

**MORI-ZWANZIG FORMALISM BASED REDUCED-ORDER MODELING FOR  
DECISION-MAKING IN MARINE AUTONOMY**

A Dissertation  
Presented to  
The Academic Faculty

By

Mengxue Hou

In Partial Fulfillment  
of the Requirements for the Degree  
Doctor of Philosophy in the  
School of Electrical and Computer Engineering

Georgia Institute of Technology

August 2022

Copyright © Mengxue Hou 2022

**MORI-ZWANZIG FORMALISM BASED REDUCED-ORDER MODELING FOR  
DECISION-MAKING IN MARINE AUTONOMY**

Approved by:

Dr. Fumin Zhang, Advisor  
School of Electrical and Computer  
Engineering  
*Georgia Institute of Technology*

Dr. Catherine R. Edwards  
Department of Marine Sciences  
*University of Georgia*

Dr. Seth Hutchinson  
School of Interactive Computing  
*Georgia Institute of Technology*

Dr. Haomin Zhou  
School of Mathematics  
*Georgia Institute of Technology*

Dr. Enlu Zhou  
School of Industrial and Systems  
Engineering  
*Georgia Institute of Technology*

Dr. Ye Zhao  
School of Mechanical Engineering  
*Georgia Institute of Technology*

Date Approved: July 21, 2022

I dedicate this dissertation to my parents, Prof. Zhongsheng Hou and Xiujuan Zhang, for the constant support throughout my life, and my advisor, Prof. Fumin Zhang, who guided me to pursue my dreams.

## ACKNOWLEDGEMENTS

This thesis would not have been possible without the help and support of many wonderful people I have had the privilege of meeting and working with over the course of my PhD studies.

A great deal of thanks goes to my advisor Prof. Fumin Zhang. Your help and advise have been invaluable to me. Your passion and enthusiasm inspire me to explore the world of robotics and autonomy. Without your encouragement, I wouldn't be able to go through the hardest days. I will always be grateful for the past six years I spent with you.

I would like to thank Prof. Catherine R. Edwards, who has been a real mentor and friend to me. It has been an honor collaborating with you. I am grateful for the advise you gave me over the years. Also, I would like to thank Prof. Haomin Zhou and Prof. Enlu Zhou. I learned so much from you in our collaboration. The discussion and meetings we had motivates me to dive deeper in my research. I would like to thank Prof. Seth Hutchinson, for the discussion we had in Shenzhen, and also at the defense. Thank you for your time and valuable suggestions. Lastly, I would like to Prof. Ye Zhao, for the great comments and encouragement.

I would like to thank my friends L. Wang, J. Wang, and Y. Pan. I appreciate all the good time we spent together, and thank you for the support through my hard times. Hope we can see each other soon. Special thanks to my fellow labmates, who provide great collaboration and assistance in my study.

## TABLE OF CONTENTS

<b>Acknowledgments</b> . . . . .	iv
<b>List of Tables</b> . . . . .	ix
<b>List of Figures</b> . . . . .	ix
<b>Summary</b> . . . . .	xiv
<b>Chapter 1: Introduction</b> . . . . .	1
<b>Chapter 2: Background and motivation</b> . . . . .	4
2.1 Deterministic planning methods . . . . .	4
2.1.1 Mixed-Integer type decision-making . . . . .	5
2.1.2 Bounded cost search methods . . . . .	7
2.2 Belief abstraction . . . . .	8
2.3 The Mori-Zwanzig formalism . . . . .	9
<b>Chapter 3: Reduced-order modeling of ocean flow field</b> . . . . .	11
3.1 Problem formulation . . . . .	11
3.1.1 Vehicle dynamics . . . . .	11
3.1.2 Data-driven flow modeling . . . . .	12
3.2 Data-driven flow modeling . . . . .	14

3.2.1	Basis function of the flow field model . . . . .	14
3.2.2	Adaptive parameter estimation . . . . .	16
3.3	Theoretical justification . . . . .	17
3.4	Simulation results . . . . .	24
3.4.1	Experimental and simulation setup . . . . .	24
3.4.2	Flow modeling using glider experimental data . . . . .	25
<b>Chapter 4:</b>	<b>Path planning in reduced-order flow field: the method of evolving junctions . . . . .</b>	<b>28</b>
4.1	Problem formulation . . . . .	29
4.2	Our method . . . . .	30
4.2.1	Minimize total travel time . . . . .	32
4.2.2	Minimize energy . . . . .	34
4.2.3	Construction of decision tree . . . . .	36
4.2.4	Complexity analysis . . . . .	41
4.3	Completeness . . . . .	43
4.4	Simulation results . . . . .	44
4.4.1	Jet flow in 3D space . . . . .	44
4.4.2	Surface ocean flow . . . . .	46
<b>Chapter 5:</b>	<b>Path planning in reduced-order flow field: the modified potential search method . . . . .</b>	<b>50</b>
5.1	Problem formulation . . . . .	51
5.2	A modified potential search method for AUV path planning . . . . .	51
5.3	Theoretical justification . . . . .	56

5.3.1	Optimality proof . . . . .	56
5.3.2	Complexity analysis . . . . .	60
5.4	Simulation results . . . . .	61
<b>Chapter 6: Mori-Zwanzig Formalism based Belief Abstraction for Uncertainty-aware Decision-making . . . . .</b>		<b>65</b>
6.1	Preliminaries and problem formulation . . . . .	66
6.1.1	Preliminaries . . . . .	66
6.1.2	Problem formulation . . . . .	71
6.2	Focused finite partitioning . . . . .	75
6.3	Identification of the partitioned belief dynamics . . . . .	76
6.3.1	Learning the memory terms via LSTM . . . . .	78
6.3.2	Bayesian update in the partitioned state space . . . . .	79
6.3.3	Online M-Z formalism based belief dynamics . . . . .	81
6.4	Theoretical analysis . . . . .	81
6.4.1	Model reduction error . . . . .	82
6.4.2	Error bound analysis . . . . .	82
6.5	Simulation results . . . . .	87
6.5.1	Predicate transition . . . . .	88
6.5.2	Simulation setup . . . . .	90
6.5.3	Identification of the reduced dynamics . . . . .	94
6.5.4	Symbolic planning . . . . .	95
<b>Chapter 7: Conclusion and Future Work . . . . .</b>		<b>97</b>

7.1	Conclusion . . . . .	97
7.2	Future research directions . . . . .	97
	<b>References . . . . .</b>	<b>106</b>

## LIST OF TABLES

3.1	Root mean square error between the estimated and the true flow parameters.	27
4.1	Comparison between using the proposed method, LSM for time-optimal path planning . . . . .	45
5.1	Computation time comparison of A*, Level Set Method, and the proposed algorithm. Avg. comp. time represents the averaged computation time for each simulation scenario, and STD comp. time represents the standard deviation of the computation time. % of increase describes the percentage increase in the computation cost when $d$ increases. . . . .	64

## LIST OF FIGURES

3.1	<p><b>(Left):</b> Survey domain near Cape Hatteras. The curve represents glider trajectory during the first PEACH deployment. The red line path is the pre-assigned sampling pattern. Squares denote the glider surfacing positions along trajectory, and color of the trajectory depicts timestamps. The arrows represent the NCOM-predicted flow field at the starting time of the deployment.<b>(Right):</b> Flow partition error when the number of cells is set as <math>k = 1, \dots, 30</math>.</p>	25
3.2	<p><b>(Left):</b> Partitioned cells of the survey domain. The polygons are the partitioned regions. The blue arrows represent uniform flow speed in each of the cells generated from the proposed algorithm. <b>(Right):</b> Estimated flow parameters and the ground truth value in cell 7.</p>	26
4.1	<p>Example of parameterizing a feasible path in the partitioned domain by the cell sequence and the junction position. The purple triangles are the junction position. The path goes through 3 cells. Index of 3 cells are <math>c_1 = 1, c_2 = 2, c_3 = 4</math>. On the boundary <math>f_{12}</math> and <math>f_{24}</math> there are two junctions, position of which is parameterized by <math>\lambda_1</math> and <math>\lambda_2</math>.</p>	32
4.2	<p><b>(Left):</b> Example of a partitioned space. The domain is partitioned into 6 cells. the red line represents one feasible path from the start to the destination position, with the junctions represented by the purple triangles. <b>(Right):</b> The entire decision tree for this partitioned space. The root node shown by the green circle is the boundary curve containing the starting position, and the terminal node (yellow circle) represents the boundary curve containing the destination node. For each node, it is connected to its child node if it is an adjacent boundary to its child. The red nodes represent the nodes with no unvisited neighboring nodes. The red path in the graph corresponds to the cell sequence crossed by the feasible path shown on the left figure.</p>	41

4.3	<p><b>(Left):</b> Time optimal path planned by the proposed method. <b>(Right):</b> Energy optimal path planned by the proposed method.. In both plots, boundaries of the jet flow are denoted by the colored surfaces. The flow speed in the domain is represented by the blue arrows. The optimal path is marked by black line, while the marker position denotes junction points computed by the proposed method. . . . .</p>	45
4.4	<p><b>(Left):</b> Surface ocean flow field on May 27, 2017, 00:00 UTC at Cape Hatteras, NC. Red triangle and star indicate the starting and goal position (when <math>d = 100</math> km). . . . .</p>	47
4.5	<p>Comparison between using the proposed method and existing algorithms for time-optimal path planning. We compare the LSM and the A* method with the proposed algorithm, in the cases when the flow field is partitioned into 19, 39 and 59 cells. The bar is showing the computation time and total cost of each algorithm. The computation time is normalized by the largest computation time among all the algorithms (the largest computation time is normalized as 1), and the total cost is also normalized by the largest among all the algorithms (the largest total cost is normalized as 1). <b>(Left):</b> 40 km case; <b>(Middle):</b> 100 km case; <b>(Left):</b> 130 km case. . . . .</p>	48
4.6	<p>Optimal path when <math>d = 40</math> km (Upper row), <math>d = 100</math> km (Middle row), and <math>d = 130</math> km (Lower row). <b>(Left):</b> Time optimal path planned by the proposed method, the A* method and the LSM. <b>(Right):</b> Energy optimal path planned by the proposed method, when <math>C</math> takes different value. The optimal path is marked by colored line, while the marker position denotes junction points computed by the proposed method. . . . .</p>	49
5.1	<p><b>(A)</b> Partitioned cells in the domain. On each boundary of two adjacent cells there is a candidate junction point, represented as the purple triangle. <b>(B)</b> Graph representation of the workspace. The vertices represent the candidate junctions, while the edges are the path segment between the adjacent junctions. The red line on both of the plots represent the same example path. . . . .</p>	53
5.2	<p>Illustration of computing <math>PT(n_1)</math> and <math>PT(n_2)</math>. The red triangle is <math>S_1 = \{(x, y)   h_1y + g_1x &lt; C, x \in [1, H_{\max}], y \in [1, H_{\max}]\}</math>, and the green triangle is <math>S_2 = \{(x, y)   h_2y + g_2x &lt; C, x \in [1, H_{\max}], y \in [1, H_{\max}]\}</math>. . . . .</p>	58
5.3	<p>Example of simulation case. The resulting path is computed by the proposed method. . . . .</p>	62

6.1	Illustration of AUV actions. To achieve the hotspot search goal, the vehicle can perform two sets of actions, to move in a certain direction, or to go to the surface of the ocean for active localization. The circles demonstrate the uncertainty level of vehicle position. The uncertainty level goes up as the vehicle performs move action (upper panel), while the vehicle can actively reduce the uncertainty by taking the surface/diving action (lower panel). . . .	72
6.2	Illustration of the focused finite partition algorithm. We stack up the particle trajectories at all timesteps in the Monte-Carlo simulation, and use it to solve (6.24). . . . .	76
6.3	Graphical representation of the M-Z based TD-NODE for modeling the memory effect. Each cell in the LSTM layer represents an LSTM cell described by (6.28). . . . .	80
6.4	Comparison between the full belief dynamics and the reduced-order approximation model in one iteration of the estimation process. The full dynamics includes the prediction step and the updating step (the Bayesian law). The approximation model consists of prediction using the TD-NODE model and the update step by the parameter space Bayesian law (6.32). . . . .	82
6.5	An illustration of the belief state evolution in an environment with 2 actions, move to the left ( $\text{move}_\pi$ ), and move to the right ( $\text{move}_0$ ). <b>(Left)</b> : The decision tree at the initial timestep. Each branch of the decision tree represents the agent taking a specific action. After the action is taken, the belief at the next timestep is computed by the MZ-based TD-NODE model. The decision tree construction is finished until a node fulfills the terminal constraint. <b>(Right)</b> : Updated belief state after a $\text{move}_0$ is taken at the initial timestep, and a new observation is received, which confirms the state is contained in the green cell. After the Bayesian update, the new belief state is taken as the root of a new decision tree, and the agent takes the same procedure for decision tree construction to begin a new search. . . . .	89
6.6	Setup of the simulation domain. . . . .	92
6.7	Partitioned state space. The colored squares represent datapoint position in Monte Carlo simulation. Squares with same color belong to the same partition. <b>(Left)</b> : Partition for vehicle state. <b>(Right)</b> : Partition for hotspot state. . . . .	94

6.8 Model reduction error comparison between the proposed algorithm and the GCM. **(Left)**: One step model reduction error (training set) **(Middle)**: One step model reduction error (validation set). **(Right)**: Multi-step model reduction error comparison between the GCM and the LSTM-based modeling. The model reduction error of GCM shows significant growth over time, while the model reduction error of LSTM-based modeling stays relatively constant, which is consistent as the theoretical analysis. . . . . 95

6.9 Simulation comparison between proposed algorithm and GCM, with omniscient  $A^*$  as baseline. . . . . 96

## SUMMARY

Some robotic systems, such as the autonomous underwater vehicles (AUV), are strongly influenced by the environmental processes, which introduces high complexity in system dynamics. For such autonomous systems, directly specifying the full system behavior in real-time can be computationally impractical. Hence it is desired to introduce abstraction of system dynamics and modularity in actions, such that the computation cost can be significantly reduced. However, the abstraction introduces new challenges. First is the mixed-integer type decision-making due to the abstraction of system dynamics and the hierarchical representation of system behaviors. Another challenge is the degraded solution quality due to the model reduction error incurred from the abstraction. This thesis first considers the reduced-order modeling and path planning of a deterministic system, an example application of which is the AUV deterministic planning problem in the ocean flow field. We develop a data-driven reduced-order model of the flow dynamics. The proposed algorithm partitions the flow field into piece-wise constant flow speed, and leverages the real-time AUV measurements to learn the parameters in the reduced-order model. Such abstraction transforms the infinite dimensional planning problem into a mixed integer optimization problem (MIP). We develop an interleaved MIP solution with guaranteed completeness and optimality, and the computation cost is shown to be lower than the existing AUV planning methods. To further reduce the computation cost of solving the MIP, a bounded cost search algorithm is developed to compute a bounded cost sub-optimal solution of the MIP, with proved completeness. Further, we consider the reduced-order modeling and belief space planning of a continuous-state POMDP (partially observable Markov decision process), and develops a belief abstraction method to facilitate symbolic planning in the belief space. As unmodeled residual state has impact on the reduced-order dynamics, the abstracted belief dynamics is non-Markovian. Hence we identify the abstracted belief dynamics using a recurrent neural network, and develops a reduced-

order Bayesian law based on the Mori-Zwanzig formalism. Both theoretical analysis and simulation results show that the proposed method provides high fidelity approximation of the belief dynamics. Further, we show through simulation that the belief abstraction reduces the computation cost in solving the planning problem in belief space.

# CHAPTER 1

## INTRODUCTION

Over the last few decades, AUVs have been employed for ocean sampling [1, 2], surveillance and inspection [3, 4], and many other applications. The AUVs need to be capable to perform long-term autonomy tasks requiring minimal human intervention in the face of uncertain, unanticipated, and dynamically changing underwater environment. All these factors pose challenges to long-term marine autonomy missions, and require the autonomous system to be aware of the uncertainty, identify the environmental changes, and adapt its decision-making strategies to the environmental changes in real-time. However, the ocean processes are often modeled by highly complicated geophysical partial differential equations. On one hand, accurately model the environmental dynamics is crucial. Due to the limited actuation forces, the AUVs are highly influenced by the environmental states. Hence the solution quality of the decision-making process is highly dependent on the accuracy of the environmental modelling and estimation of its state. On the other hand, due to the complexity of ocean flow dynamics, directly specifying the full behavior policy for AUV in real time can be impractical since the computation cost can be inhibitingly high, especially for complex tasks. Hence it is desired to develop computationally efficient decision-making algorithms, such that the computation cost can be significantly reduced.

One existing framework to reduce the computation cost of decision making is through abstraction in system dynamics and modularity in actions [5]. The dimension of the search space can be significantly reduced by introducing a hierarchical structure and decomposing tasks into subtasks, Similar strategies have also been introduced to use logical and abstracted representation of the system dynamics and precondition of tasks. However, introducing abstraction into modeling poses new challenges in solving the planning problem. As the logical representation introduces a discrete representation of the system, the continuous

planning problem is converted to a Mixed Integer Optimization (MIP) problem, where the integer is resulted from both the modular representation of the system dynamics and the hierarchical task structure. Solving the resulting MIP problem is challenging. The challenge arises both in computation cost, as well as in the theoretical aspect to guarantee solution optimality and completeness of algorithms.

The goal of this thesis is to develop high fidelity abstracted representation of system and environmental dynamics in marine autonomy applications, such that high quality optimal policy can be derived with lower computation cost through symbolic planning. We first consider the AUV decision making problem in a deterministic environment, where we derive a reduced-order data-driven representation of the environmental dynamics, and propose an algorithm to identify the parameters in the reduced-order model. Given such approximation model, two algorithms are proposed to solve the path planning problem in the reduced-order flow map, as an instance of the MIP problem. Further, we consider the AUV navigation problem under environmental uncertainty and localization uncertainty, and develop a belief abstraction algorithm for symbolic planning. The major contribution of this work lies on both construction of the abstracted dynamics and the optimal path planning on the reduced-order model, as listed below:

- 1 *Data-driven computation of abstracted flow field:* In Chapter 3, we develop a data-driven computational flow model that approximates the true flow field in the region of interest with a piece-wise constant one, to facilitate AUV path planning. The proposed data-driven flow model divides the flow field into cells of uniform flow speed, based on the similarity of flow vectors at different locations in the domain. To improve model accuracy, an adaptive learning method is developed to learn the flow parameter in each partitioned cell from the real-time AUV measurements, with justified convergence.
- 2 *Interleaved solver for MIP:* In Chapter 4, we propose a novel computationally efficient method to solve the path planning problem in the partitioned flow field, as an instance

of the MIP, and proved completeness and optimality of the algorithm. Different from the existing works that decouples the process of optimizing the cell sequence assignment and the intersection position, our method optimizes the cell sequence assignment and the intersection position between the path and the cell boundaries at the same time. By incorporating a Branch and Bound (BnB) technique in the Depth first search (DFS), the method avoids exhaustive search through the full decision tree. Hence it solves the planning problem with low computation cost.

3 *A bi-level fast sub-optimal solver for bounded cost planning:* In Chapter 5, we develop a bi-level computationally efficient MIP solver. Instead of solving for the optimal solution, we reduce the MIP to finding solutions with highest probability to have total cost less than an upper bound. The work is a significant extension to the bounded cost search algorithm [6]. Unlike the bounded cost search method, which assumes that the branch cost of the graph is known exactly, our method deals with problems where the branch cost of the graph is uncertain. We present theoretical justification on the optimality of the proposed modified bounded cost search method.

4 *Mori-Zwanzig based learning of abstracted belief dynamics:* In Chapter 6, we propose a data-driven approach to identify the high-fidelity symbolic representation of the belief state and its dynamics in order to solve the belief space planning problems. We prove that by leveraging the Mori-Zwanzig formalism to model the memory effect, the belief abstraction scheme retains a time-uniform model reduction error upper bound.

The rest of the document is organized as follows. Chapter 2 reviews the literature relevant to the proposed research. Chapter 3 presents a reduced-order modeling method of ocean flow field modeling. Chapter 4 and Chapter 5 provides two path planning methods in the abstracted flow field. Chapter 6 describes a belief abstraction method for POMDP problems. Chapter 7 concludes the thesis by providing a summary and future work.

## CHAPTER 2

### BACKGROUND AND MOTIVATION

This Chapter provides background of planning methods in deterministic and stochastic environment. Specifically, the first part of the Chapter provides literature review on deterministic planning methods, specifically focusing on path planning methods for AUV traveling in ocean flow fields. The second part of the Chapter gives an overview on belief abstraction and symbolic planning for continuous-state POMDP. Lastly, we provide an overview of the M-Z formalism for reduced-order modeling.

#### 2.1 Deterministic planning methods

We first present survey of deterministic planning methods, focusing on its application on marine autonomy. In short, the task is finding an optimal path subject to a cost function, under the influence of a dynamic flow field, for an AUV to reach a predefined target point. Path planning has been studied extensively in robotics over the years. Several popular algorithms that have been applied to underwater vehicle navigation include graph based methods such as the A\* method [7, 8, 9, 10] and the Sliding Wavefront Expansion method (SWE) [11], probability based methods like the Rapidly exploring Random Trees (RRTs) [12, 13, 14, 15, 16], and methods that approximate the solution of HJ (Hamilton-Jacobi) equations, such as the Level Set Method (LSM) [17, 18].

When the A\* method is applied for an AUV, the continuous flow field is discretized into grid cells. At each step, it compares the cost of going from the current position to its neighboring cells so as to identify a path with the lowest cost. However, when the resolution, which is inverse proportional to the cell size, is not high enough, it may fail to find a feasible path even if there exists one. The SWE addresses the incompleteness issue of A\* by introducing the sliders, which are points that slide on the boundary of cells, as the nodes

in the graph search problem. As all feasible path connecting the starting and target position can be formed by a sequence of the sliders, the path planning problem is converted to a Mixed Integer Optimization problem (MIP) on cell sequence and slider positions. RRT and RRT\* explore the flow field by using random sampling, with a bias towards the unexplored area [19, 20]. Like many other probabilistic based methods, RRTs provide, guaranteed in the asymptotic sense, a globally optimal solution only when the samples are dense enough. Methods to improve the convergence rate of RRT and RRT\* have been introduced [21, 14]. LSM computes a propagating front, incorporating both flow and vehicle speeds, to approximate the solution of the HJ equation. Then the optimal path is computed by back tracking along the normal direction of the wavefront from the destination position. LSM can plan a shortest time path over time-varying flow field, usually at the cost of longer computational time.

### 2.1.1 Mixed-Integer type decision-making

Using a regular grid to discretize the flow field can result in unnecessary large number of cells, which increases the computational burden of the planning methods. Since the flow speed in adjacent position is usually similar, the flow field can be partitioned into piece-wise constant subfields, within each the flow speed is a constant vector [22, 10, 23, 24]. Taking advantage of the flow field partitioning, we can parameterize a path by the index of cells crossed by the path, and the intersection points between the path and the boundaries of regions. In this way, the original path planning problem in the continuous flow field is reduced to a finite dimensional MIP, with the decision variables being the index of cells crossed by the path, and the intersection points between the path and the boundaries of regions. Because of the coupling between the integer and continuous variable, the optimization problem is high dimensional, and is computationally expensive to solve.

The mixed-integer type decision-making is a commonly seen formulation in robotics and control community, for instance, task and motion planning, optimal control of hybrid

systems , scheduling-control co-design, and multi-agent task allocation problems. Below we review some existing solutions in solving the mixed-type decision-making.

Existing works following this approach solves the MIP in three kinds of methods [5]: sequence first, satisfaction first, and interleaved methods. The sequence first methods first determine the high level integer variables by performing graph search with an estimated cost that will incur from the lower-level actions, and then given the upper-level assignment, optimize the lower-level continuous variable with nonlinear optimization methods. Such method is first introduced in [25], where the STRIPS (Stanford Research Institute Problem Solver) is developed to generate the a higher level task plan. [26, 27] develops a hierarchical solver for the MIP, where the high-level task planning problem is solved first by utilizing the linear temporal logic for a reactive game synthesis, and then the lower-level motion planning problem is solved given the specified integer variable sequence. Such methods are computationally efficient, since the size of the search space is significantly reduced by decoupling the integer and continuous variable. [11, 10] propose a bi-level solver where the integer variable is optimized using the A\* method, with the branch cost of the decision tree being the optimal solution derived from the lower-level controller. However, such algorithms cannot guarantee optimality of the solution without imposing more assumptions on the problem [28], due to the decoupling between the integer and continuous variable. Further, the generated solution may not be feasible, since the lower-level dynamics is ignored when planning the high-level integer variables.

The satisfaction first methods first generate a set of feasible solution of lower-level nonlinear optimization problem, usually by sampling, then applies the search methods to find the optimal solution among the sampled solutions. [29] develops an algorithm to iteratively alternate between sampling the lower-level actions and planning until it is able to find a solution. The sampling phase creates a set of abstracted symbolic actions, and the planning phase performs discrete search to find a feasible solution. [30, 31] extends the probability road map method to solving multi-modal planning problems. These satisfaction

first methods generate feasible solutions, and is shown to be probabilistically complete. However, due to the large search space of MIP, a large number of sampled trajectories has to be generated, which introduces higher computation cost.

The interleaved methods iteratively optimize the integer and continuous variables. In general, there are two types of algorithms developed. One type of methods approximate the problem with quadratic objective and constraints, then use the MIP solvers, such as the ADMM (Alternating direction method of multipliers) and the MIQP (Mixed integer quadratic programming) to solve the problem. [32] is one of the earliest work in this field. It presents a Model Predictive Control (MPC) formulation with mixed logical dynamics, and optimize a quadratic objective function using the MIQP. [33] develops a branch and bound Mixed Integer Convex Program method by a novel convex relaxation of a class of bilinear constraints. [34] solves the mixed-type decision-making with polynomial objective and constraints as a quadratically constrained quadratic programming problem, by the ADMM. The other kind of methods apply the graph search techniques. [35] proposes a backward-forward method that first construct a reachability graph by backward sampling from the target set, such that the sampling is strongly focused on the actions towards the goal. Then a forward hill-climbing method is implemented to search for the optimal strategy in the reachability graph.

### 2.1.2 Bounded cost search methods

To reduce the computational cost of the path planning problem, the search can be reduced to find paths with total cost less than an upper bound. [6] present two algorithms to solve the bounded cost search problem: the Potential Search (PTS) and the Anytime Potential Search (APTS). The PTS method defines the Potential Ordering Function, which is an implicit evaluation of the probability that a node is on a path satisfying the bounded cost constraint, and iteratively expands the nodes in the graph with the highest Potential Ordering Function value. The wavefront expansion terminates when the goal node has been expanded,

and the path is found by backtracking of the wavefronts. The APTS method runs the PTS algorithm iteratively to improve on the incumbent solution, with the upper bound on total cost lowered in each iteration of the algorithm. Later work [36] improves on the PTS method by minimizing both the potential and an estimation of the remaining search effort, so that the bounded cost search problem will be solved faster.

## 2.2 Belief abstraction

Partially observable Markov decision processes (POMDPs) are Markov decision processes (MDPs) in which decisions must be made under state uncertainty. In continuous-state POMDPs, the introduction of a belief state, a continuous posterior distribution conditioned on historical observations [37], allows for the transformation of a POMDP problem to an infinite dimensional MDP problem. However, the infinite dimensional MDP has to be approximated by a finite rank model such that computation of the MDP becomes tractable. However, such approximation suffers from the curse of dimensionality, which makes calculation of an optimal policy intractable for high dimensional problems.

As a result, many methods in the POMDP literature have been proposed to approximate the continuous belief state so that the search for an optimal control strategy becomes tractable. The authors of [38] use Gaussian distribution to approximate the belief state and construct a deterministic approximation of the belief state dynamics. For nonlinear or non-Gaussian models, though, a single Gaussian distribution approximation is not a sufficient statistics and leads to suboptimal controllers. The paper [39] proposes a Gaussian mixture model, and develops a clustering-based technique for mixture condensation that scales well to large belief space. [40] develops a projected particle filter which parameterize the belief state with the exponential family of probability densities, and proves upper bound of the model reduction error resulting from the parameterization. Other methods, such as the generalized cell mapping method proposed in [41], discretize the belief state into rectangular grids and represent the belief dynamics as a Markov chain. However, such approximations may

result in MDPs that are either too large to solve computationally or insufficiently precise. Other methods as described in [42] design a Voronoi-diagram based partitioning scheme of the belief state by leveraging piece-wise constant clusters and the performing Monte Carlo simulations to identify the transition probabilities among clusters. However, as the number of partitions are not fixed, this approach may also result in an excessive number of partitions as the number of trajectories in the Monte Carlo simulations are directly related to the number of partitions. A significant number of simulations are necessary in order to precisely approximate the belief dynamics. Several continuous state POMDP approximation approaches rely on sampling methods [43, 44]. See [45] for a review.

Once an approximation of the belief state and deterministic belief dynamics have been identified, symbolic planning strategies can then search for an optimal sequence of control actions while under constraints. For example, classic graph search methods such as breadth first search, iterative deepening, and A\* have all been applied to solve symbolic search problems [46]. Compared to other continuous-state POMDP solutions, e.g., point-based and grid-based discretization strategies [47], [48] shows that symbolic search methods, due to the reduced dimension of the search space, can reduce the computation time of solving a POMDP. However, large action spaces can still dramatically hinder plan construction due to the large branching factor. As such, hierarchical task network (HTN) planners use primitive action groupings to prune the breadth of the search tree [49]. These approaches take advantage of the hierarchical nature of tasks, and search for plans by decomposing compound tasks until all tasks are decomposed into primitive actions. Thus, such approaches are well-suited to problems in which large groupings of actions are available.

### **2.3 The Mori-Zwanzig formalism**

All the methods mentioned above leverage an approximation model of the belief dynamics, converting the infinite dimensional MDP into a finite dimensional one. The principle of reduced-order modeling is to reduce the system such that the modeled state computed from

the approximation model stays close to the actual state. However, having a high-fidelity approximation model usually results in more complex structure of the model, which induces higher computation cost in both estimating the model parameters and solving the planning problem. On the other hand, with coarser modeling of the system the planning problem can be solved faster, but at the cost of sacrificing fidelity of the modeled dynamics, and thus may lead to suboptimal behavior and incur higher risk.

The trade-off between modeling accuracy and computation efficiency can be addressed by introducing memory effects into the approximated model based on the Mori-Zwanzig (M-Z) formalism [50]. As the model reduction error is partly induced by the impact of unmodeled dynamics on the reduced-order modeled dynamics, the approximation model can be improved by incorporating the coupling between the resolved (modeled) state and unmodeled state into the approximation model. The M-Z equation offers an exact description of the impact of the unmodeled variables on the reduced set of variables as a sequence of memory terms. To numerically compute the memory effect terms, many different approaches have been proposed, such as the T-model [51, 52] and the function derivatives methods [53], with error bound analysis established in [54]. In recent years, data-driven modeling methods have also been developed [55, 56]. However, all existing work address residual learning of time-invariant dynamical systems. However, dynamics of most robotic systems are not time-invariant. The actions take by the robot affect system dynamics, making the system dynamics time-varying. Thus the system dynamics described by the M-Z formalism not only depend on historical system states, but also depend on historical actions taken by the system, which makes the M-Z equation extremely high dimensional. Besides, all existing work addressing the residual learning using M-Z equation is considering fully observable systems. Partial observation will introduce significant difficulty in applying the M-Z formalism since the historical observation received will also affect the belief dynamics. The learned model must adapt to the received measurement online. All the existing algorithms on learning the M-Z equation cannot be applied in such scenarios.

## CHAPTER 3

### REDUCED-ORDER MODELING OF OCEAN FLOW FIELD

In this Chapter, we present a flow field partition method that extracts the key features, which are the spatial and temporal variation of the flow field. The partition method is developed based on K-means algorithm. Given the partitioning, we develop a learning algorithm that estimates the flow field parameters with justified convergence.

#### 3.1 Problem formulation

##### 3.1.1 Vehicle dynamics

Let  $F_R : \mathcal{D} \rightarrow \mathbb{R}^2$  represent a spatially distributed vector field for the ambient flow velocity, where  $\mathcal{D} \in \mathbb{R}^2$  is the domain of interest. Let  $[T_0, T_f]$  be the AUV deployment time interval. The AUV model is described as

$$\dot{x} = F_R(x) + V_R \Psi_C(t), \quad (3.1)$$

where  $x \in \mathcal{D}$  denotes vehicle position.  $V_R$  is the through-water speed of the vehicle, and  $\Psi_C(t) = [\cos \psi_C, \sin \psi_C]^T$  is a unit vector that represents the direction of the vehicle motion along heading angle  $\psi_C$ .

**Assumption 3.1.1.** *During the operation,  $V_R$  is an unknown constant.*

**Remark 3.1.1.** *Actual vehicle speed may depend on a number of factors that affect an AUV's speed, including water depth, efficiency of propulsion, and bio-fouling. These effects are difficult to estimate. Hence the vehicle forward speed is assumed to be an unknown constant.*

**Assumption 3.1.2.** We assume that the heading  $\Psi_C(t)$  can be controlled for all time  $t$ , and the vehicle trajectory  $x(t)$  can be measured or estimated for all time.

**Remark 3.1.2.** Though the actual location of a vehicle may only be known occasionally when the vehicle is underwater, the trajectory of the vehicle can be estimated through localization algorithms, which incorporates the known locations and the heading angle commands as inputs to generate the optimal state estimation.

**Assumption 3.1.3.** We assume the flow field is time-invariant throughout the deployment.

**Remark 3.1.3.** Even though there are existing work that considers the time-variant flow field in solving the AUV planning problem, such as [57, 58, 59], we make this assumption due to the patterns of the flow field in this domain. In the domain of interest considered in this paper, which is near Cape Hatteras, NC, the current field is driven by a combination and interaction of Gulf Stream, wind, and buoyancy forcing [60, 61]. Because magnitude and spatial gradients of the flow field are significant relative to the temporal variation of the flow field (mostly the tidal flow component), time variation of the flow does not have a significant influence over the planned path.

### 3.1.2 Data-driven flow modeling

Flow speed at neighboring grid points often exhibits similarity in both strength and direction. Hence we assume that at the time scale of an AUV deployment, the flow field can be divided into finite number of regions  $\{\mathcal{R}_i\}_{i \in I_R}$ , with the union of all cells being the domain,  $\cup_{i \in I_R} \{\mathcal{R}_i\} = \mathcal{D}$ . The regions are separated by continuous boundary curves. Boundary curves  $\{f_{i,j}\}_{i,j \in I_R}$ , and  $f_{ij}(x) = 0$  is the one dimensional compact boundary of the region  $\mathcal{R}_i$  and  $\mathcal{R}_j$ . We define an indicator function  $\phi^i(x)$  as follows:

$$\phi^i(x) = \mathbb{1}\{x \in \mathcal{R}_i\} = \begin{cases} 1 & \text{if } x \in \mathcal{R}_i \\ 0 & \text{otherwise} \end{cases}.$$

This function indicates whether  $x$  is in  $\mathcal{R}_\alpha$ . Let  $\phi : \mathbb{R}^2 \rightarrow \mathbb{R}^N$  be defined as  $\phi(x) = \begin{bmatrix} \phi^1(x) & \dots & \phi^N(x) \end{bmatrix}^\top$ . Then  $\phi(x)$  are a set of spatial basis functions of  $\mathcal{D}$ .

In order to compute the partition, which is represented by the basis functions  $\phi$ . We need to use prior information of the environment obtained either from forecast data of the existing ocean models, or from historical datasets. Let  $F_0 : \mathcal{D} \times [T_0, T_f] \rightarrow \mathbb{R}^2$  denote the discretized flow map forecast available on a set of grid points in  $\mathcal{D}$ , and let  $y \in \mathbb{R}^4, y = [x^\top, F_0(x, t)^\top]^\top$  denote the vector at position  $x$  and time  $t$ . We can define a distance function as  $\text{dist} : \mathbb{R}^4 \times \mathbb{R}^4 \rightarrow \mathbb{R}$  as  $\text{dist}^2(y, y') = (y - y')^\top Q (y - y')$ , where  $y, y' \in \mathbb{R}^4$ ,  $Q$  is a weight matrix. For each cell  $\mathcal{R}_i$ , let  $\bar{r}_i$  represents its center, and let  $\nu_i$  be the flow vector in this cell. Our goal is to find the optimal values of  $\phi(x)$  and  $\{\nu_i\}_{i \in I_R}$  by solving the following optimization problem:

$$\min_{\phi(x), \{\nu_i\}_{i \in I_R}} J = \sum_{i \in I_R} \sum_{x \in \mathcal{R}_i} \sum_{t \in [T_0, T_f]} \text{dist}^2(y(x, t), [\bar{r}_i^\top, \nu_i^\top]^\top). \quad (3.2)$$

After the optimal partition is computed from forecast or historical datasets, we need to compute the strength of the flow in each partition based on the path information of the AUV while moving through the flow field. Again, we assume that the true flow field is constant in each of the partitioned cell. Let  $\theta \in \mathbb{R}^{2N}$  be the true flow vectors in all of the partitioned cells,  $\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} = \begin{bmatrix} \theta_1^1 & \dots & \theta_1^N \\ \theta_2^1 & \dots & \theta_2^N \end{bmatrix}$ , with  $\theta^i = [\theta_1^i, \theta_2^i]^\top$  denoting the flow vector in partitioned region  $\mathcal{R}_\alpha$ . Then the partitioned flow field can be represented as

$$F_R(x) = \theta \phi(x). \quad (3.3)$$

To estimate the true flow field  $F_R$ , we use the AUV path data  $x(t)$ . Let

$$\xi(t) = \begin{bmatrix} \xi_1(t) \\ \xi_2(t) \end{bmatrix} = \begin{bmatrix} \xi_1^1(t) & \dots & \xi_1^N(t) \\ \xi_2^1(t) & \dots & \xi_2^N(t) \end{bmatrix}$$

be our estimate of the parameter  $\theta$  and  $V_L(t)$  be our estimate for  $V_R$ . We will design a learning algorithm to achieve convergence of  $\xi(t)$  and  $V_L(t)$  to the true values e.g.,  $\xi(t) \rightarrow \theta$  and  $V_L(t) \rightarrow V_R$  as  $t \rightarrow \infty$ .

## 3.2 Data-driven flow modeling

### 3.2.1 Basis function of the flow field model

First let us describe Algorithm 1. We derive the spatial basis function and the initialized flow model parameters by solving (3.2) using the K-means algorithm. Since this optimization depends on both spatial and temporal variance of the flow field, solving this problem can be computationally expensive. To simplify this problem, instead of optimizing the difference between the time-varying flow forecast and the partitioned flow field, as described in (3.2), we optimize the difference between the time-averaged flow forecast and the partitioned flow field:

$$\min_{\phi(x), \{\nu_i\}_{i \in I_R}} J' = \sum_{i \in I_R} \sum_{x \in \mathcal{R}_i} \text{dist}^2(\bar{y}(x), [\bar{r}_i^\top, \nu_i^\top]^\top), \quad (3.4)$$

where  $\bar{y}(x) = [x^\top, \frac{1}{T_f - T_0} \sum_{t \in [T_0, T_f]} F_0(x, t)^\top]^\top$  denote the time averaged flow observation at position  $x$ .

To implement the K-means algorithm, we start by randomly selecting  $k$  cell centroids, and then use Lloyd iterations to solve the optimization problem. The Lloyd iteration contains two steps, first assign the points that are closest to a centroid to that centroid, and then recompute the cell centroid. These two steps are repeated until cell membership no longer changes. The K-means algorithm requires proper selection of the number of partitioned cells,  $k$ , the choice of which affects the path planning performance and flow modeling quality. If the field is divided into too many regions, then it results in a complicated flow structure and potentially increases the computational cost of path planning. On the other hand, dividing the field into too few regions may result in a large error between the true flow field and the modeled flow field, which potentially leads to significant path planning error. Therefore, we

---

**Algorithm 1: Flow Field Partition Algorithm**

---

**Data:** flow field observations  $\{y(x_i, t)\}, i \in [1, m], t \in [T_0, T_f]$ , flow partition error threshold  $\epsilon$ .

**Output:** spatial basis function  $\phi(x_i)$ , piece-wise constant flow speed of cells  $\nu_j, j \in [1, k]$ .

- 1  $\bar{F}_0(x_i) = \frac{1}{T_f - T_0} \sum_{t \in [T_0, T_f]} y(x_i, t), \bar{y}_i = [x_i^\top, \bar{F}_0(x_i)^\top]^\top$ ;
- 2 Initialize cluster number  $k = 1, \bar{r}_1 = \frac{1}{m} \sum_{i=1}^m x_i, \nu_1 = \frac{1}{m} \sum_{i=1}^m \bar{F}_0(x_i)$ ;
- 3 Compute  $\delta F$  using (3.5) ;
- 4 **while**  $\delta F > \epsilon$  **do**
- 5      $k = k + 1$ ;
- 6     Randomly initialize cluster centroids;
- 7     **while not converges do**
- 8         For all  $i \in [1, m]$ , set  $c_i = \arg \min_j \text{dist}^2(\bar{y}_i - [\bar{r}_j^\top, \nu_j^\top]^\top)$  ;
- 9         For all  $j \in [1, k]$ , set  $\bar{r}_j = \frac{\sum_i \mathbb{1}(c_i=j)x_i}{\sum_i \mathbb{1}(c_i=j)}, \nu_j = \frac{\sum_i \mathbb{1}(c_i=j)\bar{F}_0(x_i)}{\sum_i \mathbb{1}(c_i=j)}$  ;
- 10        Compute  $\delta F$  using (3.5) ;
- 11     **end**
- 12 **end**
- 13 For all  $i \in [1, m], \phi(x_i) = [\mathbb{1}\{c_i = 1\} \dots \mathbb{1}\{c_i = k\}]^\top$ ;
- 14 For all  $j \in [1, k], \mathcal{R}_j = \cup_i \mathbb{1}\{c_i = j\}x_i$ ;

---

introduce an iterative K-means algorithm that can guarantee a bounded flow field partition error, and at the same time utilize the smallest number of partition regions.

Let  $\bar{F}_0 : \mathcal{D} \rightarrow \mathbb{R}^2$  denote the time-averaged flow field over the time interval  $[T_0, T_f]$ , and  $\nu_i$  is the uniform flow velocity in  $\mathcal{R}_i$ . Define the flow field partition error as:

$$\delta F = \max_{i \in I_R} \max_{x \in \mathcal{R}_i} \|\bar{F}_0(x) - \nu_i\|. \quad (3.5)$$

Given an initialized  $k$ , we will iteratively perform the K-means algorithm (lines 4, 5), and check if the flow partition error satisfies  $\delta F < \epsilon$ , where  $\epsilon$  is a pre-defined upper bound on the flow partitioning error. If this condition is satisfied, then the current  $k$  is designated for partitioning; otherwise, the number of cells is increased by 1, and we recompute the solution to (3.4) using K-means method.

---

**Algorithm 2:** Flow Estimation Algorithm

---

**Data:** Vehicle trajectory  $x(t)$ , estimated vehicle trajectory  $z(t)$ , heading angle  $\Psi_C(t)$ , initial estimated parameter  $\bar{\xi}(0)$ , initial estimated speed  $V_L(0)$ .

**Output:** Estimated parameter  $\bar{\xi}(t+1)$ , estimated flow parameter  $\xi(t+1)$ , estimated vehicle speed  $V_L(t+1)$ .

```
1 while  $t < \text{ending time}$  do
2    $e(t) = x(t) - z(t)$ ;
3   Update  $\beta(t)$  using (3.8);
4   Update  $\bar{\xi}$  and  $V_L(t)$  using (3.10);
5   Update  $z(t)$  by integrating (3.6);
6 end
```

---

### 3.2.2 Adaptive parameter estimation

Now let us explain Algorithm 2. An estimate  $z(t)$  for the vehicle trajectory can be computed by integrating

$$\dot{z} = \xi(t)\phi(z) + V_L(t)\Psi_C + \beta(t), \quad (3.6)$$

where  $\beta(t) \in \mathbb{R}^2$  is introduced as a learning injection parameter. The term  $e = x - z$  is the controlled Lagrangian Localization error (CLLE, [62, 63]), which describes how much the actual trajectory deviates from the estimated trajectory. A learning algorithm will then compute  $\beta(t)$ ,  $\xi(t)$ ,  $V_L(t)$  so that the CLLE can be reduced.

The CLLE dynamics can be derived from (3.6) and (3.1),

$$\dot{e} = \dot{x} - \dot{z} = \theta\phi(x) - \xi(t)\phi(z) + (V_R - V_L(t))\Psi_c - \beta(t). \quad (3.7)$$

We design the learning parameter injection as

$$\beta(t) = \xi(t)\phi(x) - \xi(t)\phi(z) + Ke, \quad (3.8)$$

and hence the CLLE dynamics becomes

$$\dot{e} = -Ke + (\theta - \xi(t))\phi(x) + (V_R - V_L(t))\Psi_c. \quad (3.9)$$

The learning algorithm updates parameters  $\xi(t)$  and  $V_L(t)$  so that the CLLE converges to zero. Let  $\bar{\xi}(t) = [\xi_1^1(t), \dots, \xi_1^N(t), \xi_2^1(t), \dots, \xi_2^N(t)]^\top$ ,  $\bar{\theta} = [\theta_1^1, \dots, \theta_1^N, \theta_2^1, \dots, \theta_2^N]^\top$ , and  $e \otimes \phi = [e_1\phi^1, \dots, e_1\phi^N, e_2\phi^1, \dots, e_2\phi^N]^\top$ , where  $\otimes$  is the Kronecker product. We design the updating rules for parameter estimation as follows,

$$\begin{aligned}\dot{\bar{\xi}}(t) &= \rho e \otimes \phi(x) \\ \dot{V}_L(t) &= \rho e^\top \Psi_c.\end{aligned}\tag{3.10}$$

These rules are then used in Algorithm 2.

### 3.3 Theoretical justification

Algorithm 1 can be theoretically justified by proving that the optimal solution to (3.4) is the optimal solution to (3.2). We also prove that Algorithm 2 achieves error convergence and parameter convergence, indicating that the estimated trajectory converges to the actual trajectory, and the estimated parameter converges to the true values.

**Lemma 3.3.1.** *The optimal flow partition derived by solving (3.2) is the same as the optimal flow partition derived from (3.4).*

*Proof.* Let  $\delta y(x, t) = y(x, t) - \bar{y}(x)$ . Since  $\sum_{t=T_0}^{T_f} \delta y(x, t) = \sum_{t=T_0}^{T_f} y(x, t) - (T_f - T_0)\bar{y}(x) = 0$ , the following equality holds

$$\begin{aligned}J &= \sum_{i=1}^N \sum_{x \in \mathcal{R}_i} \sum_{t=T_0}^{T_f} \text{dist}^2(y(x, t), \mu_i) \\ &= \sum_{i=1}^N \sum_{x \in \mathcal{R}_i} \sum_{t=T_0}^{T_f} \text{dist}^2(\bar{y}(x) + \delta y(x, t), \mu_i) \\ &= \sum_{i=1}^N \sum_{x \in \mathcal{R}_i} \left[ (T_f - T_0) \text{dist}^2(\bar{y}(x), \mu_i) + \sum_{t=T_0}^{T_f} \text{dist}^2(y(x, t), \bar{y}(x)) \right].\end{aligned}$$

The second term in  $J$  represents the temporal variation of flow speed on one grid point,

which does not change with respect to the partitioning of the flow field. Hence  $\arg \min_{\phi(x), \nu} J = \arg \min_{\phi(x), \nu} J'$ , where  $J'$  is defined in (3.4). Thus the optimal solution of (3.2) equals the optimal solution of (3.4), which implies that the optimal flow partition of the time-varying flow field is equivalent to the optimal flow partition of the time-invariant flow field, computed by taking the time-average of the flow field observations, as described in line 1, Algorithm 1.  $\square$

Next we will prove that under Assumption 3.1.3, the estimated trajectory converges to the actual trajectory, and that the estimated parameter converges to the true value using adaptive control theory. In order to prove convergence, the persistent excitation condition must be demonstrated, and is given below.

**Definition 3.3.2.** [64, 65] *A vector signal  $u$  is persistently exciting if there exist positive constants  $\kappa_1, \kappa_2$ , and  $T$  such that  $\kappa_2 I \geq \int_t^{t+T} u(\tau)u^\top(\tau)d\tau \geq \kappa_1 I \forall t$ .*

Let  $\tilde{\phi}_1 = \begin{bmatrix} \phi^1 & \dots & \phi^N \\ 0 & 0 & 0 \end{bmatrix}$  and  $\tilde{\phi}_2 = \begin{bmatrix} 0 & 0 & 0 \\ \phi_1 & \dots & \phi^N \end{bmatrix}$ . Let  $w = [\tilde{\phi}_1, \tilde{\phi}_2, \Psi_C]^\top \in \mathbb{R}^{(2N+1) \times 2}$ , which is the input signal to (3.12). We can construct a matrix  $W(t) \in \mathbb{R}^{(2N+1) \times (2N+1)}$

as follows

$$W(t) = \int_t^{t+T} \begin{bmatrix} \phi^1 \phi^1 & \phi^1 \phi^2 & \dots & \phi^1 \phi^N & 0 & \dots & 0 & \phi^1 \cos \psi_c \\ \phi^2 \phi^1 & \phi^2 \phi^2 & \dots & \phi^2 \phi^N & 0 & \dots & 0 & \phi^2 \cos \psi_c \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \phi^N \phi^1 & \phi^N \phi^2 & \dots & \phi^N \phi^N & 0 & \dots & 0 & \phi^N \cos \psi_c \\ 0 & 0 & \dots & 0 & \phi^1 \phi^1 & \dots & \phi^1 \phi^N & \phi^1 \sin \psi_c \\ 0 & 0 & \dots & 0 & \phi^2 \phi^1 & \dots & \phi^2 \phi^N & \phi^2 \sin \psi_c \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & \phi^N \phi^1 & \dots & \phi^N \phi^N & \phi^N \sin \psi_c \\ \phi^1 \cos \psi_c & \phi^2 \cos \psi_c & \dots & \phi^N \cos \psi_c & \phi^1 \sin \psi_c & \dots & \phi^N \sin \psi_c & 1 \end{bmatrix} d\tau.$$

The persistent excitation condition is critical to prove the convergence of parameters [66]. When  $W(t)$  is singular the estimation errors of parameters may not converge to zero. The persistent excitation condition requires that the trajectories traveled by the robot to spread over all the partitioned cells, as stated in the following Lemma.

**Lemma 3.3.3.** *The signal vector  $w$  is persistently exciting if the vehicle visits all the partitioned cells.*

*Proof.* Since the partitioned cells do not overlap with each other, for  $\forall \tau, x(\tau)$  can only be in one cell. Hence for all  $i, j \in I_R$ ,

$$\phi^i(x(\tau))\phi^j(x(\tau)) = \mathbb{1}\{i = j\} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}.$$

Thus  $W(t)$  can be simplified to

$$W(t) = \int_t^{t+T} \begin{bmatrix} \phi^1 \phi^1 & & & & & & \phi^1 \cos \psi_c \\ & \ddots & & & & & \vdots \\ & & \phi^N \phi^N & & & & \phi^N \cos \psi_c \\ & & & \phi^1 \phi^1 & & & \phi^1 \sin \psi_c \\ & & & & \ddots & & \vdots \\ & & & & & \phi^N \phi^N & \phi^N \sin \psi_c \\ \phi^1 \cos \psi_c & \dots & \phi^N \cos \psi_c & \phi^1 \sin \psi_c & \dots & \phi^N \sin \psi_c & 1 \end{bmatrix} d\tau.$$

If  $\forall i, \exists \tau \in [t, t+T]$ , s.t.  $\phi^i(x(\tau)) = 1$ , meaning that the vehicle visits all cell during time  $[t, t+T]$ , then  $W(t)$  is full rank, and hence  $w$  is persistently exciting.  $\square$

The persistent excitation condition must be satisfied in order to have the flow parameters of all the cells and vehicle speed estimation converge to the true value. The persistent excitation condition requires the vehicle to visit all the partitioned regions. If this condition is not satisfied, not all flow parameters in the partitioned cells can be accurately estimated. We will further address this condition in the simulation and experimental result section.

The convergence of CLLE is presented as follows.

**Theorem 3.3.4.** *Under the updating law (3.10), CLLE converges to zero when time goes to infinity.*

*Proof.* Consider the following Lyapunov function,

$$V(e, \bar{\xi}, V_L) = \frac{1}{2} \left( e^\top e + \frac{1}{\rho} (\bar{\theta} - \bar{\xi}(t))^\top (\bar{\theta} - \bar{\xi}(t)) + \frac{1}{\rho} (V_R - V_L(t))^2 \right).$$

Since  $e^\top(\theta - \xi(t))\phi(x) = (\bar{\theta} - \bar{\xi}(t))e \otimes \phi(x)$ , the derivative of  $V$  is

$$\begin{aligned}\dot{V} &= \left( -Ke + (\theta - \xi)\phi(x) + (V_R - V_L(t))\Psi_C \right)^\top e + \frac{1}{\rho} \left( -\rho e \otimes \phi(x) \right) (\bar{\theta} - \bar{\xi}) \\ &\quad + \frac{1}{\rho} (V_R - V_L(t)) (-\rho e^\top \Psi_C) \\ &= e^\top Ke \leq 0.\end{aligned}$$

$\dot{V}$  is negative semi-definite, which implies that  $e, \bar{\xi}, V_L(t)$  are bounded. In addition, the second order time derivative of  $V$  is

$$\ddot{V} = -2e^\top K \left( (\theta - \xi(t))\phi(x) + (V_R - V_L(t))\Psi_C - Ke \right).$$

Thus  $\ddot{V}$  bounded, and hence  $\dot{V}$  is uniformly continuous. Therefore  $\lim_{t \rightarrow \infty} \dot{V}(t) = 0$ . Since  $K$  is a diagonal matrix,  $e(t) \rightarrow 0$  as  $t \rightarrow \infty$ .  $\square$

**Theorem 3.3.5.** *Under the updating law (3.10), if the vehicle visits all the partitioned cells,  $\bar{\xi}(t)$  and  $V_L(t)$  converges to  $\bar{\theta}$  and  $V_R$  respectively as time goes to infinity.*

*Proof.* Let  $\eta_1 = \theta_1 - \xi_1(t), \eta_2 = \theta_2 - \xi_2(t), \eta_3 = V_R - V_L(t)$ , then the CLLE dynamics can be written as

$$\dot{e} = \tilde{\phi}_1(x)\eta_1 + \tilde{\phi}_2(x)\eta_2 + \eta_3\Psi_C - Ke.$$

We define a new state variable  $X = [e^\top, \eta_1^\top, \eta_2^\top, \eta_3]^\top$ , and an output variable  $Y = e$ , then the

dynamics for the state variable and the output variable satisfy

$$\dot{X} = A(t)X, Y = CX,$$

$$\text{where } A(t) = \begin{bmatrix} -K & \tilde{\phi}_1 & \tilde{\phi}_2 & \Psi_C \\ -\rho\tilde{\phi}_1 & 0 & 0 & 0 \\ -\rho\tilde{\phi}_2 & 0 & 0 & 0 \\ -\rho\Psi_C & 0 & 0 & 0 \end{bmatrix}, C = \begin{bmatrix} I & 0 & 0 & 0 \end{bmatrix}.$$

Our goal is to show that the origin of  $\dot{X} = A(t)X$  is uniformly asymptotically stable, which indicates that  $\bar{\xi}$  converges to  $\bar{\theta}$ , and  $V_L(t)$  converges to  $V_R$ . Let

$$P = \begin{bmatrix} \frac{1}{2}K^{-1} & 0 & 0 & 0 \\ 0 & \frac{1}{2\rho}K^{-1} & 0 & 0 \\ 0 & 0 & \frac{1}{2\rho}K^{-1} & 0 \\ 0 & 0 & 0 & \frac{1}{2\rho}K^{-1} \end{bmatrix}.$$

There exists some  $c_1, c_2$  such that  $c_1I \leq P \leq c_2I$ , and there exists some constant  $0 < \nu < 1$  such that

$$A(t)^\top P + PA(t) + \dot{P} + \nu C^\top C = (1 - \nu) \begin{bmatrix} -I & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix},$$

which is negative semi-definite.

Then by the Lyapunov theorem (Theorem 3.8.4 in [67]),  $\dot{X} = A(t)X$  is uniformly asymptotically stable if we can prove that  $(C, A)$  is uniformly completely observable. First, we will find a bounded matrix  $L$ , and show that  $(C, A + LC)$  is uniformly completely observable. Then, this will lead to the conclusion that  $(C, A)$  is uniformly completely

observable.

Let  $L = \begin{bmatrix} 0 \\ \rho\tilde{\phi}_1 \\ \rho\tilde{\phi}_2 \\ \rho\Psi_C^\top \end{bmatrix}$ . Since  $\Psi_C$  is uniformly bounded, and all elements in  $\tilde{\phi}$  is either 0 or 1,

$L$  is uniformly bounded, and

$$A + LC = \begin{bmatrix} -K & \tilde{\phi}_1 & \tilde{\phi}_2 & \Psi_C \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

Thus, we now consider the observability of

$$\begin{aligned} \dot{X} &= (A + LC)X \\ Y &= CX. \end{aligned} \tag{3.11}$$

Let  $\eta = [\eta_1, \eta_2, \eta_3]^\top$ , then the system (3.11) has the following form:

$$\begin{aligned} \dot{e} &= -Ke + w^\top \eta \\ \dot{\eta} &= 0 \\ Y &= e \end{aligned} \tag{3.12}$$

Due to the assumption that the vehicle visits all cells, by Lemma 3.3.3,  $w$  is persistently exciting. Let  $\Phi(\tau) = \int_t^\tau e^{-K(\tau-\sigma)} w(\sigma) d\sigma$  be the output of (3.12) given input  $w$ . Then  $\Phi(\tau)$  satisfies persistently exciting conditions because  $w(\sigma)$  is persistently exciting, and the transfer function of (3.12),  $(sI + K)^{-1}$  is stable, minimum phase, proper rational transfer function. Therefore, there exists constant  $\kappa_1, \kappa_2, T_0 > 0$  such that  $\kappa_2 I \geq \frac{1}{T_0} \int_t^{t+T_0} \Phi(\tau) \Phi(\tau)^\top d\tau \geq \kappa_1 I, \forall t \geq 0$ . By applying Lemma 4.8.4 in [67], we can conclude that the system of (3.12) is uniformly completely observable. In other words, we have proved that  $(C, A + LC)$  is

uniformly completely observable. By applying Lemma 4.8.1 in [67] the system  $(C, A)$  is uniformly completely observable.

Therefore, the origin of  $\dot{X} = AX$  is uniformly asymptotically stable, that is,  $X \rightarrow 0$  as  $t \rightarrow \infty$ . This means that  $\eta_1, \eta_2$  and  $\eta_3$  go to zeros, individually. Thus  $\bar{\xi}$  and  $V_L(t)$  converges to  $\bar{\theta}$  and  $V_R$ , respectively.  $\square$

### 3.4 Simulation results

In this section, we provide the results of the implementation of our flow field modeling and path planning methods in a simulated experiment. First, we describe the simulated experimental set-up and recent field experiments, which serve as a strong test of the methods due to the magnitude and variability of the flow. We validate the proposed flow modeling algorithm by comparing the estimated flow model parameters generated from the proposed flow estimation algorithm with the glider estimated flow collected during the experiment. Based on the estimated flow model, simulation of the bounded cost path planning algorithm is performed, and its performance is compared with other AUV path planning algorithms.

#### 3.4.1 Experimental and simulation setup

Our study is motivated by the use of underwater gliders off the coast near Cape Hatteras, North Carolina, US as part of a 16-month experiment (Processes driving Exchange At Cape Hatteras, PEACH) to study the processes that cause exchange between the coastal and deep ocean at Cape Hatteras, a highly dynamic region characterized by confluent western boundary currents and convergence in the adjacent shelf and slope waters. Underwater gliders, AUVs that change their buoyancy and center of mass to “fly” in a sawtooth-shaped pattern, were deployed on the shelf and shelf edge to capture variability in the position of the Hatteras Front, the boundary between cool, fresh water on the shelf of the Mid Atlantic Bight and the warmer, saltier water observed in the South Atlantic Bight.

We deployed one glider off Oregon Inlet, NC May 16, 2017 and recovered it 14 days

later. For its mission, the glider was initially tasked to sample along a path with offshore, inshore, and triangular segments designed to sample critical flow features (Fig. 3.1 (Left)), and was not used with a thruster. The glider surfaced approximately every 4 hours to update its position with a GPS fix, communicate with shore, transmit a subset of data, and most importantly, receive mission updates and commands to adapt sampling.

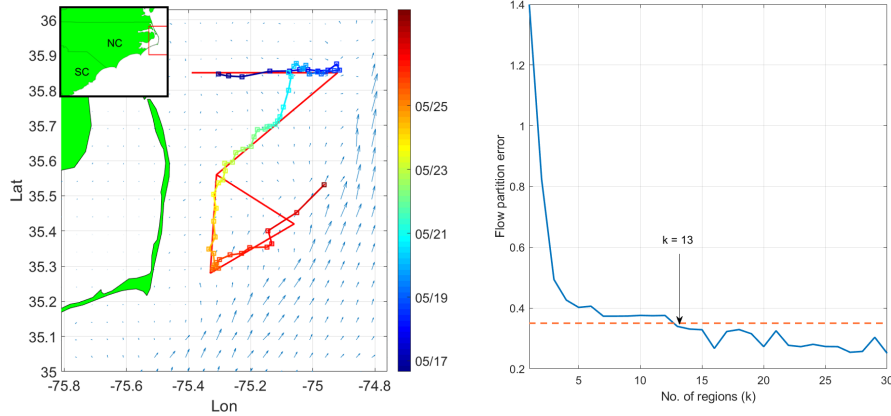


Figure 3.1: **(Left)**: Survey domain near Cape Hatteras. The curve represents glider trajectory during the first PEACH deployment. The red line path is the pre-assigned sampling pattern. Squares denote the glider surfacing positions along trajectory, and color of the trajectory depicts timestamps. The arrows represent the NCOM-predicted flow field at the starting time of the deployment. **(Right)**: Flow partition error when the number of cells is set as  $k = 1, \dots, 30$ .

### 3.4.2 Flow modeling using glider experimental data

In this example, we present flow modeling results using the proposed flow partition and parameter estimation methods. The flow map forecast is given by a 1-km horizontal resolution version of the Navy Coastal Ocean Model (NCOM, [68]), made available by J. Book and J. Osborne (Naval Research Laboratory, Stennis Space Center, US). In the domain of interest, the ocean model flow forecast is given at  $106 \times 106$  rectangular grid points. Tidal flow accounts for much of the short-term ( $< 24$  hour) temporal variation of the flow field. Hence the partition time interval is taken over multiple periods of the largest tidal constituent, the lunar semidiurnal  $M_2$  tide (period 12.42 hours). Maximum flow speed in

this area is 2.2788 m/s, approximately 7.5 times the vehicle speed, and 4.5 times the speed of a hybrid glider using a thruster. We set the upper bound for flow partition error to be 0.35 m/s, which is about 15% of the maximum flow speed in the domain. Figure 3.1 (Right) describes the flow partition error in the case of different selection of cell number. Since the flow partition error goes below the upper bound when  $k = 13$ , the number of cells is chosen as 13. We smooth the cell boundaries into straight lines using the Least Mean Square method. Even though smoothing the cell boundaries might overlook more detailed spatial variability of the flow field, it helps to reduce the computational cost of solving the planning problem. The partitioned flow field is shown in Figure 3.2 (Left). Comparing Figures 3.2 (Left) and 3.1 (Left), it can be seen that the proposed algorithm captures the major spatial variation of the flow field, by separating the high speed flow regions from the area where the flow is at lower speed.

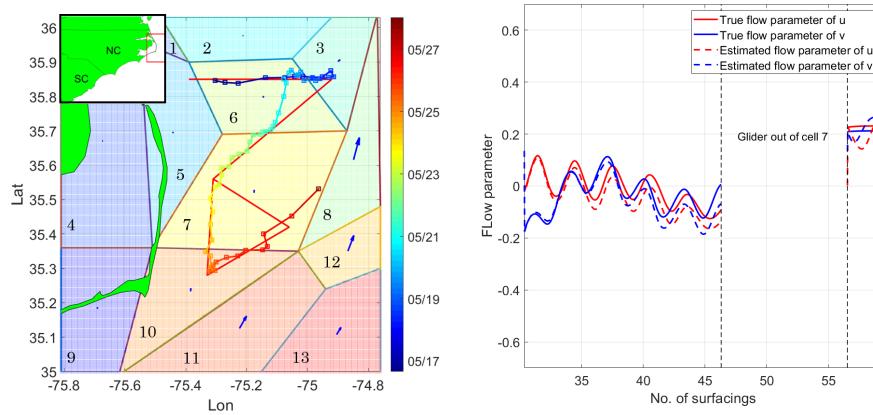


Figure 3.2: **(Left):** Partitioned cells of the survey domain. The polygons are the partitioned regions. The blue arrows represent uniform flow speed in each of the cells generated from the proposed algorithm. **(Right):** Estimated flow parameters and the ground truth value in cell 7.

At each surfacing, the vehicle position is given by the GPS location. When the glider is underwater, we use linear interpolation to estimate the heading and vehicle position. The vehicle's forward speed is zero when it is at the surface of the ocean, and the vehicle drifts freely with the surface current. This violates the constant forward speed assumption stated

in Assumption 3.1.1. Hence, we remove the segment of data when the vehicle is drifting at surface, and then compute the estimated flow parameters by the proposed algorithm. Glider speed is initialized to be 0.3 m/s, while the flow parameters are initialized by the flow vectors found by partitioning the NCOM data. Since the vehicle trajectory crosses cell 3, 6, 7, 10, and does not enter other cells, the glider trajectory does not satisfy persistent excitation condition described in Lemma 3.3.3. Hence only the flow parameters in cells 3,6,7,10 can be updated by the adaptive updating law, while the flow parameters in the rest cells remain to be the initial value. To justify performance of the proposed flow parameter estimation algorithm, we use the ADCIRC (Advanced Circulation) model output [69] to model the tidal flow component, and derive the non-tidal glider estimated flow speed by subtracting ADCIRC reported flow from the flow parameter estimate. The de-tided glider estimated flow speed is considered as the ground truth of flow parameters in the corresponding cells. The root mean square error (rmse) between the estimated parameter and the ground truth in cell 3, 6, 7, 10 is shown in Table 3.1. It can be seen that in all of the four cells, the estimated flow parameters is in good agreement with the true flow parameters. The rmse in all of the four cells is within 5% of the maximum flow speed in the domain. Figure 3.2 (Right) shows the comparison between the estimated flow parameters and the true flow parameter value in one of the cells that the glider trajectory visits. It is shown that in cell 7, the estimated flow parameter matches well with the true value.

Table 3.1: Root mean square error between the estimated and the true flow parameters.

Cell number	Flow parameter	Estimation error (m/s)
Cell 3	W-E flow	0.0558
	N-S flow	0.0082
Cell 6	W-E flow	0.0526
	N-S flow	0.0831
Cell 7	W-E flow	0.0415
	N-S flow	0.0388
Cell 10	W-E flow	0.0961
	N-S flow	0.0975

## CHAPTER 4

### PATH PLANNING IN REDUCED-ORDER FLOW FIELD: THE METHOD OF EVOLVING JUNCTIONS

In this Chapter, we present a novel method for computing time and energy optimal paths for vehicles traveling in the partitioned flow field as proposed in Chapter 3. Different from existing strategies that solves the MIP using a bi-level approach [9, 10, 11], we solve the MIP in an interleaved way, with guaranteed optimality. When a cell sequence connecting the start and the destination is found by the depth-first search (DFS), the intersection position between the optimal path and the cell boundary curves is computed by the intermittent diffusion (ID) method [70]. For large search tree with high branching factor, DFS has shown to be computationally impractical [71]. To address this challenge, we propose a novel branch-and-bound (BnB) DFS method that reduces the number of nodes to be searched in the decision tree, and hence reduces the computational cost in solving the problem. The key insight is that we can find a lower bound of the the stage cost induced from traveling in one partitioned cell. Leveraging the lower bound, we compute a lower bound on the cost-of-arrival, and stop the DFS at the current node if lower bound on its cost-of-arrival exceeds the best solution found so far. Therefore, this work proposes a novel method to solve the AUV path planning problem, as an instance of MIP, without excessive demand on computing resources. We evaluate the performance of the proposed algorithm through simulation of AUV path planning in both simulated and realistic ocean flow fields, and compare its computational cost and solution quality with the LSM and A\* method. The proposed method achieves comparable path cost, and has significantly lower computation cost compared with the LSM and A\* method. Moreover, we provide time complexity analysis of the proposed algorithm, to provide insight on its computation cost with respect to different level of partitioning.

## 4.1 Problem formulation

We consider the vehicle moving in the space  $\Omega \subset \mathbb{R}^d$  with dimension  $d$ , equipped with a dynamic  $\dot{x} = u + v$ , where  $u$  is the environment flow velocity, and  $v$  is the vehicle velocity or the control variable, satisfying  $v \in \mathcal{U}$ , providing that  $\mathcal{U}$  is a compact space such that  $\|v\| \leq V$ , where  $V$  is the max speed of the vehicle. By taking advantage of the partitioning method proposed in Chapter 3, the flow field is divided into finite number of convex regions  $\{R_\alpha\}_{\alpha \in I_R}$  by boundary curves (surfaces if is in  $\mathbb{R}^d, d \geq 3$ )  $\{f_{\alpha\beta}\}_{(\alpha,\beta) \in I}$ . Here

$$I = \{(\alpha, \beta) \in I_R \times I_R : \dim(\partial R_\alpha \cap \partial R_\beta) = d - 1\},$$

where  $\dim(S)$  returns the dimension of the set  $S$  and  $\partial S$  is the boundary of  $S$ , and  $f_{\alpha\beta}(x) = 0$  is the  $(d - 1)$ -dimensional compact boundary of the region  $R_\alpha$  and  $R_\beta$  (can be denoted as  $\partial R_\alpha$  and  $\partial R_\beta$  respectively). Let  $x$  denote a point on the boundary, we can parameterize the cell boundary by defining a piece-wise diffeomorphism

$$x(\lambda) : D \subset \mathbb{R}^{d-1} \longrightarrow \{y : f_{\alpha\beta}(y) = 0\},$$

where  $D$  is a  $(d - 1)$ -dimensional unit ball. Also within each region, we suppose that the flow velocity  $u$  is a constant vector. Hence we can denote the flow velocity in each region  $R_\alpha$  separately by  $u_\alpha$ . The vehicle needs to be controlled from an initial position  $x_0$ , crossing different regions and finally reaching the target position  $x_f$ .

Since there could be infinitely many feasible paths linking  $x_0$  and  $x_f$ , a cost function is introduced to measure the travel expense with respect to different potential trajectories. We denote the cost function to be

$$J(v, T) = \int_0^T L(v(t)) dt, \quad (4.1)$$

letting  $\gamma(t)$  be a continuous path with  $\gamma(0) = x_0$ ,  $\gamma(T) = x_f$  and  $\dot{\gamma}(t) = u + v$ . In this paper, we discuss the problem with the cost function specifically being the total travel time, that is  $L(x, v) = 1$  and the kinetic energy combining a constant running cost, in which case  $L(x, v) = \|v\|^2 + C$  where  $C$  and  $\|v\|$  are not simultaneously 0 for all  $x \in \Omega$ . Our goal is to find the optimal control function  $v(t)$  with minimum cost, and can be expressed as the following problem:

$$\begin{aligned}
& \min_{v, T} \int_0^T L(v) dt \\
& s.t. \quad \dot{x} = v + u, \\
& \quad \quad x(0) = x_0, \\
& \quad \quad x(T) = x_f, \\
& \quad \quad \max_{t \in [0, T]} \|v\| \leq V
\end{aligned} \tag{4.2}$$

In the next section, we will concretely discuss our method, the idea of which is to seek a way to change the infinite dimensional optimal control problem into a finite dimensional optimization problem.

## 4.2 Our method

All possible trajectories will continuously pass through a sequence of regions. Since the flow velocity  $u$  is a constant vector in each region, we have the following theorem that will be proved in the Appendix:

**Theorem 4.2.1.** *In the optimal solution for*

$$\begin{aligned} & \min_{v,T} \int_0^T L(v) dt \\ \text{s.t. } & \dot{x} = f(v + u), \\ & x(0) = x_0, \\ & x(T) = x_f, \\ & \max_{t \in [0,T]} \|v\| \leq V. \end{aligned}$$

where  $u$  is a constant vector and  $f$  is invertible, the velocity  $v$  is a constant vector.

With Bellman's principle, it is the fact that the optimal path should be formed via a piece-wise constant velocity motion. Thus, we restrict the velocity of the vehicle in each region to be a constant. Further, the regions are convex, therefore, the straight line path always lies inside each of the cell crossed and the path is continuous. Now we can introduce the notation  $v_i$  to be the vehicle velocity in the  $i^{\text{th}}$  cell crossed.

Because of Theorem (4.2.1), we can parameterize solution to (4.2) by i) the cell sequence that the path goes through, and ii) the position where the path intersects with the cell boundaries. We assume that the initial and the terminal position in (4.2) are on the boundary of the partitioned cells. In general this can be achieved by incorporating a fixed rule to partition the cells containing  $x_0$  and  $x_f$  into two separate cells, with the new generated boundary going through  $x_0$  and  $x_f$ .

Let  $\mathcal{C} = \{c_1, c_2, \dots, c_n\}$  denote a sequence of the index of cells that a path travels through,  $c_i \in I_R, \forall i \in [1, n]$ . We define the junction set  $\{x_i\}_{i=0}^n$  to be the intersections between a trajectory and the cell boundaries. The junction  $x_i$  is on the boundary between the two regions indexed as  $c_i$  and  $c_{i+1}$ . The junction  $x_0$  is the initial junction and the destination  $x_f$  is the last junction e.g.  $x_n = x_f$ . For each region  $R_i$ , the entrance junction is  $x_{i-1}$  and the exit junction is  $x_i$ . Fig. 4.1 shows illustration of parameterizing a feasible path using junctions and the cell sequence.

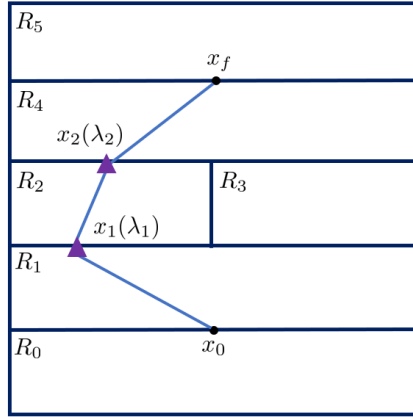


Figure 4.1: Example of parameterizing a feasible path in the partitioned domain by the cell sequence and the junction position. The purple triangles are the junction position. The path goes through 3 cells. Index of 3 cells are  $c_1 = 1, c_2 = 2, c_3 = 4$ . On the boundary  $f_{12}$  and  $f_{24}$  there are two junctions, position of which is parameterized by  $\lambda_1$  and  $\lambda_2$ .

Given the above mentioned parameterization to feasible trajectories, to solve the planning problem, we need to i) determine the sequence of regions that the optimal path goes through, and ii) compute the optimal position of intersection position between the trajectory and the cell boundaries. In this section, we propose a novel method to solve the planning problem. The key idea of the method is based on two insights. First, we only need to determine the cell sequence when the vehicle reaches cell boundary. Each assignment of visited cell sequence will produce a branch of a decision tree, with each branch in the tree representing a cell that the vehicle crosses. The decision tree contains a finite number of branches, and a feasible solution must be a path from the root of the tree to the target position. Second, the optimal junction position can be easily computed if we fix the assignment of cell sequence.

#### 4.2.1 Minimize total travel time

When minimizing total travel time is the goal, we further let the vehicle move in maximum speed  $V$ . Let  $u_{c_i}$  denote the flow speed in the cell indexed by  $c_i$ . Then the cost function can

be converted in the format below:

$$J(v, T, \mathcal{C}) = \int_0^T ds = T = \sum_{i=1}^n g_i^t(x_i, x_{i-1}, c_i),$$

where  $g_i^t$  is the travel time in region  $R_{c_i}$  and is expressed as:

$$g_i^t = \frac{\|x_i - x_{i-1}\|}{\|v_i + u_{c_i}\|}.$$

Since flow and vehicle velocities are constant in each cell, the motion must be in straight

line, meaning that  $\frac{x_i - x_{i-1}}{\|x_i - x_{i-1}\|} = \frac{v_i + u_{c_i}}{\|v_i + u_{c_i}\|}$ , and it leads to:

$$v_i + u_{c_i} = \frac{x_i - x_{i-1}}{\|x_i - x_{i-1}\|} \|v_i + u_{c_i}\|. \quad (4.3)$$

We start from the following equality:

$$(v_i + u_{c_i})^T [(u_{c_i} + v_i) - 2u_{c_i} + (u_{c_i} - v_i)] = 0,$$

and inserting (4.3) into the above equation yields:

$$\|v_i + u_{c_i}\|^2 - \frac{2(x_i - x_{i-1})^T u_{c_i}}{\|x_i - x_{i-1}\|} \|v_i + u_{c_i}\| + \|u_{c_i}\|^2 - V^2 = 0.$$

Solving the above equation gives us the following results:

$$\|v_i + u_{c_i}\| = \frac{(x_i - x_{i-1})^T u_{c_i}}{\|x_i - x_{i-1}\|} \pm \left( \left( \frac{(x_i - x_{i-1})^T u_{c_i}}{\|x_i - x_{i-1}\|} \right)^2 + V^2 - \|u_{c_i}\|^2 \right)^{\frac{1}{2}}.$$

To minimize the travel time, we take the plus sign so that

$$\|v_i + u_{c_i}\| = \frac{(x_i - x_{i-1})^T u_{c_i}}{\|x_i - x_{i-1}\|} + \left( \left( \frac{(x_i - x_{i-1})^T u_{c_i}}{\|x_i - x_{i-1}\|} \right)^2 + V^2 - \|u_{c_i}\|^2 \right)^{\frac{1}{2}} \quad (4.4)$$

and have the travel time in region  $R_{c_i}$  as

$$g_i^t = \frac{1}{\|u_{c_i}\|^2 - V^2} \left( (x_i - x_{i-1})^T u_{c_i} - \sqrt{((x_i - x_{i-1})^T u_{c_i})^2 + \|x_i - x_{i-1}\|^2 (V^2 - \|u_{c_i}\|^2)} \right). \quad (4.5)$$

At last, since  $x_i$  is on the boundary of the regions indexed by  $c_i$  and  $c_{i+1}$ , we have the smooth parameterization of each  $x_i$  as  $x_i = x_i(\lambda_i)$  where  $\lambda_i \in D \subset \mathbb{R}^{d-1}$ , transforming finally the cost function to be

$$J(\lambda_1, \dots, \lambda_{n-1}, \mathcal{C}) = g_1^t(x_1(\lambda_1), x_0, c_1) + g_n^t(x_{n-1}(\lambda_{n-1}), x_f, c_n) + \sum_{i=2}^{n-1} g_i^t(x_i(\lambda_i), x_{i-1}(\lambda_{i-1}), c_i).$$

and the problem is changed to a finite dimensional optimization problem

$$\min_{\lambda_i \in D, c_i \in I_R, i=1, \dots, n-1} J(\lambda_1, \dots, \lambda_{n-1}, \mathcal{C}). \quad (4.6)$$

#### 4.2.2 Minimize energy

If the cost function is the kinetic energy with a constant running cost  $C \geq 0$

$$J(v, T) = \int_0^T \|v\|^2 + C dt,$$

we first consider the constant velocity motion in each region  $R_i$  within time  $t_i$ . Fixing the entrance and exit junctions  $x_{i-1}$  and  $x_i$ , we can derive vehicle speed as

$$v_i = \frac{x_i - x_{i-1}}{t_i} - u_{c_i},$$

and

$$\|v_i\|^2 = \frac{\|x_i - x_{i-1}\|^2}{t_i^2} + \|u_{c_i}\|^2 - \frac{2(x_i - x_{i-1})^T u_{c_i}}{t_i}. \quad (4.7)$$

the cost function in the specific region  $R_{c_i}$  can be rewritten as

$$\begin{aligned}
J(t_i) &= \int_0^{t_i} \|v_i\|^2 + C ds \\
&= \frac{\|x_i - x_{i-1}\|^2}{t_i} + (\|u_{c_i}\|^2 + C)t_i - 2(x_i - x_{i-1})^T u_{c_i} \\
&\geq 2\sqrt{\|u_{c_i}\|^2 + C}\|x_i - x_{i-1}\| - 2(x_i - x_{i-1})^T u_{c_i}
\end{aligned} \tag{4.8}$$

with equality holds at

$$t_{i,1}^* = \frac{\|x_i - x_{i-1}\|}{\sqrt{\|u_{c_i}\|^2 + C}}. \tag{4.9}$$

Hence, if the maximum vehicle speed is large enough, that is,

$$V^2 \geq \frac{\|x_i - x_{i-1}\|^2}{t_{i,1}^{*2}} + \|u_{c_i}\|^2 - \frac{2(x_i - x_{i-1})^T u_{c_i}}{t_{i,1}^*}, \tag{4.10}$$

we can find the optimal vehicle forward speed by replacing  $t_i$  in (4.7) with the optimal solution in (4.9):

$$\|v_i\|^2 = 2\|u_{c_i}\|^2 + C - \frac{2(x_i - x_{i-1})^T u_{c_i}}{\|x_i - x_{i-1}\|} \sqrt{\|u_{c_i}\|^2 + C}. \tag{4.11}$$

Hence, if (4.10) holds, we let the vehicle move in the speed of  $\|v_i\|$  in (4.11).

However, if the (4.10) does not hold, we set the vehicle speed to be the maximum speed  $V$  and the cost function is reduced to be  $J = (V^2 + C)t_{i,2}^*$  where

$$\begin{aligned}
t_{i,2}^* &= \frac{1}{\|u_{c_i}\|^2 - V^2} \left( (x_i - x_{i-1})^T u_{c_i} - \left( ((x_i - x_{i-1})^T u_{c_i})^2 \right. \right. \\
&\quad \left. \left. + \|x_i - x_{i-1}\|^2 (V^2 - \|u_{c_i}\|^2) \right)^{\frac{1}{2}} \right),
\end{aligned} \tag{4.12}$$

which is a root of

$$(\|u_{c_i}\|^2 - V^2)t_i^2 - 2(x_i - x_{i-1})^T u_{c_i} t_i + \|x_i - x_{i-1}\|^2 = 0.$$

Thus the energy spent in each region is

$$g_i^e(x_i, x_{i-1}) = \begin{cases} 2\sqrt{\|u_{c_i}\|^2 + C}\|x_i - x_{i-1}\| - 2(x_i - x_{i-1})^T u_i & \text{if (4.10) holds} \\ (V^2 + C)t_{i,2}^* & \text{otherwise} \end{cases}$$

By using the same parametrization as in the time-optimal planning, we finally have the problem to be a finite dimensional optimization formulated as

$$\min_{\lambda_i \in D, c_i \in I_R, i=1, \dots, n-1} J(\lambda_1, \dots, \lambda_{n-1}, \mathcal{C}) \quad (4.13)$$

where

$$\begin{aligned} J(\lambda_1, \dots, \lambda_{n-1}, \mathcal{C}) = & g_1^e(x_1(\lambda_1), x_0, c_1) + g_n^e(x_{n-1}(\lambda_{n-1}), x_f, c_n) \\ & + \sum_{i=2}^{n-1} g_i^e(x_i(\lambda_i), x_{i-1}(\lambda_{i-1}), c_i). \end{aligned}$$

Furthermore,  $g_i^t$  and  $g_i^e$  has the following properties:

**Proposition 4.2.1.** *If there exists a feasible trajectory from  $x_{i-1}$  to  $x_i$  in  $R_i$  and  $V \neq \|u_i\|$ , then  $g_i^t(x_i, x_{i-1}, c_i)$  and  $g_i^e(x_i, x_{i-1}, c_i)$  are differentiable.*

With this proposition, we can take the derivative of the objective function, which is pivotal for applying the Intermittent Diffusion method described in Section 4.2.3.

### 4.2.3 Construction of decision tree

We introduce a tree structured graph to model how the sequence of cell that the path crosses affects the total cost. In the tree, each node represents a boundary curve that contains a junction point. To construct the decision tree, we first define two boundary curves  $f_{c_1, c_2}, f_{c_3, c_4}$  as adjacent if

$$\{c_1, c_2\} \cap \{c_3, c_4\} \neq \emptyset, \quad (4.14)$$

indicating that the two curves are two boundaries of the same cell.

### *Decision tree traversal*

Starting from the boundary curve containing the initial position, which is the root node, a directed decision tree can be formed by connecting the adjacent boundary curves in the domain. The branch generation stops when the node is the boundary curve containing the destination position. Each node in the decision tree represents a boundary curve that a feasible path will go through. A connected path in the decision tree, starting from the root node to the target node represent a sequence of cells that a feasible path will cross. Fig. 4.2 shows the decision tree constructed from a partitioned workspace.

The construction of the entire decision tree is not necessary, and is time-consuming. However, for the purpose of clearly presenting the concept of the planning method, we will discuss how the tree can be fully constructed, and then present the branch pruning technique. We construct the decision tree iteratively using the depth-first search method. Let  $n_c$  represent the current node. At each step, we search for all the boundary curves adjacent to the current node, and call the current node as the predecessor of the new node. Since the optimal path will not visit one boundary curve more than one time, the optimal cell sequence should not include loops. Hence, when searching for the adjacent nodes to be expanded next, we will not expand an adjacent node of  $n_c$  if it is already visited. This node generation process terminates when the target node is visited.

By constructing the decision tree we can find all the cell sequences connecting the root and the terminal node. For one cell sequence, the MIP (4.6) and (4.13) reduces to a finite dimensional non-convex optimization problem over the junction positions. Next we show how this optimization problem can be solved using the Intermittent Diffusion method.

### *Intermittent Diffusion*

The objective functions in (4.6) and (4.13) are both differentiable. Hence we use the Intermittent Diffusion (ID) to get the global minimizer [72], the key idea of which is adding white noise to the gradient flow intermittently. Namely, we solve the following stochastic

differential equation (SDE) on the configuration space

$$d\lambda = -\nabla J(\lambda)d\theta + \sigma(\theta)dW(\theta) \quad (4.15)$$

where  $\lambda = (\lambda_1, \dots, \lambda_{n-1}) \in D^{n-1}$  and  $W(\theta)$  is the standard Brownian motion. The diffusion is a piece-wise constant function defined by

$$\sigma(\theta) = \sum_{i=1}^N \sigma_i I_{[S_i, T_i]}(\theta)$$

where  $\sigma_i$  are constant and  $I_{[S_i, T_i]}(\theta)$  is the characteristic function on interval  $[S_i, T_i]$  with  $0 \leq S_1 < T_1 < S_2 < T_2 < \dots < S_N < T_N < S_{N+1} = T$ .

Thus, if  $\sigma(\theta) = 0$ , we obtain the gradient flow back while when  $\sigma(\theta) \neq 0$ , the solution of (4.15) has positive probability to escape the current local minimizer. The theory of ID indicates that the solution of (4.15) visits the global minimizer of  $J$  with probability arbitrarily close to 1 if  $\min_i |T_i - S_i|$  is large enough, which is guaranteed by analysis in [72].

We use the forward Euler discretization to discretize the above SDE and get

$$\lambda^{k+1} = \lambda^k - h\nabla J(\lambda^k) + \sigma_k \xi^k \sqrt{h}. \quad (4.16)$$

The constant  $h$  is the step size,  $\sigma_k$  is the coefficient chosen to add the intermittent perturbation and  $\xi^k \sim \mathcal{N}(0, 1)$  is a Gaussian random variable. In practice, the global minimizer can be reached by tuning the white noise strength  $\sigma_k$  as well as setting the total evolution round  $N$  long enough.

We summarize the ID algorithm in Algorithm 4 with the objective function being (4.6) or (4.13).

*Branch cost lower bound*

We leverage a BnB technique to prune the branches in the decision tree, in order to save the computation cost in both node generation and solving for the optimal junction position. We approach this problem by leveraging the lower bound of the decision tree branch cost. In the decision tree, each branch represents a path segment connecting the junction on one boundary to the junction on one of its adjacent boundary. Hence, the branch cost of the decision tree represents the stage cost generated from going from one junction to another. However, since the optimal junction position is unknown when we construct the decision tree, the exact optimal branch cost cannot be calculated. Hence we introduce the lower bound of the branch cost, and use the DFSBnB technique [73] to prune some of the branches in the tree. For the time-optimal and energy-optimal planning, we find the upper bound of the branch cost as follows. Let us define the maximum and minimum distance between two junctions  $x_{i-1}$  and  $x_i$ ,

$$\begin{aligned}
 d_{\max} &= \max_{x_i, x_{i-1}} \|x_i - x_{i-1}\|, \\
 \text{s.t. } & f_{c_i, c_{i+1}}(x_i) = 0, f_{c_{i-1}, c_i}(x_{i-1}) = 0, \\
 d_{\min} &= \min_{x_i, x_{i-1}} \|x_i - x_{i-1}\|, \\
 \text{s.t. } & f_{c_i, c_{i+1}}(x_i) = 0, f_{c_{i-1}, c_i}(x_{i-1}) = 0.
 \end{aligned}$$

We can find a lower bound on the minimum time spent on the path segment,

$$\begin{aligned}
 g_i^t &\geq \frac{1}{V^2 - \|u_{c_i}\|^2} (\|x_i - x_{i-1}\| V \\
 &\quad - \sqrt{\|x_i - x_{i-1}\|^2 \|u_{c_i}\|^2 - ((x_i - x_{i-1})^T u_{c_i})^2} - (x_i - x_{i-1})^T u_{c_i}) \\
 &\geq \frac{1}{V^2 - \|u_{c_i}\|^2} (d_{\min} V - \|(x_i - x_{i-1})\| \|u_{c_i}\| + \|(x_i - x_{i-1})^T u_{c_i}\| - (x_i - x_{i-1})^T u_{c_i}) \\
 &\geq \frac{d_{\min}}{V + \|u_{c_i}\|} \triangleq g_{i,lb}^t.
 \end{aligned} \tag{4.17}$$

This lower bound is the travel time when the vehicle travels in the largest possible total speed  $V + \|u_{c_i}\|$ , which is the situation that  $u_{c_i}$  is in the same direction as the shortest path segment from  $x_{i-1}$  to  $x_i$ .

Similarly, given (4.8) we find a lower bound, denoted as  $g_{i,lb}^e$  on the energy spent in one cell,

$$g_{i,lb}^e = \max\{2d_{\min}\sqrt{\|u_{c_i}^2 + C\|} - 2d_{\max}\|u_{c_i}\|, 0\}. \quad (4.18)$$

### *Branch-and-Bound method*

We use the DFS algorithm to iteratively generate the nodes, starting from the root and terminates when it reaches the destination node. After the first cell sequence connecting the root with the destination node is found, we use ID algorithm to compute the optimal junctions that result in the minimum total travel cost. To avoid traversing all feasible cell sequences, the algorithm maintains the lowest-cost path to the target found so far, and its cost. At each step of node generation, for each of the adjacent cell  $m_i$  of the current node  $n_c$ , we compute a lower bound of the total cost of arrival, from the root to  $v_i$ ,

$$f_g(m_i) = f_g(n_c) + g_{i,lb}, \quad (4.19)$$

where for the time-optimal planning,  $g_{i,lb} = g_{i,lb}^t$  is computed by (4.17), and for the energy-optimal planning,  $g_{i,lb} = g_{i,lb}^e$  is computed by (4.18). If  $f_g(m_i)$  is larger than the lowest-cost path found so far, then all the path that goes through the cell boundary represented by  $m_i$  cannot be the optimal solution, since its total cost will be larger than the lowest-cost path found so far. Thus we stop the DFS from  $m_i$  to its child nodes, and go to the next adjacent cell of  $n_c$  to continue the search. The algorithm is presented in Alg. 3.

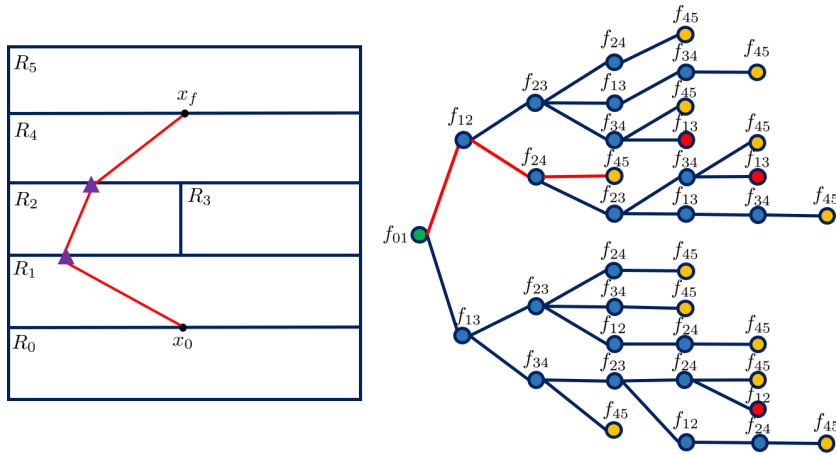


Figure 4.2: **(Left):** Example of a partitioned space. The domain is partitioned into 6 cells, the red line represents one feasible path from the start to the destination position, with the junctions represented by the purple triangles. **(Right):** The entire decision tree for this partitioned space. The root node shown by the green circle is the boundary curve containing the starting position, and the terminal node (yellow circle) represents the boundary curve containing the destination node. For each node, it is connected to its child node if it is an adjacent boundary to its child. The red nodes represent the nodes with no unvisited neighboring nodes. The red path in the graph corresponds to the cell sequence crossed by the feasible path shown on the left figure.

#### 4.2.4 Complexity analysis

In this subsection, we conduct the time complexity analysis of the proposed algorithm, and compare it with other graph search algorithms, such as the A\* method. Let us assume that the flow field forecast is available on  $N \times N$  grid points in the domain, and the OPEN set of A\* algorithm is implemented using a heap data structure. Then each iteration of A\* node expansion has a worst case running time of  $\mathcal{O}(\log mN^2)$ , where  $m$  is the number of neighbors to be expanded for each node, and there are  $N^2m$  nodes in the OPEN set in the worst case. Hence the worst case running time of A\* is  $\mathcal{O}(mN^2 \log(mN^2))$ . For the proposed algorithm, we assume the domain is partitioned into  $K$  number of cells in total, and each boundary curve has the number of adjacent boundaries no more than  $B$ . We first derive the worst case running time. If the computation of Algorithm 4 can be performed in a constant time, and the depth of optimal solution is  $D$ . Then the worst case running time of

---

**Algorithm 3:** Main algorithm

---

**Data:** initial position  $x_0$ , final position  $x_f$ , partitioned cell  $\{R_\alpha\}_{\alpha \in I_R}$   
**Output:** optimal junction position  $x(\lambda^{opt})$

```
1 visited  $\leftarrow$  {FALSE};
2  $\mathcal{C} \leftarrow$  {};
3 CostArrival = 0;
4 TotalCost_ub = 0;
5 Start node  $s = [c_0, c_1]$ , goal node  $d = [c_n, c_{n+1}]$ ;
6  $\lambda^{opt} = \text{findAllCellSeq}(s, d, \text{visited}, \mathcal{C}, \text{adjacency}, f_g(s), \text{TotalCost\_ub})$ ;
7
8 Function findAllCellSeq ( $n_c, d, \text{visited}, \mathcal{C}, f_g(n_c), \text{TotalCost\_ub}$ ):
9   visited( $n_c$ ) = TRUE;
10   $\mathcal{C}.\text{append}(n_c)$ ;
11  if  $n_c = d$  then
12     $\lambda = \text{Intermittent\_Diffusion}(\mathcal{C})$ ;
13    if  $J(\lambda) < \text{TotalCost\_ub}$  or  $\text{TotalCost\_ub} = 0$  then
14      TotalCost_ub  $\leftarrow$   $J(\lambda)$ ;
15       $\lambda^{opt} = \lambda$ ;
16    end
17  end
18  else
19    for all adjacent node  $\{m_j\}_{j=1}^M$  of  $n_c$  do
20      if visited( $m_j$ ) = FALSE then
21        Compute  $g_{j,lb}$  using (4.17) or (4.18);
22        Compute  $f_g(m_j)$  using (4.19);
23        if  $f_g(m_j) > \text{TotalCost\_ub}$  and  $\text{TotalCost\_ub} \neq 0$  then
24          continue;
25        end
26        findAllCellSeq( $m_j, d, \text{visited}, \mathcal{C}, f_g(m_j) + g_{j,lb}, \text{TotalCost\_ub}$ );
27      end
28    end
29     $\mathcal{C}.\text{pop}$ ;
30    visited( $n_c$ ) = FALSE;
31  end
```

---

Algorithm 3 can be found by considering the scenario that each branch of the decision tree has to be searched at each iteration of decision tree expansion. This leads to the worst case time complexity  $\mathbb{O}(B^D)$ . To make the worst case running time of iBnBDFS comparable to that of A\*, we let  $B^D \leq mN^2 \log mN^2$ . As the depth of the decision tree  $D$  of the proposed algorithm must be smaller than the total number of partitions  $K$  in the domain, the above inequality leads to

$$K \leq 2 \frac{\log N}{\log B} + \frac{\log m + \log \log mN^2}{\log B}.$$

---

**Algorithm 4:** Intermittent Diffusion

---

**Data:** cell sequence  $\mathcal{C}$   
**Output:** the optimal junction position  $\lambda^*$  given the fixed cell sequence  $\mathcal{C}$

- 1 Initialize  $\lambda^0 = 0$ ;
- 2 Set evolution step number  $N$ ;
- 3 Choose threshold  $\epsilon$ ;
- 4 **for**  $i = 1, \dots, N$  **do**
- 5     Choose perturbation duration  $T_i$ ;
- 6     Choose perturbation intensity  $\sigma_i$ ;
- 7     **for**  $j = 1, \dots, T_i$  **do**
- 8         Update  $\lambda^i$  using (4.16);
- 9     **end**
- 10    Set  $\sigma_i = 0$ ;
- 11    **while not converges do**
- 12       Update  $\lambda^i$  using (4.16);
- 13    **end**
- 14 **end**
- 15 Set  $\lambda^* = \arg \min_{i \leq N} J(\lambda^i)$ ;

---

Although the second term on the right hand side depends on  $N$ , it is much smaller than  $\frac{\log N}{\log B}$ . Hence the worst-case computation time of the iBnBDFS algorithm can be lower or comparable to that of A\* if  $K \leq 2 \frac{\log N}{\log B}$ . Since the worst case iBnBDFS running time is the same as the worst case of the brute force DFS method, this is a conservative bound.

### 4.3 Completeness

In this section, we demonstrate that Algorithm 3 is complete if  $L(v)$  is a convex function of  $v$ . The detailed proof can be found in [74].

**Theorem 4.3.1.** *If the flow field is piece-wise constant and*

$$\max_{\alpha \in I_R} \|u_\alpha\| < V, \quad (4.20)$$

*let  $Q$  be the set of global minimizers,  $U$  be a neighborhood of  $Q$ . Then for any  $\epsilon > 0$ , there exists  $T_0, N_0, \sigma_0$  such that if  $T_i > T_0$ ,  $\sigma_i < \sigma_0$  (for  $i = 1, 2, \dots, N$ ) and  $N > N_0$  where  $T_i, \sigma_i, N$  are parameters in Algorithm 4,  $\mathbb{P}(\lambda^{opt} \in U) \geq 1 - \epsilon$ , where  $\lambda^{opt}$  is the optimal solution found by Algorithm 3. Thus, Algorithm 3 is complete.*

## 4.4 Simulation results

In this section, we provide simulations to validate the strength of the proposed method. First, the time-optimal and energy-optimal path planning examples with vehicle travel in simple canonical time flow field are presented. This example serves as a benchmark example wherein, we compare the solution obtained by our algorithm to solution derived from other path planning methods. Then we present a path planning example of using the proposed method to plan the time-optimal and energy-optimal path in a realistic ocean surface flow field. This simulation is intended to verify the performance of the proposed method in a highly complicated and strong real ocean flow field.

### 4.4.1 Jet flow in 3D space

For this benchmark example, we present path planning using the proposed method in a jet flow in 3D space. The domain consists of three regions, divided by two boundary surfaces,  $z = 10$  and  $z = 15$ . In the region where  $z \in (0, 10)$ , the flow speed is  $(0.5, 0, 0)$ . There is strong jet flow in the region where  $z \in (10, 15)$  with flow speed  $(2, 1, 0)$ . The flow speed is zero in the region where  $z \in (15, 20)$ . The starting position is assigned at the origin, while the goal position is assigned at  $(0, 0, 20)$ .

The left figure in Fig. 4.3 shows the time-optimal path planned by the proposed method. The time-optimal solution is compared with the time-optimal path planned by the LSM and the A\* method. The comparison result is shown in Table 4.1. In this comparison, assuming the path segment  $x_{i+1} - x_i$  travels from the boundary surface  $f_{\alpha_i\beta_i}$  to reach the boundary surface  $f_{\alpha_{i+1}\beta_{i+1}}$ , we define  $\theta_i$  as the angle between path segment  $x_{i+1} - x_i$  and the boundary surface  $f_{\alpha_{i+1}\beta_{i+1}}$ .  $\gamma_i$  is defined as the angle between the projection of  $x_{i+1} - x_i$  on the boundary surface  $f_{\alpha_i\beta_i}$  and the x-axis of  $f_{\alpha_i\beta_i}$ ,  $\theta_i \in (0, 90^\circ]$ ,  $\gamma_i \in (-180^\circ, 180^\circ]$ . From the table,  $\theta_i$  and  $\gamma_i$  computed from the proposed method and the LSM are similar, with approximately  $1^\circ$  difference. Travel time of the optimal path planned by the proposed method and LSM are

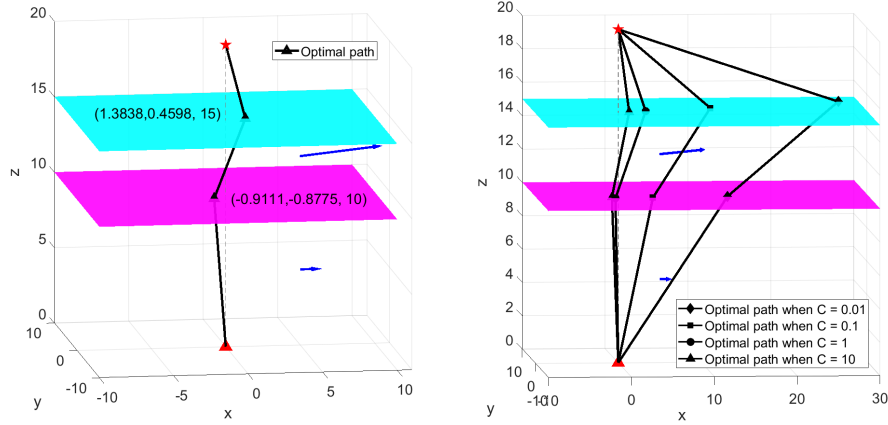


Figure 4.3: **(Left)**: Time optimal path planned by the proposed method. **(Right)**: Energy optimal path planned by the proposed method.. In both plots, boundaries of the jet flow are denoted by the colored surfaces. The flow speed in the domain is represented by the blue arrows. The optimal path is marked by black line, while the marker position denotes junction points computed by the proposed method.

also approximately the same. Since A\* method discretizes the domain and only search for the optimal path on the grid points, comparing the angles of the planned path between the proposed algorithm and A\* would yield little or no insight in performance of the algorithms. Thus we only compare the optimal total cost of the proposed algorithm and the A\*. Total cost of the optimal path planned by the A\* method is 7.1298, which is comparable to the optimal solution found by the proposed algorithm. This shows that the proposed algorithm converges to the optimal solution.

Table 4.1: Comparison between using the proposed method, LSM for time-optimal path planning

	Proposed Method	LSM
$\theta_1$	82.7924	83.5659
$\theta_2$	62.0255	63.3118
$\theta_3$	73.7397	73.8027
$\gamma_1$	-136.0775	-135.6592
$\gamma_2$	30.2293	30.2407
$\gamma_3$	-161.6199	-161.2246
Total cost	6.9096	6.9826

The energy-optimal path is shown in the right figure of Fig. 4.3. The energy-optimal path

when  $C = 10$  is exactly the same as the time-optimal path. As the assigned  $C$  decreases, the energy cost is attached relatively more weight in the cost function. Therefore, the vehicle tends to save more energy to go with the flow in the bottom region and the jet flow region. Thus, the energy-optimal path deviates from the time-optimal path as  $C$  decreases.

#### 4.4.2 Surface ocean flow

In this section we present path planning simulation of an underwater glider traveling in real ocean surface flow field near Cape Hatteras, North Carolina. While the energy efficiency of the glider's propulsion mechanism permits endurance of weeks to months, the forward speed of the vehicles is fairly limited (0.25-0.30 m/s), which can create significant challenges for navigation in strong currents. Use of a thruster in a so-called "hybrid" glider configuration can increase forward speed to approximately 1 m/s [75], but at great energetic cost. The continental shelf near Cape Hatteras is strongly influenced by the presence of the Gulf Stream, which periodically intrudes onto the shelf, resulting in strong and spatially variable flow that can be nearly an order of magnitude greater than the forward speed of the vehicle (2+ m/s). Due to the high flow speed, we consider the deployment of a hybrid underwater glider in this simulation, and consider the vehicle speed  $V = 1$  m/s.

The input flow map for path planning is given by a 1-km horizontal resolution version of the Navy Coastal Ocean Model (NCOM) [68] made available by J. Book and J. Osborne (Naval Research Laboratory, Stennis Space Center). The domain contains  $130 \times 130$  grid points. One snapshot of the dynamic flow field is shown in Fig. 4.4. We partition the flow field using the algorithm proposed in [22], and choose the number of partitioned cells using the "Elbow criterion" [76]. The domain is partitioned into 19 cells according to the "Elbow criterion".

We perform 3 sets of simulation, with each set contains 10 test cases. The start and destination position are chosen such that the distance between the two points is 40, 100, or 130 km. For the 3 set of simulations, the computation cost and total travel cost is

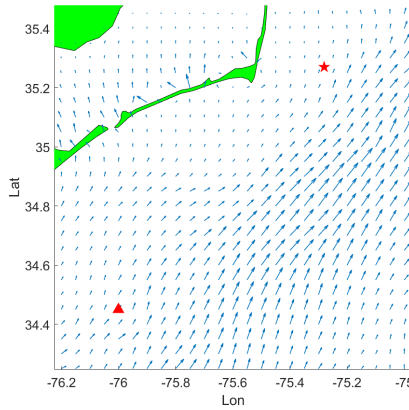


Figure 4.4: **(Left)**: Surface ocean flow field on May 27, 2017, 00:00 UTC at Cape Hatteras, NC. Red triangle and star indicate the starting and goal position (when  $d = 100$  km).

averaged over the 10 test cases. To verify performance of the proposed algorithm, we compare its simulation result with the A\* and the LSM. Both A\* and LSM run on the  $130 \times 130$  rectangular grid cells. To avoid the incompleteness issue of A\* [9, 11], in the node generation process of A\*, we consider each grid point have 16 neighboring nodes. The planned path of each test case is shown in Fig. 4.6. Since the input to the proposed algorithm is the partitioned flow field, while the input to the LSM and the A\* algorithm is the grid represented flow field, thus the optimal path computed by the proposed algorithm is different from the result of LSM and A\*. However, as shown in Fig. 4.5, solution quality of the proposed algorithm is comparable to that of the A\* and the LSM. Note that even though the proposed algorithm computes the planned path in the partitioned flow field, we compute the total travel cost of the vehicle tracking the planned path in the original flow field given by NCOM.

For all 3 sets of simulation, the proposed algorithm takes less computation time to compute the optimal solution to the minimum-time planning problem, when the flow field is partitioned into 19 cells according to the Elbow criterion. Even though the total cost of the optimal solution found by the proposed algorithm is slightly larger, about 2% – 9% larger than that of the A\* and LSM, it takes more than 40% lower computation cost than the A\* and the LSM. The proposed algorithm takes significantly less computation time in

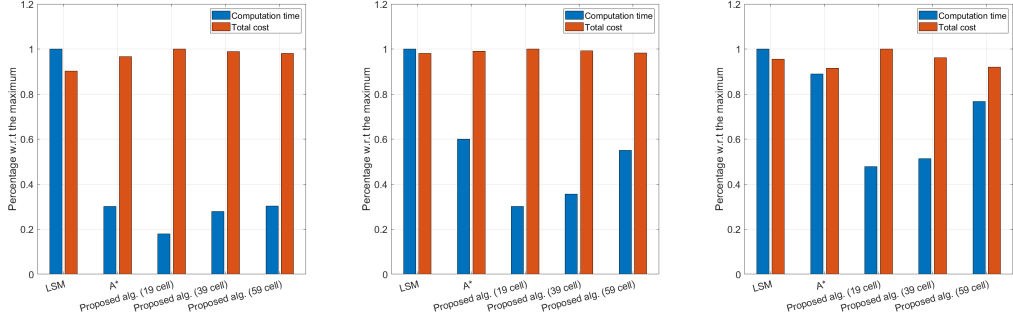


Figure 4.5: Comparison between using the proposed method and existing algorithms for time-optimal path planning. We compare the LSM and the A\* method with the proposed algorithm, in the cases when the flow field is partitioned into 19, 39 and 59 cells. The bar is showing the computation time and total cost of each algorithm. The computation time is normalized by the largest computation time among all the algorithms (the largest computation time is normalized as 1), and the total cost is also normalized by the largest among all the algorithms (the largest total cost is normalized as 1). **(Left)**: 40 km case; **(Middle)**: 100 km case; **(Left)**: 130 km case.

the simulation sets with shorter distance between the start and the destination node. The reason is that with smaller  $d$ , the decision tree is shallow, and the proposed algorithm only need to search through a small number of nodes to find the optimal solution. In addition, we increase the number of partitions into 39 and 59 cells, to demonstrate how the computation cost of the proposed algorithm scales with respect to different level of partitioning. The proposed algorithm takes more time to compute the optimal solution as the number of partitions increases. Even though the more detailed partition gives improved solution quality, the improvement is very little. Hence in order for the proposed algorithm to achieve good performance, it is desired to choose the proper level of partitioning to achieve the best trade-off between solution quality and computation cost.

The Energy optimal path is shown in Fig. 4.6. The proposed algorithm generates same results when  $C = 1e03$  and  $C = 20$ . In these cases, the running cost is much larger than the vehicle speed. Thus the energy-optimal planned path is identical to the time-optimal path. When  $C = 2e - 3$ , the running cost is much less than vehicle speed. In this case, instead of making use of the strong jet flow, the proposed method plans a path different from the time-optimal one.

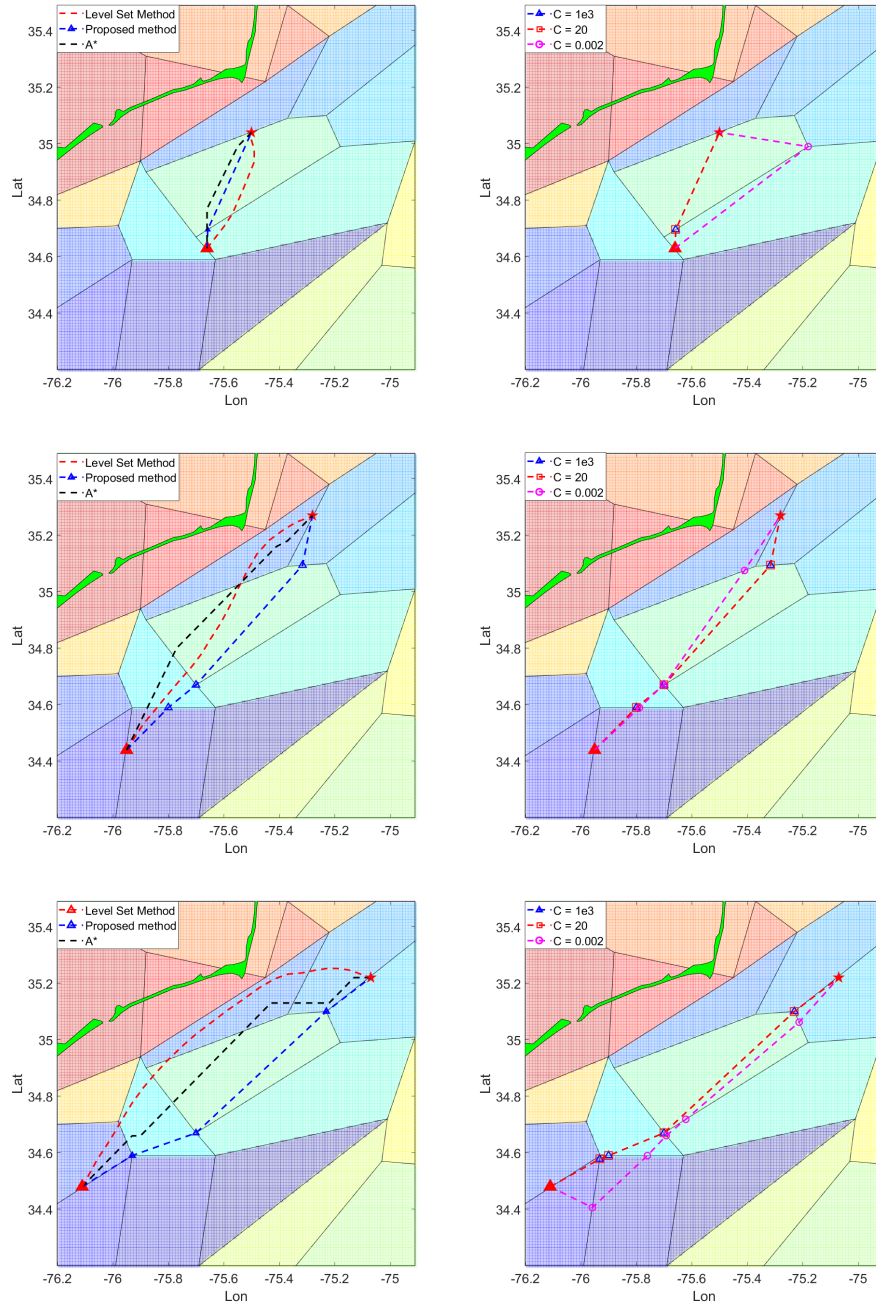


Figure 4.6: Optimal path when  $d = 40$  km (Upper row),  $d = 100$  km (Middle row), and  $d = 130$  km (Lower row). **(Left):** Time optimal path planned by the proposed method, the A\* method and the LSM. **(Right):** Energy optimal path planned by the proposed method, when  $C$  takes different value. The optimal path is marked by colored line, while the marker position denotes junction points computed by the proposed method.

## **CHAPTER 5**

### **PATH PLANNING IN REDUCED-ORDER FLOW FIELD: THE MODIFIED POTENTIAL SEARCH METHOD**

In this Chapter, we develop an algorithm that solves the AUV bounded cost path planning problem in the partitioned flow field. Given that the vehicle is traveling in a flow field represented by the data-driven computational model proposed in Chapter 3, the goal of this Chapter is to design a path that connects AUV initial position with goal position with the highest probability to have travel cost less than a pre-assigned upper bound. By introducing the key function, which is an implicit evaluation function of the probability that a path satisfies the bounded cost constraint, the optimal path is computed by searching for the nodes with lowest key function value using an informed graph search method.

The main novelty of this work is introducing the modified PTS method to solve the bounded cost search problem. Unlike the PTS method [6], which assumes that the branch cost of the graph is known exactly, our method deals with problems where the branch cost of the graph is uncertain. Given assumptions on the distribution of cost-of-arrival and cost-to-go, we prove that the proposed algorithm guarantees optimality of the planned path, that is, the planned path has the highest probability of satisfying the bounded cost constraint. To the best of our knowledge, this is the first time that the optimality of the modified PTS solution to bounded cost problems is proved. The proposed bounded cost path planning method can be viewed as an extension to iBnBDFS-ID in Chapter 4. Compared to iBnBDFS-ID, the modified PTS algorithm is computationally more efficient, since a best first search method is adopted to search for the junction positions. At the same time, optimality of the planned path to the bounded cost problem can be theoretically justified. The major benefit of the proposed bounded cost path planning algorithm lies in that it plans a path faster, while at the same time still guarantees the path quality.

## 5.1 Problem formulation

Our goal is to find a path connecting the vehicle current position  $x_0$  to the final position  $x_f$  that results in total travel time less than an upper bound  $C$ . Thus we formulate the following optimization problem, in which the decision variable is the vehicle's heading angle,  $\psi_C(t)$ . The optimization problem is to find the decision variable that is most likely to satisfy the bounded cost constraint:

$$\begin{aligned} \max_{\psi_C(t) \in [-\pi, \pi]} \quad & \Pr\left(T(\psi_C(t)) < C\right) \\ \text{s.t.} \quad & \dot{x} = v\Psi_C(t) + \theta\phi(x), \\ & x(T_0) = x_0, \\ & x\left(T_0 + T(\psi_C(t))\right) = x_f. \end{aligned} \tag{5.1}$$

where the total travel time to start from the initial position  $x_0$  to reach the destination position  $x_f$  under the control signal  $\psi_C(t)$  is denoted as  $T(\psi_C(t))$ .

## 5.2 A modified potential search method for AUV path planning

We partition the domain into a finite number of regions  $\{\mathcal{R}_i\}_{i \in I_R}$  using the method developed in Chapter 3. Let  $\{f_{\alpha\beta}\}_{(\alpha,\beta) \in I}$  be the boundary curves. Thus, all possible trajectories cross a sequence of cells of uniform flow, and finally reach the goal position. Let  $x_1, x_2, \dots, x_n$  denote the chain of junctions position, and  $c_1, c_2, \dots, c_{n+1}$  represent the index of the sequence of cells that the path crosses. Given the minimum travel cost to the sub problem  $g_i^t$  derived in (4.5), the planning problem (5.1) can be transformed into the following MIP, where the decision variables are the cell sequence and the junctions position,

$$\begin{aligned} \max_{\{x_i\}_{i=1}^n, \{c_i\}_{i=1}^{n+1}} \quad & \Pr\left(g_i^t(x_1, x_0, c_1) + \sum_{i=2}^{n-1} g_i^t(x_i, x_{i-1}, c_i) + g_i^t(x_f, x_{n-1}, c_n) < C\right), \\ \text{s.t.} \quad & f_{c_i, c_{i+1}}(x_i) = 0, \forall i \in [1, n], \end{aligned} \tag{5.2}$$

Now let us explain the proposed solution to (5.2). The solution is presented in Algorithm 5. We propose a bounded cost path planning algorithm that solves for the path that is most likely to satisfy the bounded cost constraint. The solution contains two steps, the first step is solving for the optimal sequence of cells that is most likely to result in a bounded cost path, in the discretized flow map described by the piece-wise constant flow cells. In this step, the junction positions are unknown. An informed graph search method is used for the first step of the proposed solution. The second step is optimizing junction positions on the boundaries of the optimal cell sequence.

The proposed solution is presented in Algorithm 5. First we describe the first step of our solution. Consider having a candidate junction on boundaries of all cells. Two junctions  $x_i$  and  $x_j$  are defined as adjacent if  $f_{p,q}(x_i) = 0$ ,  $f_{r,s}(x_j) = 0$ ,  $\{p, q\} \cap \{r, s\} \neq \emptyset$ , indicating that the two junctions are on different boundaries of the same cell. Two adjacent junctions are connected by an edge. A non-directed graph  $\mathcal{G}$  can be formed with the vertices being all the candidate junctions in the domain, and the edges being the path segment between the adjacent candidate junctions. Let  $n_{i,j}$  describe the node that corresponds to the junction on boundary curve between  $\mathcal{R}_i$  and  $\mathcal{R}_j$ , and let  $s$  and  $g$  denote the node corresponding to the starting and final position. Then in this context, a path  $x$  from the starting position  $x_0$  to the final position  $x_f$ , crossing the cells  $\mathcal{R}_{c_1}, \dots, \mathcal{R}_{c_{n+1}}$  in sequence can be represented by a sequence of nodes  $s \rightarrow n_{c_1 c_2} \rightarrow n_{c_2 c_3} \rightarrow \dots \rightarrow n_{c_n c_{n+1}} \rightarrow g$  on the graph. Figure 5.1 is an example of the graph representation of the workspace, in which case the flow field is partitioned into 4 cells.

The branch cost of the graph is defined as the travel time from one junction to another adjacent junction. The travel time can be computed by (4.5) if the two junction positions are known. However, since the junction positions are unknown when optimizing the cell sequence, the branch cost of the graph cannot be explicitly computed. Hence we introduced the following assumption:

**Assumption 5.2.1.** *We assume that the branch cost for all edges in  $\mathcal{E}$  is a random variable,*

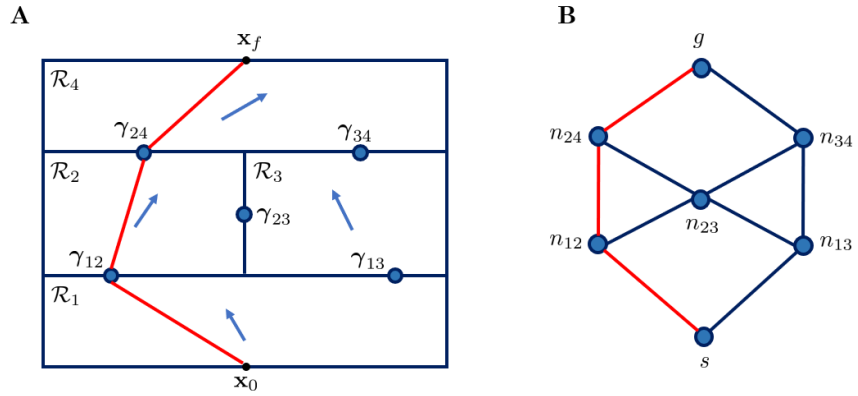


Figure 5.1: **(A)** Partitioned cells in the domain. On each boundary of two adjacent cells there is a candidate junction point, represented as the purple triangle. **(B)** Graph representation of the workspace. The vertices represent the candidate junctions, while the edges are the path segment between the adjacent junctions. The red line on both of the plots represent the same example path.

*with a known minimum value.*

The informed graph search method we propose is an extension to a class of graph search algorithms called potential search (PTS) [6]. The PTS algorithms can be viewed as modifications to the celebrated A\* algorithm for path planning [77]. To determine which nodes should be searched, the algorithms maintain an OPEN list and a CLOSED list. A graph node is labeled NEW if it has not been searched by the algorithm. The OPEN list contains all the nodes that are searched, but still have a NEW neighbor. The CLOSED list consists of all the nodes that have been accessed by the search algorithm.

To determine which cells should be searched first by the algorithm, the algorithm computes the cost-of-arrival, which is the minimal cost of going from the starting node  $s$  to an arbitrary node  $n$ , and cost-to-go, which is the minimal cost of going from  $n$  to the goal point  $g$ . Let  $g^*(n)$  denote the actual cost-of-arrival, and let  $h^*(n)$  denote the actual cost-to-go of a node  $n$ . Since the actual cost-to-go is unknown during the search, a heuristic cost  $h(n) \leq h^*(n)$ , is usually used by the search algorithm. The A\* search algorithm sort the OPEN list according to the value of  $g^*(n) + h(n)$ . The node with lowest value is

searched first.

In our problem, the following estimated cost-to-go is used to guide the search:

$$h(n) = \frac{\min_{x_n} |x_f - x_n|}{V + \max_{i \in I_R} \|\theta^i\|}. \quad (5.3)$$

The heuristic function defined in (5.3) is the travel time of the vehicle traveling in the most favorable flow condition, reaching goal position from the junction position that is closest to the goal. Hence,  $h(n) \leq h^*(n)$ , which is required by A\* search. However, in our problem, the branch cost is unknown. The exact value of actual cost-of-arrival cannot be computed during the search process, which is different from a typical path planning problem that can be solved by A\* or PTS method. Hence we introduce the estimated cost-of-arrival, denoted as  $g(n)$ . The estimated cost-of-arrival is computed by summation of the minimum branch cost along the path, thus  $g(n) \leq g^*(n)$ .

The goal of A\* search is to find the path with minimum cost. In our problem, due to the uncertainties in the branch cost, this goal is overly ambitious. Hence our problem formulation (5.2) aims to find a path with bounded cost. We define a potential function described as follows:

**Definition 5.2.1.** *The potential of a node  $n$ , denoted as  $\text{PT}(n)$ , is  $\Pr(h^*(n) + g^*(n) < C)$ .*

The potential function characterizes the probability that a node is on a path that satisfies the bounded cost constraint. Nodes with high potential have higher probability to be part of the desired path. However, the exact potential of nodes cannot be computed or compared, since both  $h^*(n)$  is unknown before the optimal path is found. Therefore, PTS algorithms usually design a key function to determine the nodes that need to be searched at each step of the graph search. Nodes in the OPEN list are sorted by the key function value instead of  $g^*(n) + h(n)$ , which is the main difference between the PTS algorithms and the A\* method. Various key functions have been proposed for different path planning problems with bounded cost [36, 6, 78].

One contribution of this Chapter is in extending the PTS method to solving bounded cost problems with uncertain branch cost, by introducing a new form of key function  $K(n) \in \mathbb{R}_{\geq 0}$  as

$$K(n) = \begin{cases} \frac{h(n)g(n)}{(C - h(n) - g(n))^2}, & \text{if } h(n) + g(n) < C \\ \infty & \text{, otherwise} \end{cases}. \quad (5.4)$$

$K(n)$  is an indication of the probability of the node  $n$  being on a path that satisfies the bounded cost constraint. Nodes with lower key function value have a higher probability of being on a path satisfying the bounded cost constraint. The intuition is that, if  $h(n) + g(n) < C$ , the key function value increases if either  $h(n)$  or  $g(n)$  is larger. In this case, the estimated cost  $h(n) + g(n)$  increases, and will be closer to  $C$ , then it is less likely that the true cost satisfies the bounded cost constraint, and the node  $n$  is less likely to be on a feasible path. If  $h(n) + g(n) \geq C$ , then  $n$  cannot be on a path satisfying the bounded cost constraint, since  $h^*(n) + g^*(n) \geq h(n) + g(n) \geq C$ . In this case, the key function is set as positive infinity.

The PTS with our new key function is then applied to search for the optimal cell in Algorithm 5. The only difference between our PTS and A\* is that the total cost used by A\* to sort the OPEN list is replaced by the key function, as shown in line 11. Similar to A\*, The search algorithm consists of two processes: the expansion process and the backtracking process. During the iterative expansion process, the algorithm orders the nodes in the OPEN set according to the key function value, and inserts the node with the lowest key function value to the CLOSED set (lines 17, 18). Neighbors of this node and their key function values are updated if the neighboring nodes can be reached with a lower cost through the current node (lines 24, 26, 27). The propagation continues until the OPEN list is depleted, or the goal node is in the OPEN set. Starting from the goal position, the backtracking process searches for the predecessor of the last node in the path set and add it to the path, until the starting node is included in the path (lines 35, 36).

The PTS algorithm fulfills step one of the bounded cost path planning solution. We have found the vector of indices  $\{c_i\}_{i=1}^{n+1}$ , which indicates the optimal indices that is most likely to result in a bounded cost path. In step two, we find the optimal junction positions that leads to the minimum total cost. Given the optimal cell sequence, the problem (5.2) converts to an optimization problem depending on the junction positions  $\{x_i\}_{i=1}^n$  in all cells,

$$\begin{aligned} \min_{\{x_i\}_{i=1}^n} & g_i^t(x_0, x_1, c_1) + \sum_{i=1}^n g_i^t(x_i, x_{i+1}, c_i) + g_i^t(x_n, x_f, c_{n+1}) \\ \text{s.t.} & f_{c_i, c_{i+1}}(x_i) = 0. \end{aligned} \quad (5.5)$$

This optimization problem is solved by the interior-point method.

The optimal heading angle can be computed from the junction positions. In each cell of the sequence  $\{c_i\}_{i=1}^n$ , given the optimal junction position  $x_{i+1}$  and  $x_i$ , the heading angle in cell  $\mathcal{R}_{c_i}$  can be derived by

$$\Psi_C = \frac{1}{V} \left( \frac{x_{i+1} - x_i}{\|x_{i+1} - x_i\|} \sqrt{V^2 + 2V\theta^{\top}\Psi_C + \|\theta^i\|^2} - \theta^i \right). \quad (5.6)$$

### 5.3 Theoretical justification

In this subsection, we prove that Algorithm 5 finds the optimal solution of (5.1). Assumptions 5.3.1 and 5.3.2 are required for the optimality proof.

#### 5.3.1 Optimality proof

**Assumption 5.3.1.** *Consider any node not the starting node  $s$  or the goal node  $g$ , the estimated cost-of-arrival  $g(n)$  and the estimated cost-to-go  $h(n)$  are bounded below,  $g(n) \geq g_{\min}$ ,  $h(n) \geq h_{\min}$ , where  $g_{\min} > 0$  and  $h_{\min} > 0$ .*

**Remark 5.3.1.** *For any node that is not the goal node,  $h(n)$  reaches its minimum when  $n$  is an adjacent node of  $g$ . Similarly, for any node that is not the start node,  $g(n)$  reaches its minimum when  $n$  is adjacent to  $s$ . Since the flow partition algorithm is performed over*

discrete grid points in  $\mathcal{D}$ , size of the cells cannot be infinitely small. Therefore,  $h(n)$  can only be zero if the junction represented by the node  $n$  is sliding on the same boundary of the goal point, and  $g(n)$  can only be zero if the junction represented by the node  $n$  is sliding on the same boundary of the start point. However, by junction assignment, only one junction can be assigned on each boundary. Hence there exists  $h_{\min}$  and  $g_{\min}$  that bound  $h(n)$  and  $g(n)$  from below, and the lower bound cannot be infinitely small.

Let  $H_{\max} = \max\{\frac{C}{h_{\min}}, \frac{C}{g_{\min}}\}$ . Consider  $\{X_n\}_{n=1}^N, \{Y_n\}_{n=1}^N$  to denote sequences of independent and identically distributed random variables uniformly distributed over  $[1, H_{\max}]$ . To prove optimality of the algorithm, we make assumptions on the statistical relationship between  $h^*(n), h(n)$ , and  $g^*(n), g(n)$  as follows.

**Assumption 5.3.2.** *The true cost-to-go,  $h^*(n)$  and the heuristic function  $h(n)$ , as well as the true cost-of-arrival,  $g^*(n)$  and estimated cost-of-arrival,  $g(n)$  both satisfy  $h^*(n) = h(n)Y_n, g^*(n) = g(n)X_n$ .*

**Remark 5.3.2.** *Both  $h^*(n)$  and  $g^*(n)$  are summation of branch cost along the optimal path. Since the branch cost is the travel time between two adjacent junctions sliding on two boundaries, the branch cost of all edges must have both a lower bound and an upper bound. Hence both  $h^*(n)$  and  $g^*(n)$  are assumed to be a uniform distribution, with the minimum of it being  $h(n)$  and  $g(n)$ , and the maximum being  $h(n)H_{\max}$  and  $g(n)H_{\max}$ . In practice, the statistical model of  $h^*(n)$  and  $g^*(n)$  depends on the distribution of the flow field, and may not be uniform distribution in some flow cases. However, the following theoretical analysis can be adapted to other parameterization of the statistical model of  $h^*(n)$  and  $g^*(n)$ .*

We will show below that, by expanding the nodes with the lowest key function value without explicitly calculating the potential of nodes, the proposed algorithm expands the nodes with the highest potential value, and thus guarantees to find the optimal solution to (5.1). Lemma 5.3.1 states that the key function is an equivalent evaluation of the potential value of nodes. Lemma 5.3.2 demonstrates that the optimal path can be equally defined by

either the potential or the key function value of nodes. Finally, given the two Lemmas, we justify the optimality of the proposed algorithm, which is stated in Theorem 5.3.3.

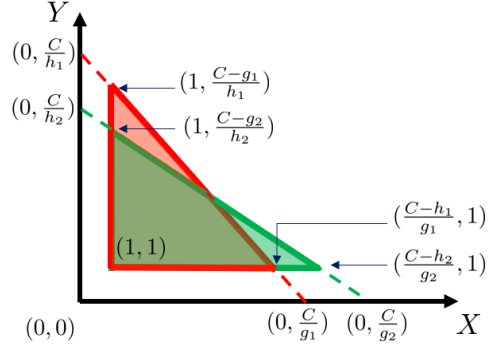


Figure 5.2: Illustration of computing  $PT(n_1)$  and  $PT(n_2)$ . The red triangle is  $S_1 = \{(x, y) | h_1 y + g_1 x < C, x \in [1, H_{\max}], y \in [1, H_{\max}]\}$ , and the green triangle is  $S_2 = \{(x, y) | h_2 y + g_2 x < C, x \in [1, H_{\max}], y \in [1, H_{\max}]\}$ .

**Lemma 5.3.1.** For all  $n_1, n_2 \in \mathcal{G}$ ,  $PT(n_1) < PT(n_2)$  if and only if  $K(n_1) > K(n_2)$ .

*Proof.* To simplify notation, let  $h_1, h_1^*, g_1, g_1^*$  denote  $h(n_1), h^*(n_1), g(n_1)$ , and  $g^*(n_1)$ , respectively. The Lemma trivially holds in the cases where either  $K(n_1)$  or  $K(n_2)$  is infinity. Below we show that the Lemma holds in the case where both  $K(n_1)$  and  $K(n_2)$  are not infinity; equivalently,  $h_1 + g_1 < C$  and  $h_2 + g_2 < C$ . Due to the i.i.d. distribution assumption stated in Assumption 5.3.2,  $X_1, X_2$  can be written as a single random variable uniformly distributed on  $[1, H_{\max}]$ , and  $Y_1, Y_2$  also can be written as a single random variable uniformly distributed on  $[1, H_{\max}]$ . Therefore,  $PT(n_1) < PT(n_2)$  if and only if

$$\Pr(h_1 Y + g_1 X < C) < \Pr(h_2 Y + g_2 X < C).$$

Terms in the above inequality can be computed by integration of the probability density function,

$$\iint_{S_1} \rho_{X,Y}(x, y) dx dy < \iint_{S_2} \rho_{X,Y}(x, y) dx dy,$$

where  $S_1 = \{(x, y) | h_1 y + g_1 x < C, x \in [1, H_{\max}], y \in [1, H_{\max}]\}$ ,  $S_2 = \{(x, y) | h_2 y + g_2 x < C, x \in [1, H_{\max}], y \in [1, H_{\max}]\}$ . By Assumption 5.3.2,  $X \leq \frac{C}{g_{\min}}$ ,  $Y \leq \frac{C}{h_{\min}}$ , and therefore  $S_1, S_2$  are two triangles, as shown in Figure 5.2. Due to the uniform distribution of  $X, Y$ , the above integration can be easily computed by multiplying area of  $S_1, S_2$  with the joint distribution  $\rho_{X,Y}(x, y)$ , which is a constant. Hence,

$$\frac{1}{2} \rho_{XY} \left( \frac{C - g_1}{h_1} - 1 \right) \left( \frac{C - h_1}{g_1} - 1 \right) < \frac{1}{2} \rho_{XY} \left( \frac{C - g_2}{h_2} - 1 \right) \left( \frac{C - h_2}{g_2} - 1 \right)$$

which implies  $K(n_1) > K(n_2)$ . □

Let the ordered sequence  $x$  denote a path connecting the start node  $s$  with the goal node  $g$  in the graph  $\mathcal{G}$ . Define  $K_{\max}(x)$  as the highest key function value of all nodes on the path  $x$ , that is,  $K_{\max}(x) = \max_{n \in x} K(n)$ .

**Lemma 5.3.2.** *The optimal path minimizes  $K_{\max}(x)$  over all paths in the graph.*

*Proof.* Let  $x^*$  denote the optimal path that maximizes  $\Pr(h^*(n) + g^*(n) < C)$ . Suppose that there is a path  $x'$  that is different from the optimal path  $x^*$  with  $K_{\max}(x') < K_{\max}(x^*)$ , then there exists  $n' \in x'$ , and  $n \in x^*$  that satisfies  $K(n') < K(n)$ . By Lemma 5.3.1,  $\text{PT}(n') > \text{PT}(n)$ , indicating that  $\Pr(h^*(n') + g^*(n') < C) > \Pr(h^*(n) + g^*(n) < C)$ , which contradicts the assumption that  $x^*$  is the optimal path. Hence, a path is the optimal one if it minimizes  $K_{\max}(x)$ .

Conversely, let  $\Gamma = \arg \min_{\Gamma'' \in \mathcal{G}} K_{\max}(\Gamma'')$ , then for all  $\Gamma'$  that is different from  $\Gamma$ , for all  $n \in \Gamma, \exists n' \in \Gamma'$ , such that  $K(n) < K(n')$ . Thus by Lemma 5.3.1,  $\text{PT}(n) > \text{PT}(n')$  for  $n'$  in any arbitrary path that is not  $\Gamma$  in the graph, and hence  $\Gamma$  that minimizes  $K_{\max}(\Gamma)$  is the optimal path. □

**Theorem 5.3.3.** *When a feasible solution exists, the proposed algorithm terminates if and only if an optimal path is found.*

*Proof.* Algorithm 5 can only terminate by finding the goal node, or after depleting the OPEN set. However, the OPEN set can never be empty before termination if there is a feasible path from  $s$  to goal point. Hence Algorithm 5 must terminate by finding a goal point.

Next we show that Algorithm 5 terminates only by finding an optimal path to the goal node. Suppose that the algorithm terminates by finding a path,  $x'$  other than the optimal path  $x^*$ , then by Lemma 5.3.2,  $K_{\max}(x^*) < K_{\max}(x')$ , that is, there exists  $n' \in x'$ ,  $n \in x^*$  such that  $K(n) < K(n')$ . Thus during the propagation process, Algorithm 5 would have selected  $n$  for expansion rather than  $n'$ , contradicting the assumption that the algorithm terminates by finding  $x'$ . Hence the algorithm must terminate by finding the optimal path to the goal node.  $\square$

### 5.3.2 Complexity analysis

In our analysis, we derive the worst case running time for Algorithm 5, and compare it with dynamic programming based planning methods, such as A\*, to demonstrate the computational efficiency of the proposed planning algorithm. Let us suppose the flow field forecast is available on  $N \times N$  grid points in the deployment domain, and suppose that the domain is partitioned into  $M$  cells by performing Algorithm 1.

To derive the worst case running time of the proposed algorithm, we first consider the partitioning. Since one junction must be formed by the boundary of at least two cells, the total number of junctions cannot exceed  $M(M - 1)$ , and hence the total number of nodes in the graph is at most  $M(M - 1)$ . In one iteration process, the sorting operation (line 11), and computation of the key function, the minimum branch cost, and the heuristics (line 27, line 21 and line 25 are performed for one time. Suppose that the OPEN set is implemented using a heap data structure, the worst case running time of the operation in line 11 is  $\mathcal{O}(\log(M(M - 1)))$ . We assume that the computation of the key function, the minimum branch cost, and the heuristics can be performed in constant time. There can be at most  $M(M - 1)$  iterations during the entire execution, before the OPEN set is depleted.

Hence, the worst case running time of Algorithm 5 is  $\mathcal{O}\left(M(M-1)\log(M(M-1))\right)$ .

The worst case running time of A\* is  $\mathcal{O}(2N^2\log N)$  [79]. Thus, the proposed algorithm is more computationally efficient than A\* if  $M(M-1) < N^2$ , meaning that Algorithm 1 partitions the domain into less number of cells than the number of rectangular cells in the original gridded domain.

#### 5.4 Simulation results

In this example, we present simulation results of AUV bounded cost path planning. Since the flow field is of high speed in the domain of interest, we assume that the glider is sampling the domain using combined propulsion of buoyancy and thrusters for the operation. Hence the AUV through-water speed is set to be 0.5 m/s. The simulations are run on a core i7 at 1.80GHz PC with 32GB RAM.

Figure 5.3 shows one example of the proposed bounded cost path planning method. The start position and goal position is assigned as  $(-75.60, 35.06)$  and  $(-74.98, 35.83)$  in longitude and latitude, respectively. Upper bound of the travel time is set as 72 hours. The travel cost of the resulting path is 62.650 hours, which satisfies the bounded cost constraint. As shown in the figure, the generated path makes a detour and takes advantage of the strong northward ocean flow to travel to the goal position.

We perform A\* [80] and Level Set Method (LSM) [58] as comparison to the proposed method. 15 test cases are generated in total. Each test case  $\mathbb{T} = \{\text{Start}, \text{Goal}, d\}$  is built by first assign the distance between the start and the goal node  $d$  to be 20 km, 50 km, 80, or 100 km, then randomly place the Start point in the domain, and select the Goal node so that the distance to the start node is  $d$ . The computation time column in Table 5.1 shows comparison of the averaged computational time for A\*, LSM, and the proposed algorithm. The difference between the three algorithms is due to the number of nodes in the graph. By partitioning the domain into 13 cells, the proposed algorithm searches for the optimal path in a graph with only  $13 \times 12$  nodes, while both the A\* and LSM algorithm searches for the

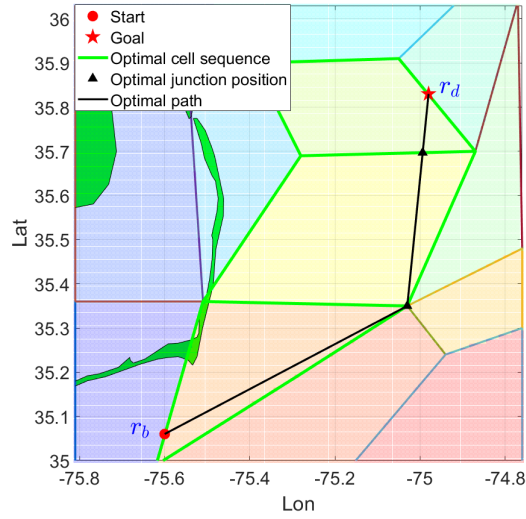


Figure 5.3: Example of simulation case. The resulting path is computed by the proposed method.

optimal path in a domain containing  $106 \times 106$  nodes. Thus, the computational cost of the proposed algorithm is significantly lower than the other two methods.

Further, we compare the percentage of increase in the computation time when  $d$  increases. In Table 5.1, the % of increase column shows the increase in the computation cost when  $d$  increases from 20 to 50, 80, and 100, respectively. The percentage is calculated by considering the computation time of each algorithm when  $d = 20$  km as the base time, and divide the increase of computation time when  $d$  scales up by the base time. It can be seen that when the domain of interest scales up, the computational cost of the proposed algorithm has the least increase, compare with A\* and LSM. This is because as  $d$  increases, both A\* and LSM have to expand significantly more nodes before finding the optimal solution. For the proposed algorithm, the number of nodes to be expanded stays relatively constant as  $d$  increases. Hence, its computation cost does not increase as much as A\* and LSM as the domain scales up.

---

**Algorithm 5: Bounded Cost Search in Piece-wise Constant Flow Field**

---

**Data:** Start and goal node  $s, g$ , start and goal position  $x_0, x_f$ , travel cost upper bound  $C$ , graph  $\mathcal{G}$ .

**Output:** Optimal heading angle  $\Psi_C$ .

```
1 PTS( $s, g, C, \mathcal{G}$ )  $\rightarrow \mathcal{C}$  IntermittentDiffusion( $\mathcal{C}, x_0, x_f$ )  $\rightarrow \Psi_C$ 
2 Function PTS( $s, g, C, \mathcal{G}$ ) :
3   Initialization.  $g(n) = \infty, h(n) = \infty, K(n) = \infty, \forall n \in \mathcal{G}$ ;
4    $s \rightarrow \{\text{CLOSED}\}$  ;
5   Set the heuristics and estimated cost-of-arrival of  $s$ ;
6   (OPEN, CLOSED) = Expand( $s, \text{OPEN}, \text{CLOSED}, \mathcal{G}$ );
7   while OPEN is not empty do
8     if  $g \in \text{CLOSED}$  then
9       Backtracking( $s, g$ )  $\rightarrow \alpha$ 
10    end
11     $v = \arg \min_{n \in \text{OPEN}} K(n)$ ;
12    (OPEN, CLOSED) = Expand( $v, \text{OPEN}, \text{CLOSED}$ );
13  end
14  return  $\alpha$ 
15
16 Function Expand( $v, \text{OPEN}, \text{CLOSED}, \mathcal{G}$ ) :
17   $\{\text{OPEN}\} \setminus v \rightarrow \{\text{OPEN}\}$ ;
18   $v \cup \{\text{CLOSED}\} \rightarrow \{\text{CLOSED}\}$ ;
19  Find adjacent nodes  $\{n_i\}_{i=1}^m$  to  $v$  in  $\mathcal{G}$ ;
20  for  $i = 1$  to  $m$  do
21    Compute minimum branch cost  $w^*(v, n_i)$  by solving (4.5) ;
22    if  $w^*(v, n_i) + g(v) < g(n_i)$  then
23       $n_i.\text{predecessor} = v$ ;
24       $\{\text{OPEN}\} \cup n_i \rightarrow \{\text{OPEN}\}$ ;
25      Compute  $h(n_i)$  using (5.3);
26       $g(n_i) = g(v) + w^*(v, n_i)$ ;
27      Update  $K(n_i)$  using (5.4);
28    end
29  end
30  return OPEN, CLOSED
31
32 Function Backtrack( $s, g$ ) :
33   $g \rightarrow \mathcal{C}$ ;
34  while  $s \notin \{\mathcal{C}\}$  do
35     $v = \mathcal{C}(\text{end})$ ;
36     $v.\text{predecessor} \cup \mathcal{C} \rightarrow \mathcal{C}$  ;
37  end
38  return  $\mathcal{C}$ 
39
40 Function IntermittentDiffusion( $\mathcal{C}, x_0, x_f$ ) :
41  Initialize junction set  $x^0$  on the boundary curves of the cell sequence  $\mathcal{C}$ ;
42  Compute junction positions by solving (5.5);
43  Compute heading angle from junction positions by (5.6);
44  return  $\Psi_C(t)$ 
```

---

Table 5.1: Computation time comparison of A\*, Level Set Method, and the proposed algorithm. Avg. comp. time represents the averaged computation time for each simulation scenario, and STD comp. time represents the standard deviation of the computation time. % of increase describes the percentage increase in the computation cost when  $d$  increases.

Method	$d$	Avg. comp. time	STD comp. time	% of increase
Algorithm 5	20 km	0.1576 s	0.0365	-
	50 km	0.1603 s	0.0385	1.7%
	80 km	0.2796 s	0.1127	77%
	100 km	0.4276 s	0.2318	171%
A*	20 km	1.6776 s	0.5199	-
	50 km	7.4376 s	0.7692	343%
	80 km	11.8376 s	1.3216	605%
	100 km	14.9040 s	1.1900	788%
LSM	20 km	86.8376 s	9.4397	-
	50 km	145.7043 s	30.0730	67%
	80 km	210.6376 s	23.4036	142%
	100 km	241.9043 s	25.6065	178%

## CHAPTER 6

### MORI-ZWANZIG FORMALISM BASED BELIEF ABSTRACTION FOR UNCERTAINTY-AWARE DECISION-MAKING

In this Chapter, we propose a novel method for memory constrained belief abstraction to solve a POMDP problem. Motivated by the need to plan optimal paths for AUVs, we propose an abstraction strategy of the belief space that allocates a fixed number of partitions to regions where the system state most likely will visit. Unlike other approximation strategies, we use a K-means-based partition approach to allocate a fixed number of partitions based on a set of Monte Carlo simulations. As we pre-define our partition count, our approach avoids the curse of dimensionality problem other abstraction methods must deal with while retaining abstraction accuracy. To improve modeling accuracy, we develop a data-driven method to identify the non-Markovian impact of the residual unmodeled dynamics on the reduced-order approximation model by a recurrent neural network. Given the identified non-Markovian approximated belief dynamics, we leverage the HTN (Hierarchical Task Network) planner to solve the symbolic planning problem in belief space. Theoretical analysis on the model reduction error upper bound is provided. Finally, simulation results demonstrate that the proposed algorithm retain accurate description of the belief dynamics, and the proposed abstraction algorithm enables efficient solution to the symbolic planning problem. Our major contributions are as follows: **i)** we propose a novel discretization strategy of the belief state that allows us to allocate a fixed number of partitions over regions most frequently visited, **ii)** based on the M-Z formalism, we propose a data-driven approach to identify the high-fidelity symbolic representation of the belief state and its dynamics in order to solve the planning problems in belief space, and **iii)** we prove that by leveraging the M-Z formalism to model the memory effect, the belief abstraction scheme retains a time-uniform upper bound of the model reduction error, and **iv)** we provide simulation studies comparing our

proposed method with other approximation strategies demonstrating that our approximation achieves significantly reduced computation time, and at the same time does not sacrifice solution quality.

## 6.1 Preliminaries and problem formulation

### 6.1.1 Preliminaries

#### *Continuous-state POMDP*

Let  $x_k \in \mathcal{D}$  be the system state, where the domain  $\mathcal{D}$  is a compact subset of  $\mathbb{R}^{n_x}$ .  $y_k \in \mathbb{R}^{n_y}$  be the observation in a continuous space. Action of the system  $u_k$  belongs to a finite action space  $U \in \mathbb{R}^{n_u}$ . System dynamics and observation model are then described by

$$x_{k+1} = f(x_k) + u_k + \omega_k \quad (6.1)$$

$$y_k = h(x_k, n_k), \quad (6.2)$$

where  $\omega_k \in \mathbb{R}^{n_\omega}$  and  $n_k \in \mathbb{R}^{n_n}$  are i.i.d. continuous random vectors with known distributions.

We retain a density  $b_k$ , called the belief state, that is updated by the noisy inputs and observations  $u_k$  and  $y_k$ . Let  $I_k = [y_k, y_{k-1}, \dots, y_0]$  denote the sequence of historical observations,

$$b_k(x) \triangleq \Pr(x_k = x | I_k). \quad (6.3)$$

At each timestep, the belief state can be updated by the system dynamics and the noisy observation. The system equation (6.1) determines a probability transition kernel  $T(u_k) : \mathcal{P}(\mathbb{R}^n) \times \mathbb{R}^n \rightarrow \mathbb{R}$  by

$$b_{k+1}^{(-)}(dx_{k+1}) = \int_{\mathbb{R}^{n_x}} b_k(dx_k) T_k(dx_k, x_{k-1}, u_k). \quad (6.4)$$

To simplify the notation, we use an operator  $\mathcal{T}(u_k) : b_k \rightarrow b_{k+1}^{(-)}$  to represent (6.4) so that

$$b_{k+1}^{(-)} = \mathcal{T}(u_k)b_k. \quad (6.5)$$

Updating the prior distribution with the observation via the Bayesian law yields the belief state at the next timestep,

$$b_k = \frac{\Psi_k b_k^{(-)}}{\langle b_k^{(-)}, \Psi_k \rangle}, \quad (6.6)$$

where  $\Psi_k(x) \triangleq \Pr(y_k|x)$  denotes the likelihood function, and  $\langle f, g \rangle = \int f(x)^\top g(x) dx$ . To guarantee that the Bayesian update is properly defined, we assume that for all  $k$ ,  $\langle b_k^{(-)}, \Psi_k \rangle > 0$ . We define an operator  $\mathcal{L}(y_k) : b_k^{(-)} \rightarrow b_k$  to represent the above Bayesian updating law:

$$b_k = \mathcal{L}(y_k)b_k^{(-)}. \quad (6.7)$$

To simplify the notation, we shorten the notation by writing the two operators  $\mathcal{T}(u_k)$  and  $\mathcal{L}(y_k)$  as  $\mathcal{T}_k$  and  $\mathcal{L}_k$  respectively, in the following sections. The control  $u_k$  and the observation  $y_k$  are embedded in the operators  $\mathcal{T}_k$  and  $\mathcal{L}_k$ . Then the belief dynamics can be described simply as

$$b_{k+1} = \mathcal{L}_k \mathcal{T}_k b_k. \quad (6.8)$$

### *Mori-Zwanzig formalism*

The key idea of the M-Z formalism is to convert a high-dimensional Markovian dynamical system into an equivalent, lower-dimensional non-Markovian system. The framework of M-Z formalism is centered around projection operators that separate the state of a dynamical system into resolved and a lower dimensional unresolved manifolds. We use the M-Z formalism to describe both the dynamics of the prior distribution (6.4) and the dynamics of the posterior distribution (6.8).

**Offline prior dynamics** We describe the prior dynamics with the M-Z formalism. To avoid confusion of incorporating the influence of historical observation into prior dynamics, here we consider prior dynamics without observation. We call it the offline prior dynamics, and use  $p_k(x)$  to denote the offline prior state at timestep  $k$ , to distinguish with the online prior state that is affected by historical observation.

We refer to (6.4) as the *full dynamics* of prior distribution. Note that the full dynamics is infinite dimensional. We then define a reduced-order prior state as the *resolved state*, which lies in a finite dimensional space spanned by linearly independent functions,  $\Phi = [\phi^1, \dots, \phi^K]^\top$ , then for any probability distribution function  $p$ , we let  $\mathcal{P}$  be the projection from the full state to the resolved state:

$$\mathcal{P}p_k(x) = \Phi(x)\langle \Phi, \Phi \rangle^{-1} \langle \Phi, p_k \rangle. \quad (6.9)$$

Further, we define a projection operator that is orthogonal to  $\mathcal{P}$  as

$$\mathcal{Q} = \mathcal{I} - \mathcal{P} \quad (6.10)$$

where  $\mathcal{I}$  is the identity map. The operator  $\mathcal{Q}$  acting on a probability distribution function  $p$  leads to  $\mathcal{Q}p_k = p_k - \mathcal{P}p_k$ . Hence the operator  $\mathcal{Q}$  describes states that cannot be modeled by the reduced-order resolved states. We call  $\mathcal{Q}p_k$  the *unresolved state*.

The M-Z formalism provides an exact description of the resolved state. To derive the M-Z formalism, we start from the resolved and unresolved state dynamics:

$$\mathcal{P}p_{k+1} = \mathcal{P}\mathcal{T}_k(\mathcal{P}p_k + \mathcal{Q}p_k), \quad (6.11)$$

$$\mathcal{Q}p_{k+1} = \mathcal{Q}\mathcal{T}_k(\mathcal{P}p_k + \mathcal{Q}p_k). \quad (6.12)$$

We can solve (6.12) by iteratively decomposing  $\mathcal{Q}b_k$  on the right hand side of (6.12) into

$\mathcal{Q}\mathcal{T}_{k-1}(\mathcal{P}p_{k-1} + \mathcal{Q}p_{k-1})$ :

$$\mathcal{Q}p_{k+1} = \sum_{n=0}^k G(k, n)\mathcal{P}p_n + G(k, 0)\mathcal{Q}p_0, \quad (6.13)$$

where  $G(k, n) = (\mathcal{Q}\mathcal{T}_k)(\mathcal{Q}\mathcal{T}_{k-1}) \dots (\mathcal{Q}\mathcal{T}_n)$  is the memory kernel. Inserting (6.13) into (6.11) yields the M-Z equation as

$$\mathcal{P}p_{k+1} = \mathcal{P}\mathcal{T}_k\mathcal{P}p_k + \mathcal{P}\mathcal{T}_k \sum_{n=0}^{k-1} G(k, n)\mathcal{P}p_n + \mathcal{P}\mathcal{T}_k G(k, 0)\mathcal{Q}p_0. \quad (6.14)$$

The first Markovian term is the “best guess” by functions in the span of  $\Phi$ . The second “memory” term captures the non-Markovian effects resulting from coupling effects in the range of  $\Phi$ . The last, “noise” term represents the contribution of the unresolved state at the initial timestep to the future resolved state.

**Online belief dynamics** Similar as the above derivation on the prior dynamics, here we can follow the same steps to derive the belief state with the M-Z formalism:

$$\mathcal{P}b_{k+1} = \mathcal{P}\mathcal{L}_k\mathcal{T}_k\mathcal{P}b_k + \mathcal{P}\mathcal{L}_k\mathcal{T}_k \sum_{n=0}^{k-1} \tilde{G}(k, n)\mathcal{P}b_n + \mathcal{P}\mathcal{L}_k\mathcal{T}_k\tilde{G}(k, 0)\mathcal{Q}b_0, \quad (6.15)$$

where  $\tilde{G}(k, n) = (\mathcal{Q}\mathcal{L}_k\mathcal{T}_k)(\mathcal{Q}\mathcal{L}_{k-1}\mathcal{T}_{k-1}) \dots (\mathcal{Q}\mathcal{L}_n\mathcal{T}_n)$  is the memory kernel of the belief dynamics. Since the belief state at current timestep depends on historical sequence of observation, the memory kernel of the belief state contains the Bayesian update, which is described by the operator  $\mathcal{L}$ . Hence the memory kernel of the belief dynamics is different from that of the prior dynamics.

Note that (6.14) and (6.15) are not approximation. They are actually exact representation of the resolved part of state in the span of  $\Phi$ . Unfortunately, such equation is still of little practical use. The memory term in (6.14) and (6.15) depends on infinite number of historical resolved states. Moreover, even if we approximate the sequence of infinite number of

memory terms with a finite one, it is still computationally difficult to numerically compute the memory kernel, as it is composition of a sequence of infinite dimensional operators. However, the memory terms contain a set of delay embedding coordinates, which naturally inspires us to introduce the approximation solution in Section 6.3.

### *Generalized Cell Mapping method*

The basis of the Generalized Cell Mapping (GCM) method is to identify a discrete-time Markov chain model whose “local properties” are consistent with the belief dynamics [41, 81, 82]. GCM uses a set of piece-wise constant basis functions, typically hyperrectangle or simplex, to construct a discretized representation of the belief state. Suppose that there are  $K$  discretized partitions in the domain, and each partition is denoted as  $\mathcal{R}_i, i \in [1, K]$ . Then we approximate any distribution by projecting it onto the following linearly independent functions,  $\Phi = [\phi^1, \dots, \phi^K]^\top$ , where

$$\phi^i(x) = \frac{1}{m(\mathcal{R}_i)} \mathbb{1}(x \in \mathcal{R}_i) = \begin{cases} \frac{1}{m(\mathcal{R}_i)}, & \text{if } x \in \mathcal{R}_i \\ 0, & \text{otherwise} \end{cases}, \quad (6.16)$$

is a piece-wise constant basis function, and  $m(\mathcal{R}_i)$  denote the mass of the cell  $i$ . Thus the projection can be written as:

$$\mathcal{P}b(x) = \sum_{i=1}^K \frac{\mathbb{1}(x \in \mathcal{R}_i)}{m(\mathcal{R}_i)} \int_{\mathcal{R}_i} b(u) du. \quad (6.17)$$

Note that the composition of  $\mathcal{P}$  and  $\mathcal{T}_k$  leads to:

$$\mathcal{P}\mathcal{T}_k\phi^i = \sum_{j=1}^K \frac{1}{m(\mathcal{R}_j)} \int_{\mathcal{R}_j} \mathcal{T}_k\phi^i(x) dx \phi^j. \quad (6.18)$$

Hence  $\mathcal{P}\mathcal{T}_k$  is a Markovian transition model. The GCM identifies a reduced-order prior dynamics in the discretized state space, by fixing the control input of the system to be a

constant and run Monte-Carlo simulation for each of the possible control input. Let us assume that  $u$  stays a constant vector,  $u_i \in \mathcal{U}$  throughout the whole simulation time interval. We define  $T_\theta \in \mathbb{R}^{K \times K}$ , and let  $T_\theta(u_i) = \mathcal{PT}(u_i)$  describe the markovian transition model. To evaluate (6.18), assume that we are provided with the set of simulated particle trajectories  $\{r_k^{(m)}\}_{m=1}^M$  with control input fixed to  $u_i$  throughout the whole Monte Carlo simulation, we can estimate the transition matrix by the number of particles moving from one cell to another,

$$T_\theta^{(ij)}(u_i) = \frac{\sum_{m=1}^M \mathbb{1}(r_k^{(m)} \in \mathcal{R}_i) \mathbb{1}(r_{k-1}^{(m)} \in \mathcal{R}_j)}{\sum_{m=1}^M \mathbb{1}(r_{k-1}^{(m)} \in \mathcal{R}_j)}. \quad (6.19)$$

The denominator represents the total mass of particles inside the cell  $\mathcal{R}_j$ , while the numerator is the mass transported from cell  $\mathcal{R}_j$  to  $\mathcal{R}_i$  at the next time step.

### 6.1.2 Problem formulation

Our goal is to present a general strategy to plan a sequence of control strategy  $u_k$  to accomplish a mission in a partially known environment. The mission is modeled through a planning problem, with the cost function  $J_k$  to be optimized, the belief dynamics constraint (6.8), and the initial and terminal belief space constraint, for instance the system state having high probability to lie inside a region.

For the scope of this paper, we focus on systems with limited control effort  $u_k$  that the system state cannot be commanded to follow arbitrary trajectories in the state space. This applies to scenarios of robotic applications where the environment exhibits significant influence on vehicle dynamics, such as AUV traveling in ocean flow fields, or flying robots with limited actuation forces.

While our formulation is general, we use a motivating example to guide the reader through the derivation. The motivation is the decision-making of AUV in partially-known underwater environment, as shown in Fig. 6.1. The AUV can receive acoustic localization signals from the underwater acoustic receivers located in the region. It can also actively

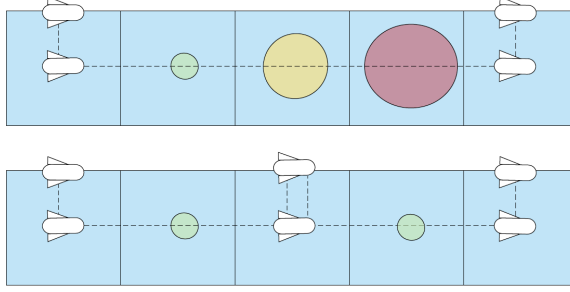


Figure 6.1: Illustration of AUV actions. To achieve the hotspot search goal, the vehicle can perform two sets of actions, to move in a certain direction, or to go to the surface of the ocean for active localization. The circles demonstrate the uncertainty level of vehicle position. The uncertainty level goes up as the vehicle performs move action (upper panel), while the vehicle can actively reduce the uncertainty by taking the surface/diving action (lower panel).

acquire localization information by going to the ocean surface to receive GPS localization signals. The objective of the robot is to track a group of fish, while preserving an acceptable localization accuracy. Note that even though the formulation of problem is general, we relate to this motivating example to help to better explain our methods.

For many robotic applications, the environmental dynamics is not known a priori. However, historical data collected from either simulation or past experiments might be available. Hence in this work we aim to design optimal policies by learning the system's behavior directly from past trajectories, and then leverage automated planning to address the decision-making problem. Rather than attempting to learn a parametric model of the state equation and belief dynamics, we only assume that a set of simulated historical system trajectories are available  $\kappa = \{\{r_i(k)\}_{k=0}^T, \{u_i(k)\}_{k=0}^T\}_{i=1}^M$  under the influence of the process noise  $\omega_k$ , where  $r_i(k)$  is the  $i^{\text{th}}$  system state at timestep  $k$ , and  $u_i(k)$  is the corresponding action performed. The structure and parameters of the system dynamics (6.1) are not assumed to be known.

To address the planning problem, we first establish an approximation model of the belief dynamics directly from the past simulated trajectories  $\kappa$ , based on the M-Z equation, by decomposing the problem into three subsequent problems: **i)** the partitioning of the continuous belief state into a discretized distribution with a finite number of cells, **ii)** the

identification of the transition probabilities between the cells, and **iii**) the transformation of the cell distribution to a symbolic representation for planning purposes.

### *Finite partitioning of the belief state*

The belief state  $b_k$  defined in (6.3) is a recursively updated continuous distribution, leading to an intractable computation of an optimal input  $u_k$ . As such, we are interested in computing an abstraction of  $b_k$  that makes computing an optimal  $u_k$  tractable. In addition, while other methods in the literature have utilized partitioning to make computing an optimal  $u_k$  feasible, such methods are subject to scaling issues as the dimensionality of the system increases. However, in many robotic applications, large portions of the belief space are completely unutilized. Instead, we seek to find a partitioning that focuses a fixed number of  $K$  partitions in regions of the belief space most frequently visited in order to ensure our representation achieves a close approximation of the belief state without scaling exponentially as the dimensionality of the system increases.

We assume that we are provided a set of zero-input trajectories  $\{(\bar{x}_m, \bar{u}_m = 0)\}_{m=1}^M$  drawn from a distribution of initial states  $x_0$  and simulated forward using system (6.1). Note here that the zero-input trajectories reveal information about the system dynamics as our input to the system is additive as shown in (6.1).

**Problem 6.1.1** (Finite Partitioning). *Given the set of offline simulated trajectories  $\{(\bar{x}_m, \bar{u}_m = 0)\}_{m=1}^M$ , find a partitioning of the belief state  $b$  denoted  $\{\mathcal{R}_i\}_{i=1}^K$ . The union of the partitions cover the portion of the state space that can be reached by the zero-input trajectory.*

### *Identification of the partitioned belief dynamics*

Having found a focused partitioning, we define a set of basis functions denoting the discretization. Let  $m(\mathcal{R}_i)$  denote the mass of partitioned cell  $\mathcal{R}_i$ , we define  $\phi^i$  as in (6.16), and  $\Phi = [\phi^1, \dots, \phi^K]^\top$  is the finite-dimensional basis function for parameterizing the belief state. We define  $\theta_k^i$  as the probability that the system state  $x_k$  is inside the cell  $\mathcal{R}_i$ ,  $\theta_k^i = \int_{\mathcal{R}_i} b_k(x) dx$ ,

and  $\Theta_k = [\theta_k^1, \dots, \theta_k^K]^\top$ . Then the projection operator  $\mathcal{P}$  in (6.9) from the full belief state to the resolved state can be found as:

$$\mathcal{P}b(x) = \sum_{i=1}^K \frac{\mathbb{1}(x \in \mathcal{R}_i)}{m(\mathcal{R}_i)} \int_{\mathcal{R}_i} bdu = \Theta^\top \Phi(x). \quad (6.20)$$

Similarly, we project the offline prior state on the basis functions:

$$\mathcal{P}p(x) = \Theta^{p\top} \Phi(x). \quad (6.21)$$

Having found a focused partitioning, with the initial resolved state distribution  $\Theta_0 = \mathcal{P}b_0$ , we then seek to derive a dynamics model for the resolved state, based on the M-Z equation. Suppose we consider modeling the M-Z equation memory effect up to order  $N$ . Let  $\bar{\Theta}_{k,N}^p = [\Theta_{k-N}^p, \Theta_{k-N+1}^p, \dots, \Theta_k^p]^\top$  be the offline simulated prior state projected on the basis functions, and  $\bar{u}_{k,N} = [u_{k-N}, u_{k-N+1}, \dots, u_k]^\top$ , then we are interested in identifying the evolution of  $\Theta_k^p$  for each sequence of discrete actions  $\bar{u}_{k,N}$ . Next we derive the dynamics of  $\Theta_k^p$  based on the M-Z formalism (6.14). As shown in (6.18), the transition of the best guess term in (6.14) can be represented as a Markovian transition matrix  $T_\theta$ , and can be identified by the GCM method, as described in (6.19). For the non-Markovian terms in (6.14), inserting (6.21) into the M-Z equation (6.14) yields dynamics of  $\Theta_k^p$ :

$$\Theta_{k+1}^p = T_\theta(u_k) \Theta_k^p + g_m(\bar{\Theta}_{k-1,N}^p, \bar{u}_{k-1,N}) + g_n(\mathcal{Q}p_0), \quad (6.22)$$

where  $g_m(\bar{\Theta}_{k-1,N}^p, \bar{u}_{k-1,N})^\top \Phi = \mathcal{P}\mathcal{T}_k \sum_{n=k-1-N}^{k-1} G(k,n) \Theta_n^{p\top} \Phi$  represents the memory term in (6.14), and  $g_n$  is the noise term, where  $g_n(\mathcal{Q}p_0)^\top \Phi = \mathcal{P}\mathcal{T}_k G(k,n) \mathcal{Q}p_0$ . However, since  $\mathcal{Q}p_0$  is a continuous distribution, it is infinite dimensional and is difficult to compute numerically. Hence, we only aim to identify  $g_m$  in the above equation for every possible sequence of control input applied in  $[k-N, k]$ . Let  $\bar{\mathcal{U}} = \{\bar{u}^i\}_{i=1}^{N \times |\mathcal{U}|}$  denote the set of all possible sequence of control input in  $N$  timesteps, where  $\bar{u}^i = [u_{k-N}, u_{k-N+1}, \dots, u_k]^\top, \forall k$ .

We assume that for each  $\bar{u}^i \in \bar{U}$ , a set of trajectories  $\kappa = \{\{r_k^i\}_{k=1}^N, \bar{u}^i\}_{i=1}^M$  drawn i.i.d. from a distribution of initial states  $x_0$  and simulated forward using system (6.1) for  $N$  timesteps are available, where the control input in the  $N$  timesteps is set as  $\bar{u}^i$ .

**Problem 6.1.2** (Finite Belief Dynamics). *For  $\bar{u}^i \in \bar{U}_N$ , given a set of trajectories  $\kappa(\bar{u}^i)$ , estimate the Markovian transition matrix on the partitioned belief space  $T_\theta$ , and the non-Markovian dynamics  $g$  that describes the evolution of the partitioned prior dynamics as a function of the applied input.*

## 6.2 Focused finite partitioning

In this section, we discuss our approach to solving Problem 6.1.1. To identify the optimal cell partition, we use a finite number of particles to approximate the dynamics of the belief state. Suppose  $\{r_k^{(i)}\}_{i=1}^M$  are drawn i.i.d. from a probability distribution  $p_k$ , and each particle has equal weight  $\omega = \frac{1}{M}$ . Then  $p_k(x)$  can be approximated by the probability mass function

$$p'_k(x) = \sum_{i=1}^M \omega \delta(x - r_k^{(i)}), \quad (6.23)$$

where  $\delta$  denotes the Kronecker delta function. At each timestep, the particle position is updated according to (6.1). We simulate the particles for  $k_N$  steps forward in time. By the off-line Monte Carlo simulation,  $p'_k$  is an approximation of the true prior distribution. Hence we derive the partition of the state space given the simulated particle trajectory.

Let  $K$  denote the total number of partitioned cells, and let  $\{\mathcal{R}_i\}_{i=1}^K$  represent the partitioned cells,  $\cup_{i=1}^K \mathcal{R}_i = \mathcal{D}$ . We use simulated trajectory at all timesteps during the Monte-Carlo simulation to derive the partitions. Let  $Z = \cup_{k=1}^n \cup_{m=1}^M r_k^{(m)}$  denote the set of all simulated particle positions at each timestep.  $Z$  is all the particle trajectories compressed into one snapshot, as shown in Fig. 6.2.

The partitioned prior state should well approximate the actual prior state  $p_k$ . Hence we formulate the following optimization problem to minimize the difference between the

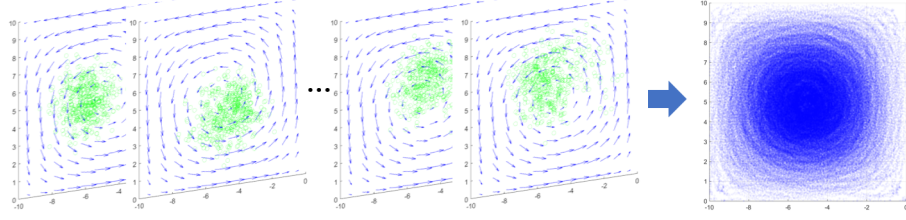


Figure 6.2: Illustration of the focused finite partition algorithm. We stack up the particle trajectories at all timesteps in the Monte-Carlo simulation, and use it to solve (6.24).

simulated particle position at all timesteps and the centroid of each cell,

$$\min_{\{\mathcal{R}\}_{i=1}^K} \sum_{i \in [1, K]} \sum_{z \in Z} \|z - \bar{r}^{(i)}\|^2 \quad (6.24)$$

where  $\bar{r}^{(i)}$  is the centroid of cell  $i$ .

Problem (6.24) can be solved by the K-means algorithm. We start by randomly selecting  $K$  cell centroids in the state space, and then use Lloyd iterations to solve the optimization problem. The Lloyd iteration contains two steps, the first of which is to assign the point  $z \in Z$  that are closest to a centroid to that centroid,

$$c(z) = \arg \min_{i \in [1, K]} \|z - \bar{r}^{(i)}\|^2, \quad (6.25)$$

where  $c(z) \in [1, K]$  is the index of the partition that datapoint  $z$  is assigned to. The second step is recomputing the cell centroid by taking average of all the data points assigned to this centroid.

$$\bar{r}^{(i)} = \frac{\sum_{z \in Z} \mathbb{1}(c(z) = i)z}{\sum_{z \in Z} \mathbb{1}(c(z) = i)}. \quad (6.26)$$

These two steps are repeated until cell membership no longer changes.

### 6.3 Identification of the partitioned belief dynamics

Leveraging the solution to Problem 6.1.1 designed in Section 6.2, we now discuss our solution to Problem 6.1.2.

---

**Algorithm 6:** Focused finite partition

---

**Data:** simulation time horizon  $n$ , number of simulated particles  $M$ , number of partitioned cells  $K$ , simulated zero-input trajectory  $\{r_k^{(m)}\}_{m=1}^M$   
**Output:** partitioned cells  $\{\mathcal{R}_i\}_{i=1}^K$

- 1  $Z = \cup_{k \in [1, n]} \cup_{m \in [1, M]} r_k^{(m)}$ ;
- 2 Randomly initialize cluster centroids  $\bar{r}^{(i)}, i \in [1, K]$ ;
- 3 **while not converges do**
- 4     For  $z \in Z$ , assign  $z$  to the centroid that is closest to it using (6.25) ;
- 5     Recompute all centroid using (6.26) ;
- 6 **end**
- 7 For all  $i \in [1, K]$ ,  $\mathcal{R}_i = \cup_{z \in Z} (\mathbb{1}\{c(z) = i\}z)$ ;

---

---

**Algorithm 7:** Identification of the M-Z equation

---

**Data:** Length of memory terms  $N$ , number of simulated particles  $M$ , GCM identified  $T_\theta(u)$   
**Output:** LSTM based model of the M-Z equation

- 1 Initialize particles  $\kappa_0 = \{r_0^i\}_{i=1}^M$  by drawing i.i.d. from probability distribution ;
- 2 Initialize  $\bar{U}_1 = \mathcal{U}$ ;
- 3 **for**  $k \in [1, k_{\text{train}}]$  **do**
- 4     **for**  $u_j \in \mathcal{U}$  **do**
- 5          $\bar{u} = \{\bar{u}, u_j\}$ ;
- 6         For each particle in  $\kappa_{k-1}$ , apply  $u_j$  and simulate it forward for one timestep by (6.1), and concatenate the updated particle position to  $\kappa_{k-1}$  to get the updated trajectory  $\kappa(\bar{u})$ ;
- 7     **end**
- 8 **end**
- 9 **for**  $\bar{u}_i \in \bar{U}$  **do**
- 10     **for**  $k = 1$  to  $k_{\text{train}}$  **do**
- 11         Compute  $\theta_k^i = \frac{1}{M} \sum_{m=1}^M \mathbb{1}(r_k^{(m)} \in \mathcal{R}_i)$  for all  $k$ ;
- 12          $\Theta_k = [\theta_k^1, \dots, \theta_k^K]^\top$ ;
- 13          $\mathbf{X}_k = \{\Theta_k, \Theta_{k-1}, \dots, \Theta_0\}$ ;
- 14          $\mathbf{Y}_k = \Theta_k - T_\theta(u_i)\Theta_{k-1}$ ;
- 15     **end**
- 16     LSTMnet = Train\_LSTM( $\{\mathbf{X}_k\}_{k=1}^{k_{\text{train}}}$ ,  $\{\mathbf{Y}_k\}_{k=1}^{k_{\text{train}}}$ );
- 17 **end**

---

### 6.3.1 Learning the memory terms via LSTM

The non-Markovian memory terms with time-lagged state information lead us to the use of Long Short-Term Memory (LSTM) [83]. LSTM has been widely used in modeling time-series, since it can capture the non-Markovian time dependency in sequence of data, and it avoids the vanishing gradient problem, unlike other Recurrent Neural Networks [84].

If we fix the control input to a sequence  $\bar{u} \in \bar{\mathcal{U}}$  in (6.22), the memory kernel  $G(k, n)$  depends only on the time-delay coefficient  $k - n$ , and is independent of the current timestep  $k$ . This form of the M-Z equation aligns with the time-delayed neural ordinary equation (TD-NODE):

$$\begin{cases} \Theta_{k+1}^p = T_\theta(u_k)\Theta_k^p + g_{\text{LSTM}}(\bar{\Theta}_{k,N}^p, \bar{u}_{k,N} = \bar{u}; \nu), & k \geq k_0 \\ \Theta_0^p = \Gamma_0, & \text{otherwise} \end{cases}, \quad (6.27)$$

where  $\nu$  are the unknown parameters of the neural network, and  $N \in \mathbb{N}$  represents the order of delay embedding.  $\Gamma_0$  is the initialization function at the initial timestep. The initialization function can be found by projecting the initial distribution of the system state onto the partitions  $\Phi$ . We use  $g_{\text{LSTM}}$  to denote the LSTM network. Specifically, one cell of the LSTM network can be described as:

$$\begin{aligned} q_k &= \sigma_g(W_{xf}\Theta_k + W_{hf}h_{k-1} + b_f) \\ i_k &= \sigma_g(W_{xi}\Theta_k + W_{hi}h_{k-1} + b_i) \\ o_k &= \sigma_g(W_{xo}\Theta_k + W_{ho}h_{k-1} + b_o) \\ \tilde{C}_k &= \sigma_c(W_{x\tilde{C}}\Theta_k + W_{h\tilde{C}}h_{k-1} + b_{\tilde{C}}) \\ C_k &= q_k \circ C_{k-1} + i_k \circ \tilde{C}_{k-1} \\ h_k &= o_k \circ \sigma_c(C_k), \end{aligned} \quad (6.28)$$

where  $\sigma_g$  denotes the sigmoid function,  $\sigma_c$  is the hyperbolic tangent function, and  $\circ$  denotes element-wise multiplication.  $\Theta_k$  is the input for the current LSTM block at time  $k$ ,  $h_k$  is the

hidden states at time  $k$ , and  $C_k$  are cell states.  $q_k$ ,  $i_k$  and  $o_k$  are outputs of forget gate layer, input gate layer and output gate layer respectively.

**LSTM training** We train the TD-NODE to obtain a M-Z based model of the resolved state by using data from off-line high-fidelity simulations. First we discuss how to generate the training data from Monte-Carlo simulation. In order to learn the memory terms, for each possible combination of control input applied to the system in  $N$  timesteps, a set of Monte-Carlo simulation trajectories need to be collected. Going through all elements in  $\bar{U}$  in a brute force way can be computationally expensive. To save computation cost, at each timestep, we apply  $u_i \in \mathcal{U}$ , and concatenate the new trajectories with the historical trajectory.

Suppose we draw  $\{r_0^{(i)}\}_{i=1}^M$  i.i.d. from the initial distribution  $\rho_0$ , and simulate the particles forward in time with the system dynamics (6.1) with the control input being  $\bar{u}$ . The simulated particles form distribution  $p'_k(\bar{u})$ . Then at each timestep of the off-line simulation, we compute the difference between the full state  $\mathcal{P}p'_k$  and the best guess term, and solve a regression problem to minimize the difference between the ground truth and the TD-NODE output. Let  $\mathbf{X}_k(\bar{u}) = \{\bar{\Theta}_{k,N}^p\}$ , and  $\mathbf{Y}_k(\bar{u}) = \{\Theta_{k+1}^p - T_\theta(u_k)\Theta_k^p\}$ , then  $\{\mathbf{X}_k, \mathbf{Y}_k\}_{k=0}^{k_{\text{train}}}$  are the input-output pairs of the training dataset. We formulate the regression problem as follows, with the loss function  $l$  being the  $L_1$  mean absolute error:

$$\min_{\nu} \sum_{k=0}^{k_{\text{train}}} l(g_{\text{LSTM}}(\mathbf{X}_k, \bar{u}; \nu), \mathbf{Y}_k(\bar{u})). \quad (6.29)$$

The algorithm for identifying the transition dynamics in Algorithm 7.

### 6.3.2 Bayesian update in the partitioned state space

Given the modeling of prior dynamics in Section 6.3.1, we now discuss the on-line Bayesian update in the partitioned state space. The true likelihood function is approximated by

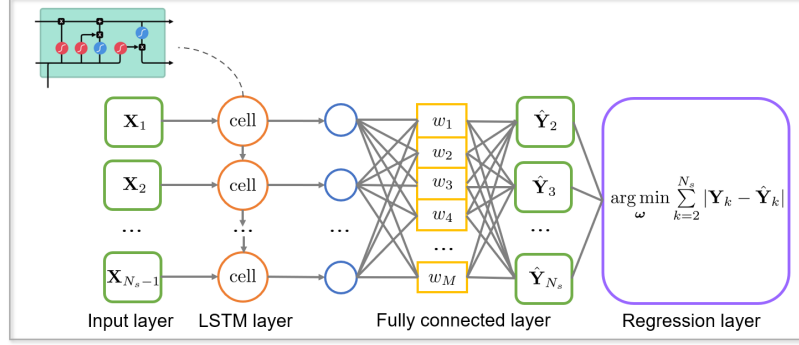


Figure 6.3: Graphical representation of the M-Z based TD-NODE for modeling the memory effect. Each cell in the LSTM layer represents an LSTM cell described by (6.28).

projecting it onto the partitioned cells:

$$\mathcal{P}\Psi_k(x) = \sum_{i=1}^K \frac{\mathbb{1}(x \in \mathcal{R}_i)}{m(\mathcal{R}_i)} \int_{\mathcal{R}_i} \Pr(y_k|x) dx. \quad (6.30)$$

At timestep  $k$ , let  $\xi_k^i$  be the approximated likelihood function value in each partitioned region,  $i \in [1, K]$ , and  $\xi_k = [\xi_k^1, \xi_k^2, \dots, \xi_k^K]^\top$ , then  $\mathcal{P}\Psi_k = \xi_k^\top \Phi$ . By defining the approximated likelihood function we have an approximated Bayesian updating law, which we denote as the operator  $\hat{\mathcal{L}}_k : \Theta_k^{(-)\top} \Phi \rightarrow \Theta_k^\top \Phi$ :

$$\begin{aligned} \hat{\mathcal{L}}_k \Theta_k^{(-)\top} \Phi(x) &= \frac{\Theta_k^{(-)\top} \Phi(x) \xi_k^\top \Phi(x)}{\langle \Theta_k^{(-)\top} \Phi, \xi_k^\top \Phi \rangle} \\ &= \frac{\Theta_k^{(-)\top} \Phi(x) \xi_k^\top \Phi(x)}{\sum_{i=1}^K \frac{\theta_k^{(-)i} \xi_k^i}{m(\mathcal{R}_i)}}. \end{aligned} \quad (6.31)$$

Note that  $\hat{\mathcal{L}}_k$  is an approximation of the true Bayesian update  $\mathcal{L}_k$ . Let  $\Theta_k^\top \Phi = \hat{\mathcal{L}}_k \Theta_k^{(-)\top} \Phi(x)$  be the posterior distribution updated by the approximated Bayesian law. Simplifying (6.31) yields:

$$\Theta_k = \frac{\Theta_k^{(-)} \circ \xi_k \circ \kappa^{\circ-1}}{(\Theta_k^{(-)} \circ \kappa^{\circ-\frac{1}{2}})^\top (\xi_k \circ \kappa^{\circ-\frac{1}{2}})}, \quad (6.32)$$

where  $\kappa = [m(\mathcal{R}_1), m(\mathcal{R}_2), \dots, m(\mathcal{R}_K)]^\top$ .  $\circ$  denotes the element-wise Hadamard product, and  $(\cdot)^{\circ-1}$  is the element-wise Hadamard inverse. For convenience of describing the belief

dynamics in the partitioned state space, we define an operator  $L_k : \Theta_k^{(-)} \rightarrow \Theta_k$  to describe the parameter space Bayesian law (6.32).

### 6.3.3 Online M-Z formalism based belief dynamics

Combining the offline identified LSTM based model of the M-Z equation memory effect and the Bayesian update in the partitioned state space, we now develop the online approximated propagation of the belief dynamics in the partitioned state space. The exact belief state updating law is given in (6.15). However, the memory terms in (6.15) cannot be formulated as a TD-NODE and be identified using the LSTM, since the memory kernel  $\tilde{G}(k, n)$  in (6.15) contains  $\mathcal{L}_k$ , which depends on the historical observation. Hence, we approximate  $\tilde{G}(k, n)$  with  $G(k, n)$  to derive the following updating laws. The online reduced-order non-Markovian prior dynamics is designed as:

$$\begin{cases} \Theta_{k+1}^{(-)} = T_\theta(u_k)\Theta_k + g_{\text{LSTM}}(\bar{\Theta}_{k,N}, \bar{u}_{k,N}), & \text{if } k > k_{\text{train}} \\ \Theta_k = \Theta_k^p, & \text{otherwise} \end{cases}, \quad (6.33)$$

where  $\bar{\Theta}_{k,N} = [\Theta_{k-N}, \Theta_{k-N+1}, \dots, \Theta_k]^\top$ , and each element of  $\bar{\Theta}_{k,N}$  is computed by (6.32). Hence the online belief dynamics in parameter space is derived by combining (6.33) and (6.32):

$$\Theta_{k+1} = L_k T_\theta(u_k) \Theta_k^{(-)} + L_k g_{\text{LSTM}}(\bar{\Theta}_{k,N}, \bar{u}_{k,N}; \nu). \quad (6.34)$$

## 6.4 Theoretical analysis

The proposed algorithm develops a reduced-order approximation model of the belief dynamics, which raises the following question: how well does the approximation model match with the true belief state? To answer this question, we first mathematically formulate the problem, then provide theoretical analysis on the error bound. In this section, we consider the observation process takes a sequence of arbitrary but fixed value.

$$\begin{aligned}
\text{Full dynamics:} \quad & b_{k-1} \xrightarrow{\text{predict}} b_k^{(-)} \xrightarrow{\text{update}} b_k \\
\text{Resolved dynamics:} \quad & \Theta_{k-1}^\top \Phi \xrightarrow{\text{predict}} \Theta_k^{(-)\top} \Phi \xrightarrow{\text{update}} \Theta_k^\top \Phi
\end{aligned}$$

Figure 6.4: Comparison between the full belief dynamics and the reduced-order approximation model in one iteration of the estimation process. The full dynamics includes the prediction step and the updating step (the Bayesian law). The approximation model consists of prediction using the TD-NODE model and the update step by the parameter space Bayesian law (6.32).

#### 6.4.1 Model reduction error

We define the model reduction error as the difference between the true belief state and the resolved state computed from (6.34):

$$e_k = \|b_k - \Theta_k^\top \Phi\|_\infty, \quad (6.35)$$

where  $\|\cdot\|_\infty$  is the supremum norm, i.e.,  $\|b\|_\infty = \sup_{x \in \mathbb{R}^n} |b(x)|$ ,  $b \in B(\mathbb{R}^n)$ .

#### 6.4.2 Error bound analysis

To derive an upper bound on  $e_k$ , we decompose  $e_k$  into several components, and find an upper bound for each of the components,

$$b_k - \Theta_k^\top \Phi = (b_k - \mathcal{L}b_k^{(-)'}) + (\mathcal{L}_k b_k^{(-)' } - \mathcal{L}_k \Theta_k^{(-)\top} \Phi) + (\mathcal{L}_k \Theta_k^{(-)\top} \Phi - \hat{\mathcal{L}}_k \Theta_k^{(-)\top} \Phi), \quad (6.36)$$

where  $b_k^{(-)' } = \mathcal{T}_k(\mathcal{L}_{k-1} \mathcal{T}_{k-1}) \dots (\mathcal{L}_{k_{\text{train}}+1} \mathcal{T}_{k_{\text{train}}+1}) \mathcal{L}_{k_{\text{train}}} b_{k_{\text{train}}}^{(-)' }$ . The first term results from the Monte-Carlo simulation in the training phase, where  $b_k^{(-)} - b_k^{(-)' }$  is the error between the true prior distribution and the prior distribution formed by the Monte-Carlo simulation samples, propagated from timestep  $k_{\text{train}}$  to timestep  $k$ . The second term  $\mathcal{L}_k b_k^{(-)' } - \mathcal{L}_k \Theta_k^{(-)\top} \Phi$  is the error due to the piece-wise constant representation of the prior distribution and its

dynamics. The last term results from the approximated Bayesian update in partitioned state space. Since the Monte-Carlo simulation is performed only for a finite number of timesteps for data collection, we assume the error due to Monte-Carlo simulation, which is the first term in (6.36), is bounded by  $e_{mc}$ . Then we compute upper bound for the second and third terms in (6.36). For this reason we introduce the following assumptions.

**Assumption 6.4.1.** *We assume that for any distribution  $b$ ,  $\|\mathcal{P}b - b\|_\infty$  is bounded above by  $e_p$ . Moreover, we assume for any likelihood function  $\Psi$ ,  $\|\mathcal{P}\Psi - \Psi\|_\infty$  is bounded above by  $e_l$ .*

**Remark 6.4.1.** *For any two distributions, the supreme norm distance must be less than or equal to 1. Hence there exists  $e_p, e_l$  that bounds the projection error. For distributions that are Lipschitz continuous and differentiable, a tighter upper bound on the projection error can be derived, as shown in [85].*

**Assumption 6.4.2.** *The LSTM approximates arbitrary number of memory terms in the M-Z equation with error bounded by a constant value  $e_L$ :*

$$\|\mathcal{PT}_k \sum_{n=0}^{k-1} \tilde{G}(k, n) \mathcal{P}b_n - g_{\text{LSTM}}(\bar{\Theta}_{k,N}, \bar{u}_{k,N}; \nu)^\top \Phi\|_\infty \leq e_L.$$

**Remark 6.4.2.** *Assumption 6.4.2 is our main assumption. It has been shown that Recurrent Neural Networks are universal approximators [86]. Hence we assume that the LSTM has enough breadth and depth to approximate the M-Z memory term with a constant error bound. In addition, we provide validation of this assumption in the simulation section.*

**Proposition 6.4.1.** *The memory kernel of M-Z equation is contractive: for any probability distribution function  $b$ ,  $\|\mathcal{Q}b\|_\infty \leq \|b\|_\infty$ .*

*Proof.* First, the Markov operators are contractions. By Prop. 3.1.1 in [87], for any probability distribution function  $b$ ,  $\|\mathcal{T}b\|_\infty \leq \|b\|_\infty$ . Next we show that the operator  $\mathcal{Q}$  is

contractive,  $\|Qb\|_\infty \leq \|b\|_\infty$ . Let us define two operators

$$(\cdot)^+ = \sup(0, \cdot), \quad (\cdot)^- = \inf(0, \cdot).$$

First, notice that  $(Qb(x))^+ \leq b^+(x)$  since

$$\begin{aligned} (Qb(x))^+ &= ((b(x) - \mathcal{P}b(x))^+ \\ &= \sup(0, \sup(0, b(x) - \mathcal{P}b(x))) - \inf(0, b(x) - \mathcal{P}b(x)) \\ &\leq \sup(0, b(x)) \\ &= b(x)^+. \end{aligned}$$

Similarly,  $(Qb(x))^- \geq b(x)^-$ . By the definition of the supreme norm, we have  $\|Qb\|_\infty \leq \|b\|_\infty$ , and thus  $QT$  is contractive.  $\square$

Now we present Lemma 6.4.1 to bound the second term in (6.36).

**Lemma 6.4.1.**  $\|\mathcal{L}(b_k^{(-)'} - \Theta_k^{(-)\top}\Phi)\|_\infty \leq 3\delta_1(e_l + e_p)\|\Psi_k\|_\infty, \forall k \in \mathbb{N}$ .

*Proof.* We derive the upper bound of error between the prior distribution  $b_k^{(-)'}$  and  $\Theta_k^{(-)\top}\Phi$  by decomposing it into two terms:

$$\begin{aligned} \|b_k^{(-)' - \Theta_k^{(-)\top}\Phi}\|_\infty &\leq \|b_k^{(-)' - \mathcal{P}b_k^{(-)'}\|_\infty + \|\mathcal{P}b_k^{(-)' - \Theta_k^{(-)\top}\Phi}\|_\infty \\ &\leq e_p + \|\mathcal{P}b_k^{(-)' - \Theta_k^{(-)\top}\Phi}\|_\infty. \end{aligned}$$

The first term is due to projecting the distribution  $b_k^{(-)'}$  onto the partitions. The second term is due to the difference between the true partitioned belief dynamics and the LSTM modeled dynamics. Substituting the modeled dynamics of  $\Theta_k^{(-)\top}\Phi$  in the above equation with (6.33), and replacing the true reduced dynamics  $\mathcal{P}b_k^{(-)'}$  in the above equation with the M-Z equation

(6.14) leads to:

$$\begin{aligned} \|\mathcal{P}b_k'^{(-)} - \Theta_k^{(-)\top}\Phi\| &= \|(T_\theta(u_{k-1})\Theta_{k-1} + \mathcal{PT}_k \sum_{n=0}^{k-2} \tilde{G}(k, n)(\Theta_n^\top\Phi) + \mathcal{PT}_{k-1}\tilde{G}(k-1, 0)\mathcal{Q}b_0' \\ &\quad - (T_\theta(u_{k-1})\Theta_{k-1} - g_{\text{LSTM}}(\bar{\Theta}_{k-1, N}; \bar{u}_{k-1, N}))^\top\Phi\|_\infty. \end{aligned}$$

Because of Assumption 6.4.2, and by Proposition 6.4.1, the above equation has an upper bound as follows:

$$\begin{aligned} \|\mathcal{P}b_k'^{(-)} - \Theta_k^{(-)\top}\Phi\|_\infty &= \|e_L + \mathcal{PT}_k G(k, 0)\mathcal{Q}b_0'\|_\infty \\ &\leq e_L + \|\mathcal{PT}_k \mathcal{Q}b_0'\|_\infty \\ &\leq e_L + e_p. \end{aligned}$$

Next we find the error bound after the Bayesian updating step. Note that

$$\begin{aligned} \|\mathcal{L}_k b_k'^{(-)} - \mathcal{L}_k \Theta_k^{(-)\top}\Phi\|_\infty &= \left\| \frac{\Psi_k b_k'^{(-)}}{\langle \Psi_k, b_k'^{(-)} \rangle} - \frac{\Psi_k \Theta_k^{(-)\top}\Phi}{\langle \Psi_k, \Theta_k^{(-)\top}\Phi \rangle} \right\|_\infty \\ &\leq \left\| \frac{\Psi_k b_k'^{(-)}}{\langle \Psi_k, b_k'^{(-)} \rangle} - \frac{\Psi_k b_k'^{(-)}}{\langle \Psi_k, \Theta_k^{(-)\top}\Phi \rangle} \right\|_\infty \\ &\quad + \left\| \frac{\Psi_k b_k'^{(-)}}{\langle \Psi_k, \Theta_k^{(-)\top}\Phi \rangle} - \frac{\Psi_k \Theta_k^{(-)\top}\Phi}{\langle \Psi_k, \Theta_k^{(-)\top}\Phi \rangle} \right\|_\infty \tag{6.37} \\ &= \left\| \frac{\Psi_k b_k'^{(-)} \langle \Psi_k, \Theta_k^{(-)\top}\Phi \rangle - \Psi_k b_k'^{(-)} \langle \Psi_k, b_k'^{(-)} \rangle}{\langle \Psi_k, b_k'^{(-)} \rangle \langle \Psi_k, \Theta_k^{(-)\top}\Phi \rangle} \right\|_\infty \\ &\quad + \left\| \frac{\Psi_k (b_k'^{(-)} - \Theta_k^{(-)\top}\Phi)}{\langle \Psi_k, \Theta_k^{(-)\top}\Phi \rangle} \right\|_\infty \end{aligned}$$

Then we find the upper bound of the two terms in (6.37). Let  $\epsilon = \Theta_k^{(-)\top}\Phi - b_k'^{(-)}$ . Since the denominator of the full state Bayesian update  $\langle \Psi_k, b_k'^{(-)} \rangle$  must be positive, there must exist  $x \in \mathcal{D}$  such that  $\Pr(y_k|x) \neq 0$ , meaning that  $\Psi_k$  is not all zero in the domain. Hence there

exists  $\delta_1 > 0$  such that  $\langle \Psi_k, \Theta_k^{(-)\top} \Phi \rangle \geq \frac{1}{\delta_1}$ . For the first term, note that

$$\begin{aligned}
& \left\| \frac{\Psi_k b_k'^{(-)} \langle \Psi_k, \Theta_k^{(-)\top} \Phi \rangle - \Psi_k b_k'^{(-)} \langle \Psi_k, b_k'^{(-)} \rangle}{\langle \Psi_k, b_k'^{(-)} \rangle \langle \Psi_k, \Theta_k^{(-)\top} \Phi \rangle} \right\|_\infty \\
&= \left\| \frac{\Psi_k (\langle \Psi_k b_k'^{(-)}, \epsilon \rangle - \epsilon \langle \Psi_k, b_k'^{(-)} \rangle)}{\langle \Psi_k, b_k'^{(-)} \rangle \langle \Psi_k, \Theta_k^{(-)\top} \Phi \rangle} \right\|_\infty \\
&\leq \left\| \frac{\Psi_k \langle \Psi_k, b_k'^{(-)} \rangle}{\langle \Psi_k, b_k'^{(-)} \rangle \langle \Psi_k, \Theta_k^{(-)\top} \Phi \rangle} \right\|_\infty \|\epsilon\|_\infty + \left\| \frac{\Psi_k \epsilon}{\langle \Psi_k, \Theta_k^{(-)\top} \Phi \rangle} \right\|_\infty \\
&\leq 2\delta_1(e_l + e_p) \|\Psi_k\|_\infty. \tag{6.38}
\end{aligned}$$

Similarly, the second term in (6.37) can be bounded by:

$$\left\| \frac{\Psi_k (b_k'^{(-)} - \Theta_k^{(-)\top} \Phi)}{\langle \Psi_k, \Theta_k^{(-)\top} \Phi \rangle} \right\|_\infty \leq \delta_1(e_l + e_p) \|\Psi_k\|_\infty. \tag{6.39}$$

Hence, combining (6.38) and (6.39) yields  $\|\mathcal{L}b_k^{(-)'} - \mathcal{L}\Theta_k^{(-)\top} \Phi\|_\infty \leq 3\delta_1(e_l + e_p) \|\Psi_k\|_\infty$ .  $\square$

Finally we bound the third term in (6.36), which is the error resulting from the projected Bayesian update.

**Lemma 6.4.2.** *For all  $k \in \mathbb{N}$ ,  $\|\hat{\mathcal{L}}\Theta_k^{(-)\top} \Phi - \mathcal{L}\Theta_k^{(-)\top} \Phi\|_\infty \leq \delta_2 e_l + (\delta_1 + \delta_2) \|\Psi_k\|_\infty$ .*

*Proof.*

$$\begin{aligned}
\|\hat{\mathcal{L}}\Theta_k^{(-)\top} \Phi - \mathcal{L}\Theta_k^{(-)\top} \Phi\|_\infty &\leq \left\| \frac{\Theta_k^{(-)\top} \Phi \xi_k^\top \Phi}{\langle \Theta_k^{(-)\top} \Phi, \xi_k^\top \Phi \rangle} - \frac{\Theta_k^{(-)\top} \Phi \Psi_k}{\langle \Theta_k^{(-)\top} \Phi, \xi_k^\top \Phi \rangle} \right\|_\infty \\
&\quad + \left\| \frac{\Theta_k^{(-)\top} \Phi \Psi_k}{\langle \Theta_k^{(-)\top} \Phi, \xi_k^\top \Phi \rangle} - \frac{\Theta_k^{(-)\top} \Phi \Psi_k}{\langle \Theta_k^{(-)\top} \Phi, \Psi_k \rangle} \right\|_\infty \\
&\leq \left\| \frac{\Theta_k^{(-)\top} \Phi (\xi_k^\top \Phi - \Psi_k)}{\langle \Theta_k^{(-)\top} \Phi, \xi_k^\top \Phi \rangle} \right\|_\infty + \left\| \frac{\Theta_k^{(-)\top} \Phi \Psi_k}{\langle \Theta_k^{(-)\top} \Phi, \xi_k^\top \Phi \rangle} \right\|_\infty \\
&\quad + \left\| \frac{\Theta_k^{(-)\top} \Phi \Psi_k}{\langle \Theta_k^{(-)\top} \Phi, \Psi_k \rangle} \right\|_\infty
\end{aligned}$$

As shown in the proof of Lemma (6.4.1), there exists  $\delta_1 > 0$  such that  $\langle \Psi_k, \Theta_k^{(-)\top} \Phi \rangle \geq \frac{1}{\delta_1}$ .

Further, since  $\Psi_k$  cannot be all zero in the domain,  $\xi_k$  is not an all-zero vector. Hence there exists  $\delta_2 > 0$  such that  $\langle \Theta_k^{(-)\top} \Phi, \xi_k^\top \Phi \rangle \geq \frac{1}{\delta_2}$ . Since the supremum norm of all distribution cannot be larger than 1, the first term in the above equation can be bounded as:

$$\begin{aligned} \left\| \frac{\Theta_k^{(-)\top} \Phi (\xi_k^\top \Phi - \Psi_k)}{\langle \Theta_k^{(-)\top} \Phi, \xi_k^\top \Phi \rangle} \right\|_\infty &\leq \delta_2 \|\Theta_k^{(-)\top} \Phi\| \|\xi_k^\top \Phi - \Psi_k\|_\infty \\ &\leq \delta_2 e_l. \end{aligned}$$

Hence we bound the error from the projected Bayesian filter as follows:

$$\|\hat{\mathcal{L}}\Theta_k^{(-)\top} \Phi - \mathcal{L}\Theta_k^{(-)\top} \Phi\|_\infty \leq \delta_2 e_l + (\delta_1 + \delta_2) \|\Psi_k\|_\infty.$$

□

Finally, we present the main result on error analysis of the proposed belief abstraction algorithm.

**Theorem 6.4.3.** *The model reduction error for  $k \in \mathbb{N}$  is bounded by*

$$e_k \leq e_{mc} + (3\delta_1(e_l + e_p + 1) + \delta_2(e_l + 1)) \|\Psi_k\|_\infty. \quad (6.40)$$

Note that the upper bound of model reduction error given in (6.40) does not grow with time. The reason is that by incorporating the LSTM to model the memory effects in the M-Z equation, the time-growing term in the model reduction error is driven to zero. The error analysis indicates that by constructing a TD-NODE we achieve an accurate abstracted belief dynamics model if the LSTM can model the M-Z equation to certain accuracy.

## 6.5 Simulation results

In this section, we first discuss utilizing the belief abstraction method to build logical representation of belief dynamics. Further we present simulation setup of a marine autonomy

application, and use the simulation example to demonstrate performance of the proposed algorithm.

### 6.5.1 Predicate transition

Section 6.3 provide us with a decomposition of the infinite dimensional belief space into a set of basis functions  $\Phi$  and an associated distribution  $\Theta$ . However, to solve planning problems with symbolic planning tools such as HTN or STRIPS (Stanford Research Institute Problem Solver) [88], we must construct a logical representation of the belief state. To do so, we must find a mapping from  $\Theta$  to a new set of predicates that discretizes how strongly we believe the system state lies within any particular partition in  $\Phi$ .

In this section, we define and describe the mapping function  $\Gamma$  to construct our current set of active predicates  $s_k$  given our distribution over  $\Phi$ ,  $\Theta_k$  which will allow us to solve planning problems. We design  $\Gamma$  to evaluate each predicate according to the following rules

$$\Gamma(\text{Bel}, \Theta_k) = \begin{cases} 1, & \text{Bel} = \text{BelH}_i, \Theta_k^{(i)} > p_H \\ 1, & \text{Bel} = \text{BelM}_i, p_H \geq \Theta_k^{(i)} > p_L \\ 1, & \text{Bel} = \text{BelL}_i, p_L \geq \Theta_k^{(i)} \\ 0, & \text{otherwise.} \end{cases} \quad (6.41)$$

Here, parameters  $p_H$  and  $p_L$  are threshold values to discretize the belief values. These values can be chosen to reflect the confidence precision necessary for satisfying desired planning goals. Applying (6.41) allows us to identify  $K$  active predicates that denote our confidence over each partition. We can then evaluate the predicates in  $\mathcal{S}$  at each iteration by enumerating over  $\text{BelH}_j, \text{BelM}_j, \text{BelL}_j$  for  $j \in \{1, \dots, K\}$  with  $\Gamma$ . Let  $s_k = \{s | \Gamma(s, \Theta_k) = 1\}$  be the current set of active predicates computed by enumerating over each  $\text{BelH}_j, \text{BelM}_j, \text{BelL}_j$  for  $j \in \{1, \dots, K\}$  with  $\Gamma$  and retaining a memory of the historical predicates. From any  $s_k$ , the effect of taking action  $u_k$  can be derived from (6.4) and (6.31). We can construct a

planning problem  $\Sigma = (\mathcal{S}, \mathcal{U}, \gamma, s_0, g)$  where  $g \in \mathcal{S}$  is a set of desired predicates we would like to activate, and  $\gamma : \mathcal{S} \times \mathcal{U} \rightarrow \mathcal{S}$  is a predicate transition function that determines the effects of taking an action  $u$  from a state  $s$ .

We can now construct  $\gamma$  based on a graph search using (6.41) to construct branches based on our action set  $\mathcal{U}$  where each branch cost is given by  $L(\Theta_k, u_k)$  and using precondition of actions to prune branches according to whether they satisfy our desired preconditions. Finding optimal sequences of actions can then be solved used an A\* search. In addition, during execution, the inclusion of additional acoustic beacon information allows us to update  $\Theta_k$  with  $y_k$  and possibly re-plan. An example of one iteration of our graph search is shown in Figure 6.5.

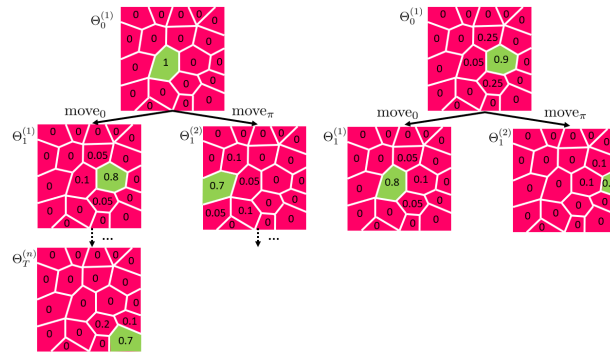


Figure 6.5: An illustration of the belief state evolution in an environment with 2 actions, move to the left ( $\text{move}_\pi$ ), and move to the right ( $\text{move}_0$ ). **(Left)**: The decision tree at the initial timestep. Each branch of the decision tree represents the agent taking a specific action. After the action is taken, the belief at the next timestep is computed by the MZ-based TD-NODE model. The decision tree construction is finished until a node fulfills the terminal constraint. **(Right)**: Updated belief state after a  $\text{move}_0$  is taken at the initial timestep, and a new observation is received, which confirms the state is contained in the green cell. After the Bayesian update, the new belief state is taken as the root of a new decision tree, and the agent takes the same procedure for decision tree construction to begin a new search.

In this work, we solve these planning problems using a HTN planner to handle the preconditions of taking actions while using an A\* planner to find the cost-optimal sequence of actions. Implementation of our symbolic dynamics approximation and path planning methods is presented in a simulated experiment of AUV deployment. Simulations of the proposed algorithms are performed, and its performance is compared with the Generalized

Cell Mapping algorithm.

### 6.5.2 Simulation setup

We evaluate performance of the proposed algorithm using a simulation of marine autonomy example, where the AUV is deployed to map and track the movement of acoustically tagged fish.

**Vehicle model** Let  $x_k \in \mathcal{D}$  be the three-dimensional position of the vehicle, where the domain is a compact set in  $\mathbb{R}^3$  defined as  $\mathcal{D} = [-10, 0] \times [0, 10] \times [-1, 1]$ . We let  $y_k \in \{0, 1\}^N$  denote the underwater observations from  $N$  acoustic beacons, and  $u_k \in \mathcal{U}$  is the controlled heading angle and depth change at timestep  $k$ . The vehicle's through-water speed is denoted as  $V \in \mathbb{R}$ . The AUV obeys the dynamics

$$x_k = F(x_k) + \begin{bmatrix} V(\cos(u_{1,k}) - \frac{\sqrt{2}}{2}|u_{2,k}|) \\ V(\sin(u_{1,k}) - \frac{\sqrt{2}}{2}|u_{2,k}|) \\ u_{2,k} \end{bmatrix} + w_k, \quad (6.42)$$

where the flow field, denoted as  $F : \mathbb{R}^3 \times \mathbb{R} \rightarrow \mathbb{R}^3$  is given by a gyre model,

$$\begin{aligned} F^{(1)}(x, k) &= -\pi A \sin(\pi g(x^{(1)}, k)) \cos(\pi x^{(2)}) \\ F^{(2)}(x, k) &= \pi A \cos(\pi g(x^{(1)}, k)) \sin(\pi x^{(2)}) \frac{dg}{dx^{(1)}} \\ F^{(3)}(x, k) &= 0, \end{aligned} \quad (6.43)$$

where  $g(x_1, k) = a(x^{(1)})^2 + bx^{(1)}$ . For observation model  $h$ , we assume several underwater acoustic receivers are installed in the domain. The receivers detect acoustic signals sent by the vehicle with a certain probability when in range of the receivers, i.e., for a receiver  $j$

located at position  $r_j$

$$\begin{aligned}\Pr(y_k^{(j)} = 1) &= p\mathbb{1}\{\|x_k - r_j\| \leq R_j\} + (1 - p)\mathbb{1}\{\|x_k - r_j\| > R_j\}, \\ \Pr(y_k^{(j)} = 0) &= (1 - p)\mathbb{1}\{\|x_k - r_j\| \leq R_j\} + p\mathbb{1}\{\|x_k - r_j\| > R_j\}.\end{aligned}\tag{6.44}$$

We assume that the AUV can apply 8 different move actions in the x-y plane and 2 actions to change between being at the surface and being at some depth, i.e.,  $\mathcal{U} = \mathcal{U}_{xy} \cup \mathcal{U}_z$  where  $\mathcal{U}_{xy} = \left\{ \begin{bmatrix} \frac{j\pi}{4} & 0 \end{bmatrix}^\top \right\}_{j=0}^7$  are the planar move actions and  $\mathcal{U}_z = \left\{ \begin{bmatrix} \frac{\pi}{4} & 1 \end{bmatrix}^\top, \begin{bmatrix} \frac{\pi}{4} & -1 \end{bmatrix}^\top \right\}$  are two depth changing actions, the first to enable surfacing and the second to enable diving. Notice that if any action in  $\mathcal{U}_{xy}$  is taken, the AUV will move along the planar direction with no change in altitude while if any action in  $\mathcal{U}_z$  is taken, then the AUV will change depth without changing its x-y position. While  $\mathcal{U}_{xy}$  actions are useful for reaching target destinations,  $\mathcal{U}_z$  actions enable the glider to surface in order to re-acquire an improved position estimate via GPS. In this work, we additionally assume that there is no process noise on the  $\mathcal{U}_z$  actions and we can always surface or dive within one timestep. Hence the system state can be represented as  $s_k = s_{xy,k} \cup s_{z,k}$ , where  $s_{xy,k} = \{s | \Gamma(s, \Theta_k) = 1\}$  is the distribution of vehicle horizontal position, and  $s_{z,k} \in \{\text{at\_surf}, \text{at\_depth}\}$  is the vehicle depth.

The effects of  $\mathcal{U}_{xy}$  are defined by the identified transition dynamics in Section 6.3 and the effects of  $\mathcal{U}_z$  are defined as an information gathering action for position estimation. Let  $j_{\max} = \arg \max \Theta^{(j)}$  be the index of the partition with the highest probability. For planning purposes, we assume that the effects of taking surfacing action  $u_9$  on the distribution  $\Theta$  are

$$\Theta^{(i)} = \begin{cases} 1, & i = j_{\max} \\ 0, & \text{otherwise} \end{cases}\tag{6.45}$$

while taking diving action  $u_{10}$  has no effect on  $\Theta$ . However, we assume that move actions

may only be performed at depth, yielding a precondition function

$$\text{Pre}(a_i, s_k) = \begin{cases} 1, & \text{if } s_{z,k} = \text{at\_depth}, i < 9 \\ 0, & \text{if } s_{z,k} = \text{at\_surf}, i < 9 \end{cases}. \quad (6.46)$$

**Hotspot model** It has been shown that the fish motion dynamics can be modeled by a  $2D$  random walk model [89]. Hence we assume that the hotspot obeys the dynamics

$$r_{k+1} = r_k + \gamma \begin{bmatrix} \cos \nu_k^2 \\ \sin \nu_k^2 \end{bmatrix}, \quad (6.47)$$

where  $\nu = [\nu^1, \nu^2]^\top$  is i.i.d. continuous random vector with known distribution. We assume the fishes are acoustically tagged, and can be detected by the receivers with the same acoustic detection model (6.44).

Let  $X_k = [x_k^\top, r_k^\top]^\top$ , we combine the vehicle dynamics and the hotspot motion as

$$X_{k+1} = F(X_k) + B(U_k) + C(W_k). \quad (6.48)$$

We consider there are 3 receivers in the domain, each with a different detection range and detection accuracy. The beacon placement and detection range setup is shown in Figure 6.6.

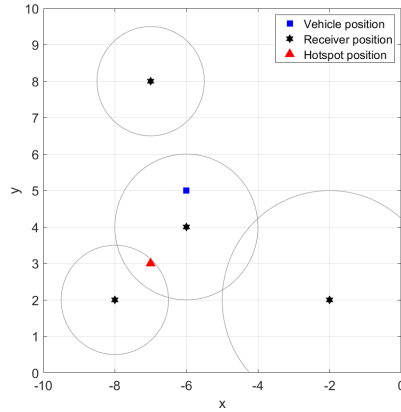


Figure 6.6: Setup of the simulation domain.

**Belief space planning** Given the vehicle and hotspot dynamics, we formulate the following problem to have the underwater vehicle localize and track the hotspot. As such, policies are found by minimizing the difference between the distribution of the vehicle position and the hotspot position, plus an additional term denoting the time cost. Let  $r_k^*$  denote the center of the cell which retains the highest probability of containing the hotspot position. The objective function is designed as  $L_k = \|\Theta_k^\top \Phi - \delta(r_k^*)\|^2 + C$ , where  $C$  is a constant variable denoting the travel time consumption factor in the objective function. This objective function incentivizes the vehicle to stay close to the possible location of the hotspot. We let  $C = 1$  for vehicle’s move action, while  $C = 5$  for each diving/surfacing action to penalize frequent surfacing/diving actions, since it hinders the vehicle’s horizontal target search progress. We set the terminal condition of the problem that the vehicle should get back to the position  $[-10, 10]^\top$  with high probability. For safety guarantee we impose a constraint on the uncertainty of the vehicle position: there cannot be less than 6 partitioned regions with its associated belief being too low,  $\sum_{i=1}^K \text{BeL}_i > 6$ . If such constraint is violated, the active sensing action is triggered, and the uncertainty on the vehicle position can be reduced by the surfacing event for inquisition of the GPS localization signal.

Since the zero dynamics of the vehicle and the hotspot are decoupled, we find the optimal partition of the vehicle state and the hotspot state separately. Fig. 6.7 shows partition of the belief space. The choice of the number of clusters is a trade-off between numerical accuracy and computational cost. We choose to partition the vehicle state into 20 cells, and the hotspot state into 30 cells, based on the “Elbow criterion” [90]. Boundaries of cells are smoothed into straight lines using the Least mean square method. As shown by Fig. 6.7, more partitions are assigned to positions where more simulated trajectory visit.

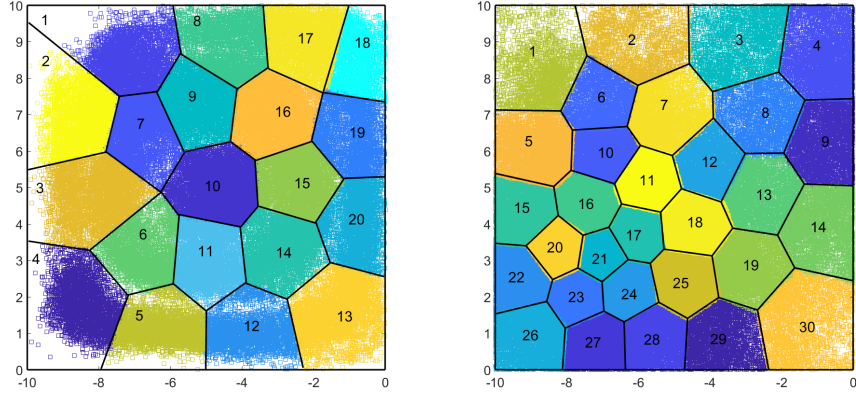


Figure 6.7: Partitioned state space. The colored squares represent datapoint position in Monte Carlo simulation. Squares with same color belong to the same partition. **(Left):** Partition for vehicle state. **(Right):** Partition for hotspot state.

### 6.5.3 Identification of the reduced dynamics

To train the LSTM model, we use the high-fidelity simulation data for  $N_{\text{train}} = 1000$  timesteps. The simulation data in the next 500 timesteps is taken as the test set.

**Justification of Assumption 6.4.2** To validate Assumption 6.4.2, that the LSTM can approximate the memory terms in the M-Z equation with a constant error bound, we present Fig. 6.8 to show how the modeling error of the proposed algorithm varies with time. Further, we compare the modeling error of the proposed algorithm with the GCM, to illustrate the effect of incorporating the memory effects. Fig. 6.8 shows comparison of the one-step error arising from the partitioned belief dynamics. The one-step model reduction error is defined as  $\|\Theta_{k+1}^p - T_\theta \Theta_k^p\|$  for GCM, and is defined as  $\|\Theta_{k+1}^p - T_\theta \Theta_k^p - g_{\text{LSTM}}(\bar{\Theta}_{k-1,N}^P)\|$  for the proposed algorithm. For both the training and testing dataset, the proposed algorithm achieves significantly lower model reduction error, compared with the GCM. Further, notice that the one-step model reduction error of the proposed algorithm is very small ( $< 0.01$ ), and is almost constant for both the training and validation set, which validates Assumption 6.4.2, that the LSTM can approximate the memory terms in the M-Z equation with a constant error bound.

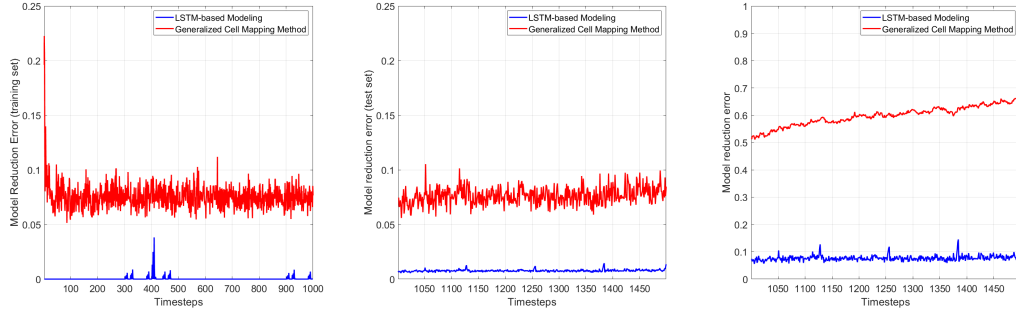


Figure 6.8: Model reduction error comparison between the proposed algorithm and the GCM. **(Left)**: One step model reduction error (training set) **(Middle)**: One step model reduction error (validation set). **(Right)**: Multi-step model reduction error comparison between the GCM and the LSTM-based modeling. The model reduction error of GCM shows significant growth over time, while the model reduction error of LSTM-based modeling stays relatively constant, which is consistent as the theoretical analysis.

**Validation of error bound analysis** Further, we examine the multi-step model reduction error growth. As shown in right figure of Fig. 6.8, the model reduction error of GCM shows significant growth over time, while it remains relatively constant for the proposed algorithm. This result coincides with the theoretical analysis, that by modeling the memory effects, the model reduction error retains a time-uniform upper bound.

#### 6.5.4 Symbolic planning

Given the identified belief dynamics model, we use HTN to perform planning in symbolic space. To evaluate performance of the proposed algorithm, in addition to comparing the proposed M-Z based approximation method with the GCM method, we also consider an omniscient solver, which has access to the perfect knowledge of the hotspot position. This omniscient solver gives a lower bound on the cost function value.

Given the identified belief dynamics from both GCM and the proposed method, we use the HTN planning to solve for the optimal sequence of actions. The simulation comparison on belief dynamics identification, and symbolic planning is shown in Fig. 6.9. As shown in the table, comparing with the GCM, the method proposed here significantly reduces the computational cost of symbolic planning. The main reason is that when grid size increases, the

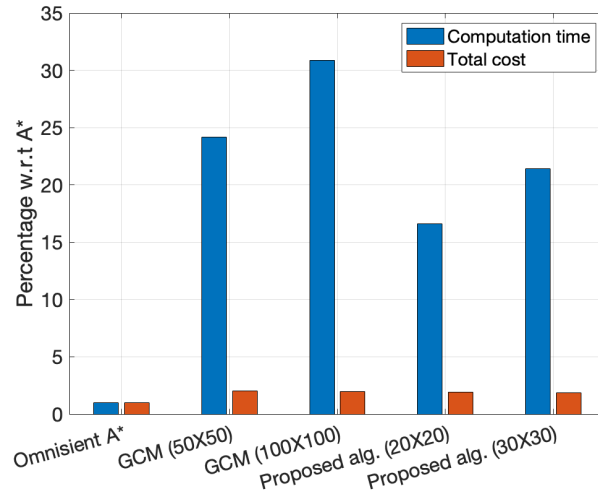


Figure 6.9: Simulation comparison between proposed algorithm and GCM, with omniscient A\* as baseline.

number of predicates will grow correspondingly. The large number of predicates introduces significant computation cost in evaluating the constraints in the symbolic planning problem. Therefore, the simulation comparison shows that by introducing the belief space partition, we significantly reduce the number of predicates, which reduces the total computation time of solving a POMDP problem.

## CHAPTER 7

### CONCLUSION AND FUTURE WORK

#### 7.1 Conclusion

This thesis develops reduced-order modeling and symbolic planning methods for both deterministic systems and stochastic systems in marine autonomy applications. First we present a reduced-order approximation scheme to the ocean flow field, and show that leveraging the abstraction reduces the computation cost of solving the deterministic planning problem. Further, we develop a belief abstraction algorithm based on the Mori-Zwanzig formalism. Both theoretical analysis and simulation result show that the reduced-order approximation model can accurately describe the full system dynamics. Further, we show that leveraging abstraction leads to lower computation cost in solving the POMDP problem.

#### 7.2 Future research directions

In this section, we discuss the future direction of the research presented in this thesis.

**Task and motion planning in unknown environments:** The MIP formulation presented in Chapter 4 and 5 can be easily applied to similar robotic planning problems, such as the Task and Motion Planning Problems (TAMP), which is a MIP that contains discrete task planning and continuous motion planning. However, the TAMP of a partially observable stochastic system is yet still an open problem in the robotics and control community. I intend to approach this problem by solving a MIP in belief space. This can be achieved by leveraging my previous contribution in belief abstraction, and the state space MIP solver.

**Distributed planing:** The path planning problem formulated in Chapter 4 and 5 can be extended to multi-agent scenarios, where multiple agents collaboratively explores an unknown environment under communication constraints. However, most existing work on

distributed planning cannot handle mixed-integer type decision-making. This may trigger a new research direction for distributed optimization and multi-agent reinforcement learning.

## REFERENCES

- [1] N. E. Leonard, D. A. Paley, R. E. Davis, D. M. Fratantoni, F. Lekien, and F. Zhang, “Coordinated control of an underwater glider fleet in an adaptive ocean sampling field experiment in Monterey Bay,” *Journal of Field Robotics*, vol. 27, no. 6, pp. 718–740, 2010.
- [2] R. N. Smith, Y. Chao, P. P. Li, D. A. Caron, B. H. Jones, and G. S. Sukhatme, “Planning and implementing trajectories for autonomous underwater vehicles to track evolving ocean processes based on predictions from a Regional Ocean Model,” *The International Journal of Robotics Research*, vol. 29, no. 12, pp. 1475–1497, 2010.
- [3] P. Ozog, N. Carlevaris-Bianco, A. Y. Kim, and R. M. Eustice, “Long-term mapping techniques for ship hull inspection and surveillance using an autonomous underwater vehicle,” *Journal of Field Robotics*, vol. 33, pp. 265–289, 2016.
- [4] X. Xiang, B. Jouvencel, and O. Parodi, “Coordinated formation control of multiple autonomous underwater vehicles for pipeline inspection,” *International Journal of Advanced Robotic Systems*, vol. 7, no. 1, p. 3, 2010.
- [5] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez, “Integrated task and motion planning,” *Annual review of control, robotics, and autonomous systems*, vol. 4, pp. 265–293, 2021.
- [6] R. Stern, A. Felner, J. van den Berg, R. Puzis, R. Shah, and K. Goldberg, “Potential-based bounded-cost search and anytime non-parametric A\*,” *Artificial Intelligence*, vol. 214, pp. 1–25, 2014.
- [7] B. Rhoads, I. Mezic, and A. C. Poje, “Minimum time heading control of underpowered vehicles in time-varying ocean currents,” *Ocean Engineering*, vol. 66, no. 1, pp. 12–31, 2012.
- [8] A. A. Pereira, J. Binney, G. A. Hollinger, and G. S. Sukhatme, “Risk-aware path planning for autonomous underwater vehicles using predictive ocean models,” *Journal of Field Robotics*, vol. 30, no. 5, pp. 741–762, 2013.
- [9] D. Kularatne, S. Bhattacharya, and M. A. Hsieh, “Going with the flow: A graph based approach to optimal path planning in general flows,” *Autonomous Robots*, vol. 42, no. 7, pp. 1369–1387, 2018.
- [10] ———, “Optimal path planning in time-varying flows using adaptive discretization,” *IEEE Robotics and Automation Letters*, vol. 3, no. 1, pp. 458–465, 2017.

- [11] M. Souhignac, “Feasible and optimal path planning in strong current fields,” *IEEE Transactions on Robotics*, vol. 27, no. 1, pp. 89–98, 2011.
- [12] S. M. LaValle, “Rapidly-Exploring Random Trees: A new tool for path planning,” Department of Computer Science, Iowa State University, Tech. Rep., 1998.
- [13] J. J. Kuffner and S. M. LaValle, “Rrt-connect: An efficient approach to single-query path planning,” in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, IEEE, vol. 2, 2000, pp. 995–1001.
- [14] J. D. Gammell, T. D. Barfoot, and S. S. Srinivasa, “Informed sampling for asymptotically optimal path planning,” *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 966–984, 2018.
- [15] Y. Chen, Z. He, and S. Li, “Horizon-based lazy optimal RRT for fast, efficient replanning in dynamic environment,” *Autonomous Robots*, vol. 43, no. 8, pp. 2271–2292, 2019.
- [16] R. Shome, K. Solovey, A. Dobson, D. Halperin, and K. E. Bekris, “dRRT\*: Scalable and informed asymptotically-optimal multi-robot motion planning,” *Autonomous Robots*, vol. 44, no. 3, pp. 443–467, 2020.
- [17] J. A. Sethian, *Level Set Methods and Fast Marching Methods: Evolving Interfaces in Geometry, Fluid Mechanics, Computer Vision and Material Science*. Cambridge University Press, 1999.
- [18] S. V. T. Lolla, “Path planning and adaptive sampling in the coastal ocean,” Ph.D. dissertation, Massachusetts Institute of Technology, 2016.
- [19] S. Karaman and E. Frazzoli, “Sampling-based algorithms for optimal motion planning,” *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.
- [20] I. Noreen, A. Khan, and Z. Habib, “Optimal path planning using RRT\* based approaches: A survey and future directions,” *International Journal of Advanced Computer Science and Applications*, vol. 7, no. 11, pp. 97–107, 2016.
- [21] L. Janson, E. Schmerling, A. Clark, and M. Pavone, “Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions,” *The International journal of robotics research*, vol. 34, no. 7, pp. 883–921, 2015.
- [22] M. Hou, H. Zhai, H. Zhou, and F. Zhang, “Partitioning ocean flow field for underwater vehicle path planning,” in *OCEANS 2019-Marseille*, IEEE, 2019, pp. 1–8.

- [23] E. Kaiser, B. R. Noack, L. Cordier, A. Spohn, M. Segond, M. Abel, G. Daviller, J. Östh, S. Krajnović, and R. K. Niven, “Cluster-based reduced-order modelling of a mixing layer,” *Journal of Fluid Mechanics*, vol. 754, pp. 365–414, 2014.
- [24] E. Ser-Giacomi, V. Rossi, C. López, and E. Hernández-García, “Flow networks: A characterization of geophysical fluid transport,” *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 25, no. 3, p. 036404, 2015.
- [25] A. I. Center, “Shakey the robot,” 1984.
- [26] A. Shamsah, J. Warnke, Z. Gu, and Y. Zhao, “Integrated task and motion planning for safe legged navigation in partially observable environments,” *arXiv preprint arXiv:2110.12097*, 2021.
- [27] Y. Zhao, Y. Li, L. Sentis, U. Topcu, and J. Liu, “Reactive task and motion planning for robust whole-body dynamic locomotion in constrained environments,” *arXiv preprint arXiv:1811.04333*, 2018.
- [28] A. Sinha, P. Malo, and K. Deb, “A review on bilevel optimization: From classical to evolutionary approaches and applications,” *IEEE Transactions on Evolutionary Computation*, vol. 22, no. 2, pp. 276–295, 2017.
- [29] C. R. Garrett, T. Lozano-Perez, and L. P. Kaelbling, “Ffrob: Leveraging symbolic planning for efficient task and motion planning,” *The International Journal of Robotics Research*, vol. 37, no. 1, pp. 104–136, 2018.
- [30] K. Hauser, V. Ng-Thow-Hing, and H. Gonzalez-Baños, “Multi-modal motion planning for a humanoid robot manipulation task,” in *Robotics Research*, Springer, 2010, pp. 307–317.
- [31] K. Hauser and J.-C. Latombe, “Multi-modal motion planning in non-expansive spaces,” *The International Journal of Robotics Research*, vol. 29, no. 7, pp. 897–915, 2010.
- [32] A. Bemporad and M. Morari, “Control of systems integrating logic, dynamics, and constraints,” *Automatica*, vol. 35, no. 3, pp. 407–427, 1999.
- [33] T. Marcucci, J. Umenberger, P. A. Parrilo, and R. Tedrake, “Shortest paths in graphs of convex sets,” *arXiv preprint arXiv:2101.11565*, 2021.
- [34] C. Sun, N. Kingry, and R. Dai, “A unified formulation and nonconvex optimization method for mixed-type decision-making of robotic systems,” *IEEE Transactions on Robotics*, vol. 37, no. 3, pp. 831–846, 2020.

- [35] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, “Backward-forward search for manipulation planning,” in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2015, pp. 6366–6373.
- [36] J. T. Thayer, R. Stern, A. Felner, and W. Ruml, “Faster bounded-cost search using inadmissible estimates,” in *Twenty-Second International Conference on Automated Planning and Scheduling*, 2012.
- [37] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, “Planning and acting in partially observable stochastic domains,” *Artificial intelligence*, vol. 101, no. 1-2, pp. 99–134, 1998.
- [38] L. P. Kaelbling and T. Lozano-Pérez, “Integrated task and motion planning in belief space,” *The International Journal of Robotics Research*, vol. 32, no. 9-10, pp. 1194–1227, 2013.
- [39] L. Burks, I. Loefgren, and N. R. Ahmed, “Optimal continuous state pomdp planning with semantic observations: A variational approach,” *IEEE Transactions on Robotics*, vol. 35, no. 6, pp. 1488–1507, 2019.
- [40] E. Zhou, M. C. Fu, and S. I. Marcus, “Solving continuous-state POMDPs via density projection,” *IEEE Transactions on Automatic Control*, vol. 55, no. 5, pp. 1101–1116, 2010.
- [41] J.-Q. Sun, F.-R. Xiong, O. Schütze, and C. Hernández, *Cell mapping methods*. Springer, 2018.
- [42] M. Grigoriu, “A method for solving stochastic equations by reduced order models and local approximations,” *Journal of Computational Physics*, vol. 231, no. 19, pp. 6495–6513, 2012.
- [43] D. Silver and J. Veness, “Monte-carlo planning in large pomdps,” *Advances in neural information processing systems*, vol. 23, 2010.
- [44] S. Brechtel, T. Gindele, and R. Dillmann, “Solving continuous pomdps: Value iteration with incremental learning of an efficient space representation,” in *International Conference on Machine Learning*, PMLR, 2013, pp. 370–378.
- [45] H. Kurniawati, “Partially observable markov decision processes and robotics,” *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 5, 2022.
- [46] Á. Torralba, V. Alcázar, P. Kissmann, and S. Edelkamp, “Efficient symbolic search for cost-optimal planning,” *Artificial Intelligence*, vol. 242, pp. 52–79, 2017.

- [47] G. Norman, D. Parker, and X. Zou, “Verification and control of partially observable probabilistic systems,” *Real-Time Systems*, vol. 53, no. 3, pp. 354–402, 2017.
- [48] L. Winterer, S. Junges, R. Wimmer, N. Jansen, U. Topcu, J.-P. Katoen, and B. Becker, “Strategy synthesis for POMDPs in robot planning via game-based abstractions,” *IEEE Transactions on Automatic Control*, 2020.
- [49] A. Herzig, L. Perrussel, and Z. Xiao, “On hierarchical task networks,” in *European Conference on Logics in Artificial Intelligence*, Springer, 2016, pp. 551–557.
- [50] R. Zwanzig, “Ensemble method in the theory of irreversibility,” *The Journal of Chemical Physics*, vol. 33, no. 5, pp. 1338–1341, 1960.
- [51] A. J. Chorin, O. H. Hald, and R. Kupferman, “Optimal prediction and the mori–zwanzig representation of irreversible processes,” *Proceedings of the National Academy of Sciences*, vol. 97, no. 7, pp. 2968–2973, 2000.
- [52] A. Chorin and P. Stinis, “Problem reduction, renormalization, and memory,” *Communications in Applied Mathematics and Computational Science*, vol. 1, no. 1, pp. 1–27, 2007.
- [53] D. Venturi, T. Sapsis, H. Cho, and G. Karniadakis, “A computable evolution equation for the joint response-excitation probability density function of stochastic dynamical systems,” *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 468, no. 2139, pp. 759–783, 2012.
- [54] Y. Zhu, J. M. Dominy, and D. Venturi, “Rigorous error estimates for the memory integral in the mori-zwanzig formulation,” *arXiv preprint arXiv:1708.02235*, 2017.
- [55] A. Gupta and P. F. Lermusiaux, “Neural closure models for dynamical systems,” *Proceedings of the Royal Society A*, vol. 477, no. 2252, p. 20 201 004, 2021.
- [56] Q. Wang, N. Ripamonti, and J. S. Hesthaven, “Recurrent neural network closure of parametric pod-galerkin reduced-order models based on the mori-zwanzig formalism,” *Journal of Computational Physics*, vol. 410, p. 109 402, 2020.
- [57] M. Eichhorn, “Optimal routing strategies for autonomous underwater vehicles in time-varying environment,” *Robotics and Autonomous Systems*, 2013, In press.
- [58] T. Lolla, P. F.J.Lermusiaux, M. P.Ueckermann, and P. J.Haley, “Time-optimal path planning in dynamic flows using level set equations: Theory and schemes,” *Ocean Dynamics 2014*, vol. 64, pp. 1373–1397, 2014.

- [59] A. Zamuda and J. D. H. Sosa, “Differential evolution and underwater glider path planning applied to the short-term opportunistic sampling of dynamic mesoscale ocean structures,” *Applied Soft Computing*, vol. 24, pp. 95–108, 2014.
- [60] D. K. Savidge, J. A. Austin, and B. O. Blanton, “Variation in the Hatteras Front density and velocity structure part 1: High resolution transects from three seasons in 2004-2005,” *Continental Shelf Research*, vol. 54, 2013.
- [61] ———, “Variation in the Hatteras Front density and velocity structure part 2: Historical setting,” *Cont. Shelf Res.*, 2013.
- [62] S. Cho and F. Zhang, “An adaptive control law for controlled lagrangian particle tracking,” in *Proceedings of the 11th ACM International Conference on Underwater Networks & Systems*, 2016, pp. 1–5.
- [63] S. Cho, F. Zhang, and C. Edwards, “Learning and detecting abnormal speed of marine robots,” *International Journal of Advanced Robotic Systems*, 2020, under review.
- [64] H. K. Khalil, *Nonlinear Systems*, 2nd. Prentice Hall, 1996.
- [65] S. Sastry and M. Bodson, *Adaptive control: stability, convergence and robustness*. Prentice Hall, 1994.
- [66] K. S. Narendra and A. M. Annaswamy, *Stable adaptive systems*. Prentice Hall, 1989.
- [67] P. A. Ioannou and J. Sun, *Robust Adaptive Control*. Prentice Hall, Inc., 1995, ISBN: 0134391004.
- [68] P. J. Martin, “Description of the navy coastal ocean model version 1.0,” Naval Research Lab, Tech. Rep. NRL/FR/7322–00-9962, 2000.
- [69] J. R. A. Luetich, J. J. Westerink, and N. W. Scheffner, “ADCIRC: An advanced three-dimensional circulation model for shelves, coasts, and estuaries,” Naval Research Lab, Tech. Rep. NRL/FR/7322–00-9962, 1992.
- [70] W. Li, J. Lu, H. Zhou, and S.-N. Chow, “Method of evolving junctions: A new approach to optimal control with constraints,” *Automatica*, vol. 78, pp. 72–78, 2017.
- [71] J. Forrest, J. Hirst, and J. A. Tomlin, “Practical solution of large mixed integer programming problems with umpire,” *Management Science*, vol. 20, no. 5, pp. 736–773, 1974.
- [72] S.-N. Chow, T.-S. Yang, and H.-M. Zhou, “Global optimizations by intermittent diffusion,” in *Chaos, CNN, Memristors and Beyond: A Festschrift for Leon Chua With DVD-ROM, composed by Eleonora Bilotta*, World Scientific, 2013, pp. 466–479.

- [73] D. L. Poole and A. K. Mackworth, *Artificial Intelligence: foundations of computational agents*. Cambridge University Press, 2010.
- [74] H. Zhai, M. Hou, F. Zhang, and H. Zhou, “Method of evolving junction on optimal path planning in flow fields,” *IEEE Transactions on Robotics*, 2020, under review.
- [75] D.-H. Ji, H.-S. Choi, J.-I. Kang, H.-J. Cho, M.-G. Joo, and J.-H. Lee, “Design and control of hybrid underwater glider,” *Advances in Mechanical Engineering*, vol. 11, no. 5, p. 1 687 814 019 848 556, 2019.
- [76] C. Shi, B. Wei, S. Wei, W. Wang, H. Liu, and J. Liu, “A quantitative discriminant method of elbow point for the optimal number of clusters in clustering algorithm,” *EURASIP Journal on Wireless Communications and Networking*, vol. 2021, no. 1, pp. 1–16, 2021.
- [77] P. Hart, N. J. Nilsson, and B. Raphael, “A formal basis for the heuristic determination of minimum cost paths,” *IEEE Transaction of Systems Science and Cybernetics*, vol. SSC-4, 1968.
- [78] R. T. Stern, R. Puzis, and A. Felner, “Potential search: A bounded-cost search algorithm,” in *Twenty-First International Conference on Automated Planning and Scheduling*, 2011.
- [79] N. J. Nilsson, *Principles of artificial intelligence*. Morgan Kaufmann, 2014.
- [80] K. P. Carroll, S. R. McClaran, E. L. Nelson, D. M. Barnett, D. K. Friesen, and G. N. Williams, “AUV path planning: An A\* approach to path planning with consideration of variable vehicle speeds and multiple, overlapping, time-dependent exclusion zones,” in *Proceedings of the 1992 Symposium on Autonomous Underwater Vehicle Technology*, 1992.
- [81] L. Crespo and J. Sun, “Stochastic optimal control of nonlinear systems via short-time gaussian approximation and cell mapping,” *Nonlinear Dynamics*, vol. 28, no. 3-4, pp. 323–342, 2002.
- [82] E. M. Bollt and N. Santitissadeekorn, *Applied and computational measurable dynamics*. SIAM, 2013.
- [83] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [84] R. Pascanu, T. Mikolov, and Y. Bengio, “On the difficulty of training recurrent neural networks,” in *International conference on machine learning*, PMLR, 2013, pp. 1310–1318.

- [85] J. Ding and A. Zhou, “Constructive approximations of markov operators,” *Journal of Statistical Physics*, vol. 105, no. 5, pp. 863–878, 2001.
- [86] A. M. Schäfer and H. G. Zimmermann, “Recurrent neural networks are universal approximators,” in *International Conference on Artificial Neural Networks*, Springer, 2006, pp. 632–640.
- [87] A. Lasota and M. C. Mackey, *Chaos, fractals, and noise: stochastic aspects of dynamics*. Springer Science & Business Media, 2013, vol. 97.
- [88] T. Bylander, “The computational complexity of propositional strips planning,” *Artificial Intelligence*, vol. 69, no. 1-2, pp. 165–204, 1994.
- [89] F. Bartumeus, M. G. E. da Luz, G. M. Viswanathan, and J. Catalan, “Animal search strategies: A quantitative random-walk analysis,” *Ecology*, vol. 86, no. 11, pp. 3078–3087, 2005.
- [90] R. Tibshirani, G. Walther, and T. Hastie, “Estimating the number of clusters in a data set via the gap statistic,” *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 63, no. 2, pp. 411–423, 2001.