pi_s(t) = \begin{cases} s'^1 \in S_s & \text{if } t > p_s^1 \\ s'^2 \in S_s & \text{else if } t > p_s^2 \\ \vdots & \vdots \\ s'^{n_{\pi_s}} \in S_s & \text{else if } t > 0 \\ \emptyset & \text{else} \end{cases}, \tag{5.1}$$

where s'^i is the state visited during the i-th subdomain, and n_{π_s} is the number of policy subdomains.

Evaluating the path travel time from each state s to the goal state $s_g \in S_g$ as a result of its travel policy $\pi_s(t)$ is difficult, as there are many combinations of state transitions after the policy transition across the time-dependent directed graph that can reach the goal. Therefore the path travel time function defined in (3.6) is re-expressed in an iterative form where $T_s^k(t)$ is the travel time for k edge transitions starting at state s. Suppose the transitions are as shown in Fig. 5.1. Then formally,

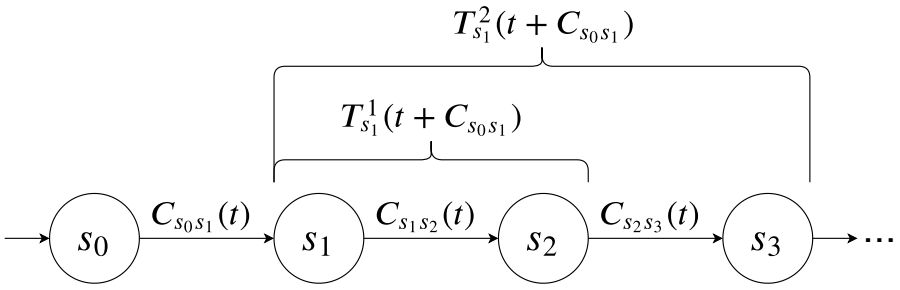


FIGURE 5.1: Example node sequence in a time-dependent directed graph.

$$
\begin{aligned}
T^0_{s_0}(t) =& 0 \\
T^1_{s_0}(t) =& C_{s_0s_1}(t) \\
T^2_{s_0}(t) =& C_{s_0s_1}(t) + C_{s_1s_2}\big(t + C_{s_0s_1}(t)\big) \\
=& C_{s_0s_1}(t) + T^1_{s_1}\big(t + C_{s_0s_1}(t)\big) \\
T^3_{s_0}(t) =& C_{s_0s_1}(t) + C_{s_1s_2}\big(t + C_{s_0s_1}(t)\big) + C_{s_2s_3}\Big(t + C_{s_0s_1}(t) + C_{s_1s_2}\big(t + C_{s_0s_1}(t)\big)\Big) \\
=& C_{s_0s_1}(t) + T^2_{s_1}\big(t + C_{s_0s_1}(t)\big) \\
& \dots
\end{aligned}
\tag{5.2}
$$

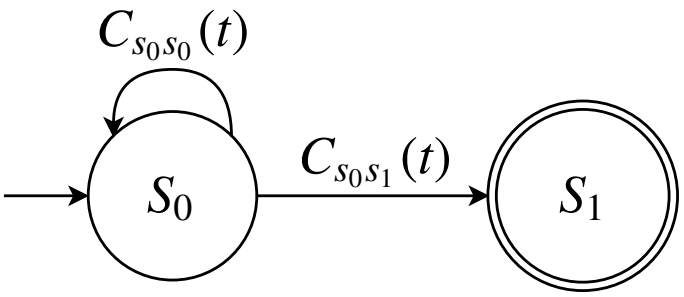
The travel time function can therefore be written recursively as

$$
T^{k+1}_s(t) = C_{ss'}(t) + T^k_{s'}(t + C_{ss'}(t)). \tag{5.3}
$$

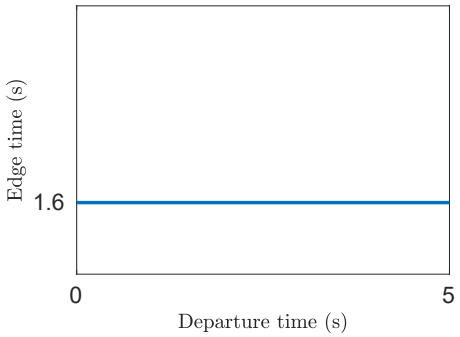
We denote $T^*_s(t)$ as the converged travel time, where $T^{k+1}_s(t) = T^k_s(t)$ for all $t \in \mathbb{R}$ and some finite $k \in \mathbb{Z}$. Given a travel policy π_s as defined in (5.1), the PF-based travel time function is written as

$$
T^{k+1}_s(t) = \begin{cases}
C_{ss'^1}(t) + T^k_{s'^1}(t + C_{ss'^1}(t)) & \text{if } t > p^1_s \\
C_{ss'^2}(t) + T^k_{s'^2}(t + C_{ss'^2}(t)) & \text{else if } t > p^2_s \\
\vdots & \vdots \\
C_{ss'^{n_{\pi_s}}}(t) + T^k_{s'^{n_{\pi_s}}}(t + C_{ss'^{n_{\pi_s}}}(t)) & \text{else if } t > 0
\end{cases}, \tag{5.4}
$$

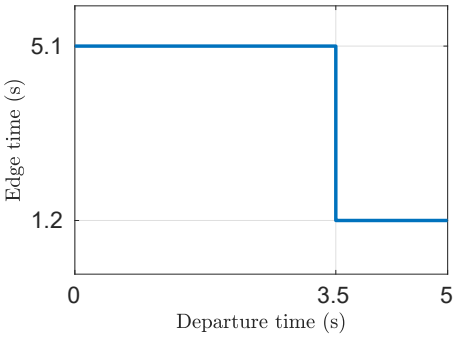
where p^k_s is the beginning of the k-th subdomain in policy π_s. The travel time for the goal states $s \in S_g$ at any iteration k is 0 for all $t \in \mathbb{R}$. For simplicity, the short form omitting



(A) Initial node s_0 and goal node s_1 with time-dependent edge time function C



(B) Edge time $C_{s_0s_0}(t)$



(C) Edge time $C_{s_0s_1}(t)$

FIGURE 5.2: A non-FIFO case two-node graph example with time-dependent edge times. The edge time $C_{s_0s_1}$ is 5.1 s for the first 3.5 s and then reduces to 1.2 s. The self-transition edge time $C_{s_0s_0}$ is 1 s for all departure time. Self-transitioning is different to waiting; the former is defined as an edge transition allowed by a graph, whereas the latter is an imposed hold duration that is not defined in a graph.

the "*else*" case is used, i.e., $t \in (-\infty, 0]$. As a result of the recurrsion and merging PF-syntax introduced in (3.15), the number of subdomain for the travel time will grow. We later shows that this growth is bounded.

Example 5.1 (Calculation example)**.** *We illustrate the calculation of travel time using PF-based travel policy for the example in Fig. 5.2. The edge times in PF form are*

$$C_{s_0s_0} = \begin{cases} 1.6 & \text{if } t > 0 \end{cases} \text{ and } C_{s_0s_1} = \begin{cases} 1.2 & \text{if } t > 3.5 \\ 5.1 & \text{else if } t > 0 \end{cases}. \tag{5.5}$$

Suppose an arbitrary policy was chosen at s_0 that causes self-transition between $0 < t < 3$, and then made to transition to s_1 thereafter. Such policy is denoted in PF as,

$$\pi_{s_0}(t) = \begin{cases} s_1 & \text{if } t > 3 \\ s_0 & \text{else if } t > 0 \end{cases}. \tag{5.6}$$

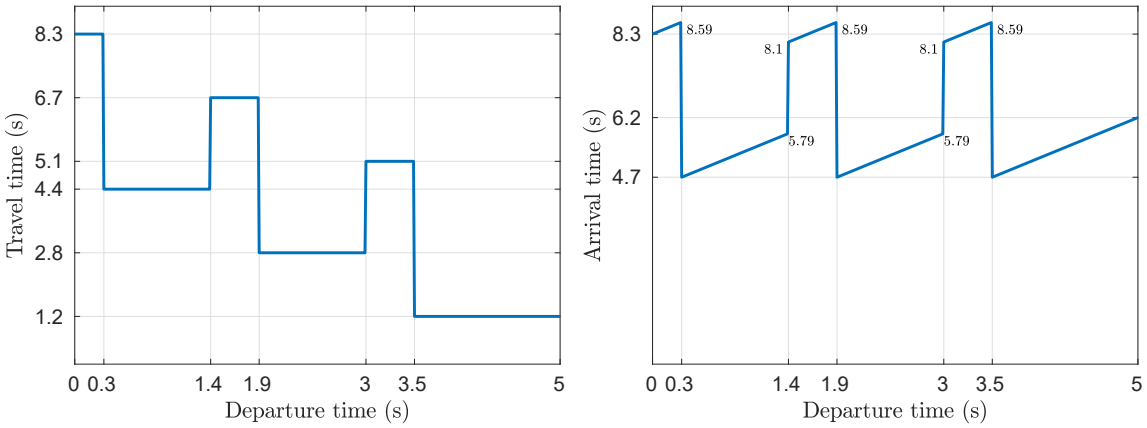
State s_1 is set as the goal state, thus denoting $T^k_{s_1}(t) = 0,\ \forall k$. The travel time at s_0 after $k+1$ edge transitions can then be evaluated with the PF syntax, including the conditioning operation, defined in Section 3.4.2,

$$\begin{aligned} T^{k+1}_{s_0}(t) &= \begin{cases} C_{s_0 s_1}(t) + T^k_{s_1}(t + C_{s_0 s_1}(t)) & \textit{if } t > 3 \\ C_{s_0 s_0}(t) + T^k_{s_0}(t + C_{s_0 s_0}(t)) & \textit{else if } t > 0 \end{cases} \\ &= \begin{cases} \begin{cases} 1.2 & \textit{if } t > 3.5 \\ 5.1 & \textit{else if } t > 0 \end{cases} & \textit{if } t > 3 \\ 1.6 + T^k_{s_0}(t + 1.6) & \textit{else if } t > 0 \end{cases} \\ &= \begin{cases} 1.2 & \textit{if } t > 3.5 \\ 5.1 & \textit{else if } t > 3 \\ 1.6 + T^k_{s_0}(t + 1.6) & \textit{else if } t > 0 \end{cases} \end{aligned} \tag{5.7}$$

Taking $T^0_{s_0}(t) = 0$, the travel time is iteratively calculated until the solution converges. The travelling time for the first 2 iterations, $T^1_{s_0}(t)$ and $T^2_{s_0}(t)$, is evaluated as:

$$T^1_{s_0}(t) = \begin{cases} 1.2 & \textit{if } t > 3.5 \\ 5.1 & \textit{else if } t > 3.0 \\ 1.6 & \textit{else if } t > 0 \end{cases}$$

$$T^2_{s_0}(t) = \begin{cases} 1.2 & & \textit{if } t > 3.5 \\ 5.1 & & \textit{else if } t > 3.0 \\ 1.6 + \begin{cases} 1.2 & \textit{if } t > 1.9 \\ 5.1 & \textit{else if } t > 1.4 \\ 1.6 & \textit{else if } t > -1.6 \end{cases} & \textit{else if } t > 0 \end{cases} = \begin{cases} 1.2 & \textit{if } t > 3.5 \\ 5.1 & \textit{else if } t > 3.0 \\ 2.8 & \textit{else if } t > 1.9 \\ 6.7 & \textit{else if } t > 1.4 \\ 3.2 & \textit{else if } t > 0 \end{cases} \cdot \tag{5.8}$$



(A) Travel time (B) Arrival Time

FIGURE 5.3: Travel and arrival time for travel policy in Example 5.1

For this example, the solution converges after 4 iterations (i.e., $T^4_{s_0} = T^3_{s_0}$). The corresponding travel time subject to the heuristic policy (5.6), is

$$T^*_{s_0}(t) = \begin{cases} 1.2 & \text{if } t > 3.5 \\ 5.1 & \text{else if } t > 3.0 \\ 2.8 & \text{else if } t > 1.9 \\ 6.7 & \text{else if } t > 1.4 \\ 4.4 & \text{else if } t > 0.3 \\ 8.3 & \text{else if } t > 0 \end{cases}. \tag{5.9}$$

Figure 5.3 illustrates both the travel time PF solution and the arrival time. The converged travel time solution can be checked by referring back to the chosen policy in (5.6). If departing from s_0 at $t = 0$, the policy would cause self-transition twice for 3.2 s. As the time at s_0 is now greater than 3 s, the policy would then force a transition to s_1, which from (5.5) would take 5.1 s. Therefore in total, the travel time from s_0 to s_1 would take 8.3 s. Consider that making one more self-transition would have resulted in a faster travel time of 6 s. Therefore the chosen policy for this example is not optimal.

5.2 Optimal travel time policy for non-FIFO TDSP problems

This section presents an optimal policy planning for non-FIFO TDSP problems using the PF-based TDSP framework defined in Section 5.1. Recall in Example 5.1 that a policy with arbitrary selections of states and time subdomains yields suboptimal results. Choosing the right state to transition to at the appropriate time is necessary for an optimal transition to goal state $s_g \in S_g$. Given a directed graph G and time-dependent edge time function $C_{ss'}(t)$, the optimal travel policy π_s^* to reach s_g from state s can be expressed as

$$\pi_s^*(t) = \begin{cases} \vdots & \vdots \\ \arg\min\limits_{s' \in S_s} C_{ss'}(t) + T_{s'}^*(t + C_{ss'}(t)) & \text{else if } t > p_s^{i*} \, . \\ \vdots & \vdots \end{cases} \tag{5.10}$$

In principle, the optimal solution of (5.10) at state s could be computed iteratively, by finding the optimal next state $s' \in S_s$ for each time $t > 0$, thus satisfying the Bellman's principle of optimality. Such an exhaustive approach is impossible as there are infinitely many subdomains to consider. This problem is avoided by finding a finite set of subdomains iteratively. It is proven later that such set exists for an optimal solution.

Let $T_{ss'}^{k+1}$ be an *immediate travel time function* in which the transition from state s to s' occurs over all time t after k edge transitions. Formally,

$$T_{ss'}^{k+1} = \Big\{ C_{ss'}(t) + T_{s'}^k(t + C_{ss'}(t)) \quad \text{if } t > 0 \cdot \tag{5.11}$$

Let P_s^k be the set of subdomains in travel time function T_s^k. Such a set for optimal travel time T_s^{k+1*} is

$$P_s^{k+1*} = \{0\} \cup \bigcup_{s' \in S_s} P_{ss'}^{k+1*}, \tag{5.12}$$

where subdomain set $P_{ss'}^{k+1*}$ is from immediate travel function $T_{ss'}^{k+1*}$ and $P_s^0 = \{0\}$. The optimisation problem in (5.10) is then solved over a finite set of subdomains P_s^{k+1*} where $p_s^{i*} \in P_s^{k+1*}$ in (5.10). The pseudocode is presented in Alg. 1.

Algorithm 1: Solving for optimal policy $\pi_s^{k*}(t)$

Inputs: Directed graph $G = (S, E)$, time-dependent edge time function C and number of edge transitions K

Outputs: Optimal travel policy π^{k*} and travel time T^{k*}

$T_s^0 \leftarrow \begin{cases} 0, & \text{if } t > 0 \end{cases}, \forall s \in S$

for $k \leftarrow 1$ ***to*** K **do**

  forall $s \in S \setminus s_n$ **do**

    $T_{ss'}^{k+1} \leftarrow \begin{cases} \infty & \text{if } t > 0 \end{cases}, \forall s' \in S$ ▷ (5.11)

    $P_s^{k+1*} \leftarrow \{0\}$ ▷ (5.12)

    forall $s' \in S_s$ **do**

      $T_{ss'}^{k+1} \leftarrow \begin{cases} C_{ss'}(t) + T_{s'}^k(t + C_{ss'}(t)) & \text{if } t > 0 \end{cases}$ ▷ (5.11)

      $P_s^{k+1*} \leftarrow P_s^{k+1*} \cup P_{ss'}^{k+1*}$ ▷ (5.12)

    end

    $$T_s^{k+1*} \leftarrow \begin{cases} \vdots & \vdots \\ \min\limits_{s' \in S_s} T_{ss'}^{k+1} & \text{ef } t > p_s^{i*}, \forall p_s^{i*} \in P_s^{k+1*} \\ \vdots & \vdots \end{cases}$$

    $$\pi_s^{k+1*} \leftarrow \begin{cases} \vdots & \vdots \\ \arg\min\limits_{s' \in S_s} T_{ss'}^{k+1} & \text{ef } t > p_s^{i*}, \forall p_s^{i*} \in P_s^{k+1*} \\ \vdots & \vdots \end{cases}$$

  end

  if $P_s^{k+1*} \equiv P_s^{k*}$ *and* $T_s^{k+1*}(t) \equiv T_s^{k*}(t)$, $\forall t \in P_s^{k*}, s \in S$ **then**

    break

  end

end

return $\pi_s^k(t)$ *and* $T_s^*(t)$

Example 5.2 (Optimal policy example)**.** *The optimal travel time with respect to the graph in Fig. 5.2a is*

$$\pi_{s_0}^*(t) = \begin{cases} s_1 & \textit{if } t > 3.5 \\ s_0 & \textit{else if } t > 0.3 \\ s_1 & \textit{else if } t > 0 \end{cases}, \; T_{s_0}^*(t) = \begin{cases} 1.2 & \textit{if } t > 3.5 \\ 2.8 & \textit{else if } t > 1.9 \\ 4.4 & \textit{else if } t > 0.3 \\ 5.1 & \textit{else if } t > 0 \end{cases} \tag{5.13}$$

The solution converged after 5 iterations.

In Example 5.1 we proposed that a policy making three self-transition yields a faster travel time than the arbitrarily chosen policy that only enforced two. However, the optimal policy presented in Example 5.2 provides an interesting non-intuitive solution. It suggests that self-transition is only optimal if the time at state s_0 is between $0.3 < t < 3.5$. For any time earlier, the policy would have the state transition directly to s_1 despite the high edge cost of $C_{s_0 s_1}(t)$ at that time. Indeed, the travel time with three self-transition is 6 sec, while the travel time with no self-transition is 5.1 sec. This solution is due to the unnecessary delay caused by the third self-transition. After the second transition, the edge cost $C_{s_0 s_1}(t)$ would get cheaper only after delaying 0.3sec, and a third transition would waste 1.3sec.

From this optimal policy, we can construct the travel time-optimal sequence of states from s_0 to s_1 for a given departure time. If the departure time was $t = 0$, the optimal state sequence is $s_0 s_1$. For departure time $t = 0.3$, the optimal state sequence is $s_0 s_0 s_0 s_1$. For departure time $t = 1.9$, the optimal state sequence is $s_0 s_0 s_1$.

5.3 Analysis

This section presents proofs shown by one of our co-authors that the solution converges and can be solved in polynomial time. Without loss of generality, we assume $S_s = S$ for all states $s \in S \setminus S_g$ (i.e., all states are immediately reachable from any state). Given (5.11) and (5.12), the set of subdomains can be written as

$$P_s^{k+1} = \{0\} \cup \left\{ p \in \bigcup_{s' \in S_s} C_{ss'} \cup \left(P_{s'}^k - C_{ss'} \right) \mid p \geq 0 \right\}, \tag{5.14}$$

where $A - B = \{a - b \mid \forall a \in A \text{ and } b \in B\}$ for two sets A and B. We slightly abuse notation for $C_{ss'}$ to denote the set of subdomains and constants in the corresponding edge time function. Using a short form $\frac{A}{0} = \{a \in A \mid a \geq 0\}$, we have

$$\begin{aligned} P_s^{k+1} =& \{0\} \cup \bigcup_{s'} C_{ss'} \cup \bigcup_{s'} \frac{P_{s'}^k - C_{ss'}}{0} \\ =& \{0\} \cup \bigcup_{s'} C_{ss'} \cup \bigcup_{s'} \bigcup_{s''} \frac{C_{s's''} - C_{ss'}}{0} \\ & \cup \bigcup_{s'} \bigcup_{s''} \frac{\frac{P_{s'}^{k-1} - C_{s's''}}{0} - C_{ss'}}{0}. \end{aligned} \tag{5.15}$$

Since all subdomain sets for edge time function are non-negative by definition, the last term can be written in a form

$$\frac{\frac{A}{0} - \frac{B}{0}}{0} = \{a - b \mid a \in A, b \in B, a \geq 0, b \geq 0, a - b \geq 0\}. \tag{5.16}$$

Since B (i.e., $C_{ss'}$) is non-negative,

$$\frac{A - B}{0} = \{a - b \mid a \in A, b \in B, a - b \geq 0 \text{ and } b \geq 0\}. \tag{5.17}$$

Intuitively, if $a - b \geq 0$ and $b \geq 0$, then $a \geq 0$. Therefore

$$\frac{\frac{A}{0} - B}{0} \equiv \frac{A - B}{0}, \tag{5.18}$$

if B is a non-negative set. Then the last term in (5.15) becomes

$$\frac{\frac{P^{k-1} - C_{s's''}}{0} - C_{ss'}}{0} = \frac{P_{s'}^{k-1} - C_{ss'} - C_{s's''}}{0}. \tag{5.19}$$

Therefore, the set of subdomains can be recursively written as

$$\begin{aligned} P_s^{k+1} = &\{0\} \cup \bigcup_{s'} C_{ss'} \cup \bigcup_{s'} \bigcup_{s''} \frac{C_{s's''} - C_{ss'}}{0} \\ &\cup \cdots \cup \bigcup_{s'} \cdots \bigcup_{s^k} \frac{C_{s^{k-1}s^k} - \cdots - C_{ss'}}{0}. \end{aligned} \tag{5.20}$$

Lemma 5.1 (Finite edge transitions for convergence given infinite length path)**.** *Given an arbitrary and infinite length path Λ, the set of subdomains converges to a unique and finite set within a finite number of edge transitions.*

Proof. All subdomains in any edge time function are non-negative by definition. Therefore subdomains in (5.20) monotonically decrease as the number of edge transitions increase. Since all subdomains in travel time function are non-negative, there exists a finite maximum number of edge transitions $K_{\max}$ before convergence. □

Lemma 5.2 (Convergence in finite edge transitions)**.** *Given an arbitrary and infinite length path Λ, the worst case number of edge transitions before convergence in subdomain*

set is

$$K_{\max} = \text{ceil}\left(\frac{\max C}{\min C}\right), \tag{5.21}$$

where $C = \bigcup_{s\in S}\bigcup_{s'\in S} C_{ss'} \setminus \{0\}$.

Proof. The worst-case number of edge transition $K_{\max}$ is the last edge transition before the subdomain set in $K_{\max}$-th term becomes empty in (5.20) since subdomains are non-negative. The longest such term is $\max C - \sum_{k=1}^{K_{\max}} \min C$. □

Remark 5.3 (Convergence and cyclic paths)**.** The length of an optimal path that may include cycles is bounded by a finite number of edge transitions found in Lemma 5.2.

Theorem 5.4 (Convergence in optimal algorithm)**.** *The optimal algorithm in Alg. 1 converges in a finite time* $K_{\max}$ *as shown in Lemma 5.2.*

Proof. By Lemma 5.1 and 5.2, there exists a unique and finite set of subdomains for travel time functions. Since the optimisation problem is to find optimal travel policy for each subdomain, the problem is then solved in a finite number of iterations. □

Theorem 5.5 (Time complexity)**.** *The overall time complexity for Alg. 1 is* $\mathcal{O}((|S| \cdot |C_m|)^{k+2})$*, where* $|S|$ *is the number of states in graph* G*,* $|C_m|$ *is the maximum number of subdomains over a set of edge time functions and* k *is the number of edge transitions.*

Proof. By (5.4), the overall complexity is related to the number of subdomains, which is bounded by

$$\begin{aligned} |P_s^{k+1}| &= \left| \{0\} \cup \bigcup_{s'} C_{ss'} \cup \bigcup_{s'} \frac{P_{s'}^k - C_{ss'}}{0} \right| \\ &\leq 1 + (|S| \cdot |C_m|) + (|S| \cdot |C_m|) \cdot |P_{s'}^k| \\ &\leq 1 + 2(|S| \cdot |C_m|) + 2(|S| \cdot |C_m|)^2 \\ &+ \cdots + 2(|S| \cdot |C_m|)^{k+1} \\ &= \mathcal{O}((|S| \cdot |C_m|)^{k+2}), \end{aligned} \tag{5.22}$$

where C_m is the set of subdomains and constants with the maximum cardinality over all edges $e \in E$. In the worst case, we find the optimal policy for each subdomain using

value iteration in Alg. 1. Since the time complexity for solving such value iteration is polynomial in number of states, the overall time complexity for the proposed algorithm is $\mathcal{O}((|S| \cdot |C_m|)^{k+2})$. □

Remark 5.6 (Time-static reduction)**.** From Theorem 5.5, the time complexity for time-static edge functions is reduced to $\mathcal{O}(|S|^2)$ (i.e., $|C_m| = 1$ and $K = 0$) which agrees with the complexity of static shortest path problems.

Remark 5.7 (Time complexity in practice)**.** The time complexity in Theorem 5.5 is based on three worst-case conditions: 1) all states are connected to all the others, 2) the maximum number of iterations depends on the smallest edge subdomain and 3) no overlapping subdomains.

The worst-case conditions in Remark 5.7 occur rarely in practice, particularly Condition 3. When subdomains in a set overlap, they merge and form a much smaller set. Therefore the set does not grow indefinitely until convergence.

By Theorem 5.5, the complexity heavily depends on the number of edge transitions (i.e., iterations). Since the maximum number of edge transitions depends on the subdomains among all edge functions as shown in Lemma 5.2, the running time of the algorithm can be improved significantly by adaptively pruning early rapid changes to increase the denominator (i.e., $\min C$). Furthermore, we can also reduce the size of the edge functions by merging consecutive constants that are similar (i.e., reducing $|C_m|$).

5.4 Optimal Planning Over Time-Dependent Flow

This section proposes a path planning framework for time-dependent flow fields. The section first presents a short argument regarding the time-dependent flow field and the FIFO condition. Then the construction of a time-dependent direct graph in a time-dependent flow field environment is presented.

5.4.1 Time-dependent flow field and FIFO condition

Suppose we have two points in a time-dependent flow field where a single control is found to traverse from one to the other. The path at time t may be different to that at $t' > t$. Since the path for departing later could yield earlier arrival time than departing earlier, time-dependent flow field does not respect FIFO. As the distance between two points approaches zero, the non-FIFO property is weakened since the surrounding flow also approaches time-static constant flow. Therefore, time-dependent flow fields are asymptotically FIFO.

5.4.2 Asymptotically optimal planning for time-dependent flow fields

An asymptotically optimal probabilistic roadmap (PRM*) algorithm is used to build a time-dependent directed graph over the oceanic environment. Details on its construction were discussed in Section 3.2.1. A property of PRM* that the non-FIFO TDSP problem benefits from is that groups of states in PRM* are connected in cycles where the neighbouring regions overlap (e.g., states s_3, s_4, s_5 in Fig. 3.2). Therefore, the PF-based TDSP framework can fully exploit the cyclic edge connections.

The graph expresses the associated edge cost $C_{ss'}(t)$ for each edge $(s, s') \in E$ as a PF with uniform time partitioning. This is approached in a receding time horizon manner. Given the time horizon t_H, partitioning of the edge cost subdomains are taken from the flow field data from the reference time t_0 up to t_H in intervals $t_\Delta = (t_H - t_0)/Q$ where $Q \in \mathbb{Z}$. For $a = \{0, 1, \cdots, Q\}$

$$C_{ss'}(t) = \begin{cases} \vdots & \vdots \\ t^a & \text{else if } t > a \cdot (t_H - t_0)/Q \cdot \\ \vdots & \vdots \end{cases} \tag{5.23}$$

By Lemma 5.1, there exists a finite number of subdomains assuming that the flow forecast is also given in a form of piecewise-constant functions. This is a practically valid assumption as discussed in Section 5.3. The edge cost at each subdomain is evaluated using forward integration with respect to the time-dependent flow fields and the vehicle dynamics. For the sake of demonstration, this chapter considers the Autonomous Surface

Vessel (ASV) model when evaluating the edge cost. The underwater glider solution for the TDSP problem will be presented in the next chapter.

Redefining the vehicle position state, $\mathbf{p} \in \mathbb{R}^2$ denotes the ASV position and $V_c(\mathbf{p}, t) = [u_{\mathbf{p},t}, v_{\mathbf{p},t}]^T$ denote the time-dependent flow field vector. The ASV operates by varying its bearing angle ϕ whilst travelling at constant speed V_{const} relative to $V_c(\mathbf{p}, t)$. The ASV velocity relative to the flow field is denoted as $\mathbf{v}_R(\phi) = [V_{const} \cos\phi, V_{const} \sin\phi]^T$. The ASV dynamic model R operating in $\mathbb{R}^2$ is written as

$$\dot{\mathbf{p}} = \mathbf{v}_R(\phi) + \mathbf{V}_c(\mathbf{p}, t). \tag{5.24}$$

Assuming the flow field vector does not vary during Δt, a discrete time model is

$$\mathbf{p}[j+1] = \mathbf{p}[j] + (\mathbf{v}_R(\theta) + \mathbf{V}_c(\mathbf{p}[j], t)) \cdot \Delta t, \tag{5.25}$$

Given two positions $\mathbf{p}_s$ and $\mathbf{p}_{s'}$, a set of trajectories $\mathbf{P} = \{\mathbf{p}^0, \cdots\}$ are enumerated across a finite set of control samples $\mathbf{U} = \{\phi_0, \cdots\}$ for a predefined time period h. For a given control $\phi_i \in \mathbf{U}$ and departure time t_Δ, the trajectory $\mathbf{p}^i \in \mathbf{P}$ is enumerated by forward integrating from $\mathbf{p}^i[0] = \mathbf{p}_s$ using (5.25) up to the discrete time horizon $H = \text{ceil}(h/\Delta t)$. We then find the trajectory $\mathbf{p}^{i^*} \in \mathbf{P}$ that approaches closest to $\mathbf{p}_{s'}$, such that

$$i^* = \arg\min_i \min_{j \leq H} \|\mathbf{p}_{s'} - \mathbf{p}^i[j]\|. \tag{5.26}$$

5.5 Examples

This section presents two simulated examples where local travel time varies with departure time. The first example is a discrete case where the TDSP framework finds an optimal policy over the discrete time-dependent directed graph and then used to find the corresponding path given the initial departure time. The second example is a flow field scenario where a graph is sampled over a continuous flow field and solve for the path planning problem in an asymptotically optimal manner. The algorithm ran on a standard laptop with Intel i5-6300 2.5GHz CPU and 8GB RAM.

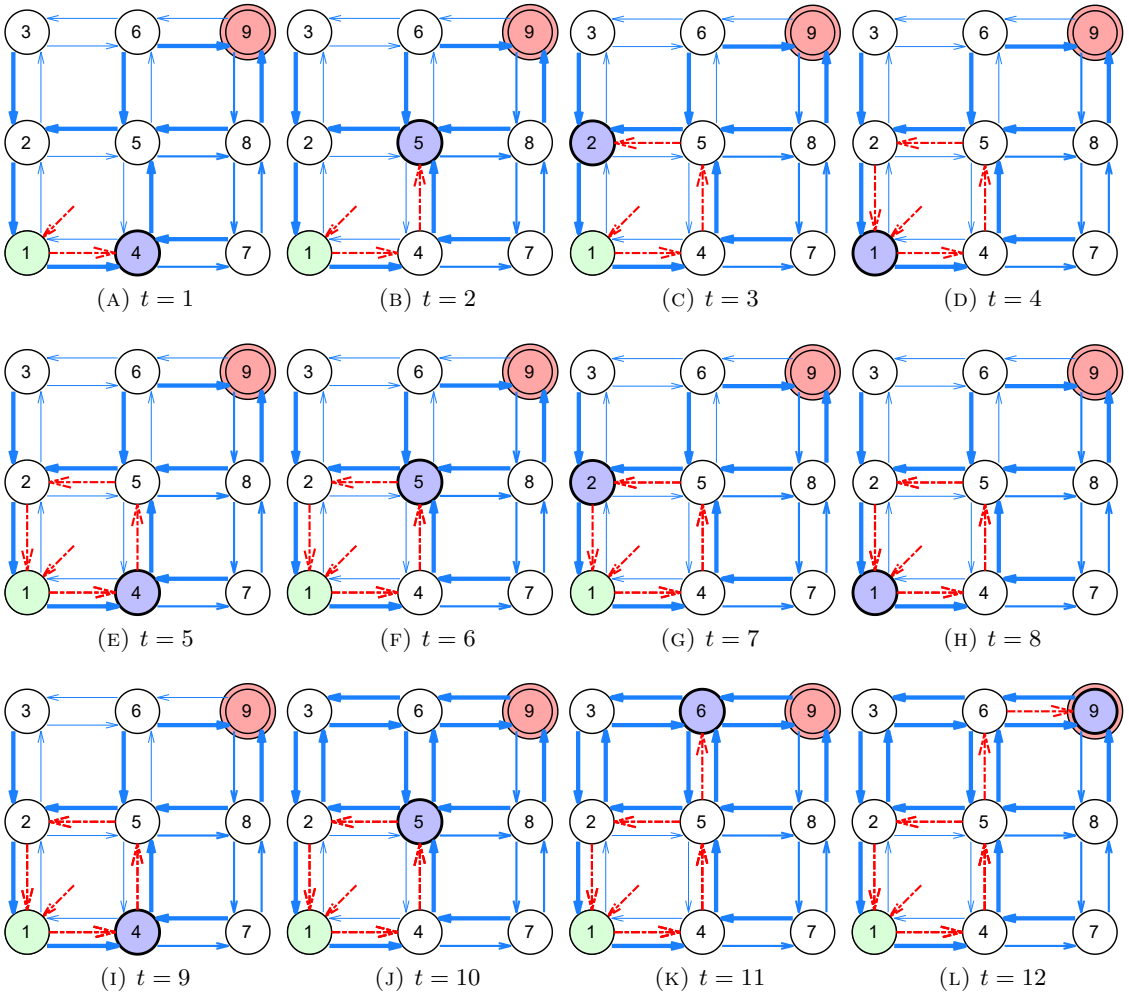


FIGURE 5.4: 3-by-3 example demonstrating optimal policy to reach s_9 (red state) from s_1 (green state). The red lines represent the optimal path starting from s_1 to s_9 and the width of blue lines represents the edge time for the corresponding edge where the thicker width illustrates shorter travel time. The current state is coloured in blue.

5.5.1 3-by-3 grid

This example considers a graph with 9 states as shown in Fig. 5.4, where the aim is to reach state s_9 from state s_1. The width of each edge line (in blue) illustrates the corresponding edge time; the thickest edge width represents an edge time of 1 while that for the thinnest is 25. The path trail across the travel time is drawn in red arrows. The current state is coloured blue.

The initial departure time from s_1 is $t_0 = 0$. At this time, all path that leads to s_9 is expensive (Fig. 5.4a). The edge costs $C_{s_2 s_3}(t)$, $C_{s_5 s_6}(t)$, and $C_{s_3 s_6}(t)$ are reduced at $t = 10$

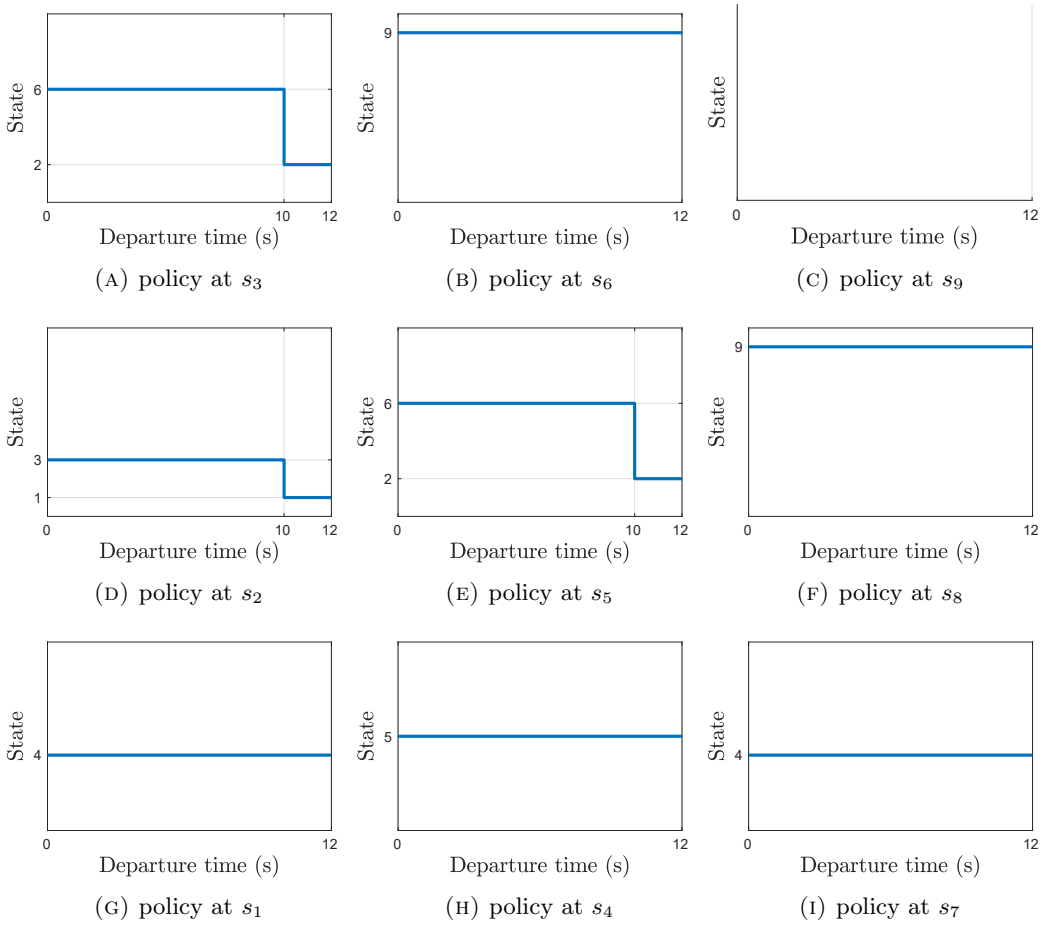


FIGURE 5.5: The optimal policy at each state. This figure is laid out like the grid configuration. There is no defined policy at s_9 as that is the goal state.

(Fig. 5.4j). The optimal policy for each states are derived using algorithms in Section 5.2 and is illustrated in Fig. 5.5. The resulting travel time-optimal path evaluated from the policies is $\Lambda^* = s_1 s_4 s_5 s_2 s_1 s_4 s_5 s_2 s_1 s_4 s_5 s_6 s_9$.

The optimal path starts with two cycles through states $\{s_1, s_2, s_4, s_5\}$ and then reaches the goal state via state s_6. The travel time for this path is 12. State s_5 plays an important role in the optimal policy, deciding when to move towards the goal state, while others' actions are greedily chosen. The travel policy for states s_1, s_2 and s_4 is to move in a cycle regardless of departure time, whereas the policy for state s_5 (Fig. 5.5e) is to transit to s_6 when the time is greater than 10. This solution differs from other conventional TDSP framework, as it makes use of cyclic paths. The optimal policy to reach s_9 from all states at all times was found after 14 iterations and took 0.418 seconds to converge.

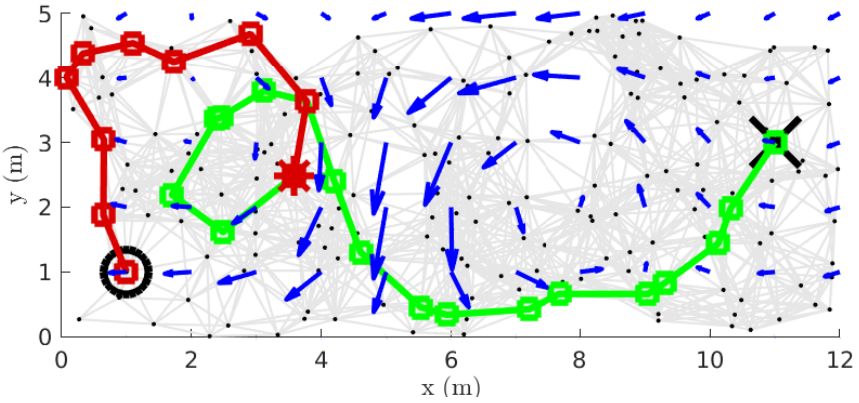
5.5.2 Time-dependent flow fields

This example implements the PF-based TDSP framework for a flow field environment. The time-dependent flow field is represented by a modified Taylor-Green gyre vortex model [63], that sweeps from left to right with respect to time. A probabilistic roadmap is built over the flow field using the PRM* algorithm, as described in Section 5.4. The algorithm randomly sampled 200 states over the space and used a connection radius of $r = 1.735$ as defined in (3.4) to connect them.

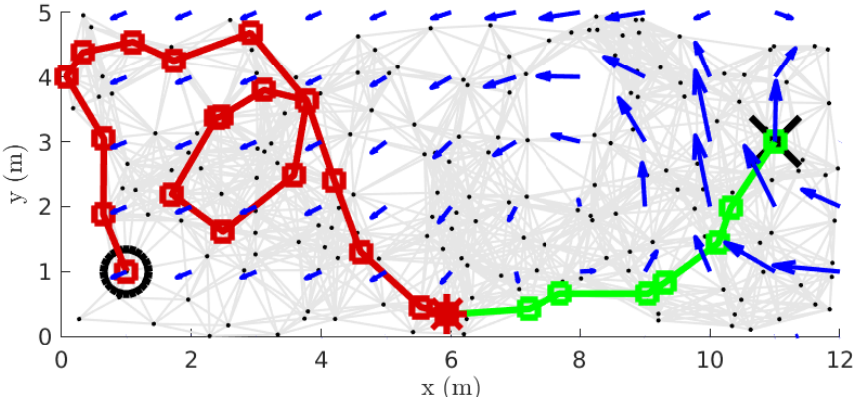
Figures 5.4a-5.4d illustrate its progress over the optimal path for initial departure time $t_0 = 0$ and the flow evolution over time. The vehicle starts from the bottom left corner (circle), aiming to reach the top right corner (cross). The red line represents the trajectory followed up to time t, and the green line is the remaining path. The time-dependnent flow field is shown in blue vector arrows. The PRM* roadmap states are shown in black and its edge connections are shown in grey.

The vehicle considers travelling straight to goal state to be unfavourable due to the current flow field environment. The resulting policy is therefore to spends time on the left side until the flow in the middle weakens. The "waiting" involves cyclic motion using the surrounding eddy currents to return back to the desired position state. The vehicle then moves towards the goal state against a weakened opposing flow. The total travel time is 42.599. The optimal travel policy was found after 44 iterations, and the time to compute the optimal policy was 583 seconds.

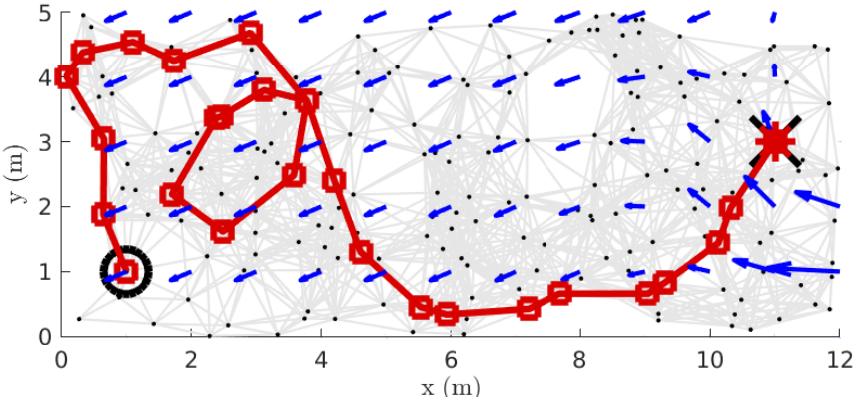
In Fig. 5.7a and 5.7b, the optimal paths for two different initial departure times, $t_0 = 12$ and 25, are demonstrated. These solutions are taken from the same optimal policy that was used to find the optimal path for $t = 0$. Since the flows are more relaxed than departing at $t_0 = 0$, the overall travel times are reduced although the arrival times are later. For many marine robotic operation, optimising travel time is more valuable than optimising arrival time, since their energy constraint is tied to travel time more than the arrival time. As the same sets of policy can be used to find solutions for various start times, the optimal travel time can be evaluated by iterating over the policy solutions.



(A) $t = 10.916$ departing at $t_0 = 0$

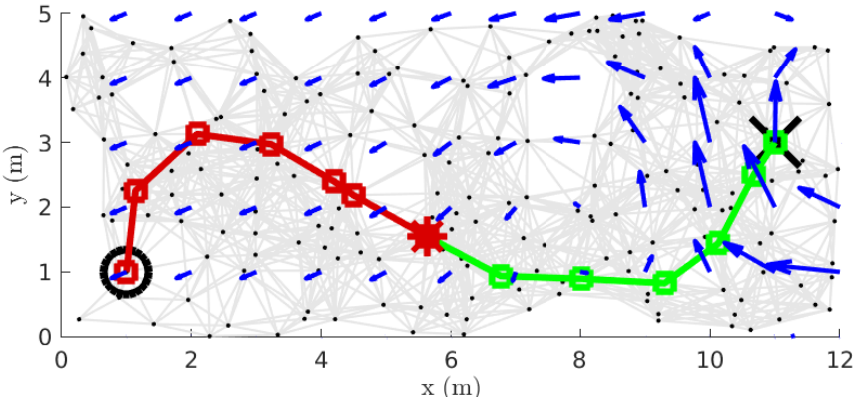


(B) $t = 29.684$ departing at $t_0 = 0$

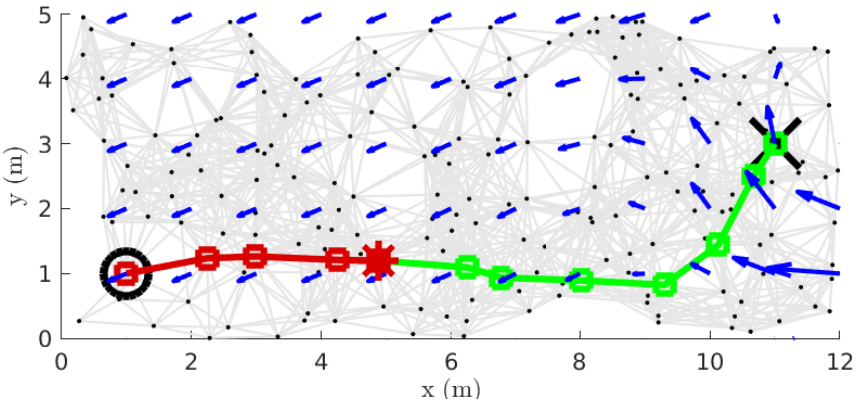


(C) $t = 42.599$ departing at $t_0 = 0$

FIGURE 5.6: The optimal path at initial departure time $t_0 = 0$ through time-dependent flow field from start (black circle) and to destination (black cross). The red line is the path prefix and the green is the suffix. The red asterisk is the current location. Blue arrows represent the flow vector. The PRM*nodes and edges are shown in black. t is the travel time at the prefix since the departure t_0. Total travel time = 42.599.



(A) $t = 31.8243$ departing at $t_0 = 12$ (total travel time = 34.8365)



(B) $t = 40.4315$ departing at $t_0 = 25$ (total travel time = 36.2631)

FIGURE 5.7: The optimal path at initial departure time $t_0 = 12, 25$ through time-dependent flow field from start (bottom left) and to destination (top right). The red line is the path prefix and the green is the suffix. The red asterisk is the current location. Blue arrows represent the flow vector. The PRM*nodes and edges are shown in black. t is the travel time at the prefix since the departure t_0.

5.6 Summary

We have established a solution for the non-FIFO TDSP problem using time dependent piecewise-constant functions in polynomial time. This provides a useful solution for navigation through 2D flow fields that changes with respect to time. This algorithm can also be applied directed to 3D problems. In the next chapter, we look at integrating the trim-based method to generate continuous 3D glider paths, based on the optimal time policy.

Chapter 6

Kinodynamic planning in non-FIFO time-dependent flow fields

This chapter extends the work on underwater flow field navigation by considering an optimal continuous path across time-dependent flow fields. The scope of Chapter 5 only considered the optimal solution in a graph environment. This chapter validates the discrete solution with real-world control schemes and dynamic path. Finding an optimal continuous path in time-dependent flow fields is NP-hard. This chapter addresses the continuous path problem using hierarchical planning by synthesising a continuous control sequence from the graph-based optimal solution. Section 6.1 proposes an alternative construction of the time-dependent graph to make it easier to synthesise the continuous path with real-world controls. This section also presents a hierarchical planning for continuous time-dependent flow field problem using the newly defined graph. Section 6.2 provides an analysis of time complexity and optimality and presents theoretical guarantees. Finally, Section 6.3 presents empirical examples derived from a common flow field model.

6.1 Time-dependent graph for hierarchical planning framework

Previously in Chapter 5, the PRM* algorithm was used to build the time-dependent graph, with the edge cost evaluated using forward integration between two vehicle states. While the state-to-state solution was evaluated with forward integration and therefore can be considered continuous, the overall path must still strictly follow the discrete roadmap. This section presents an alternative method of constructing the time-dependent directed graph over a time-varying flow field model. The new graph promotes the synthesising of a continuous path σ across the discrete graph that is not restricted to pass through sample points as in the PRM* method. This section also presents how an optimal continuous path σ^* is synthesised over the proposed graph.

6.1.1 Building a time-dependent graph in a dynamic flow field

Suppose a continuous flow field environment $\mathbf{X}$ is discretised into uniform regions. Each line segment from the boundaries of the discretisation is defined as a *state line*, ℓ_s. A set of graph states $s \in S$ is situated at the mid-points of those ℓ_s that are not part of the external boundary. The graph defines the set of neighbouring states S_s for all $s \in S$ as the states that lie on the region edges that adjoin ℓ_s. An example graph is illustrated in Fig. 6.1

Similar to the approach in Section 5.4.2, the graph expresses the associated edge time $C_{ss'}(t)$ for each edge $(s, s') \in E$ as a piecewise function with uniform time partitioning, as shown in (5.23). However, the edge time is not evaluated using forward integration across two graph states s and s'. Instead, a set of possible heading controls $\Phi = \{\phi_0, \cdots\}$ are enumerated from state s and the edge time for each time partition is set as the fastest time for a control to cross $\ell_{s'}$. We show later that this approach has minimal impact on the time complexity. No new edge time are evaluated from the position the controls crossed $\ell_{s'}$. As such, there is no guarantee of a connection between each neighbouring continuous path segments evaluated for edge time. The path must be synthesised to create a continuous path from each state lines.

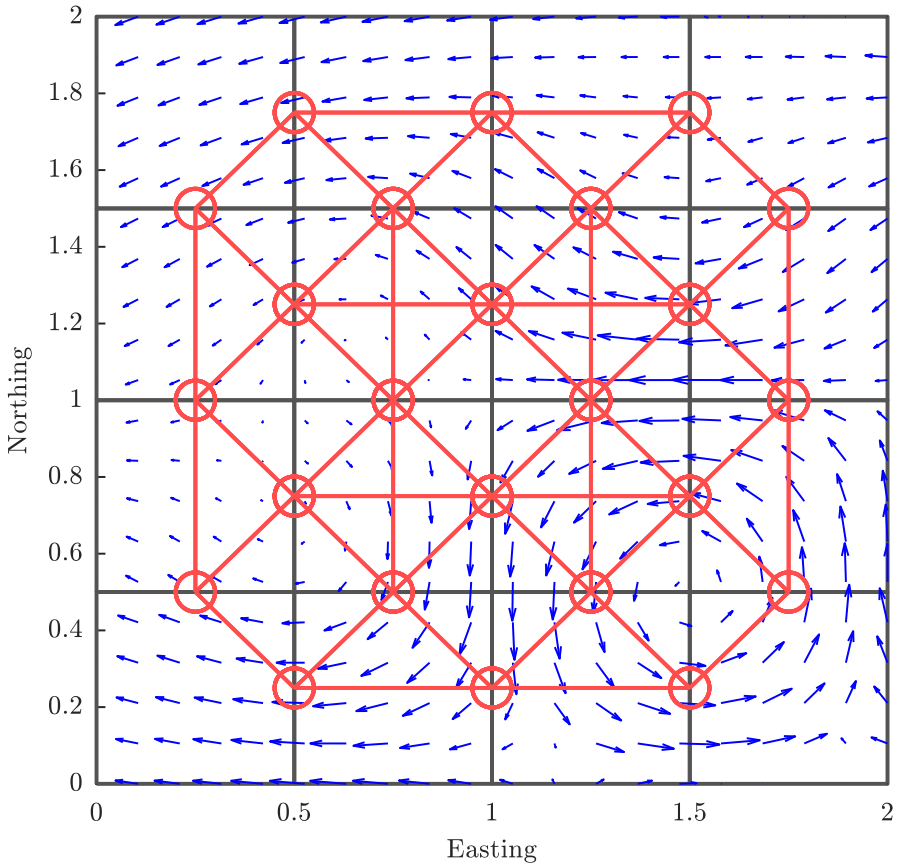
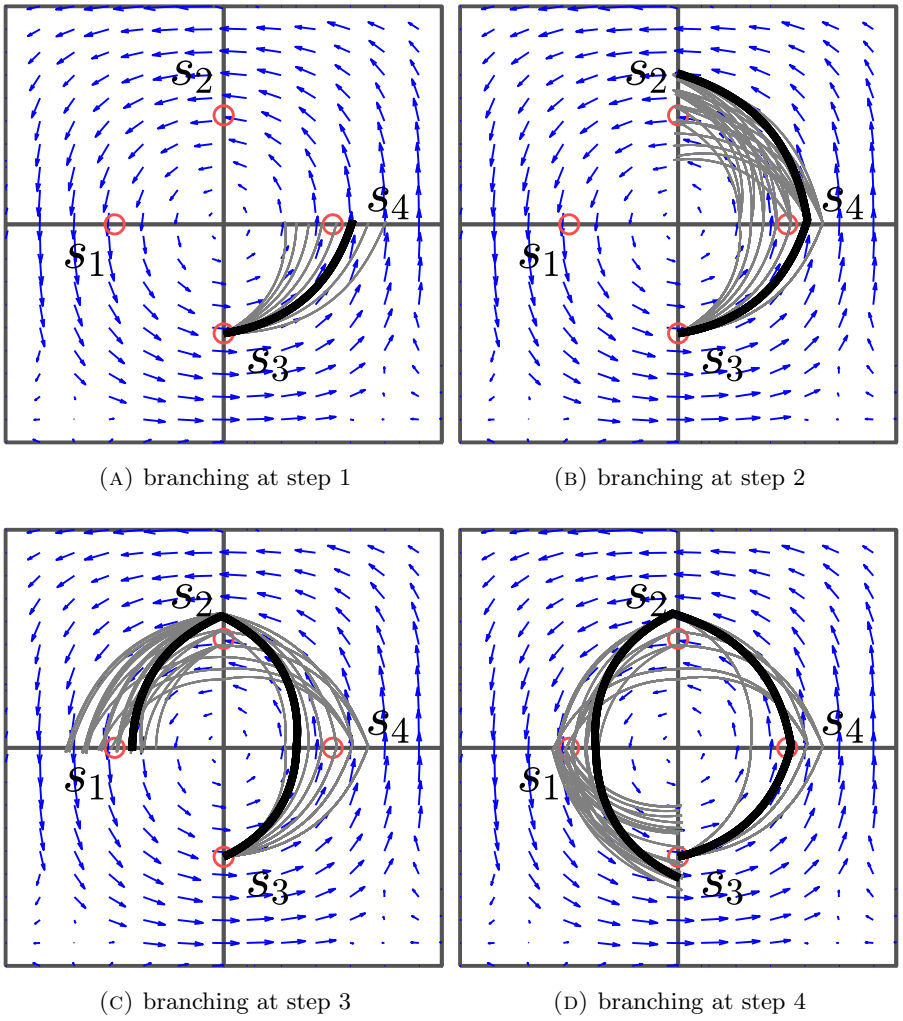


FIGURE 6.1: Example time-dependent directed graph. The environment is evenly distributed into rectangular grids (shown in black). The red circles are the graph states, defined as the mid point of each line on the rectangle. The red line is the graph edge defined by the neighbouring states.

6.1.2 Hierarchical planning for continuous TDSP

The proposed hierarchical planner synthesises a continuous path given an sequence of states $\Lambda = s_0 s_1 s_2 \cdots s_n$ defined by the time-dependent policy as defined in Example. 5.1. A finite set of heading values $\mathbf{U} = \{\phi_0, \cdots\}$ is denoted as the demand heading values the vehicle controllers can select. Given a time-dependent flow field $\mathbf{V}_c(\mathbf{p}(t), t)$ and starting from the initial state s_0 at time t_0, a set of paths are forward integrated for all controls $u \in U$ to find a set of trajectories that reaches the state line, ℓ_{s1}. For each vehicle position that reached ℓ_{s1}, a set of paths are forward integrated to find a set of trajectories that reaches the next state line, ℓ_{s2}. This branching of paths continues until it reaches the end of the sequence, where the trajectory with the fastest total travel time is picked.



(A) branching at step 1 (B) branching at step 2

(C) branching at step 3 (D) branching at step 4

FIGURE 6.2: Step-by-step visualisation of an example of a continuous path branching that follows a discrete path. N closest solutions to the discrete travel time are kept for each step. The trajectory with minimal travel time is highlighted in black.

The branching process only keeps the N best trajectories before branching to the next state line to limit the overall number of branching. At the k-th expansion, the best trajectories are continuous paths with the closest total travel time to T_{Λ_k}. This approach to pruning is necessary for solving the non-FIFO problem. If the branches are only pruned for fast paths and disregard this travel time schedule, the vehicle may arrives too early before the flow field environment has improved. Contrastingly, if the vehicle arrives too late by not keeping track of the travel time schedule, it can result in an overall suboptimal solutions. Let T_{σ_k} be the travel time of the continuous path after k-th expansion. Then, to solve for Problem 4 and Problem 5 with the hierarchical planner, an additional constraint is added:

Problem 7 (Scheduled continuous path)**.** *Given an optimal sequence of states* $\Lambda = s_0 s_1 s_2 \cdots s_n$ *within a time-dependent directed graph* $G = (S, E)$ *for start time* t_0*, time-dependent flow field* $\mathbf{v}_c$*, and some vehicle dynamics, find the continuous path* σ^* *such that:*

$$\min ||T_{\sigma_k}(t_0) - T_{\Lambda_k}(t_0)||, \ \forall k. \tag{6.1}$$

We later show that this still generates optimal continuous paths.

A visualisation of an example case is shown in Fig. 6.2. Suppose a continuous path was to be generated from a discrete path $\psi = s_3 s_4 s_2 s_1 s_3$. The hierarchical planner first propagates the trajectory across the control input $\mathbf{U}$ from s_3, picks N trajectories that intersect ℓ_{s_4} with the travel time closest to T_{Λ_1}, and prune away the rest. The trajectories are then expanded from each end point on ℓ_{s_4} across $\mathbf{U}$, then keeping N trajectories that intersect ℓ_{s_2} with the travel time closest to T_{Λ_2}. This continues until the end of ψ, resulting in a trajectory tree starting from s_3 with N branches intersecting the goal state line ℓ_{s_3}. The figure also shows the benefit of the new approach; the resulting valid trajectories do not need to pass the sampled positions.

6.2 Analysis

Solving for general non-FIFO TDSP over discrete space (i.e., graph) is NP-hard [28, 29]. In order to make use of efficient solutions to special cases of TDSP problems for problems in continuous spcae, as we do in this chapter, the additional challenge needs to be addressed. Continuous problem needs to be cast in a form that can be addressed as a TDSP problem and then restructure back to a continuous solution. The proposed hierarchical approach divides this problem by first solving for a discrete travel time-optimal policy, and then synthesising for a continuous path. This section demonstrates that the synthesised continuous path retains the optimality properties from the discrete solutions, and show that the algorithm as a whole is solvable in polynomial time. The section also remarks that the proposed framework is complete in both discrete and continuous state space.

Theorem 6.1 (Optimality of continuous path)**.** *Given an travel time-optimal sequence of states $\Lambda^* = s_0 s_1 s_2 \cdots s_n$ within a time-dependent directed graph $G = (S, E)$ (as defined in Section 5.2), time-dependent flow field $\mathbf{V}_c(\mathbf{p}(t), t)$, and some vehicle dynamics, the continuous path generated from the hierarchical planner proposed in Section 6.1.2 retains the travel time-optimality of Λ^*.*

Proof. The hierarchical planner generates the continuous path by finding a uniform control within each region in the discrete sequence. During each expansion, only N-best branches with travel time closest to that of the discrete solution are kept. The final continuous path is therefore optimal with respect to discrete regions. This implies that as the size of each region reduces, the continuous path would also approach the travel-time optimal solution in continuous state space. □

Theorem 6.2 (Time complexity for hierarchical planner)**.** *The time complexity of the hierarchical planner is linear in the number of discrete regions and polynomial in the number of variations in flow.*

Proof. The time complexity for finding an optimal discrete path is $\mathcal{O}(|S||C|^k + |\Phi||S|)$, where $|C|$ is the worst case number of variations in flow, k is number of edge transitions, $|\Phi|$ is the number of discrete controls and $|S|$ is the number of discrete sequences. Given a discrete sequence Λ, we find a continuous path for each region for $|\Lambda|$ times. Therefore, the time complexity for finding a continuous path is $\mathcal{O}(|\Lambda||U|)$. The overall complexity is $\mathcal{O}(|S||C|^k + |\Phi||S| + |\Lambda||U|)$. Since edge costs are found by enumerating all controls U and Ψ for forward integration, the part can be parallelised. The overall complexity with parallel processing is $\mathcal{O}(|S||C|^k + |S| + |\Lambda|) = \mathcal{O}(|S||C|^k + |\Lambda|)$. In either case, the time complexity is linear in the number of discrete regions and polynomial in the number of variations in flow. □

Remark 6.3 (Completeness)**.** In continuous path generation, our aim is to find a control that triggers a transition to the designated next region within a time window. Under the assumption that the flow within a region is uniform, there always exists a continuous control from one edge to another. Therefore our framework is complete in both discrete and continuous state space.

6.3 Examples

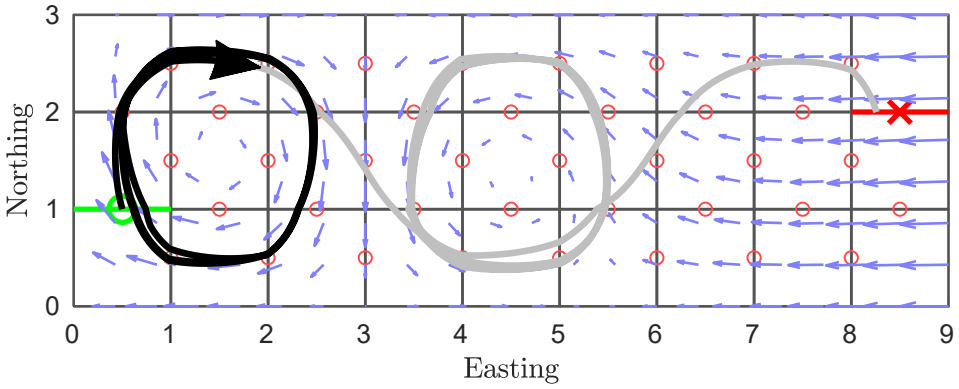
In this section, we present two case studies: an ASV travelling in a 2D time-dependent flow field and an underwater glider travelling in a 3D time-dependent flow field. The case studies demonstrate our algorithm's ability to handle various dimensions, dynamics, and problem sizes. All distance units are in meters and time units are in seconds unless otherwise stated.

6.3.1 Autonomous surface vehicle (ASV) case

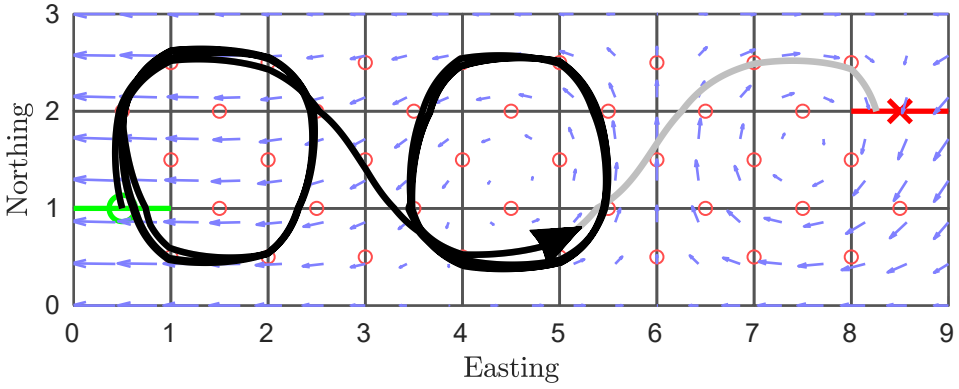
This case study generates a continuous path that follows the time-optimal discrete path starting from $t_0 = 0$. It also evaluates the optimal start time t_0^* using a standard optimisation technique, and generate a continuous path for that start time.

An ASV with a constant forward velocity of 0.5 m/s travelling from $[0.5, 1]^T$ to $[8.5, 2]^T$ through a 2D time-dependent flow field is considered. A westward current of up to 1.2 m/s dominates around the goal state, making it impossible for the ASV to travel directly east towards the goal. Vortices with radius 1.5 m appear at $[1.5, 1.5]^T$ for $t \in [0, 15]$, at $[4.5, 1.5]^T$ for $t \in [10, 30]$, and at $[7.5, 1.5]^T$ for $t \in [25, \text{inf})$. A graph with 42 states was built by discretising the environment into a 9x3 grid. The algorithm defined in Chapter. 5 evaluated the time-optimal discrete path over the constructed graph.

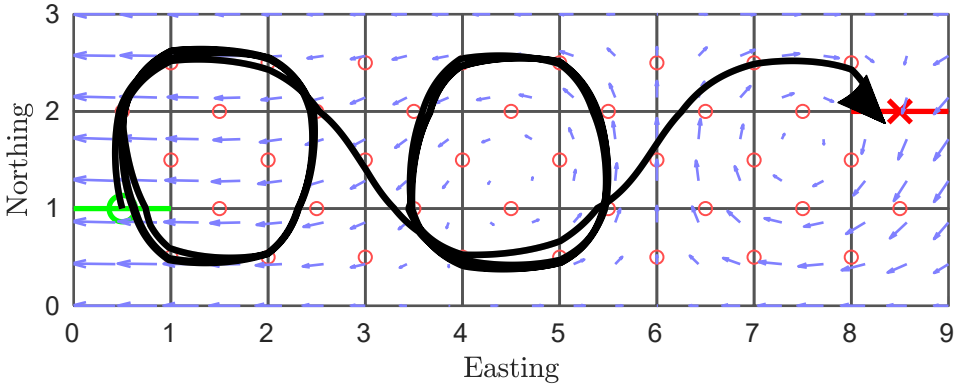
Figures 6.3a to 6.3c show time instants along the continuous path starting at $t_0 = 0$. The travel time for discrete and continuous path were $T_\psi = 31.95$ and $T_\sigma = 32.53$. The continuous ASV path visualises the discrete cyclic solution used to loiter in anticipation of advantageous transitions in the flow field. The ASV loops around the first vortex three times to wait for the time when the transition from the first to the second vortex is possible. After transitioning to the second vortex, the paths cycle another three times until it is feasible to transition into the third vortex and reaches the goal line. As demonstrated in Section 6.1.2, the synthesised continuous path was not required to cross the discrete graph states. Instead, the synthesised path crosses the state line in the order of the equivalent discrete state sequence. This flexibility ensures the ASV can reach the goal line using the policy evaluated from a piecewise-constant edge time.



(A) ASV path at 12.4 second after departing at $t_0 = 0$ departure

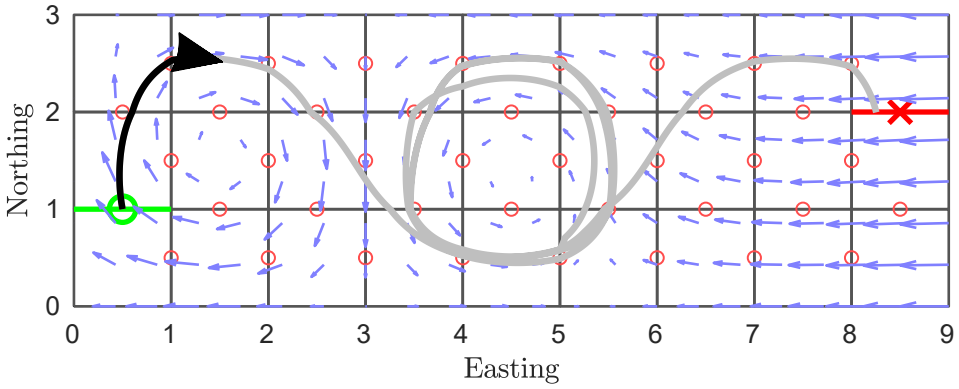


(B) ASV path at 28.2 second after departing at $t_0 = 0$

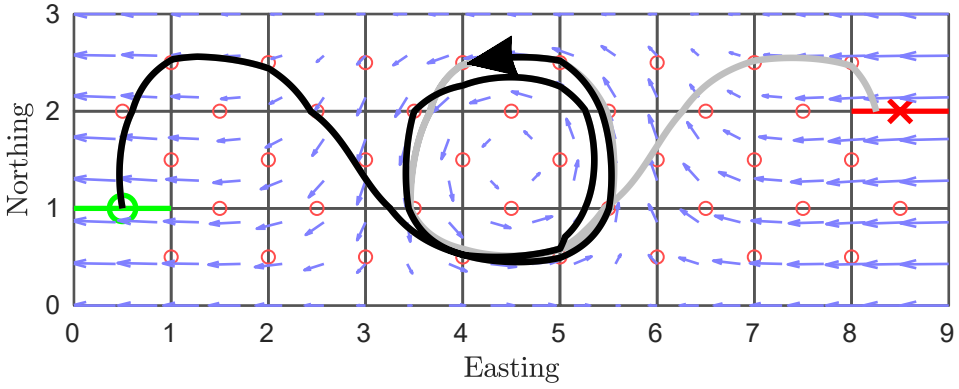


(C) ASV path at 32.53 second after departing at $t_0 = 0$

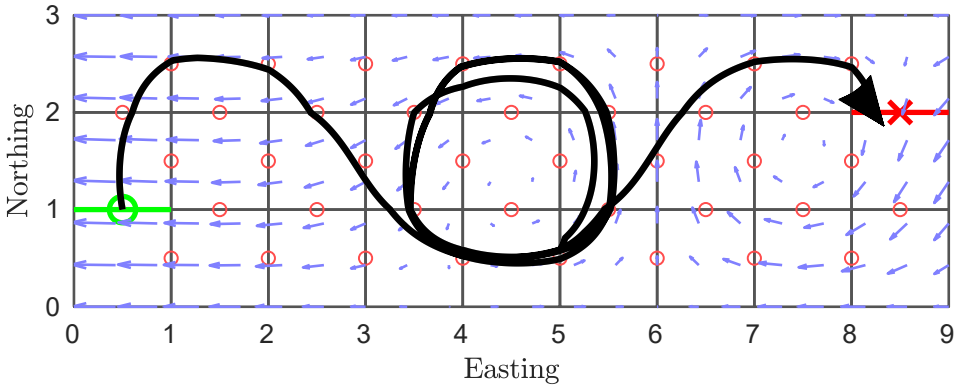
FIGURE 6.3: 2D continuous path of an ASV from $[0.5, 1]^T$ (green circle) to $[8.5, 2]^T$ (red cross) with a constant velocity of 0.5 m/s and deployed at $t_0 = 0$. A strong current flowing west opposes the glider's path to the goal. Figures (a)-(c) show the path progression across three separate time steps, where the black path is the progressed trajectory and the black arrow head is the glider position and heading. The gray path is the complete trajectory. Red circle is the graph states. Blue quiver is the time-varying flow field.



(A) ASV path at 10.1 second after departing at $t_0 = 0$ departure



(B) ASV path at 19.6 second after departing at $t_0 = 0$



(C) ASV path at 29.1 second after departing at $t_0 = 0$

FIGURE 6.4: 2D continuous path of an ASV from $[0.5, 1]^T$ (green circle) to $[8.5, 2]^T$ (red cross) with a constant velocity of 0.5 m/s and deployed at $t_0 = 9$. A strong current flowing west opposes the glider's path to the goal. Figures (a)-(c) show the path progression across three separate time steps, where the black path is the progressed trajectory and the black arrow head is the glider position and heading. The gray path is the complete trajectory. Red circle is the graph states. Blue quiver is the time-varying flow field.

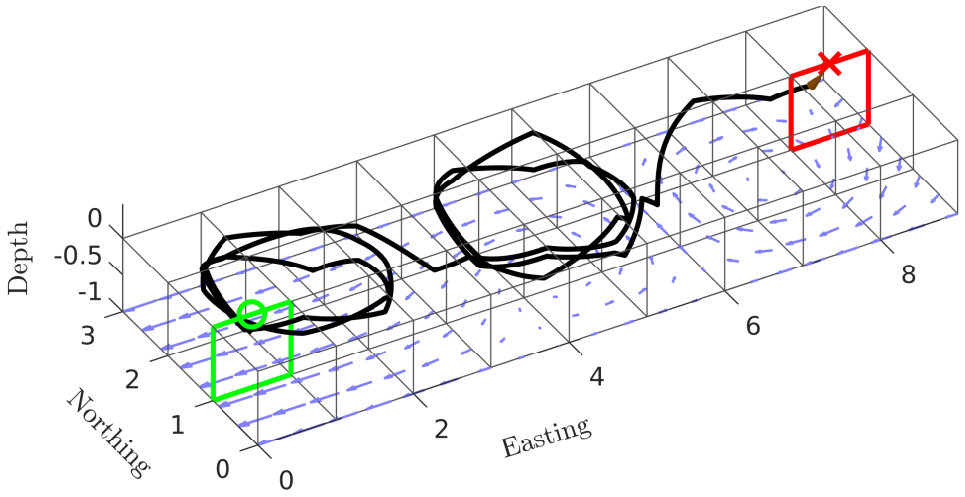
In Section 5.5.2 the discrete optimal policy defined path solutions for multiple start time and could be used to evaluate the optimal start time using standard optimisation method. Therefore, a continuous path for the optimal start time can also be evaluated. Figure 6.4 shows the trajectory for the optimal starting time $t_0^* = 9$. The travel times for discrete is $T_\Lambda = 19.75$ and the travel time for continuous path is $T_\sigma = 20.1$. Figure 6.4a shows that the ASV is deployed just as the path between the first and second vortex opens, allowing it to proceed straight through. If deployed at $t_0 = 0$, the ASV would be forced to loiter with the cycle period that would cause over-delay. The rest of the path behaviour is the same as Fig. 6.3.

6.3.2 Underwater glider in 3D ocean current case

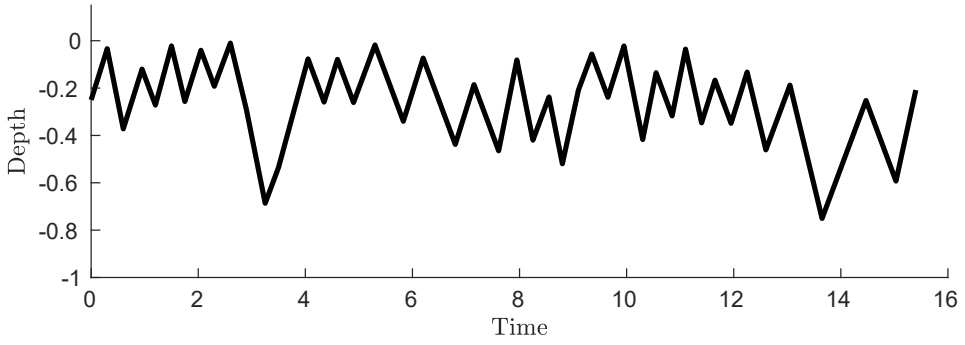
This case synthesis a continuous path using underwater glider dynamics that follows the time-optimal discrete path commencing at $t_0 = 0$. A hypothetical underwater glider presented in Section 3.1 is considered with a top horizontal speed of 0.868 m/s. The edge time was evaluated using the average glider horizontal speed of 0.7435 m/s.

The glider typically operates in a sawtooth motion, using both ballast and glide angle to control its net velocity. Recall from Chapter 4 the a sparse control scheme for the glider model can be implemented using *trim-state control*. This control scheme uses *trim-state*, that is, the maintained state of dynamic equilibrium under no disturbance or control variation.

Both the flow field environment and the time-dependent graph are similar to those described in Section 6.3.1. The start and goal positions are the same as in the previous case. The flow field has been extended across the depth to allow for manoeuvre in 3D. Consequently, the state line ℓ_s is projected into a *state plane*, $\mathcal{P}_s$. A continuous path is now synthesised by crossing the state planes in sequences defined by the discrete solution. For simplicity, we set the flow velocity to be the same for all depths. For the edge time evaluation $C_{ss'}(t)$, a set of trajectories is forward integrated from s across the glide and heading angle in a 3D flow field, and pick the fastest trajectory time that intersects with $\mathcal{P}_{s'}$.



(A) Underwater glider path in 3D time-varying flow field



(B) Underwater glider depth profile with respect to time.

FIGURE 6.5: Figure 6.5a shows the continuous path across a time-varying 3D flow field with underwater glider dynamics, travelling from $[0.5, 1, 0]^T$ (green circle) to $[8.5, 2, z]^T$ (red cross) where the goal point is at any depth. The path is shown in black and the flow field at the final time step is shown in blue quiver. The travel times for the discrete and continuous paths were both 15.4 sec. The sample points are omitted for clarity. Figure 6.5b shows the glider depth profile with respect to time.

Figure 6.5 shows the continuous 3D path with $t_0 = 0$ with travel times for discrete and continuous path found as $T_\Lambda = T_\sigma = 15.4$. The continuous trajectory illustrates properties of the glider dynamics, as the manoeuvres exhibit a sawtooth motion. Otherwise, we see the same decision making we saw with the ASV example: the glider loops around the first vortex twice before entering the second vortex, where it loops twice again until the final transition to the goal is feasible. Furthermore, as the sawtooth motion controls the glider velocity, there is greater control on T_σ to match T_Λ. In this framework, the goal state does not define which depth the glider should arrive at each $\mathcal{P}_s$. The glider is said to reach the goal once it cross the goal plane.

6.4 Summary

In this chapter, we established the hierarchical approach to continuous path planning in time-dependent flow field environment. The method makes use of the vehicle dynamics, making it a valid solution for both 2D and 2.5D flow field using realistic dynamic model. In the next chapter, we demonstrate our contribution using simulations of real world examples, both time-static and time-dependent.

Chapter 7

Real world simulations and applications

This chapter demonstrates the algorithms developed in Chapters 4-6 for real-world applications. One of the main goals of this thesis is for the underwater glider to perform real-world marine missions efficiently while minimising human interactions. Section 7.1 describes a simulation of an energy-optimal underwater glider path across a static East Australian Current using the trim-based FMT* planner, defined in Chapter 4. Section 7.2 describes a simulation a time-optimal underwater glider path across a time-varying East Australian Current using the heuristic planner with the TDSP framework, defined in Chapter 6. Section 7.3 extends the work in Chapter 6 by having the RRT algorithm synthesising a continuous path across the TDSP framework. It demonstrates the modularity of the hierarchical planning method regarding synthesis of a continuous path.

7.1 Planning in real-world static ocean environments

This section demonstrates real-world applicability of the trim-based FMT* algorithm, presented in Chapter 4, by planning an energy-optimal path with respect to the underwater glider dynamics model across the East Australian Current (EAC). A model was obtained from the School of Mathematics and Statistics at the UNSW [120]. Each grid cell in

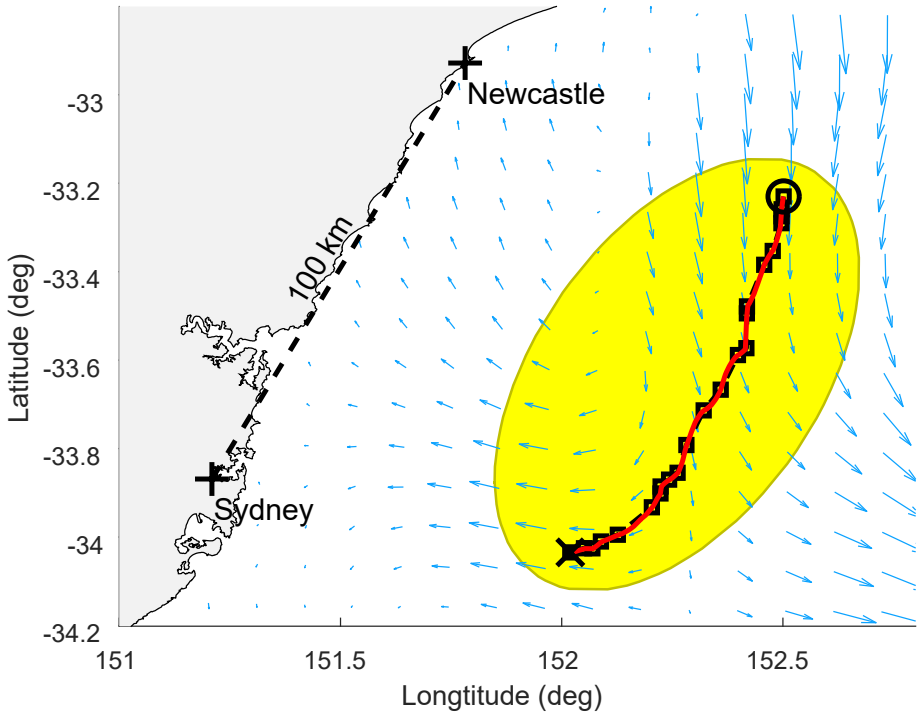


FIGURE 7.1: The EAC environment. The yellow ellipse represents the planning region. The black circle is the start position, and the black cross is the goal position. The underwater glider is tasked to find a path from $(152.5°, -33.23°)$ to $(152°, -34.03°)$ that minimises the energy expenditure; travelling ~ 100 km along the east coast of Australia from Newcastle to Sydney. The current-influenced trajectory and the edge connections are shown as red and dashed black lines, respectively. The surface flow fields are shown as blue quivers

the data set extends 3.255 km, 5.527 km and 0.165 km apart in x, y and z-directions, respectively. For the purpose of demonstrating the trim-based FMT* algorithm, we assume the EAC is time-invariant.

7.1.1 Planner implementation

For demonstration purposes, the trim-based FMT* is asked to find a path that minimises the energy cost, described in Section 4.1.3, from off the coast of Newcastle to Sydney that are approximately 100 km apart. In geographical terms, the glider traverses from $(152.5°, -33.23°)$ to $(152°, -34.03°)$. Like the simulation examples shown in Section 4.4, the framework is implemented using MATLAB. The underwater glider model was described in Section 3.1 that has been reduced to a 6-dimensional kinematic model using trim-state defined in Section 4.1. The kinematic states are in SI units with angles in

radians. As the dataset provides the flow field information in a discrete form, the ocean current at points in between grid cells are obtained through interpolation. The trim-based FMT* was implemented over sampled nodes within the pre-specified, yellow elliptical region shown in Fig. 7.1. Nodes in the depth dimension were sampled every 200 m from 50 m to 650 m.

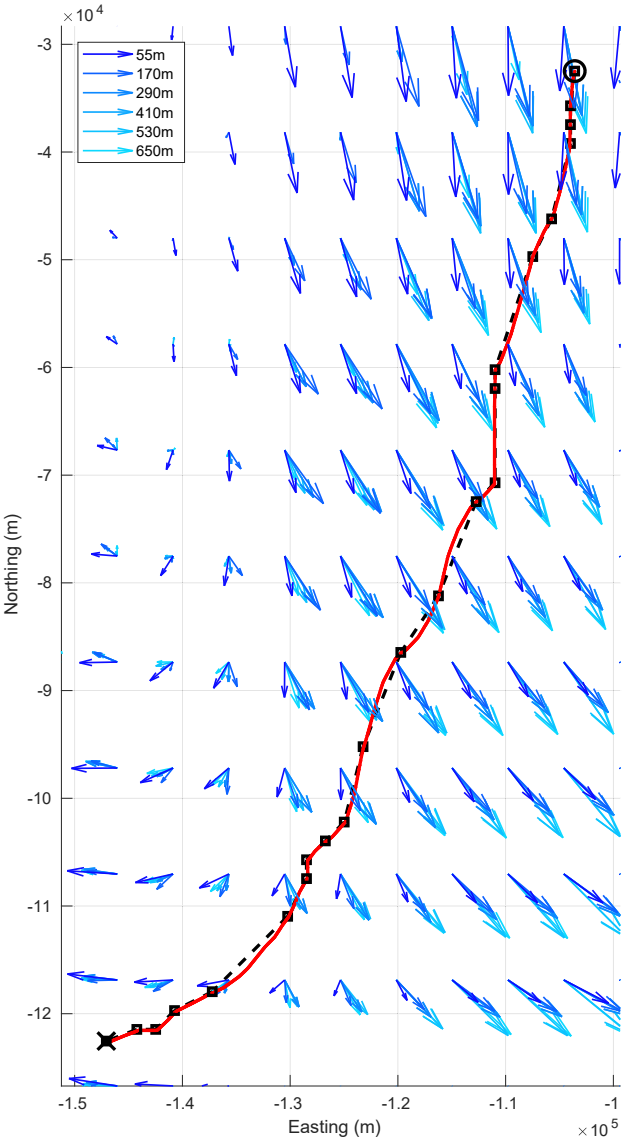
The simulation imposed the same constraints on the glide and heading angle as done in Section 4.4. When transitioning from glider state to trim state and vice versa, the maximum change to the glide angle is $\pm 30^\circ$, and the maximum change to the heading angle is $\pm 45^\circ$. The simulation was performed using a 3.1GHz CPU and 128 GB of RAM.

7.1.2 Energy-optimal path for underwater gliders across the EAC

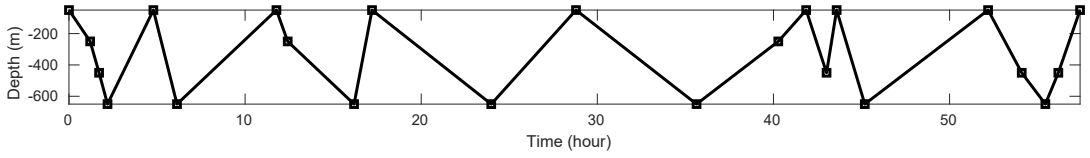
Figure 7.2a shows a more detailed view of the underwater glider path that was shown in Fig. 7.1. The path duration was 57.4 hours. Instead of diving directly towards the goal, the figure shows the glider path riding along the edge of a large vortex driven by the EAC. This result is similar to a solution presented in Fig. 4.8b.

The number of waypoints in the glider path are sparser in the middle region compared to other parts of the path. Inspecting the ocean current around the middle region reveals that the flow fields are more consistent across the depth than other regions. Also, the ocean current vector in the middle region does not offer any advantage or hindrance to the intended glider direction. As such, the glider only needs to minimise buoyancy inversion cost by travelling with sparse controls, whereas the more current dynamic start and goal regions require more control points.

Figure. 7.2b shows the glider's depth profile over time. The path is an irregular sawtooth, featuring mid-path control nodes. Recall from Fig. 7.2a, mid-depth control is needed to compensate for changing flow field properties across depth. This control strategy is most apparent around the initial and goal regions of the path, where the flow varies significantly with depth. There are also instances of different buoyancy inversion depth, which demonstrates the algorithm seeking to minimise the control buoyancy cost whenever possible. This result is similar to the solution presented in Fig. 4.6a. The evidence of



(A) A zoomed-in view of the glider's path



(B) Underwater glider depth profile with respect to time.

FIGURE 7.2: Figure 7.2a shows a more detailed view of the optimal path shown in Fig. 7.1. The current-influenced trajectory and the edge connections are shown as red and dashed black lines, respectively. The flow fields are shown as blue quivers, where the colour intensity represents different depth layers according to the legend. Figure 7.2b shows the depth profile for the trim-based FMT* path shown in Fig. 7.1. The computation time was 11.2 hours. The expected mission execution time is 57.4 hours.

intelligent navigation in a real-world oceanic environment shows the lack of need for human operators to manual plot the glider path.

The framework took 11.2 hours to find an optimal solution, which is relatively fast planning compared to the mission duration of 57.4 hours. The relative fast planning demonstrates the possibility of planning for a subsequent mission while executing the current mission. The algorithm could just as well re-plan the glider path in case of failure or changes in the oceanic environment.

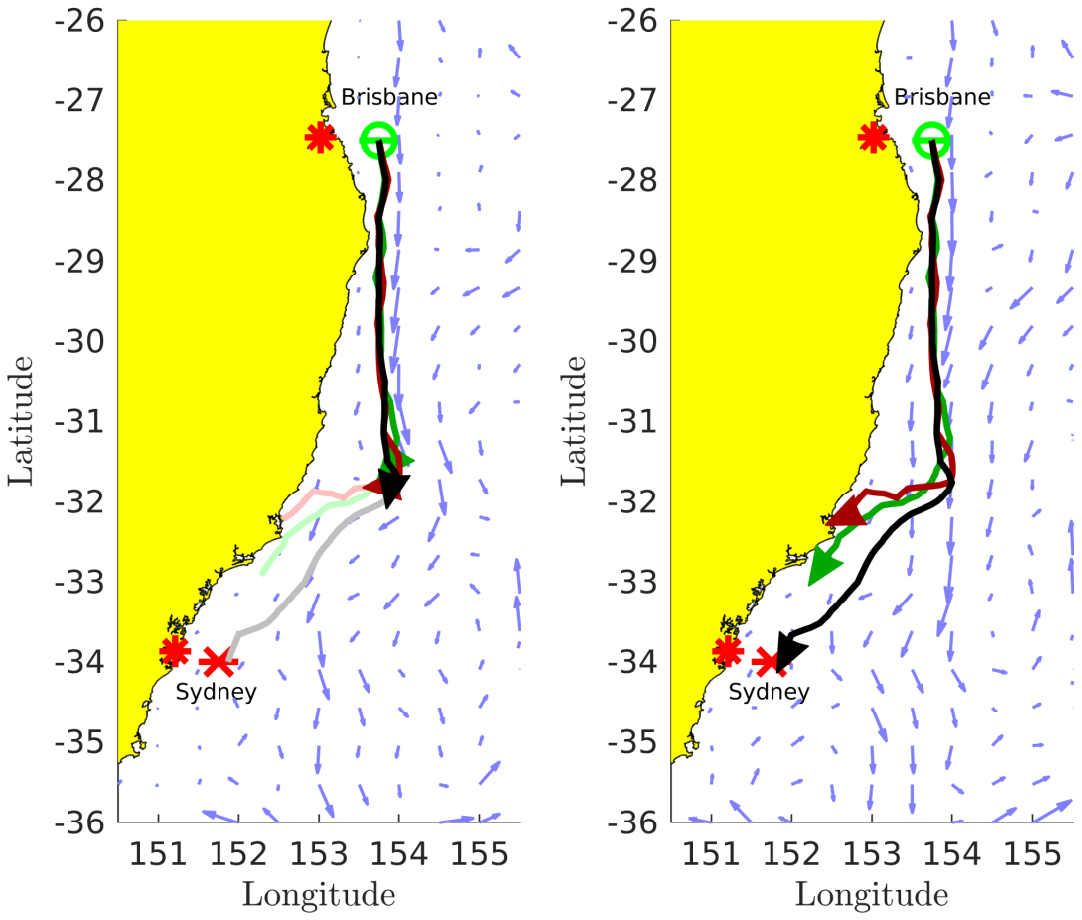
7.2 Planning in real-world dynamic ocean environments

This section demonstrates real-world applicability of the hierarchical planner with TDSP framework presented in Chapter 6, by planning a time-optimal continuous path across a dynamic EAC. For ease of demonstration, the dynamic used in this example is for the Autonomous Surface Vessel (ASV). However as demonstrated in Fig. 6.5, the glider-based trim-state can easily be implemented. The EAC dataset is the same as Section 7.1, but is now time-varying and sampled for a different date. The dataset has the time component sampled daily.

7.2.1 Planner implementation

The hierarchical planner generates a travel time-optimal path from Brisbane $(-27.5°, 154°)$ to Sydney $(-34°, 151.5°)$ along three different oceanic conditions; the time-varying EAC, the static EAC flow field fixed at $t = 0$ days, and the static EAC flow field fixed at $t = 9$ days. Like in Section 7.1.1, the ocean current value between the grid cells are found through interpolation. For the time-varying flow field, the current value is also interpolated with time.

The ASV has a constant forward velocity of 0.3 m/s and assumed deployed at the earliest time of the dataset (i.e., $t_0 = 0$). The hierarchical planner requires a time-dependent directed graph to first solve for the time-optimal discrete path using the TDSP framework, proposed in Chapter 5. The graph is built by first discretising the environment into a 20x10



(A) 7.33 days (B) 13.596 days

FIGURE 7.3: 2D continuous path from Brisbane to Sydney in a time-dependent East Australian Current representation. The vehicle velocity is set to a constant value of 0.3 m/s. The travel time for discrete and continuous path were $T_\Lambda = 13.375$ days and $T_\sigma = 14.7$ days. The green circle is the vehicle's starting point and the red cross is its end point. The environment was discretised on a 20x10 grid to build the time-dependent directed graph (not shown for clarity). The black path is the vehicle path generated from the heuristic planner. The green and red paths are the vehicle path generated from the assumption the flow fields were fixed at $t = 0$ days and $t = 7$ days, respectively. The vehicle position and heading are represented by their respective colored quiver.

grid, then generating the states and its edge connections as defined in Section 6.1. The resulting number of states is 370.

The purpose of this example is to demonstrate the necessity of planning in time-varying flow fields by comparing the disparity against time-static assumptions. The example also demonstrates the applicability of the heuristic algorithm in a real-world environment.

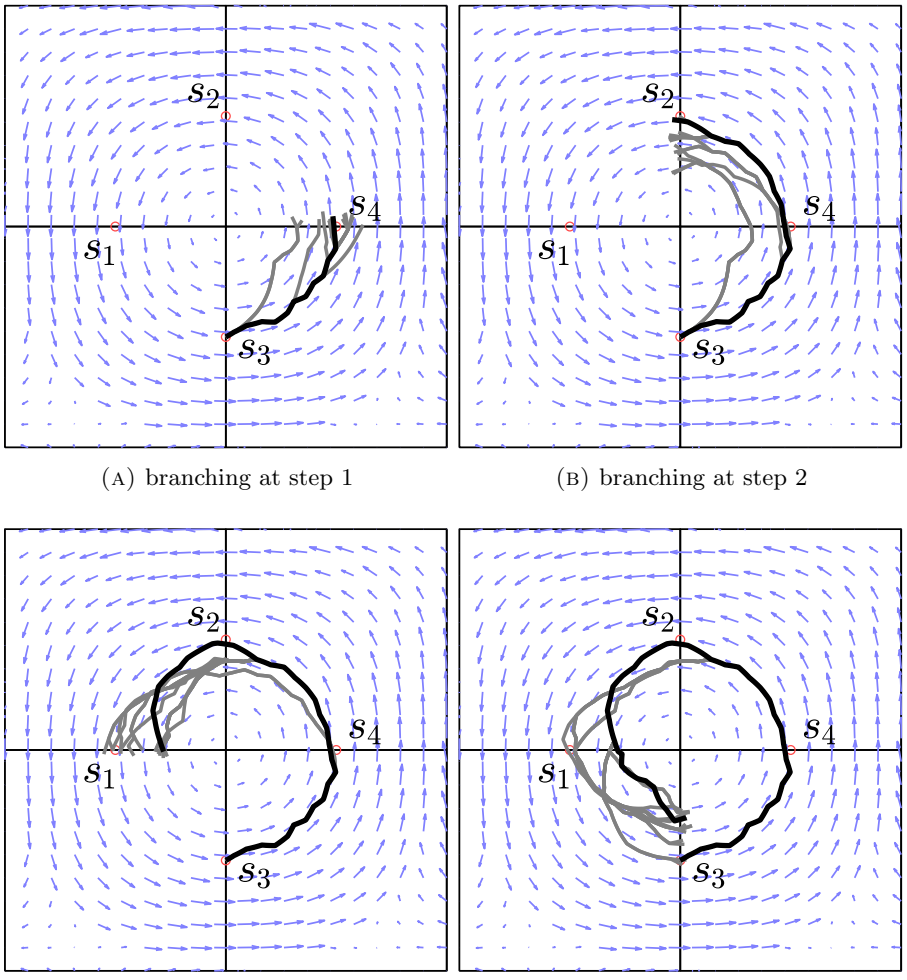
7.2.2 Travel time-optimal path across a dynamic EAC

Figure 7.3 compares the path generated with the time-varying flow field (black path) with paths generated with the assumption static flow field at $t = 0$ days (red path) and $t = 9$ days (green path). Figure 7.3a shows that all three paths initially follow the same path for the first 7.33 days, after which they start to diverge as the flow field environment changes. Figure 7.3b shows the final ASV position for each flow field assumptions. The black path arrives at the Sydney goal with a discrete travel time of $T_\Lambda = 13.375$ days and the continuous travel time of $T_\sigma = 14.7$ days. The ASV follows the movement of the strongest part of the EAC to minimise the time to the goal. The red path caused the ASV to collide with the Australian coast after 11.35 days. The green path completes its trajectory with the ASV stranded at 130.2464 km away from the goal. In comparison, the black path delivered the ASV to a point 11.2504 km away from the goal. This relatively minor position error is due to approximations made when generating the continuous path. This result demonstrates the need for planning in time-varying oceanic environment.

As with the trim-based implementation in Section 7.1.2, the computation time was also favourable. The heuristic planner was able to solve for the discrete path in 440.77 sec and the continuous path in 642.26 sec, compared to the mission time of 14.7 days. The algorithm is therefore potentially suited for planning multiple missions while executing the current mission, on-site planning, and re-planning paths in case of failure or changes in the ocean forecast. The computation time can be reduced by tuning the N number of accepted paths and the time step when propagating the continuous path. The evidence of the heuristic algorithm adapting to the time-varying oceanic environment relieves the human from constant vehicle monitoring. In the case of a re-plan, the algorithm can generate a solution quick enough without human intervention.

7.3 RRT synthesis for TDSP framework

The approach to synthesis a continuous path from the discrete TDSP solution, presented in Chapter 6, only allows control changes on the state line ℓ_s. This discrete control scheme is ill-suited for navigating through a cluttered environment. In a real-world environment,



(A) branching at step 1 (B) branching at step 2

(C) branching at step 3 (D) branching at step 4

FIGURE 7.4: Step-by-step visualisation of a continuous path branching that follows a discrete path using the RRT algorithm. The environment and graph setup is identical to the example shown in Fig. 6.2. N closest solutions to the discrete travel time are kept for each step. The trajectory with fastest travel time is highlighted in black.

small islands, buoys and other naval installation would represent these obstacles. One approach to address this issue is to brute force a solution by partitioning the environment into denser grids, allowing for more frequent control changes. This section instead introduces a more natural approach in synthesising a continuous path using the RRT motion planning algorithm. An overview of how RRT works is found in Section 3.2.2.

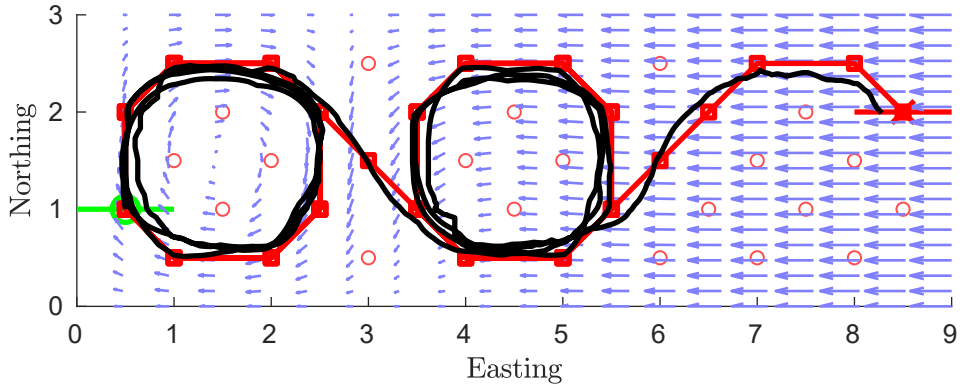
7.3.1 Planner implementation

The planner framework is largely unchanged from the proposed method in Section 6.1.2. But instead of forward integrating a set of controls from a state on state line ℓ_{s_k} to the next state line $\ell_{s_{k+1}}$, the RRT algorithm is used. As the trajectory only needs to connect between ℓ_{s_k} and $\ell_{s_{k+1}}$, RRT only needs to samples states within the region that includes both state lines. Also, because the RRT path solution is biased to the discrete schedule solution, each trajectory connection retains probabilistic completeness.

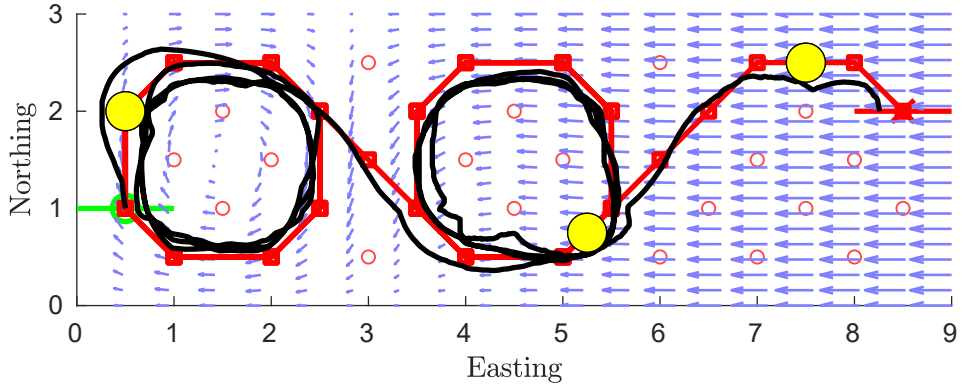
Figure 7.4 visualises the continuous path synthesis using RRT across a discrete path $\psi = s_3 s_4 s_2 s_1 s_3$. Both the flow field environment and the discrete path is the same case as shown in Fig. 6.2. Like with the forward integration approach, the RRT algorithm first expands its branch from the initial state s_3. The algorithm samples M states in the regions that contain both ℓ_{s_3} and ℓ_{s_4}. The planner keeps the N best trajectories with the closest travel time to the discrete solution that reached ℓ_{s_4}. For each trajectory point on ℓ_{s_4}, the RRT algorithm expands new trajectories towards the next state by only sampling in the region that contain both ℓ_{s_4} and ℓ_{s_2}. After each expansion, more trajectories are pruned away to keep only the N best solutions. From the resulting branch of paths, the trajectory with the fastest travel time is chosen as the solution.

7.3.2 Synthesising RRT path

Figure 7.5a demonstrates the RRT planning algorithm synthesising the continuous path across the discrete TDSP solution in a time-varying flow field with no obstacle. The profile of the RRT path appears similar to the continuous path shown in Fig. 6.3. The travel time of the RRT path was 32.7 sec, while the travel time of the continuous path was 32.53 sec. The RRT path solution was slower due to its jagged path, which is common in an RRT path. This example demonstrated that a motion planning algorithm could synthesis a continuous path across the TDSP framework. It also demonstrated that the path solution retains the planner's properties. As such, a state-of-the-art motion planning algorithm, such as RRT* and FMT*, could be used for a better path solution.



(A) No obstacle case



(B) Obstacle case

FIGURE 7.5: RRT algorithm synthesising a path across the TDSP framework in a time-varying flow field environment. The blue quiver represents the flow field at the initial start time t_0. The red circle are the states of the time-dependent directed graph. The red path is the discrete path solution and the black line is the synthesised path using the RRT algorithm. The yellow circles are the obstacles. The travel time for Fig. 7.5a was 32.7 sec. The travel time for Fig. 7.5b was 34.525 sec.

Figure 7.5b demonstrates the advantage of implementing a motion planning algorithm to synthesis a continuous path. The planning environment and discrete solution are the same as Fig. 7.5a, but with obstacles blocking the established discrete solution. Despite the obstruction along the established path, the RRT algorithm was able to synthesis a path that avoids collision while keeping as close to the discrete path solution as possible. The travel time was 34.525 sec, which would be improved by tuning the number of sample points and implementing a better motion planning algorithm.

7.4 Summary

This chapter implemented the algorithms featured in this thesis on real-world oceanic data. The trim-based planning framework featured in Chapter 4 found an optimal path across a static EAC. The heuristic planner with TDSP framework featured in Chapter 6 found a travel time-optimal path for a marine vehicle across the time-varying EAC. Both methods have demonstrated relatively fast planning in a large real-world oceanic environment, making on-site planning and re-planning strategies viable. This chapter also demonstrated synthesising the RRT path across the TDSP framework to demonstrate the modularity of the hierarchical planner. Synthesising a continuous path from the RRT algorithm allowed more intricate navigation around a cluttered environment. In the next chapter, we conclude the thesis and propose possible future work.

Chapter 8

Conclusion and future work

This chapter concludes the thesis. Section 8.1 summaries the thesis in detail. Section 8.2 highlights the contributions made. Section 8.3 presents a list of pursuable potential future work.

8.1 Thesis summary

This thesis has identified and solved three problems in autonomous motion planning of underwater gliders in an energy-optimal or time-optimal manner. Chapter 4 has addressed the first two problems: optimal path planning with a 12D dynamic glider model and optimal navigation in a static flow field (known as Zermelo's problem). The complex dynamic model was addressed by using trim-state to reduce the dynamic model into a more manageable 6D kinematic model without losing the glider dynamic behaviour. This representation allowed more accurate representation of the glider motion and energy cost model without resorting to planning in higher dimensions. Zermelo's problem was addressed with *Trim-based FMT**, an extension of the state-of-the-art motion planning algorithm FMT* that plans using the trim-state of an underwater glider. The algorithm was formally analysised to demonstrate it retains the properties of FMT*, namely it guarantees asymptotically optimal underwater glider path planning that minimises the glider energy cost.

The last problem was to find a time-optimal path through a time-varying flow field. This thesis addressed this problem with a hierarchical planning approach, first addressing it as a continuous-time non-FIFO TDSP problem, then synthesising a continuous path across the discrete solution. Chapter 5 formulated the TDSP problem with piecewise-constant functions (PF) to approximate the continuous-time in discrete time, and addressed the non-FIFO case by solving for an optimal PF-based policy to promote cyclic motions. Analysing this framework showed that the optimal solution was solvable in polynomial time, a first for this problem variation. An example of this framework solving a travel time-optimal path across a time-varying flow field was demonstrated by building a time-dependent directed graph using the PRM* algorithm.

Chapter 6 actualised the hierarchical planning approach by addressing the synthesis of a continuous path in time-varying flow fields across the discrete TDSP solution. The chapter presented an alternate method of building the time-dependent directed graph with *state line* associated with each state. The hierarchical planner synthesised a continuous path by growing and pruning a tree of trajectories at each sequence of the discrete state solutions, based on the vehicle dynamic and flow fields. A path is branched from each end of the tree to the next state line by forward integrating across a finite set of heading values. After branching, the trajectories are pruned to keep N-best paths with the closest discrete travel time to the discrete solution by that state. The analysis showed that the hierarchical planner retains the optimality of the discrete solutions and remains solvable in polynomial time.

This thesis then applied the demonstrated algorithms with the EAC oceanic database to demonstrate its applicability in real-world environments. Both the trim-based and the TDSP hierarchical framework showed relatively fast planning compared to the resulting travel time of the mission, making them suitable for both on-site planning and re-planning. The applicability of the TDSP hierarchical framework was further demonstrated by using a known motion planning algorithm to synthesis a continuous path. The resulting path allowed successful navigation in time-varying flow fields with cluttered environments.

8.2 Contributions

This section highlights the contributions made in this thesis. These points are readdressed from the contribution section in Chapter 1.

8.2.1 Energy-optimal path in static flow field with complex underwater glider dynamics

It has been established that Zermelo's problem was an open problem and that the underwater glider operated in a complex dynamic model. Chapter 4 demonstrated an energy-optimal path planning for underwater gliders in static flow fields using a trim-based model. This model was implemented with a state-of-the-art sampling-based motion planning algorithm that guarantees an AO path solution that also retained the properties of the glider dynamics, using the sawtooth depth profile to propel itself forward. Past work commonly evaluated the glider path as a 2D kinematic model with directly controllable velocity [36–38], which neglects the ballast component of the glider operation. This work introduces the first instance of an AO motion planning algorithm that fully captures the glider dynamics. We have demonstrated in recent work how the trim-based model can easily be incorporated into other planning frameworks [32]. This flexibility opens up a new avenue of study, including trim-based planning of a fleet of underwater gliders, and develop a stochastic variant of the glider motion planning algorithm. The stochastic studies have already shown promise in our recent work [121].

The applications of trim-based planning are numerous. An obvious application is to the deployment of underwater gliders to perform marine-based missions. The idea of trim-states can also be extended to other platforms operating in a flow field environment that favour long edge transitions, such as aircraft, thermal gliders, and satellites; the latter option considers using trim-state to emulate gravitational slingshots.

The results demonstrated by the trim-based planner also motivate changing the way gliders are currently operated. Current methods heuristically choose the sawtooth profile when deployed. This contribution makes a strong case about allowing for more control options and the use of irregular sawtooth profiles during operations.

8.2.2 Travel time-optimal path in a non-FIFO TDSP framework

It has been established that solving for time-optimal paths in a non-FIFO class time-dependent directed graph with continuous-time is hard. Chapter 5 introduces a special case for a non-FIFO TDSP framework, where the continuous edge time function is approximated with a piecewise-constant function to address this problem. The chapter demonstrates a theoretical result that, for this special case, a travel time-optimal policy can be evaluated in polynomial time. This result is the first to demonstrate this property for this special case.

The application of this theoretical work is vast, solving many time-dependent problems that can be modelled as a non-FIFO case such as special traffic and network cases. The application that we are most interested in is travel time-optimal navigation across a dynamic flow field. Section 5.4.1 established that dynamic flow fields exhibit non-FIFO properties, and can easily generate a time-dependent directed graph across the flow field environment using PRM*. Preliminary results were presented in Section 5.5.2, which opens up studies for energy-optimal paths derived from the TDSP framework. Other possible work includes stochastic variants that addresses the uncertainty in both the flow field environment and vehicle dynamics. The operational applications are considered in the next contribution.

8.2.3 Travel time-optimal path through dynamic flow fields with complex glider dynamics

Chapter 6 demonstrates a hierarchical motion planning algorithm that synthesis a continuous vehicle path across the discrete solution demonstrated in Chapter 5. Previous work approached the optimal navigation problem through dynamic flow fields using a spatial-temporal graph, making the solution resolution complete for time. Our contribution demonstrates dynamic flow field navigation using the optimal schedule defined by the special non-FIFO TDSP framework, and therefore does not have this issue. Our approach also retains both the time complexity and optimality results from the non-FIFO TDSP framework, which is better than the current method of evaluating continuous paths across

dynamic flow fields. Such complexity results encourage extension of the planner for multi-robot planning. The optimal result was possible under the assumption that the flow field data (and by extension, the forecast model) is reliable. In practice, this is not the case and opens up new studies for stochastic variances and better ocean forecast method.

The application of this result is similar to our second contribution. From a glider perspective, this further motivates moving away from the heuristic sawtooth motion, as the resulting glider path once again demonstrated irregular sawtooth motion as the time-optimal path. The results from Chapter 7 motivate field trials for a large real-world ocean environment, like the East Australian Current.

8.3 Future work

This section lists essential avenues for future work that would improve the applicability of autonomous underwater glider operations in real-world oceanic environments. The future work can be categorised in three directions: hardware validation, mission implementation, and algorithmic improvements.

8.3.1 Hardware validation

In Section 7.1 and Section 7.2, we demonstrated the algorithms across the EAC dataset. We now aim to test these algorithms on real underwater gliders under real oceanic conditions.

This future work is envisioned in three stages. First, as a point of comparison, we would like to operate the underwater glider in the ocean using traditional methods. This stage also gives us an opportunity to familiarise ourselves with real world glider operations, including control interface and deployment practices. Once we are comfortable with the baseline operation, we would like to validate our hierarchical planner presented in Chapter 6 by planning with the forecast data and comparing the results with our baseline solution. This work may require custom APIs so that the simulated optimal control sequences can be reflected by the glider platform. The final stage is to extend our hardware validation by

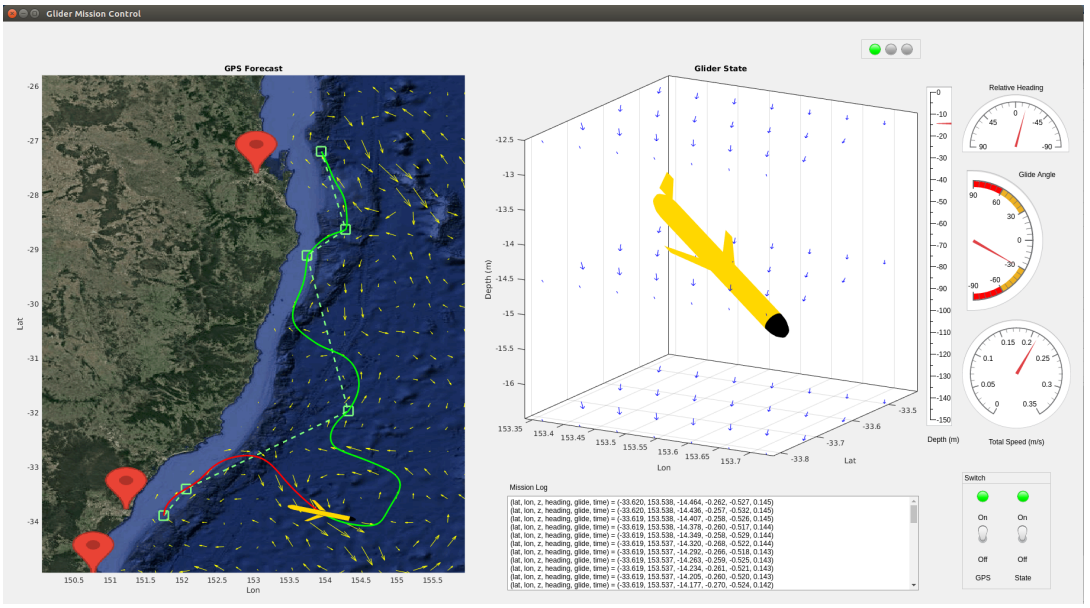


FIGURE 8.1: A mock example GUI for glider mission control

implementing a planner across a fleet of underwater gliders. The long-term goal is to achieve multi-robot actuations with a fleet of underwater gliders.

8.3.2 Mission implementation

The primary motivation of this thesis is to allow autonomous underwater gliders to operation with minimum human interaction and oversight. After hardware validation of our algorithms has been satisfied, the next step is to have the glider perform naval missions using our algorithm. The aim is to see how the mission performance improves with our algorithm regarding mission longevity compared to the traditional method of glider control. Once we have solved the stochastic variant of the problem, we also want to test the reliability of the glider completing its missions to verify a reduced need for human oversight.

We also plan to develop a *mission control GUI* for execution and observation of glider missions. This GUI would visualise the projected mission solution and the actual glider performance since the start of the mission. A mock example is shown in Fig. 8.1.

8.3.3 Algorithmic improvements

This thesis has demonstrated travel time-optimal motion planning for underwater gliders in a deterministic dynamic flow field environment. While working on the thesis, we have noticed several places where the algorithm can be improved upon, either to improve computation time or make it more applicable for real-world environment.

Efficient edge evaluation

In Chapters 4-6, all edge evaluation was done with a form of forward integration, like the shooting method. These iterative methods are computationally expensive and act as bottlenecks in the planning process. Faster computation allows for more convenient on-site planning and re-planning, and can also promote multi-robot implementations.

Recent work in [41, 122] demonstrate the use of the streamline function for a drastic reduction in computation time for edge evaluation over static flow fields. We have recently looked into incorporating this method into our trim-based framework [32]. The next step is to formulate the streamline evaluation for time-varying flow fields to speed up the algorithms demonstrated in Chapters 5-6.

Multi-robot algorithm

Naval missions, such as coastal patrol, environmental surveying, and resource tracking, benefit greatly from a coordinated effort by a fleet of gliders. Chapter 6 demonstrated that the optimal path was able to be evaluated in polynomial time. This result encourages extending the algorithm to handle multi-robot cases.

Planning with uncertainty in control and flow fields

In Section 7.1 and Section 7.2, the energy-optimal and travel time-optimal paths were planned in a deterministic EAC environment with a deterministic underwater glider model defined in Section 3.1.1. In reality, the provided ocean current forecast, glider localisation,

and glider controls are plagued with uncertainties and noise. These uncertainties leads to stochastic travel time between states. Blindly following the deterministic planner while not considering the stochastic component may cause the marine vehicle to veer out of the intended path and become lost, or enter a restricted zone where extraction is impossible.

Recent work on planning for underwater gliders has considered uncertainties in static flow field predictions [118] and control [121]. The next step is to consider the uncertainties of the glider dynamics and the dynamic flow field.

Energy-optimal policy for non-FIFO TDSP problem

Chapters 5-6 addressed the travel-time optimal path of marine vehicles across a dynamic flow field environment. We now wish to evaluate an energy-optimal path. There are several methods we can use to address this problem. The time-dependent directed graph could be redefined to accommodate the general objective function while retaining the TDSP property. Another option is to use a hierarchical approach in finding the energy-optimal path by considering the vehicle cost function as a composition between time and control components. The graphs and policy can be kept as is to find the time-energy optimal path and then synthesis the continuous path that minimises the control-energy cost.

Extension of the hierarchical planner

Section 7.3 demonstrated the use of RRT motion planning algorithm to synthesis a continuous path across the discrete TDSP solution in a dynamic flow field environment. We observed that the paths were jagged and arrived late to schedule due to the quality of the RRT algorithm. We wish to expand on this work by trying different state-of-the-art motion planning algorithms to synthesis the continuous path, such as RRT* and FMT*.

Appendix A

Underwater glider dynamics

A.1 Glider parameters

Parameter	Description	Value
g	Gravity	9.81 m/s^2
J_2	Total moment of inertia about the Y-axis	0.1 Nm^2
m	Mass of displaced fluid	11.22 kg
$\bar{m}$	Movable mass	2.0 kg
$m_{b\,\max}$	Variable ballast mass	2.0 kg
m_h	Uniformly distributed hull mass	8.22 kg
m_{f1}	Added mass matrix, X diagonal component	2.0 kg
m_{f3}	Added mass matrix, Z diagonal component	14.0 kg
z_p	Position of moving mass in Z-axis	0.04 m
K_{D_0}	Drag coefficient for zero angle of attack	5 $N(s/m)^2$
K_D	Drag coefficient	20 $N(s/m)^2$
K_{L_0}	Life coefficient for zero angle of attack	0 $N(s/m)^2$
K_L	Lift coefficient	306 $N(s/m)^2$
K_{M_0}	Pitching moment coefficient for zero angle of attack	0 $Nm(s/m)^2$
K_M	Pitching moment coefficient	-36.5 $Nm(s/m)^2$
L_H	Hotel coefficient	1 (J/s)

TABLE A.1: Definition of glider parameters and their values.

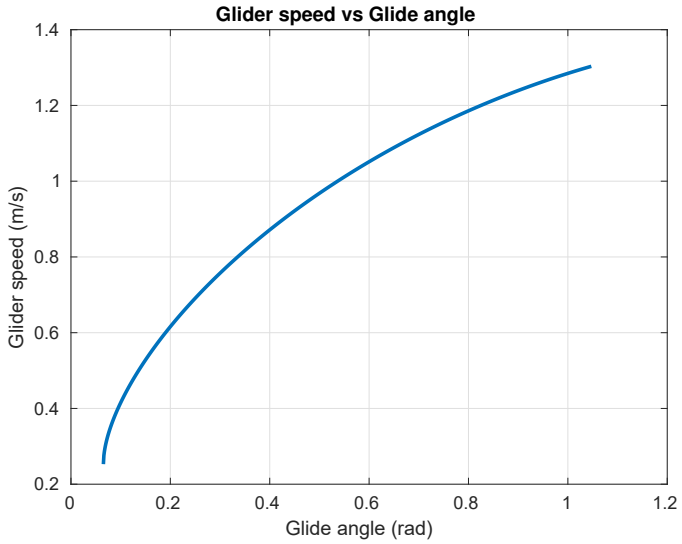


FIGURE A.1: Relation between the nominal glider speed and its glide angle. The values are evaluated from the glider parameters defined in A.1. For the chosen parameters, the glider velocity is symmetrical between the full and empty ballast state.

A.2 Trim state evaluation

This section lists the equations used to derive the trim states for reference purposes. For the sake of completeness, all relevant equations will be listed. The full derivation of these equations can be referred to in [21].

For transition between the positions $\mathbf{p}_k = [x_k, y_k, z_k]^T$ and $\mathbf{p}_{k+1} = [x_{k+1}, y_{k+1}, z_{k+1}]^T$, the desired trim state $\tau_k = [V_k, \gamma_k, \phi_k, m_{bk}]^T$ is described as follows:

$$\begin{aligned} m_{bk} &= \begin{cases} 0, & \text{if } z_k < z_{k-1} \\ m_{b\,\max}, & \text{otherwise.} \end{cases} \\ V_k(\gamma_k, m_{bk}) &= \sqrt{\frac{\left(m_{bk} - (m - m_h - \bar{m})\right) \cdot g}{-D(\gamma_k)\sin\gamma_k + L(\gamma_k)\cos\gamma_k}}, \end{aligned} \tag{A.1}$$

where

$$\begin{aligned} L(\gamma_k) &= K_{L_0} + K_L \alpha(\gamma_k) \\ D(\gamma_k) &= K_{D_0} + K_D \alpha(\gamma_k)^2 \\ \alpha(\gamma_k) &= \frac{K_L}{2K_D} \tan\gamma_k \left(-1 + \sqrt{1 - \frac{4K_D}{{K_L}^2} \cot\gamma_k (K_{D_0} \cot\gamma_k + K_{L_0})} \right), \end{aligned}$$

and the glide angle γ_k and heading angle ϕ_k are evaluated using the shooting method.

A.3 Glide angle control with moving mass

In practice, the underwater glider's glide angle is controlled by shifting the moving mass to the desired position $\mathbf{r}_p = [x_p, y_p, z_p]^T$. The energy required to change the glide angle is proportional to the displacement of the moving mass $\Delta\mathbf{r}_p$. The moving mass only needs to be shifted in the x-axis to control the glide angle (i.e., $\Delta\mathbf{r}_p = \Delta x_p$). For a glide angle γ_k, the equivalent moving mass position is [21]:

$$x_p = -z_p \tan\theta + \frac{\left((m_{f3} - m_{f1})\, u_d w_d + (K_{M_0} + K_M \alpha(\gamma_k))\, V_k(\gamma_k, m_{bk})^2\right)}{\bar{m} g \cos\theta}, \tag{A.2}$$

where,

$$\begin{aligned} \theta(\gamma_k) &= \alpha(\gamma_k) + \gamma_k \\ u_d(\gamma_k) &= V_k(\gamma_k, m_{bk}) \cos\alpha(\gamma_k) \\ w_d(\gamma_k) &= V_k(\gamma_k, m_{bk}) \sin\alpha(\gamma_k). \end{aligned}$$

Bibliography

[1] Chanyeol Yoo, Robert Fitch, and Salah Sukkarieh. Online task planning and control for fuel-constrained aerial robots in wind fields. *Int. J. Robot. Res.*, 35(5):438–453, 2016.

[2] Brunilde Girardet, Laurent Lapasset, Daniel Delahaye, and Christophe Rabut. Wind-optimal path planning: Application to aircraft trajectories. In *Proc. of ICARCV*, pages 1403–1408, 2014.

[3] Syama M., Dineshkumar M., and Arun Kishore W. C. Trajectory optimization of an autonomous dynamic soaring UAV. In *Proc. of ICCC*, pages 95–100, 2015.

[4] Chanyeol Yoo, Robert Fitch, and Salah Sukkarieh. Probabilistic Temporal Logic for Motion Planning with Resource Threshold Constraints. In *Proc. of RSS*, 2012.

[5] J. A. Cobano, Alejo D., S. Sukkarieh, G. Heredia, and A. Ollero. Thermal detection and generation of collision-free trajectories for cooperative soaring UAVs. In *Proc. of IEEE/RSJ IROS*, pages 2948–2954, 2013.

[6] Joseph Nguyen, Nicholas Lawrance, Robert Fitch, and Salah Sukkarieh. Energy-constrained motion planning for information gathering with autonomous aerial soaring. In *Proc. of IEEE ICRA*, pages 3825–3831, 2013.

[7] Roger Hine, Scott Willcox, Graham Hine, and Tim Richardson. The wave glider: A wave-powered autonomous marine vehicle. In *Proc. of MTS/IEEE OCEANS*, pages 1–6, 2009.

[8] Douglas C Webb, Paul J Simonetti, and Clayton P Jones. SLOCUM: An underwater glider propelled by environmental energy. *IEEE J. Ocean. Eng.*, 26(4):447–452, 2001.

[9] Ralf Bachmayer, N Ehrich Leonard, J Graver, E Fiorelli, Pradeep Bhatta, and D Paley. Underwater gliders: Recent developments and future applications. In *Proc. of Underwater Technology*, pages 195–200, 2004.

[10] Ki Myung Brian Lee, James Ju Heon Lee, Chanyeol Yoo, Ben Hollings, , and Robert Fitch. Active perception for plume source localisation with underwater gliders. In *Proc. of ARAA ACRA*, 2018.

[11] Roger P Stokey, Alexander Roup, Chris von Alt, Ben Allen, Ned Forrester, Tom Austin, Rob Goldsborough, Mike Purcell, Fred Jaffre, Greg Packard, and A Kukulya. Development of the REMUS 600 autonomous underwater vehicle. In *Proc. of MTS/IEEE OCEANS*, pages 1301–1304, 2005.

[12] B Claus, R Bachmayer, and C D Williams. Development of an auxiliary propulsion module for an autonomous underwater glider. *Proc. Inst. Mech. Eng. M*, 224(4): 255–266, 2010.

[13] Daniel L Rudnick, Russ E Davis, Charles C Eriksen, David M Fratantoni, and Mary Jane Perry. Underwater gliders for ocean research. *Mar. Technol. Soc. J.*, 38 (2):73–84, 2004.

[14] Louise M. Russell-Cargill, Bradley S. Craddock, Ross B. Dinsdale, Jacqueline G. Doran, Ben N. Hunt, and Ben Hollings. Using autonomous underwater gliders for geochemical exploration surveys. *The APPEA Journal*, 58:367–380, 2018.

[15] Hordur Johannsson, Michael Kaess, Brendan Englot, Franz Hover, and John Leonard. Imaging sonar-aided navigation for autonomous underwater harbor surveillance. In *Proc. of IEEE/RSJ IROS*, pages 4396–4403, 2010.

[16] Naomi E. Leonard, Derek A. Paley, Russ E. Davis, David M. Fratantoni, Francois Lekien, Fumin Zhang, Terry Huntsberger, Michael Keegan, and Robert Brizzolara. Coordinated control of an underwater glider fleet in an adaptive ocean sampling field experiment in monterey bay. *JFR*, 27(6):718–740, 2010.

[17] Josep Isern-González, Daniel Hernández-Sosa, Enrique Fernández-Perdomo, Jorge Cabrera-Gámez, Antonio C. Domínguez-Brito, and Victor Prieto-Marañón. Path

planning for underwater gliders using iterative optimization. In *Proc. of IEEE ICRA*, pages 1538–1543, 2011.

[18] Enrique Fernndez-Perdomo, Daniel Hernndez-Sosa, Josep Isern-Gonzlez, Jorge Cabrera-Gmez, Antonio C. Domnguez-Brito, and Vctor Prieto-Maran. Single and multiple glider path planning using an optimization-based approach. In *Proc. of IEEE OCEANS*, 2011.

[19] Ernst Zermelo. ber das navigationsproblem bei ruhender oder vernderlicher windverteilung. *ZAMM-J. App. Math. and Mech./Zeitschrift für Angewandte Mathematik und Mechanik*, 11(2):114–124, 1931.

[20] Henry Stommel. The Slocum mission. *Oceanography*, 2(1):22–25, 1989.

[21] Naomi Leonard and Joshua Graver. Model-based feedback control of autonomous underwater gliders. *IEEE J. Oceanic Eng.*, 24(4):633–645, 2001.

[22] Clayton Jones, Ben Allsup, and Christopher DeCollibus. Slocum glider: Expanding our understanding of the oceans. In *Proc. of IEEE OCEANS*, pages 1–10, 2014.

[23] Dushyant Rao and Stefan B. Williams. Large-scale path planning for Underwater Gliders in ocean current. In *Proc. of ARAA ACRA*, 2009.

[24] Jonas Witt and Matthew Dunbabin. Go with the flow: Optimal AUV path planning in coastal environments. In *Proc. of ARAA ACRA*, 2014.

[25] Dhanushka Kularatne, Subhrajit Bhattacharya, and M. Ani Hsieh. Time and energy optimal path planning in general flows. In *Proc. of RSS*, 2016.

[26] Nina Mahmoudian, Jesse Geisbert, and Craig Woolsey. Approximate analytical turning conditions for underwater gliders: Implications for motion control and path planning. *IEEE J. Ocean. Eng.*, 35(1):131–142, 2010.

[27] Brian C. Dean. Shortest paths in FIFO time-dependent networks: Theory and algorithms. Technical report, MIT, 2004.

[28] Daniel Delling and Dorothea Wagner. Time-dependent route planning. In Ravindra K. Ahuja, Rolf H. Möhring, and Christos D. Zaroliagis, editors, *Robust and*

Online Large-Scale Optimization, volume 5868, pages 207–230. Springer, 2009. Lecture Notes in Computer Science.

[29] Ariel Orda and Raphael Rom. Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length. *J. ACM*, 37(3):607–625, 1990.

[30] Ki Myung Brian Lee, Chanyeol Yoo, Ben Hollings, Stuart Anstee, Shoudong Huang, and Robert Fitch. Online estimation of ocean current from sparse GPS data for underwater vehicles. *Proc. of IEEE ICRA*, 2019.

[31] James Ju Heon Lee, Chanyeol Yoo, Raewyn Hall, Stuart Anstee, and Robert Fitch. Energy-optimal kinodynamic planning for underwater gliders in flow fields. In *Proc. of ARAA ACRA*, 2017.

[32] Kwun Yiu Cadmus To, James Ju Heon Lee, Chanyeol Yoo, Stuart Anstee, and Robert Fitch. Streamline-based control of underwater gliders in 3d environments. In *Proc. of IEEE CDC*, pages 8303–8310, 2019.

[33] James Ju Heon Lee, Chanyeol Yoo, Stuart Anstee, and Robert Fitch. Efficient Optimal Planning in non-FIFO Time-Dependent Flow Fields. *arXiv e-prints*, art. arXiv:1909.02198, Sep 2019.

[34] James Ju Heon Lee, Chanyeol Yoo, Stuart Anstee, and Robert Fitch. Hierarchical planning in time-dependent flow fields for marine robots. In *Proc. of IEEE ICRA*, 2020.

[35] Junliang Cao, Junjun Cao, and Zheng Zeng. Toward optimal rendezvous of multiple underwater gliders: 3D path planning with combined sawtooth and spiral motion. *J. Intell. Robot. Syst.*, 85(1):189–206, 2017.

[36] Tamer Inanc, Shawn C. Shadden, and Jerrold E. Marsden. Optimal trajectory generation in ocean flows. In *Proc. of ACC*, 2005.

[37] Enrique Fernández-Perdomo, Jorge Cabrera-Gámez, Daniel Hernández-Sosa, Josep Isern-González, Antonio C. Domínguez-Brito, Alex Redondo, Josep Coca, Antonio G. Ramos, Enrique Álvarez Fanjul, and Marcos García. Path planning for gliders using Regional Ocean Models: Application of Pinzón path planner with the ESEOAT

model and the RU27 trans-Atlantic flight data. In *Proc. of IEEE OCEANS*, pages 1–10, 2010.

[38] Chien-Chou Shih, Mong-Fong Horng, Tien-Szu Pan, Jeng-Shyang Pan, and Chun-Yu Chen. A genetic-based effective approach to path-planning of autonomous underwater glider with upstream-current avoidance in variable oceans. *Soft Comput.*, 21 (18):5369–5386, 2017.

[39] Zhu Xinke, Jin Xianglong, Yu Jiancheng, and Li Yiping. Path planning in stronger ocean current for underwater glider. In *Proc. of IEEE CYBER*, pages 891–895, 2015.

[40] Dhanushka Kularatne, Subhrajit Bhattacharya, and M. Ani Hsieh. Optimal Path Planning in Time-Varying Flows with Forecasting Uncertainties. In *Proc. of IEEE ICRA*, 2018.

[41] Kwun Yiu Cadmus To, Ki Myung Brian Lee Lee, Chanyeol Yoo, Stuart Anstee, and Robert Fitch. Streamlines for motion planning in underwater currents. *Proc. of IEEE ICRA*, 2019.

[42] Inyoung Ko, Beobkyoon Kim, and Frank Chongwoo Park. Randomized path planning on vector fields. *Int. J. Robot. Res.*, 33(13):1664–1682, 2014.

[43] Bin Li, Chao Xu, Kok Lay Teo, and Jian Chu. Time optimal zermelos navigation problem with moving and fixed obstacles. *Appl. Math. Comput.*, 224:866–875, 2013.

[44] Lantao Liu and Gaurav S. Sukhatme. A Solution to Time-Varying Markov Decision Processes. In *Proc. of IEEE ICRA*, 2018.

[45] Nina Mahmoudian and Craig Woolsey. Underwater glider motion control. In *Proc. of IEEE CDC*, pages 552–557, 2008.

[46] Blane Rhoads, Igor Mezić, and Andrew Poje. Minimum time feedback control of autonomous underwater vehicles. In *Proc. of IEEE CDC*, pages 5828–5834, 2010.

[47] Laszlo Techy. Optimal navigation in a planar time-varying point-symmetric flowfield. In *Proc. of IEEE CDC/ECC*, pages 7325–7330, 2011.

[48] Blane Rhoads, Igor Mezíc, and Andrew Poje. Efficient guidance in finite time flow fields. In *Proc. of IEEE CDC*, pages 6182–6189, 2013.

[49] Lorenz Pyta, Michael Herty, and Dirk Abel. Optimal feedback control of the incompressible Navier-Stokes-equations using reduced order models. In *Proc. of IEEE CDC*, pages 2519–2524, 2015.

[50] Francis D. Lagor, Kayo Ide, and Derek A. Paley. Touring invariant-set boundaries of a two-vortex system using streamline control. In *Proc. of IEEE CDC*, pages 2217–2222, 2015.

[51] Tapovan Lolla, Mattheus P. Ueckermann, K. Yigit, Patrick J. Haley, and Pierre F. J. Lermusiaux. Path planning in time dependent flow fields using level set methods. In *Proc. of IEEE ICRA*, pages 166–173, 2012.

[52] Tapovan Lolla, Pierre F J Lermusiaux, Mattheus P Ueckermann, and Patrick J Haley. Time-optimal path planning in dynamic flows using level set equations: Theory and schemes. *Ocean Dynam.*, pages 1373–1397, 2014.

[53] Tapovan Lolla, Patrick J. Haley Jr., and Pierre F. J. Lermusiaux. Time-optimal path planning in dynamic flows using level set equations: realistic applications. *Ocean Dynam.*, 64(10):1399–1417, 2014.

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

[55] Aleš Zamuda and José Daniel Hernández Sosa. Differential evolution and underwater glider path planning applied to the short-term opportunistic sampling of dynamic mesoscale ocean structures. *Appl. Soft Comput.*, 24:95–108, 2014.

[56] Matthew R Jardin and Arthur E Bryson. Neighboring optimal aircraft guidance in winds. *J. Guid. Control Dyn.*, 24(4):710–715, 2001.

[57] Efstathios Bakolas and Panagiotis Tsiotras. The zermelo–voronoi diagram: A dynamic partition problem. *Automatica*, 46(12):2059–2067, 2010.

[58] E. Bakolas and P. Tsiotras. Minimum-time paths for a small aircraft in the presence of regionally-varying strong winds. *AIAA Infotech Aerospace, Atlanta, GA*, pages 2010–3380, 2010.

[59] Sang Gyun Park. *Optimal Control based Method for Design and Analysis of Continuous Descent Arrivals*. PhD thesis, Georgia Institute of Technology, 2014.

[60] Sang Gyun Park and John-Paul Clarke. Vertical trajectory optimization for continuous descent arrival procedure. In *Proc. of AIAA GNCC*, 2012.

[61] Graeme C. Hays, Asbjorn Christensen, Sabrina Fossette, Gail Schofield, Julian Talbot, and Patrizio Mariani. Route optimisation and solving zermelos navigation problem during long distance migration in cross flows. *Ecology Letters*, 17(2):137–143, 2014.

[62] Rutherford Aris. *Vectors, Tensors and the Basic Equations of Fluid Mechanics*. Dover Publications, New York, 1989.

[63] Geoffrey Ingram Taylor and Albert E. Green. Mechanism of the Production of Small Eddies from Large Ones. In *Proc. of Royal Society of London. Series A, Mathematical and Physical Sciences*, pages 499–521, 1937. Available at: `http://www.ams.jhu.edu/~eyink/Turbulence/classics/TaylorGreen37.pdf`.

[64] Pierre Bonami, Alberto Olivares, Manuel Soler, and Ernesto Staffetti. Multiphase mixed-integer optimal control approach to aircraft trajectory optimization. *J. Guid. Control Dyn.*, 36(5):1267–1277, 2013.

[65] Jerome Le Ny and George J Pappas. Joint metering and conflict resolution in air traffic control. *J. Guid. Control Dyn.*, 34(5):1507–1518, 2011.

[66] Luca Corolli, Guglielmo Lulli, Lewis Ntaimo, and Saravanan Venkatachalam. A two-stage stochastic integer programming model for air traffic flow management. *IMA J. Manage. Math.*, pages 19–40, 2017.

[67] Dimitris Bertsimas, Guglielmo Lulli, and Amedeo Odoni. An integer optimization approach to large-scale air traffic flow management. *Oper. Res.*, 59(1):211–227, 2011.

[68] Douglas C. Webb, Paul J. Simonetti, and Clayton P. Jones. SLOCUM: An underwater glider propelled by environmental energy. *IEEE J. Oceanic Eng.*, 26(4):447–452, 2001.

[69] Jeff Sherman, Russ E. Davis, W. B. Owens, and J. Valdes. The autonomous underwater glider "Spray". *IEEE J. Oceanic Eng.*, 26(4):437–446, 2001.

[70] Charles C. Eriksen, T. James Osse, Russell D. Light, Timothy Wen, Thomas W. Lehman, Peter L. Sabin, John W. Ballard, and Andrew M. Chiodi. Seaglider: A long-range autonomous underwater vehicle for oceanographic research. *IEEE J. Oceanic Eng.*, 26(4):424–436, 2001.

[71] Ryan N. Smith, Yi Chao, Peggy P. Li, David A. Caron, Burton H. Jones, and Gaurav S. Sukhatme. Planning and implementing trajectories for autonomous underwater vehicles to track evolving ocean processes based on predictions from a regional ocean model. *Int. J. Robot. Res.*, 29(12):1475–1497, 2010.

[72] Ryan N. Smith and Matthew Dunbabin. Controlled drift: An investigation into the controllability of underwater vehicles with minimal actuation. In *Proc. of ARAA ACRA*, 2011.

[73] Luca Foschini, John Hershberger, and Subhash Suri. On the complexity of time-dependent shortest paths. *Algorithmica*, 68(4):1075–1097, 2014.

[74] Stuart E. Dreyfus. An Appraisal of Some Shortest-Path Algorithms. *Oper. Res.*, 17 (3):395–412, 1969.

[75] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to algorithms*. The MIT press, 2001.

[76] Stuart J. Russell and Peter Norvig. *Artifical Intelligence: A Modern Approach*. Prentice Hall, 3rd edition, 2009.

[77] Spyros Kontogiannis and Christos Zaroliagis. Distance Oracles for Time-Dependent Networks. *Algorithmica*, 74(4):1404–1434, 2016.

[78] Yunchuan Sun, Xinpei Yu, Rongfang Bie, and Houbing Song. Discovering time-dependent shortest path on traffic graph for drivers towards green driving. *J. Netw. Comput. Appl.*, 83:204–212, 2017.

[79] Ugur Demiryurek, Farnoush Banaei-Kashani, Cyrus Shahabi, and Anand Ranganathan. Online computation of fastest path in time-dependent spatial networks. In *Proc. of SSTD*, pages 92–111, 2011.

[80] Venkata M V Gunturi, Ernesto Nunes, KwangSoo Yang, and Shashi Shekhar. A critical-time-point approach to all-start-time Lagrangian shortest paths: A summary of results. In *Proc. of SSTD*, pages 74–91, 2011.

[81] Venkata M.V. Gunturi, Shashi Shekhar, and Kwangsoo Yang. A Critical-Time-Point Approach to All-Departure-Time Lagrangian Shortest Paths. *IEEE Trans. Knowl. Data Eng.*, 27(10):2591–2603, 2015.

[82] Gernot Veit Batz and Peter Sanders. Time-dependent route planning with generalized objective functions. In *Proc. of Algorithms*, volume 7501, pages 169–180, 2012.

[83] Betsy George and Shashi Shekhar. Time-aggregated graphs for modeling spatio-temporal networks. *J. Semantics Data*, 9:191–212, 2008.

[84] John F. Canny. *The complexity of robot motion planning*. The MIT press, 1988.

[85] Lydia E. Kavraki, Petr Svestka, Jean-Claude Latombe, and Mark H. Overmars. Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces. *IEEE Trans. Robot. Autom.*, 12(4):566–580, 1996.

[86] Robert Bohlin and Lydia E. Kavraki. Path Planning Using Lazy PRM. In *Proc. of IEEE ICRA*, pages 521–528, 2000.

[87] Léonard Jaillet and Thierry Siméon. A PRM-based Motion Planner for Dynamically Changing Environment. In *Proc. of IEEE/RSJ IROS*, pages 1606–1611, 2004.

[88] Ron Alterovitz, Sachin Patil, and Anna Derbakova. Rapidly-exploring roadmaps: Weighing exploration vs. refinement in optimal motion planning. In *Proc. of IEEE ICRA*, pages 3706–3712, 2011.

[89] Steven M. LaValle and James J. Kuffner. Randomized kinodynamic planning. *Int. J. Robot. Res.*, 20(5):378–400, 2001.

[90] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[91] Wei Wang, Xin Xu, Yan Li, Jinze Song, and Hangen He. Triple RRTs: An Effective Method for Path Planning in Narrow Passages. *Advanced Robotics*, 24(7):943–962, 2010.

[92] James J. Kuffner and Steven M LaValle. RRT-connect: An efficient approach to single-query path planning. In *Proc. of IEEE ICRA*, pages 995–1001, 2000.

[93] Emilio Frazzoli and Sertac Karaman. Sampling-based algorithms for optimal motion planning. *Int. J. Robot. Res.*, 30(7):846–894, 2011.

[94] Oktay Arslan and Panagiotis Tsiotras. Use of Relaxation Methods in Sampling-based Algorithms for Optimal Motion Planning. In *Proc. of IEEE ICRA*, pages 2421–2428, 2013.

[95] Otte Michael and Emilio Frazzoli. RRT^x: Asymptotically optimal single-query sampling-based motion planning with quick replanning. *Int. J. Robot. Res.*, 35(7): 797–822, 2015.

[96] Lucas Janson, Edward Schmerling, Ashley Clark, and Marco Pavone. Fast Marching Tree: A fast marching sampling-based method for optimal motion planning in many dimensions. *Int. J. Robot. Res.*, 34(7):883–921, 2015.

[97] Benjamin Cohen, Mike Phillips, and Maxim Likhachev. Planning Single–Arm Manipulations with N–Arm Robots. In *Proc. of RSS*, pages 208–216, 2014.

[98] Ioan A. Sucan and Lydia E. Kavraki. A sampling-based tree planner for systems with complex dynamics. *IEEE Trans. Robot.*, 28(1):116–131, 2012.

[99] Erion Plaku, Lydia E. Kavraki, and Moshe Y. Vardi. Discrete Search Leading Continuous Exploration for Kinodynamic Motion Planning. In *Proc. of RSS*, pages 326–333, 2007.

[100] Mike Phillips, Maxim Likhachev, and Sven Koenig. PA*SE: Parallel A* for Slow Expansions. In *Proc. of ICAPS*, pages 208–216, 2014.

[101] Neal Seegmiller, Jason Gassaway, Elliot Johnson, and Jerry Towler. The maverick planner: An efficient hierarchical planner for autonomous vehicles in unstructured environments. In *Proc. of IEEE/RSJ IROS*, pages 2018–2023, 2017.

[102] Ricardo Samaniego, Rodrigo Rodríguez, Fernando Vázquez, and Joaquín López. Efficient path planing for articulated vehicles in cluttered environments. *Sensors*, 20 (23):6821, 2020.

[103] Michael Brunner, Bernd Brüggemann, and Dirk Schulz. Hierarchical rough terrain motion planning using an optimal sampling-based method. In *Proc. of ICRA*, pages 5539–5544, 2013.

[104] Amit Bhatia, Matthew R. Maly, Lydia E. Kavraki, and Moshe Y. Vardi. Motion planning with complex goals. *IEEE Rob. Autom. Mag.*, 18(3):55–64, 2011.

[105] Yiqun Dong, Efe Camci, and Erdal Kayacan. Faster rrt-based nonholonomic path planning in 2d building environment using skeleton-constrained path biasing. *J. Intell. Robot. Syst.*, 89:387–401, 2018.

[106] Seth McCammon, Gilberto Marcon dos Santos, Matthew Frantz, T.P. Welch, Graeme Best, R. Kipp Shearman, Jonathan D. Nash, John A. Barth, Julie A. Adams, and Geoffrey A. Hollinger. Ocean front detection and tracking using a team of heterogeneous marine vehicles. *JFR*, 2020.

[107] Derek A. Paley, Fumin Zhang, and Naomi Ehrich Leonard. Cooperative control for ocean sampling: The glider coordinated control system. *IEEE Trans. Control Syst. Technol.*, 16(4):735–744, 2008.

[108] Jaime Hernandez-Lasheras and Baptiste Mourre. Dense CTD survey versus glider fleet sampling: comparing data assimilation performance in a regional ocean model west of sardinia. *Ocean Sci.*, 2018(5):1069–1084, 2018.

[109] Zhi Yan, Nicolas Jouandeau, and Arab Ali Cherif. A survey and analysis of multi-robot coordination. *Int. J. Adv. Robot. Syst.*, 10(12):1–18, 2013.

[110] Rajesh Doriya, Siddharth Mishra, and Swati Gupta. A brief survey and analysis of multi-robot communication and coordination. In *Proc. of IEEE ICCCA*, pages 1014–1021, 2015.

[111] Gildardo Sánchez and Jean-Claude Latombe. Using a PRM planner to compare centralized and decoupled planning for multi-robot systems. In *Proc. of IEEE ICRA*, pages 2112–2119, 2002.

[112] Kiril Solovey, Oren Salzman, and Dan Halperin. Finding a needle in an exponential haystack: Discrete RRT for exploration of implicit roadmaps in multi-robot motion planning. *Int. J. Robot. Res.*, 35(5):501–513, 2016.

[113] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Mach. Learn.*, 47(2-3):235–256, 2002.

[114] Daniel S. Bernstein, Robert Givan, Neil Immerman, and Shlomo Zilberstein. The complexity of decentralized control of markov decision processes. *Math. Oper. Res.*, 27(4):819–840, 2002.

[115] Frans A. Oliehoek and Christopher Amato. *A concise introduction to decentralized POMDPs*, volume 1. Springer, 2016.

[116] Micah Corah and Nathan Michael. Distributed matroid-constrained submodular maximization for multi-robot exploration: Theory and practice. *Auton. Robots*, 43 (2):485–501, 2019.

[117] Graeme Best, Oliver M. Cliff, Timothy Patten, Ramgopal R. Mettu, and Robert Fitch. Dec-MCTS: Decentralized planning for multi-robot active perception. *Int. J. Robot. Res.*, 38(2-3):316–337, 2019.

[118] Chanyeol Yoo. *Provably-Correct Task Planning for Autonomous Outdoor Robots.* PhD thesis, University of Sydney, 2014.

[119] Russ E Davis, Charles C Eriksen, and Clayton P Jones. Autonomous buoyancy-driven underwater gliders. *The technology and applications of autonomous underwater vehicles*, pages 37–58, 2002.

[120] Colette Kerry, Brian Powell, Moninya Roughan, and Peter Oke. Development and evaluation of a high-resolution reanalysis of the East Australian Current region using the Regional Ocean Modelling System (ROMS 3.4) and Incremental Strong-Constraint 4-Dimensional Variational (IS4D-Var) data assimilation. *Geoscientific Model Development*, 9(10):3779 – 3801, 2016.

[121] Chanyeol Yoo, Stuart Anstee, and Robert Fitch. Stochastic path planning for autonomous underwater gliders with safety constraints. In *Proc. of IEEE/RSJ IROS*, pages 3725–3732, 2019.

[122] K. Y. Cadmus To, Chanyeol Yoo, Stuart Anstee, and Robert Fitch. Distance and steering heuristics for streamline-based flow field planning. In *Proc. of IEEE ICRA*, pages 1867–1873, 2020.