Speeding Up Trajectory Planning for Autonomous Robots Operating in Complex Environments

by

Pradeep Rajendran

A Dissertation Presented to the
FACULTY OF THE GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(Mechanical Engineering)

December 2019

Acknowledgements

First and foremost, I would like to thank Professor Satyandra K. Gupta for being my doctoral advisor. His guidance, patience and constructive critique has been very instrumental in shaping my research philosophy and my approach to solving new problems. I am very fortunate to have an understanding and caring advisor. There are numerous valuable lessons I learned from him. These lessons will continue to guide me throughout my career.

Also, I would like to thank professors on my thesis committee: Professor Mitul Luhar and Professor Sven Koenig. I appreciate their time and effort in going over all the work I did and giving me their input. I would also like to thank the National Science Foundation (NSF grants 1634433,1634431,1526016). Without their financial support, this work would not have been possible. Next, I would like to thank the professors who taught me at University of Maryland, University of Wisconsin and Hong Kong University of Science and Technology.

I would like to thank all of my labmates for their pleasant company and numerous stimulating discussions throughout the past few years. Specifically, I would like to thank Ariyan Kabir, Di Zeng, Brual Shah, Shaurya Shriyam, Sarah Al-Hussaini, Shantanu Thakar, Prahar Bhatt, Yeo Jung Yoon, Aniruddha Shembekar, Rishi Malhan and Jason Gregory.

Finally, I would like to thank my parents and my sister who were always available and gave words of encouragement and support when I needed it most. I want to thank my wife. She has always given words of encouragement, put up with my absent-mindedness and taken on more than fair share of life's many responsibilities.

Table of Contents

List Of Tables

List Of Figures

Abstract

Advances in sensing and computing hardware have physically equipped robots to operate in complex environments. In many real-world settings, we desire robots to operate at a high-level of autonomy to reduce operating costs and manpower requirements. A high-level of autonomy can be achieved only when robots are able to plan missions and tasks themselves. Trajectory planning is a fundamental building block required to support high-level decision making in robots.

Trajectory planning for autonomous robots operating in complex environments is a challenging problem. The complexity of trajectory planning problems stems from the dimensionality of robot's state space, the complexity of the robot kinematic and dynamic model, the nature of environmental constraints (e.g., obstacles), task constraints (e.g., rules), the optimization objective function, and the planning-time requirements needed for deployment in the real world. Depending on the complexity, these problems can be solved by existing methods to produce feasible trajectories. But, in many practical applications (e.g., automated package delivery), computing a feasible trajectory alone is not enough. The quality of the computed trajectory is also important. However, in many cases, computing truly optimal trajectories is computationally intensive and thus, very time-consuming. As a result, existing methods do not satisfy planning-time constraints required by the application while maintaining optimality. We need a method that produces high-quality trajectories and at the same time produce those trajectories quickly. Anytime methods handle exactly this problem. However, these methods produce high-quality trajectories quickly only when good heuristics are used. This thesis focuses on heuristic techniques for anytime algorithms that speed up trajectory planning for autonomous robots in complex environments.

Trajectory planning for Unmanned Surface Vehicles (USVs) is one representative example that includes many of the complicating factors such as uncertainty in motion, uncertainty in perception, spatio-temporally varying environment (e.g., waves and traffic vessels), and fast planning-time requirements. Accounting for all of these factors is crucial for building reliable autonomy solutions. A wave-aware deliberative planner that avoids collisions with traffic vessels is presented. It also opportunistically traverses wavefields generated by these vessels while minimizing the execution time and the risk of failure. The planner performs a search over a 4D pose-time lattice to generate a collision-free, minimum-risk sequence of motion goals. It addresses motion uncertainty, failure risk and perception uncertainty through a parametric model. Heuristics to speed-up computation are presented. Simulation and physical experimental results are presented. They show that our methodology quickly produces plans that execute faster and safer when compared to the typical reactive planning schemes.

A real-time dynamics-aware reactive trajectory generator is presented. It produces trajectories that avoid collisions with dynamic obstacles under varying dynamic constraints imposed on the robot as its operating environment changes. It is an obstacle avoidance approach that explicitly reasons about reference trajectory deviation, varying dynamics constraints while minimizing collision risk with uncertain dynamic obstacles. Simulation results are presented in the context of the Unmanned Ground Vehicle (UGV) domain, to show that the planner is able to effectively deal with the dynamic obstacles on terrains with varying slopes.

Trajectory planning for industrial manipulators operating in cluttered environments is a representative example that involves navigating a high-dimensional configuration space. In small-volume manufacturing applications, manipulators perform non-repetitive tasks that change frequently. Every time a task changes, generating manipulator trajectories by manual placement of waypoints is tedious and inefficient. Automated trajectory-planning methods can help improve manufacturing throughput and reduce operating costs. Planning-time and trajectory-quality are the key metrics that decide the utility of an automated planning method. A highly reliable

trajectory planning framework is presented. It exploits workspace hints and uses a parameter scheduling scheme to generate high-quality trajectories quickly. The framework is also able to seamlessly incorporate human assistance through an intuitive user-interface to cooperatively (i.e., human and computer) solve difficult planning problems.

Chapter 1

Introduction

1.1 Motivation

Diverse domains such as large-scale automobile manufacturing, large-scale electronics manufacturing, medical surgery [128], self-driving urban vehicles, package delivery, marine traffic control [6] and search/rescue operations can greatly benefit from a degree of automation. These automation systems range from simple hardware mechanisms to a complex heterogenous team of autonomous robots.

Let us take the example of the marine domain to see the need for automation and the benefits it offers. Industry interest in marine automation is booming with growing pressure from heavy competition, increasing oil prices, stricter environmental regulations, elaborate global trade and emerging shipping trends. The marine industry is moving towards the phased introduction of automation technologies ranging from assistive-behaviors to fully autonomous operations. With recent advances in robotics and key enabling technologies (i.e., faster compute, sensing), utilizing autonomous robots in sectors such as ocean mining, fish farming, arctic exploration is physically realizable. In general, annual cargo-shipping costs are dominated by fuel and man power expenditures. To save on costs, naval fleets in the shipping industry are expected to slowly transition into hybrid fleets containing autonomous, semi-autonomous (reduced crew) and manned ships by

2035[5, 1]. Beyond the shipping industry, even offshore wind farm operations report that manned vessels account for as much as 60% of operating costs [4]. It is clear that marine automation can reduce operating costs. Marine automation can not only reduce operating costs but also improve efficiency and safety. Between the years 2011-2016, loss of control and ship-to-ship collision were the major casualty events occurring in the sea [3]. As much as 42% of the casualties were reported to have taken place in port environments followed by 28% in coastal areas. Upto 60% of the marine accidents that took place between 2011-2016 were caused by human error. A disproportionately large number of these events were reported have happened to cargo-ships. From these statistics,

Figure 1.1: Prime applications of marine automation: Port assistance, Search and Rescue, Port Policing

it is clear that incorporating a degree of automation in prime application areas such as in Figure 1.1 can reduce costs and save lives.

Another domain that can benefit greatly through automation is manufacturing. In manufacturing settings, robotic manipulators are widely used in the factory floor for various tasks such as automotive assembly, quality control, packing and machining [2, 144, 67, 200]. A large fraction of these manipulators follow pre-programmed sequence of actions to complete tasks. Before any task, human operators program the manipulator by manually placing waypoints to accomplish a given task. This is tedious and inefficient use of human labor.

If the environment does not change, programming the robot once is enough. The robot is going to perform millions of repetitions of the same trajectory. So, the manual programming time is going to be amortized over millions of repetitions. In this case, it seems that time spent in manual programming is justified. But, due to the fact that humans have programmed the robot, we do not get any local optimality guarantees let alone global optimality guarantees. Considering millions of repetitions, even losing a second can quickly add up and decrease manufacturing throughput.

When the environment changes or the task description changes (e.g., different car model, different door assembly) (Figure 1.2), the manipulators have to be programmed again. When the environment changes quickly (e.g., dynamic environment such as product packing area, mobile manipulators in a warehouse setting) (Figure 1.2), it becomes impractical to have humans reprogram robots.

Thus, in such scenarios, using automated trajectory planning for manipulators can clearly lead to reduced operating costs and higher productivity.

1.2 Background

Autonomous robots is a general term for various types of robots not limited to mobile robots, robotic manipulators and mobile manipulators. In order for a robot to operate autonomously, it should be able to perceive the environment it operates in and perform actions based on the

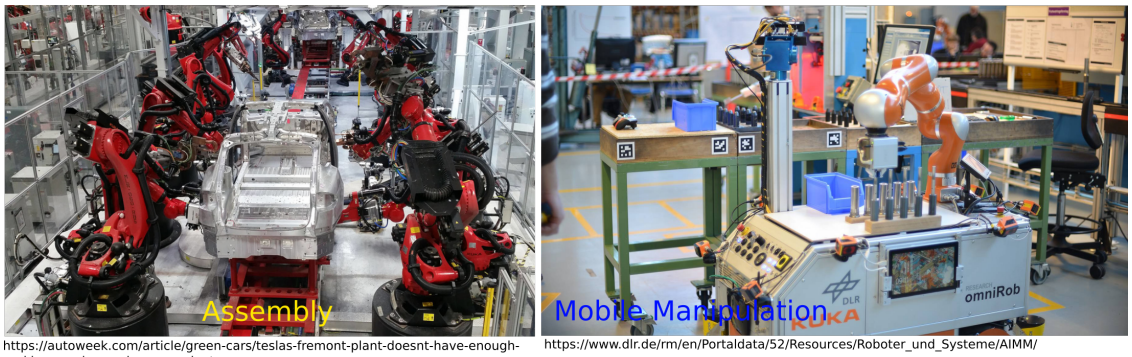

Figure 1.2: Left: An environment where tasks may change slowly (e.g., car model changes, different seat assembly), Right: An environment where tasks may change quickly (e.g., mobile robots interacting with human traffic in a warehouse setting)

perceived information to achieve a certain goal. Thus, robotic systems can be decomposed as shown in Figure 1.3.

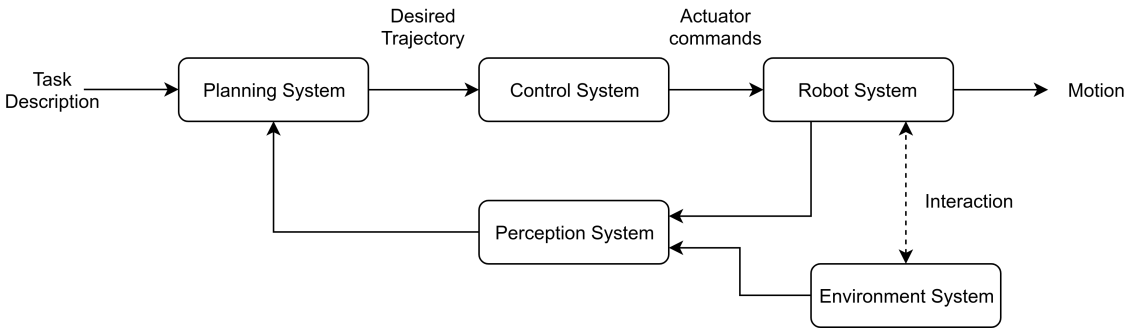

Figure 1.3: Architecture of a robotic system

Commanding and controlling autonomous robots requires the confluence of many fields of engineering such as machine design, decision-making algorithms, Artificial Intelligence (AI) planning/scheduling, trajectory planning, control algorithms, computer vision, computer networks and signal processing.

In this list, trajectory planning refers to a field of robotics that studies decision-making algorithms that plan how a robot is going to navigate the world to achieve a set of goals. For instance, the goal could be to move from one location to another location in a safe manner and in the shortest amount of time.

Many higher-level autonomy functions such as mission/task planning rely heavily on trajectory planning. Consider a heterogenous team of 10 package-delivery robots consisting of land and air units attempting to deliver 100 packages within a deadline of a few hours. Each robot is to be allocated a task of delivering a set of packages to a certain area. And, the capability of each robot is different. Land units can haul heavy payloads and stay active longer. Air units can fly fast but only take on light payloads and stay active for a shorter duration. Before all robots can be tasked, we require an understanding of whether a certain allocated task can be realistically completed by a robot (i.e., does there exist a trajectory plan for which delivery can be completed?). Thus, even at the task planning level, trajectory planning serves as a crucial tool for evaluating the feasibility of a task. Some of the relevant application areas are [192, 125, 175, 81].

Trajectory planning is a fundamental capability required to realize autonomous robot. Trajectory planning for autonomous robots is a challenging research area with immediate practical utility across many domains.

Given an representation of the environment, trajectory planning determines how a robot should operate in the environment over time in order to move from an initial state to a final state while satisfying a set of constraints.

Typically, a trajectory planning problem is defined by a robot model, a search-space, a set of environment and task constraints, an objective function, an initial robot state and goal conditions.

Depending on the nature of the planning problem, a robot model could include a robot kinematic model, a robot dynamic model or both. A kinematic model of a robot describes the relationship between the input command velocities (e.g., wheel velocity, steering velocity) and how the robot is going move (e.g., robot body velocity). A dynamic model of a robot describes the relationship between the control forces/torques (e.g., motor torque, thruster force) and how it changes velocities of the components of the robot (e.g., wheel velocity). In modern control theory and planning literature, it is typical to describe the robot model using a state-space representation. The state space of a robot refers to the collection of states a robot might take. For example,

the state of an Unmanned Ground Vehicle (UGV) can be described by the vector $(x, y, \theta, \dot{x}, \dot{y}, \dot{\theta})$. Given a state-space representation, the search-space can be defined in terms of the control signals that move the robot. For example, a UGV is moved by the throttle and steering control signals. The complexity of a trajectory planning problem is determined by the dimensionality of the state space, the complexity of the robot model, the nature of environmental constraints, task constraints and the objective function.

1.2.1 Trajectory Planning Paradigms

The goal of trajectory planning is to compute reference control signals (or reference trajectories) that move the robot from an initial state to a final state while satisfying environment and task constraints. One of the basic constraints in trajectory planning is collision avoidance with static and dynamic obstacles.

There are three general paradigms to solve the trajectory planning problem: search-based, sampling-based and optimization-based methods. Hybrid approaches combining the general methods is also not uncommon.

1.2.1.1 Search-Based Methods

Search-based methods perform a combinatorial optimization over a sequence of actions that can be performed in the state-space. Combinatorial optimization entails the systematic examination of the state-space to produce an optimal sequence of actions. Classical forms of dynamic programming fall in this category. Two popular algorithms are Dijkstra's algorithm [46] and A* algorithm [69]. A* algorithm utilizes a heuristic that guides the search towards promising actions that finally lead to the goal configuration. These algorithms work well when the environment information is crisp and known apriori. Moreover, due to the combinatorial nature of the search, high-dimensional state-space pose a significant challenge to these methods. For instance, the worst-case time complexity of a combinatorial search grows as $O(b^d)$ where b is the branching

factor and d is the depth of the search-tree. As the entire state-space is examined eventually, search-based methods are said to be *complete*. In complex environments and high-dimensional state-spaces, complete planners may not be able to produce solutions within reasonable time limits.

1.2.1.2 Sampling-Based Methods

Sampling-based approaches such as RRT (Rapidly-exploring Random Trees) [110] and PRM (Probabilistic Road Maps) [87] are very similar to search-based methods except that they do not systematically examine every possible state-space before producing a solution. Instead, these methods take a different approach. Random samples of the state-space are used to explore and build a representation of the connectivity in the state-space. For this reason, many sampling-based approaches are only *probabilistically complete*; given enough computation time, they will eventually produce a solution. One of the major challenges in these methods is generating useful state-space samples (i.e., sampling), evaluating distances/costs between two points in the state-space and finding connections between these samples.

1.2.1.3 Optimization-Based Methods

Optimization-based methods typically formulate the trajectory planning problem as an optimal control problem [73]. Environmental and task constraints are decomposed into a collection of initial, final, path, differential and algebraic constraints. Then, the optimal control problem is solved using approaches outlined in [22]. Ultimately, the problem is either solved analytically or numerically. The challenge in either case is the ability to incorporate a wide variety of constraints. For example, disjunctive constraints are hard to incorporate into the formulation without resorting to a type of Mixed-Integer Non-Linear Programming (MINLP).

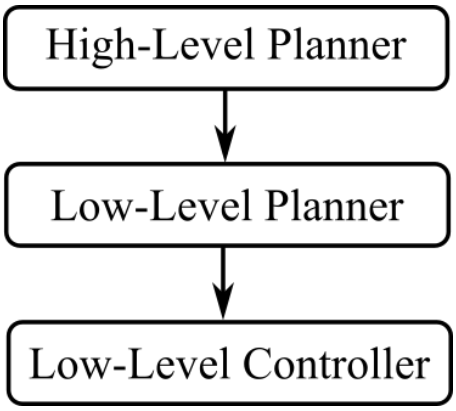

Figure 1.4: Planning hierarchy

1.2.2 Planning Hierarchy

Owing to the complexity, trajectory planning is often performed hierarchically. Typically, search-based methods are used for global planning over large areas and sampling-based methods are used for global planning in high-dimensional state-spaces. These methods are able to reason over long planning horizons produce reference trajectories that are able to easily capture a wide variety of constraints. Classically, optimization-based methods are used for low-level reactive trajectory planning where the constraints are simpler and typically have analytical representations. Optimization methods are used to produce minor tweaks to the reference trajectory produced by search-based or sampling-based method. In all of these approaches, the duration of planning horizon is a design variable. Planning horizon refers to the length of time for which decisions are evaluated. Shorter planning horizons apply local reasoning and therefore focuses predominantly on shortsighted tasks such as avoiding obstacles only in the local vicinity. This forgoes looking at the larger picture and may get the robot stuck in a local minimum. However, shorter planning horizons generally require less computation time. Longer planning horizons on the other hand allow for global reasoning and hence, result in optimal behavior. However, longer planning horizons generally require much more computation time.

1.2.3 Motion and Perception Uncertainty

Motion uncertainty refers to uncertainty in the motion of a robot as it is commanded to perform a maneuver. This typically arises due to unmodeled dynamics and disturbances a robot is exposed to as it operates in the environment. For example, consider an Unmanned Surface Vessel (USV) moving over the sea. Even though the USV may be commanded to move due North, it may deviate off course due to sea disturbances. Perception uncertainty refers to the uncertainty in the measurements recorded by the perception system. This typically arises due to occlusions, sensing range limitations and sensor update rates.

When the environment is controlled and predictable to a high-degree, the trajectory planning problem is often simple especially if the state space is low-dimensional (less than 4D). Automated warehouse management [39, 13] using mobile robots and robotic manipulators operating in an automobile assembly unit are examples where the environment is carefully controlled, uncertainty in state estimates are very low and often many parts of the trajectory planning problem can be pre-computed. Thus, in these applications the trajectory planning problem is relatively simple.

However, when the environments change over time and when the robot is able to perceive the environment only with a degree of certainty, trajectory planning becomes challenging. It becomes more challenging when the spatio-temporally changing environments also influence robot dynamics. There are many domains such as aerial robotics, ground-based robotics and marine robotics that present complex environments to an autonomous robot.

In the marine robotics domain, automated trajectory planning for Unmanned Surface Vehicles (USVs) is a representative example of the challenges of automated trajectory planning in a complex environment. Significant environmental spatio-temporally varying disturbances like waves, wakes and winds [184] impart strong motion uncertainty [26] to vessels operating over water bodies. The effects of motion uncertainty are markedly more pronounced in smaller USVs as they are particularly more susceptible to wave disturbances. Furthermore, the intentions of other traffic

vessels in the vicinity of the USV is not known apriori and only uncertain predictions of the future trajectories of these traffic vessels can be made. Collision avoidance is more challenging under these circumstances and it involves not only staying a safe distance from static obstacles and uncertain dynamic obstacles but also following COLREGs [186]. COLREGs is a set of maritime navigation rules set by the International Maritime Organization (IMO) to minimize collisions and confusion among marine vessels. Another aspect that makes trajectory planning challenging is the real-time requirements imposed by physical systems. Autonomous robots operating in real environments need to sense, plan and act as fast as possible to avoid collisions and adapt to changing environmental circumstances.

1.2.4 High-Dimensional State-Space

Another aspect that makes trajectory planning challenging is the high-dimensionality of the search space. A representative example of this is the search space associated with a high Degree-of-Freedom (DoF) robotic manipulator arm. Typically, the dimensionality of the configuration space corresponding to a manipulator is equal to the number of joints the manipulator has. Determining collision-free trajectories for manipulators amounts to finding a continuous curve in the configuration space. Explicitly mapping the configuration space ahead of planning is intractable due to the dimensionality of the space. Therefore, trajectory planning for manipulators needs to be performed without this explicit mapping.

1.3 Goal and Scope

This dissertation explores computational foundations for trajectory planning for autonomous robots operating in complex environments. Depending on the complexity, these trajectory planning problems can be solved by existing methods to produce feasible trajectories. However, in many practical applications (e.g., automated package delivery, search and rescue), computing a

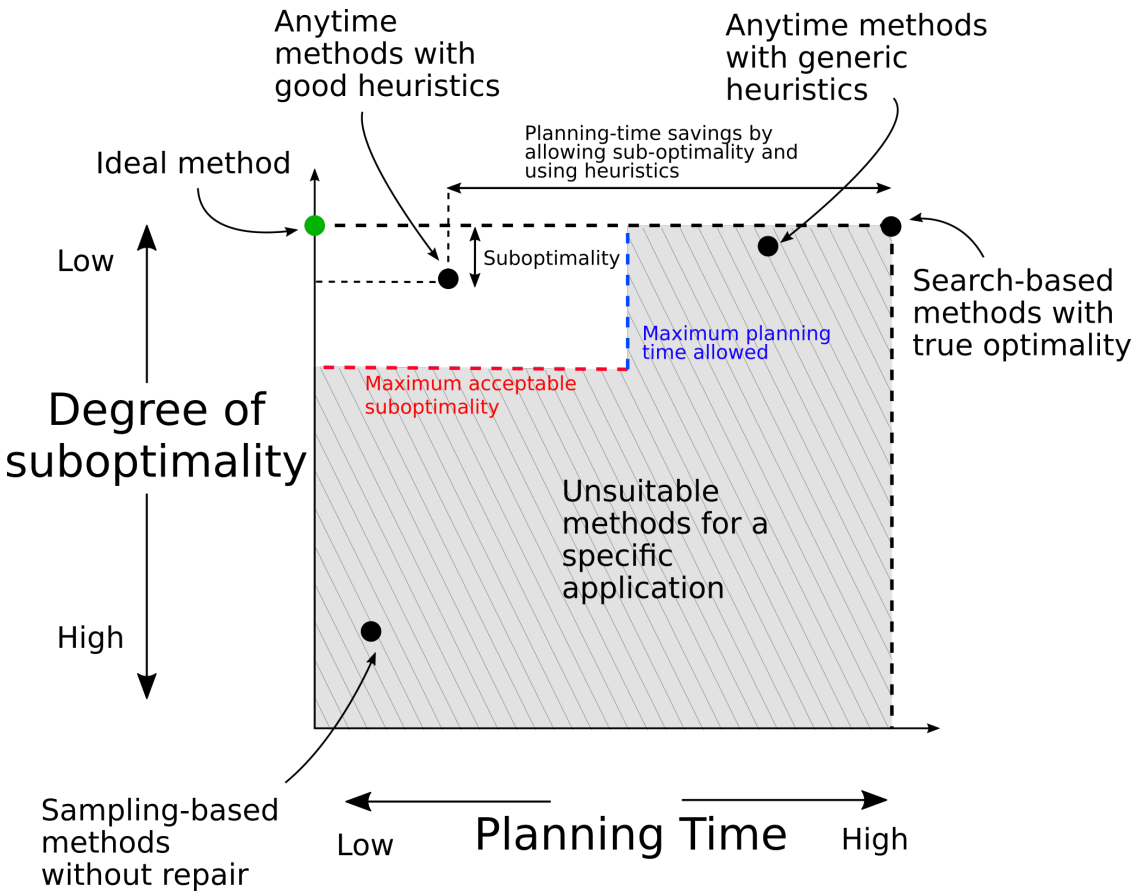

Figure 1.5: Dissertation theme. Specific applications impose constraints on minimum optimality and maximum planning-time. This rules out anytime methods with generic heuristics as candidates. Anytime methods with good heuristics are needed to produce high-quality solutions quickly.

feasible trajectory alone is not enough. The optimality of the computed trajectory is also important. Traditionally, search-based methods have been very popular as they provide strong trajectory optimality guarantees. However, seeking optimal solutions by utilizing search-based methods are only practical when low-dimensional search-spaces are considered. In practical settings, the trajectory planning problem often needs to include not only the state of the robot but also time-varying constraints such as dynamic obstacles and dynamics constraints. Thus, time-dimension needs to be included in the search-space. However, in many cases where the search-space is high-dimensional, computing truly optimal trajectories using search-based methods is computationally intensive and thus, very time-consuming. Operating in complex environments (e.g., uncertain and dynamic environments) impose strict constraints on planning-time. One the one hand, we seek trajectory optimality afforded by search-based methods. On the other hand, we also seek fast planning times that search-based methods are known to lack.

In order to make the search tractable, the field of anytime methods was introduced by researchers. Anytime methods utilize heuristics in search-based methods to accelerate and produce useful suboptimal solutions very quickly. There are two classes of heuristics: admissible heuristics and inadmissible heuristics. On the one hand, the use of admissible heuristics preserves the optimality guarantees of search-based methods. However, efficient admissible heuristics that work in time-extended search-spaces are difficult to generate. On the other hand, the use of potentially inadmissible heuristics provides solutions quickly but provides no guarantees regarding optimality. There are no general techniques that can be applied to generate these heuristics.

In any case, the optimality guarantees given by search-based methods are valid only when information about the environment and constraints are free of uncertainty. And in a lot of applications, it is acceptable to produce and use high-quality trajectories that are near-optimal. Despite the shortcomings of inadmissible heuristics, these heuristics are still valuable if they provide high-quality trajectories that are near-optimal.

For problems that are intrinsically high-dimensional (e.g., trajectory planning for manipulators), even search-based anytime methods do not scale well. Thus, sampling-based methods are the only viable alternative. However, it is well known that sampling-based methods produce suboptimal solutions unless specific effort is made to take suboptimal solutions and repair them into optimal solutions. In these circumstances, sampling-based methods can be fitted with machinery that utilizes potentially inadmissible heuristics to rapidly converge to near-optimal solutions. Figure 1.5 outlines the premise of the dissertation.

This dissertation presents computational techniques for speeding up trajectory planning for autonomous robots operating in complex environments exhibiting one or more of the following qualities:

1. Requirement of real-time operation
2. Requirement of collision avoidance
3. Existence of perception uncertainty
4. Existence of robot motion uncertainty
5. High-dimensionality of search space

The methods developed in this dissertation are domain independent. These approaches are however motivated and demonstrated in the context of trajectory planning for Unmanned Ground Vehicles (UGVs), Unmanned Surface Vehicles (USVs) and robotic manipulators.

1.4 Overview

This section provides a brief summary for each chapter in this dissertation.

Chapter 2 presents a literature review on trajectory planning methods used in applications involving unmanned ground vehicles, unmanned surface vehicles and manipulators. The focus is

on search-based, sampling-based and optimization-based methods that address dynamic obstacles, motion/perception uncertainty and high-dimensionality of search space.

Chapter 3 presents a reactive planning methodology that explicitly reasons about reference trajectory deviation and handles varying dynamics constraints while minimizing collision risk. Simulations are performed on an unmanned ground vehicle that operates over uneven terrain. It presents simulation results to show that the planner is able to effectively deal with the dynamic obstacles on terrains with varying slopes.

Chapter 4 presents a deliberative planning methodology that avoids collision with dynamic obstacles, satisfies trajectory constraints, opportunistically traverses large permeable dynamic obstacles while minimizing the risk of failure. It also describes speed-up techniques to improve search performance. Physical and simulation experiments are described for an unmanned surface vehicle operating in congested environments. It also presents results from these simulation and physical experiments.

Chapter 5 presents a sampling-based framework for trajectory planning in high-dimensional configuration space. It describes a novel bi-directional tree-search framework and its building blocks. Simulation experiments are conducted on high degree-of-freedom manipulators. It also presents results from these simulation experiments.

Chapter 6 provides a summary of the contributions of this dissertation and its potential impact. It also identifies avenues of future research.

Chapter 2

Literature Review

2.1 Overview

Trajectory planning for autonomous robots is an extensively studied problem in robotics. This chapter reviews the literature related to trajectory planning in unmanned ground vehicles, unmanned surface vehicles specifically focusing on dynamic obstacle avoidance, motion/perception uncertainty and high-dimensional state-spaces. Additionally, this chapter also reviews literature related to trajectory planning for high degree-of-freedom robots such as robotic manipulators. It also outlines some of the shortcomings of existing methods.

2.2 Handling Dynamic Obstacles

2.2.1 Reactive Trajectory Planning Methods

A variety of methods has been proposed to address the issue of fast, reactive, dynamics-aware planning that translate waypoints into control sequences that avoid collision with static obstacles and dynamic obstacles.

2.2.1.1 Zero-Order Methods

Some methods like potential field method, Vector Field Histogram [25] and Nearness Diagram Method [134] do not explicitly encode the dynamics of the robot and obstacle into the procedure. They are considered zero-order methods as they only consider the noisy instantaneous position as measured by the sensors.

As a result, a higher-level of deliberative planning is not possible. Furthermore, though most of these methods solve the given problem to a reasonable extent, they fail to provide a more general framework that can incorporate human-imposed rules (like obeying a traffic light). Furthermore, they may result in undesirable oscillatory behavior.

2.2.1.2 First-Order Methods

Methods such as Dynamic Window Approach (DWA) [53] and Velocity Obstacle (VO) Method [51] work directly in the velocity space (VS) and hence, are able to incorporate first-order behavior of obstacles and also respect acceleration limits on actuators in the planning process. In particular, VO and its variants such as Non-Linear Velocity Obstacle (NLVO) [171], Generalized Velocity Obstacles (GVO) [208] has been successful in handling the dynamic obstacle and actuator limit problem. A major reason for its success is due to planning in VS. This provides a framework to capture many constraints in a unified approach. For example, static or dynamic obstacles can be readily encoded in VS. Other constraints like traffic rules and other rules of engagement can be transformed into appropriately shaped velocity obstacles.

The original VO formulation assumes the obstacles are moving in piece-wise constant velocities between processing windows. While this assumption is relaxed in NLVO, prior information regarding the obstacle trajectory is required. Both in VO and NLVO, the robot is assumed to be capable of moving in any direction from a given location. But, car-like robots can only move in the direction they are facing. The GVO formulation incorporates such constraints through the kinematic model of the robot. However, it still retains the assumption that obstacle velocities are

piece-wise constant between processing windows. In high-speed situations, the feasible action set may become empty and this has to be detected early enough for contingency measures.

Local obstacle avoidance paradigms such as potential field method, vector field histogram [25] and nearness diagram method [134] primarily consider the instantaneous position of the obstacles while computing collision avoidance strategies. Using just instantaneous position and purely reactive strategies result in undesirable oscillatory behavior. These methods are termed as zero-order methods as they only consider position and not the velocity of obstacles.

Inevitable collision states (ICS) [55] is an ideal method that provides provable zero collision guarantees. In this method, states of the robot in which no feasible action may be performed to avoid collision are identified and deliberately avoided. Variants of ICS such as [127] and [24] differ in how much the three criteria for motion safety [54] is relaxed (i.e. knowledge of vehicle dynamics, future environment/obstacles and ability to reason over infinite time horizon). The precise characterization of ICS is computationally expensive even for low-dimensional dynamics and hence, prohibitive for real-time applications.

Other methods such as Dynamic Window Approach (DWA) [53] and Velocity Obstacle (VO) [51] directly work in the velocity space (VS). Hence, these methods are able to incorporate first-order behavior of obstacles and vehicle dynamics to yield viable velocity vectors that avoid obstacles. The original VO formulation models obstacles moving in piece-wise constant velocities between planning windows. Assumptions are relaxed in recent developments of VO such as [171, 78, 56, 17]. They differ in the representation of obstacles, assumptions related to obstacle trajectories and how they react to impending collision.

While dynamics constraints are respected in GVO, ICS and some of the recent approaches [172, 178, 72, 75, 206], they do not explicitly consider (1) reference trajectory deviation, (2) time-varying dynamics constraints due to uneven terrain. Collision avoidance alone without regard to reference path deviation may compromise mission performance.

2.2.2 Deliberative Trajectory Planning Methods

Deliberative planning involves planning for a larger time horizon and reasoning over the evolution of environmental constraints and other constraints over the entire planning horizon. Lattice-based methods used for dynamic obstacle avoidance compute trajectories using a lattice of viable alternatives. This lattice structure is computed using the kinodynamic model of the vehicle. The approach presented in [27] plans the trajectory in a 4D lattice structure. Computational efficiency is retained by adaptive variation of lattice resolution and by using the Anytime Dynamic A* (AD*) [119] algorithm for graph search and by using environment constrained heuristics to guide the search. Similar methods have been developed for unmanned surface vehicles in [167], where the motion primitives used during the graph search are scaled proportionate to the congestion in the environment.

Chen et al. utilize workspace heuristics [32] to accelerate the search. A sequence of circles is generated to connect the start and goal configurations. It is then used as a heuristic to guide the search. Dynamic obstacles are also considered in [33] where a cylinder path in time-extended workspace is first generated and later utilized as a heuristic.

Using lattice spaces with only position and orientation of the UGV results in inability to reason about dynamics. In [27], a 4D (x,y,θ,v) multi-resolution lattice based global planner is used to generate dynamically feasible trajectories for an autonomous car over 200 $\times$ 200 meter grids. Motion primitives are specifically tailored for traversing the lattice space. This ensures seamless transfer between high-resolution and low-resolution lattices. To deal with limited deliberation time, the Anytime Dynamic A* (AD*) algorithm is used to solve the graph problem. Mechanism and environment constrained heuristics are computed to more effectively guide AD* to the goal. All of these features combined enables real-time planning capability.

Time-bounded lattice [99] is an extension of [27] to account for uncertain and dynamic obstacles. It uses a hybrid state lattice that is 6D near the current UGV position and 2D when

sufficiently far away. The threshold for the transition to lower dimensional lattice is determined by the uncertainty propagation of dynamic obstacle. The transition to lower dimensional lattice is initiated when collision probability with dynamic obstacles decrease below a threshold. By doing this, real-time guarantees on the performance is given.

Optimization-based methods such as [199], [37], [151], [207], [117] have been used to compute optimal trajectories. These trajectories are generated for a relatively short planning horizon and thus, only optimal considering the short horizon. Specifically, Model predictive control (MPC) based algorithms [7, 151, 37, 199] are also used to compute trajectories for robots with dynamics constraints by solving it as an optimization problem.

In the marine domain, Abdelaal and Hahn implement collision avoidance [7] as a time-varying constraint at the controller level. Their method is also compliant with International Regulations for Preventing Collisions at Sea (COLREGS) by incorporating them as constraints depending on the state of the Unmanned Surface Vessel (USV) and the traffic vessels. However, uncertainty introduced by the perception system is not included in the formulation.

Tam and Bucknall report a cooperative path planning algorithm [189] for USVs. Their main contribution is a deterministic path planner that incorporates the evasive action that other traffic vessels may take when reacting to the motion of the USV. The path planner produces single course changes in a manner that is consistent with COLREGs. This is different from other methods where multiple course changes are prescribed.

Naeem et al. utilize the A* algorithm [135] with specialized heuristics. To implement COLREGs constraints, virtual obstacles are placed at specific locations of the oncoming traffic. This naturally encodes the COLREGs rules without any special handling.

Works such as [215, 177] consider the ocean currents as time-varying fields over which path optimization is performed. Some of these techniques are also applicable in the context of dynamic obstacle induced waves.

In the context of aerial robotics, Liu et al. use a two-stage approach [121] where the workspace is first decomposed into regions known as Safe Flight Corridor (SFC). These regions are formed by a sequence of overlapping convex polyhedra connecting the start and goal points. Once these regions are computed, these regions are used in the ensuing trajectory optimization problem as constraints. The idea is to find optimal trajectories that completely lie within the sequence of overlapping polyhedra. The candidate trajectories are constructed in a piecewise manner such that each piece of trajectory lies within a single convex polyhedron and each piece connects to the next piece (including higher-order continuity). This way a graph-search (global) method is fused with an optimization-based (local) method to generate trajectories that are not fundamentally limited to one homotopy class.

For trajectory planning on an aerial platform, Otte et al. [140] form a discrete graph over state-space where the state consists of time, pose and velocity. It handles a time-varying wind field and produces solutions in an anytime fashion by allowing a specified level of suboptimality.

Werling et al. generate optimal trajectories [206] for autonomous vehicle operating in urban traffic. It considers trajectories of dynamic obstacles and produces trajectories that realize tasks such as lane-changing, merging, distance-keeping and velocity-regulation. A family of polynomial curves are used to describe lateral and longitudinal movements in body-fixed frame of reference. Lateral and longitudinal movements are jointly optimized to form optimal trajectories considering minimization of jerk, velocity-regulation and collision avoidance. Generally, motion uncertainty and perception uncertainty are not included in the formulation of the problem.

2.3 Handling Uncertainty

In [184], motion uncertainty is handled in through the use of precomputed transition probabilities over differing sea states and contingencies are handled through a localized search in the space of

probabilistic transitions. Under perception and motion uncertainty, MDPs are transformed into POMDPs (Partially Observable Markov Decision Processes).

Belief-space (pose $\times$ covariance space) search is employed in static maps [31, 115] to capture motion uncertainty and compute plans that minimize final time and covariance.

In [9], the Probabilistic Road Map (PRM) idea is extended for belief spaces using MDPs and feedback controllers. Feedback controllers are designed to drive the agent into a sampled state in belief space called a FIRM node. Then, through the use of a precomputed policy for the MDP over FIRM nodes, a sequence of controllers that drive the agent from the initial state to the final state is obtained. This way, POMDPs is made computationally tractable for larger scale problems involving only static obstacles. It is hard to extend these methods to work in domains with spatio-temporal disturbances.

Rafieisakhae et al. solve certain class of POMDPs [155] using a two-stage process. They attempt to break the curse of history by moving the robot towards states for which observation uncertainty is lower. Doing this allows for a more accurate knowledge of the robot state as a function of time. This in turn allows for better trajectory plans even though the robot may deviate significantly from an initially planned path.

When non-linear inequality constraints (e.g., obstacle avoidance constraints) are certain, one can easily verify if these constraints are satisfied or not satisfied. However, when non-linear inequality constraints are uncertain and their probability distribution has unbounded support, one cannot deterministically claim these constraints are satisfied – only a probabilistic claim can be made. Thus, instead of hard/crisp inequality constraints, we have a chance constraint (probabilistic constraint). An example of a chance constraint is

$$P(g(x) \leq 0) > 0.99$$

Here, $g(x) \leq 0$ cannot be satisfied perfectly. But, the chance constraint requires that $g(x) \leq 0$ be satisfied with a probability of 0.99.

In [23, 183], chance constraints are used to guarantee the failure probability to be below a certain prescribed value while finding paths through static obstacles defined by convex polygons. [23] introduces the notion of risk budget and risk allocation. It allocates a large part of the risk budget to trajectory segments where collisions are more likely, and only small part of the budget to trajectory segments where collisions are less likely. Dynamic obstacles are not considered in these works.

Toit and Burdick use a belief-space formulation [196, 47] and use chance constraints to avoid dynamic obstacles. By incorporating the fact that a new measurement of the robot state will be made in the future and the robot controller will attempt to regulate based on that measurement, they are able to produce more aggressive trajectories in the presence of uncertainties.

A similar approach is taken in [202] where apriori probability distribution of state and control is computed using a linearized dynamic model under LQR control and a Kalman filter based state estimator. The idea is to analyse in advance how the trajectory of the robot will evolve over time. The uncertainty in the trajectory is then used to perform probabilistic collision detection along segments of the trajectory.

In [131] and [66], terrain variation and its effect on the low-level trajectory tracking error is considered in the planning framework. In the trajectory tracking context, tracking error is typically split into two parts: cross-track error and along-track error [169, 114]. Trajectory tracking error is characterized in terms of cross-track and along-track error for various terrain types. And, it is modeled as a function of terrain. The probability of successful navigation through the terrain is computed during search via a cost function that uses terrain-dependant error model.

Bry et al. [29] produce trajectories by reasoning about collision avoidance in belief space. It uses the fact that a motion plan is going to be executed with a linear state estimator and controller to prediction future distributions of the state vector. The goal is to reduce uncertainty

by exploiting measurement regions (regions where measurement is accurate) and also find low cost paths.

A similar approach is taken by Berg et al. in [203]. The main novelty arises from the fact that it does not perform discretization of the state-space, does not use maximum-likelihood observations and runs in polynomial time. Instead, it makes a quadratic approximation the value function and approximates the belief dynamics using a EKF(Extended Kalman Filter). It considers the robot dynamics model as well as the perception model.

In [145], a learning-based approach is taken to account for motion uncertainty. Samples of executed trajectories on some representative terrain is used to train a Gaussian Process Regression (GPR) based model. This model is then used in the MDP-based policy planner. This allows the MDP-based policy planner to handle traversal of rigid terrain and also deformable terrain such as small rocks.

A Monte Carlo sampling scheme is used to compute collision probabilities [104]. We favor the Monte Carlo sampling approach as it is amenable to parallelization. However, this approach requires extensive sampling when high confidence estimates of collision probabilities are required especially when chance of a collision is rare. This problem can be alleviated if adaptive sampling techniques are used [164]. Computing exact collision probabilities is important when probabilistic constraints are used. When only relative safety of trajectories are considered, the exact collision probability does not matter. In [79], the reported planning times are suitable in a reactive planning context. A semi-analytical method to calculate collision probability [143].

Accurate prediction of trajectories of moving obstacles [137] is a fundamental requirement for deliberative trajectory planning. Several threads of research work are in progress to provide predictions. An intention model is used in [168] to predict the motion of civilian vessels and develop contingency aware plans. In the marine domain, AIS (Automatic Identification System) data [198] can be used for coarse grained trajectory prediction. Though AIS is susceptible to communication interruptions [105], when it is used in conjunction trajectory prediction methods

using GPR [92, 57, 195, 14], can produce good estimates for long-term planning purposes [113]. Waves generated by moving vessels can be forecast upto 180 s into the future using radar systems [71, 100]. In this thesis, it is assumed that such a perception system is available in the sensing suite to aid deliberative planning.

2.4 Handling High-Dimensional Configuration Spaces

A prime example of trajectory planning in high-dimensional configuration space is manipulator trajectory planning. In this context, trajectory planning is typically done in two levels. In the first level, paths satisfying problem constraints are generated. Problem constraints can consist of many constraints such as contact constraints (e.g., tracing workpiece borders, touching objects), manipulability constraints, grasping constraints and collision avoidance constraints. At the second level, trajectories are computed by meeting robot dynamics constraints. This section reviews methods used in the first level. Specifically, it considers only methods that handle collision avoidance constraints (e.g., point-to-point trajectory planning for manipulators).

Point-to-point trajectory planning for manipulators is a well studied problem and researchers have developed many methods spanning potential-field-based methods [204, 89], sampling-based methods [87, 95, 86, 79, 59], search-based methods [150, 211, 35, 36, 60, 61, 191], and optimization-based methods [84, 219, 221, 193].

Search-based methods such as [10, 146] have been successful in motion planning for high degree of freedom systems. In particular, Phillips et al. propose a trajectory planning method that utilizes plans from past episodes to accelerate the search. The premise of this method is that despite dynamically changing environments, large portions of the environments that the robot encounters are static. The planner should exploit this fact and learn to solve similar problems faster as more experience is gained. In this method, past solutions are stored in terms of an experience graph that allows this method to work for unseen start/goal configurations. For unseen

start/goal configurations, the method attempts to find solutions that pass through the experience graph. This way regions explored by the experience graph is effectively reused. When new regions need to be included, the experience graph is updated seamlessly. By reusing and updating the experience graph every new planning query, this method is able to rapidly learn to focus planning effort in promising regions of the search-space.

In [10], Aine et al. propose a search-based method that utilizes multiple heuristics to solve planning problems. The core idea is that a single guiding heuristic that works well in all regions of the search space is hard to generate. Often a single heuristic fails make progress as enters a heuristic depression region (a region where there is a large discrepancy between the heuristic value and the actual cost-to-go). Instead, multiple inadmissible heuristics can be generated. Each of these heuristics can be given an opportunity to make progress in the search space in a round-robin fashion. These different heuristics complement each other in such a way that not all of the heuristics experience a depression region at the same time. This allows the search to proceed even when a few of the heuristics are not effective.

Sampling-based approaches have been extensively used for motion planning of high degree of freedom systems. Many variants of Probabilistic Roadmaps (PRMs)[87], and Rapidly-exploring Random Trees (RRTs)[95] have been developed. Many techniques have been applied to achieve these goals. These include introducing a biased sampling scheme, providing avenues for anytime convergence, heuristic guided sampling, and allowing a specified factor of sub-optimality [12, 18, 59, 86, 136, 187, 96, 107, 220, 190, 28, 159, 103, 213, 160, 218, 38, 162, 216, 217, 11, 126, 111, 68, 129]. In [49], recent developments of RRT and its variants are outlined. Recent approaches such fast matching trees (FMT*)[123] and batch informed trees (BIT*)[59, 58] attempt to outperform *RRT* in high dimensional problems.

Traditional search-based methods suffer the curse of dimensionality. When the dimensionality of the search space is large, search-based methods need to explore an astronomical number of configuration points to compute a sequence of actions leading to the goal. Typically, trajectory

planning problems in five or greater dimensional space are generally computationally very expensive and take a lot of time to solve. This is due to the systematic evaluation of potential actions at each configuration space point which provides completeness at the expense of planning time.

Traditional sampling-based methods do not face this combinatorial blow-up as they sample in configuration space. Sampling-based algorithms such as RRT [107] and PRM [87] capture the connectivity of free configuration space through the incremental construction of a free space connectivity graph. However, sampling configurations point by point to establish connectivity of the free-space is inefficient and potentially useful information regarding the local structure of the free configuration space is discarded.

Recent works have attempted to incorporate the information about local configuration space in various ways. Firstly, caching methods are used in [142] where the configuration space is learned through a sequence of configuration space validity queries. These methods build a predictive model of the configuration space to quickly answer validity queries. The method presented in [41] not only learns the configuration space but also engages in active learning to improve prediction accuracies.

In other works such as [8, 174, 102], the local configuration space information is directly used in the search process. In [174], free configuration space volume information is used to build free space ball in configuration space. A tree of such balls is constructed using RRT and tree search is performed over the tree of balls to find a sequence of balls leading to the goal configuration. A similar method is used in [143], [8]. [102] builds on [152], extends it and provides a method to approximate the free space volume in configuration space. An efficient method to construct volumes of free C-space (i.e. bubbles of free C-space) was first introduced in [152]. The basic idea behind the method is to use workspace clearance-distance information to analytically define

a volume in C-space that is guaranteed to be collision free. A bubble $\mathcal{B}(\mathbf{q}, d_c) \in C_f$ is a region of free C-space around a configuration $\mathbf{q}$ defined as

$$\mathcal{B}(\mathbf{q}, \mathbf{d_c}) = \{\mathbf{y} | \sum_{\mathbf{i=1}}^{\mathbf{n}} \mathbf{r_i} |\mathbf{y_i} - \mathbf{q_i}| \leq \mathbf{d_c}\} \tag{2.1}$$

where d_c is the clearance distance between the robot and the closest obstacle in the workspace. And, r_i is the radius of a virtual cylinder that is coaxial with the rotational joint i and encompasses the rest of the robot from link i onwards till the end-effector. Details can be found in [152, 101]. Lacevic et al. extended it by introducing the notion of complete bubbles [101] which can be approximated by a set of *spines* (i.e. rays emanating from $\mathbf{q}$). Adaptation laws to determine the fidelity of the approximate volume is presented. Using these better approximates of the free space volume, RRT style search is performed to yield a solution. Through the use of RRT method in constructing the search tree, the method is susceptible to missing some promising free space volumes and prone to finding sub-optimal solutions. Furthermore, if free space volume approximates are generated through caching methods such as [41], then these volumes may not be guaranteed to be free of obstacles. This possibility is explored in [174].

Researchers have also combined sampling-based approaches with Jacobian-based control to improve performance[182, 181].

Other ideas for improving computational performance include examining only promising subspaces of the overall search space [60, 62, 61], performing a multi-stage search [148, 147, 149], and the use of motion primitives [35, 36, 132].

Each method targets one or more of the following goals.

- Reducing planning time
- Increasing solution quality
- Providing optimality guarantees

In the context of bi-directional tree search, these goals are achieved by various techniques that modify:

- How sampling is performed
- How promising nodes are identified
- How promising directions to explore are found
- How small steps (i.e., extensions) are performed
- How two trees are connected

2.4.1 Efficient Collision Detection

Efficient collision-detection and effective sampling distribution are fundamental requirements for the efficiency of RRT-style planners [138]. Collision detection dominates planning time; efficient collision-detection reduces planning time. Effective sampling not only reduces planning time but also increases solution quality by focusing on regions where high-quality solutions are likely to occur.

Recent efforts to improve efficiency of collision-detection try to build approximate models of $\mathcal{C}$-space [85, 142, 41] or attempt to find mathematical conditions to avoid unnecessary collision-detection queries [152, 101, 139].

2.4.2 Sampling Strategies

Workspace cues have been used to guide sampling-based approaches [201, 98, 219, 158, 90]. Rickert et al. follow a two stage approach [158] where a RRT-based planner operating in C-space leverages workspace cues. Workspace paths connecting start and goal poses are used to bias the search into promising regions in [158]. As a precomputation step, a search-based method is used to find a workspace path for the end-effector. This workspace path provides workspace cues that focus

RRT sampling and attempts to drag the end-effector along the workspace path. This method was deliberately designed as an incomplete method. This method works well in uncluttered workspaces and in situations where connectivity in $\mathcal{C}$ is largely influenced by that of the $\mathcal{W}$-space (e.g. configuration spaces of mobile manipulators). It behaves like a potential field method when $\mathcal{W}$-space connectivity mirrors $\mathcal{C}$-space connectivity and works very well for many practical scenarios involving mobile manipulators and less cluttered manipulator workspaces. However, these methods suffer when there is a large discrepancy between the connectivity in the cluttered workspace than that of the high-dimensional configuration-space.

In [16], a novel sampling strategy is proposed. It uses a learning-based approach to perform approximate collision detection. It also ensures the sampled point is in a relevant region before continuing with an accurate collision checker. As this method is presented only in 2D configuration spaces, it is unclear if this method is applicable to higher-dimensions.

Kiesel et al. use a focusing scheme [90] where a subspace of $\mathcal{C}$-space (i.e. essentially $\mathcal{W}$-space) is discretized into regions. By incrementally establishing connectivity between these regions in an online fashion, easy regions of the $\mathcal{C}$-space are identified. Planning effort is focused into easy regions as they are uncovered. This scheme also suffers when there is a mismatch between connectivity in W-space and C-space.

Although techniques that provide a diverse set of paths as in [21, 153, 94, 214] can be employed, explicitly constructing a workspace path for the end-effector in a cluttered workspace while efficiently pruning workspace paths having infeasible C-space counterparts seems to be just as hard as the original problem itself. Nevertheless, if we insist on utilizing workspace cues, then quickly detecting that a workspace path is misguiding is crucial for conserving computational resources. Clearly, there is a trade-off. We could invest computational resources attempting to finesse through a workspace path and succeed or admit failure and give up after a while resorting to uniform sampling. We require the focusing scheme to reliably predict a misguiding workspace path and give up quickly.

Other workspace guidance methods as in [133, 112] will not work for a cluttered 3D scene unless the workspace is carefully decomposed into a tractable number of disjoint polytopes. Depending heavily on any particular workspace-focused-sampling strategy may be detrimental to performance.

2.4.3 Node and Target Selection Strategies

In [205, 173, 112], the Jacobian pseudo-inverse is used to generate target configurations. Specifically, Weghe et. al. use a goal biasing scheme that drives the search towards workspace goal regions [205]. Jacobian pseudo-inverse is a potent technique for bridging the workspace and the configuration space. While it helps cover large distances in the workspace, intricate configuration space maneuvers cannot easily be performed using this technique alone. When solutions pass through narrow channels in the configuration space, sampling-based approaches tend to miss them unless special care is taken to sample these channels. Methods such as Retraction-based RRT [141],Spark PRM [170] and [187] focus on these problems. Variants such as RRT*, PRM* [86] guarantee almost-sure asymptotic optimality. However, without effective focused-sampling, neither probabilistic completeness nor asymptotic optimality is practically appreciable cf. RRT*[86], BIT*[59].

In [154], a novel target selection method is employed. A Contractive Auto Encoder (CAE) is trained using a fixed number of colliding configurations. Random maps and random start/goal configurations are used to generate optimal paths. These paths are segmented into many linear segments. Given the start point of a segment, goal configuration and the encoded obstacle map, a neural network is trained to produce the end point of the segment. Any lacking in the representation of the obstacles can severely affect performance. The main difficulty in this method is being able to faithfully represent an unseen environment using the CAE. And, the MPNet does not adapt to changing tree-state, rather it just represents a static policy determined through

only through the initial sampling of the obstacles. Adapting to tree-state changes is required for robustness against any representational lacking.

End-to-end Machine Learning (ML) methods to solve the motion planning problem are generally not preferable as they may have unreliable performance (e.g., completeness issues). They typically require large training datasets. It is better to use Deep Learning (DL) based techniques as acceleration scheme for existing methods. In this spirit, a novel target selection strategy is proposed in [74] that uses Q-learning. It attempts to unify the benefits of PRM (i.e., multi-query method) and also avoids the need for graph repair when obstacles shift under one method. In this work, $\mathcal{C}$-space is approximated by fitting Radial-Basis-Functions (RBFs) using pre-sampled $\mathcal{C}$-space samples and their collision labels. The trained RBFs output a number denoting the approximate degree of collision. Distance functions representing the configuration space distance from an arbitrary state to the goal and workspace distance between EE position of the arbitrary state to that of the goal are defined. The Q-value determined through weighted combination of the distance function and RBF approximation. The weights are learnt online.

Another method that is meant to be used as a multi-query method is OracleNet [20]. It uses Long-Short-Term-Memory (LSTM) networks to learn the optimal next move to make in the configuration-space. While OracleNet has an advantage in terms of its applicability in a real-time setting (i.e., an one-step move can be immediately generated to move in a dynamic environment), it is likely to not work well for cluttered environments where a more intricate configuration-space maneuver needs to be made. In other words, even if obstacle context is added, it unlikely that LSTM networks can generalize and learn the required behavior for previously unseen and cluttered environments. This particularly true when narrow passages in static environments are missed by the training set. The main difficulties in applying this method are being able to sample effectively in narrow passages (for generating training set) and finding a way to impart high-fidelity obstacle context to the network.

For example, in [76], the basic functions required by sampling-based methods are replaced by encoder, decoder and classifier networks. The basic idea is to move the original problem into a latent-space and solve the latent-space problem. Training these networks requires a lot of data.

The problem of tuning *strategy vectors* (i.e., planner parameters) can be posed as reinforcement learning problem [165] where we learn a policy that sets planner parameters as a function of the search-state. The main drawback is developing a feature vector to encode the search-state which includes a dynamically growing tree. Specifically, we can apply a Deep Reinforcement Learning (DRL) approach to automatically extract features. If DRL scheme is used, we require a graph-embedding that accommodates a dynamic graph-network such as the two search-trees and the collision data collected so far. These entities grow in size over time. The embedding needs to be permutation invariant, node label invariant. Finding an embedding for these dynamic entities is challenging. Methods such as [124, 197, 64, 65, 63] deal with dynamic graphs but are not immediately applicable in our context. Another difficulty in applying DRL is the fact that motion planning problems only have sparse rewards. For instance, we get a reward (i.e., solution found) after the whole episode of executing a policy. Moreover, the reward is stochastic which further affects the convergence properties of the DRL method.

2.4.4 Connection Strategies

Variants such as RRT*, PRM* [86] guarantee almost-sure asymptotic global optimality. BIT* [59] uses a configuration space focusing schedule and provides optimality guarantees. While these methods are guaranteed to find optimal solutions eventually, being able to utilize workspace guidance when it is accurate can greatly complement their strengths [34]. In [45], a workspace RRT is grown from the goal position. The resulting tree is used to bias the growth of the configuration space tree. This method works well for mobile manipulation in uncluttered environments.

2.4.5 Incorporating Human Assistance in Planning

In many applications, there is always a human-operator managing robot operations even if these robots are automated. Often these operators perform a final verification before the automatically-generated trajectories are executed. Despite the use of specialized techniques, automated path-planning methods may not always find a solution within a given time budget; even if they find a solution, the human-operator may reject it due to excessively long trajectories or uncomfortable proximity to obstacles. In such cases, the human-operator may as well play a proactive role in shaping the produced paths. The field of cooperative-planning studies how human assistance can be incorporated into an underlying automated path planner. The goal of cooperative-planning is to harness the power of human intuition in focusing the automated search and produce high-quality solutions quickly.

The level of assistance by a human-operator is a crucial factor in the usability and efficiency of a cooperative-search. Denny et al. discuss the various levels of assistance in [44] such as specifying intermediate configurations, providing a rough path and providing hints [130, 50] in the form of focus regions to the underlying automated search. In [188, 19], haptic devices are used to generate intermediate configurations. Humans generate intermediate configurations slowly; intermediate configurations also pose a significant User-Interface (UI) design-challenge unless expensive haptic devices are used. Generating a rough path, though effective [44], is mentally more demanding than providing focus regions. A method of providing focus regions is given in [43]. In [77], search progress stagnation is detected using heuristic values and auxiliary metrics related to nodes that are popped out of the priority queues. Once a stagnation is identified, the user is prompted for assistance in the form of a configuration point.

Chapter 3

Dynamics-Aware Reactive Trajectory Planning

3.1 Introduction

The ability to successfully avoid collisions with dynamic obstacles is a fundamental capability needed to realize autonomous robots. The collision avoidance approach must take into account the robot's performance constraints, such as maximum achievable velocities, accelerations, braking distances, and turning radii. Collision avoidance with dynamic obstacles on uneven terrains is challenging because a robot's performance constraints change based on the robot state, terrain characteristics, terrain slope, and robot's orientation with respect to the terrain slope. Consider the stopping distance constraint as an example. The stopping distance constraint cannot be defined just based on the robot characteristics alone. The stopping distance changes based on robot's velocity. It also depends on the traction available on the terrain. The slope of the terrain affects this as well. Finally, the stopping distance is shorter if the robot is traveling uphill compared to the same robot traveling downhill. Thus, the planner must be dynamics-aware and use accurate estimates of robot's dynamic constraints based on the current state of the robot and the terrain.

Using overly conservative dynamics constraints can avoid collisions by stopping or steering the robot far from the obstacles, but lead to significant deviations from the intended trajectories and

compromise mission performance. By contrast, using constraints that overestimate robot's capabilities leads to an increased risk of collision. Therefore, using accurate constraints is important during the generation of reactive plans to avoid collision with dynamic obstacles.

In order to avoid collision with dynamic obstacles, the robot must consider modifications to the intended path by generating path alternatives and consider regulating speed along the alternative paths to find a trajectory that avoids collision and minimizes deviations from the planned nominal trajectory. This trajectory modification must be done in real-time. Based on the available time, the planner should then automatically adjust the number of options that it evaluates to ensure that computation time is tractable and does not reduce the reaction time available to the robot.

Finally, the reactive planner needs to account for uncertainty in the obstacle position and velocity when evaluating trajectories for collision risks. The uncertainty in obstacle position and velocity reduces as obstacles approach the robot. The planner should delay committing to a specific trajectory until the last possible moment to fully exploit the reduction in uncertainty in obstacle position and velocity.

Dynamic obstacle collision avoidance is a well-studied problem (see Section 2 for a detailed discussion). Popular methods include Generalized Velocity Obstacles (GVO) [208] and its variants. In this work, a methodology that is real-time, dynamics-aware and generates trajectories which avoid collision with dynamic obstacles is presented. It works by blending the path selection and the speed profile selection. Conceptually, our method can be viewed in the same family as GVO.

Our method differs from GVO in the way control/input space is partitioned. In GVO, control-space sampling and collision detection determines the available feasible control actions. These control actions impart changes to both the path and speed of the UGV. So, a dense control-space sampling is required to explore a variety of trajectories. In addition, one way time-varying dynamics constraints can be incorporated into the selection of control actions is to use costly forward simulations of at least a low-fidelity UGV dynamics model to verify feasibility of the

control actions. Otherwise, it is customary to assume an overly conservative UGV capability and use conservative control actions resulting in degradation in mission performance in some cases.

In this method, trajectory generation is decoupled using the standard approach of geometric path selection followed by speed profile selection. As a result, the geometric paths can be more spatially focused for collision avoidance avoiding the need for dense control-space sampling. Furthermore, the variation in dynamics constraints along these paths is easier to incorporate as a function of the position of the UGV on the path. For example, a path crossing a slippery or sloped region imposes acceleration constraints that can easily be incorporated into the spatio-temporal configuration space of the path along with the dynamic obstacles. These paths are chosen with the curvature and velocity limits of the UGV.

Our trajectory generator accounts for uncertainty in the obstacle position and velocities when evaluating alternative trajectories for collision risks and uses conservative values of estimated dynamics constraints to ensure that the risk of collision is minimized. It also uses deferred value binding to utilize improved estimates of obstacles' states as they come closer to the robot. Finally, it automatically adjusts the number of options being evaluated based on the estimated time to collision. A robot may encounter multiple dynamic obstacles. In order to maintain real-time performance, the approach presented in this work either groups multiple obstacles into a single composite obstacle in spatio-temporal configuration space or deals with them sequentially (i.e. prioritizing obstacles based on the estimated time to collision). This approach works well for both cases when obstacles are too close or too far apart.

3.2 Background

Autonomous operation of an Unmanned Ground Vehicle (UGV) requires a path and trajectory planner that respects both the vehicle's kinematic and dynamic constraints [106]. Typical UGV

missions may span several kilometers and computing the entirety of a dynamically feasible trajectory over such large distances is time consuming and impractical in the face of dynamic obstacles. Thus, a hierarchical planning architecture is used.

The hierarchical planning architecture is composed of three layers: (i) the Global Path Generator (GPG), (ii) the Trajectory Generator respecting Kinematic Constraints (TGKC) and (iii) the reactive Trajectory Generator respecting Dynamics Constraints (TGDC). Each layer in the architecture has its own planning horizon defined by distance or time. And, each layer sets up a reference trajectory for the layer below. In the case where a reference trajectory imposed by a upper layer is found to be infeasible by the lower layer, the lower layer raises an exception that is handled by the upper layer. In this work, the GPG described in [166] is employed. It produces a globally optimal, any-angle, geometrically feasible path on a planning horizon which spans several hundred kilometers.

The TGKC computes trajectories respecting the vehicle's kinematic constraints while tracking the geometric path laid out by the GPG. It employs a planning horizon spanning hundreds of meters. This relatively short planning horizon enables it to re-compute plans at higher frequency than the GPG. The TGKC computes a trajectory consisting of a reference path and a reference speed profile along the path. Only static obstacles are taken into consideration in TGKC while the dynamic obstacles are delegated to the TGDC.

The focus of this work is the development of the third layer of the planning architecture: Trajectory Generation respecting Dynamics Constraints (see Section 3.3). The TGDC computes trajectories that respect the dynamics constraints of the vehicle and avoids moving/dynamic obstacles over uneven terrains, planning trajectories with a horizon of up to several seconds. Its re-planning frequency is significantly higher than that of the TGKC, which is essential in the presence of dynamic obstacles. While avoiding dynamic obstacles, the TGDC also attempts to follow the trajectory provided by the TGKC as closely as possible, altering the velocity profile on the reference path. Where appropriate, the TGDC also locally alters the reference path prescribed

by the TGKC. These alterations are performed in view of vehicle capabilities on uneven terrain. An example is presented in Section 3.2 to briefly illustrate the trajectory generation process in TGDC.

Let us assume that TGKC layer has commanded a UGV to reach the top of the ramp in Figure 3.1. The UGV is on the ramp, attempting to climb it. A dynamic obstacle is poised to cross the intended path of the UGV. In this situation, it is crucial to know if the UGV is capable of accelerating (against gravity) to pass the on-coming obstacle. On the same note, if the UGV happens to be facing down the slope, it is crucial to know if the UGV is capable of slowing down (against gravity) to avoid on-coming obstacles.

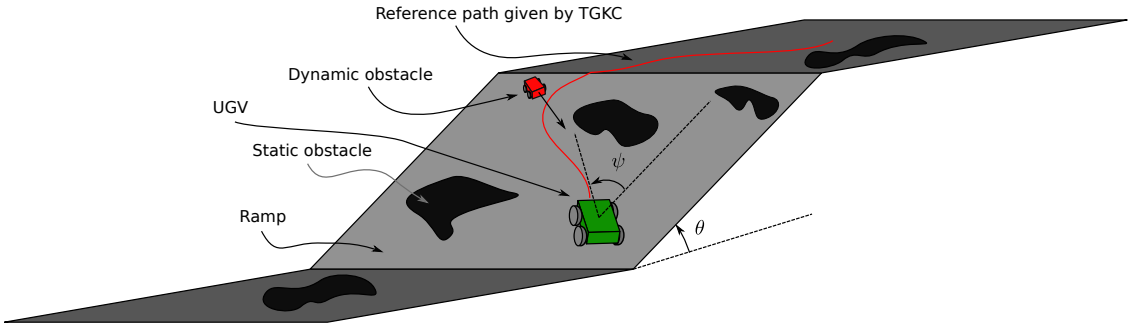

Figure 3.1: An illustrative example showing the need for vehicle-dynamics aware planning

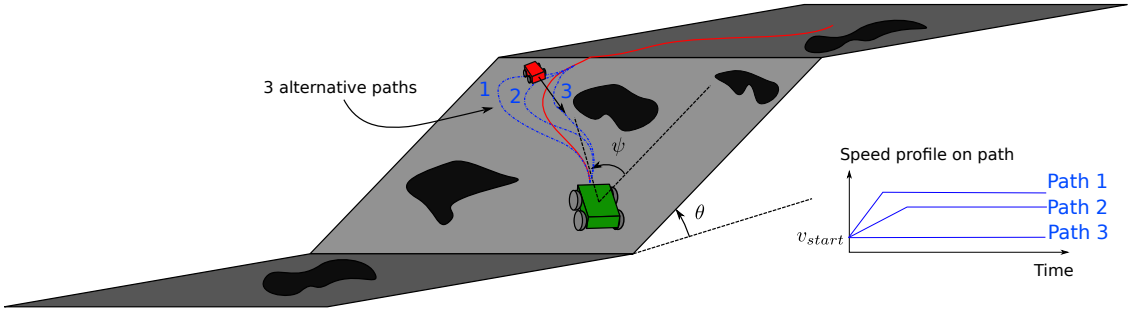

Figure 3.2: Alternative paths choices are shown. For each path, there can be many associated speed profiles that can circumnavigate the dynamic obstacle in the face of dynamics constraints. Only one speed profile is shown for each path. On each path, a few speed profiles are evaluated in TGDC to yield the best path and speed profile.

Suppose the UGV's throttle actuator is already close to saturation. If we simply assume that the UGV is capable of high acceleration and go full throttle ahead, the UGV may not able to

speed up fast enough to avoid the dynamic obstacle and end up colliding with it. Conversely, assuming overly conservative acceleration bounds for the UGV and not accelerating when it is in fact doable results in reference trajectory deviation. This means future commands to the speed of UGV have to be selected carefully. Broadly, the selected speed profiles perform one of three strategies: *pass* obstacles (by rushing ahead along the path before the obstacle crosses the path), *yield* to obstacles (by waiting for obstacles to cross the path) or continue along path with *no change* to speed.

Given that the UGV is already throttle-saturated, what other options are there to reach the top of the ramp quickly and safely? Instead of accelerating, the UGV could try modifying its path slightly to "buy" some time for maneuvering around the dynamic obstacle at the same speed. This means alternative future paths (see Figure 3.2) have to be selected opportunistically while minimizing the deviation from the reference path provided by TGKC.

The rest of the chapter covers the technical details behind the TGDC. Symbols and terms used in the following sections are briefly described below.

- $s \in \mathbb{R}$: Parameter variable in the arc-length parameterization of a curve in 3D space.
- $K(s)$: A curve $K : \mathbb{R} \mapsto \mathbb{R}^3$ in 3D space parameterized by s.
- l_K : Total length of curve K.
- $t \in \mathbb{R}$: Time.
- $v \in \mathbb{R}$: Speed on a curve K.
- Speed profile : This refers to various speeds the UGV travels at along the path. Note that this is slightly misrepresented in the figures as a position vs time function (instead of speed vs time function).
- C_K (or $s - t$ space) : A configuration space C_K defined over s and t. Each point $p_0 = (s_0, t_0) \in C_K \subset \mathbb{R}^2$ in this space represents a particular position s_0 on a certain curve $K(s)$

at a particular time t_0. The subscript K denotes the association of the space with the curve K. (see Chapter 7.1.3 [106])

- $PW_K(s_0, t_0, l, t_h)$ (or planning window) : A rectangular subset of C_K defined by the rectangle's two diagonal points $p_{lower\_left} = (s_0, t_0)$ and $p_{upper\_right} = (s_0 + l, t_0 + t_h)$ where t_h is the horizon (look-ahead) time and l is the total length of the curve K.

- $\tau_K(t)$ (or $s-t$ trajectory) : A trajectory $\tau_K : \mathbb{R} \mapsto \mathbb{R}$ defined over a planning window PW_K with time t as the independent variable. In other words, this trajectory is a specification of position on the curve K as a function of time t.

- UGV state : A tuple of numbers consisting of the bounding circle radius, position, velocity, orientation and direction (ψ, see Figure 3.1) with respect to the slope of the terrain.

- $\tau_{accel,K}(t)$: The $s-t$ trajectory on the curve K as a function of time as the UGV accelerates under full throttle along the curve. This function also depends on UGV state.

- $\tau_{decel,K}(t)$: The $s-t$ trajectory on the curve K as a function of time as the UGV decelerates under full braking along the curve. This function also depends on UGV state.

- DO : A tuple of numbers consisting of the bounding circle radius, position and velocity of a dynamic obstacle.

- $x-y-t$ obstacle : This is a projection of a dynamic obstacle onto the local terrain plane (i.e. discarding the z components)

- $C_{K,DO}$ (or $s-t$ obstacle) : This is the set of all points $C_{K,DO} \subset PW_K$ corresponding to a particular $x-y-t$ obstacle moving across the curve K. These points mark the position and time at which the UGV will be in collision with the dynamic obstacle DO. The UGV is in collision if the position of the UGV on the curve is s_{ugv} at t_n and the point $(s_{ugv}, t_n) \in C_{K,DO}$. Thus, $C_{K,DO}$ can computed by sampling points in PW_K and checking if the sampled point corresponds to a collision between the UGV and a dynamic obstacle.

- UL vertex (or upper left vertex) : This is the upper left vertex of a $s-t$ obstacle $C_{K,DO}$. $UL(C_{K,DO}) = (s_{UL}, t_{UL}) \in C_{K,DO}$ where $s_{UL} = \max_{(s,t)\in C_{K,DO}}(s)$ and $t_{UL} = \min_{(s,t)\in C_{K,DO}}(t)$
- LR vertex (or lower right vertex) : This is the lower right vertex of a $s-t$ obstacle $C_{K,DO}$. $LR(C_{K,DO}) = (s_{LR}, t_{LR}) \in C_{K,DO}$ where $s_{LR} = \min_{(s,t)\in C_{K,DO}}(s)$ and $t_{LR} = \max_{(s,t)\in C_{K,DO}}(t)$
- Velocity tuning (or speed regulation) : The process of finding a $s-t$ trajectory τ_K over C_K such that the trajectory does not pass through any $s-t$ obstacle $C_{K,DO}$. (see Chapter 7.1.3 [106])

3.3 Problem Formulation

Given:

(a) A kinematically feasible, collision free reference path $K_r(s) \in \mathbb{R}^3$ parameterized by the arc-length parameter s and a reference speed profile $v_r(s)$ over the path $K_r(s)$

(b) The current time t_{start}

(c) UGV state consisting of:

- $p_{ugv} \in \mathbb{R}^3$: The current position of the UGV
- $v_{start} \in \mathbb{R}$: The current velocity in direction of motion
- θ : The average slope angle of the terrain under footprint of the UGV
- ψ : The yaw angle with respect to the slope

(d) A look-up table of speed, acceleration and deceleration constraints indexed by θ and ψ

(e) A set of n_{do} dynamic obstacles $D = \{DO_k\}_{k=1}^{n_{do}}$ sensed by the perception system, each defined by:

- Instantaneous position measurement $p_{do} \in \mathbb{R}^3$ and the associated perception variance $\sigma_p \in \mathbb{R}$ following the Gaussian distribution $\mathcal{N}(p_{do}, I_{3\times 3} \cdot \sigma_p)$

- Instantaneous velocity measurement $v_{do} \in \mathbb{R}^3$ and the associated perception variance $\sigma_v \in \mathbb{R}$ following the Gaussian distribution $\mathcal{N}(v_{do}, I_{3\times3} \cdot \sigma_v)$
- Distance between obstacle and the UGV perception system d
- Noise saturation threshold d_t
- Position uncertainty parameter
  - $\sigma_p(d) = \begin{cases} \sigma_{p,max} \cdot \frac{d}{d_t}, & d < d_t \\ \sigma_{p,max}, & d \geq d_t \end{cases}$
- Velocity uncertainty parameter
  - $\sigma_v(d) = \begin{cases} \sigma_{v,max} \cdot \frac{d}{d_t}, & d < d_t \\ \sigma_{v,max}, & d \geq d_t \end{cases}$

(f) A set of kinematically feasible paths (computed offline) A, which locally modify $K_r(s)$ by connecting p_{ugv} to a common point $K_r(s_{end})$ on the path K_r

- An alternative path: K_m of length l_{K_m} where $K_m(0) = p_{ugv}$ and $K_m(l_{K_m}) = K_r(s_{end})$
- $A(p_{ugv}, s_{end}) = \{K_m(s)\}_{m=1}^{n_{alt}}$

(g) Full-throttle acceleration trajectory for the current UGV state $\tau_{accel,K}(t, t_0, v_0, s_0, \theta(s), \psi(s))$ and full braking deceleration trajectory $\tau_{decel,K}(t, t_0, v_0, s_0, \theta(s), \psi(s))$ defined over C_K space. s_0 denotes the current position of the UGV on $K(s)$. t_0 denotes the current time. Additionally, the tuple (t_0, v_0, s_0) defines an initial condition over $s-t$ space. $\theta(s)$ is the local slope angle of the terrain at the point $K(s)$ and $\psi(s)$ is the angle the tangent at $K(s)$ makes with the slope.

Compute: A dynamically feasible trajectory, which minimizes collision risk and minimizes reference path and reference speed deviation.

3.4 Approach

The approach used in the TGDC to compute a dynamically feasible plan is outlined in Algorithm 1. The approach is described with reference to the motivational example.

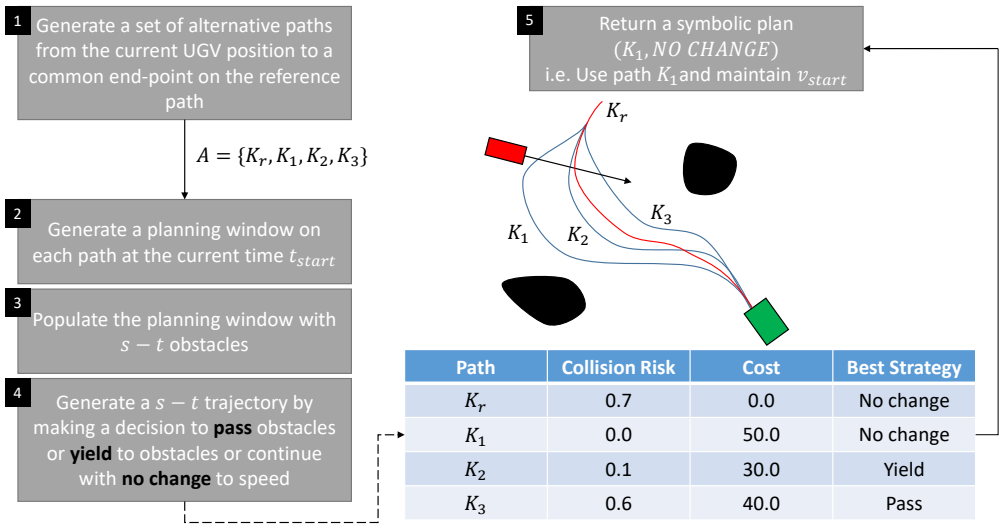

Figure 3.3: Graphical overview of the approach

3.4.1 Construction of a Planning Window

Line 3 of Algorithm 1: The kinematically feasible path $K_r(s)$ and speed profile $v_r(s)$ provided by TGKC is used as a reference by the TGDC. They are termed reference path and reference speed profile respectively.

Each time the TGDC is invoked, the TGDC generates a set of paths A. This path generation can be done online or offline. In this work, A is constructed by picking a few paths from a library of pre-computed paths described in [72]. Some of the these paths may be in collision with static obstacles when rotated, translated and placed in front of the UGV. While some other paths may not satisfy the curvature constraints imposed by the vehicle kinematics. These paths are removed from A. The number of alternative paths n_{alt} is scaled based on the time to collision with the next

Algorithm 1 Trajectory planning with dynamics constraints

procedure ComputeDynamicallyFeasiblePlan
    Depending on time to collision with the nearest obstacle, choose the number of alternative paths n_{alt} to generate.
    Generate the path set $A(p_{ugv}, s_{end})$ based on the current vehicle position and look-ahead distance.
    Obtain the dynamic obstacle state measurements $D = \{DO_k\}$ from the perception system
    for each path $K_m \in A$ **do**
        Construct a planning window PW_{K_m}
        Compute vehicle dynamics constraints τ_{accel,K_m}, τ_{decel,K_m} based on θ and ψ along the path K_m and the UGV state
        Pre-process the reference speed profile on the path using vehicle dynamics constraints to form a dynamically feasible reference trajectory
        for each $DO_k \in D$ **do**
            Obtain n_s samples of DO_k (from the Gaussian distribution) and compute the $s-t$ obstacle samples $\{C_{K_m,DO_k,n}\}_{n=1}^{n_s}$
        If combining multiple obstacles is appropriate, coalesce $s-t$ obstacles $\{C_{K_m,DO_k,n}\}_{k=1}^{n_{do}}$ to form n_s samples of a single composite $s-t$ obstacle $\{C_{K_m,DO,n}\}_{n=1}^{n_s}$
        for each strategy $\in \{PASS, YIELD, NO\ CHANGE\}$ **do**
            Compute $p_{coll,strategy}$, $c_{strategy}$ using the $s-t$ obstacle samples $\{C_{K_m,DO,n}\}_{n=1}^{n_s}$ using the method in Section 3.4.4
        Append the cost tuple $p_{coll,pass}, c_{pass}, p_{coll,yield}, c_{yield}, p_{coll,no\ change}, c_{no\ change}$ for each path option K_m and each of the strategies to the cost matrix
    Generate a symbolic plan $(K_{best\ path}, c_{best\ strategy})$ consisting of the best path option and best strategy on that path by using criteria in Section 3.4.5

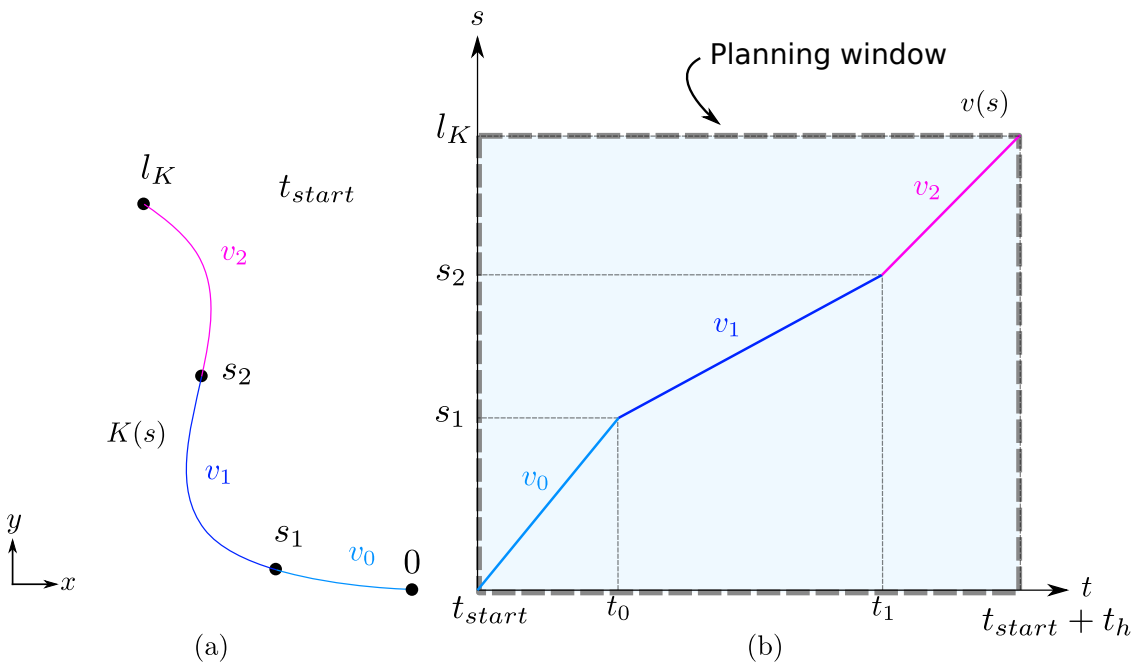

Figure 3.4: (a) Path K and (b) its corresponding planning window $PW_K(0, t_{start}, l_K, t_h)$ showing a reference speed profile $v(s)$ to follow

dynamic obstacle. This allows for fast reactive collision avoidance even when available reaction time is limited. In this work, n_{alt} ranges between 2 and 10. Each path option starts at p_{ugv} and ends at $K(s_{end})$. It deviates slightly from reference path midway. Speed profile on the alternative path could be specified using the dynamic trajectory space [178] or adapted from the reference speed profile.

For example, in Figure 3.1 and 3.3, $A = \{K_r, K_1, K_2, K_3\}$. K_r will not be included if the UGV happens to be significantly away from K_r. Each path in A is such that it starts at the current position of the UGV and ends at a common point $K_r(s_{end})$ where $s_{end} = s_{start} + l_{deviation}$. s_{start} is parameter value on K_r such that $K_r(s_{start})$ is the closest point on K_r to the current UGV position. $l_{deviation}$ is the distance over K_r in which the UGV will be deviating from the reference path K_r. $l_{deviation}$ typically has to scale with the obstacle size. In this work, $l_{deviation}$ was fixed to 10 m as obstacles were of a fixed size. Note that $K_m(l_{K_m}) = K_r(s_{end})$. For each path $K_m \in A$,

a planning window $PW_{K_m}(0, t_{start}, l_{K_m}, t_h)$ is constructed (see Figure 3.4b). Thus, the planning window is t_h wide starting at the current time t_{start} and l_{K_m} tall starting at 0.

So far, the planning windows are all blank. The next step is to populate each window with dynamic constraints.

3.4.2 Construction of Dynamics Constraints

Line 7: The terrain type (i.e. the local slope at the vehicle footprint) is used to compute the associated dynamics constraints. In this work, a simple look-up table (Table 3.1) is used to determine parameters describing the acceleration and deceleration capabilities (3.1) of the vehicle in a particular terrain type. Acceleration and deceleration constraints are specified via trajectories in the planning window. Terrain parameters and the orientation of the vehicle on the terrain affect the local acceleration and deceleration constraints. In this work, only the local slope θ under the vehicle's footprint and the direction ψ with respect to the slope are used to form equations of acceleration and deceleration. However, other parameters affecting the powertrain can be easily incorporated into these equations [178].

The upper bound on acceleration a^+ and the upper bound on deceleration a^- used in this work are shown in (3.1) and Table 3.1. For a certain initial state (s_0, t_0, v_0) and a path K, an acceleration trajectory $\tau_{accel,K}(t, t_0, v_0, s_0, \theta(s), \psi(s))$ and a deceleration trajectory $\tau_{decel,K}(t, t_0, v_0, s_0, \theta(s), \psi(s))$ can be computed (see Figure 3.5) as the double integral of the equations describing upper bounds on acceleration and deceleration along the path. Note that θ and ψ change along the path K.

Table 3.1: Scaling parameters for each terrain type

Parameter	Slope angle θ (°)		
	-10	0	10
v_{max}(m/s)	2.0	1.8	1.6
k	$0.3 + 0.2c_\psi$	0.3	$0.3 - 0.2c_\psi$
a_{min}(m/s^2)	$-0.2 + 0.1c_\psi$	-0.2	$-0.2 - 0.1c_\psi$

$$a^{+}(v) = k(v_{max} - v)$$
$$a^{-} = a_{min} \tag{3.1}$$

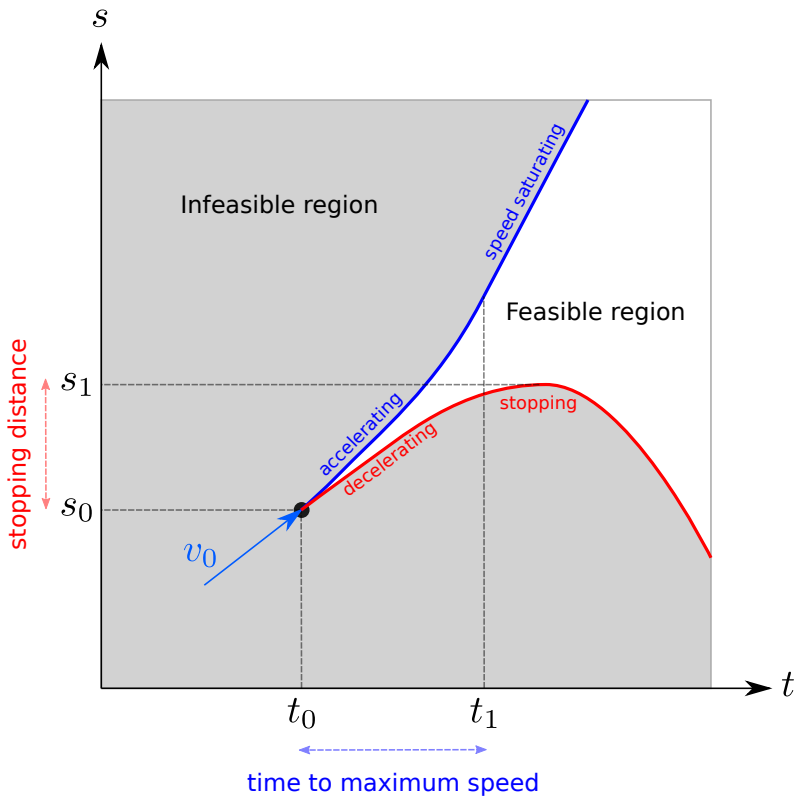

Figure 3.5: Maximum speed, acceleration and deceleration constraints are trajectories in the planning window corresponding to a certain curve.

The acceleration trajectory (blue) shows the resulting $s-t$ trajectory if maximum throttle were to be applied from the initial state (s_0, t_0, v_0) onwards. Similarly, the deceleration trajectory (red) shows the resulting $s-t$ trajectory if maximum braking were to be applied from the initial state. Thus, the region above the acceleration curve and the region below the deceleration curve are not *reachable* from the initial state. In this context, a target point in $s-t$ space is considered reachable if there is a continuous function between the initial state and the target point that does

not pass through $s-t$ obstacles or the infeasible region. At this point, planning windows have dynamic constraints. The next step is to populate them with dynamic obstacles.

3.4.3 Construction of $s-t$ Obstacles

Line 8: A given reference speed though kinematically feasible, may be dynamically infeasible when vehicle is on certain types of terrain (e.g. high-speed over uphill terrain). In such circumstances, the reference speed profile is capped at the maximum speeds allowed by the local terrains in the planning window. This pre-processing step ensures that reference speed profile is reasonably altered for the terrain conditions but not necessarily collision free.

Line 10: Collision avoidance using the velocity tuning method requires dynamic obstacles to be translated from $x-y-t$ space to the planning window PW_{K_m}. Dynamic obstacles that cut across the reference path are termed $x-y-t$ space obstacles henceforth. Figures 3.7 and 3.6 collectively illustrate the relationship between $x-y-t$ space and $s-t$ space. For example, the $s-t$ obstacle $C_{K,DO}$ corresponding to the dynamic obstacle cutting across the path K in Figure 3.6 is computed by moving the inflated obstacle time step by time step from $t=t_0$ to $t=t_1$ and noting the intersection between the inflated obstacle and the path. Thus, in this case, for the path K, $C_{K,DO}$ is $[s_a, s_b] \times [t_0, t_1]$ as shown in Figure 3.7. Note that the subscript m is omitted from K_m for notational simplicity. However, it is to be understood that K has to be replaced with each of the paths in A to yield path specific $s-t$ obstacles such as $C_{K_m,DO}$.

3.4.4 Evaluating Strategies for Obstacle Avoidance

Given a $s-t$ obstacle, a schematic view of two possible trajectories is illustrated in Figure 3.7. The two possible trajectories, *pass* and *yield*, are computed over the time range $[t_{start}, t_{end}]$. The two classes of speed profiles maneuvering around the $s-t$ obstacles are termed *strategies*. When the UGV executes the *pass* strategy, the UGV passes in the front of the dynamic obstacle before tracking reference speed. When the UGV executes the *yield* strategy, it slows down to pass the

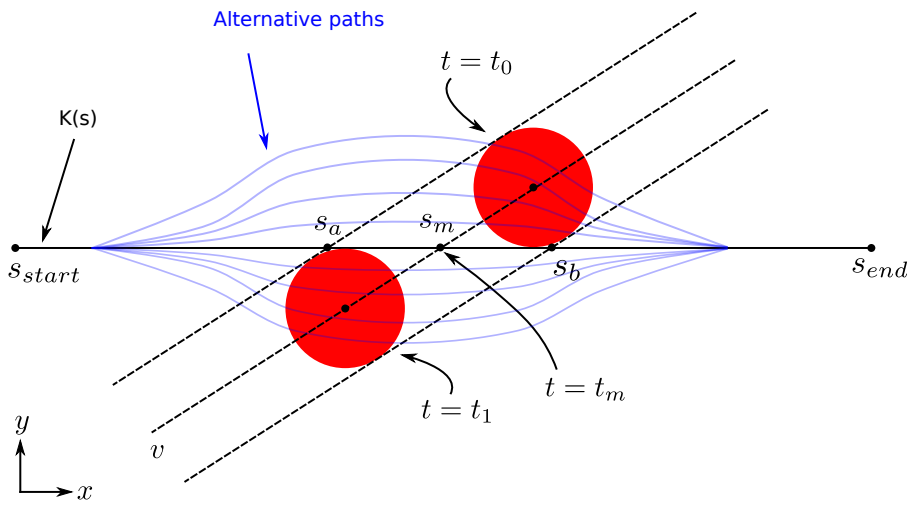

Figure 3.6: Snapshots of an inflated (vehicle radius + obstacle radius) $x - y - t$ space obstacle as it passes diagonally across a few paths at velocity v. (also see Figure 3.7)

dynamic obstacle before tracking reference speed. There is a third strategy *no change* that is not shown. This strategy is applicable when reference speed profile does not intersect with the $s - t$ obstacle and hence, the UGV can follow the speed profile with no changes.

Multiple dynamic obstacles cutting across the path are either dealt with sequentially one after the other or coalesced into a composite obstacle depending on how these obstacles are spaced apart in the planning window. Figure 3.8 shows how two $s - t$ obstacles are combined into one single $s-t$ obstacle by obtaining the bounding box of C_{K,DO_1} and C_{K,DO_2}. Section 3.5.4 specifies a condition to determine if multiple obstacles should be merged depending on Δs and Δt. Either way, multiple dynamic obstacles are reduced to a single $s-t$ obstacle to be dealt within a planning window. This way, the pass and yield strategies developed so far still work the same way even with multiple obstacles in the planning window.

Before any informed decision can be made regarding the strategy to choose, perception uncertainty associated with the dynamic obstacle state needs to be incorporated. In order to incorporate dynamic obstacle state (position and velocity) uncertainty and assess the risk associated with each strategy, the uncertainty model of a $x - y - t$ obstacle has to be utilized. Multiple realizations of the dynamic obstacle state is sampled from the Gaussian distribution yielding samples of position

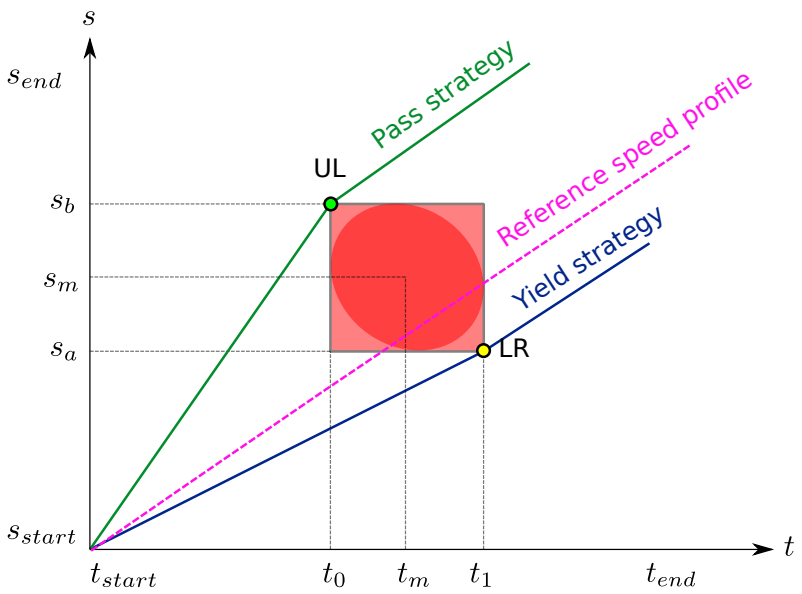

Figure 3.7: A planning window in which a $s-t$ space obstacle (dark red) and its bounding box approximation (light red) are shown. The reference speed profile is infeasible as it intersects the $s-t$ obstacle. Two trajectories start at (s_{start}, t_{start}) in the planning window. The passing trajectory (green) accelerates in front of the obstacle using the upper left (UL) vertex. The yielding trajectory (purple) decelerates and yields to the obstacle using lower right (LR) vertex. (also see Figure 3.6)

and velocity. In this work, a sample size n_s of 50 is used. For each sample $p_{do,n}$ and $v_{do,n}$, a corresponding $s-t$ obstacle $C_{K,DO,n}$ can be computed. A few samples of $s-t$ obstacles corresponding to a dynamic obstacle is illustrated in Figure 3.9. In order to circumnavigate the obstacle, there are two strategies that can be employed - *passing* and *yielding*. A passing strategy involves accelerating and passing in front of the obstacle, which requires targeting and reaching the upper left (UL) vertex from the initial state (s_0, t_0, v_0). Similarly, a yielding strategy involves decelerating and letting the obstacle pass, requiring targeting and reaching the lower right (LR) vertex from the initial state. For a passing strategy, the risk of collision can be estimated as the ratio of the number UL vertices that are unreachable to the total number of UL vertices. Similarly, for a yielding strategy, the risk of collision can be estimated as the ratio of the number of LR vertices that are unreachable to the total number of LR vertices. For the passing strategy shown in Figure

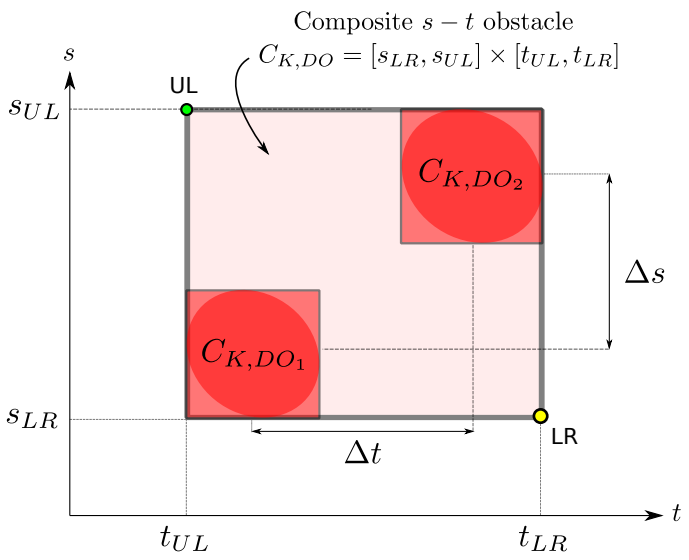

Figure 3.8: An example of how two $s-t$ obstacles C_{K,DO_1} and C_{K,DO_2} are merged into one single $s-t$ obstacle $C_{K,DO}$

3.10, the collision risk is 1/6 since there is only one UL vertex that is situated in the infeasible region. In other words,

$$R(p) = \begin{cases} 1, & \text{if point } p \text{ is not reachable} \\ 0, & \text{if point } p \text{ is reachable} \end{cases} \tag{3.2}$$

$$p_{coll,pass} = \frac{1}{n_s} \sum_{n=1}^{n_s} R(UL(C_{K_m,DO,n})) \tag{3.3}$$

$$p_{coll,yield} = \frac{1}{n_s} \sum_{n=1}^{n_s} R(LR(C_{K_m,DO,n})) \tag{3.4}$$

For the no change strategy, the ratio of $s-t$ obstacles instances intersecting the reference speed profile to the total number of $s-t$ obstacle instances is taken as the collision risk. This way perception uncertainty and satisfaction of dynamics constraints are taken care of. Note that the notion of collision employed here not only involves a physical collision but also counts inability to satisfy dynamics constraints as a collision.

In addition to collision risk, each strategy has an associated reference trajectory deviation cost. Section 3.4.6 describes the cost function used to evaluate a strategy. Once collision risk

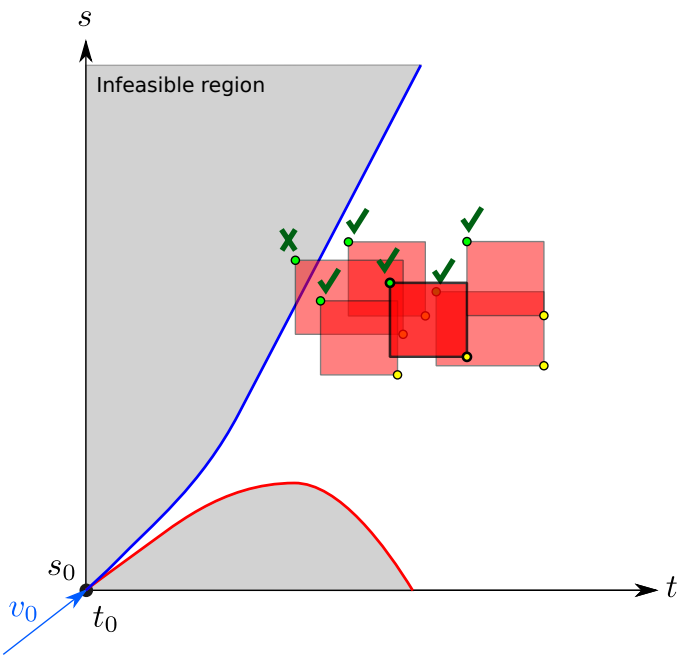

Figure 3.9: Qualitative example of risk assessment for a passing strategy where 6 samples of a *st* obstacles on a particular path are used. One of the samples has the upper left (UL) vertex in the infeasible region. Thus, the collision risk is 1/6 for the passing strategy

and trajectory deviation cost are computed for each strategy, they are assembled into a 6-tuple containing collision risk p_{coll} and reference deviation cost c for each of the three possible strategies that can be attempted over the reference path. An example of such a tuple is shown in the first row of Table 3.2.

Table 3.2: An example of the cost matrix for 4 path choices

Path choice	Strategy choice					
	Pass		Yield		No change	
	p_{coll}	c_{pass}	p_{coll}	c_{yield}	p_{coll}	$c_{no\,change}$
K_r	0.5	37.2	0.0	43.3	0.9	0
K_1	0.1	45.2	0.0	108.6	0.1	40
K_2	0.0	77.2	0.0	103.3	0.0	50
K_3	0.2	63.2	0.1	93.3	0.0	60

3.4.5 Generating Dynamically Feasible Plans

Line 13: For each path in A, each strategy is evaluated. And, a cost matrix is populated to form the entire table shown in Table 3.2.

Line 15: A combination of path option and strategy pair is termed as a *symbolic* plan. For example, $(K_3, YIELD)$ is a symbolic plan. The risk assessment method implicitly handles the dynamics constraints. Hence, picking symbolic plans with zero risk satisfies dynamics constraints. In some scenarios, there might be multiple symbolic plans that are risk free. In these cases, the expected cost is used to favor one symbolic plan over the other. Conversely, in some scenarios, there might be no symbolic plan that is risk free. In such cases, the symbolic plan exhibiting the least risk is chosen without regard for costs. An exception is raised and fed to the higher-level layer to trigger a replan.

Note that the symbolic plan does not specify any specific speed or specific coordinates on the planning window that has to be targeted and reached. Rather, it only specifies an abstract class of trajectories over the planning window. A value binding operation that converts the symbolic plan to a *numerical* plan occurs only when the actual obstacle is observed at a closer range during execution. Being at closer range, more accurate $s-t$ obstacles are observed and this accuracy is exploited to minimize reference trajectory violation while retaining safety. This value binding operation is needed because the perception system may be running at a higher frequency and thus, provide fast updates to dynamic obstacle measurements. During the value binding operation, new samples of $s-t$ obstacles are used to construct a worst case $s-t$ obstacle by obtaining a bounding box around the samples (see blue rectangle in Figure 3.11). If the strategy decision was to *pass*, the planner defaults to following the reference speed profile (i.e. no change strategy) till a critical time after which value binding operation is triggered. This critical time depends on the acceleration curve as shown in Figure 3.11. For the pass strategy, the value binding operation uses the UL vertex of the worst case $s-t$ obstacle and computes the command velocity along the chosen path as $v_{command} = s_{UL}/(t_{UL} - t_{start})$. Similarly, if the strategy decision was to *yield*, the planner defaults to following the reference speed profile till a critical distance after which value binding operation is triggered. This critical distance depends on the stopping distance associated with the deceleration curve. For the yield strategy, the value binding operation uses the LR vertex

of the worst case $s-t$ obstacle and computes the command velocity along the chosen path as $v_{command} = s_{LR}/(t_{LR} - t_{start})$. The rationale behind such triggering is to incur as little reference trajectory deviation as possible and wait till it is absolutely necessary to act to avoid collision. Value binding triggered this way minimizes reference speed deviation. After the *pass* or *yield* maneuver, the planner resumes following the reference speed profile.

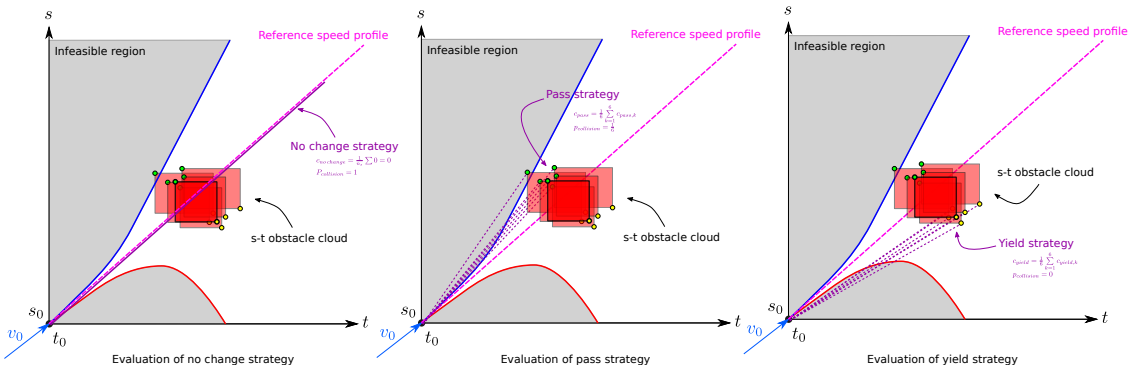

Figure 3.10: Evaluation of strategies

3.4.6 Trajectory Deviation Cost Function

A distance based metric is used to evaluate symbolic plans consisting of a path choice and a strategy choice. The trajectory deviation cost consists of path deviation penalty c_{path} incurred while using alternative paths and speed deviation penalty incurred when passing and yielding

strategies are used . The cost of pass, yield and no change strategies on a certain path are computed as follows:

$$c_{path} = \begin{cases} l_{K_m}, & \text{if path is } K_m \\ 0, & \text{if path is } K_r \end{cases} \tag{3.5}$$

$$c_{pass} = \frac{s_{UL} + s_{LR}}{2} \frac{|v_p - v_r|}{v_r} + c_{path} \tag{3.6}$$

$$c_{yield} = \frac{s_{UL} + s_{LR}}{2} \frac{|v_y - v_r|}{v_r} + c_{path} \tag{3.7}$$

$$c_{no\,change} = 0 + c_{path} \tag{3.8}$$

The cost function captures the distance over which speed violation happens and also weights it by the degree of speed deviation from the reference speed v_r (Figure 3.10). Note that the cost computed by this cost function corresponds to only one sample of the $s-t$ obstacle. Over n_s samples selected from a $s-t$ obstacle distribution, an expected cost can be computed for each of the strategies as follows:

$$E[c_{strategy}] = \frac{1}{n_s} \sum_{k=1}^{n_s} c_{strategy,k} \tag{3.9}$$

3.5 Results and Discussion

The following performance metrics are used to evaluate the executed trajectory:

- Number of collisions
- Execution time: the time taken in moving from start point to end point
- Speed violation ratio: the ratio of execution time spent in violation of the reference speed profile to the overall execution time
- Path deviation distance: the distance over which reference path was not followed.

The TGDC was implemented on the scenario shown in Figure 3.12. The scenario features terrain strips with varying slopes. A point mass dynamic model of a vehicle was used. The capability of the vehicle was artificially capped at the limits allowed by the terrain. For example, if the terrain only allowed a certain maximum speed $2.0\,m\,s^{-1}$, any further throttle will have no effect after the vehicle reached $2.0\,m\,s^{-1}$. The vehicle was placed at the start point and commanded to reach the end point. Then, a kinematically feasible trajectory was supplied to the TGDC. Dynamic obstacles were positioned along the reference path at various points and with their trajectories crossing the path at various angles.

Each dynamic obstacle was assigned a trigger region, as the vehicle entered the trigger region, the dynamic obstacle was set in motion and assigned a specific speed. This speed was such that collision would occur if the vehicle continued traveling at the reference speed. Such triggering emulates the appearance of dynamic obstacles within the vehicle's sensing radius. Once triggered, a dynamic obstacle's state information is available to TGDC. Dynamic obstacle speed was also randomly varied around the nominal obstacle speed.

3.5.1 Impact of Inexact Dynamics Constraints on Performance

The TGDC's sensitivity to inexact dynamics constraints was measured by deliberately supplying a range of dynamics constraints around the correct dynamics constraints. Each dynamics constraint parameter in Table 3.1 was inflated by constraint augmentation factor δ to yield new parameters that were either more conservative or aggressive than the correct values. Figure 3.13 shows the collision rate and speed violation as the dynamics constraints are altered. Speed violation ratio has been normalized with respect to that of the exact dynamics constraints ($\delta = 0$). Figure 3.13 shows that conservative ($\delta < 0$) dynamics constraints results in zero collision at the cost of increased speed violations. On the other hand, aggressive ($\delta > 0$) dynamics constraints result in lower speed violations at the cost of a higher collision rate.

Table 3.3: Comparison of speed-regulation, path-variation against the hybrid approach

(a) Pre-critical value binding (dynamic obstacles detected on time)

Normalized metric	Collision avoidance approach		
	Path-variation only	Speed-regulation only	Hybrid
Collision rate	0	0	0
Execution time	0.95	1.05	1.0
Path deviation	1.89	-	1.0
Speed violation ratio	-	1.3	1.0

(b) Post-critical value binding (dynamic obstacles detected too late)

Normalized metric	Collision avoidance approach		
	Path-variation only	Speed-regulation only	Hybrid
Collision rate	6.34	2.65	1.0
Execution time	0.77	0.96	1.0
Path deviation	0.38	-	1.0
Speed violation ratio	-	1.2	1.0

3.5.2 Impact of Perception Uncertainty on Performance

To observe the effect of perception uncertainty on TGDC performance, the dynamic obstacle noise model parameters (position uncertainty parameter, $\sigma_{p,max}$ and speed uncertainty parameter $\sigma_{v,max}$) were varied, while the TGDC was supplied conservative dynamics constraints ($\delta = -20\%$). These noise model parameters control the interval width of the uniform distribution used in the obstacle position and velocity measurement. Note that $\sigma_{v,max}$ scales with the velocity of the dynamic obstacle. For each combination of noise parameters, 50 independent experiments were performed. The noise saturation distance threshold d_t (see Section 3.3) was set to 10 meters.

The normalized mean execution time and normalized speed violation ratio are shown in Figure 3.14. These quantities were normalized with respect to the zero noise case. No collisions were observed in these experiments. However, as noise increased, the execution time and speed violation ratio increased by up to 31.7% and 88.1% respectively. This result shows that increased dynamic obstacle state measurement noise does not compromise the safety of our method.

3.5.3 Need for Both Speed-Regulation and Path-Variation

Relying on speed-regulation alone (with no alternative paths) to avoid dynamic obstacles results in longer execution time and a larger reference speed deviation. This is undesirable when dynamic obstacles stop along the reference path, which causes the vehicle to stop on the reference path indefinitely. Similarly, avoiding dynamic obstacles by solely relying on moving on alternative paths (without speed-regulation) results in a larger reference path deviation. This is especially undesirable when no alternative paths exist (e.g. in a narrow passage). Table 3.3 compares the execution time, path deviation and speed violation ratio of the path-variation and speed-regulation approaches against the hybrid approach. No collisions were observed when dynamic obstacles were sensed sufficiently early (Table 3.3). These results show that the hybrid approach used in TGDC, combining speed-regulation and path-variation incurs a lower path-deviation and speed violation ratio at the cost of a marginal increase in execution time.

The sudden appearance of dynamic obstacles impacts collision avoidance. The value of the hybrid approach is more apparent when dynamic obstacles appear suddenly and hence, are sensed late (after the critical point). In such a situation, speed-regulation alone may not avoid collisions due to the UGV's inability to quickly decelerate or accelerate. Similarly, swerving to the extreme left or right without modifying speed, may cause vehicle to overturn. In contrast, the hybrid approach allows for these actions whilst decelerating. Allowing such actions results in lower collision rate (Table 3.3).

3.5.4 Sequential Processing of Dynamic Obstacles

The spacing between multiple $s-t$ obstacles in the planning window affects TGDC performance. If the $s-t$ obstacles are well spaced, it may be better for the TGDC to consider each $s-t$ obstacle sequentially. Otherwise, it is better to coalesce them into a single composite $s-t$ obstacle as shown in Figure 3.8. To quantitatively study the effect of spacing between two $s-t$ obstacles, a grid

of $s-t$ points is used (see Figure 3.16). Each point in the grid corresponds to the centroid-to-centroid vector $(\Delta t, \Delta s)$ between two $s-t$ obstacles (e.g. $(0,0)$ implies both obstacles are coincident, see Figure 3.8). The s and t axes of Figure 3.15 are normalized by vehicle stopping distance (corresponding to maximum speed) and vehicle stopping time, respectively. Figure 3.15 shows collisions when $s-t$ obstacles separated by 1.5 stopping distances (Δs) and 1 stopping time (Δt). This arises due to sequential processing of obstacles, where overtaking one obstacle put the vehicle on a collision course with the next. From the examination of Figure 3.15, $s-t$ obstacles with centroids less than one stopping time away from each other or less than approximately 2 stopping distances can be safely handled by creating a composite $s-t$ obstacle.

However, Figure 3.16 shows an increase in speed violation ratio up to a factor of approximately 3.7 when these composite $s-t$ obstacles are used. This trade-off must be taken into consideration while processing multiple dynamic obstacles within a single planning window.

3.6 Summary

In this work, a method to generate collision risk-aware, dynamically feasible trajectories for a UGV operating over uneven terrain is presented. This method is shown to produce collision-free trajectories when the correct dynamics constraints are used even in the presence of sensor noise.

Many recent approaches for avoiding collisions with dynamic obstacles respect the dynamic constraints of the vehicle. However, as the terrain changes, dynamic constraints imposed on the vehicle also change. A common practice is to assume conservative dynamics constraints that work on all terrains. This reduces overall mission performance. In this work, the focus is on developing a real-time dynamics-aware reactive trajectory generator which produces trajectories that avoid collisions with dynamic obstacles under varying dynamic constraints. The trajectory generator considers modifications to the intended path, generating alternatives in real-time. It also considers regulating speed along the various modified paths, finding a trajectory that avoids collision while

minimizing deviation from the intended trajectory. It accounts for uncertainty in the obstacle position and velocity while evaluating the alternative trajectories and uses conservative estimates of the vehicle's dynamic constraints to ensure collision risk is minimized. It also uses "deferred value binding", to exploit more accurate estimates of the dynamic obstacle states as obstacles approach the vehicle. It automatically adjusts the number of options being evaluated based on the estimated time to collision. In order to ensure real-time performance, the trajectory generator handles multiple dynamic obstacles by either grouping them into a single composite obstacle in the configuration space or deals with them sequentially by prioritizing obstacles based on the estimated time to collision. Simulation results are presented to show that the planner is able to effectively deal with the dynamic obstacles on terrains with varying slopes.

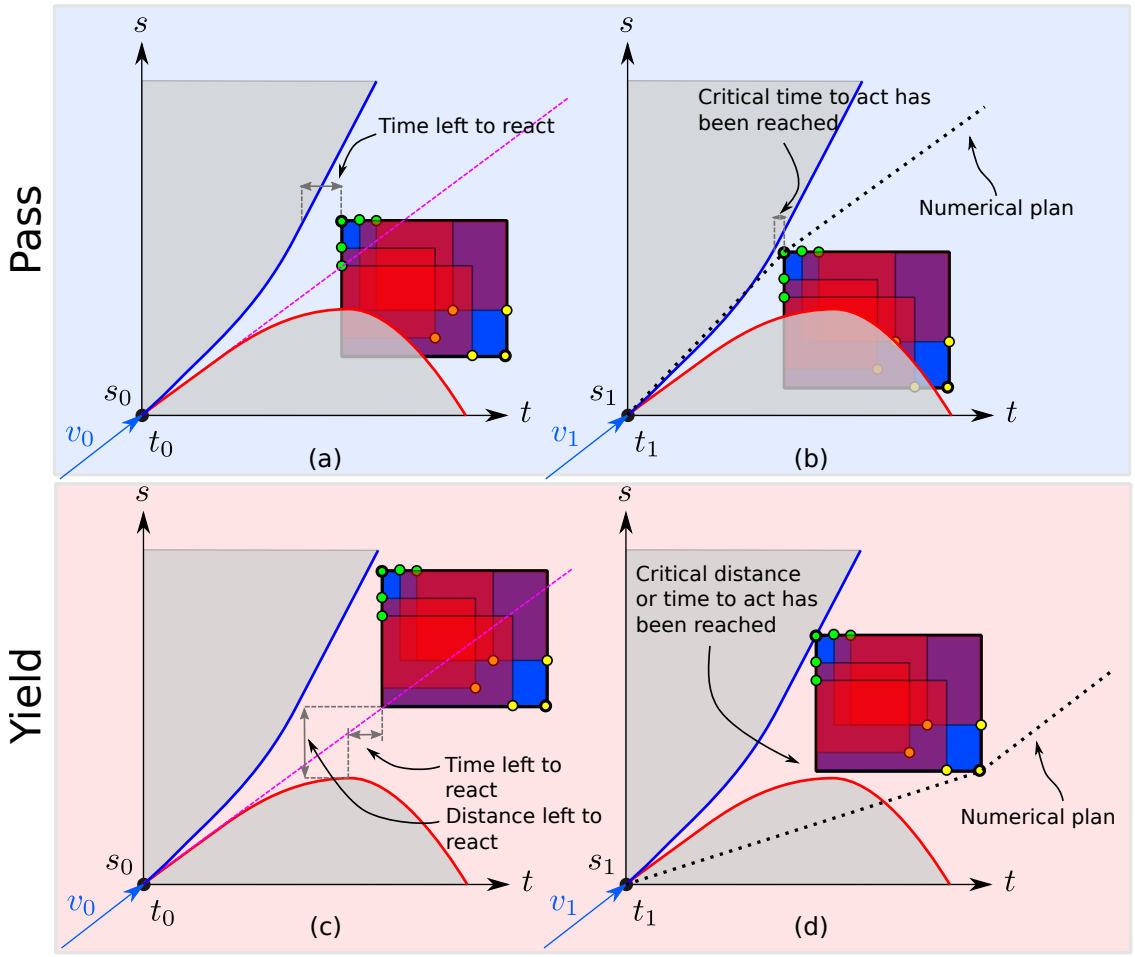

Figure 3.11: Triggering of value binding operation depends acceleration/deceleration curve. (a) and (c) illustrate the scenario where pass and yield strategies have been decided but reference speed profile is followed. (b) and (d) illustrate the scenario a little later at t_1 and just after value binding has occurred. The computed numerical plan is shown as a dotted black line. The blue rectangle shows the worst-case $s-t$ obstacle corresponding to the red $s-t$ obstacle samples used in value binding process

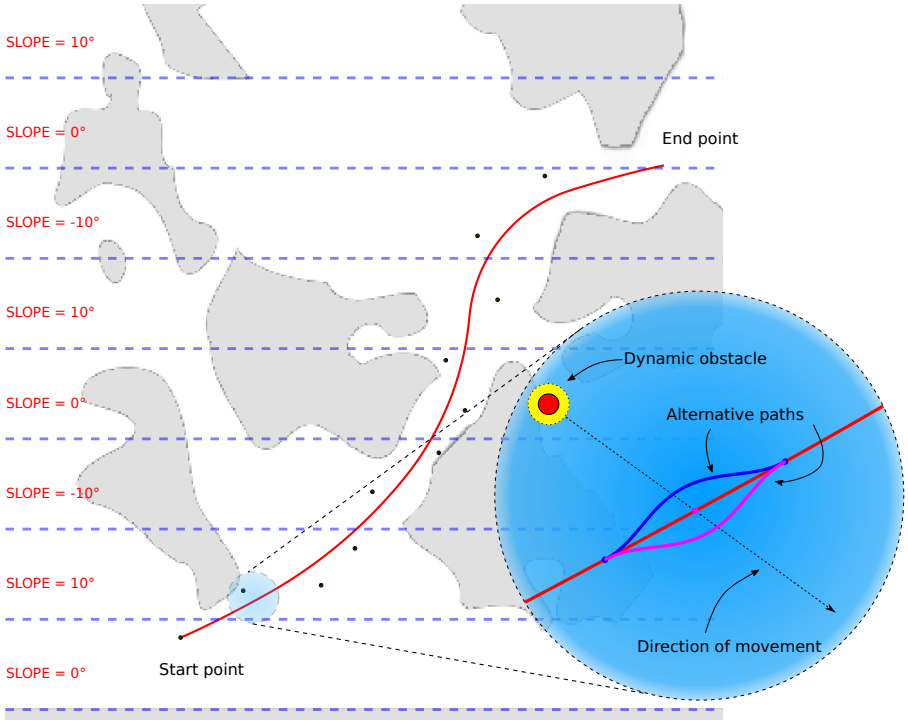

Figure 3.12: Top view of the scenario used for generating the results. Dynamic obstacles are poised to cut across the reference path from either the left or the right of the reference path. Inset: a magnified view of the dynamic obstacle (red with yellow buffer region), reference path (red) and alternative paths (blue, pink).

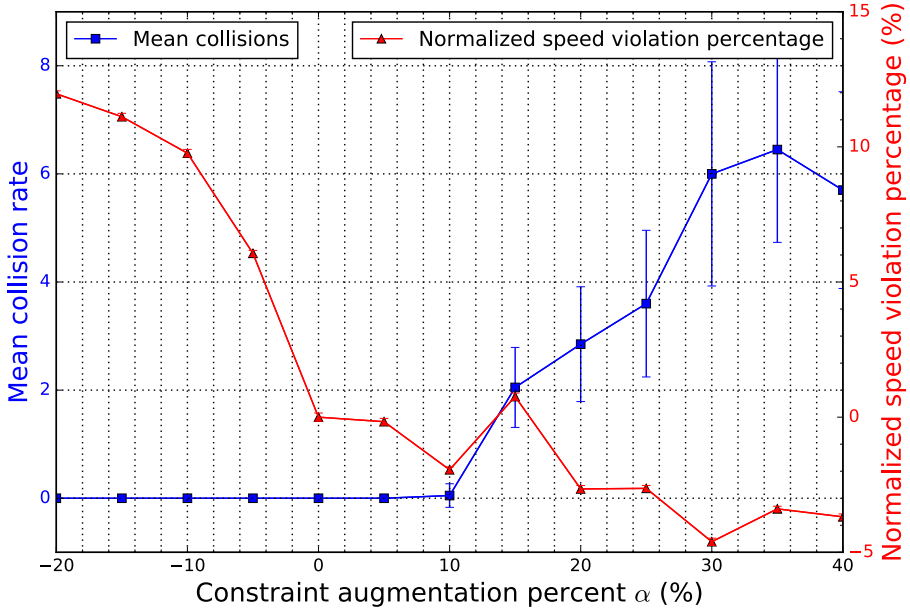

Figure 3.13: Relationship between collision rates and speed violation as the constraint augmentation factor is varied.

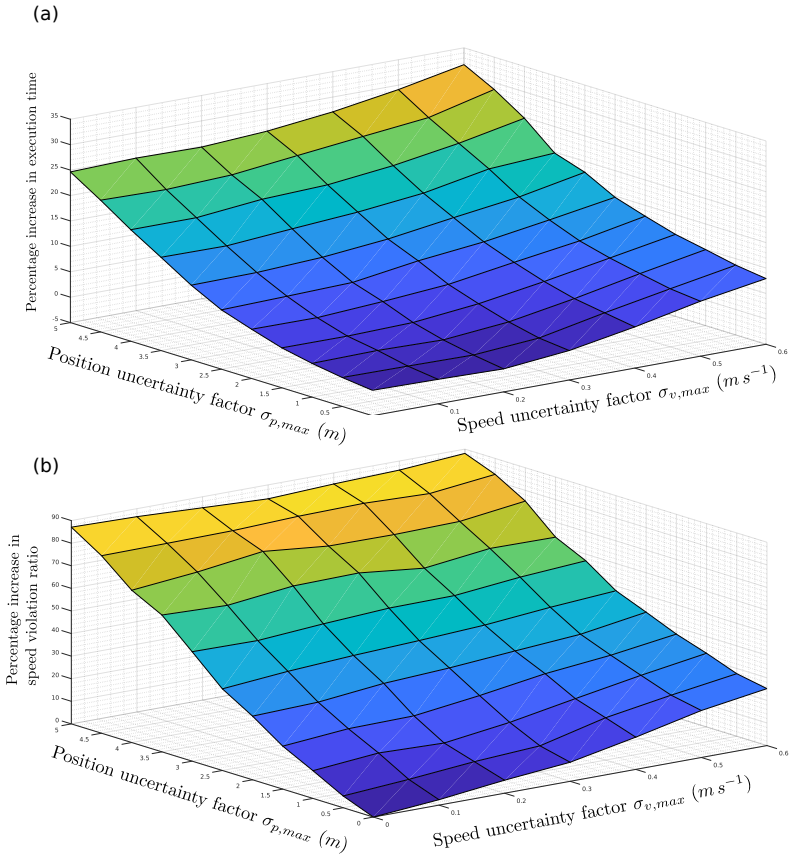

Figure 3.14: (a) Percent increase in execution time and (b) percent increase in speed violation ratio as uncertainty parameters $\sigma_{p,max}$ and $\sigma_{v,max}$ are varied.

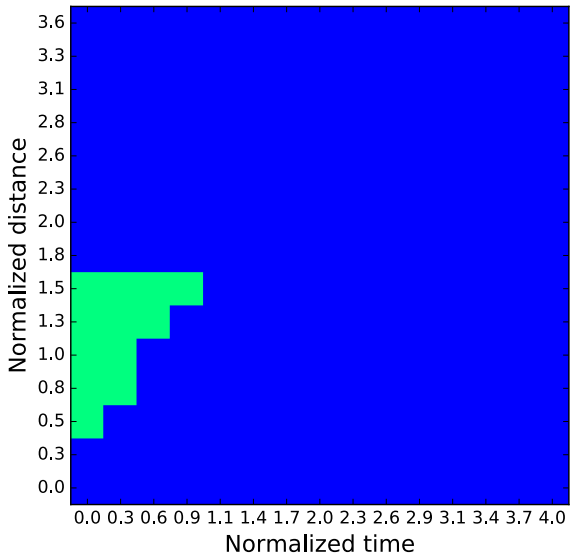

Figure 3.15: When handling two obstacles sequentially, collisions (green) happen when $s - t$ obstacles are spaced less than 1.5 stopping distances and 1 stopping time

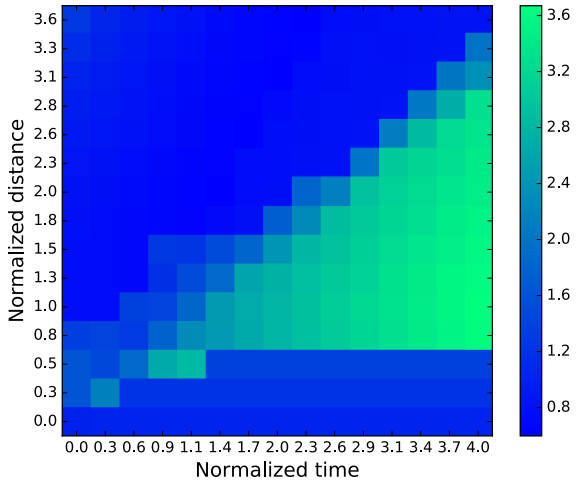

Figure 3.16: When handling two obstacles as a single composite obstacle, speed violation ratio increases when $s - t$ obstacles are spaced apart further than 0.5 stopping distances

Chapter 4

Deliberative Trajectory Planning

4.1 Introduction

Automated trajectory planning is a prerequisite for realizing autonomous unmanned vehicles. Collision avoidance with static and dynamic obstacles under perception uncertainty is a basic requirement for generating useful trajectories. The focus of this work is a framework for deliberative trajectory planning in the presence of large semi-permeable dynamic obstacles and motion uncertainty.

Typically, trajectory planning for unmanned vehicles is decoupled into two parts. First, a path planning problem is solved to obtain a tentative path, ignoring dynamic obstacles. Then, a motion planning problem is solved, this time considering dynamic obstacles, using the tentative path as a reference to yield a final trajectory. The motion planning problem is conventionally solved using reactive planning approaches like generalized velocity obstacles (GVO) [209] and velocity tuning [179, 156]. These methods work well when the dynamic obstacles are small and the introduction of these dynamic obstacles does not introduce drastic differences between the true optimal path and the tentative path. Thus, in the case of small dynamic obstacles, a reactively-planned trajectory is likely to be close to the true optimal trajectory while following a tentative path computed while ignoring small dynamic obstacles.

However, in the case of large dynamic obstacles, it is generally likely for reactively-planned trajectories to be highly sub-optimal. For instance, on the one hand reactively avoiding (i.e., going around) large dynamic obstacles may introduce unnecessary slow-downs and increase execution time. On the other hand, punching through semi-permeable dynamic obstacles may decrease execution time but increase the risk of failure and collisions. Therefore, handling large semi-permeable dynamic obstacles requires deliberative planning that can carefully balance the competing priorities (i.e., reduce execution time, reduce failure/collision rate). For example, reducing failure/collision rate should be the first priority. When it is possible to have zero failures/collisions, reducing execution time is the next priority. In some situations such as those that arise in the marine robotics domain, navigation around or through large dynamic obstacles is required (Figure 4.1).

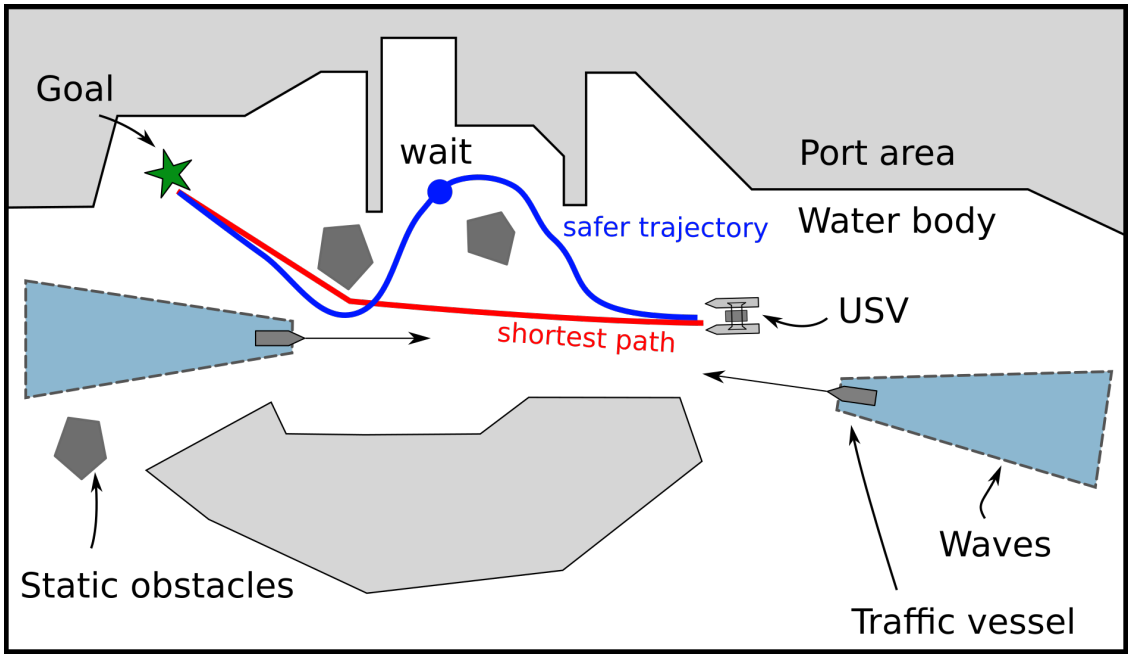

Figure 4.1: Handling large dynamic obstacles requires deliberative planning.

The wavefield generated by a marine vessel is a good example of a large semi-permeable dynamic obstacle. In a majority of practical scenarios, unmanned ground vehicles exhibit significantly less motion uncertainty compared to unmanned surface vehicles. Trajectory planning for

unmanned ground vehicles is a well-studied problem and in the recent years, significant progress has been made in solving it using a wide variety of methods such as graph search [118], stochastic tree search [57], MDPs (Markov Decision Processes), optimal control [88, 210]. Graph search methods like state lattice search have become popular as global planning methods due their optimality guarantees and efficiency in large environments.

Unmanned surface vehicles exhibit more motion uncertainty due to significant external disturbances like waves, wakes and winds [184]. Collision avoidance is more challenging under these circumstances and it involves not only staying a safe distance from static obstacles and dynamic obstacles but also following COLREGs [186]. COLREGs is a set of maritime navigation rules set by the International Maritime Organization (IMO) to minimize collisions and confusion among marine vessels. The effects of motion uncertainty is markedly more pronounced in smaller USVs as they are particularly more susceptible to wave disturbances. Motion uncertainty can be rigorously handled in a MDP (Markov Decision Process) framework for environments with static obstacles. It requires a state transition model of the USV. A state transition model to account for wave disturbances is developed in [194] for use in path planning amid static obstacles. In addition to ambient sea waves, wavefields generated by the motion of dynamic obstacles over the sea create yet another source of motion uncertainty and adds to the failure modes. The wavefield induces external disturbances that vary both spatially and temporally. Trajectory planning for USVs in the presence of dynamic obstacles needs to consider a state vector that includes position, orientation, velocity and time. For time-varying environments, the regular MDP can be converted to a time-extended MDP [120]. Time-extended MDP over high dimensional state spaces is complex and computationally prohibitive [83, 161]. On the other hand, naïve lattice-based search over motion primitives require careful collision risk assessment and specialized heuristics for dealing with dynamic obstacles in the presence of significant motion uncertainty. Large-scale POMDPs can be approximately solved only in a reactive online fashion using methods developed in [176, 212]. The main contribution of this work is a deliberative planning framework which 1) uses a trajectory

tracking error model to handle motion uncertainty, 2) introduces a failure-risk assessment scheme for moving wave-fields generated by traffic vessels, and 3) utilizes speed-up techniques for the search process to generate trajectories rapidly to be useful in dynamic environments. A state lattice search [118] approach is employed in this work. And, the waves generated by the moving vessels are modeled as trailing spatio-temporally changing penalty regions.

4.2 Background

4.2.1 Vehicle Model

For trajectory planning purposes, a simplified kinematic vehicle model is used to describe the motion of the USV.

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}, \mathbf{w}') \tag{4.1}$$

$$= \begin{bmatrix} \cos\psi & 0 \\ \sin\psi & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} + \mathbf{w}' \tag{4.2}$$

The USV pose is $\mathbf{x} = (x, y, \psi)^T$ where x and y denote the position in North-East-Down (NED) coordinates. ψ_k denotes heading with respect to north. USV control variables are $\mathbf{u} = (v, \omega)^T$ and $\mathbf{w}'$ is a zero-mean random process noise capturing the effects of environmental disturbances.

A state $\mathbf{s}$ is defined by the tuple (x, y, ψ, t, p_s) encoding the USV pose, the time-stamp and probability of being operational upto t units of time from the start time. A motion goal is a tuple $MG = (x_f, y_f, \psi_f, t_f)$ encoding a pose (x_f, y_f, ψ_f) to be reached at time instant t_f. Given an initial state $\mathbf{x}_i$ and a final state $\mathbf{x}_f = (x_f, y_f, \psi_f)^T$ specified by the motion goal MG, and letting $\mathbf{w}' = 0$, a nominal trajectory $\check{\mathbf{x}}(t)$ satisfying $\check{\mathbf{x}}(t_i) = \mathbf{x}_i \, , \check{\mathbf{x}}(t_f) = \mathbf{x}_f$ can be computed using trajectory generation methods [42, 88]. It is assumed that the low-level controller of the USV will

guarantee that target pose $\mathbf{x}_f$ is achieved by the time deadline t_f. In this work, Dubins curves [48] is used as the optimal trajectory between motion goals using 70% of the maximum allowable USV speed (as the controller may need to exercise full throttle occasionally to compensate for time losses while rejecting disturbances). A motion goal set $MGS_{r,n_{dirs}}(\mathbf{s})$ is a collection of motion goals spaced out a distance of r units relative to a particular state $\mathbf{s}$ and directed at n_{dirs} different angles. Each of the motion goals is given a label l corresponding to its direction from $\mathbf{s}$. Thus, $MGS_{r,n_{dirs}}(s) = \{MG_l\} \cup MG_O$, where $l = k * 360/n_{dirs}\,, k \in [0, n_{dirs} - 1]$. The label O indicates a wait action of duration Δt_{wait} at the same pose. A station-keeping controller is assumed to be active for the duration of the wait action. Figure 4.2 shows an example of a labeled motion goal set $MGS_{r=30m,n_{dirs}=8}(\mathbf{s})$ where $\mathbf{s} = (x = 0, y = 0, \psi = 0, t = 0)$. r is adapted to have multiple resolutions depending on how congested the local region is. This allows for high resolution motion goals in areas which are difficult.

4.2.2 Traffic Vessel Profiles

A traffic vessel profile TVP is a collection of quantities describing the geometry and state of a traffic vessel underway as predicted by a trajectory prediction system [14]. A typical traffic vessel profile TVP_i made up of the following quantities:

- Predicted trajectory $\bar{p}_i(t)$
- Instantaneous position covariance matrix $\Sigma_i(t)$
- Vessel zones $\{Z_k\}_{k=1}^3$ where each Z_k is an ellipse attached to and defined in the vessel body frame (see Figure 4.3).

The vessel zones travel along with the traffic vessels as vessels move. Vessel zone 1 is a region with 100% probability of failure . Vessel zone 2 and 3 mark regions where significant wave disturbances are expected and hence, also specify the local wavefront direction in that region. It is assumed that waves that are outside the zones have negligible effect on the USV. The sizes of these vessel

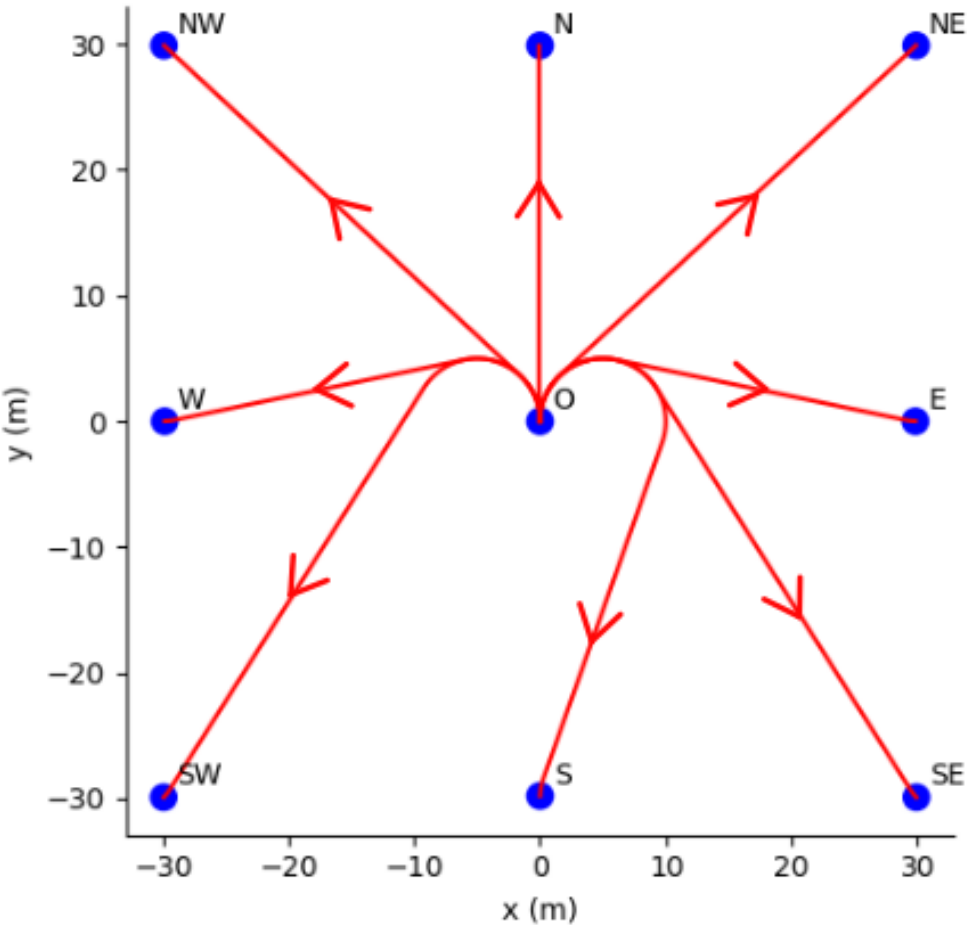

Figure 4.2: Motion goal set with $r = 30m$ containing $n_{dir} = 8$ motion goals (in blue) directed toward cardinal and intercardinal directions. Eight optimal trajectories (in red) from $(x = 0, y = 0, \psi = 0, t = 0)$ to each of the motion goals is shown. A ninth motion goal labeled O is also shown which corresponds to a wait-action of duration Δt_{wait}. The trajectories start with $\psi = 0$ and end at the heading angle corresponding to the cardinal directions.

zones are application-dependant. In this work, the length of zone 1 is fixed to three boat-lengths long, and it can be made longer for vessels traveling at a higher relative speed compared the USV. The length of zones 2 and 3 have been chosen based on the attenuation of the wavefield and where the wavefield poses no danger to the USV. A set of traffic vessel profiles is denoted by $TVPS = \{TVP_k\}_{k=1}^{n_{tv}}$.

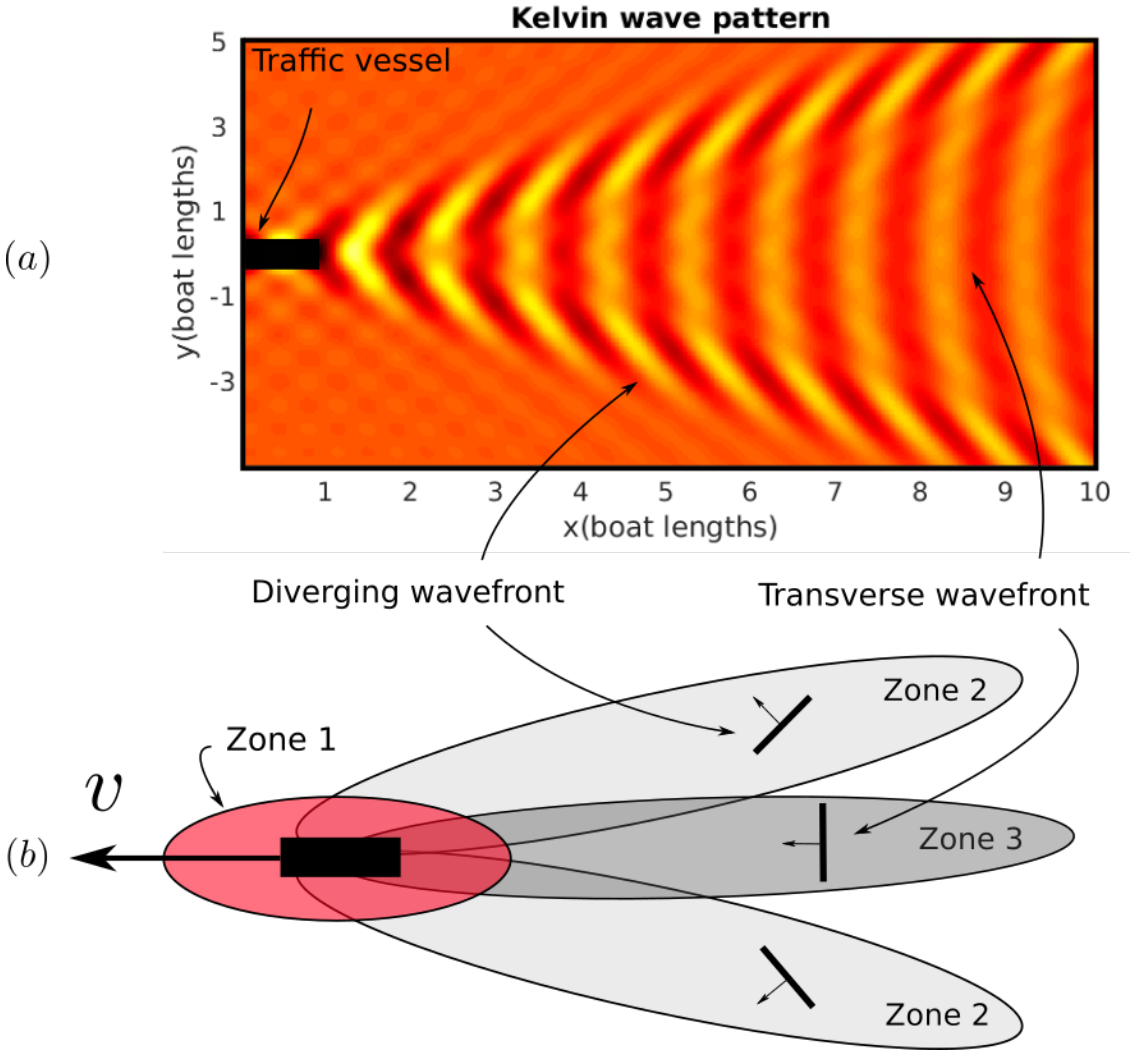

Figure 4.3: (a) A Kelvin wave pattern for a traffic vessel underway westward, operating at hull Froude number 0.5 and waterline boat length $10\,m$. Crests are shown in yellow and troughs are shown in red. Two local wavefronts near two instances of the USV is labeled 1 and 2. (b) Simplification of wave pattern into zones with different local wavefronts.

4.2.3 Environmental Disturbances

Environmental disturbances are encoded parametrically through a parameter vector defined for each state in the state-space as in Figure 4.4. For example, at point D, $\gamma_D = \{\text{sea state} = \text{calm}, \text{near vessel} = \text{false}\}$ where sea states are defined as in the Douglas sea scale [52]. Point C lies in a static obstacle region and hence, $\gamma_C = \{\text{invalid} = \text{true}\}$ is left undefined and handled as a special case. Whenever a traffic vessel passes through a point, for example, point A at a certain time t, the parameter vector is updated as $\gamma_A = \{\text{sea state} = \text{calm}, \text{near vessel} = \text{true}, \text{wave zone} = Z_2\}$. Similarly, at point B, $\gamma_B = \{\text{sea state} = \text{rough}, \text{near vessel} = \text{true}, \text{wave zone} = Z_3\}$. In both cases A and B, the local wave amplitude and wave direction can be determined given the wave zone and the pose of the traffic vessel at t. The field *near vessel* in γ indicates that the point is in some *zone* of some traffic vessel.

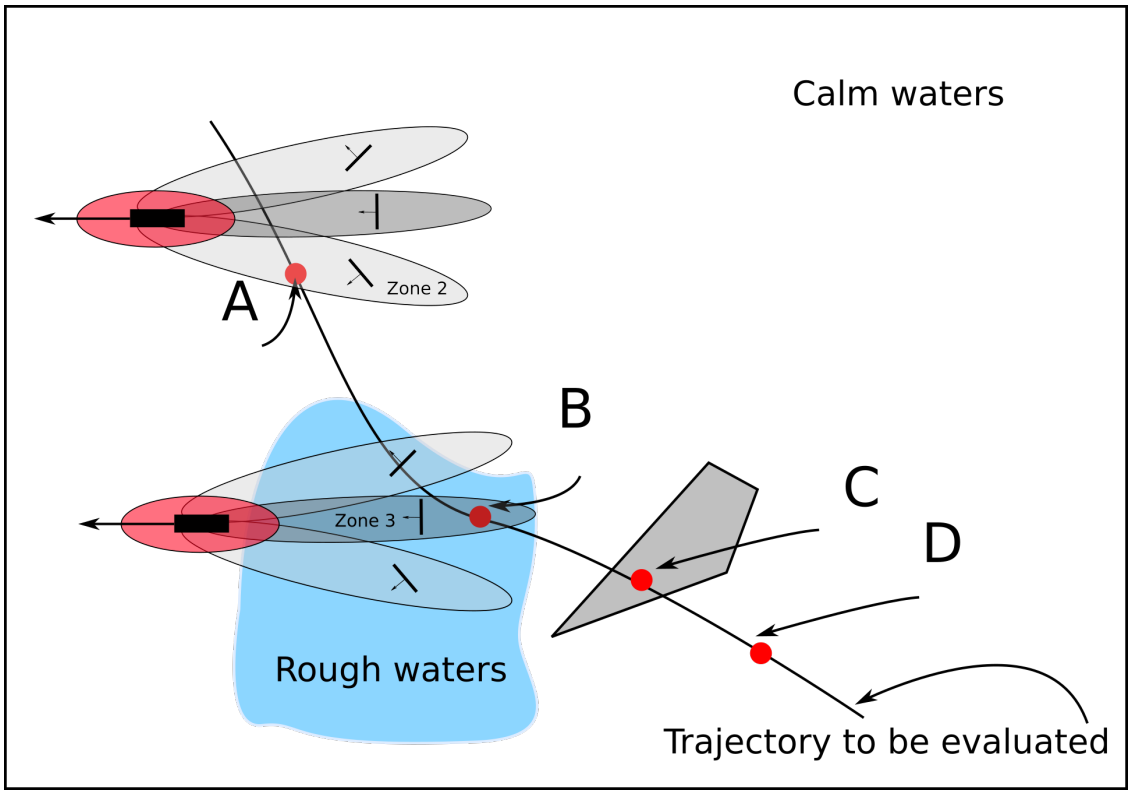

Figure 4.4: An illustration showing how parameter vector γ is defined over a trajectory at a few characteristic query points A, B, C and D.

4.2.4 Failure Model

We seek to model failure probabilities using a robot reliability model [180]. The probability that a robot remains functional up to time t is given by

$$p_s(t) = e^{-t/M}, \tag{4.3}$$

where M is the Mean-Time-Between-Failures (MTBF), which is estimated as

$$M = \frac{\text{Total observation duration}}{\text{Number of failures during observation}}, \tag{4.4}$$

the ratio of number of failures to the total number of attempts within an observation time window. Developing a precise model of the failure (i.e., capsize) of the USV in varying sea-states and waves is difficult. It involves allowing the robot to fail in the process of counting the number of failures within an observation time window. This is not practical.

A safer alternative is to measure how close to failure the robot got to when exposed to a certain local disturbance. In this work, a local disturbance is described parametrically as the tuple consisting of sea-state and proximity to a vessel. Two sea-states are considered *calm* (undisturbed waters) and *rough* (choppy waters due to wind conditions/multiple traffic vessels). And, proximity to vessel is a Boolean flag that is determined by whether the USV happens to be in any of the wavezones around the traffic vessel (see Figure 4.7). If the USV is in proximity to a vessel, then the wavezone is also part of the local disturbance parameter.

USVs typically have certain design limits such as roll limits or pitch limits. We define a failure event as an event when any of these design limits is exceeded. Thus, given a set of trajectory samples $\{\mathbf{s}_k(t)\}_{k=1}^{n_k}$ as in Section 4.4.1.1 of duration t_1. For each trajectory sample, we count the number of failure events. With failure events labeled and counted, compute M as in Equation 4.4 for each trajectory sample. We aggregate the MTBF over trajectory samples for

each combination of local disturbance parameter γ. In this way a look-up table that is indexed by γ can be generated (see Table 4.2). Once MTBF estimates are computed for an USV operating in different local disturbance parameters, it can later be used in the exponential reliability model to compute failure probabilities using Equation 4.3.

4.3 Problem Formulation

Given:

(a) Initial state $\mathbf{s}_i = (x_i, y_i, \psi_i, t_i, p_s = 1)$

(b) A region G around the goal state $\mathbf{s}_f = (x_g, y_g, \psi_g)$

(c) A set of traffic vessel profiles $TVPS = \{TVP_i\}_{k=1}^{n_v}$ sensed by the perception system.

(d) A map of static obstacles defined by a set of polygons SO

Compute: A sequence of states $\{\mathbf{s}_i\}_{i=0}^{p}$ such that

(a) $\mathbf{s}_0 = \mathbf{s}_i$

(b) $\mathbf{s}_p \in G$

(c) the trajectory is COLREGs compliant

(d) probability of success p_s at $\mathbf{s}_p$ is maximized

(e) expected time of execution is minimized.

4.4 Approach

The problem is posed as an implicit graph search problem and utilize the Theta* graph search algorithm [40] to solve it. The essential components of the Theta* graph search implementation are outlined in the following subsections.

4.4.1 Node Expansion and Cost Functions

A node n is a tuple defined as $n = \{\mathbf{s}, g, h, f\}$ where g is the cost-to-come, h is the cost-to-go, f is the total cost ($f = g + h$), $\mathbf{s}$ is the state. An initial node n_i is constructed as $n_i = \{s_i, 0, h(s_i, s_f), 0 + h(s_i, s_f)\}$. When a parent node n_p is expanded, it is assigned a set of child nodes $\{n_{c,k}\}_{k=1}^{N_c}$ that contain with states corresponding to the nearby motion goals. Here N_c is the total number of child nodes expanded for n_p. Each of the child nodes also contains the reference trajectory $\check{\mathbf{x}}(\mathbf{s}_p, \mathbf{s}_{c,k})$ specified by the Dubins curve linking the parent state to the child state. During execution, there is motion uncertainty as the controller tracks the reference trajectory. This uncertainty is handled while evaluating the cost function.

To illustrate the cost function, let us focus on the expansion of an arbitrary parent node n_p yielding a child node n_c ending in the state $\mathbf{s}_c$ as in Figure 4.9. Let us assume the reference trajectory $\check{\mathbf{x}}(\mathbf{s}_p, \mathbf{s}_c)$ starts at $t = t_p$ and ends at $t = t_c$.

In order to evaluate the cost function, the reference trajectory must be discretized into temporal segments. Performing regular discretization, we have a sequence of time samples $TS = \{t_q\}_{q=1}^{n_{ts}}$ where $t_q \in [t_p, t_c]$, $t_{q+1} - t_q = \Delta t$. Suppose for a $\check{t} \in TS$, the reference trajectory specifies the USV state to be $\check{\mathbf{s}}$ with the 2D position being $\check{\mathbf{p}}$ and time being $\check{t}$. And, similarly, all of traffic vessels are moved to $\check{t}$ on their mean predicted trajectory. The uncertainty in the predicted trajectory will be considered Section 4.4.1.6. Having moved the traffic vessels to their positions in the environment map, the local disturbance vector $\check{\gamma}$ is obtained by looking up $\check{\mathbf{p}}$ on the environment map and identifying which of the characteristic scenarios apply (see Figure 4.4). Let us now see how $\check{\gamma}$ is used in the trajectory evaluation process.

4.4.1.1 Identification of Tracking Error Model

In order to characterize the motion of the USV, we first generate an optimal control solution from a initial state $\mathbf{s}_0 = (x = 0, y = 0, \psi = 0, t = 0)$ to $\mathbf{s}_1 = (x = x_1, y = 0, \psi = 0, t = t_1)$ in the absence of disturbances and refer to it as the reference trajectory $\check{\mathbf{x}}(t)$. Next, we want to characterize how

much deviation from $\check{\mathbf{x}}(t)$ is expected in the presence of disturbances. Let the initial position be $\mathbf{p}_0$. When the low-level controller is commanded a motion goal to s_1, it tries to track the reference in the presence of disturbances. Thus, the target position to be achieved at $t = t_1$ is $\mathbf{p}_1 = (x_1, 0, 0)$. It is assumed that the motion goal is conservative enough to be feasible even under heavy disturbances (i.e. s_1 can be reached by t_1). The controller is able to achieve the target position while absorbing the effect of disturbances (by temporarily maneuvering at faster or slower speeds than the reference speed). Many trials are conducted where the USV is repeatedly made to move from $\mathbf{s}_0$ to $\mathbf{s}_1$ under different conditions γ (i.e. sea states and wave directions) and USV trajectory $\mathbf{s}(t)$ is recorded. The USV is subject to superposition of waves of varying frequency and amplitude expected in a particular sea state and varying local waves generated by traffic vessels. The k^{th} trial captures the evolution of the USV state under a particular realization of γ. Given a set of trajectory samples $\{\mathbf{s}_k(t)\}_{k=1}^{n_k}$, the deviation from the optimal reference trajectory can be characterized using two quantities: maximal *cross-track* error standard deviation (σ_{ct}), maximal *along-track* (σ_{at}) error standard deviation.

$$\sigma_{ct} = \max_{q \in [0,t_1]} \sigma[\{N_q[\tau_k(q) - \check{\tau}(q)]\}_{k=1}^{n_k}]$$

$$\sigma_{at} = \max_{q \in [0,t_1]} \sigma[\{T_q[\tau_k(q) - \check{\tau}(q)]\}_{k=1}^{n_k}]$$

Here τ_k and $\check{\tau}$ denote the $x - y$ path associated with $\mathbf{s}_k$ and $\check{\mathbf{x}}$ respectively. $\sigma[\cdot]$ denotes the standard deviation operator. $N_q[\cdot]$ and $T_q[\cdot]$ denote the normal projection, tangential projection operator on the reference path point $\check{\tau}(q)$.

The maximal cross-track error standard deviation is the maximum standard deviation of the normal distance between the USV and the path over the entire time interval $t \in [0, t_1]$. The maximal along-track error standard deviation is the maximum standard deviation of the tangential

distance between the USV and the reference position over the entire time interval $t \in [0, t_1]$. σ_{ct} and σ_{at} collectively capture the lateral position and speed uncertainty experienced by the USV as it travels toward a motion goal under influence of disturbance parameter γ.

Given a set of trajectory samples $\{\mathbf{s}_k(t)\}_{k=1}^{n_k}$ of duration t_1. For each trajectory sample, count the number of failure events. A failure event is when a roll limit or pitch limit has been exceeded or violations of other design limits. With failure events labeled and counted, compute M as in Equation 4.4 for each trajectory sample. Aggregate the MTBF over trajectory samples for each combination of local disturbance parameter γ. This way a look-up table that is indexed by γ can be generated.

4.4.1.2 Obtaining Tracking Error and Failure Model Parameters

Local environmental conditions such as local sea state and being in the vicinity of a moving vessel affect the trajectory tracking performance as the USV moves along a reference trajectory. These conditions are encoded into $\check{\gamma}$ (Figure 4.4) and can be used to obtain tracking error and failure model parameters.

If the USV were to be controlled in an open-loop configuration, then trajectory tracking error would build up over time. However, in the presence of a low-level stabilizing feedback controller, the tracking error can be bounded within an envelope around the reference trajectory [29]. The effects of the environment and controller can be extensively studied either using offline Monte Carlo simulations to obtain the various realizations of the transition trajectories [194] or empirically evaluated in real scenarios. An example is shown in Figure 4.5. In either case, the goal is to characterize the motion of the USV as it is commanded to move in a straight line from an initial state $\mathbf{s}_0$ towards a motion goal ahead in the presence of parameterized disturbances in the environment. We seek to model the tracking error and the probability of successful navigation along the reference trajectory as a function of the local disturbance parameter vector $\check{\gamma}$. Tracking

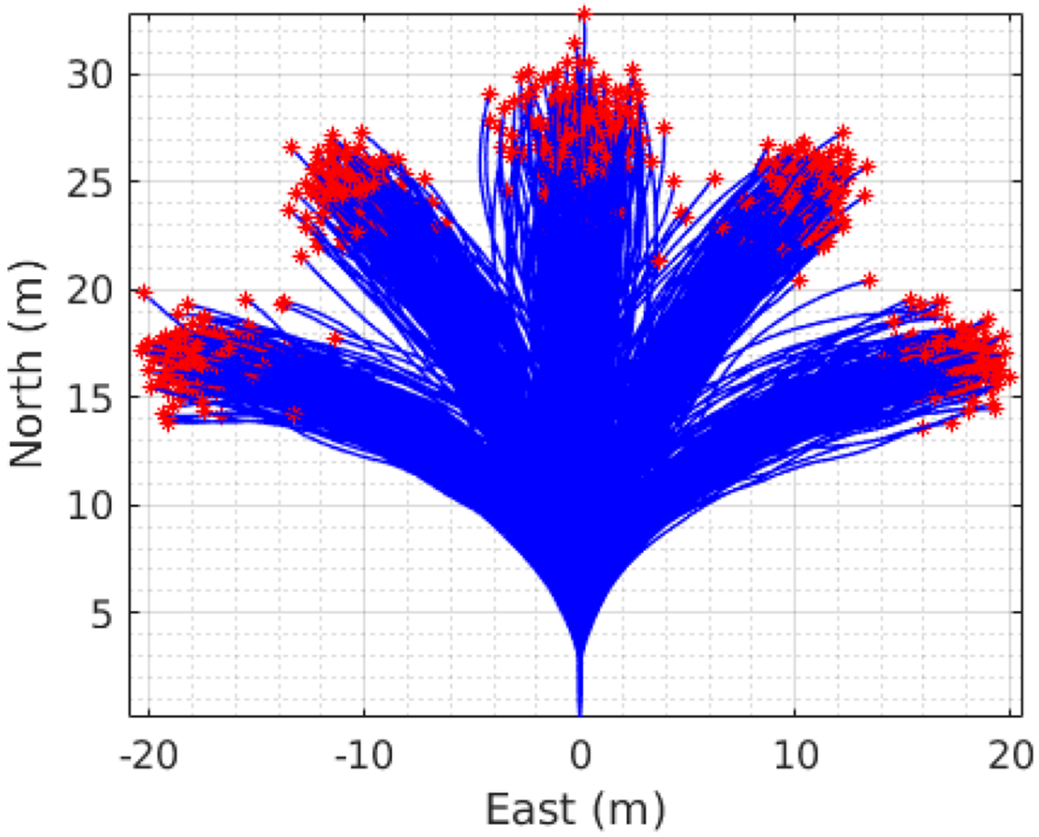

Figure 4.5: Realization of multiple simulated trajectories as the USV is commanded to go towards five different locations. Deviation from the nominal trajectories can be observed by looking at the spread in the realized trajectories.

error can be characterized by observing cross-track, along-track error (see Figure 4.8). Cross-track error at any point in time is the deviation in a direction normal to the reference trajectory (i.e. lateral deviation). Along-track error at any point in time is the difference in track-projected current position and the target point on the trajectory (i.e. lag or overshoot). Probability of successful navigation can be characterized by measuring MTBF under different conditions.

To this end, preliminary experiments were conducted with a 16 foot WAM-V USV [163] in Stranahan river, south-eastern Florida. The WAM-V platform was chosen due to its robustness under wave action. This way physical experiments could be safely performed while not risking the total loss of the robot. However, this work is applicable to platforms that are not specifically designed to withstand waves that may cause hull damage. Data was collected using wave gauges and the navigation sensors (IMU, GPS, Compass) of the USV (Figure 4.6). The data was studied to understand the effect of waves generated by a traffic vessel on the WAM-V.

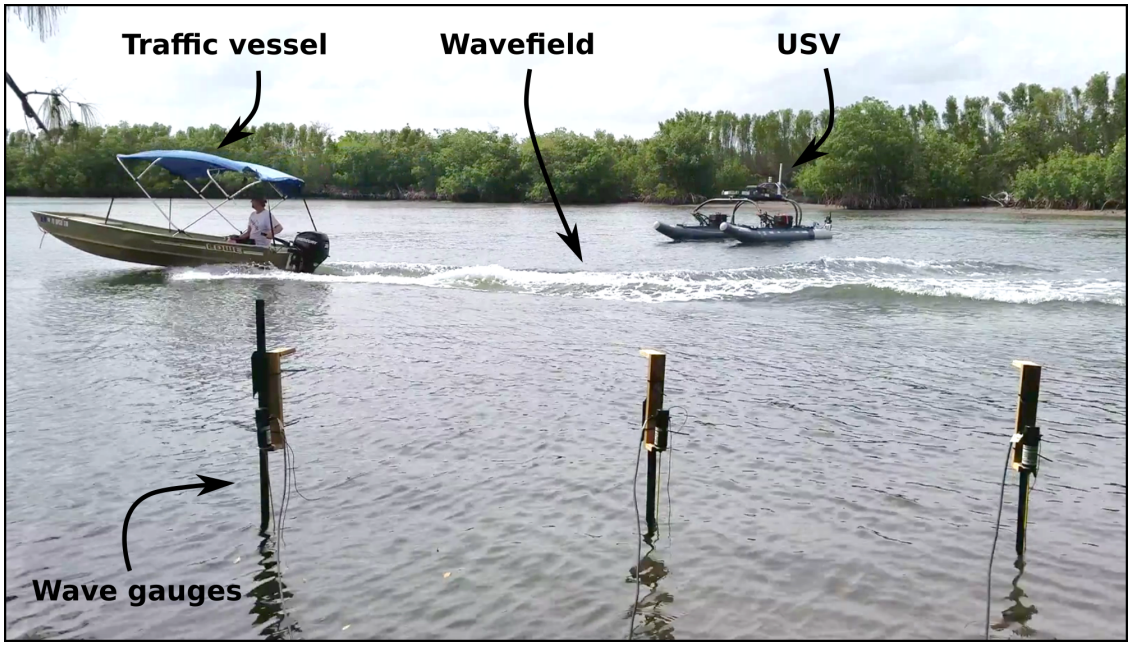

Figure 4.6: Wavefield experiment setup.

By observing the collected data, we generated Table 4.2 for MTBF (M) and Table 4.1 for cross-track error standard deviation (σ_{ct}), along-track error standard deviation (σ_{at}). Due to

Table 4.1: Tracking error model.

Disturbance parameter γ	Trajectory tracking error std.	
	Along-track	Cross-track
{sea state = calm, near vessel = false}	σ_{at}	σ_{ct}
{sea state = rough, near vessel = false}	$2\sigma_{at}$	$2\sigma_{ct}$
{sea state = any, near vessel = true}	$2\sigma_{at}$	$2\sigma_{ct}$

Table 4.2: Mean-time-between-failure (MTBF) model.

Disturbance parameter γ	MTBF
{sea state = calm, near vessel = false}	M_c
{sea state = rough, near vessel = any}	M_r
{sea state = calm, near vessel = true, wave zone = Z'}	M_t
{invalid = true},	0

the difficulties of controlling sea-state, only two states *calm* and *rough* were considered. Calm condition refers to a situation when water level across the test area is roughly constant, and no wind effects are seen on the surface of the water. Rough condition refers to a situation when water levels are changing due to interacting waves generated by passing traffic vessels and wind. In both conditions, the USV was made to traverse a $20\,m$ linear path 10 times and the cross-track σ_{ct} and along-track σ_{at} error standard deviation in the tracked trajectory was used to populate Table 4.1. Similarly, in both sea-state conditions, the pitch and roll angles recorded and the procedure in Section 4.2.4 is used to compute M_r and M_c in Table 4.2. M_t is not generated using physical experiments instead we use a surrogate model defined by Equation 4.5. In a very rough sea-state, the data from IMU is likely to be noisy and other sensing modalities might be needed for producing better estimates. Since we performed experiments in a water channel, our experiments were shielded from disturbances and did not experience noisy IMU measurements and we were able to observe roughly sinusoidal variations as the waves passed by.

Tables 4.2 and 4.1 are indexed by the local disturbance parameters in γ and they are briefly explained below.

4.4.1.3 Obtaining Tracking Error Parameters

As for the tracking error model, when the query state lies over calm waters and is not in vicinity of any traffic vessel, we use nominal values for tracking error parameters. Otherwise, tracking error parameters are doubled.

4.4.1.4 Obtaining MTBF

Since traffic-free calm waters are very safe, $M_c = \infty$. Since rough waters are not safe regardless of traffic, $M_r = 600$. Assigning MTBF over calm waters near traffic vessels involves determining angles between waves and USV heading. If the query state is located within any zone of the traffic vessels, we determine the relative position $\mathbf{p}_d$ from the center of the traffic vessel and the incidence angle θ_i between the USV heading and the wave direction of zone Z' (see Figure 4.7). $\mathbf{p}_d$ is calculated in the body coordinates of the traffic vessel and $d = \begin{bmatrix} 1 & 0 \end{bmatrix} \mathbf{p}_d$. Adhering to this notion, we use the following assignment for MTBF:

$$\alpha = \begin{cases} 1.0, & -\frac{\pi}{8} \leq \theta_i - \frac{\pi}{4} \leq \frac{\pi}{8} \\ 0.25, & \text{otherwise} \end{cases} \tag{4.5}$$

$$M_t = \begin{cases} 0, & p_d \in Z_1 \\ \alpha M_{Z'} d / L_z, & \exists k \in \{2, 3\},\ p_d \in Z_k \\ M_c, & \text{otherwise} \end{cases} \tag{4.6}$$

Here $M_{Z'}$ is MTBF value associated with the zone Z' and is defined as $M_{Z_1} = 0$, $M_{Z_2} = 1000$, $M_{Z_3} = 3000$. And, L_z is the maximum trailing length of the traffic vessel zones.

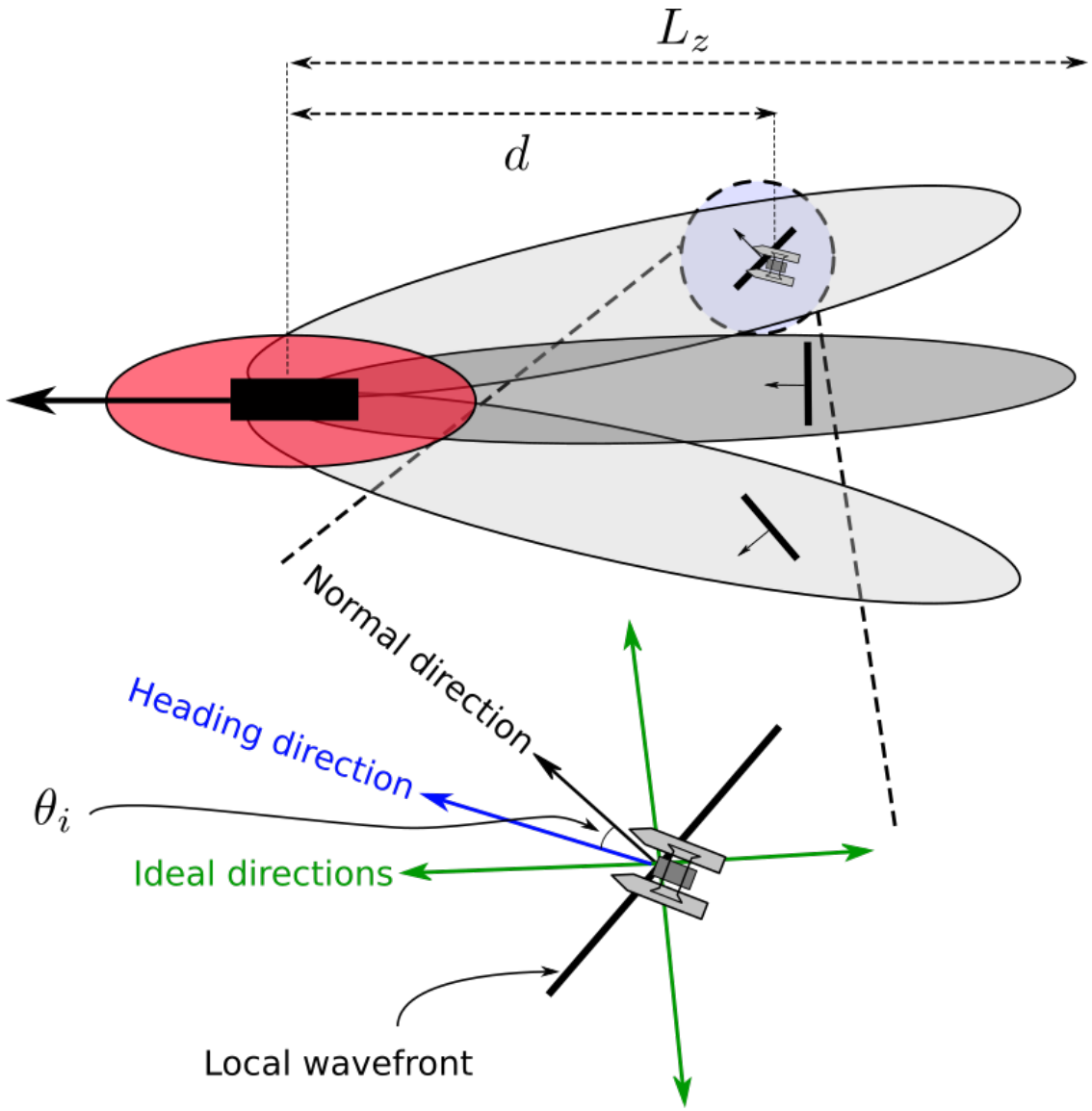

Figure 4.7: Determining wave incidence angle and relative position from a traffic vessel.

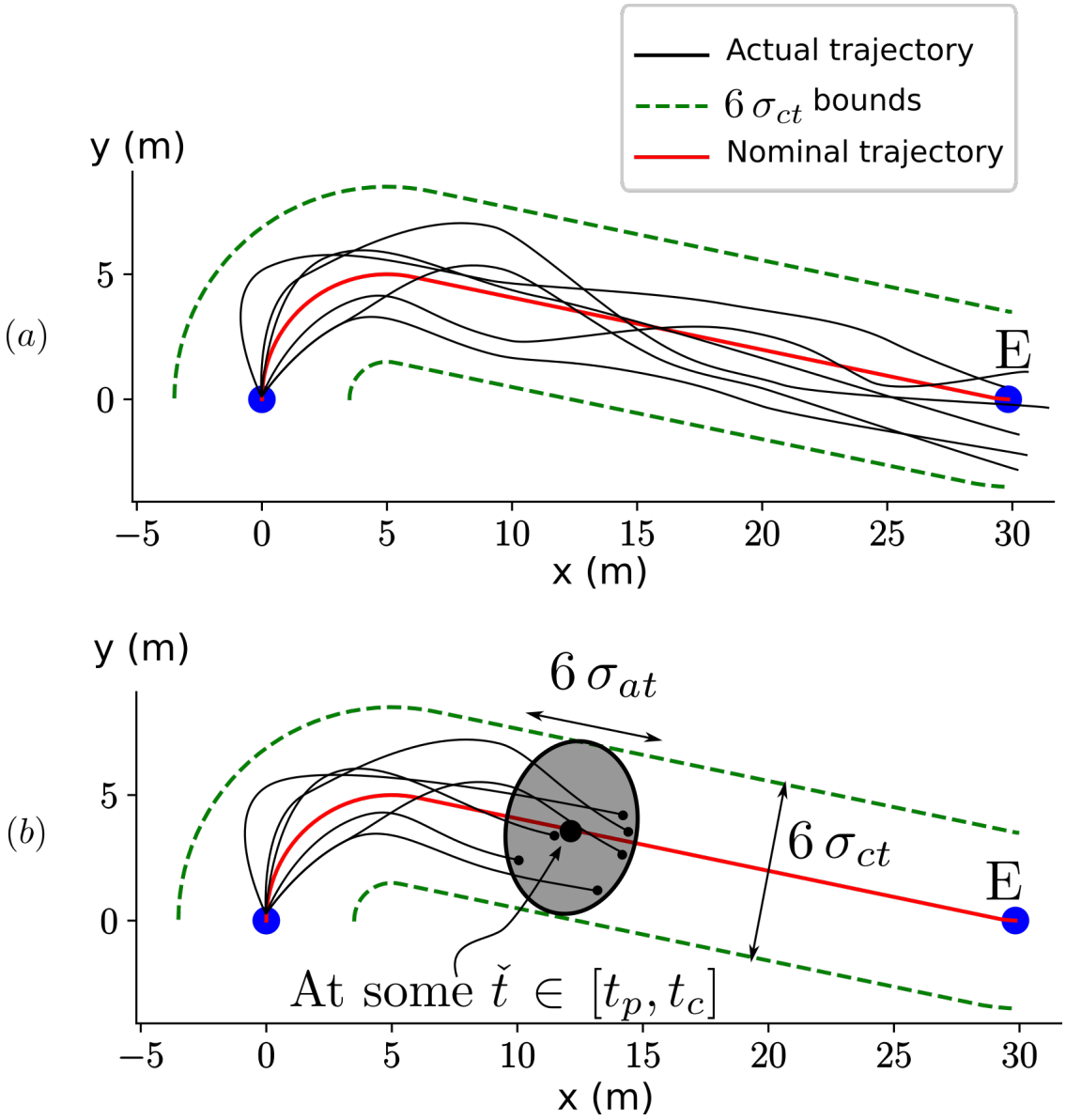

Figure 4.8: (a) Trajectory tracking error envelope defined by σ_{ct} and σ_{at} for a trajectory starting at $p_q = (0, 0, 0)$, $t_q = 0$ and reaching east under a particular sea state γ, (b) The 3σ error ellipse at an intermediate time $\bar{t}$ along the nominal trajectory.

4.4.1.5 Cost Assessment

Having determined tracking error and MTBF parameters using $\check{\gamma}$, we proceed to evaluate the cost function along the reference trajectory $\check{\mathbf{x}}(\mathbf{s}_p, \mathbf{s}_c)$.

Tracking error parameters σ_{at} and σ_{ct} encode the deviation along the path and the deviation in the normal direction from the position $\check{\mathbf{p}}$, respectively (see Figure 4.8). Thus, the actual position $\mathbf{p}$ of the USV at $\check{t}$ is given by

$$\mathbf{p} = \check{\mathbf{p}} + c\,N[\check{\mathbf{p}}] + a\,T[\check{\mathbf{p}}] \tag{4.7}$$

where $c \sim \mathcal{N}(0, \sigma_{ct})$ and $a \sim \mathcal{N}(0, \sigma_{at})$. $N[\check{\mathbf{p}}]$ and $T[\check{\mathbf{p}}]$ are the normal and tangent unit-vectors at $\check{\mathbf{p}}$. An ellipse $E_{\check{t}}$ centered at $\check{\mathbf{p}}$ and aligned to the tangent direction $T[\check{\mathbf{p}}]$ and normal direction $N[\check{\mathbf{p}}]$ with axis lengths $6\,\sigma_{at}$ and $6\,\sigma_{ct}$ describes a 99.7% confidence region within which the USV can be expected to be at $t = \check{t}$ (99.7% comes from the 68-95-99.7 empirical rule associated with the normal distribution).

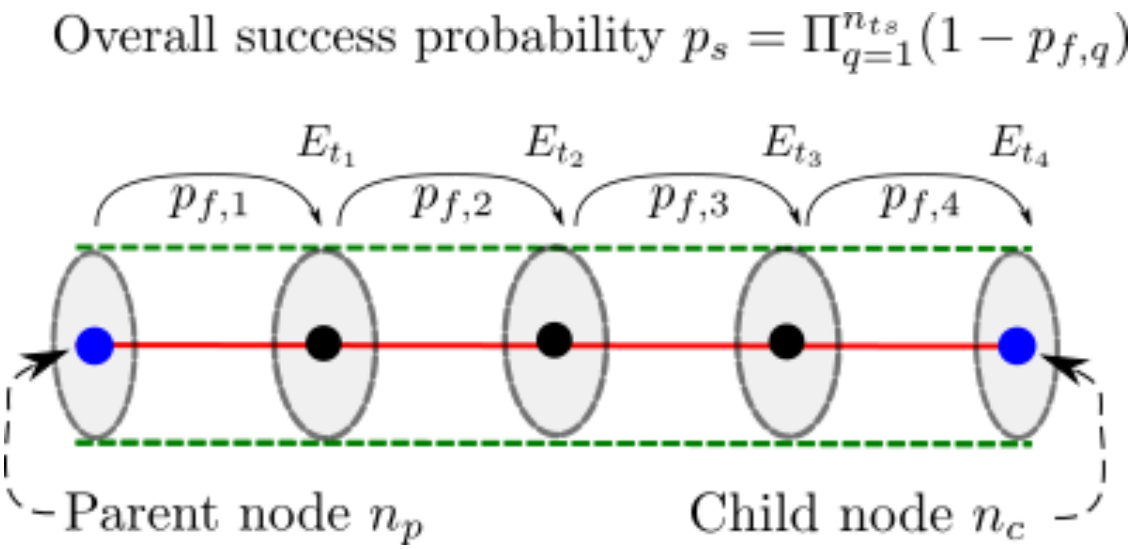

Figure 4.9: Collision cost assessment for a trajectory starting at a parent node and ending at a child node.

For each ellipse E_{t_q}, the probability of failure $p_{f,q}$ is computed as follows.

$$p_{f,q} = 1 - \exp(-\frac{\Delta t}{M_q}) \tag{4.8}$$

where M_q is computed by averaging the MTBF values over a uniform set of points in E_{t_q}. More specifically, these MTBF values are computed by looking up each point in Γ and then using Table 4.2 to find the MTBF. Assuming independence of the failure events, the overall probability of success for executing $\tilde{\mathbf{x}}(\mathbf{s}_p, \mathbf{s}_c)$ is given by

$$p_s = \Pi_{q=1}^{n_{ts}}(1 - p_{f,q}) \tag{4.9}$$

Thus, the probability of success assigned to the child state is $p_s(n_c) := p_s(n_p) * p_s$ where $p_s(n_p)$ and $p_s(n_c)$ are the probabilities of success of the parent and child states respectively. The child node n_c is then assigned a cost as follows:

$$g(n_c) = g(n_p) + (1 - \omega_f)c_t - \omega_f F_f \log(p_s) + c_c F_c \tag{4.10}$$

where time cost is $c_t = t_c - t_p$, F_f is a large constant with units of time penalizing risky motion causing failure, F_c is a large constant with units of time penalizing moves that violate COLREGs. $c_c \in \{0, 1\}$ encodes the violation of COLREGs computed using [185, 168]. $\omega_f \in [0, 1]$ is a user defined weight parameter.

4.4.1.6 Accounting for Perception Uncertainty

A Monte Carlo sampling scheme [104] is used to account for the perception uncertainty in the measurement of the traffic vessel profiles TVP. The Monte Carlo sampling approach is favored as it is a general method and amenable to parallelization. However, this approach requires extensive sampling when high confidence estimates of collision probabilities are required especially when

chance of a collision is rare. This problem can be alleviated if adaptive sampling techniques are used [164]. In the previous section, $\check{\gamma}$ was computed using only the mean position of traffic vessels at $\check{t}$. However, to account for uncertainty, the covariance information needs to be used.

Let us pick one traffic vessel profile $TVP_i \in TVPS$ to illustrate the process. A set of possible instantaneous positions $PS_i = \{p_k\}_{k=1}^{n_{ps}}$ is sampled from the distribution $\mathcal{N}(\bar{p}_i(\check{t}), \Sigma_i(\check{t}))$. The traffic vessel are moved to each possible position $p_k \in PS_i$, then γ_k is looked-up as in Figure 4.4. The corresponding M_k are computed using Table 4.2 and Figure 4.7. Finally, the average M is computed over all the computed M_k. This average M is used in place of the original MTBF in Equation 4.8.

4.4.2 Speed-up Techniques

Some techniques are described to improve the search performance.

4.4.2.1 Static Obstacle Heuristics (SOH)

Since the cost function includes the time duration of going from one position to another (i.e. c_t) and static obstacles remain fixed for at least a few planning cycles, it is beneficial to compute a time-to-goal map (also known as arrival time map). This map is termed SOH and it stores the lower bound on the time cost of going from any position in the workspace to the goal position g. This map is computed using Fast Marching Method (FMM) to approximately solve an Eikonal equation [122, 15]. The level sets of the solution of the Eikonal equation correspond to the frontiers of a wave emanating from the goal position. The speed of the wave is set to the maximum speed of the USV. As the frontier (i.e. the open set in Dijkstra's algorithm) bends around corners, the time-to-goal estimate is more informative than the naïve Euclidean distance based estimate. The raw time-to-goal values given FMM are naturally not admissible unless they scaled down by a factor that depends on the discretization used in computing the map [118]. After appropriate scaling, each cell in this map contains an underestimate of the time to reach the goal from that

cell location. This value is used to compute an estimate of c_t to be accumulated till the goal is reached. Given a query position s, a heuristic value $h_s(s, g)$ is looked up from the SOH.

4.4.2.2 Space-Time Exploration Heuristics

Developing admissible heuristics in the presence of dynamic and static obstacles is challenging and can often be just as hard as the original problem. However, by relaxing the original problem, effective heuristics can be generated. We follow [33] and drop to 3D configuration space by excluding ψ (i.e. we use only $x - y - t$ space) to generate a heuristic trajectory using a point robot kinematic model (relaxed version of the USV kinematic model) from initial state $\mathbf{s}_i$ to the goal region G. Maximum speed v_{max} of the point robot is the same as that of the USV. Zones 2 and 3 of traffic vessels are ignored such that only zone 1 is considered as a dynamic obstacle. In other words, for the purpose of heuristics generation, the point robot is allowed to move freely across zone 2 and 3 without any penalty.

Suppose the point robot is at the initial state $\mathbf{p}_i \equiv (x_i, y_i)$ at the query time t_i (see Figure 4.10). One can observe that set of all points this point robot can reach in Δt is given by a *cone* C_i with apex at (x_i, y_i, t_i), perpendicular height Δt and radius $v_{max}\Delta t$. Δt is chosen such that it is as large as possible without the cone touching any configuration space obstacle. Suppose $d_c(t)$ is the clearance distance between $\mathbf{p}_i$ and any workspace obstacle at time t. Then,

$$\Delta t_i = \min_{t \in [t_i, t_i + \Delta_{t,max}]} d_c(t)/v_{max}.$$

This cone represents a collision-free volume in the 3D configuration space. In order to explore the space, the idea is to grow new *cones* at the top face of cone C_i and continue this process until the goal region is reached by a certain cone. In this way we generate a sequence of overlapping cones forming a collision-free volume in configuration space which connects the initial state and the goal region.

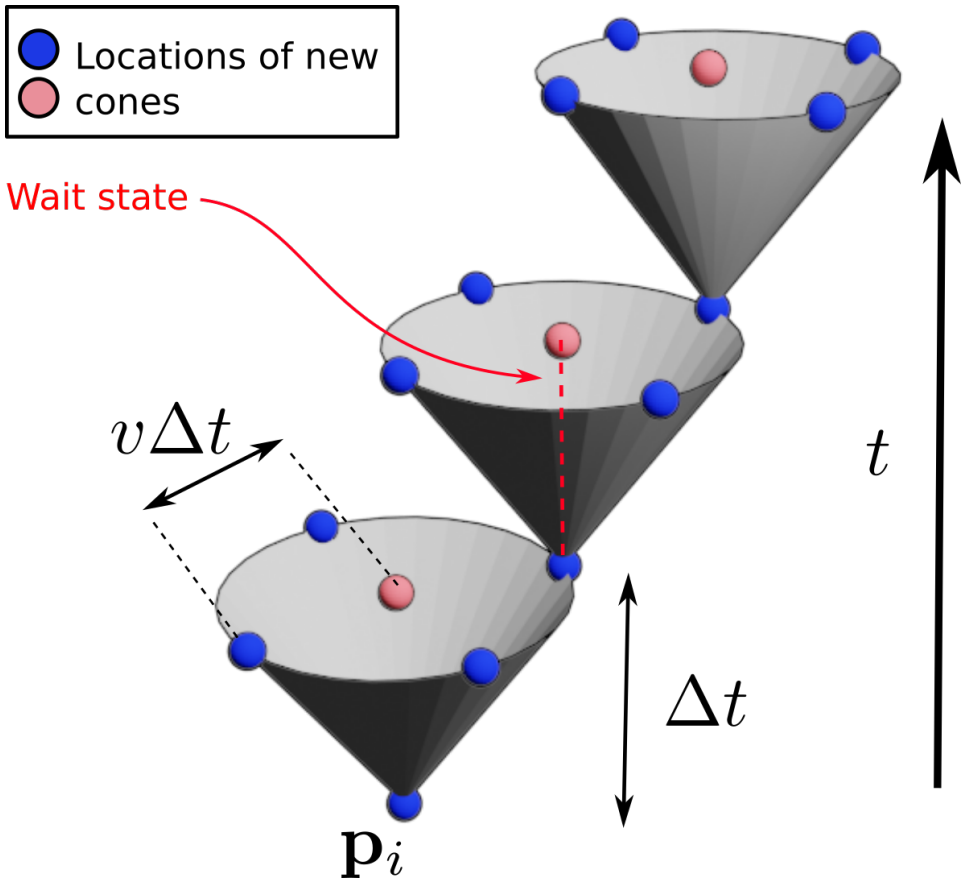

Figure 4.10: Generation of heuristic space-time path.

Algorithm 2 Heuristic Space-Time Path

```
1: procedure ComputeSTHeuristicPath(s_i, G)
2:     S_closed ← ∅
3:     S_open ← {n_start}
4:     while S_open ≠ ∅ do
5:         n_k ← PopTop(S_open)
6:         if ExistsOverlap(n_k, G) then ▷ Check if cone of n_k has non-empty intersection with G
7:             return RetracePath()
8:         else if n_k ∉ S_closed then
9:             D ← ExpandNode(n_k, r_min, r_max)                ▷ Store child nodes in D
10:            S_open ← D ∪ S_open
11:            S_closed ← n_k ∪ S_closed
12:        else
13:            continue
14:    return failure
```

Nodes are described using tuples of the form $n_k = (x_k, y_k, t_k, r_k, C_k, g_k, h_k, f_k)$. C_k is the cone situated at (x_k, y_k, t_k) with radius $r_k = v_{max}/\Delta t_k$. The cost-to-come $g_k = t_k + c_c F_c$ is the sum of time elapsed and COLREGs violation cost. And, the cost-to-go is the value from the SOH lookup table, $h_k = h_s(\mathbf{p}_k, g)$. The total cost is $f_k = g_k + h_k$. The space exploration process proceeds in an implicit A* graph search manner as in Algorithm 2 by first pushing the start node n_i in to the open set S_{open}. The open set S_{open} stores the nodes to be explored in a sorted order with respect to the cost f in ascending order and it is implemented using a priority queue. For as long as S_{open} is non-empty, the node with the lowest f value is chosen for further expansion. In this context, an expansion of a node involves the creation of new child nodes at the top face of C_k. The perimeter of the top face is equally sampled to form candidate locations for new child nodes. These locations indicate constant velocity motion. Furthermore, an additional candidate location is sampled at the center of the top face. This candidate location captures the notion of a wait state (Figure 4.10). For each candidate location (x_c, y_c, t_c), a new cone C_c is constructed with apex at the candidate location such that it has the largest height and does not collide with configuration space obstacles. If such a cone can be constructed, a new node $n_c = (x_c, y_c, t_c, r_c, C_c, g_c, h_c, f_c)$ is generated where $g_c = t_c + c_c F_c$, $h_c = h_s(\,(x_c, y_c), g)$, $f_c = g_c + h_c$. Incorporation of COLREGs constraints through the term $c_c F_c$ improves the effectiveness of the heuristics. Once Algorithm 2 ends, a sequence of cones $CS = \{C_q\}$ starting at the initial state and ending in the goal region is obtained. Let τ be the sequence of apexes of the cones in CS and it represents a coarse-grained trajectory in the (x, y, t) space that avoids static and dynamic obstacles.

The computation of heuristic value is first described. We first project τ to (x, y) space and denote it as τ_p. For a given a query state $\mathbf{s}$, we use the position component $\mathbf{p}$ of $\mathbf{s}$ to find the closest point $\mathbf{p}_c$ on τ_p. The heuristic is computed as in [33]. It is the sum of time difference between the end-point of τ and the closest point on τ and time required to reach $\mathbf{p}_c$ (i.e., $\frac{|\mathbf{p}-\mathbf{p}_c|}{v_{max}} + t_{end} - t_c$ where $\tau(t_c) = \mathbf{p}_c$ and t_{end} is the timestamp of the end-point of τ).

4.4.2.3 Adaptive Motion Goal Set (AMGS)

In order to accelerate the Theta* graph search, the motion goal set resolution parameters r and n_{dirs} are modified depending on the distance of the parent node position from moving vessels and static obstacles. A precomputed Euclidean distance transform is used to query the shortest distance from any of the static obstacles. Shortest distance from any of the traffic vessels is computed online.

$$(r, n_{dirs}) = \begin{cases} (3L_u, 16), \text{if within } 10L_u \text{of a moving vessel} \\ \text{ or within } 5L_u \text{ of a static obstacle} \\ (5L_u, 8), \text{if within } 25L_u \text{ of a moving vessel} \\ \text{ or within } 10L_u \text{ of static obstacle} \\ (8L_u, 8), \text{otherwise} \end{cases}$$

This way, in open waters, longer strides are made and the progress towards goal is faster.

4.4.3 Parameter Tuning

Table 4.3 summarizes the list of parameters used in the planner. These parameters are chosen by the user.

4.4.3.1 User Preference Parameters

User preference parameters balance safety and performance. The failure penalty F_f can be chosen as the duration of time required to dispatch a functional USV in lieu of the failed USV. This choice roughly captures the real time-impact a failed USV will cause. The COLREGs violation penalty F_c should be much smaller than F_f to favor COLREGs non-compliant trajectories over trajectories that lead to collision. ω_f controls the extent to which large time delays are accepted in order to

Table 4.3: Planner parameters.

Parameters		
Symbol	Description	Typical Range
r	Motion goal spacing	$5\,m$-$30\,m$
n_{dirs}	Number of directions	$\{8, 16\}$
Δt_{wait}	Wait state duration	$10\,s$
M	Failure probability model parameter	$500 - 6000\,s$
L_u	Length of USV	$5\,m$
L_v	Length of traffic vessel	$10\,m$
L_z	Maximum extent of the wavefield extending behind vessels	$100\,m$
σ_{ct}	Trajectory tracking cross-track σ	$\sim 1\,m$
σ_{at}	Trajectory tracking along-track σ	$\sim 1\,m$
F_f	Time penalty for failure	~ 300
F_c	Time penalty for COLREGs violation	~ 300
ω_f	Weight parameter favoring collision avoidance over travel time	0.5
n_s	Sample size for handling perception uncertainty	20

minimize failure. The closer to ω_f is to 1, the more conservative the trajectories are in terms of avoiding failure.

4.4.3.2 Planning Speed Parameters

Decreasing r and increasing n_{dirs} creates a denser motion goal lattice and thus, increases planning times. Using longer wait state durations Δt_{wait} may introduce undesirable looping behavior as the cost of gyrating about a point may be cheaper than waiting at the same point for a duration of Δt_{wait}. Introducing an extra cost term in Equation 4.10 capturing energy cost will remove this behavior (e.g. $c_p = \text{PathLength}(n_p, n_c)$).

4.5 Results and Discussion

The wave-aware trajectory planner was implemented in MATLAB. A set of simulation experiments was conducted. The setup and results of these experiments are described below.

4.5.1 Simulation Experiments

Four different maps were generated to test the trajectory planner. Traffic vessels are shown in their initial configuration in Figures 4.11, 4.13, 4.12 and 4.14. Magenta arrows indicate the intended direction of motion. The USV and traffic vessels were simulated using three degrees-of-freedom dynamic model as in [93]. In the USV pose observations used for planning, noise was added following the distribution in Equation 4.7. The response of traffic vessels to the motion of the USV not simulated. Only the USV is allowed to react to the traffic vessels.

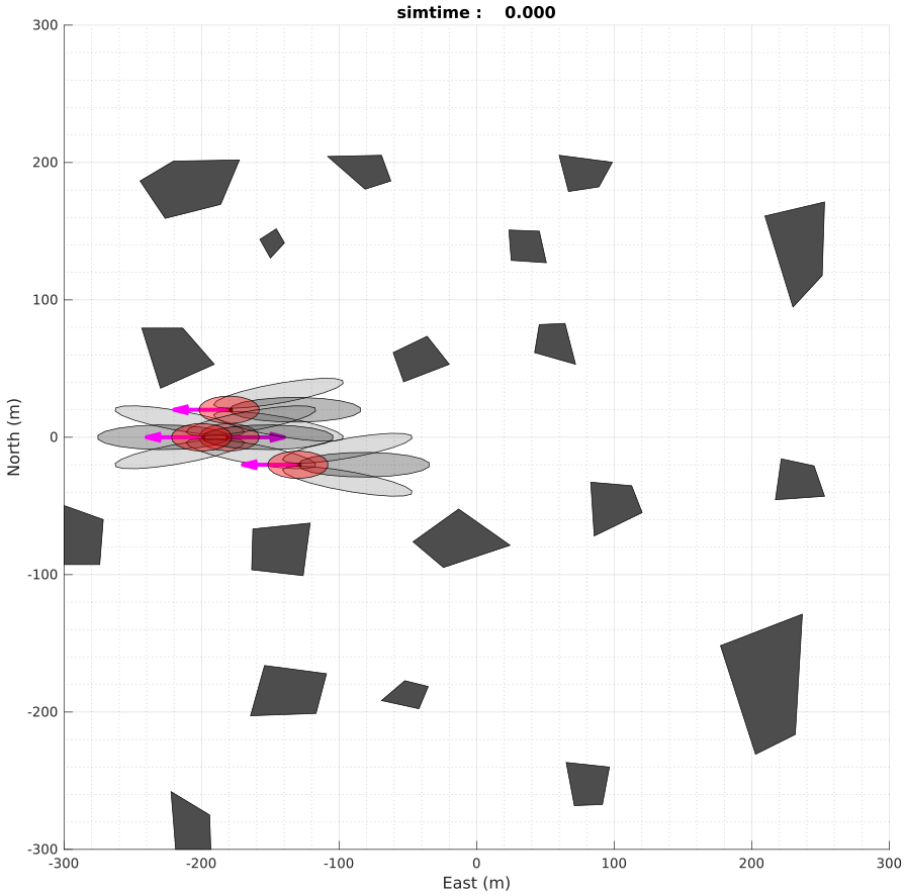

Figure 4.11: *Islands* map

Some important parameters used are: Map size = $600\,m \times 600\,m$, USV length = $L_u = 5\,m$, traffic vessel length = $L_v = 10\,m$, traffic vessel speed = $1 - 2\,m\,s^{-1}$. Other parameters are as in Table 4.3. The turning radius and maximum speed of the USV is $5\,m$ and $2\,ms^{-1}$ respectively.

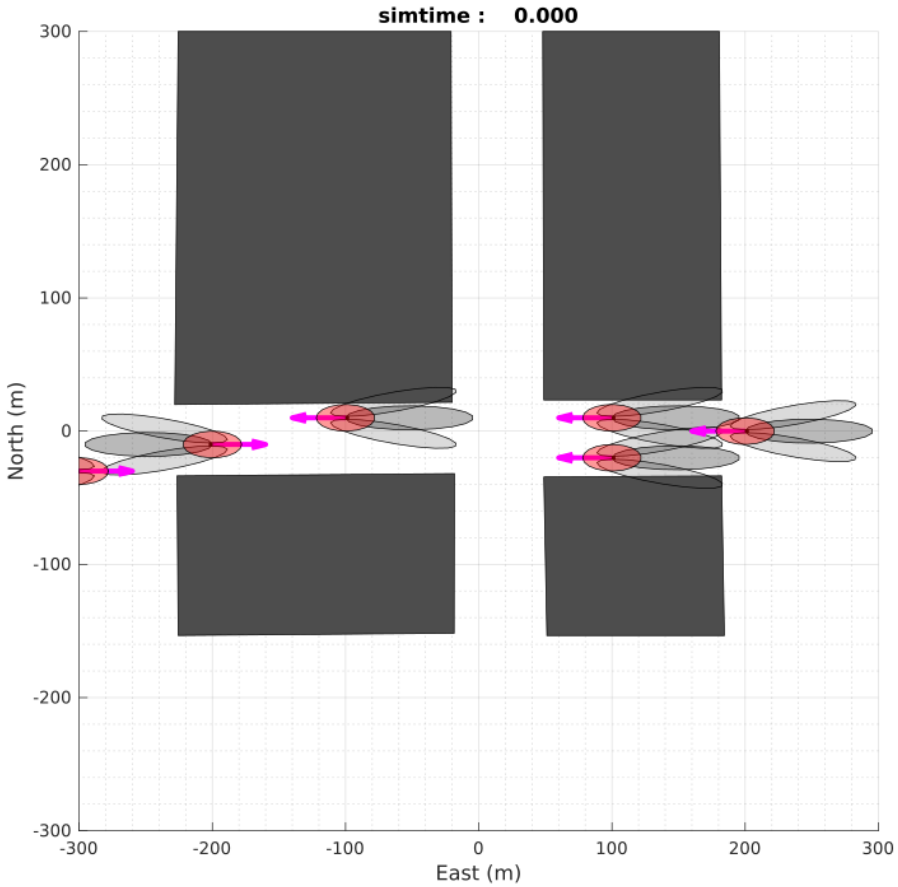

Figure 4.12: *Busy* map

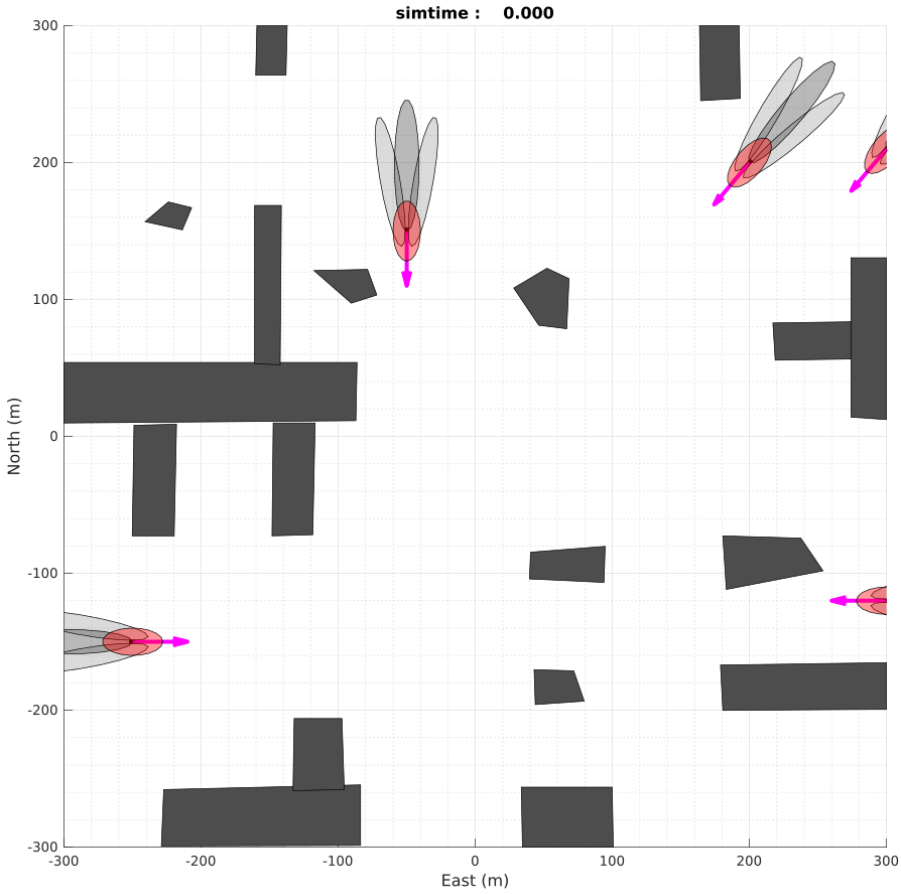

Figure 4.13: *Port* map

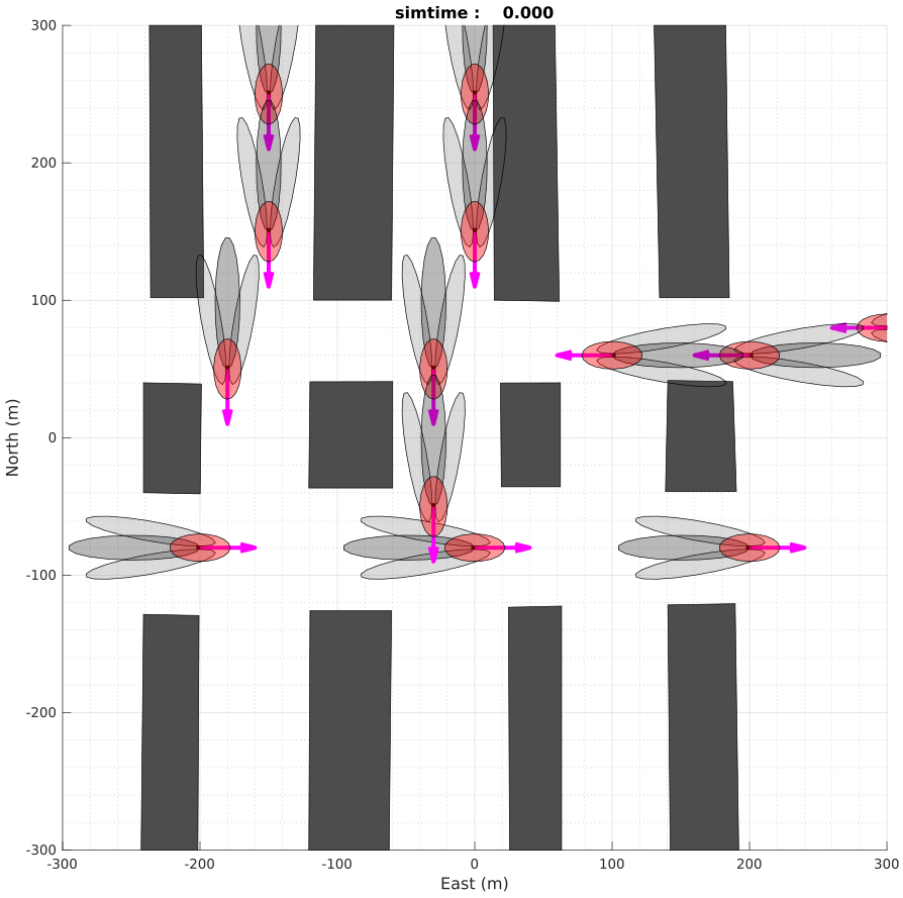

Figure 4.14: *Stress* map

The traffic vessels are placed to trigger interesting dynamic obstacle avoidance as well as wave field crossing behavior.

Our approach (WA-TP: Wave-Aware Trajectory Planner) is compared with three different configurations of the planning stack. These configurations are summarized in Table 4.4. Global Path Planner (G-PP) ignores the dynamic obstacles and generates only kinematically feasible paths to the goal. G-PP can be realized by removing traffic vessels from the implementation of WA-TP. Conservative Trajectory Planner (C-TP) considers the traffic vessels along with their wavefields as a non-permeable dynamic obstacles to yield conservative trajectories. C-TP can be realized through setting $M = 0$ within the wavefield in the implementation of WA-TP. The configuration G-PP + VO follows the generated path and avoids traffic vessels using Velocity Obstacles (VO). G-PP + C-VO is a conservative version of G-PP + VO which treats traffic vessels along with their wavefields as obstacles. The configuration C-TP + VO follows the conservative trajectory generated by C-TP and avoids traffic vessels using VO.

Table 4.5 the execution time, collision count, success probability and distance to an obstacle at Closest Point of Approach (CPA). Success probability is calculated after each run using the executed trajectory as in Section 4.4.1.5 (cost assessment).

Table 4.4: Planning stack configurations

Configuration	Feature Enabled		
	Traffic vessel	Wavefield	VO
G-PP + VO	✗	✗	✓
G-PP + C-VO	✗	✗	✓*
C-TP + VO	✓	✓($M = 0$)	✓
WA-TP + VO	✓	✓	✓

Table 4.5 shows aggregated performance metrics for 20 runs of three different scenarios for different configurations of the planning stack. Relying on path planning and reactive collision avoidance with large dynamic obstacles as in G-PP + VO and G-PP + C-VO results in collisions, high failure probabilities and increased execution time. On the other hand, conservative trajectory

planning as in C-TP helps prevent collisions and failures at the cost of increased execution time. The results show that the proposed approach safely reduces execution time by roughly 13%, 24% and 3% for scenarios S1, S2 and S3 respectively when compared to the next safe alternative.

Table 4.5: Comparison of performance.

Metric	Scenario	G-PP + VO	G-PP + C-VO	C-TP + VO	WA-TP + VO
Execution time [s]	S1	442.6	456.7	465.1	405.2
	S2	436.7	422.4	484.1	367.5
	S3	384.3	415.5	367.2	355.1
Collision count	S1	2.0	2.0	0.0	0.0
	S2	1.2	1.9	0.0	0.0
	S3	0.0	0.2	0.0	0.0
Success probability	S1	0.0	0.0	1.0	0.9975
	S2	0.0	0.0	0.9964	0.9991
	S3	0.9739	0.0728	0.9716	0.9894
CPA distance [m]	S1	1.6	1.8	17.8	46.3
	S2	7.5	5.6	20.5	17.7
	S3	22.9	12.9	20.4	15.3

The effects of changing the MTBF parameter of the failure model is shown in Table 4.2. Figure 4.15 shows the initial configuration of USV and traffic vessels in the *islands* map. Traffic vessels were initialized to speeds between $1 - 2\,m\,s^{-1}$. The USV was setup to use WA-TP + VO and commanded to cross the busy channel. In this experiment, the parameter controlling the danger level of the wave region (i.e., MTBF value M_t) is changed. Two different values of M_t are used simulate two different levels of danger. In Figure 4.16 and 4.17, the wavefields are made to be dangerous by assigning a lower MTBF parameter ($M_t = 1000$). This results in the USV taking a longer trajectory to reach the goal point while altogether avoiding any contact with the wavefield. In Figure 4.18 and 4.19, the wavefields are made to be permeable by assigning a high MTBF parameter ($M_t = 3000$), making the traversal of the wavefield less dangerous. Thus, in this case, the USV attempts crossing the wavefield at an appropriate angle as it makes it way to the goal point. Figure 4.16 and Figure 4.17 show snapshots of the executed trajectory when $M_t = 1000$. Similarly, Figure 4.18 and Figure 4.19 show the case when $M_t = 3000$.

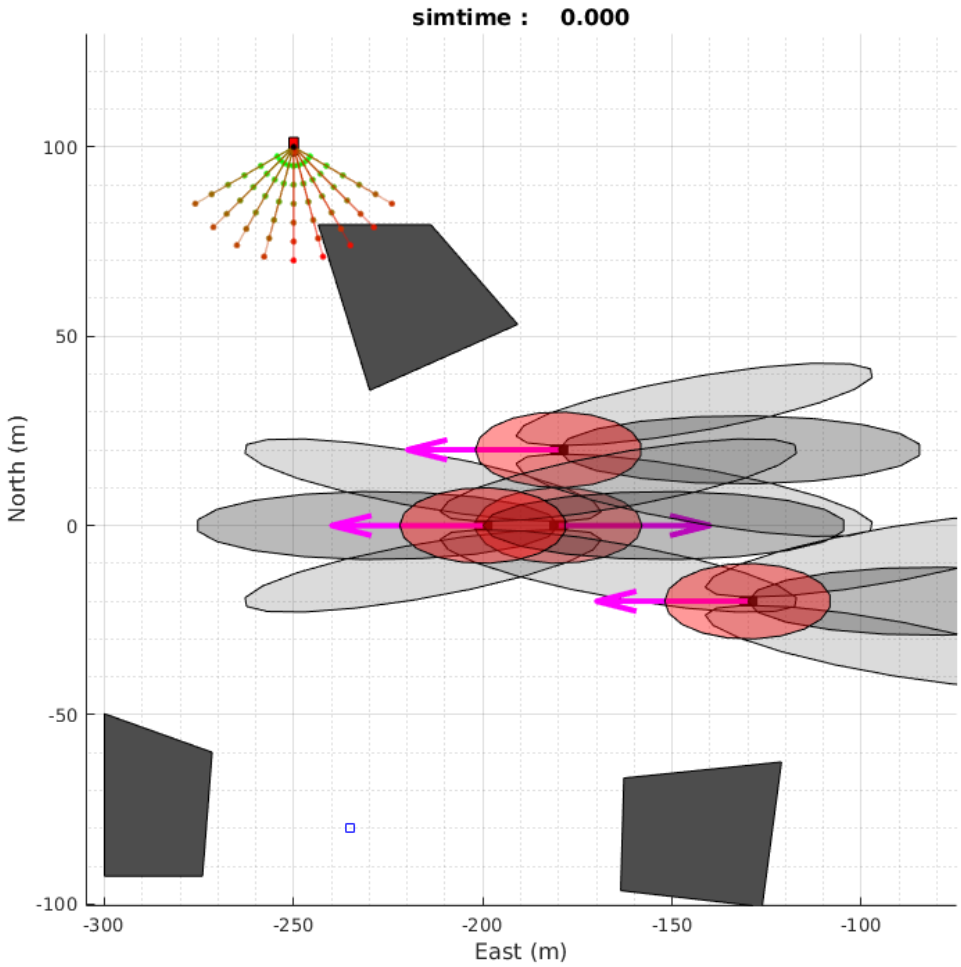

Figure 4.15: Scenario to illustrate the effects of changing the MTBF parameter M_t. USV is positioned at the start point $(-250, 100)$ and commanded to reach the blue square at $(-235, 80)$

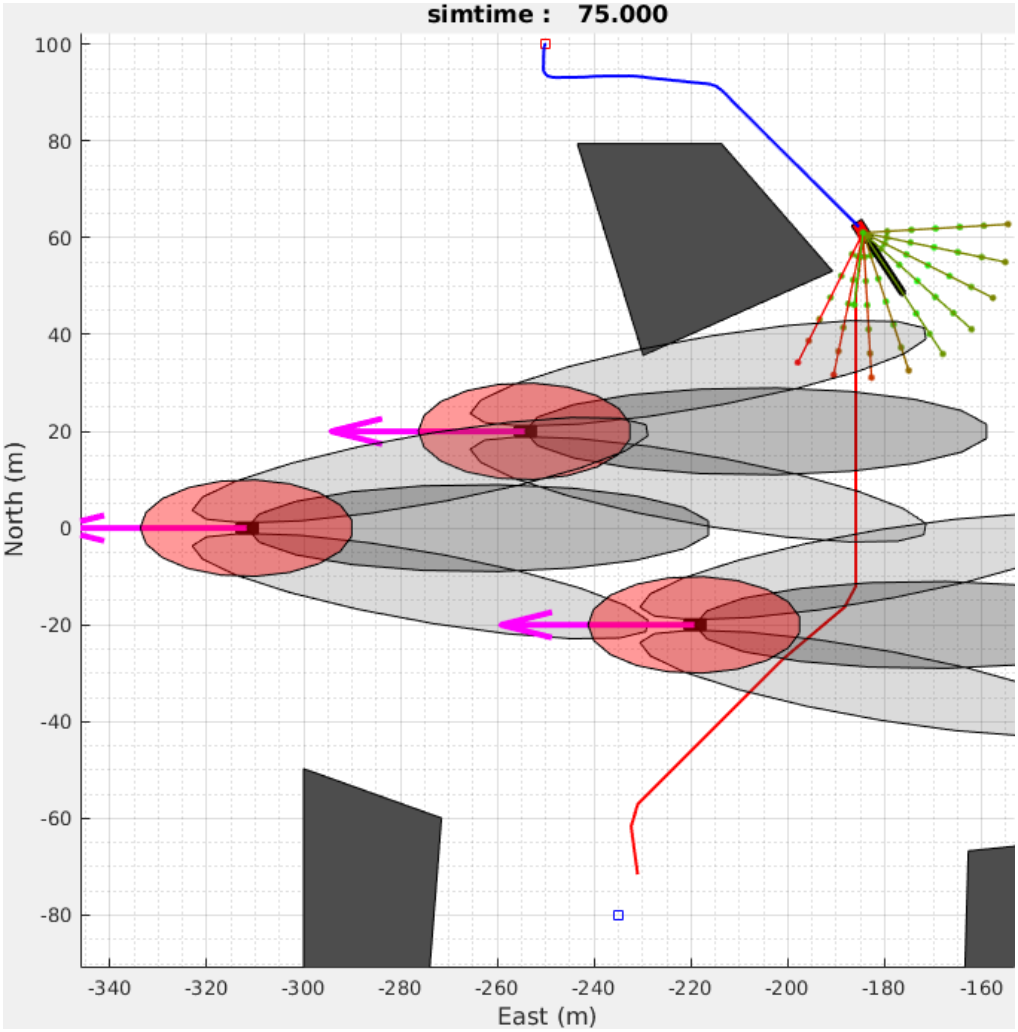

Figure 4.16: Case $M_t = 1000$. At $t = 75.0$, the USV is seen avoiding crossing wavefield and taking a longer route to the goal point.

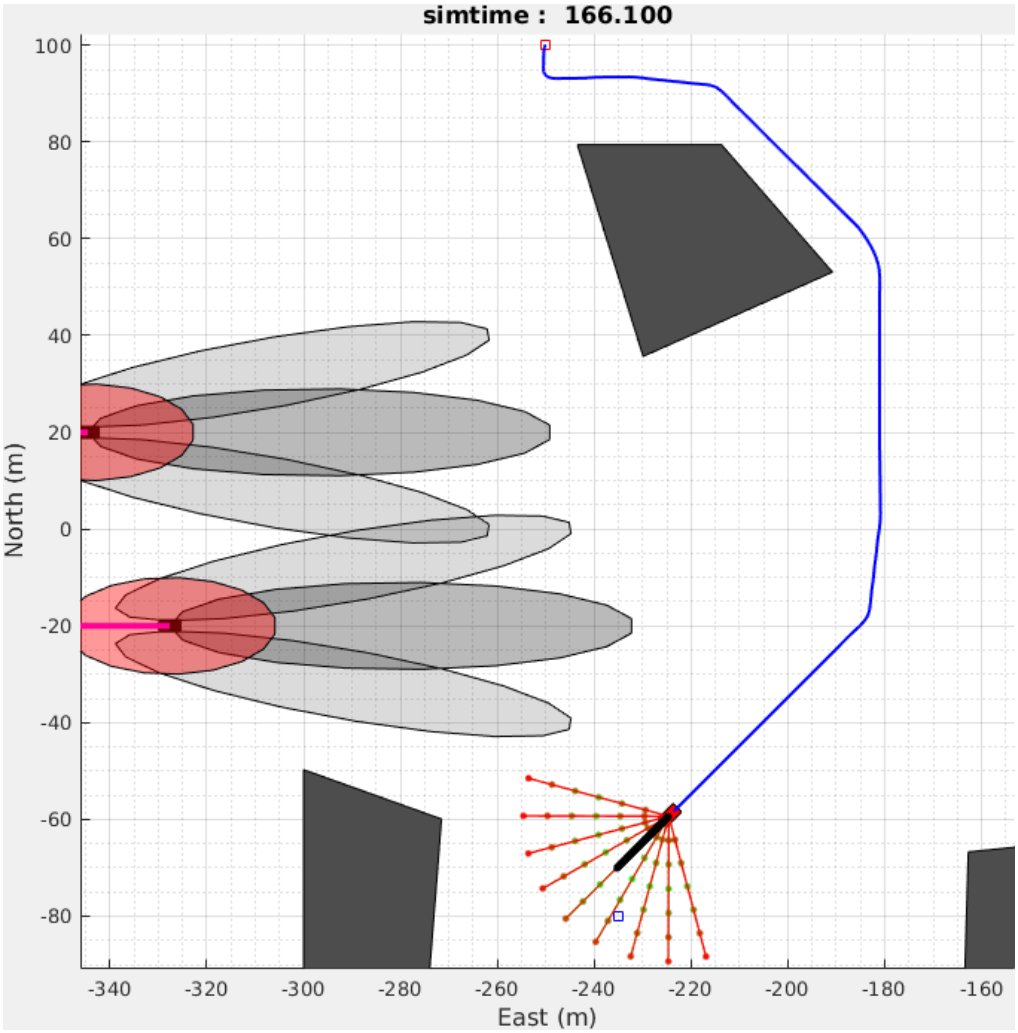

Figure 4.17: Case $M_t = 1000$. USV reaches goal point at $t = 166.1$

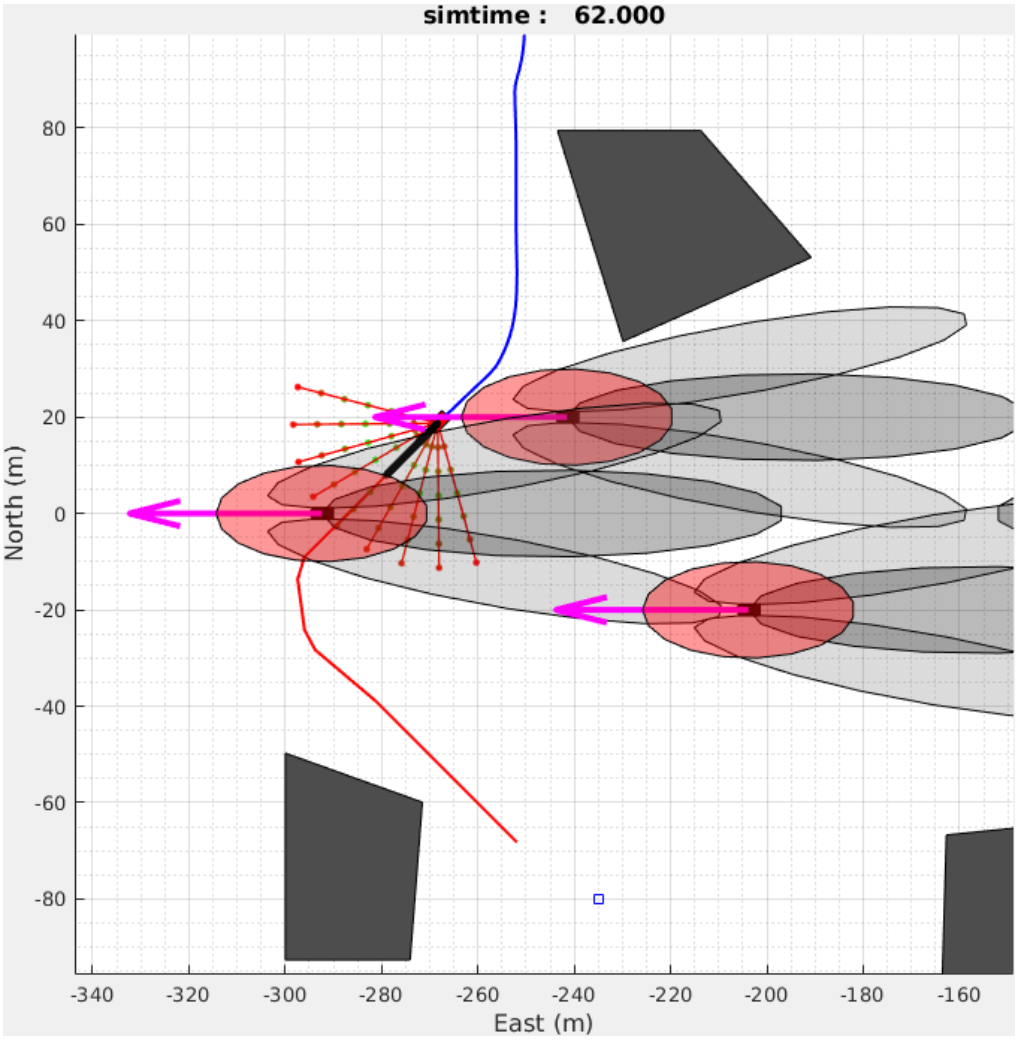

Figure 4.18: Case $M_t = 3000$. At $t = 62.0$, the USV is seen a crossing wavefield at the appropriate angle.

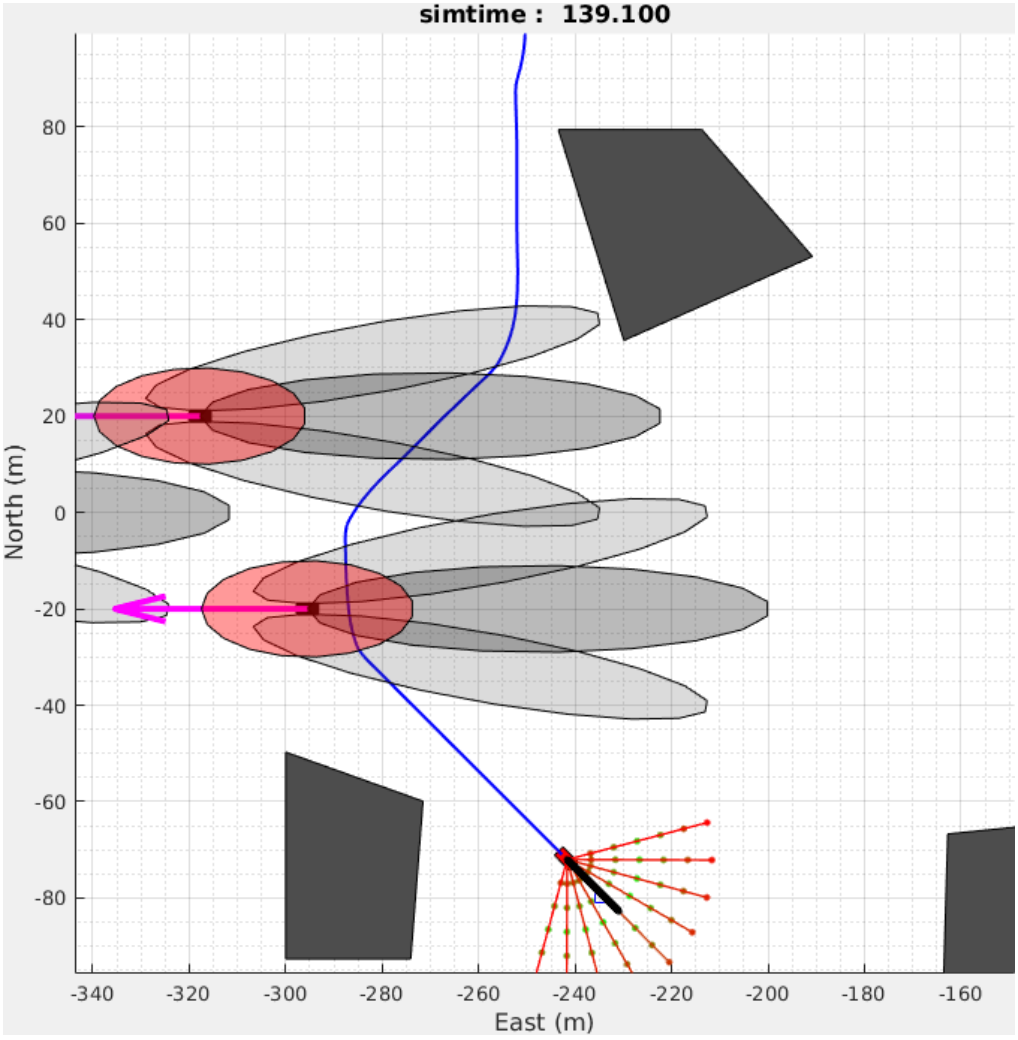

Figure 4.19: Case $M_t = 3000$. USV reaches goal point at $t = 139.1$

To study the effects of performance enhancing techniques, the planner was invoked in the *port* map enabling and disabling subsets of these techniques. The baseline configuration of the planner uses Euclidean distance based time heuristic (i.e. $h(s, g)$ = time taken using straight line path from s to g using maximum speed), the finest motion goal set and a sample size of $n_s = 30$. The results are shown in Table 4.6.

Table 4.6: Impact of speed-up techniques on performance.

Speed-up technique	Planning Time [s]			Expansion Count			Solution Cost		
	S1	S2	S3	S1	S2	S3	S1	S2	S3
Baseline	144.0	148.9	38.0	597	565	190	373.8	308.5	298.0
SOH	19.1	63.5	12.2	64	277	42	385.9	314.3	315.7
SOH + DOH	16.8	34.1	8.7	52	125	28	389.5	338.1	322.4
SOH + DOH + AMGS	7.2	15.2	6.2	20	32	22	406.3	358.2	341.1
SOH + DOH + AMGS + AS	4.5	4.1	2.5	20	28	12	397.7	351.1	341.7
Improvement over baseline %	96.8	97.2	93.4	96.6	95.0	93.7	-6.4	-13.8	-14.7

Between the two heuristics (SOH and DOH), the effect of static obstacle heuristics (SOH) has a huge impact on the performance, reducing the expansion counts by at least 51%. While the dynamic obstacle heuristics help reduce expansion counts on top of SOH, the effect is not so pronounced when only a few dynamic obstacles are encountered for a particular trajectory connecting the start/goal pair. There is a slight increase in the solution cost due to the resolution of the heuristic lookup table. Introducing adaptive motion goals reduces the expansion count by at least 21%, but increases the solution cost at worst by only 6%. Adaptive sample sizes for the Monte Carlo failure probability computation helped reduce the planning time by only sampling densely where interaction with static obstacles and dynamic obstacles were possible. It was observed that approximately $60\% - 70\%$ of the planning time was spent in sampling dynamic obstacles and performing collision checking.

4.5.1.1 Sensitivity to Number of Dynamic Obstacles

The performance of the planner was studied when the number of dynamic obstacles was varied between $1 - 15$. Dynamic obstacles were randomly spawned along the sampling lines the map going towards various directions at speeds between $1.0 - 3.0\, m\, s^{-1}$ as shown in Figure 4.20. Thirty trials were conducted for random start points in the start region and random end points in the goal region for each possible number of dynamic obstacles. Figure 4.21 shows the variation in the planning time as the number of dynamic obstacles encountered increases. It is observed that the planning times are under $6\, s$ for dynamic obstacle counts of less than 7. Beyond 7 dynamic obstacles, the planning time grows linearly with the number of dynamic obstacles.

4.5.1.2 Sensitivity to Dynamic Obstacle Uncertainty

A study on how increased dynamic obstacle perception uncertainty affects the executed trajectory was conducted. Specifically, two levels of perception uncertainty were used: low and high in a narrow channel scenario shown in Figure 4.22.

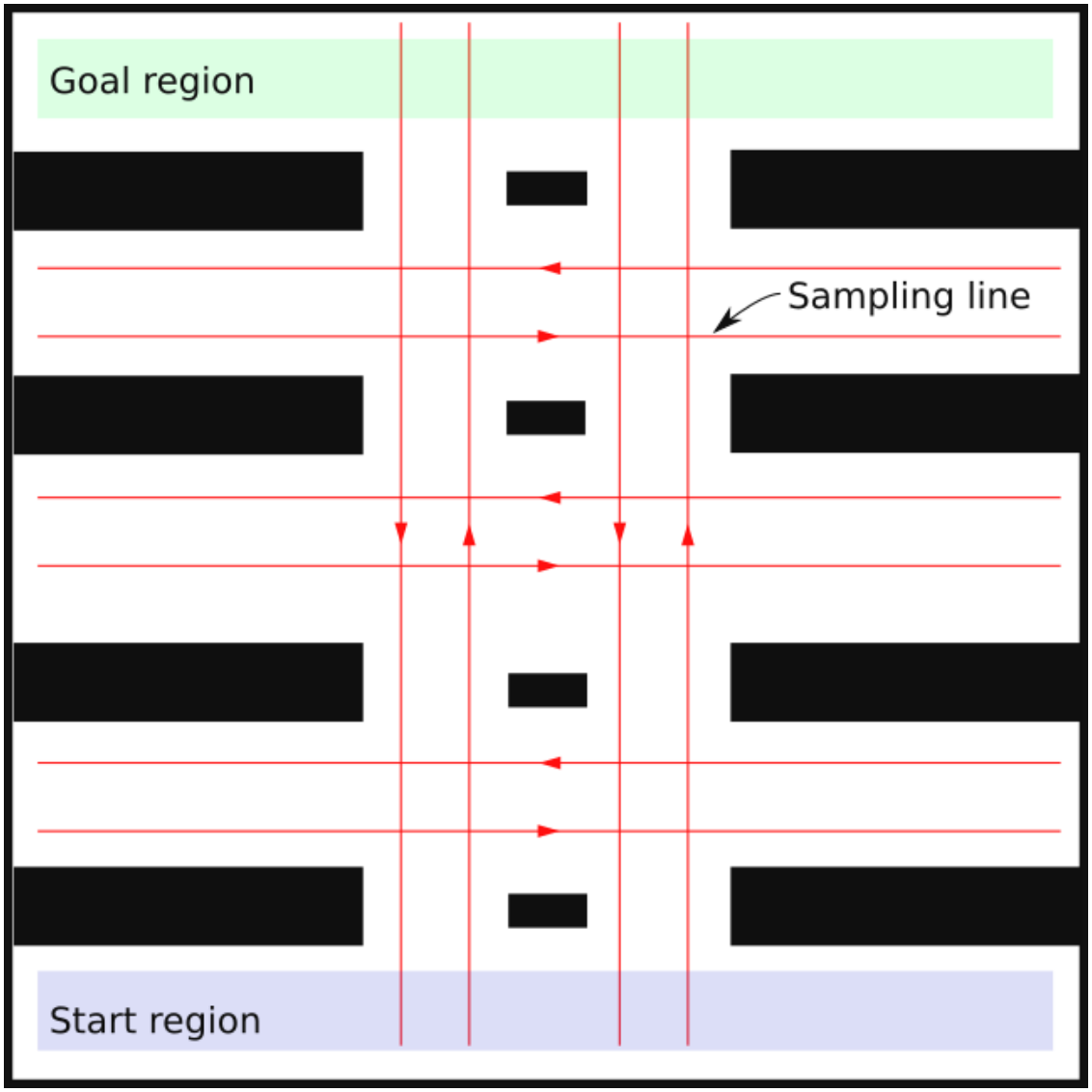

Figure 4.20: The scenario used for evaluating the effect of increasing the number of dynamic obstacles. Random start states and random goal states are sampled from the start region and the goal region respectively. Dynamic obstacles are randomly spawned on the sampling line and made to move at the indicated directions at various speeds.

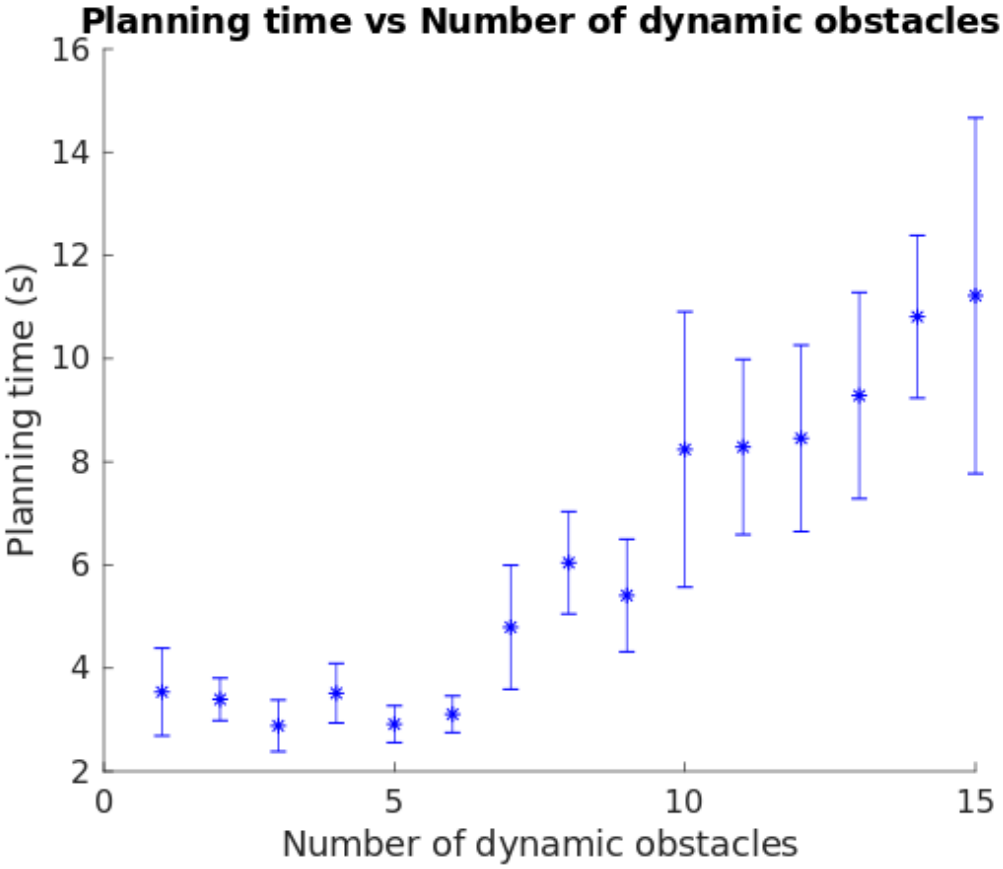

Figure 4.21: Planning time as the number of dynamic obstacles was varied between $1 - 15$.

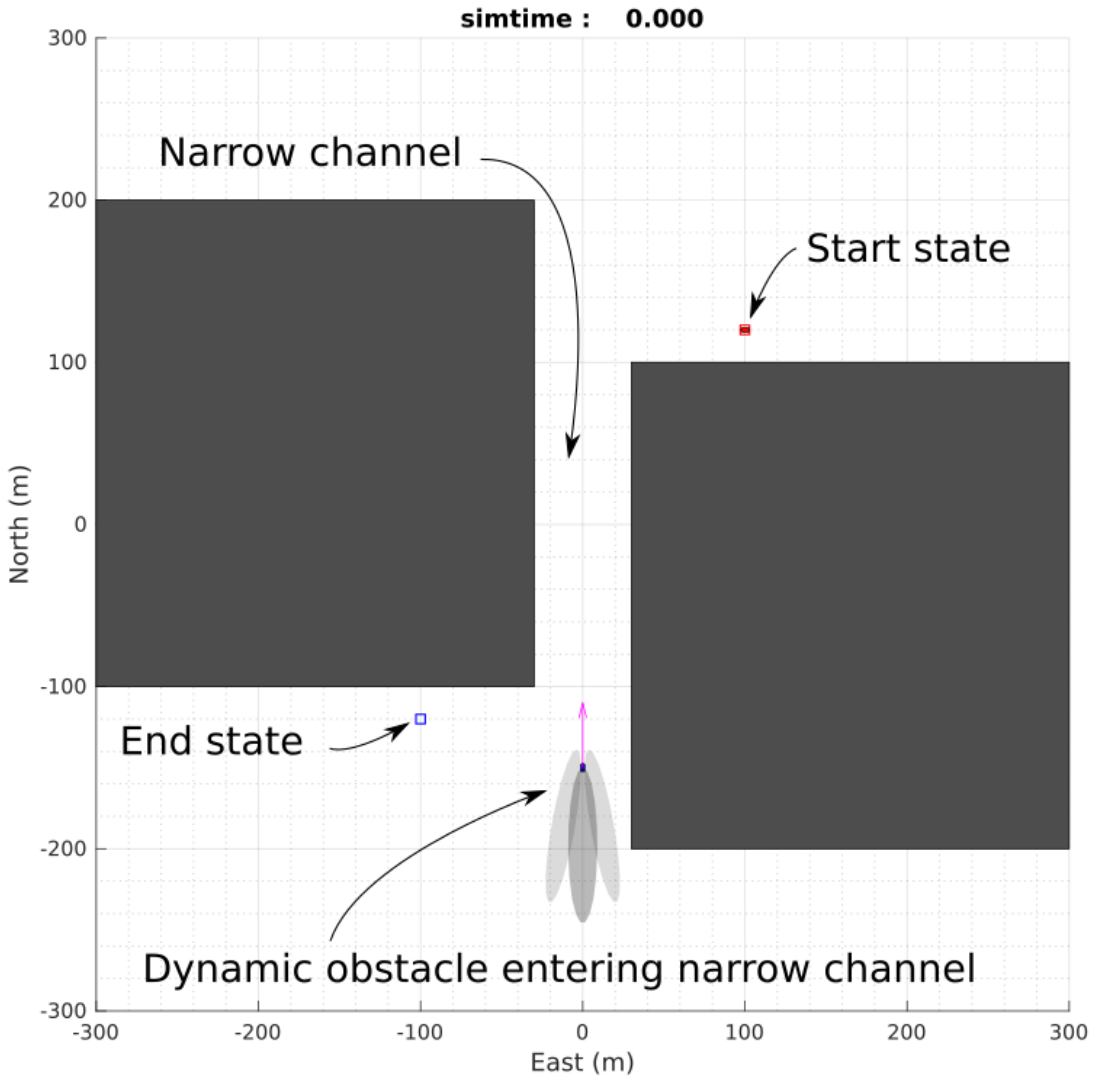

Figure 4.22: Scenario to illustrate the effects of low and high dynamic obstacle perception uncertainty. A dynamic obstacle is entering a narrow channel. Start and goal states are separated by the narrow channel. The USV must access the risks of entering the channel while another vessel is underway.

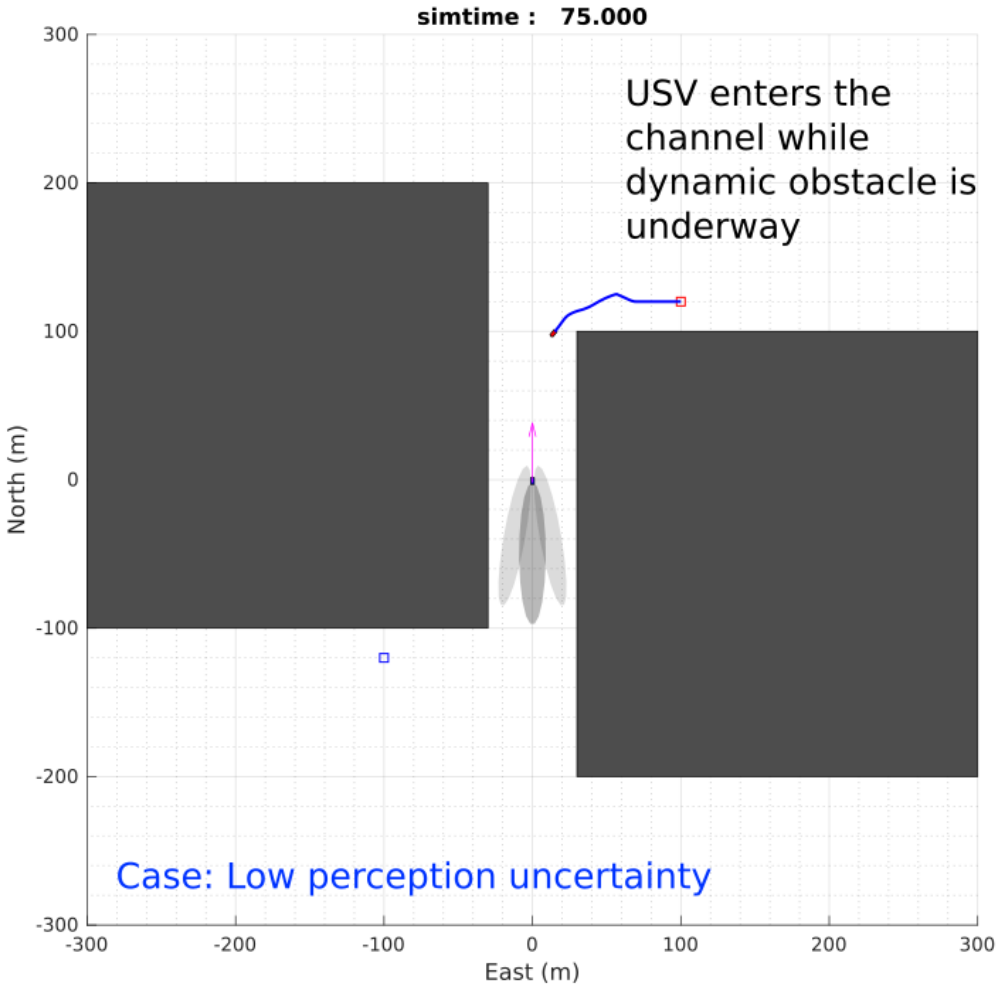

Figure 4.23: USV enters the channel as the estimates of dynamic obstacle are sufficient for passing the channel.

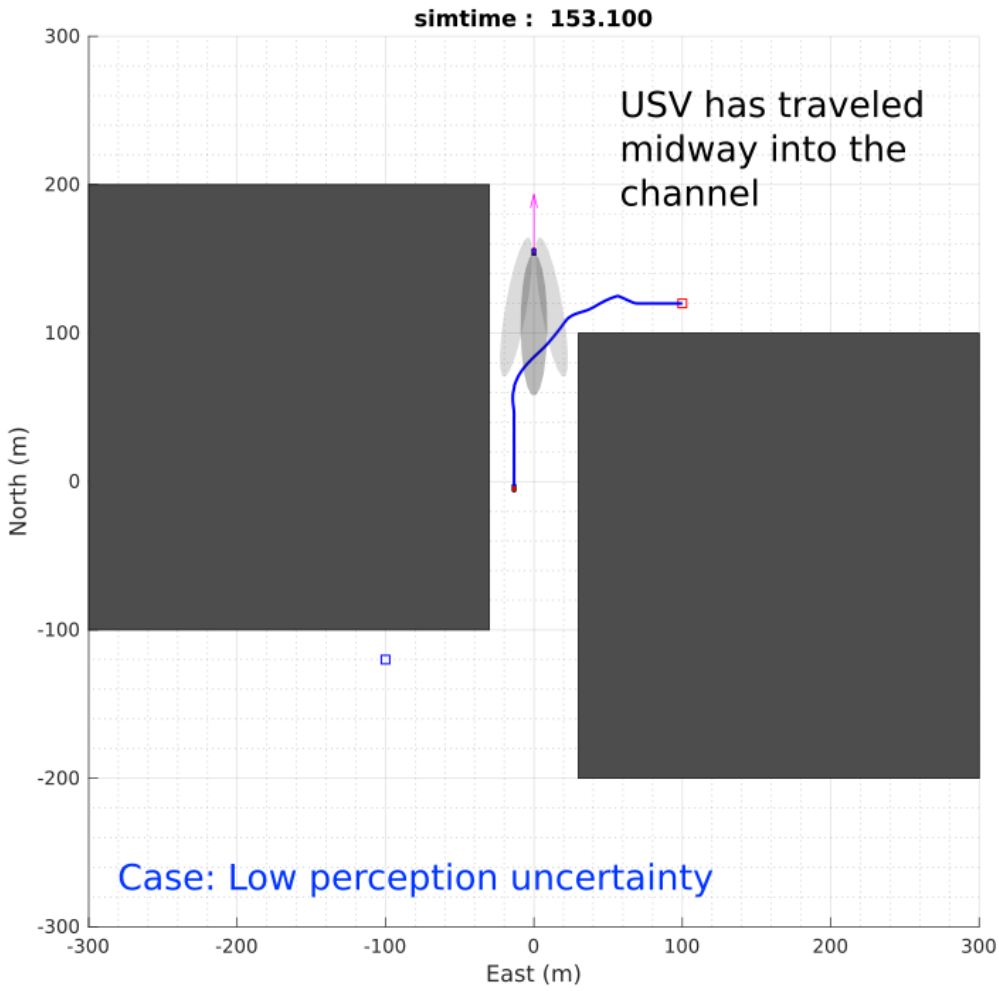

Figure 4.24: USV has traveled halfway into the channel after encountering the dynamic obstacle in a COLREGs compliant way.

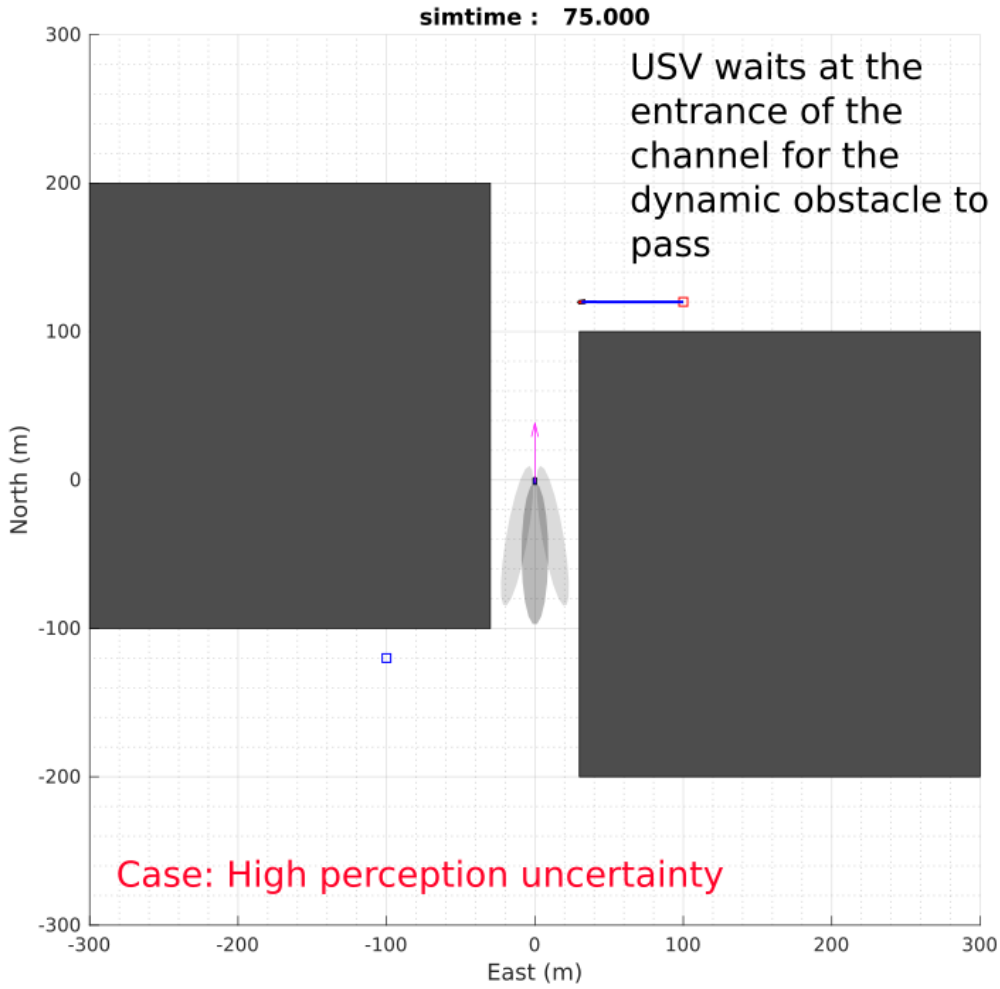

Figure 4.25: USV waits at the mouth of the channel for the dynamic obstacle to pass as it is too dangerous to attempt passing the channel when the estimate of the dynamic obstacle is bad.

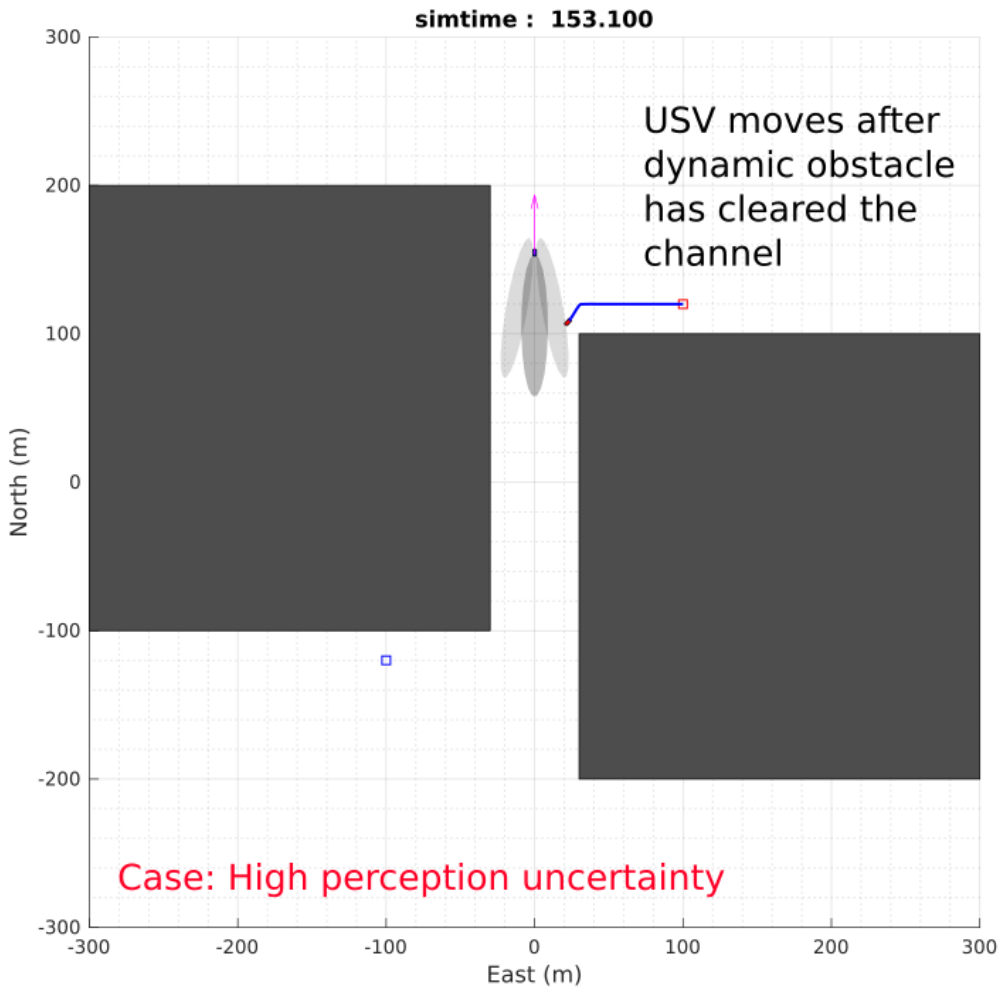

Figure 4.26: USV starts to move only after the dynamic obstacle has cleared the channel.

Figures 4.25 and 4.23 showcase the different behaviors of the USV when faced with differing levels of uncertainty in the perception of the dynamic obstacle. When the perception uncertainty is high, we observe the USV to act conservatively and wait at the mouth of the narrow channel for the dynamic obstacle to clear the channel. This behavior is due to the high perception uncertainty associated with the sensing of the traffic vessel. Entering the channel in this situation is a risky move. If the channel is entered without an accurate state estimate of the traffic vessel, a collision might be inevitable when the USV is unable to maneuver around the traffic vessel. This behavior is avoided. When the perception uncertainty is low, we observe the USV to enter the channel and pass the dynamic obstacle on the portside in compliance with COLREGs.

4.5.2 Physical Experiments

4.5.2.1 Experimental Setup

A set of experiments was performed using a 16 feet WAM-V USV in North Lake, Hollywood, Florida to verify the effectiveness of the proposed approach. Instead of physical static and dynamic obstacles, virtual obstacles were introduced to the navigation system in lieu of the perception subsystem.

4.5.2.2 Experimental Results

Table 4.7: Comparison of execution time.

Scenario	Planner Type	Execution time(s)	Reduction (%)
$P1$	C-TP + VO	148.9	-
	WA-TP + VO	136.5	8.3
$P2$	C-TP + VO	163.6	-
	WA-TP + VO	122.5	25.1

Dynamic obstacles were moving at constant speeds between $1.0 - 2.0\, m\, s^{-1}$. During experiments, we noted south-easterly winds ranging between $10 - 12\, KTS$. From Table 4.7, we observe that execution times are reduced by as much as 25.1% under the proposed approach.

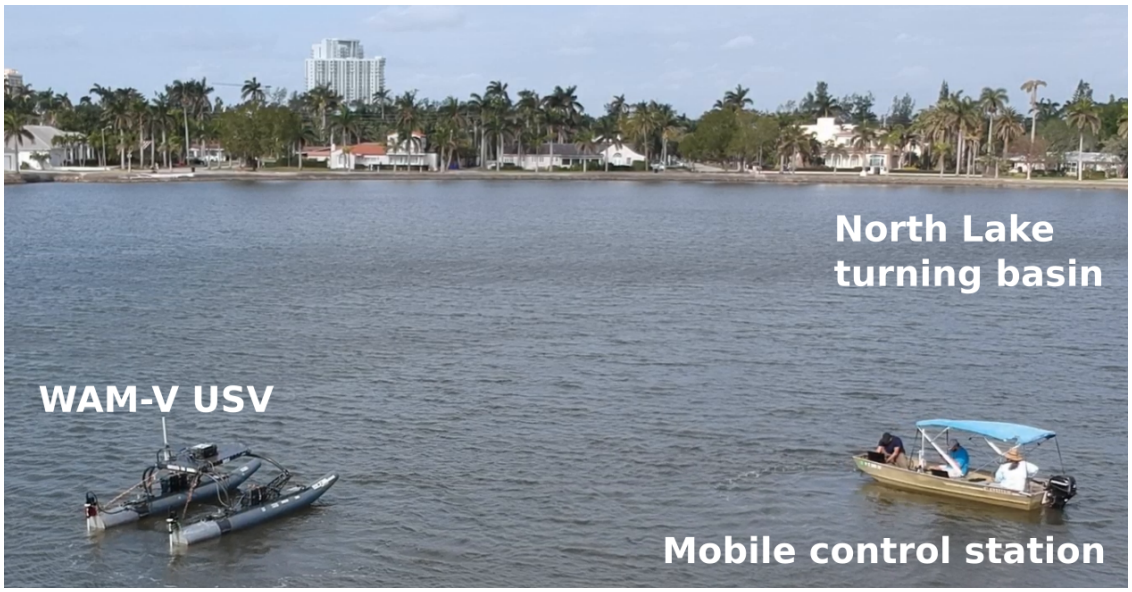

Figure 4.27: Setup.

4.5.3 Comparison of MDP and Graph Search

Although, motion planning under motion and perception uncertainty is formally described as a POMDP. In practical trajectory planning scenarios for USVs, the localization accuracy is good enough and does not warrant a POMDP formulation. In light of this, we ignore perception uncertainty pertaining to localization of the agent and construct an MDP formulation of trajectory planning in the presence of dynamic obstacles. We prove that, under some assumptions, MDP problems arising in the context of practical trajectory planning can be solved using any DP (Dynamic Programming) techniques instead of SDP (Stochastic Dynamic Programming) without compromising the optimality of solutions.

Let $\mathcal{X} \subset \mathbb{R}^n$ be a discrete state space consisting of the state $\mathbf{s} = (x, y, t)$. This state space is not densely sampled over the spatial and time dimensions. Rather, $\mathcal{X}$ consists of clusters of states that are separated by a fixed distance (see Figure 4.29). Let $C = \{\mathbf{s}_0, \cdots, \mathbf{s}_k\}$ denote a cluster of states. If a state $\mathbf{s}$ belongs to C and we want to draw attention to it, we denote it using the notation ${}^{C}\mathbf{s}$. Each cluster is centered on a nominal state $\mathbf{s}_c$. Let $\mathcal{C}$ denote the set of all clusters in $\mathcal{X}$. Let $\mathcal{U} = \{L, R, F, B, FL, FR, BL, BR, S\}$ be the space of actions. The

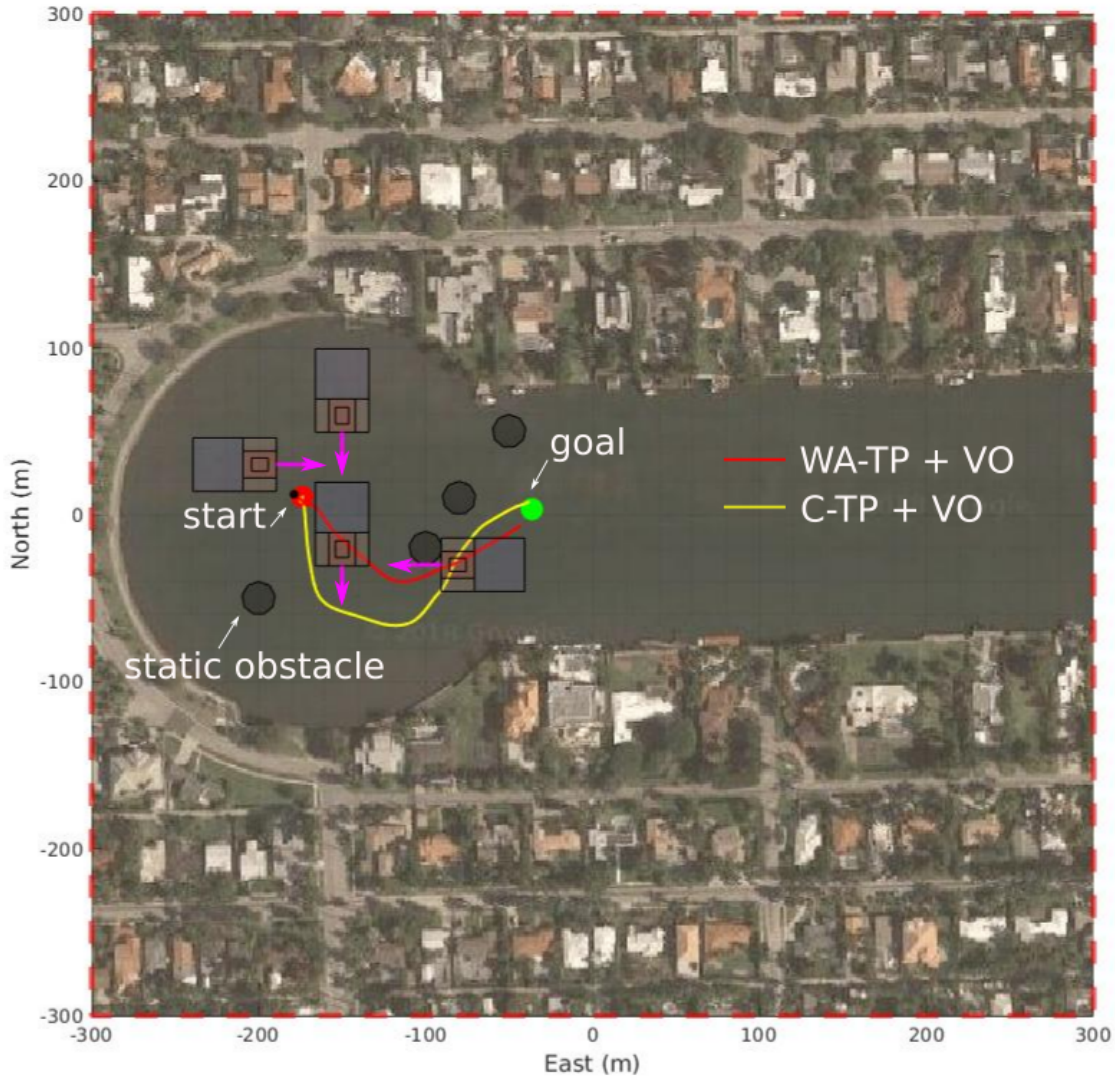

Figure 4.28: One set of trajectories from scenario $P1$ comparing wave-aware trajectory with a conservative trajectory.

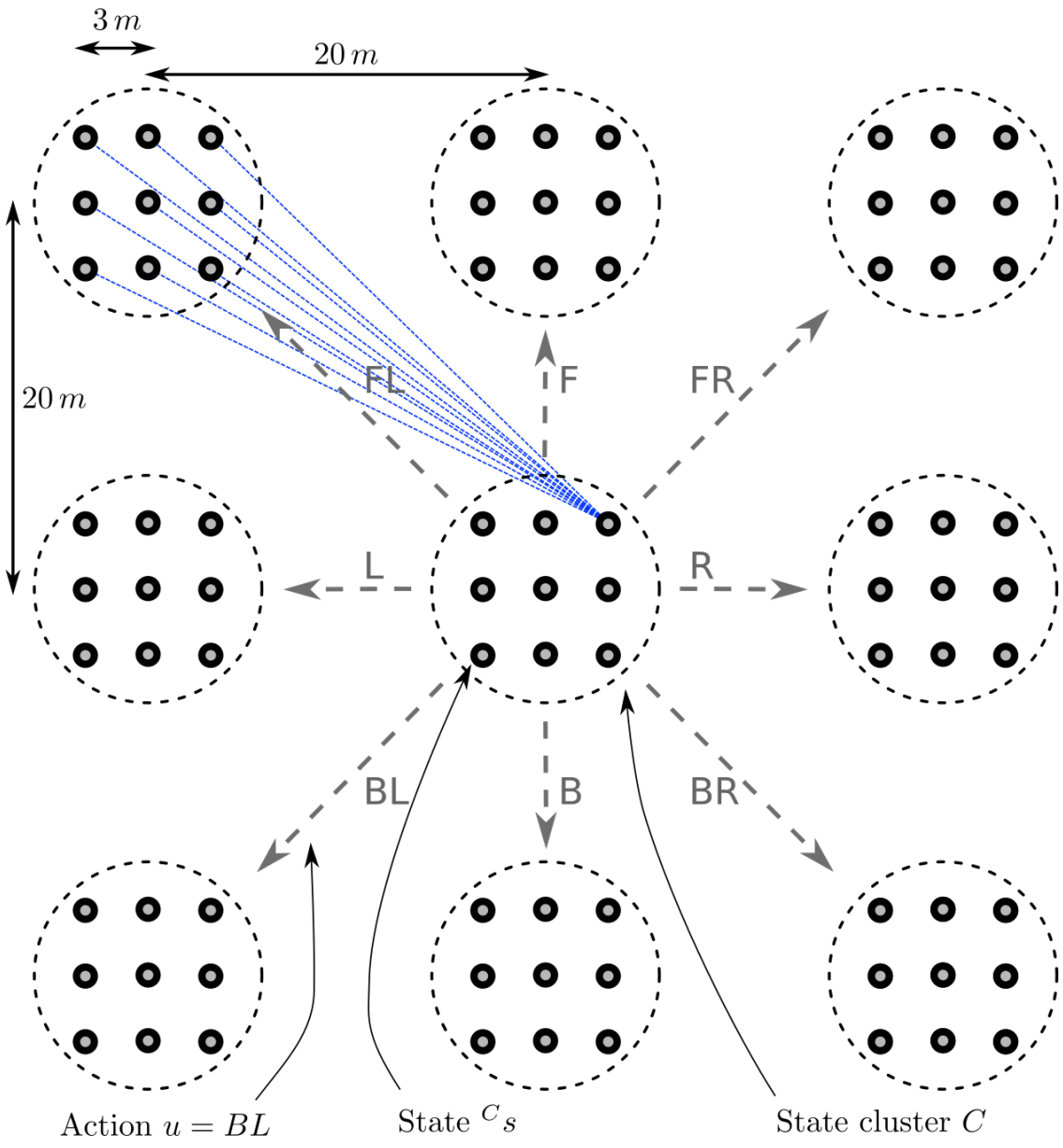

Figure 4.29: States, clusters and actions are illustrated. For a particular state, the possible next states when the action $u = FL$ is applied is shown using dashed blue lines.

actions in $\mathcal{U}$ are shorthand for cardinal motions over 2D space (i.e. L=left, R=right, F=forward, B=backward, FL=forward-left, FR=forward-right, BL=backward-left, BR=backward-right and S=stay). The stay action waits at the same spatial location for a fixed period of time. Suppose feedback controllers realize these motions over distances of upto $20\,m$ (e.g. $20\,m$ forward) while minimizing drift due to motion uncertainty. It is assumed that the typical motion uncertainty faced is relatively small (i.e. under $5\,m$ compared to the large $20\,m$ motion). And, therefore these controllers are robust enough to suppress the motion uncertainty propagation and guarantee that the USV moves as planned from any state in one cluster to within some state in another cluster. As a result, the inter-cluster transitions are deterministic and given by the invertible mapping $T_c \colon \mathcal{U} \times \mathcal{C} \to \mathcal{C}, \quad (u, C_{src}) \mapsto C_{dst}$. Thus, for a given action $u \in \mathcal{U}$ from any state $\mathbf{s}$ in the source cluster C_{src}, the destination cluster C_{dst} is determined without any stochasticity. For example, given any state $\mathbf{s}$ in a cluster C_{src} centered at $(x = 0, y = 0, t = 0)$, the action $u = R$ results in the controller moving the USV at an average speed of $5\,m\,s^{-1}$ while targeting the state $\mathbf{s}_c$ in the cluster C_{dst} centered at $(x = 20\,m, y = 0\,m, t = 4\,s)$. Let $F_c \colon \mathcal{C} \times \mathcal{C} \to \mathbb{R}^+$ be a cluster transition cost function that assigns a cost for the move from one cluster to another. At this point, we recognize the tuple $W_c = \langle \mathcal{C}, T_c, F_c \rangle$ as a specification of a dynamic programming (DP) problem minimizing cumulative transition costs over nominal states of clusters between the start cluster C_s and the goal cluster set $G = \{C_{g,k}\}_{k=0}^{K}$. Each $C_{g,k} \in G$ has the same spatial location but varies in time value. This captures the open-ended time dimension upto time step K.

While the feedback controllers are able to drive the state to the general vicinity of a destination cluster C_{dst}, the exact state ${}^{C_{dst}}\mathbf{s}'$ reached by the controller is stochastic. To incorporate this motion uncertainty, let us formulate an MDP. Let $E \subset \mathcal{X} \times \mathcal{X}$ be the set of feasible state transitions satisfying the condition $(\mathbf{s}_i, \mathbf{s}_f) \in E \iff \exists C_{src}, C_{dst} \in \mathcal{C}, \exists u \in \mathcal{U}$ s.t. $C_{dst} = T_c(u, C_{src}), \mathbf{s}_i \in C_{src}, \mathbf{s}_f \in C_{dst}$. The transition probabilities between any two states in $\mathcal{X}$ are given by the mapping $P \colon E \to [0, 1]$ (the action space $\mathcal{U}$ is redundant and hence, omitted because T_c is a invertible map. i.e. the action is completely determined by the source and destination cluster). Let $F \colon E \to \mathbb{R}^+$

be a state transition cost function that assigns a cost for the move from one state to another. Let $R\colon E \to \mathbb{R}$ be a state transition reward function that assigns a cost for the move from one state to another such that:

$$R(\mathbf{s}_1, \mathbf{s}_2) = \begin{cases} 0 & \text{if } L[\mathbf{s}_2], L[\mathbf{s}_1] \in G \\ -F(\mathbf{s}_1, \mathbf{s}_2) & \text{otherwise.} \end{cases}$$

Let $\pi\colon \mathcal{X} \to \mathcal{U}$ be a policy that gives an action $u \in \mathcal{U}$ for every state in $\mathcal{X}$. Thus, the MDP defined by the tuple $M = \langle \mathcal{X}, \mathcal{U}, P, R \rangle$. The goal of the MDP formulation is to find a optimal policy π^* such that it maximizes the cumulative rewards accumulated over state transitions leading to one of the goal states from any given state.

We would like to know if SDP is the only way to solve this MDP and under what conditions a DP approach can be used. To illustrate this question, consider an example of a DP as shown in Figure 4.30. Suppose, ${}^{C_s}\mathbf{s}_c$ is the start state and ${}^{C_g}\mathbf{s}_c$ is one possible goal state. Ignoring motion uncertainty, let us suppose we perform DP (e.g. A*) over the nominal states of clusters, and suppose it yields an optimal path $({}^{C_s}\mathbf{s}_c, {}^{C_a}\mathbf{s}_c, {}^{C_b}\mathbf{s}_c, {}^{C_g}\mathbf{s}_c)$ through nominal states of the clusters (shown in dashed green line). We then proceed with the transition from the cluster C_s to C_a. However, due to motion uncertainty, we are not able to reach ${}^{C_a}\mathbf{s}_c$ precisely. But, suppose we reached a nearby state ${}^{C_a}\mathbf{s}_n$ instead. Would the transition to C_b still be the next optimal action ? Would SDP produce a policy that prescribes a different action at ${}^{C_a}\mathbf{s}_n$ instead of the action towards C_b.

Theorem 1. *Suppose the following statements are true:*

1. *The* controller robustness condition *is satisfied.*

2. *An optimal solution to the problem W_c is a sequence of N clusters $q_c = \{C_k\}_{k=0}^{N-1}$.*

3. *We start in cluster C_s and end in some $C_g \in G$.*

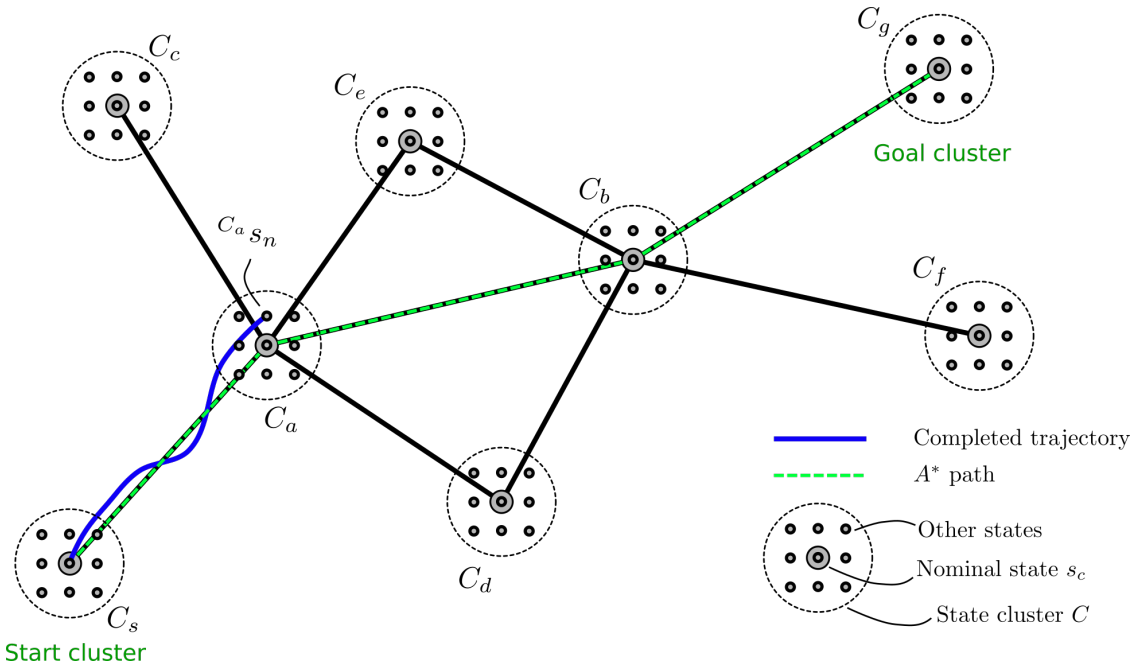

Figure 4.30: Example of a cluster graph showing a shortest trajectory from C_s to C_g computed using A^*. Part of the trajectory is complete but the trajectory has not been faithfully executed resulting in a slight deviation at C_a. Would the optimal MDP policy change in light of this deviation ?

Then, the cluster transition costs of reaching some C_g from C_s using the optimal sequence q_c is exactly the cluster transition costs incurred through following the optimal policy generated for the MDP M.

Proof. Let L be an operator that extracts the cluster a state belongs to (i.e. $L[^C\mathbf{s}] = C$). Let the condition $F(\mathbf{s}_1, \mathbf{s}_2) = F_c(L[\mathbf{s}_1], L[\mathbf{s}_2])$ where $(\mathbf{s}_1, \mathbf{s}_2) \in E$ be called the *controller robustness condition* (C.R.C.).

The structure of the MDP M allows for a reduction to a smaller MDP M_c defined over clusters. We proceed by following the work of [116] to reduce the MDP. Let $E_c \subset \mathcal{C} \times \mathcal{C}$ be the set of feasible cluster transitions. E_c satisfies the condition

$$(\mathbf{s}_1, \mathbf{s}_2) \in E \iff (L[\mathbf{s}_1], L[\mathbf{s}_2]) \in E_c.$$

Let $w\colon \mathcal{X} \to [0,1]$ be any weighting function where $\Sigma_{\mathbf{s}\in C} w(\mathbf{s}) = 1, \forall C \in \mathcal{C}$. This required for well-defined transition probability and reward functions. We define $P_c\colon E_c \to [0,1]$ as:

$$\begin{aligned} P_c(C_{src}, C_{dst}) &= \Sigma_{\mathbf{s}_1 \in C_{src}} \Sigma_{\mathbf{s}_2 \in C_{dst}} w(\mathbf{s}_1) P(\mathbf{s}_1, \mathbf{s}_2) \\ &= \Sigma_{\mathbf{s}_2 \in C_{dst}} P(\mathbf{s}_2) \Sigma_{\mathbf{s}_1 \in C_{src}} w(\mathbf{s}_1) \\ &= \Sigma_{\mathbf{s}_2 \in C_{dst}} P(\mathbf{s}_2) \\ &= 1 \end{aligned}$$

We define a reward function $R_c\colon E \to \mathbb{R}^+ \cup 0$:

$$\begin{aligned} R_c(C_1, C_2) &= \Sigma_{\mathbf{s}_1 \in C_1, \mathbf{s}_2 \in C_2} w(\mathbf{s}_1) P(\mathbf{s}_1, \mathbf{s}_2) R(\mathbf{s}_1, \mathbf{s}_2) \\ &= \Sigma_{\mathbf{s}_2 \in C_2} (P(\mathbf{s}_2) \Sigma_{\mathbf{s}_1 \in C_1} w(\mathbf{s}_1) R(\mathbf{s}_1, \mathbf{s}_2)) \end{aligned}$$

Applying the *controller robustness condition*,

$$\begin{aligned} R_c(C_1, C_2) &= \begin{cases} 0 & \text{if } C_1, C_2 \in G, \\ \Sigma_{\mathbf{s}_2 \in C_2} (P(\mathbf{s}_2) & \\ (-F_c(C_1, C_2) \Sigma_{\mathbf{s}_1 \in C_1} w(\mathbf{s}_1)) & \text{otherwise.} \end{cases} \\ &= \begin{cases} 0 & \text{if } C_1, C_2 \in G, \\ -F_c(C_1, C_2) & \text{otherwise.} \end{cases} \end{aligned}$$

Thus, a state-aggregated version of the MDP M can now be defined as $M_c = \langle \mathcal{C}, \mathcal{U}, P_c, R_c \rangle$ Note that M_c is now a dynamic programming problem because:

1. The set E_c expresses an adjacency list between the clusters,

2. The actions have deterministic outcomes (i.e. $P_c(C_1, C_2) = 1, \forall (C_1, C_2) \in E_c$),

Taking the negative of the reward function as the cost function, we can create another instance of DP as $W = \langle \mathcal{C}, E_c, -R_c \rangle$.

Furthermore, $F_c = -R_c$ (except for any pair in the goal set G). Graph search will converge as soon as any cluster in the goal set is reached excluding the possibility of examining any cost between pairs of clusters in the goal set.

Thus, the graph search problem W_c is exactly the same as W and hence, their solutions have identical optimal costs. □

For a deeper understanding, the MDP problem M and the graph search problem W_c was solved using the Python programming language on a GNU/Linux machine with Intel Xeon E3-1245 CPU running at 3.5GHz and 32 GB RAM. The MDP and A^* algorithms were run for the test scenario shown in Figure 4.31. In this scenario, there are 8 dynamic obstacles (in green and some are hidden) and a few static obstacles (black) over a $500\,m \times 500\,m$ area.

The clusters are spaced out uniformly every $20\,m$ in both x and y directions yielding $25 \times 25 = 625$ clusters. Each cluster has 9 possible states around the central nominal state (inclusive). These 9 states are arranged on a grid with $3\,m$ spacing. This attempts to capture the various possible end states at any cluster after a trajectory tracking controller reaches that cluster. The agent was assumed to move at a constant speed of $5\,m\,s^{-1}$. Diagonal moves between clusters such as FL, FR, BL, BR take $\sqrt{2}$ times longer (i.e. $\approx 5.65\,s$) than non-diagonal moves such as F, B, L, R that only take $4\,s$. Thus, time dimension needs to include all possible combinations of diagonal moves and non-diagonal moves resulting in 530 possible times values between $t = 0\,s$ and $t = 150\,s$. Thus, $|\mathcal{X}| = 25 \times 25 \times 9 \times 530 = 2981250$ states. The cost function for transitioning from one cluster to another cluster is the time difference between the clusters (i.e. destination cluster time - source cluster time = $\Delta t \in \{4\sqrt{2}, 4\}$). If a transition results in collision, then the

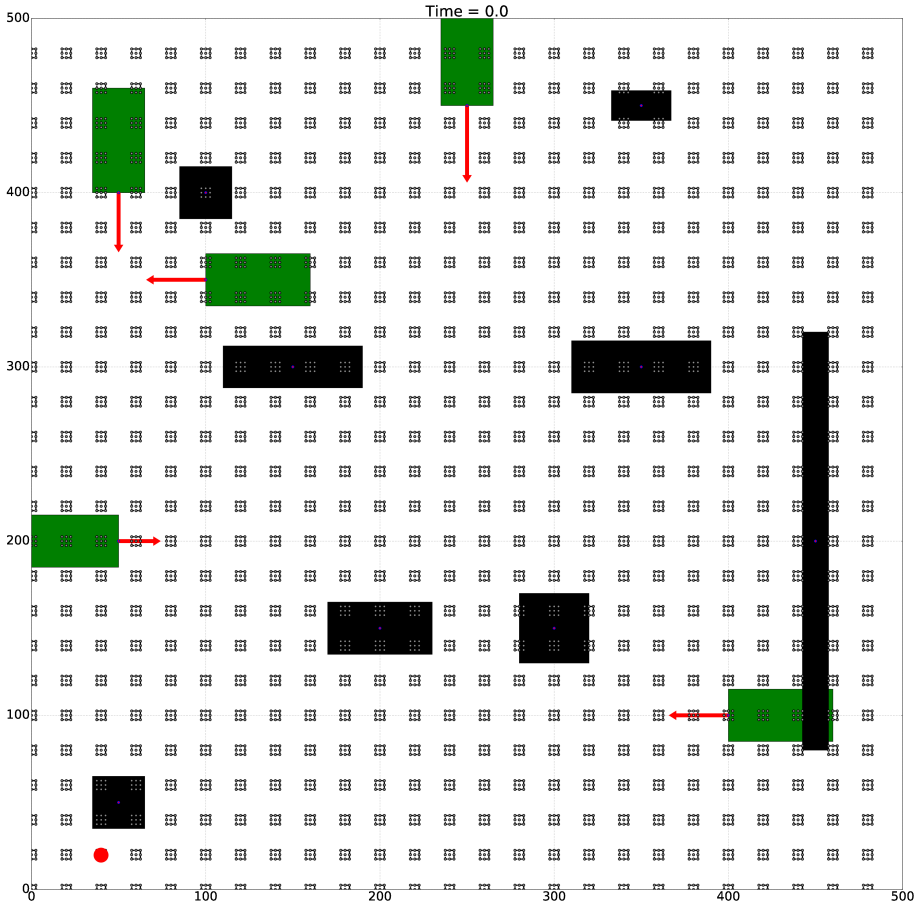

Figure 4.31: A test scenario to compare MDP and A^* solutions. The goal state set is shown as a red dot.

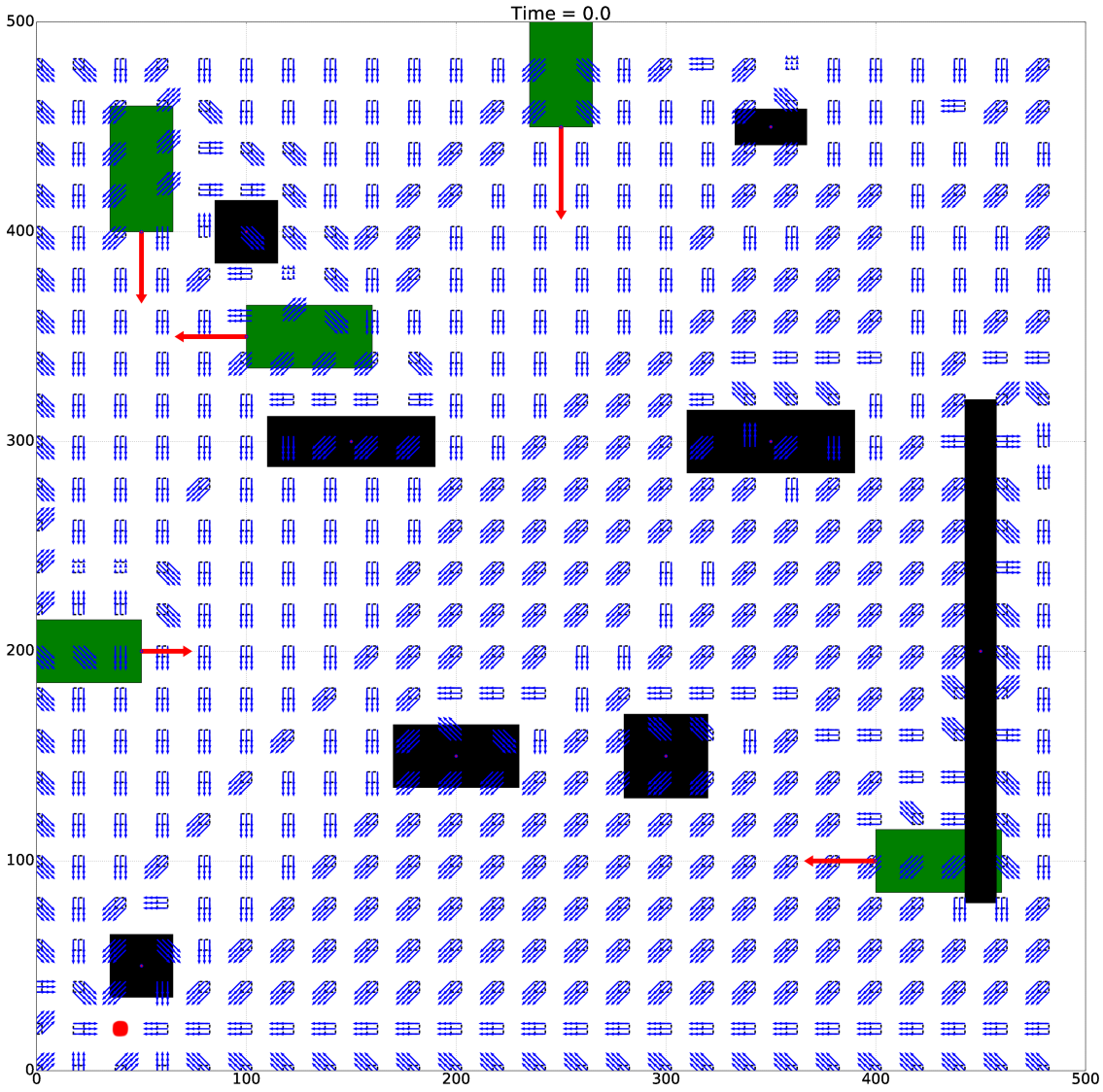

Figure 4.32: MDP policy computed for $t = 0.0$.

reward is set to $-1000\,s$. A transition is considered to be in collision if the any of the sampled points along the trajectory envelope collides with obstacles (see Figure 4.8). It took roughly 2 days to compute an optimal policy for the MDP using SDP. However, solving the DP problem (under C.R.C. assumption) using A^* produced solutions in around $103\,s$.

Figure 4.33 and Figure 4.34 show the trajectory obtained while the computed plans where executed in simulation for SDP and A^* respectively. Both plans had the final time of 149.8 and comparable collision rates. Though the trajectories varied in shape and sometimes the homotopy class, the solution cost was the same. The variation in trajectories are possibly due the extraction order in the priority queue used in the implementation of the A^* algorithm. The SDP solution had a collision rate of 3/300 and the DP solution had a collision rate of 4/500 when tighter trajectory envelopes and smaller buffer distances where used.

Although both SDP and DP formulations yield almost identical solutions, the SDP approach is computationally intensive and intractable for online trajectory planning. It is difficult to incorporate heuristic techniques to speed it up. Moreover, there is no principled way of incorporating the uncertainty in the perception of dynamic obstacles into the SDP approach. Thus, A^* search is more favorable than SDP.

4.6 Summary

Many practical operations like search and rescue operations are time-sensitive missions. Automation of such operations using Unmanned Surface Vehicles (USVs) requires automated trajectory generation. Automated trajectory generation is challenging as it requires collision avoidance with static obstacles, moving traffic vessels and the wavefields they leave behind. By ignoring the dangers of traversing these wavefields, USVs take undue risk and compromise their survivability. On the other hand, acting conservatively by circumnavigating large wavefields wastes mission time. In both cases, the timely completion of the mission is in jeopardy. USVs are susceptible to the

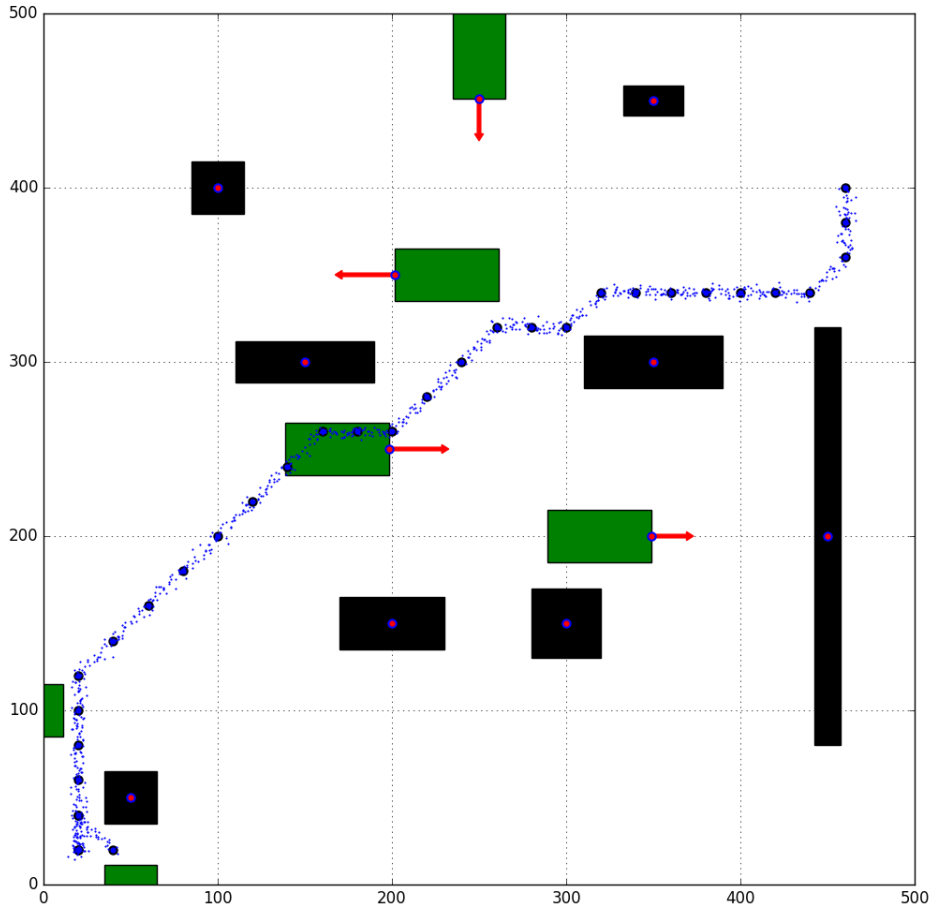

Figure 4.33: An execution run of the plan computed for the test scenario using SDP.

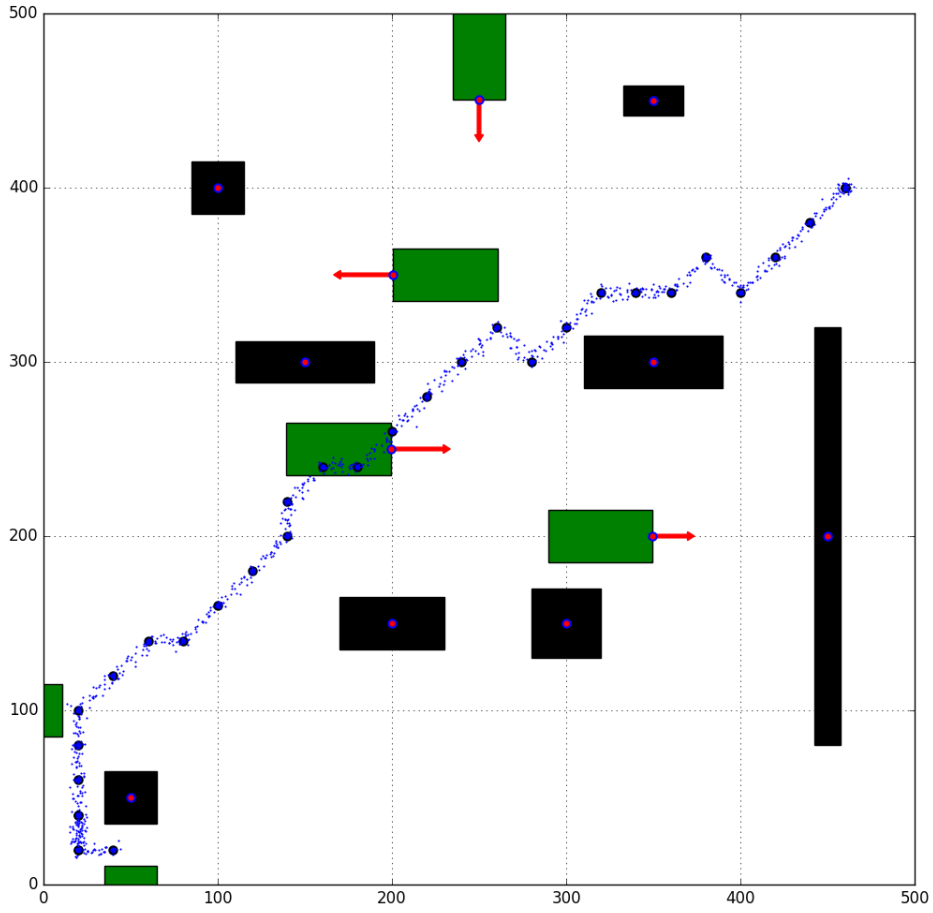

Figure 4.34: An execution run of the plan computed for the test scenario using A^*.

effects of wave disturbances resulting in motion uncertainty and at times total failure. The common practice of decoupling trajectory planning into long-term global path planning in conjunction with short-term reactive planning leads to delays, reducing mission performance. Furthermore, regions with significant wave disturbance is typically avoided completely to ensure safety. In this work, an attempt to strike a balance is made. And, a wave-aware deliberative planner is presented. It avoids collisions with traffic vessels and also opportunistically traverses wavefields generated by these vessels while minimizing the risk of failure. The planner performs a search over a 4D pose-time lattice to generate a collision-free, minimum-risk sequence of motion goals. It addresses motion uncertainty and failure risk parametrically via a pre-computed look-up table. Furthermore, speed-up techniques such as heuristics, adaptive action-set and adaptive sampling to speed up the search are also presented. Results from simulation experiments are presented. They show that the wave-aware planner:

1. is able to plan trajectories over long distances while avoiding static obstacles, traffic vessels and the waves generated by them
2. produces plans that execute faster when compared to the typical reactive planning scheme
3. can compute plans in under a minute

Results from physical experiments showing the feasibility of the proposed approach is presented as well.

Chapter 5

Trajectory Planning in High-Dimensional Configuration Spaces

5.1 Introduction

Though the motion planning problem for robots operating in a 3D workspace is PSPACE-hard [157], many automated trajectory planning methods are available when the *completeness* requirement is relaxed. For example, popular sampling-based planners such as Rapidly-exploring Random Trees (RRTs) [108] and Probabilistic Road-Maps (PRMs) [87] are *probabilistically complete*. Deterministic graph-based planners [35, 61] are *resolution complete*. On the one hand, the inherent randomness of sampling-based methods results in unpredictable solution quality; sometimes in cluttered environments, these methods may fail unless the narrow passage problem is handled using specialized techniques [141, 170]. Sampling-based motion planning algorithms are well studied methods and they have been extremely successful in solving a wide variety of motion planning problems. The inherent stochasticity of these sampling-based algorithms results in unpredictable solution quality between multiple planning queries. Although local optimization methods such as STOMP [84] and Shortcuts [70] may alleviate the problem, these methods are generally limited to the homotopy class of the seed solution. Even after local optimization, we are

likely to observe locally-optimal trajectories belonging to a variety of homotopy classes. These homotopy classes may each harbor locally-optimal trajectories of diverse quality.

In many applications, we prefer trajectories that lie not only in a predictable homotopy class but also those that are aesthetically pleasing. For instance, consider a manipulator operating in a factory floor as in Figure 5.1(a). We desire the end-effector trajectories between any two locations in (A, B, C, D, E) to be short and roughly linear motions; indeed, workspace cues are very informative partly due to the absence of clutter. While informative, workspace cues can also be unreliable at times. Consider Figure 5.1(b), where the L-shaped sanding tool is to be moved from its starting pose to the goal pose. Clearly, there is no room to slide between the gap circled red. Relying purely on workspace cues (i.e., attempting sliding) is likely to produce no solution. The only way is to move through the slot circled blue to exit the cavity. In light of this, we motivate our context-dependent bi-directional tree-search framework that can encode preferences on how much and when to exploit workspace cues.

At the core of sampling-based bi-directional tree-search methods are the following key modules: focusing module, sampling module, collision-detection module, inter-tree and intra-tree connection module. A specific implementation of a module is referred a *strategy*. These modules have seen wide interest in the planning community and significant effort has been made adapting them for various applications, resulting in a rich set of alternative strategies for each module; these strategies are often a drop-in replacement (see Section 5.2). For example, there are a set of alternative strategies for (1) handling the narrow passage problem [141, 170, 187, 34], (2) focused sampling [59, 158]. Intelligently switching between these strategies is one way to: (1) improve the resiliency of the search method, (2) encode user preferences (e.g. planning speed vs. sub-optimality trade-off). In prior work [82], a deterministic graph-search based method called *COntext DEpendent Search Strategy Switching version 1 (CODES3v1)* has been used. It integrates multiple heuristics and switches between these heuristics during planning. This work explores a sampling-based approach and applies biasing to increase trajectory quality.

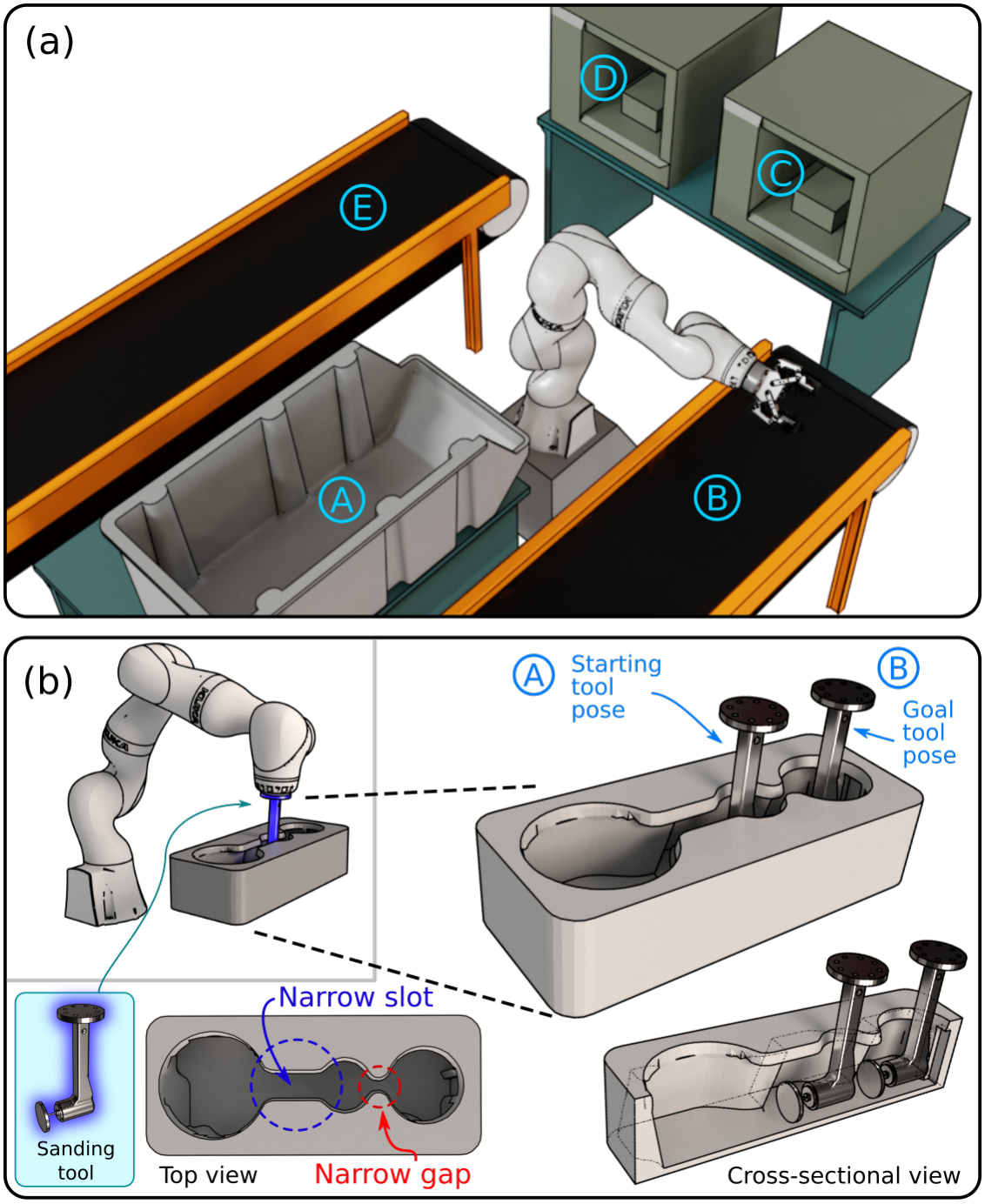

Figure 5.1: (a) Point-to-point planning between any two points in (A, B, C, D, E) can benefit from workspace guidance. (b) A complex planning problem where workspace guidance is unlikely to produce solutions. A sanding tool is to be moved from the starting pose to the goal pose. The gap circled red is too narrow. The only way out is across the slot circled blue.

In application domains such as small-volume manufacturing, industrial manipulators perform non-repetitive tasks that change frequently; manual programming of trajectories is time-consuming and impractical. In such applications, we require automated trajectory planning that quickly produces collision-free trajectories. We also require these trajectories to be of high quality as this directly impacts manufacturing throughput. High-quality trajectories are typically short and tend to minimize the execution time and energy spent.

Manipulator motion may be classified into two categories. (1) Point-to-Point Motion: Here the manipulator is given start and goal joint configuration. And, the task is to find a sequence of collision-free joint-space configurations that connect them. (2) Trajectory-Tracking Motion: Here the manipulator is given a workspace ($\mathcal{W}$-space) trajectory to be followed. And, the task is to find a corresponding configuration-space ($\mathcal{C}$-space) trajectory.

In this work, the focus is on a general point-to-point trajectory planning problem where the start and goal end-effector workspace configurations (i.e., transform frames) are given. Under-specified goal configurations frequently occur in manipulator trajectory planning where the final end-effector pose is important but the configuration of the rest of the arm is not. This problem is challenging as the workspace is often cluttered with obstacles with complex geometries.

The primary contribution is a flexible framework that enables switching between alternative strategies to yield a resilient search method amenable to machine learning. The other contribution is a simple condition for tree swapping. Its efficacy is demonstrated on a variety of challenging planning problems through *CODES3v2*, a simple instance of the proposed framework. In the rest of the chapter, *CODES3* is used to refer to *CODES3v2*, an improvement over *CODES3v1*. The switching conditions in CODES3 encode the following preference; we should first utilize focusing strategies that are likely to quickly produce high-quality solutions. Then, only in response to growing evidence that the focusing strategies are misguiding, we should resort to other seemingly less lucrative strategies.

5.2 Background

Deterministic search-based methods are used in [35, 61, 191] to solve the trajectory planning problem for manipulators. Without specialized heuristics, these methods take a longer time to solve the problem than sampling-based methods like RRT. Recent developments related to RRT and its variants are given in [49]. Each of these variants introduce novel strategies in the sampling-based tree-search algorithm. The goal of these strategies is to either reduce planning time or improve solution quality.

5.2.1 Sampling Strategies

Specifically, workspace sampling strategies have been studied in the past. Rickert et al. follow a two stage approach [158] where a RRT-based planner operating in $\mathcal{C}$-space leverages workspace cues. It works very well whenever $\mathcal{W}$-space connectivity roughly mirrors $\mathcal{C}$-space connectivity. For instance, this is true in the case of fixed and mobile manipulators operating in uncluttered workspaces.

Kiesel et al. use a focusing scheme [90] where a subspace of $\mathcal{C}$-space (i.e., essentially $\mathcal{W}$-space) is discretized into regions. By incrementally establishing connectivity between these regions in an online fashion, easy regions of the $\mathcal{C}$-space are identified. Planning effort is focused on easy regions as they are uncovered. Both methods suffer when there is a large discrepancy between the connectivity in the workspace to that of the high-dimensional configuration-space (Figure 5.1). Although techniques that provide a diverse set of trajectories as in [21, 153, 94, 214] can be employed, explicitly constructing a workspace path for the end-effector in a cluttered workspace while efficiently pruning workspace paths having infeasible $\mathcal{C}$-space counterparts is just as hard as the original problem itself. Other workspace guidance methods as in [133, 112] do not scale well for a cluttered 3D scene unless special care is taken to carefully breakdown workspace into a

computationally tractable number of disjoint polytopes. Depending heavily on any single focused-sampling strategy may be detrimental to performance.

Workspace cues have been used to guide sampling-based approaches [201, 98, 219]. These methods involve decomposition/analysis of workspace [98].

5.2.2 Node/Target Selection Strategies

Works such as [205, 173, 112] use the Jacobian pseudo-inverse to generate target configurations. Specifically, Weghe et. al. use a goal biasing scheme that drives the search towards workspace goal regions [205]. Jacobian pseudo-inverse is a potent technique for bridging the workspace and configuration space. While it helps cover large distances in the workspace, intricate configuration space maneuvers cannot easily be performed using this technique alone. When solutions pass through narrow channels in the configuration space, sampling-based approaches tend to miss them unless special care is taken to sample these channels. Methods such as [141, 170, 187] focus on these problems.

5.2.3 Connection Strategies

Variants such as RRT*, PRM* [86] guarantee almost-sure asymptotic global optimality. BIT* [59] uses a configuration space focusing schedule and provides optimality guarantees. While these methods are guaranteed to find optimal solutions eventually, being able to utilize workspace guidance when it is accurate can greatly complement their strengths [34]. In [45], a workspace RRT is grown from the goal position. The resulting tree is used to bias the growth of the configuration space tree. This method works well for mobile manipulation in uncluttered environments.

5.3 Problem Formulation

Let the workspace of the robot be denoted as $\mathcal{W} \subset \mathbb{R}^3$. Let $\mathbf{q} = \{q_1, q_2, \ldots, q_n\}$ be the joint configuration of a serial-link manipulator with n joints. Assume the geometric and kinematic models ($\mathcal{R}$) of the robot are well defined and available. Let the geometry of the manipulator at a configuration $\mathbf{q}$ be represented as a set of rigid bodies $\mathcal{M}(\mathbf{q}) \subset \mathcal{W}$. Let $\mathcal{O} \subset \mathcal{W}$ be the set of workspace obstacles. Let $\mathcal{C}$ be the configuration space of the manipulator containing all joint configurations of the manipulator. The set of joint configurations that lead to collision is denoted by $\mathcal{C}_{obs} = \{\mathbf{q} \in \mathcal{C} : \mathcal{M}(\mathbf{q}) \cap \mathcal{O} \neq \emptyset\}$. The set of joint configurations that are collision-free is $\mathcal{C}_{free} = \{\mathbf{q} \in \mathcal{C} \setminus \mathcal{C}_{obs}\}$. Let $\mathbf{T}$ be the homogeneous transformation matrix representing the pose of the End-Effector (EE) in $\mathcal{W}$. For a given joint configuration $\mathbf{q}$, $\mathbf{T}$ can be found by applying Forward Kinematics (FK). In this work, the dot notation is used to extract quantities (e.g. $\mathbf{T}.\mathbf{p}$ denotes the position component). For given a pose of the end-effector, a set of corresponding joint configurations can be found by computing Inverse Kinematics (IK). A tree node $\mathbf{n}$ consists of a configuration $\mathbf{q}$, the EE transform frame $\mathbf{T}(\mathbf{q})$. The terms nodes and configurations are used interchangeably and the distinction is made where appropriate. $\mathcal{T}_s$ denotes the tree of nodes rooted at the start node and $\mathcal{T}_g$ for the tree rooted at the goal node.

We are interested in the trajectory planning problem for a manipulator that needs to move from a starting end-effector transform frame $\mathbf{T}_s$ to a goal transform frame $\mathbf{T}_g$. Given a robot model $\mathcal{R}$, workspace obstacles $\mathcal{O}$, start and end transform frames $\mathbf{T}_s, \mathbf{T}_g$, the objective is to find a collision-free trajectory in the form of a sequence of points $Q = \{\mathbf{q}_k\}_{k=0}^{K}$ where $\text{FK}(\mathbf{q}_0) = \mathbf{T}_s, \text{FK}(\mathbf{q}_K) = \mathbf{T}_g$.

5.4 Approach

Our context-dependent bi-directional tree search algorithm (Figure 5.2) proceeds as in Algorithm 3. For each module, one of the alternatives from Table 5.1 is dynamically selected during each

Algorithm 3 Context Dependent Bi-directional Tree Search

```
1: function SOLVE(T_s, T_g)
2:     T_s ← IK(T_s), T_g ← IK(T_g)
3:     T_c ← T_s, T_o ← T_g
4:     while true do
5:         T_c, T_o ← TREESELECTION(S_e, T_s, T_g)
6:         F_s, F_t ← FOCUSSELECTION(T_c)
7:         n_c ← NODESELECTION(T_c, F_s)
8:         n_t ← TARGETSELECTION(n_c, F_t)
9:         S_e, n_e ← EXTENDSTRATEGY(n_c, n_t)
10:        T_c ← T_c ∪ {n_e}
11:        S_c ← CONNECTSTRATEGY(S_e, n_e, T_o, T_c)
12:        if REACHED == S_c then
13:            return RETRACEPATH(T_s, T_g)
```

iteration. These alternatives are outlined and some insights into when they might be useful is provided.

5.4.1 Tree Selection

Tree selection refers to picking a *current tree* $\mathcal{T}_c$ from the two choices – $\mathcal{T}_s$ and $\mathcal{T}_g$. After a *current tree* is picked, the *other tree* $\mathcal{T}_o$ refers to the tree that was not picked. In a broad sense, tree selection is a type of focusing scheme where the effort is focused in growing the *current tree*. In many cases, unconditionally swapping trees before every iteration works well (TS_A). However, it is observed that this may not always be the best strategy. Suppose the goal tree $\mathcal{T}_g$ is harder to grow (e.g. the goal configuration has a tool in a cavity) and the start tree $\mathcal{T}_s$ is easy to grow. It is crucial to focus more effort growing $\mathcal{T}_g$ than $\mathcal{T}_s$. Adding nodes to $\mathcal{T}_s$ may not alleviate the problem of exiting the cavity as much as adding useful nodes in $\mathcal{T}_g$ would.

A simple strategy is to switch to the other tree only if a valid (i.e., $S_e \neq TRAPPED$) new node was added to the current tree in the previous iteration (TS_B). While TS_B allows progress on both trees, greedy connection strategies may introduce significant node-count difference (i.e.,

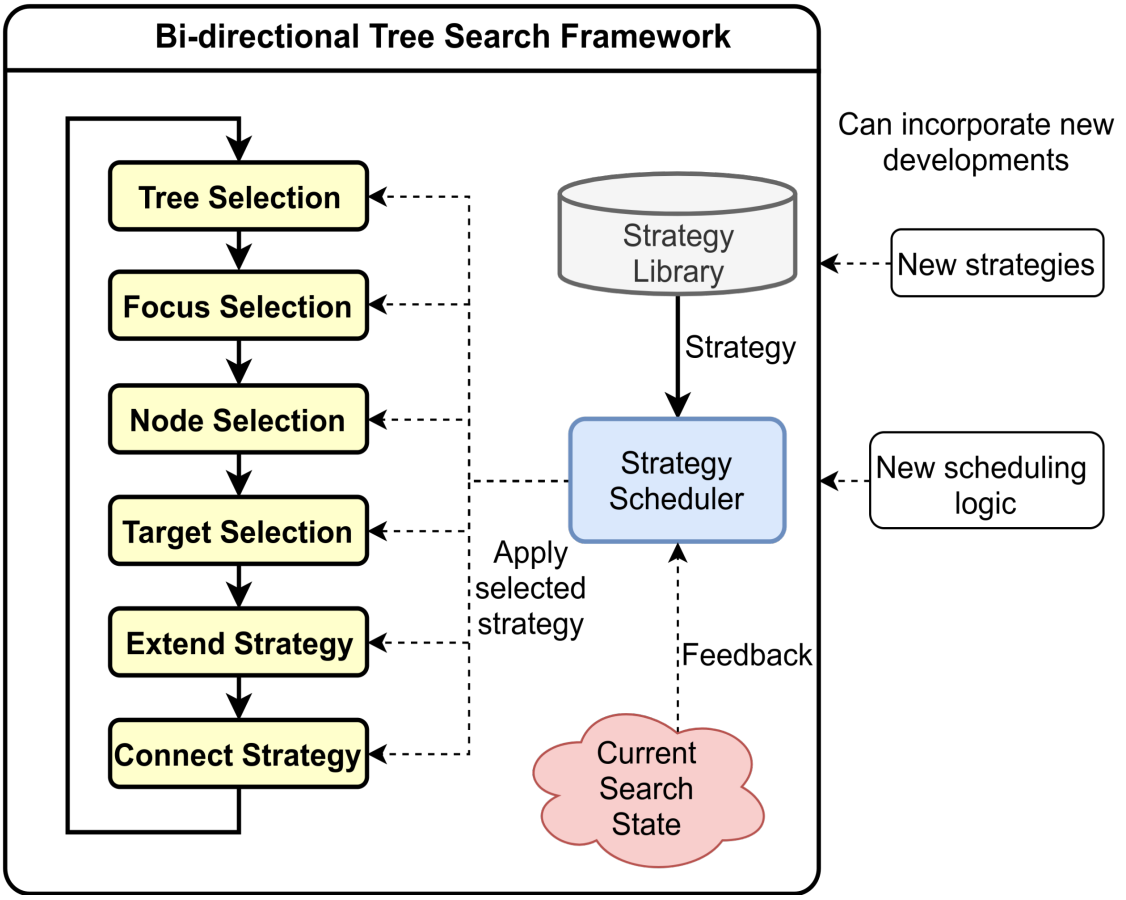

Figure 5.2: Our framework dynamically switches strategies selected from a library of strategies based on feedback from the current search state. It can incorporate new strategies and scheduling logic for application specific needs

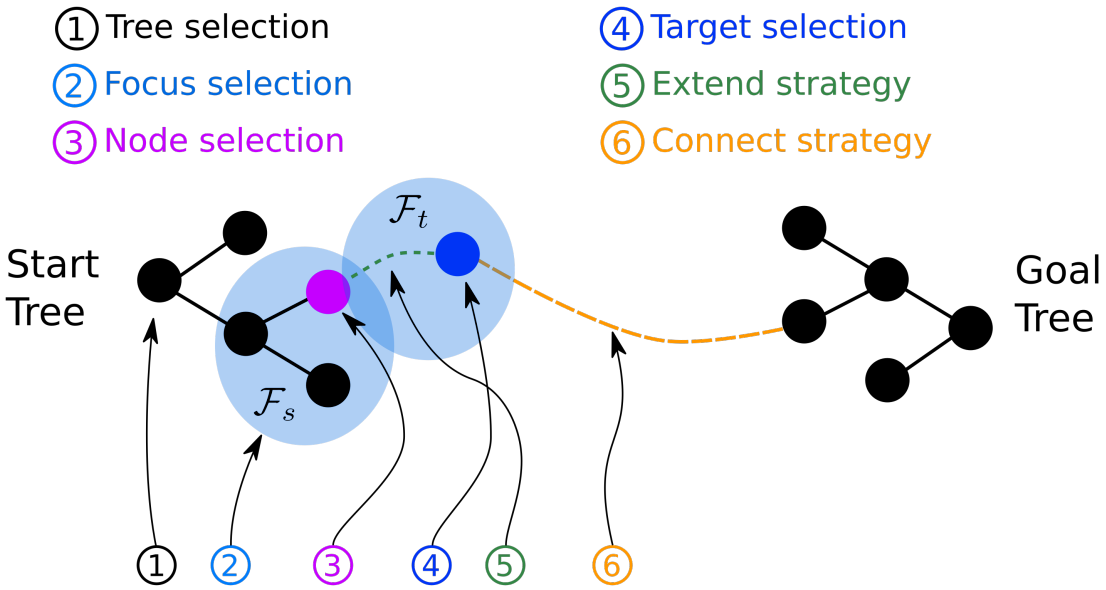

Figure 5.3: An illustration of the six primitive steps in realizing the bi-directional tree search method.

$\text{abs}(|\mathcal{T}_s| - |\mathcal{T}_g|)$) as intermediate nodes are laid down between the trees. Another simple strategy is to switch only when the normalized node-count difference

$$\frac{\text{abs}(|\mathcal{T}_s| - |\mathcal{T}_g|)}{|\mathcal{T}_s| + |\mathcal{T}_g|} \leq \omega$$

between the two trees is less than a threshold ω (TS_C). This has the effect of roughly equalizing node-count on both trees resulting in a balanced growth (see Figure 5.4). Such a balanced growth results in the drastic reduction of failure rate (see Section 5.5). For TS_C to work, during the extend and connect steps, only the chosen tree must be extended and the other tree must be left untouched. This ensures that the node-count balancing feedback is negative.

5.4.2 Focus Region Selection

Having chosen a tree, focus regions are picked next. Focus regions are regions of the search space where planning effort is expended. There are two types of focus regions: $\mathcal{W}$-space focus regions

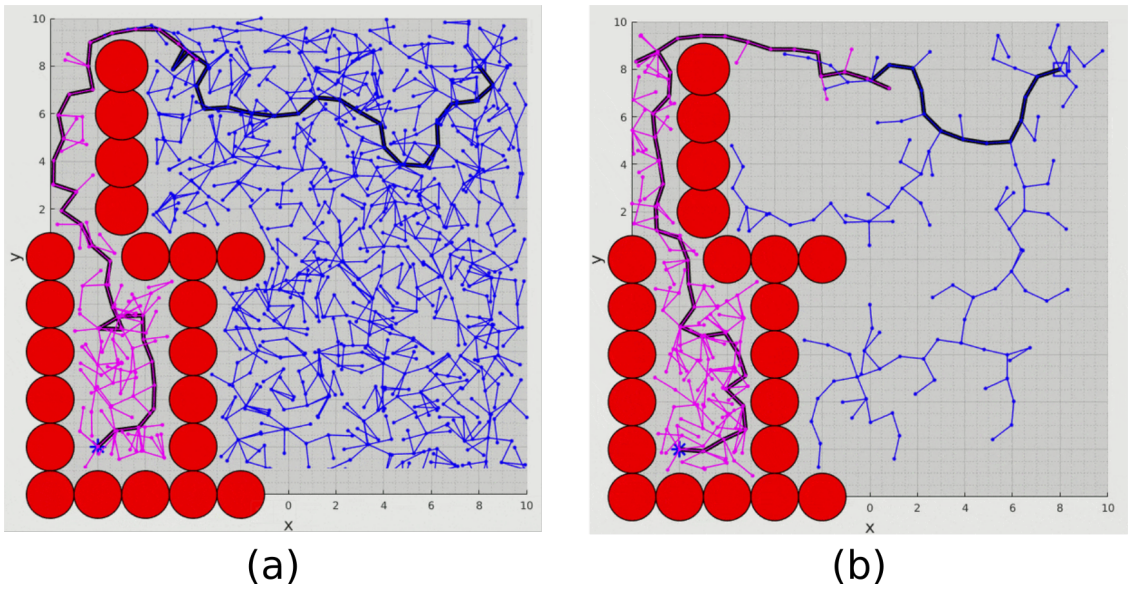

Figure 5.4: (a) An illustration of bi-directional tree search using TS_A, (b) An illustration of bi-directional tree search using TS_C. We observe that the balanced growth of both trees results in a solution without sampling a lot of unnecessary (blue) nodes.

and $\mathcal{C}$-space focus regions. Methods such as [21, 153, 94, 214] can be used to generate $\mathcal{W}$-space focus regions. Domain specific workspace focusing can also be triggered upon certain conditions (e.g. moving along confined workspace slits/slots as in scenario S13 in Figure 5.10). Similarly, $\mathcal{C}$-space focus regions can be generated through methods outlined in [59, 187, 154, 97].

For either type of focusing, two subsets can be defined: node-focus region $\mathcal{F}_s$ and target-focus-region $\mathcal{F}_t$ in their respective spaces. Node-focus region is a region used for selecting nodes, while target-focus region is used for selecting targets (see Figure 5.3). Targets are points in $\mathcal{C}$ or $\mathcal{W}$ towards which the chosen tree is attempted to extend towards.

In this work, $\mathcal{C}$-space focusing was not used (i.e., for $\mathcal{C}$-space sampling, $\mathcal{F}_s = \mathcal{F}_t = \mathcal{C}$). However, a $\mathcal{W}$-space focusing method given in [158] is used. The wavefront propagation method (Algorithm 4) is used to find a workspace path for a key point located at the end-effector flange.

The wavefront algorithm greedily grows a search tree from the start point $\mathbf{p}_s$ towards the goal point $\mathbf{p}_g$. We seed with the first node containing a workspace ball centered at $\mathbf{p}_s$. The radius of this ball is set by querying the workspace $\mathcal{W}$ for the distance to the closest obstacle from $\mathbf{p}_s$.

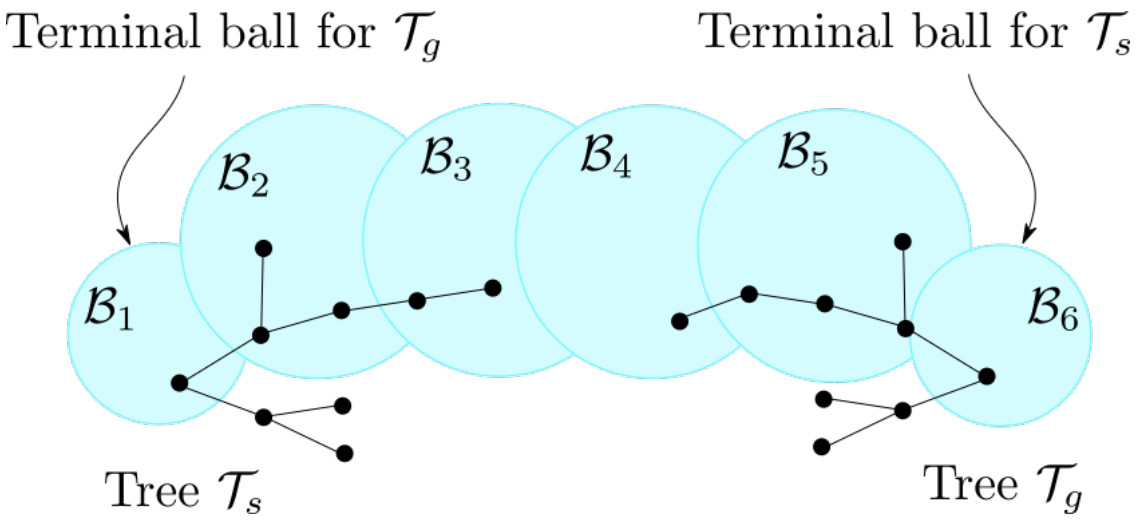

Figure 5.5: An illustration of free-space balls in $\mathcal{W}$ ($K = 6$). Tree $\mathcal{T}_s$ has progressed upto ball $\mathcal{B}_3$. Tree $\mathcal{T}_g$ has progressed upto ball $\mathcal{B}_4$. Note that for illustration purposes, both trees have been projected onto $\mathcal{W}$ based on the EE positions of the node.

This node is then put into a priority queue Q which sorts nodes in the order of least Euclidean distance to the goal position. Whenever a node is popped from this queue, we check if the ball contains the goal point. If it contains the goal point, we terminate the search and retrace the search tree to return a sequence of workspace-balls. If it does not contain the goal point, the surface of the ball is uniformly sampled. And, each sample $\mathbf{p}_i$ that is already contained by the existing balls in V is ignored. The remaining samples are inserted into the queue by performing a distance query. This process repeats until a sequence of workspace-balls $\mathcal{S}$ is returned. This workspace path denotes the sequence of free-space balls $\{\mathcal{B}_k\} \subset \mathcal{W}$ (starting at $\mathbf{p}_s$ leading to the workspace goal $\mathbf{p}_g$) through which the end-effector can be dragged along (Figure 5.5).

The progress of the search tree is monitored along this ball path using a *progress index* $m \in [1, K]$ where K is the number of balls in the path. For a particular tree, say $\mathcal{T}_s$, a progress index $m = 3$, indicates that there exists a configuration $\mathbf{q}'$ in this tree whose EE workspace position has progressed 3 balls from the initial position $\mathbf{p}_s$ towards the final ball (i.e., $\mathbf{T}(\mathbf{q}').\mathbf{p} \in \mathcal{B}_3$). With a slight abuse of notation, we say that $\mathbf{q}' \in \mathcal{B}_3$. Since our framework is bi-directional, there are two progress indices, one for each tree. Initially, $m_s = 1$ and $m_g = K$. The intent is to increment

Algorithm 4 Wavefront Algorithm

```
1: function WorkspaceBallPath(q_s, q_g)
2:     p_s ← ForwardKinematics(q_s)
3:     p_g ← ForwardKinematics(q_g)
4:     V ← ∅
5:     E ← ∅
6:     Q ← ∅
7:     r_s ← DistanceQuery(p_s)
8:     Insert(Q, s, ‖p_g − p_s‖ − r_s)
9:     while true do
10:        s ≡ (p_center, r, s_parent) ← Pop(Q)
11:        V ← V ∪ {s}
12:        E ← E ∪ {(s_parent, s)}
13:        if ‖p_g − p_center‖ ≤ r then
14:            S ← RetraceTree(s)
15:            return S
16:        P ← UniformSamplesOnSphere(s)
17:        for p_i ∈ P do
18:            for s_i = (p_center,i, r_i, s_parent) ∈ V do
19:                if ‖p_i − p_center,i‖ > r_i then
20:                    r' ← DistanceQuery(p_i)
21:                    s' ← (p_i, r', s)
22:                    Insert(Q, s', s)
```

m_s till $m_s = K$ and decrement m_g till $m_g = 1$ in the hope that both trees will progress towards their respective *terminal balls* (Figure 5.5). For $\mathcal{T}_s$ and $\mathcal{T}_g$, the *terminal balls* are $\mathcal{B}_K$ and $\mathcal{B}_1$ respectively. To this intent, a biased coin is tossed to choose between the following assignments:

$$(\mathcal{F}_s, \mathcal{F}_t) \leftarrow (\mathcal{B}_m, \mathcal{B}_{m+d})$$

on heads and

$$(\mathcal{F}_s, \mathcal{F}_t) \leftarrow (\mathcal{W}, \mathcal{W})$$

on tails where $d = 1$ when $\mathcal{T}_c = \mathcal{T}_s$ and $d = -1$ when $\mathcal{T}_c = \mathcal{T}_g$. The bias for heads is denoted by ζ.

5.4.3 Node Selection

Node selection refers to picking a node $\mathbf{n}_s$ from the current tree $\mathcal{T}_c$ using the focus region $\mathcal{F}_s$. Much like RRT-Connect (RRTC) [95], strategy NS_A picks a random configuration $\mathbf{q}_r \in \mathcal{F}_s$ and finds the

closest node to $\mathbf{q}_r$ in $\mathcal{T}_c$ (NS_A). This selection method has the Voronoi-bias property that allows for rapid exploration (i.e., nodes of $\mathcal{T}_c$ are selected with frequency proportional to the Voronoi-region-volume of each node). However, rapid exploration happens only when a suitable distance-metric is used [109]. In many applications, it is often beneficial to make a move in workspace by applying Voronoi-bias in workspace [173]. This is done by picking a random reachable workspace frame $\mathbf{T}_r \in \mathcal{F}_s$ and finding the closest node in $\mathcal{T}_c$ (i.e., $\underset{\mathbf{n} \in \mathcal{T}_c}{\operatorname{argmin}}\ d(\mathbf{n}.\mathbf{T} - \mathbf{T}_r)$) ($NS_B$). Nodes can also be selected based on criteria such as a cost estimate as in graph-search methods like A* (NS_C).

5.4.4 Target Selection

Target selection refers to picking a target in the vicinity of the chosen node $\mathbf{n}_c$ in order to make a local connection. This target could be a point in either $\mathcal{C}$ or $\mathcal{W}$. TaS_A produces a target configuration $\mathbf{q}_t = \textsc{Interpolate}(\mathbf{q}_c, \mathbf{q}_r, \epsilon_q)$ where $\mathbf{q}_r \in \mathcal{F}_t$ and ϵ_q is the step-size. TaS_B produces a target transform frame $\mathbf{T}_t = \textsc{Interpolate}(\mathbf{T}_c, \mathbf{T}_r, \epsilon_p)$ where $\mathbf{T}_r \in \mathcal{F}_t$ and ϵ_p specifies the step-size. While TaS_A and TaS_B receive target points from the node selection module, the target point could be chosen independently of node selection. In TaS_C, n directions from the chosen node can be sampled and a direction with least cost can be picked.

In difficult areas (e.g., narrow passages and entrance/exit of narrow passages), extending a tree can become difficult as many of the sampled targets are in collision. In such cases, building an approximate local analytical $\mathcal{C}$-space model ($\widehat{\mathcal{C}}_{obs}$) can be helpful. This analytical model ($\widehat{\mathcal{C}}_{obs}$) allows for quick inference and it can be used for selection promising targets (TaS_D). Since this building this model takes a non-negligible amount of time, we judiciously build these local models only for regions of $\mathcal{C}$ that are difficult. Difficult regions of $\mathcal{C}$-space can be identified in several ways. After every extend step, we log the status of the extend operation into the source node (n_s). For example, suppose that an extend step is performed. Then, $n_s.ea \leftarrow n_s.ea + 1$ (extend attempts). If the return code for an extend step is TRAPPED, then $n_s.ef \leftarrow n_s.ef + 1$. A node

is considered to be in a difficult region when $n_s.ea > N_a$ and $n_s.ef/n_s.ea > \sigma_d$ and a local model is built (i.e., $n.\widehat{\mathcal{C}}_{obs}$). N_a is a threshold that waits for enough number of attempts (~ 20) and σ_d is also a threshold (~ 0.8).

Suppose that $\mathbf{Q} = \{\mathbf{q}_m \in \mathcal{C}\}_{m=1}^{M} \in \mathbb{R}^{M \times n}$ a sequence of configuration points. Suppose that $\mathbf{L} = \{l_m\}_{m=1}^{M}$ where $l_m = \mathbb{1}_{\mathcal{C}_{obs}}(\mathbf{q}_m) \in \mathbb{R}^{M \times 1}$ and $\mathbb{1}$ is the indicator function. We can obtain a local approximation of the $\mathcal{C}$-space by fitting a RBF model. In order to reduce the computational burden when M is large, we can first perform clustering on $\mathbf{X}$ alone (disregarding $\mathbf{L}$). Suppose that after clustering $\mathbf{Q}$ using k-means algorithm with $k = K$, we have $\mathbf{C} = \{\mu_k\}_{k=1}^{K} \in \mathbb{R}^{K \times n}$ where $\mu_{\mathbf{k}} \in \mathcal{C}$ are cluster centers and cluster variance is $\mathbf{\Sigma} = \{\sigma_k^2\}_{k=1}^{K} \in \mathbb{R}^{K \times 1}$. Suppose that we represent the RBF basis set as

$$\mathbf{\Psi} = \{\psi_k\}_{k=1}^{K} \tag{5.1}$$

where

$$\psi_k(q) = \exp(-\beta_k \|\mathbf{q} - \mu_{\mathbf{k}}\|^2) \tag{5.2}$$

and

$$\beta_k \equiv 1/(2\sigma_k^2) \tag{5.3}$$

We wish to approximate $\mathcal{C}_{obs}$ using the following RBF model.

$$V(\mathbf{q}, \mathbf{\Theta}) \equiv \sum_{k=1}^{K} \theta_k \psi_k(\mathbf{q}) \approx \mathbb{1}_{\mathcal{C}_{obs}}(\mathbf{q}) \tag{5.4}$$

The $\mathcal{C}$-space approximation problem can be formulated as follows.

$$\mathbf{A\Theta} = \mathbf{L} \tag{5.5}$$

where

$$\mathbf{A} = \begin{bmatrix} \psi_1(\mathbf{q}_1) & \psi_2(\mathbf{q}_1) & \cdots & \psi_K(\mathbf{q}_1) \\ \psi_1(\mathbf{q}_2) & \psi_2(\mathbf{q}_2) & \cdots & \psi_K(\mathbf{q}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \psi_1(\mathbf{q}_M) & \psi_2(\mathbf{q}_M) & \cdots & \psi_K(\mathbf{q}_M) \end{bmatrix}$$

, $\mathbf{q}_m \in \mathbf{Q}$ and $\mathbf{\Theta} \equiv \begin{bmatrix} \theta_1 & \theta_2 & \cdots & \theta_K \end{bmatrix}^T$.

The approximation of the $\mathcal{C}$-space can be determined by finding a least-squares solution

$$\mathbf{\Theta}_{LS} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{L} \tag{5.6}$$

One can also compute the gradient of V as

$$\nabla_{\mathbf{q}} V(\mathbf{q}) = -\sum_{k=1}^{K} (2\theta_k \beta_k \psi_k(\mathbf{q}))(\mathbf{q} - \mu_k) \tag{5.7}$$

To build a local $\mathcal{C}$-space model ($\widehat{\mathcal{C}}_{obs}(\mathbf{q}_0)$) at a point $\mathbf{q}_0$, we can sample a distribution of points $\mathbf{Q}_0$ (a domain around $\mathbf{q}_0$) and their corresponding labels $\mathbf{L}_0$. Then, we can construct a local model as $\widehat{\mathcal{C}}_{obs}(\mathbf{q}_0) \equiv (\mathbf{q}_0, \mathbf{Q}_0, \mathbf{L}_0, \mathbf{\Theta_{LS}})$ and store in a node (i.e., $n.\widehat{\mathcal{C}}_{obs} = \widehat{\mathcal{C}}_{obs}(\mathbf{q}_0)$). Whenever a selected node (n_s) falls in the domain of an existing local model $\widehat{\mathcal{C}}_{obs,0}$, we set $n_s.\widehat{\mathcal{C}}_{obs} = \widehat{\mathcal{C}}_{obs,0}$. Given that we start at $n_s.\mathbf{q} \equiv \mathbf{q}_s$, we want to find suitable directions to extend and avoid obstacles. An easy way to accomplish this is to move orthogonal to the gradient of V as shown in Figure 5.6. Suppose that $\hat{q}_r$ is a random unit vector ($\hat{v}$ denotes normalization of v). An orthogonal vector can be constructed as

$$q_r = \hat{q}_r - (\hat{q}_r^T \nabla\hat{V}(\mathbf{q}_s) + \xi)\nabla\hat{V}(\mathbf{q}_s) \tag{5.8}$$

Thus, the target can be computed as

$$\mathbf{q}_t = \mathbf{q}_s + \epsilon\, \mathbf{q}_r \tag{5.9}$$

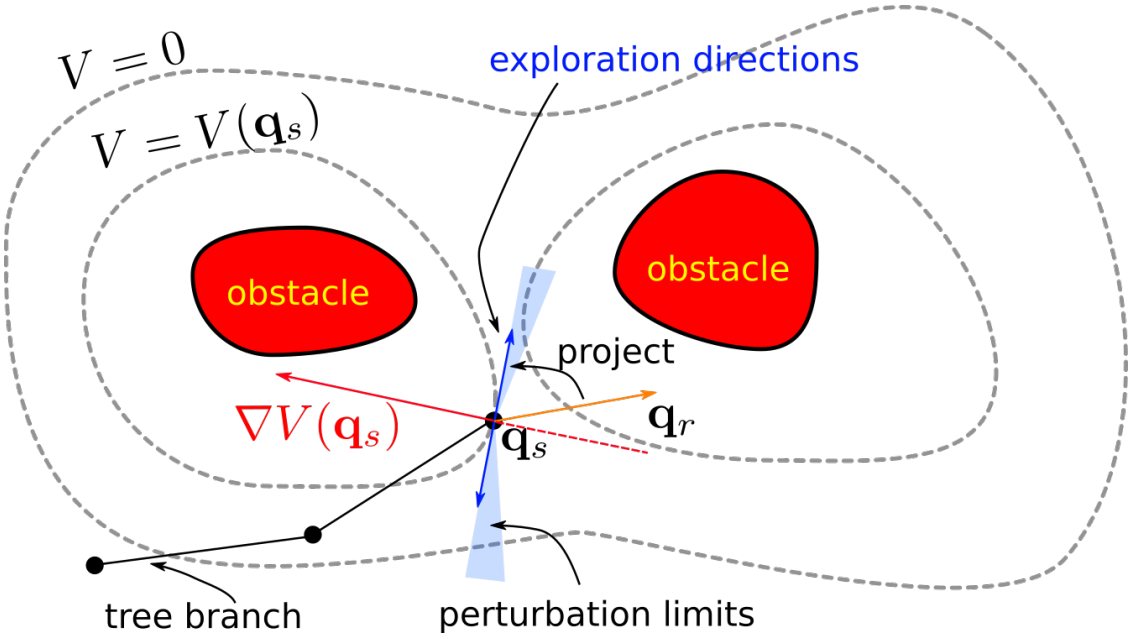

Figure 5.6: An illustration of how a local model ($dim(\mathcal{C}) = 2$) is utilized for exploration. Directions roughly orthogonal to the gradient vector are promising directions.

Without a large number of samples, this model is unlikely to be perfect and thus, we cannot rely on it exclusively. We handle this drawback in two simple ways. First, we add a slight perturbation $\xi \sim \mathcal{U}(0, 0.1)$ that helps avoid obstacles a little further. Second, every time a model is created, we compute the accuracy (p_r) of the model. The outcome of a biased (p_r) coin-toss is used to decide if model-based target selection is performed. The more accurate the model is, the more likely model-based target selection is used.

5.4.5 Extend Strategy

Extend strategy refers to how the selected node $\mathbf{n}_c$ and the selected target are connected. There are two types of targets: $\mathbf{q}_t$ ($\mathcal{C}$-space target) and $\mathbf{T}_t$ ($\mathcal{W}$-space target). If the selected target is $\mathbf{q}_t$, then ES_A and ES_C apply. ES_A performs a linear interpolation between $\mathbf{q}_c \equiv \mathbf{n}_c.\mathbf{q}$ and $\mathbf{q}_t$; it checks for collisions along the way. Judiciously applied non-linear connections [34, 91] in $\mathcal{C}$-space (ES_C) have the potential to improve performance significantly. In this work, non-linear connections are not included.

ES_B uses the geometric Jacobian ($\mathbf{J}_g$) to drive $\mathbf{q}_c$ to $\mathbf{T}_t$ using the pseudo-inverse method [30].

$$\delta\mathbf{p} = \mathbf{T}_t.\mathbf{p} - \mathbf{T}_c.\mathbf{p}.$$

$$\delta\mathbf{o} = \mathbf{R}_c * (\textsc{Quat}(\mathbf{R}_c^T\mathbf{R}_t).xyz)$$

where $\textsc{Quat}(\mathbf{R})$ is the quaternion representation of $\mathbf{R}$ and $\mathbf{R}_c$ is the rotation component of $\mathbf{T}_x$. Let $\beta(\mathbf{v}, \mu) = \mu / \max(\|\mathbf{v}\|_\infty, \mu)$.

$$\Delta\mathbf{x} = [\beta(\delta\mathbf{p}, \mu_p)\delta\mathbf{p}\,; \beta(\delta\mathbf{o}, \mu_o)\delta\mathbf{o}] \tag{5.10}$$

$$\delta\mathbf{q} = (\mathbf{J}_g^T\mathbf{J}_g + \gamma\mathbf{I})^{-1}\mathbf{J}_g^T\Delta\mathbf{x} \tag{5.11}$$

$$\Delta\mathbf{q} = \beta(\delta\mathbf{q}, \mu_q)\delta\mathbf{q} \tag{5.12}$$

$$\mathbf{q}_n \leftarrow \mathbf{q}_n + \Delta\mathbf{q} \tag{5.13}$$

The adaptive scaling function β allows finer control over step-size and increased stability at the cost of more iterations to reach $\mathbf{T}_t$.

A status flag S_e and a new node $\mathbf{n}_e$ is returned by all extend strategies; $S_e \leftarrow TRAPPED$ when a collision-free move cannot happen at $\mathbf{q}_c$, $S_e \leftarrow ADVANCED$ when a collision-free move can be made and $S_e \leftarrow REACHED$ when the target is reached without collisions. $\mathbf{n}_e$ contains $\mathbf{q}_e$, the resultant collision-free configuration after extending $\mathbf{q}_c$.

If the target was selected using the workspace balls, we follow [158] to update progress indices; suppose $S_e \neq TRAPPED$ and the extension produced $\mathbf{q}_e$ in the current tree $\mathcal{T}_c$. We start scanning from the terminal ball towards $\mathcal{B}_m$ and identify the first ball $\mathcal{B}_p$ such that $\mathbf{q}_e \in \mathcal{B}_p$. Then, we perform the update $m_c = p$. Suppose $S_e = TRAPPED$, we follow the backtracking technique in [158] where the current ball is inflated by a small factor. On recurring entrapment,

the progress index is regressed so that a more comprehensive effort can be made from a previous ball.

5.4.6 Connection Strategy

Connection strategy refers how a connection can be made between the start and the goal trees. Specifically, it refers to how a node (*source* node) from $\mathcal{T}_c$ is connected to another node (*destination* node) in $\mathcal{T}_o$.

One way to pick the *source* node is to pick $\mathbf{n}_e \in \mathcal{T}_c$ whenever the status flag $S_e \neq TRAPPED$. The *destination* node in $\mathcal{T}_o$ can be picked in many ways. The closest node to $\mathbf{n}_e.\mathbf{q}$ in $\mathcal{T}_o$ can be picked as the destination node; the source and destination nodes can be connected using linear-interpolation in $\mathcal{C}$-space (CS_A). Due to the bi-directional search scheme, some nodes of $\mathcal{T}_c$ and $\mathcal{T}_o$ often intertwine in $\mathcal{W}$-space; however, in $\mathcal{C}$-space, these nodes are relatively distant. Such nodes in $\mathbf{T}_o$ which are in proximity to $\mathbf{n}_e$ in $\mathcal{W}$-space are good candidates as destination nodes as they have the potential to connect the two trees. Thus, a candidate set can be generated by filtering nodes of $\mathcal{T}_o$ that are within a workspace-ball of radius r_c centered at $\mathbf{n}_e.\mathbf{T}.\mathbf{p}$. A random candidate is picked from this set as the destination node and attempt connect to it from the source node (CS_B).

Connection is performed by repeatedly using the extend strategy ES_A until the destination node is reached or collision is detected. As noted in [95], the greediness of the connect strategy has an impact on performance; It is observed that limiting the number of times the extend strategy is repeatedly called improves planning time and solution cost. Furthermore, inhibiting the addition of intermediate nodes (i.e., nodes generated while attempting to connect the source and destination, excluding the end-points) has the effect of decreasing planning time at the expense of decreasing trajectory quality. A maximum limit of 5 extends per connect was used. And, intermediate nodes were not added. Occasionally, with a very low frequency, it is beneficial to select a random node from $\mathcal{T}_o$ (CS_C). This helps when the distance-metric is not accurate.

In RRT-Connect [95], a newly generated node in the chosen tree ($\mathcal{T}_c$) is connected greedily along a straight-line to the nearest node in the other tree ($\mathcal{T}_o$). While it is simple and works well in many cases, the connection strategy can be made adaptive to information gathered about the $\mathcal{C}$-space so far. The idea is to make connection effort not just along a straight-line but a general region of $\mathcal{C}$-space (CS_D). In this work, CS_D is also referred to as the Connection Biasing (CB) strategy. Consider the scenario in Figure 5.7 where n_a and n_c (closest on the other tree) have

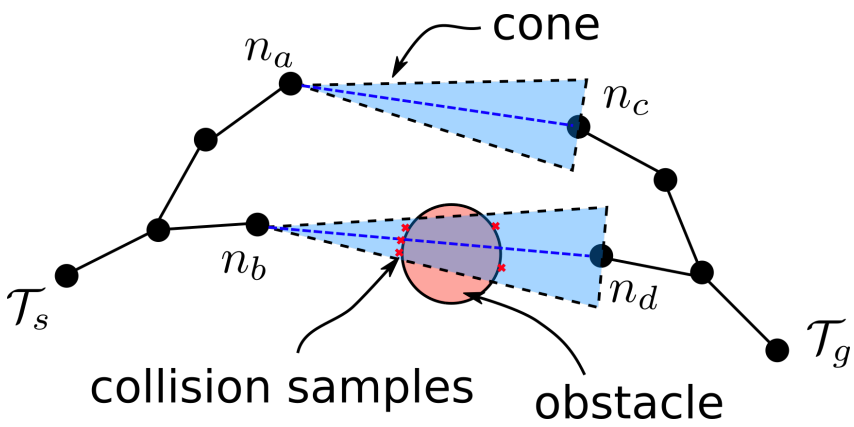

Figure 5.7: Connection biasing

an unobstructed straight-line path between them while n_b and n_d have an obstruction. Initially, this information is unknown; whenever, n_a (or n_b) is chosen, attempting connection directly to n_c (or n_d) is desired. However, as time progresses, collision samples are observed during attempts to connect. These samples indicate difficulty in connection. A cone c_α of a certain angle α between $\mathbf{q}_1$ and $\mathbf{q}_2$ can be defined as $c_(\mathbf{q}_1, \mathbf{q}_2, \alpha) = \{\mathbf{q} \in \mathcal{C} | \|\mathbf{q}_{par}\| \leq d, \|\mathbf{q}_{perp}\| \leq d \tan(\alpha), \mathbf{q}_{par} \equiv ((\mathbf{q} - \mathbf{q}_1)^T \mathbf{q}_{dir})\mathbf{q}_{dir}, \mathbf{q}_{perp} = (\mathbf{q} - \mathbf{q}_1) - \mathbf{q}_{par}\}$ where $\mathbf{q}_{perp}$ and $\mathbf{q}_{par}$ denote the perpendicular and parallel components along the direction $\mathbf{q}_{dir} \equiv (\mathbf{q}_2 - \mathbf{q}_1)/\|\mathbf{q}_2 - \mathbf{q}_1\|$. Suppose that a node n_c is selected and the closest node on the other tree is n_o. A cone $c(\mathbf{q}_c, \mathbf{q}_o, \alpha)$, $\alpha = 10\,\text{deg}$. Within the cone c, the ratio r of collision samples to the total number of observed samples can be computed. This can be done efficiently by culling operations utilizing R-trees. Here observed samples consists of the collision samples and nodes (of both trees). Based on this ratio r, we can make a decision on what to do next. The decisions to be made are (1) should a connection attempt be made at

all?, (2) if we are attempting a connection, which direction to move towards? We use the following rule. If $r < 0.2$, attempt connection directly towards $\mathbf{q}_o$. Else if $r < 0.5$, make 5 extend steps consecutively along a random ray passing through the apex of the cone (i.e., $\mathbf{q}_c$) and a random point on the cap of the cone. If $r > 0.5$, skip the connection step and move on to tree selection.

Table 5.1: Summary of Strategies

Module	**Strategy**	**Description** (more details in Section 5.4)
Tree Selection	TS_A	Swap trees
	TS_B	Swap trees only on success
	TS_C	Swap trees on low node count difference
Focus Selection	FS_A	Use configuration space focusing
	FS_B	Use workspace focusing
Node Selection	NS_A	Pick closest node in $\mathcal{F}_s$ to a random $\mathbf{q}_r \in \mathcal{F}_t \subset \mathcal{C}$
	NS_B	Pick closest node in $\mathcal{F}_s$ to a random workspace frame $\mathbf{T}_r \in \mathcal{F}_t \subset \mathcal{W}$
	NS_C	Pick least f-cost node from priority queue
Target Selection	TaS_A	Pick target ϵ_q away in direction of $\mathbf{q}_r - \mathbf{q}_c$
	TaS_B	Pick target ϵ_p away in direction of $\mathbf{T}_r.\mathbf{p} - \mathbf{T}(\mathbf{n}_s.\mathbf{q}).\mathbf{p}$
	TaS_C	Sample n directions and pick direction with least f-cost
	TaS_D	Use an approximate local $\mathcal{C}$-space model to quickly identify promising targets (RBF)
Extend Strategy	ES_A	Linear interpolation in $\mathcal{C}$-space
	ES_B	Linear interpolation in $\mathcal{W}$-space
	ES_C	Non-linear interpolation in $\mathcal{C}$-space
Connect Strategy	CS_A	Connect new node to nearest (in $\mathcal{C}$) tree node in other tree
	CS_B	Connect new node to nearest (in $\mathcal{W}$) tree node in other tree
	CS_C	Connect new node to random tree node in other tree
	CS_D	Connect new node to nearest node in other tree based *cones* (Connection Biasing)

5.4.7 Scheduling of Strategies

The choice of strategies across each of these modules alters the behavior of the search method. For each module, one of many alternative strategies can be chosen. Let $\{\lambda_k\}$ represent the *strategy vector*, a stochastic vector containing the probabilities of selecting a strategy for module k. For example, $\lambda_{TS} = [TS_A : 0,\ TS_B : 0.2,\ TS_C : 0.8]$ indicates the selection of TS_B and TS_C with a probability of 0.2 and 0.8 respectively. Certain strategies are not compatible with each other. In particular, selecting a workspace target (i.e., strategy TS_B), limits options in the selection of the extend strategy. E_B is the only strategy that can be selected as the extend strategy as there is no joint configuration as a target. In such cases, the probability values are zeroed out to conform to the constraints. A *strategy sequence* Λ can be defined as a tuple of *strategy vectors* $\Lambda = (\lambda_{TS}, \lambda_{FS}, \lambda_{NS}, \lambda_{TaS}, \lambda_{ES}, \lambda_{CS})$.

Conforming to such constraints, a sequence of strategies could be *statically* determined before the search starts and kept constant during the search. An approximation of RRTC [95] (A-RRTC) and an approximation of a bi-directional EET (A-EETC) [158] can be represented as *static* tuples as in Table 5.2 where

$$\Lambda_R \equiv (TS_A : 1, FS_A : 1, NS_A : 1, TaS_A : 1, ES_A : 1, CS_A : 1)$$

and

$$\Lambda_E \equiv (TS_A : 1, FS_B : 1, NS_B : 1, TaS_B : 1, ES_B : 1, CS_A : 1)$$

with $\zeta = 1.0$.

The sequence can also be *dynamically* determined during the search. By changing strategies during the search, some of the synergistic interaction between these strategies can be exploited. Conversely, mixing dissonant strategies can have the opposite effect. For example, whenever the focus strategy is FS_B, using only the connection strategy CS_B yields better performance as

the intermediate nodes created while using CS_A often provide little value in making progress in workspace.

While a dynamic strategy sequence captures synergistic interaction between strategies, it is not context-dependant; therefore, it may miss contextual-cues and select unpromising strategies. For instance, A-RRTC+A-EETC, a $50-50$ blend of A-RRTC and A-EETC, is context-independent as it does not depend on the search state and it only depends on an independent coin-toss (see Table 5.2). In the interest of space, A-RRTC+A-EETC is also labeled as BLEND in the results and they refer to the same method.

CODES3 can be represented as a context-dependent *dynamic* strategy sequence. CC refers to the collision-check counts since the search started. SF is a flag indicating if a solution has already been found. Initially, CODES3 behaves almost like A-EETC till $CC < CL_1$ using $\Lambda_1 = (TS_C, \{FS_A : 0.0, FS_B : 1.0\}, \{NS_A : 0.50, NS_B : 0.50\}, \{TaS_A : 0.50, TaS_B : 0.49, TaS_C : 0.01\}, \{ES_A : 0.5, ES_B : 0.5\}, \{CS_A : 0.49, CS_B : 0.49, CS_C : 0.02\})$ and $\zeta = 0.9$. As CC increases and no solution has been found, it is likely that the workspace cues are misguiding. We switch gears and use a copy $\Lambda_2 \leftarrow \Lambda_1$ with the change $\Lambda_2.\lambda_{FS} = \{FS_A : 0.4, FS_B : 0.6\}$, $\zeta = 0.5$. In making this change, we are affirming our ambivalence regarding the effectiveness of the workspace cues. The conditions involving limits CL_1 and CL_2 act as a gauge of problem complexity. When CC exceeds CL_2 and no solution has been found, it is unlikely that workspace cues are useful at all; $\Lambda_3 \leftarrow \Lambda_1$ is used along with the change $\Lambda_3.\lambda_{FS} = \{FS_A : 0.9, FS_B : 0.1\}$, $\zeta = 0$. In making this change, we are affirming our guess that perhaps configuration space focusing is the only way to a solution.

As soon as a solution is found, we revert back to heavy workspace focusing by using Λ_1 and $\zeta = 0.9$ again. In doing so, we are redoubling efforts to find a higher-quality solution based on workspace cues, content in the fact that we have at least a feasible solution.

Instead of using fixed threshold of collision count (CC) to determine if workspace sampling was helpful. This threshold is arbitrary and may not be determined using any principled approach.

The Auto-Focus (AF) workspace/configuration rule is presented. It controls λ_{FS} and the rule is of the form $\lambda_{FS} = [FS_A : 1 - p_{FS_B}, FS_B : p_{FS_B}]$. The basic idea is to monitor progress along the sequence of balls and change p_{FS_B} accordingly. Progress refers to the update of the progress index towards the terminal ball. Whenever progress is made along the ball sequence, we increase the probability of workspace focus selection (p_{FS_B}). When no progress is made, we decrease p_{FS_B} by a fixed amount. The following equation describes the rule succinctly.

$$p_{FS_B}[k+1] = \min(0.8, \max(0.2, p_{FS_B}[k] - d + u)) \tag{5.14}$$

where $u = 0.1$ if progress is made and $u = 0$ otherwise. And, $d = 0.01$ is a decay parameter, k refers to the iteration count of the main while-loop. p_{FS_B} is limited to the range $[0.2, 0.8]$ to ensure that FS_B gets chosen even when no progress along the ball sequence has been made in a long time.

Table 5.2: Strategy Assignments

Name	**Strategy Sequence**
A-RRTC	Λ_R
A-EETC	Λ_E and setting $\zeta = 1$
A-RRTC+A-EETC (BLEND)	$\begin{cases} \Lambda_R, & \text{if heads on coin-toss} \\ \Lambda_E, & \text{if tails on coin-toss} \end{cases}$
CODES3	$\begin{cases} \Lambda_1, & CC \in [0, CL_1) \wedge \neg SF \\ \Lambda_2, & CC \in [CL_1, CL_2) \wedge \neg SF \\ \Lambda_3, & CC \in [CL_2, \infty) \wedge \neg SF \\ \Lambda_1, & SF \end{cases}$

5.4.8 Incorporating Human Assistance

In many applications, there is always a human-operator managing robot operations even if these robots are automated. Often these operators perform a final verification before the automatically-generated trajectories are executed. Despite the use of specialized techniques, automated trajectory-planning methods may not always find a solution within a given time budget; even if they find a

Table 5.3: Parameters

Parameter	Value
ϵ_q	0.12 rad
ϵ_p	0.03 m
μ_q	0.1 rad
μ_p	0.1 m
μ_o	0.1
γ	0.05
ω	0.05
CL_1	3000
CL_2	10 000

solution, the human-operator may reject it due to excessively long trajectories or uncomfortable proximity to obstacles.

In such cases, the human-operator may as well play a proactive role in shaping the produced trajectories. The field of cooperative-planning studies how human assistance can be incorporated into an underlying automated trajectory planner. The goal of cooperative-planning is to harness the power of human intuition in focusing the automated search and produce high-quality solutions quickly. For instance, humans intuitively know the shortest trajectories for scenarios as in Figure 5.8, whereas standard sampling-based methods may produce aesthetically disturbing trajectories.

This is particularly frustrating when there is high perception uncertainty in the vicinity of the computed trajectories and a replan is required. Playing an active role can reduce frustration and bridge the gap between the human-operator's expectation and the automatically produced trajectories. By cooperatively interacting with the planner, the operator can impart human intuition in focusing the search and help the automatic planner produce high-quality solutions quickly. Figure 5.8 shows example scenarios in which humans intuitively know the shortest end-effector paths to reach the goal pose.

While the performance of CODES3 is already better than alternative methods, it is beneficial to incorporate human assistance whenever CODES3 is unlikely to find a solution quickly or it is likely to find suboptimal solutions. Such a cooperative-planning scheme is useful to a robot operator only when it satisfies the following conditions: (C1) presents an intuitive graphical

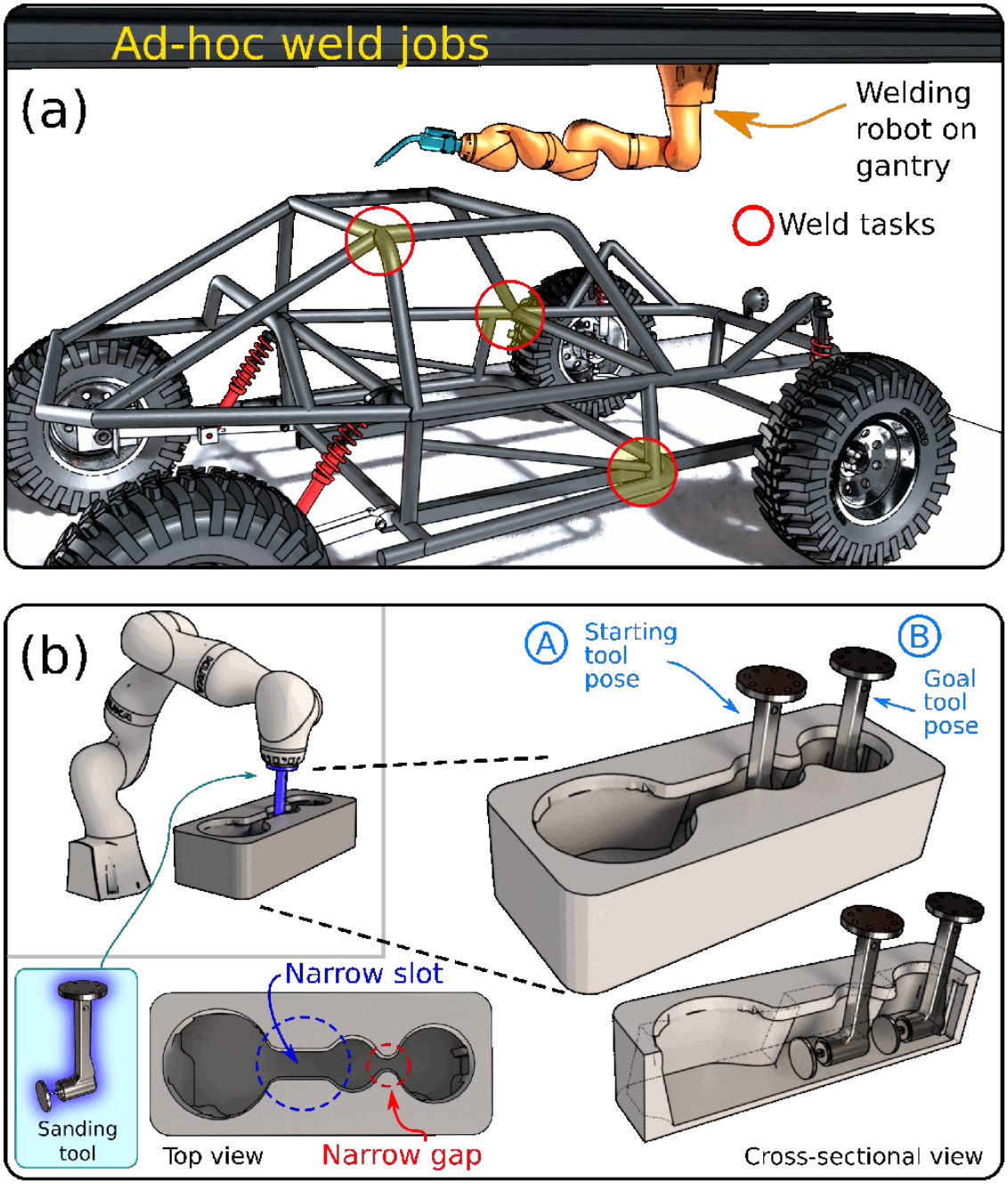

Figure 5.8: (a) A complex planning scenario where a human operator intuitively knows how to maneuver around obstacles, (b) Humans intuitively know the sanding tool cannot be slid across the narrow gap. By inspection, humans immediately know that the tool has to exit the cavity to reach the goal pose.

interface to the underlying planner, (C2) allows dynamic changes to the planner behavior over time, (C3) controls the underlying planner effectively, (C4) requests human assistance only when help is needed.

In this work, Human assisted CODES3 is referred to as HA-CODES3. In this work, the human operator provides the focus regions. An illustrative example of a cooperative-planning workflow is described using Figure 5.9. Figure 5.9 shows a scenario where a roll-cage assembly is welded for a buggy. Given a welding scenario as in Figure 5.9, CODES3 starts off un-assisted to solve the welding problem. Meanwhile, the human-operator is presented a graphical-user-interface (GUI) as in Figure 5.9. Depending on the progress of the search, the operator can choose to make the following changes:

1. **Tweaks to Focus Selection Bias:** The human-operator can dynamically change λ_{FS} using the slider. Suppose the slider value is $v \in [0, 1]$ ($v = 0$ at FS_A). Then, $\lambda_{FS} = \{FS_A : 1 - v, FS_B : v\}$. This gives the operator an option to override and disallow workspace biasing,

2. **Tweaks to the positions and sizes of the workspace balls by moving them around:** Moving one ball will affect the positions of other balls in a manner similar to a string of beads; this way the operator spends the least time positioning all the balls. Whenever the balls are modified, the progress indices are reset,

3. **Clicking workspace balls:** The operator can also observe a live value from the CODES3 algorithm indicating the usefulness of the selected ball (the red icon). The usefulness value acts as the feedback to the operator and it is determined by the ratio between the number of successful extends and the total number of extends that were made within the selected ball. The operator uses it to make an informed guess at the correct value of $\zeta \in [0, 1]$ ($\zeta = 0$ is the cross symbol) specific to the selected ball by sliding. The workspace projection of both trees is also displayed (for clarity, this has not been visualized in Figure 5.9). As the tree

search continues, the ball corresponding to the progress indices of both trees is also shown to the operator using a tinted color.

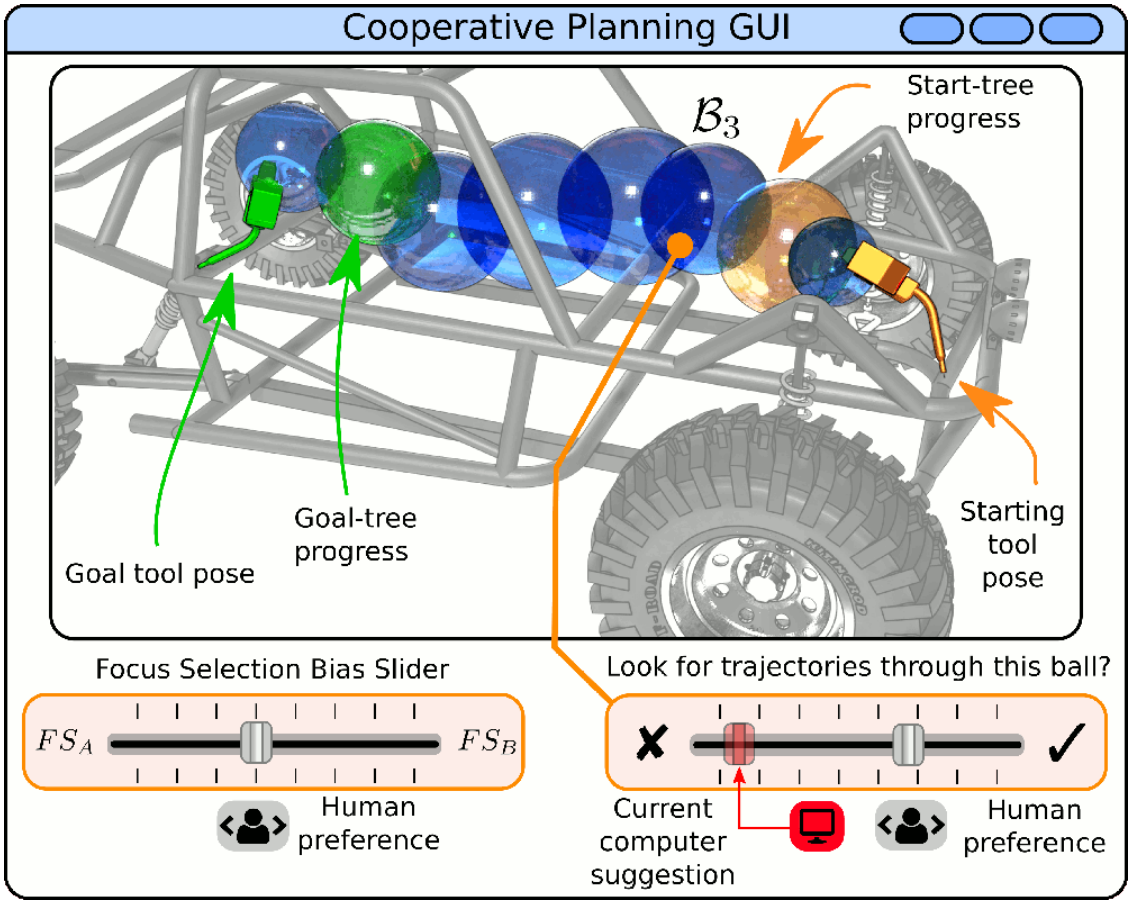

Figure 5.9: An example of a cooperative-planning GUI. Focus regions in the form of workspace balls are created by the human-operator to guide the welding tool around the roll-cage assembly. In the lower-right area, a slider control pops up when the human operator clicks a workspace ball.

5.5 Results and Discussion

The proposed method was implemented in MATLAB on a Dell workstation with Intel Xeon 3.5 GHz CPU and 32 GB RAM. Table 5.3 lists some of the important parameters we used. EETC and RRTC* refer to the standard implementation of bi-directional variant of [158] and [86] respectively. The proposed method was tested in a diverse set of 30 scenarios. This set contains

scenarios which are: easy, hard, workspace-misguiding, workspace-guiding. Figure 5.10 shows a representative subset of the test scenarios.

For each scenario, a 3D Euclidean Distance Transform (EDT) of the environment [80] was precomputed. Collision checks were performed between the environment and a ball-approximation of the robot using the EDT. All methods were given a planning timeout of $T_o = 7.5\,s$ or $15\,s$ to produce any number of valid solutions and a smoothing timeout of $T_s = 2.5\,s$ to achieve less than 1% relative cost difference between consecutive smoothing iterations. If any method took longer than T_o, the trial was recorded as a failure; otherwise, for each of the raw solutions found, the distance traveled by the tool-tip was also recorded as a metric of trajectory quality. Finally, the raw solution with the least tool-path distance was smoothed using a simplified version of [70]. Independently, the optimal trajectory for each scenario was computed in order to assess the *trajectory cost sub-optimality factor* (i.e., ratio of trajectory cost obtained to the optimal trajectory cost) for each of the smoothed solutions produced. Results shown in Figure 5.11, Figure 5.12, Table 5.5 and Table 5.4 were computed using the data collected over 50 trials of each planner-scenario combination using different planning timeouts.

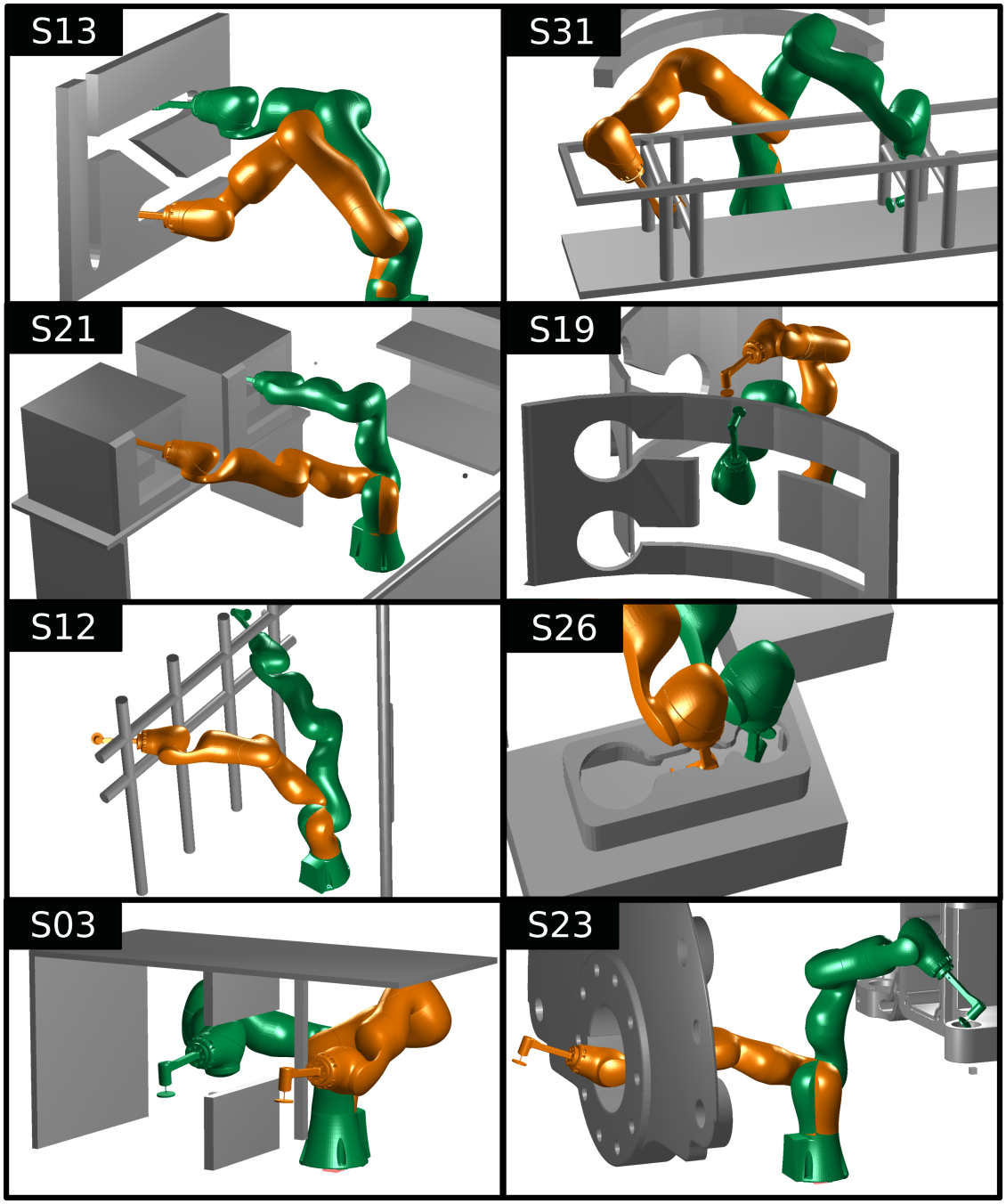

Figure 5.10: A representative subset of test scenarios. For clarity, only one of many possible joint configurations is shown for each scenario.

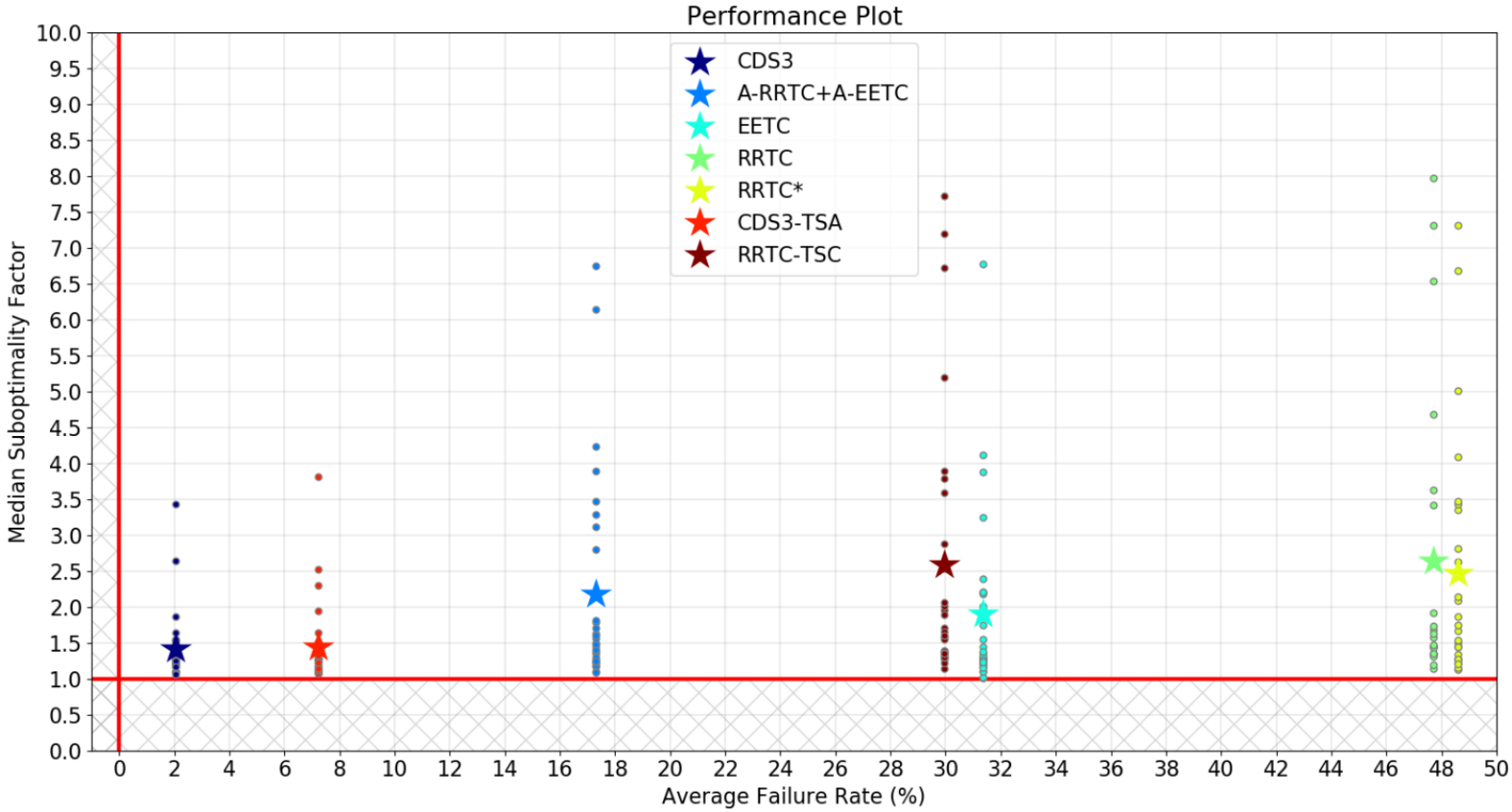

Figure 5.11: Performance metrics of CODES3 and alternative methods aggregated over 30 diverse scenarios. All methods were given time budget $T_o = 7.5\,s$ to compute solutions. The dots indicate median sub-optimality factor for individual scenarios and the stars indicate average performance over all scenarios tested. The point $(0, 1)$ is an ideal point indicating zero failure rate and optimal solution quality. Some data points that have a sub-optimality factor greater than 8 are not visualized.

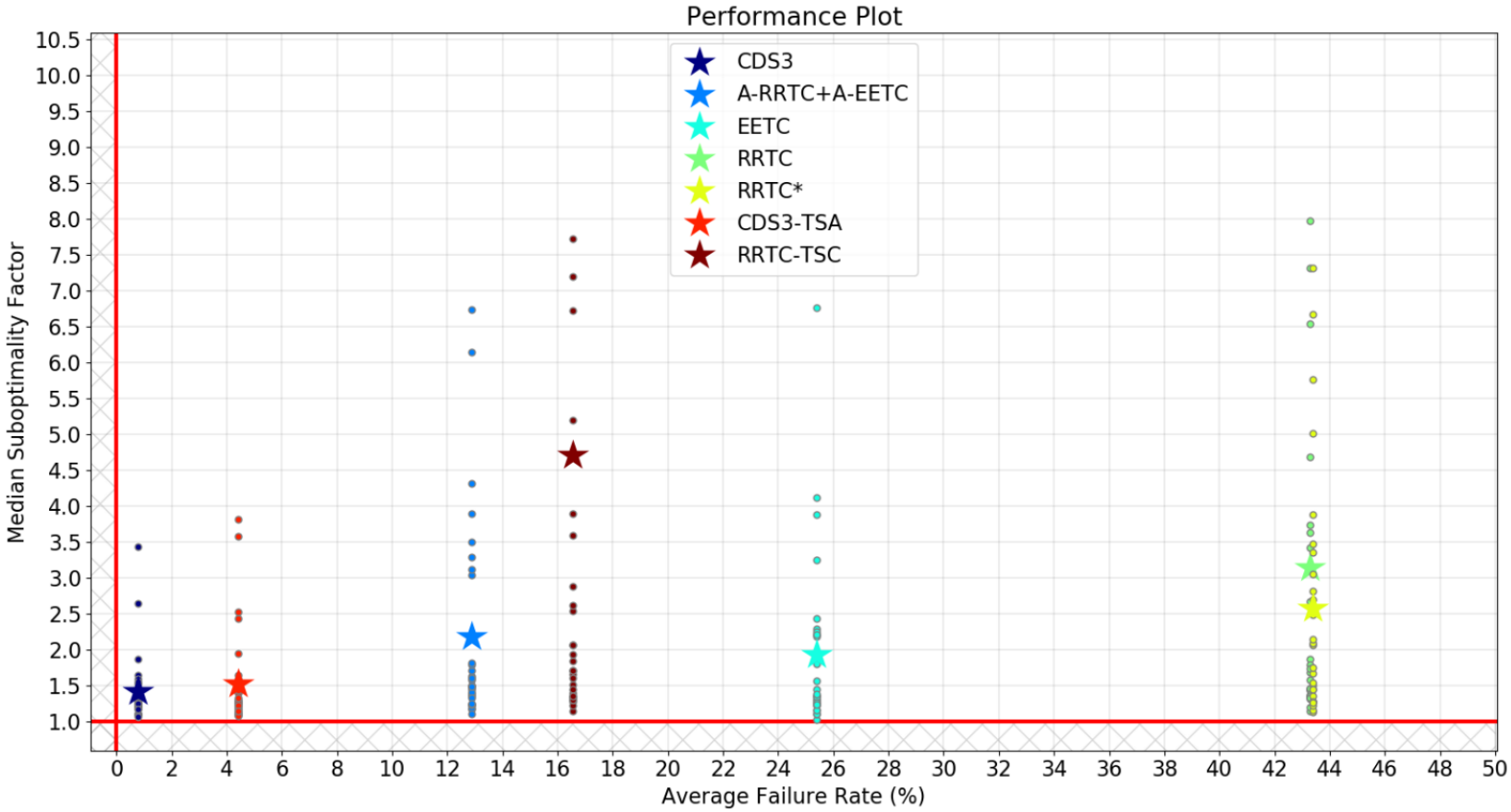

Figure 5.12: Performance metrics of CODES3 and alternative methods aggregated over 30 diverse scenarios. All methods were given time budget $T_o = 15\,s$ to compute solutions. The dots indicate median sub-optimality factor for individual scenarios and the stars indicate average performance over all scenarios tested. The point $(0, 1)$ is an ideal point indicating zero failure rate and optimal solution quality. Some data points that have a sub-optimality factor greater than 8 are not visualized.

Table 5.4: Failure rate and suboptimality factor for all tested methods ($T_o = 7.5\,s$)

Method	**Failure Rate (%)**	**Suboptimality Factor**
RRTC	47.71	3.14
RRTC-TSC	29.94	4.72
RRTC*	48.59	2.56
EETC	31.35	1.93
A-RRTC+A-EETC (BLEND)	17.30	2.18
CODES3-TSA	7.23	1.53
CODES3	2.05	1.42

Table 5.5: Failure rate and suboptimality factor for all tested methods ($T_o = 15\,s$)

Method	**Failure Rate (%)**	**Suboptimality Factor**
RRTC	43.29	2.65
RRTC-TSC	16.56	2.59
RRTC*	43.37	2.47
EETC	25.39	1.90
A-RRTC+A-EETC (BLEND)	12.89	2.18
CODES3-TSA	4.40	1.45
CODES3	0.79	1.41

Results shown in Table 5.6 were computed using the data collected over 30 trials of each planner-scenario combination.

A brief description of some of the test scenarios is given below. *S13* features a slot along which a sanding tool is to be moved. In *S31*, the sanding tool is lodged between two horizontal bars, requiring some tool reorientation before it comes free. In *S21*, the manipulator is reaching into a machine and transferring a workpiece to another machine. In *S19*, the manipulator is reorienting the tool to better reach the other side of the large curved plate. In *S12*, the manipulator is exposed to a pillar problem where it has to move from one side to the other. *S26* is skipped as it is described in detail in Figure 5.1(b). In *S03*, the manipulator is attempting to reach to the opposite side under table. In *S23*, the manipulator needs to retract through a tunnel to reach the other part.

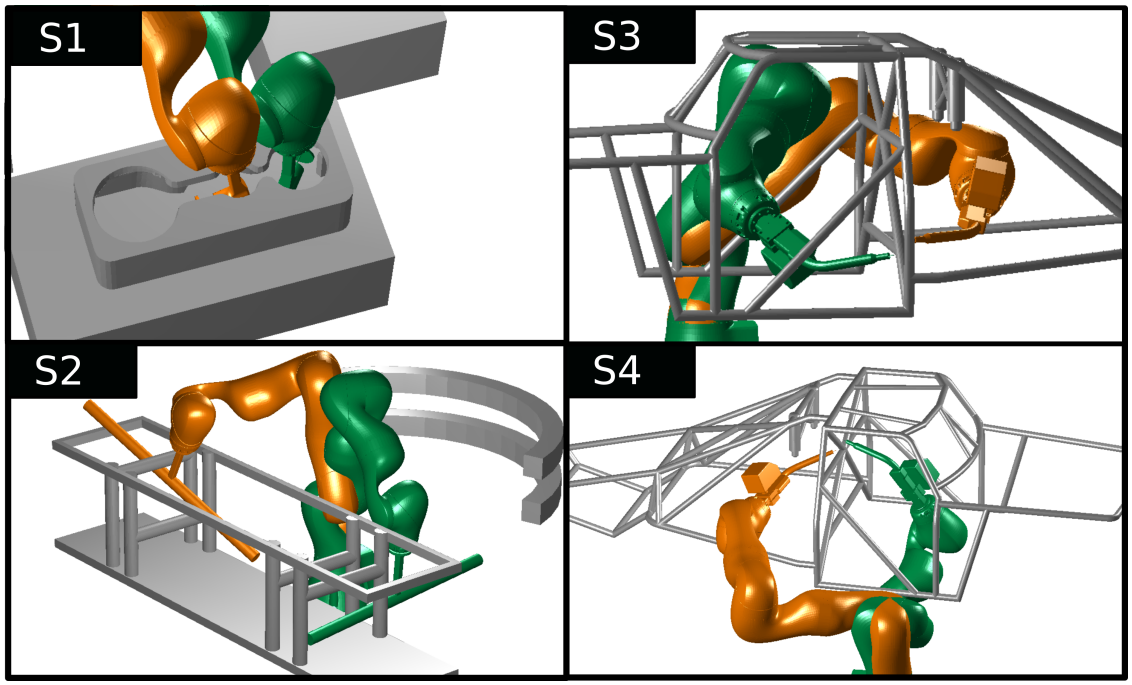

Figure 5.13: Test scenarios where human operator assistance is utilized. For clarity, only one of many possible joint configurations is shown for each scenario.

RRTC and RRTC* have the highest failure rates. In Figure 5.11, compared to RRTC, RRTC* yields only a slightly better solution quality and incurs an increased failure rate. This is due to time spent rewiring nodes that were never going to be part of a high-quality solution. Given more time, both *RRTC* and *RRTC** manage to achieve a roughly equal failure rate.

EETC dominates RRTC in terms of failure rate and trajectory quality. The blending of EETC and RRTC as in A-RRTC+A-EETC (BLEND) yields lower failure rates and demonstrates the synergistic interaction between the strategies. This positive interaction is particularly noticeable in the workspace-misguiding scenarios (S12, S26, S19, S31) where EETC is likely to fail often. However, it does little to increase trajectory quality.

Context-awareness and synergistic interaction of strategies help drive the failure rate down and increase trajectory quality in CODES3. Specifically, random sampling in $\mathcal{W}$-space helps to easily identify narrow passages in $\mathcal{W}$-space that are close to the end-effector; thus, it effectively identifies the corresponding $\mathcal{C}$-space narrow passages (e.g. scenarios S23, S21). Although the trajectory

Table 5.6: Failure Rate (FR) and Path Length (PL) for all tested methods

Scenario	Metric	RRTC	RRTC*	EETC	CODES3	HA-CODES3
S1	FR (%)	86.67	96.67	86.67	0.03	0.00
	PL (m)	0.64	0.59	0.71	0.68	0.62
S2	FR (%)	100.00	100.00	86.67	26.67	0.00
	PL (m)	-	-	2.16	2.10	1.72
S3	FR (%)	100.00	100.00	100.00	53.33	0.00
	PL (m)	-	-	-	1.92	1.69
S4	FR (%)	100.00	100.00	100.0	0.00	0.00
	PL (m)	-	-	-	1.70	1.45

quality of alternative methods are occasionally better in some tough scenarios, CODES3 has a considerably lower failure rate and suffers only slightly in terms of trajectory quality. Sub-optimal feasible solutions obtained at a low failure-rate are often more valuable than optimal solutions with high failure-rates. Nevertheless, in general, the trajectory quality is better when CODES3 is used; better trajectory quality is attained quickly compared to other methods. CODES3 is able to achieve a low suboptimality factor of 1.42 within the first 7.5 s, whereas the next best alternative $EETC$ only manages to achieve 1.93 (Table 5.4).

The impact of the tree selection strategy TS_C is studied. The effect of using TS_C is very pronounced whenever one of the trees is rooted in a narrow passage (e.g. scenario S26). The extent to which employing TS_C benefits RRTC is tested by considering RRTC-TSC, a variant of RRTC which uses TS_C instead of TS_A. It is observed that the use of TS_C decreased the failure rate from 47.71% to 29.94%. CODES3-TSA, a variant of CODES3 using TS_A instead of TS_C was also tested. Not using TS_C resulted in an increase of failure rate from 2.05% to 7.23% in the case of CODES3-TSA.

5.5.1 Impact of Human Assistance

A brief description of the test scenarios in Figure 5.13 is given. *S1* is skipped as it is described in detail in Figure 5.1(b) as well as in [82]. In *S2*, a long rod-like tool is lodged between two

horizontal bars and it needs to be moved to the opposite side. *S3* and *S4* feature a scenario where a miniature jeep roll cage is welded. The manipulator is reaching into the jeep's window to weld the tubes. Navigating the voids between the tubular structures is a challenge.

In Table 5.6, compared to RRTC, RRTC* gives slightly better solution quality. However, due to the time spent rewiring nodes that were never going to be part of a high-quality solution, RRTC* suffers an increased failure rate.

Context-awareness and synergistic interaction of strategies help drive the failure rate down and increase trajectory quality in CODES3. Specifically, random sampling in $\mathcal{W}$-space helps to creep along narrow passages close to the end-effector. As a result, the corresponding $\mathcal{C}$-space narrow passages are identified effectively. Though the trajectory quality of alternative methods may be better in S1, CODES3 has a very low failure rate and yet comes very close to that of RRTC*. One often prefers low failure-rate over solution quality. In light of this, it is observed that HA-CODES3 performs very well with zero failure rate and a very good solution quality.

5.5.2 Impact of Strategies

To study the impact of strategies, test scenarios shown in Figure 5.14 is used. A brief description of some of the test scenarios is given below. *S03* features a table-like structure where the tool has to reach the other side of the central partition. In *S04*, a sanding tool needs to be moved from one side of the mold to the middle slot. There is a narrow constriction blocking the way and the tool cannot be slid through. In *S07*, the manipulator is reaching into the void formed by the cross-bars supporting the cylindrical structure. In *S12*, the manipulator is exposed to a pillar problem where it has to move from one side to the other. In *S14*, the manipulator needs to move from one mold to another mold. But, its motion highly restricted by the long vertical rod in the middle. *S26* is skipped as it is described in detail in Figure 5.1(b). In *S30*, the manipulator needs to move the tool from a confined pose between two bars. The tool has to be taken out into a new pose where it is relatively less confined.

Figure 5.15 was generated using the data collected over 30 runs of each method-scenario combination (methods are listed in the legend). In Figure 5.15, C3 refers to CODES3. And, C3-P refers to CODES3 including the strategies TaS_D (RBF), CS_D (CB, Connection Biasing) and Auto-Focusing (AF). C3-P performs very well compared to the rest of the methods. Specifically, it beats C3 (CODES3) in terms of failure rate. C3-P reduces the failure rate by 4 times (compared to C3) and brings it down to 1%. In terms of suboptimality factor, both C3-P and C3 perform equally well and have no discernible difference. This is expected as none of the strategies (i.e., RBF, CB and AF) were focused towards improving solution quality.

Table 5.7 compares the performance of C3 as different strategies are enabled and disabled. For instance, the column +CB consists of the baseline C3 and the inclusion of CB only. Similarly, the column +AF and +RBF refers to C3+AF and C3+RBF respectively. The method C3-P consists of the baseline C3 and the inclusion of RBF, CB and AF.

When planning times are considered, the introduction of RBF has the most dramatic effect compared to that of CB and AF. Notably, introducing RBF in S07, S12, S26 and S30 reduces planning time by a large factor. This is due the presence of a few difficult regions (e.g., narrow passage) in these scenarios. Local models help accelerate the search and reduce planning time. However, in S14 and S23, there are too many of non-overlapping difficult regions in the configuration space (e.g., the tunnel in S23). Many local models are created and thus, the advantage of RBF is greatly diminished resulting in higher planning time than C3. No significant change is observed in sub-optimality factor or failure rate.

Introduction of AF generally reduces planning time with the exception of S26. S26 is a very challenging problem that involves intricate configuration space maneuvers. And, in this case AF increases the failure rate to 100% from the baseline of 88%. No significant change is observed in sub-optimality factor. Introduction of CB consistently reduces planning time across all tested scenarios.

Through combining strategies, C3-P is able to generally reduce planning time by atleast 13% and upto 70%. In some cases, it does worse by as much as 20%. However, the overall failure rate is improved by C3-P compared to C3.

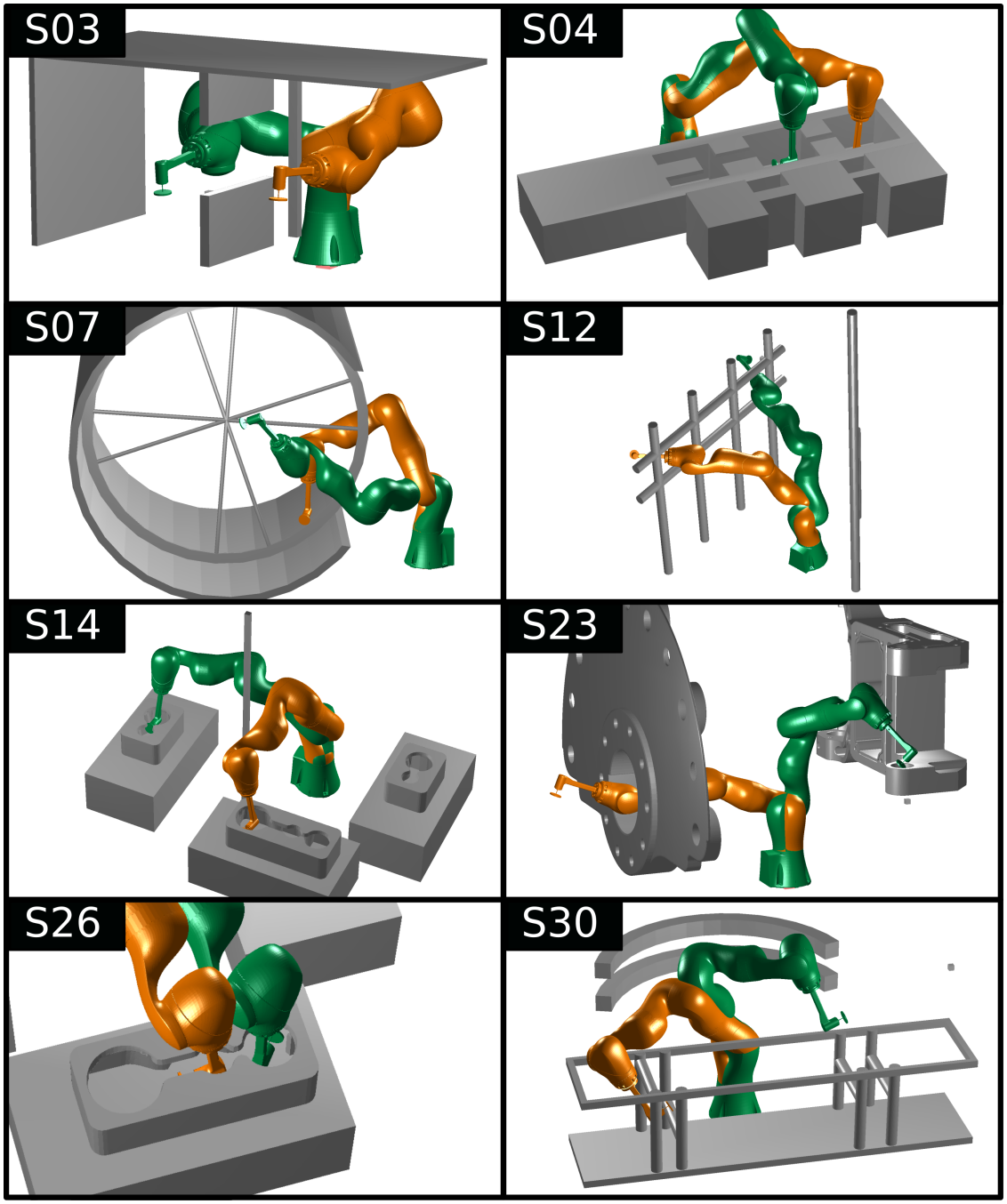

Figure 5.14: A representative subset of test scenarios used for analyzing the impact of various strategies. For clarity, only one of many possible joint configurations is shown for each scenario.

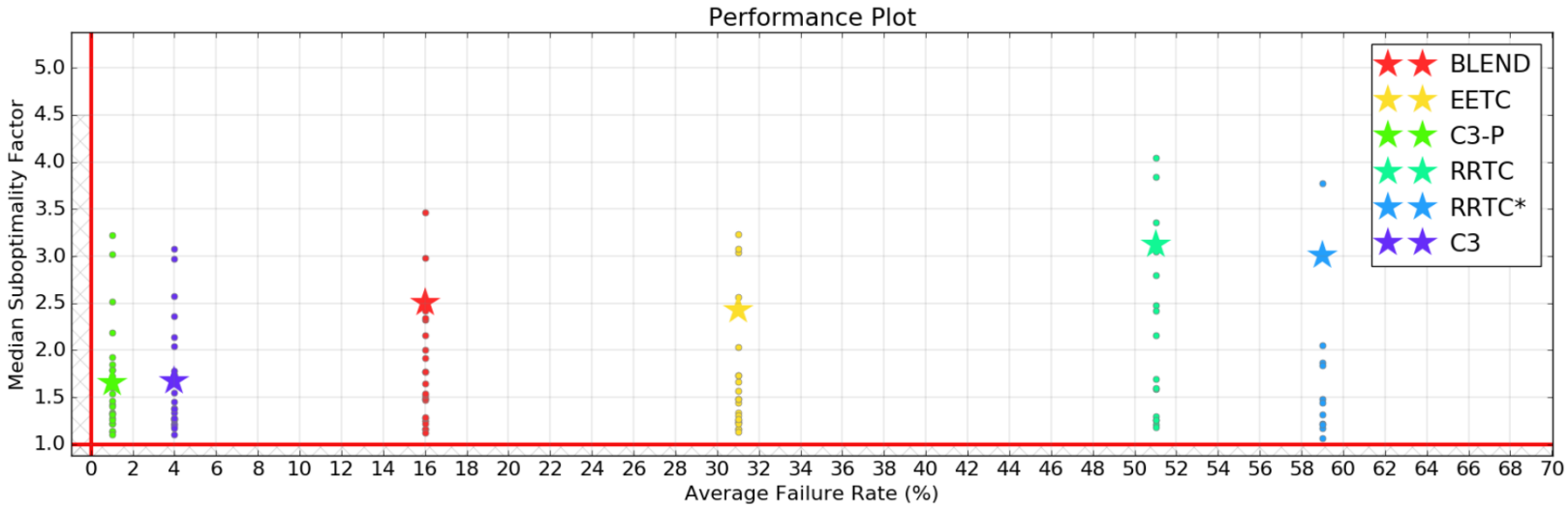

Figure 5.15: Performance of CODES3 (C3), CODES3 with RBF, AF, and CB (C3-P) and other methods

Table 5.7: Impact of strategies on a subset of tested scenarios

Scen.	Planning Time (s)						Sub-Optimality Factor						Failure Rate (%)					
	C3	**+CB**	**+AF**	**+RBF**	**C3-P**	**Red. (%)**	**C3**	**+CB**	**+AF**	**+RBF**	**C3-P**	**Red. (%)**	**C3**	**+CB**	**+AF**	**+RBF**	**C3-P**	**Red. (%)**
S03	5.1	5.0	4.5	3.9	5.8	−13.1	10.6	10.9	9.8	9.8	9.7	8.4	0.0	0.0	0.0	0.0	0.0	–
S04	1.1	0.8	0.6	0.6	0.6	42.5	6.0	5.7	6.2	6.4	6.1	−0.4	0.0	0.0	0.0	0.0	0.0	–
S07	2.6	1.0	1.0	0.3	0.8	70.0	7.6	7.6	7.6	7.6	7.6	−0.1	0.0	0.0	0.0	0.0	0.0	–
S12	9.4	9.6	6.8	6.5	6.9	26.4	1.6	1.6	1.6	1.5	1.7	−4.4	8.0	0.0	0.0	0.0	0.0	100.0
S14	7.8	6.8	6.9	9.7	9.3	−19.3	1.9	1.9	1.9	1.8	1.8	2.0	0.0	15.8	5.0	0.0	5.0	–
S23	2.9	2.4	2.7	2.8	2.6	13.0	1.5	1.5	1.5	1.4	1.5	1.5	0.0	0.0	0.0	0.0	0.0	–
S26	16.6	15.7	–	10.8	14.5	14.1	1.3	1.8	–	1.7	1.6	−28.5	88.0	20.0	100.0	78.0	36.0	59.5
S30	1.6	1.0	0.7	0.5	0.8	48.2	3.1	3.2	3.1	3.0	3.0	0.8	0.0	0.0	0.0	0.0	0.0	–

5.6 Summary

A context-dependent bi-directional tree-search framework for point-to-point trajectory planning for manipulators is presented in this chapter. Conceptually, the framework is composed of six modules: tree selection, focus selection, node selection, target selection, extend selection and connection type selection. Each module consists of a set of interchangeable strategies. Many strategies are explained in detail in this chapter. Human assistance is also considered a strategy that could be used by the planner. By exploiting synergistic interaction between these strategies and selecting appropriate strategies based the contextual cues from the search state, the proposed method computes high quality solutions in a variety of complex scenarios with a low failure rate. It is also shown that some popular trajectory planning methods in the literature can be easily represented in the proposed framework. The proposed approach is compared with these popular methods in a diverse set of test scenarios. The proposed approach yields a 15-fold reduction in failure rate coupled with at least a 26% drop in solution suboptimality when compared to the best of the alternative methods. With human assistance, the proposed method has a zero failure rate as well as high solution quality in the test scenarios.

Chapter 6

Conclusion

6.1 Intellectual Contributions

This dissertation develops computational techniques for speeding up for trajectory planning for autonomous robots operating in complex environments. This dissertation presents advancements in speed-up techniques to perform: (1) dynamic obstacle avoidance while satisfying dynamics constraints and accounting for perception uncertainty, (2) dynamic obstacle avoidance while traversing large uncertain dynamic obstacles, satisfying trajectory constraints and accounting for motion and perception uncertainty, (3) obstacle avoidance in high-dimensional configuration spaces. The key contributions are outlined below.

- *Reactive Trajectory Planning* A speed-up scheme where the trajectory planning problem is decoupled into path generation and velocity tuning. A collision-risk assessment method that incorporates the varying dynamics constraints of the robot and uncertainty in the measurement of obstacle's state is presented. The speed-up scheme adapts to varying number of dynamic obstacles by prioritizing critical obstacles over non-critical obstacles. The scheme also is able to maintain real-time behavior and able to handle suddenly appearing obstacle by prioritizing it over other obstacles.

- *Deliberative Trajectory Planning* The main contributions of this work are speed-up techniques applicable to trajectory planning over long distances while avoiding static obstacles and large dynamic obstacles. , introduces a failure-risk assessment scheme for traversing large dynamic obstacles. A speed-up scheme based on heuristics generated by an approximate lower-dimensional problem for the search process to generate trajectories rapidly and be useful in dynamic environments. It also provides dynamic obstacle heuristics in the form of a precomputed look-up table. The generated heuristics are aware of large dynamic obstacles and help reduce the planning effort. A collision-risk assessment method that uses a precomputed trajectory tracking error model to handle motion uncertainty, incorporates perception uncertainty and large permeable dynamic obstacles. The incorporation of the risk of traversing large permeable dynamic obstacles into the planning process improves the success probability of the planned trajectory. The speed-up scheme produces safe trajectories that execute faster than purely reactive methods on the physical robot platform. It also reacts safely to varying levels of perception uncertainty.

- *Trajectory Planning for High Degree-of-Freedom Robots Operating in Cluttered Environments* The main contribution is a context-dependent bi-directional tree-search framework that schedules multiple speed-up techniques. It is a flexible framework that enables switching between multiple heuristics to yield a resilient search method amenable to machine learning. It is also able to represent many existing methods. Other contributions include new heuristics for tree selection, target selection, connection strategy selection. Moreover, a user-friendly interface for incorporating human assistance in a cooperative planning setting is presented. Finally, rules for scheduling heuristics is also given. Using these scheduling rules, CODES3, a high-performance trajectory planner based on the framework is presented.

6.2 Summary of Lessons Learnt

The previous section went over the major contributions of this dissertation. This section is focused on discussing lessons learned from work in this dissertation and how these can be used by readers for speeding up trajectory planning.

The simplest technique that can be used for speeding up computation is the technique of precomputation. Many methods rely on precomputed data for efficient planning. For example, consider a problem where repeated query of the distance from a 3D point to the closest obstacle in 3D space is required. This distance computation can be performed on-demand and this may be computationally expensive. When many distance queries are required, a lot of computation time is required to perform these queries. Euclidean Distance Transform (EDT) is a technique where the distance to obstacles is precomputed for every point in a query region (e.g., box) and stored in a 3D matrix. Having the EDT computed, the distance query becomes very fast. When a point is queried, the cell (query cell) corresponding the point in the 3D matrix is first calculated. Then, the distance entry recorded in that cell is an approximate distance between the point and the closest obstacle. The distance is approximate due to the required discretization of the query region and only one distance is stored per cell. This highlights the need to be aware of the level of error that discretization of the problem brings about. This error can somewhat be mitigated by interpolating values between neighboring cells for a particular query cell. Furthermore, the finer the discretization level, the more memory is required to store the precomputed values. There is a trade off between storage space and computation time. There are few tips to keep in mind when picking a quantity to precompute. Sometimes the quantity to be precomputed may not be obvious. Some property of the system/problem instance may have to be precomputed. A contrived yet illustrative example would be the following. Suppose that set (S) of 10000 unique numbers are given. In that array, one number is randomly removed. Now, given the leftover set (S') of 9999 numbers, we are tasked to find the number that was randomly removed. One of the

possible ways to find the removed number is to compute an property of the original set and utilize it. In this case, the sum s of all numbers in S is a good property. Similarly, the sum s' of all numbers in S' can be computed. The missing number can be calculated as $s - s'$. If a property is used, then it is a good idea to pick properties that are easy to compute. In the above case, the sum of these numbers is a simple property that can be computed very quickly.

When the search-space is large, any search method becomes slow. If efficient heuristics can be computed, then large search-spaces can be handled easily. However, finding efficient heuristics can be more of an art than a scientific procedure.

In the context of A* search, a heuristic is function that gives cost-to-go values for a given query state. The cost-to-go value represents the potential accumulated cost of going to the goal state from the query state. In such a case, to develop a good heuristic, a good starting point is to observe the terms of the cost function and identify terms for which good approximations can be quickly computed. For example, if a particular cost term is known to dominate the total cost at the goal state, then that term is a good candidate for an attempt at approximation.

Other non-numeric heuristics can be generated by observing the regions of search-space that optimal/high-quality solutions pass through. If a function can be developed that takes cues from the search-space/search-state and produces promising regions in which high-quality solutions may lie, it can be used to guide search in the predicted promising regions.

Another class of techniques is approximation techniques. The idea is to approximate the problem by simplifying certain less important details. For example, the popular non-linear programming method Sequential Quadratic Programming (SQP) follows this idea. In SQP, the original problem is approximated by a series of QPs (Quadratic Problem). As each QP is solved, the solution gets closer to the solution of the original problem. This way a series of QP is solved to finally reach the solution of the original problem. A similar technique can be adopted to solve trajectory search problems. A local analytical approximation of the constraints (e.g., obstacles) in the search space can be used to find approximate solutions. These approximate solutions may

not be feasible considering the original constraints (i.e., actual obstacles). Nevertheless, the approximate solutions can be repaired or used in some manner to aid generation of the solution to the original problem. Another approximation scheme could be dimensionality reduction. Instead of approximating constraints, the dimensionality of the state space can be reduced by ignoring the less dominant dimensions. The lower dimensional problem may be easier to solve quickly. The solutions of the lower dimensional problem can be repurposed as a heuristic/guess for the original high dimensional problem.

Another related idea is to decouple complex problems into its constituent sub-problems. This involves first observing where the problem is loosely coupled. For example, a trajectory generation problem can be decoupled into path generation and timing generation. Path generation is performed first, followed by timing generation. Path generation deals with how to navigate around obstacles while timing generation deals with when to be at a particular point on the path. There can be certain conditions that need to be satisfied before such a decoupling can be justified. In the example above, the dynamics of the robot could prevent the robot from executing a certain path due to restrictions on velocity. However, restrictions on velocity are considered only in timing generation. Therefore, it is necessary to ensure the decoupling does not introduce infeasiblity. Once the sub-problems of path generation and timing generation are solved, the solution to the trajectory generation problem can be computed by assembling the solutions of the sub-problems.

6.3 Anticipated Benefits

This dissertation presents speed-up techniques for automated trajectory planning for autonomous mobile robots subject to motion uncertainty, perception uncertainty. Many mobile robots in practical settings require automated trajectory planning. Specifically, it is anticipated that the methodology presented in this work will be applicable to mobile robots operating in an outdoor setting such as uneven terrain, water bodies. In such settings, the speed-up techniques will

allow the robot to quickly react to the environment and perform tasks safely. Depending on the application domain, this will serve as an enabling technology for advanced services. For instance, in the marine domain, applications in areas such as automated port assistance (e.g., container ship docking), water taxis, marine structure inspection are possible. It will reduce operating costs, reduce vessel collisions, improve traffic flow, and increase robot survivability.

This dissertation also presents a general framework accelerates trajectory planning for robots with high degrees of freedom through the speed-up techniques. Many industrial processes are currently use high degree-of-freedom manipulators that are manually programmed by a human operator. Methods presented in this work can greatly simplify workflows related to manipulators. For instance, humans can offload the time-consuming process of manipulator trajectory generation to the proposed method. This can increase human productivity and manufacturing throughput. This work also serves as a enabling technology for higher-level task planning methods.

6.4 Future Directions

- *Reactive Trajectory Planning for Mobile Robots* The overall trajectory generation architecture does not make assumptions regarding sensing modalities and vehicle models. Though the trajectory generation method is presented in the context of UGVs over uneven terrain, it is widely applicable to a variety of mobile robot platforms and with some modifications, even unmanned surface vehicles operating in dynamic sea states encountering dynamic obstacles.

  For each terrain type, the dynamics constraints were assumed to be known before hand. A natural extension would be to perform online learning of dynamics constraints, starting with conservative estimates and converging to the correct constraints. While effective, this approach has some limitations. Opportunities to explore multiple homotopy classes while navigating around $s - t$ obstacles are eliminated due to the way multiple obstacles are handled. This issue can be addressed.

- *Deliberative Trajectory Planning for Mobile Robots* Through the use of heuristics and adaptive motion goal sets, the planning effort and time has been reduced. However, further reduction is required for practical deployment in congested harbors. Harbors have speed limits ranging between $1.8\,m\,s^{-1} - 5\,m\,s^{-1}$ and reacting to traffic vessels potentially moving at those speeds requires faster planning times than possible by the current MATLAB implementation.

  Furthermore, the proposed method does not address the reactive behaviors of traffic vessels which is crucial for practical situations. Potential directions of future work are: (1) developing an optimized C++ implementation to achieve a faster planning time by utilizing GPU and multi-core frameworks to perform the computations in the Monte Carlo scheme used for failure probability computations, (2) incorporating reactive behavior of traffic vessels to the actions of the USV to ensure safe trajectories are generated. The interaction of wavefields generated by multiple traffic vessels has been ignored. The impact of this interaction remains to be studied.

- *Trajectory Planning for High Degree-of-Freedom Manipulators Operating in Cluttered Environments* Future avenues of research include developing more informed switching conditions that help switch between heuristics. There is significant potential for improving the performance by introducing more informed switching conditions. And, non-linear connection strategies can developed to improve performance. Developing a polished and intuitive GUI for obtaining human assistance is a promising future work. Incorporating machine learning techniques to identify switching conditions automatically is a promising future direction. In addition, applying Deep Reinforcement Learning to learn a policy for changing heuristics is powerful and has a very high impact potential. Adapting incremental tree repairing techniques can provide asymptotic optimality guarantees.

Reference List

[1] Bhp billiton plans for roboships within a decade — workboat. https://www.workboat.com/news/bluewater/bhp-billiton-plans-roboships-within-decade/. (Accessed on 05/21/2018).

[2] The business of automation, betting on robots.

[3] Latest news - annual overview of marine casualties and incidents 2017 - emsa - european maritime safety agency. http://www.emsa.europa.eu/news-a-press-centre/external-news/item/3156-annual-overview-of-marine-casualties-and-incidents-2017.html. (Accessed on 06/01/2018).

[4] New project to prove use of autonomous surface vessels for offshore wind. https://ore.catapult.org.uk/press-releases/. (Accessed on 05/21/2018).

[5] Rolls-royce might build the worlds first autonomous naval vessel. https://futurism.com/rolls-royce-might-build-the-worlds-first-autonomous-naval-vessel/. (Accessed on 05/21/2018).

[6] Nato and rafael carry out unmanned surface vehicle missile exercise off israeli coast, May 2019.

[7] M. Abdelaal and A. Hahn. Nmpc-based trajectory tracking and collision avoidance of unmanned surface vessels with rule-based colregs confinement. In *2016 IEEE Conference on Systems, Process and Control (ICSPC)*, pages 23–28, Dec 2016.

[8] A. Ademovic and B. Lacevic. Path planning for robotic manipulators using expanded bubbles of free c-space. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 77–82, May 2016.

[9] A. Agha-mohammadi, S. Chakravorty, and N. M. Amato. Firm: Feedback controller-based information-state roadmap - a framework for motion planning under uncertainty. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4284–4291, Sept 2011.

[10] Sandip Aine, Siddharth Swaminathan, Venkatraman Narayanan, Victor Hwang, and Maxim Likhachev. Multi-heuristic a*. *The International Journal of Robotics Research*, 35(1-3):224–243, 2016.

[11] Hossein Akbaripour and Ellips Masehian. Semi-lazy probabilistic roadmap: a parameter-tuned, resilient and robust path planning method for manipulator robots. *The International Journal of Advanced Manufacturing Technology*, 89(5-8):1401–1430, 2017.

[12] Baris Akgun and Mike Stilman. Sampling heuristics for optimal motion planning in high dimensions. In *International Conference on Intelligent Robots and Systems*, pages 2640–2645, 2011.

[13] Vivek Annem, Pradeep Rajendran, Shantanu Thakar, and Satyandra K. Gupta. Towards remote teleoperation of a semi-autonomous mobile manipulator system in machine tending tasks.

[14] Georges S Aoude, Brandon D Luders, Joshua M Joseph, Nicholas Roy, and Jonathan P How. Probabilistically safe motion planning to avoid dynamic obstacles with uncertain motion patterns. *Autonomous Robots*, 35(1):51–76, jul 2013.

[15] Csar Arismendi, David lvarez, Santiago Garrido, and Luis Moreno. Nonholonomic motion planning using the fast marching square method. *International Journal of Advanced Robotic Systems*, 12(5):56, 2015.

[16] Oktay Arslan and Panagiotis Tsiotras. Machine learning guided exploration for sampling-based motion planning algorithms. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2646–2652. IEEE, 2015.

[17] Daman Bareiss and Jur van den Berg. Generalized reciprocal collision avoidance. *The International Journal of Robotics Research*, 34(12):1501–1514, 2015.

[18] Timothy Barfoot. Informed RRT*: Optimal Sampling-based Path Planning Focused via . (Iros):1–8, 2014.

[19] O. Burhan Bayazit, Guang Song, and Nancy M. Amato. Ligand binding with obprm and user input. In *ICRA*, 2001.

[20] Mayur Bency, Ahmed Qureshi, and Michael Yip. Neural path planning: Fixed time, near-optimal path generation via oracle imitation. 04 2019.

[21] Subhrajit Bhattacharya, Maxim Likhachev, and Vijay Kumar. Identification and representation of homotopy classes of trajectories for search-based path planning in 3d. In *Proceedings of Robotics: Science and Systems*, Los Angeles, CA, USA, June 2011.

[22] Francesco Biral, Enrico Bertolazzi, and Paolo Bosetti. Notes on numerical methods for solving optimal control problems. *IEEJ Journal of Industry Applications*, 5(2):154–166, 2016.

[23] L. Blackmore, M. Ono, and B. C. Williams. Chance-constrained optimal path planning with obstacles. *IEEE Transactions on Robotics*, 27(6):1080–1094, Dec 2011.

[24] M. Blaich, S. Weber, J. Reuter, and A. Hahn. Motion safety for vessels: An approach based on inevitable collision states. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1077–1082, Sept 2015.

[25] J. Borenstein and Y. Koren. The vector field histogram-fast obstacle avoidance for mobile robots. *Robotics and Automation, IEEE Transactions on*, 7(3):278–288, Jun 1991.

[26] B. Braginsky and H. Guterman. Trajectory controller for autonomous surface vehicle under sea waves. In *OCEANS 2015 - MTS/IEEE Washington*, pages 1–5, Oct 2015.

[27] O. Brock, J. Trinkle, and F. Ramos. *Planning Long Dynamically-Feasible Maneuvers for Autonomous Vehicles*, pages 214–221. MIT Press, 2009.

[28] Oliver Brock and Lydia E Kavraki. Decomposition-based motion planning: A framework for real-time motion planning in high-dimensional configuration spaces. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 2, pages 1469–1474, 2001.

[29] A. Bry and N. Roy. Rapidly-exploring random belief trees for motion planning under uncertainty. In *2011 IEEE International Conference on Robotics and Automation*, pages 723–730, May 2011.

[30] F. Caccavale and B. Siciliano. Quaternion-based kinematic control of redundant spacecraft/manipulator systems. In *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, volume 1, pages 435–440 vol.1, May 2001.

[31] A. Censi, D. Calisi, A. De Luca, and G. Oriolo. A bayesian framework for optimal motion planning with uncertainty. In *2008 IEEE International Conference on Robotics and Automation*, pages 1798–1805, May 2008.

[32] Chao Chen. *Motion Planning for Nonholonomic Vehicles with Space Exploration Guided Heuristic Search*. Dissertation, Technische Universitt Mnchen, Mnchen, 2016.

[33] Chao Chen, Markus Rickert, and Alois Knoll. Motion planning under perception and control uncertainties with Space Exploration Guided Heuristic Search. *IEEE Intelligent Vehicles Symposium, Proceedings*, (Iv):712–718, 2017.

[34] S. Choudhury, J. D. Gammell, T. D. Barfoot, S. S. Srinivasa, and S. Scherer. Regionally accelerated batch informed trees (rabit*): A framework to integrate local information into optimal path planning. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4207–4214, May 2016.

[35] Benjamin J Cohen, Sachin Chitta, and Maxim Likhachev. Search-based Planning for Manipulation with Motion Primitives. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2010.

[36] Benjamin J Cohen, Gokul Subramanian, Sachin Chitta, and Maxim Likhachev. Planning for Manipulation with Adaptive Motion Primitives. In *International Conference on Robotics and Automation*, 2011.

[37] Camilla Colombo, Massimiliano Vasile, and Gianmarco Radice. Optimal low-thrust trajectories to asteroids through an algorithm based on differential dynamic programming. *Celestial mechanics and dynamical astronomy*, 105(1-3):75–112, 2009.

[38] Sébastien Dalibard and Jean-Paul Laumond. Linear dimensionality reduction in random motion planning. *The International Journal of Robotics Research*, 30(12):1461–1476, 2011.

[39] R. D'Andrea. Guest editorial: A revolution in the warehouse: A retrospective on kiva systems and the grand challenges ahead. *IEEE Transactions on Automation Science and Engineering*, 9(4):638–639, Oct 2012.

[40] Kenny Daniel, Alex Nash, Sven Koenig, and Ariel Felner. Theta*: Any-angle path planning on grids. *Journal of Artificial Intelligence Research*, 39:533–579, 2010.

[41] Nikhil Das, Naman Gupta, and Michael Yip. Fastron: An online learning-based model and active learning strategy for proxy collision detection. In Sergey Levine, Vincent Vanhoucke, and Ken Goldberg, editors, *Proceedings of the 1st Annual Conference on Robot Learning*, volume 78 of *Proceedings of Machine Learning Research*, pages 496–504. PMLR, 13–15 Nov 2017.

[42] V. Delsart and T. Fraichard. Tiji, a generic trajectory generation tool for motion planning and control. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1439–1444, Oct 2010.

[43] Jory Denny, Read Sandström, Nicole Julian, and Nancy M Amato. *A Region-Based Strategy for Collaborative Roadmap Construction*, pages 125–141. Springer International Publishing, Cham, 2015.

[44] Jory Denny, Read Sandstrm, and Nancy Amato. *A General Region-Based Framework for Collaborative Planning*, pages 563–579. 01 2018.

[45] Rosen Diankov. Bispace planning: Concurrent multi-space exploration. *Proceedings of Robotics: Science and Systems IV*, 63, 2008.

[46] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numer. Math.*, 1(1):269–271, December 1959.

[47] Noel E. Du Toit and Joel W. Burdick. Robot motion planning in dynamic, uncertain environments. *IEEE Transactions on Robotics*, 28(1):101–115, 2012.

[48] L. E. Dubins. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of Mathematics*, 79(3):497–516, 1957.

[49] M. Elbanhawi and M. Simic. Sampling-based robot motion planning: A review. *IEEE Access*, 2:56–77, 2014.

[50] Aimée Vargas Estrada, Jyh-Ming Lien, and Nancy M Amato. Vizmo++: a visualization, authoring, and educational tool for motion planning. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 727–732. IEEE, 2006.

[51] Paolo Fiorini and Zvi Shiller. Motion planning in dynamic environments using velocity obstacles. *I. J. Robotic Res.*, 17(7):760–772, 1998.

[52] T. Fossen. *Guidance and Control of Ocean Vehicles*. Wiley, New York, NY, 1995.

[53] D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *Robotics Automation Magazine, IEEE*, 4(1):23–33, Mar 1997.

[54] T. Fraichard. A short paper about motion safety. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 1140–1145, April 2007.

[55] T. Fraichard and H. Asama. Inevitable collision states. a step towards safer robots? In *Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 1, pages 388–393 vol.1, Oct 2003.

[56] C. Fulgenzi, A. Spalanzani, and C. Laugier. Dynamic obstacle avoidance in uncertain environment combining PVOs and occupancy grid. In *IEEE International Conference on Robotics and Automation, 2007*, pages 1610–1616. IEEE.

[57] C. Fulgenzi, C. Tay, A. Spalanzani, and C. Laugier. Probabilistic navigation in dynamic environment using rapidly-exploring random trees and gaussian processes. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1056–1062, Sept 2008.

[58] J. D. Gammell, T. D. Barfoot, and S. S. Srinivasa. Informed sampling for asymptotically optimal path planning. *IEEE Transactions on Robotics*, pages 1–19, 2018.

[59] Jonathan D. Gammell, Siddhartha S. Srinivasa, and Timothy D. Barfoot. Batch Informed Trees (BIT): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs. In *IEEE International Conference on Robotics and Automation*, volume 2015-June, pages 3067–3074, 2015.

[60] Kalin Gochev, Benjamin Cohen, Jonathan Butzke, Alla Safonova, and Maxim Likhachev. Path Planning with Adaptive Dimensionality. In *The Symposium on Combinatorial Search (SoCS)*, 2011.

[61] Kalin Gochev, Venkatraman Narayanan, Benjamin Cohen, Alla Safonova, and Maxim Likhachev. Motion Planning for Robotic Manipulators with Independent Wrist Joints. In *International Conference on Robotics and Automation*, 2014.

[62] Kalin Gochev, Alla Safonova, and Maxim Likhachev. Planning with adaptive dimensionality for mobile manipulation. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 2944–2951, 2012.

[63] Palash Goyal, Sujit Rokka Chhetri, and Arquimedes Canedo. dyngraph2vec: Capturing network dynamics using dynamic graph representation learning, 2018.

[64] Palash Goyal and Emilio Ferrara. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, 151:78–94, 2018.

[65] Palash Goyal, Nitin Kamra, Xinran He, and Yan Liu. Dyngem: Deep embedding method for dynamic graphs. *arXiv preprint arXiv:1805.11273*, 2018.

[66] M. Greytak and F. Hover. Motion planning with an analytic risk cost for holonomic vehicles. In *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*, pages 5655–5660, Dec 2009.

[67] Eric Guizzo. Three engineers, hundreds of robots, one warehouse. *IEEE spectrum*, 45(7):26–34, 2008.

[68] Li Han. A kinematics-based probabilistic roadmap method for closed chain systems. In *International Workshop on Algorithmic Foundations of Robotics (WAFR*. Citeseer, 2000.

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

[70] K. Hauser and V. Ng-Thow-Hing. Fast smoothing of manipulator trajectories using optimal bounded-acceleration shortcuts. In *2010 IEEE International Conference on Robotics and Automation*, pages 2493–2498, May 2010.

[71] T. Hilmer and E. Thornhill. Observations of predictive skill for real-time deterministic sea waves from the WaMoS II. In *OCEANS 2015 - MTS/IEEE Washington*, pages 1–7, Oct 2015.

[72] Thomas M Howard and Alonzo Kelly. Optimal rough terrain trajectory generation for wheeled mobile robots. *The International Journal of Robotics Research*, 26(2):141–166, 2007.

[73] T.M. Howard and A. Kelly. Trajectory and spline generation for all-wheel steering mobile robots. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 4827–4832, Oct 2006.

[74] Jinwook Huh and Daniel D Lee. Efficient sampling withq-learning to guide rapidly exploring random trees. *IEEE Robotics and Automation Letters*, 3(4):3868–3875, 2018.

[75] Karl Iagnemma, Shingo Shimoda, and Zvi Shiller. Near-optimal navigation of high speed mobile robots on uneven terrain. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4098–4103. IEEE, 2008.

[76] Brian Ichter and Marco Pavone. Robot motion planning in learned latent spaces. *CoRR*, abs/1807.10366, 2018.

[77] Fahad Islam, Oren Salzman, and Maxim Likhachev. Online, interactive user guidance for high-dimensional, constrained motion planning. In *IJCAI*, 2018.

[78] S. Guy D. Manocha J. van den Berg, J. Snape. Reciprocal collision avoidance with acceleration-velocity obstacles. In *IEEE Int. Conf. on Robotics and Automation*, 2011.

[79] Lucas Janson, Edward Schmerling, and Marco Pavone. Monte carlo motion planning for robot trajectory optimization under uncertainty. *CoRR*, abs/1504.08053, 2015.

[80] Derek Jung and Kamal K. Gupta. Octree-based hierarchical distance maps for collision detection. *Proceedings of IEEE International Conference on Robotics and Automation*, 1:454–459 vol.1, 1996.

[81] A. M. Kabir, J. D. Langsfeld, S. Shriyam, V. S. Rachakonda, C. Zhuang, K. N. Kaipa, J. Marvel, and S. K. Gupta. Planning algorithms for multi-setup multi-pass robotic cleaning with oscillatory moving tools. In *2016 IEEE International Conference on Automation Science and Engineering (CASE)*, pages 751–757, Fort Worth, Texas, USA, Aug 2016.

[82] A. M. Kabir, B. C. Shah, and S. K. Gupta. Trajectory planning for manipulators operating in confined workspaces. In *2018 IEEE International Conference on Automation Science and Engineering (CASE)*, Munich, Germany, Aug 2018.

[83] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1):99 – 134, 1998.

[84] Mrinal Kalakrishnan, Sachin Chitta, Evangelos Theodorou, Peter Pastor, and Stefan Schaal. STOMP: Stochastic Trajectory Optimization for Motion Planning. In *International Conference on Robotics and Automation*, 2011.

[85] H. Kang and F. C. Park. Configuration space learning for constrained manipulation tasks using gaussian processes. In *2014 IEEE-RAS International Conference on Humanoid Robots*, pages 1088–1093, Nov 2014.

[86] Sertac Karaman, Matthew Walter, Alejandro Perez, Emilio Frazzoli, and Seth Teller. Anytime Motion Planning Using the Rrt. In *International Conference on Robotics and Automation*, pages 1478–1483, 2011.

[87] Lydia E. Kavraki, Petr Svec, Jean-Paul Laumond, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration space. *IEEE Transactions on Robotics*, 1996.

[88] Alonzo Kelly and Bryan Nagy. Reactive nonholonomic trajectory generation via parametric optimal control. *The International Journal of Robotics Research*, 22(7-8):583–601, 2003.

[89] Oussama Khatib. Real-Time Obstacle Avoidance for Manipulators and Mobile Robots. *The International Journal of Robotics Research*, 1986.

[90] Scott Kiesel, Tianyi Gu, and Wheeler Ruml. An effort bias for sampling-based motion planning. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pages 2864–2871. IEEE, 2017.

[91] Donghyuk Kim, Youngsun Kwon, and Sung-Eui Yoon. Dancing prm: Simultaneous planning of sampling and optimization with configuration free space approximation. In *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE Robotics and Automation Society, 2018.

[92] Kihwan Kim, Dongryeol Lee, and I. Essa. Gaussian process regression flow for analysis of motion trajectories. In *2011 International Conference on Computer Vision*, pages 1164–1171, Nov 2011.

[93] Wilhelm B Klinger, Ivan R Bertaska, Karl D von Ellenrieder, and Manhar R Dhanak. Control of an unmanned surface vehicle with uncertain displacement and drag. *IEEE Journal of Oceanic Engineering*, 42(2):458–476, 2016.

[94] Ross A Knepper, Siddhartha S Srinivasa, and Matthew T Mason. Toward a deeper understanding of motion alternatives via an equivalence relation on local paths. *The International Journal of Robotics Research*, 31(2):167–186, 2012.

[95] James J Kuffner and Steven M LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 2, pages 995–1001. IEEE, 2000.

[96] James J Kuffner and Steven M LaValle. RRT-connect: An efficient approach to single-query path planning. In *IEEE International Conference on Robotics and Automation (ICRA).*, volume 2, pages 995–1001. IEEE, 2000.

[97] Yen-Ling Kuo, Andrei Barbu, and Boris Katz. Deep sequential models for sampling-based planning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6490–6497. IEEE, 2018.

[98] Hanna Kurniawati and David Hsu. Workspace-based connectivity oracle: An adaptive sampling strategy for prm planning. *Springer Tracts in Advanced Robotics*, 47:35–51, 01 2006.

[99] Aleksandr Kushleyev and M. Likhachev. Time-bounded lattice for efficient planning in dynamic environments. In *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*, pages 1662–1668, May 2009.

[100] J. G. Kusters, K. L. Cockrell, B. S. H. Connell, J. P. Rudzinsky, and V. J. Vinciullo. Futurewaves: A real-time ship motion forecasting system employing advanced wave-sensing radar. In *OCEANS 2016 MTS/IEEE Monterey*, pages 1–9, Sept 2016.

[101] B. Lacevic, D. Osmankovic, and A. Ademovic. Burs of free c-space: A novel structure for path planning. pages 70–76, May 2016.

[102] B. Lacevic, D. Osmankovic, and A. Ademovic. Path planning using adaptive burs of free configuration space. In *2017 XXVI International Conference on Information, Communication and Automation Technologies (ICAT)*, pages 1–6, Oct 2017.

[103] Bakir Lacevic, Dinko Osmankovic, and Adnan Ademovic. Burs of free c-space: a novel structure for path planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 70–76, 2016.

[104] A. Lambert, D. Gruyer, and G. Saint Pierre. A fast monte carlo algorithm for collision probability estimation. In *2008 10th International Conference on Control, Automation, Robotics and Vision*, pages 406–411, Dec 2008.

[105] Philipp Last, Christian Bahlke, Martin Hering-Bertram, and Lars Linsen. Comprehensive analysis of automatic identification system (ais) data in regard to vessel movement prediction. *Journal of Navigation*, 67(5):791809, 2014.

[106] S. M. LaValle. *Planning algorithms.* Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu.

[107] Steven M LaValle. Rapidly-exploring random trees: A new tool for path planning. 1998.

[108] Steven M LaValle. Rapidly-exploring random trees: A new tool for path planning. 1998.

[109] Steven M LaValle. From dynamic programming to rrts: Algorithmic design of feasible trajectories. In *Control Problems in Robotics*, pages 19–37. Springer, 2003.

[110] Steven M. LaValle, James J. Kuffner, and Jr. Rapidly-exploring random trees: Progress and prospects, 2000.

[111] Duong Le and Erion Plaku. Guiding sampling-based tree search for motion planning with dynamics via probabilistic roadmap abstractions. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 212–217, 2014.

[112] J. Lee and S. Yoon. Prot: Productive regions oriented task space path planning for hyper-redundant manipulators. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6491–6498, May 2014.

[113] S Lefèvre. A survey on motion prediction and risk assessment for intelligent vehicles. *Robomech ...*, pages 1–14, 2014.

[114] Anastasios M Lekkas. *Guidance and Path-Planning Systems for Autonomous Vehicles.* Number April. 2014.

[115] D. Lenz, M. Rickert, and A. Knoll. Heuristic search in belief space for motion planning under uncertainties. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2659–2665, Sept 2015.

[116] Lihong Li, Thomas J. Walsh, and Michael L. Littman. Towards a unified theory of state abstraction for mdps. In *ISAIM*, 2006.

[117] Xiaohui Li, Zhenping Sun, Qi Zhu, and Daxue Liu. A unified approach to local trajectory planning and control for autonomous driving along a reference path. In *2014 IEEE International Conference on Mechatronics and Automation*, pages 1716–1721. IEEE, 2014.

[118] Maxim Likhachev and Dave Ferguson. Planning long dynamically feasible maneuvers for autonomous vehicles. *Int. J. Rob. Res.*, 28(8):933–945, August 2009.

[119] Maxim Likhachev, David I Ferguson, Geoffrey J Gordon, Anthony Stentz, and Sebastian Thrun. Anytime dynamic A*: An anytime, replanning algorithm. In *ICAPS*, pages 262–271, 2005.

[120] Lantao Liu and Gaurav S Sukhatme. A Solution to Time-Varying Markov Decision Processes. 2018.

[121] S. Liu, M. Watterson, K. Mohta, K. Sun, S. Bhattacharya, C. J. Taylor, and V. Kumar. Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3-d complex environments. *IEEE Robotics and Automation Letters*, 2(3):1688–1695, July 2017.

[122] Y. Liu, R. Song, and R. Bucknall. A practical path planning and navigation algorithm for an unmanned surface vehicle using the fast marching algorithm. In *OCEANS 2015 - Genova*, pages 1–7, May 2015.

[123] Marco Pavone Lucas Janson, Edward Schmerling, Ashley Clark. Fast Marching Tree: a Fast Marching Sampling-Based Method for Optimal Motion Planning in Many Dimensions. *International Journal of Robotics Research*, 34(7):883–921, 2015.

[124] S. Mahdavi, S. Khoshraftar, and A. An. dynnode2vec: Scalable dynamic network embedding. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 3762–3765, Dec 2018.

[125] Rishi K. Malhan, Ariyan M. Kabir, Brual C. Shah, and Satyandra K. Gupta. Identifying feasible workpiece placement with respect to redundant manipulator for complex manufacturing tasks. In *IEEE International Conference on Robotics and Automation (ICRA)*, Montreal, Canada, May 2019.

[126] James D Marble and Kostas E Bekris. Asymptotically near-optimal is good enough for motion planning. In *Robotics Research*, pages 419–436. Springer, 2017.

[127] L. Martinez-Gomez and T. Fraichard. Collision avoidance in dynamic environments: An ics-based solution and its comparative evaluation. In *2009 IEEE International Conference on Robotics and Automation*, pages 100–105, May 2009.

[128] Kayla Matthews. 6 industries that have been improved by robotic automation.

[129] Troy McMahon, Shawna Thomas, and Nancy M Amato. Sampling-based motion planning with reachable volumes for high-degree-of-freedom manipulators. *The International Journal of Robotics Research*, 37(7):779–817, 2018.

[130] Siddhartha Mehta, Cuong Ton, Michael Mccourt, Zhen Kan, E.A. Doucette, and W Curtis. Human-assisted rrt for path planning in urban environments. pages 941–946, 10 2015.

[131] D. Mellinger and V. Kumar. Control and planning for vehicles with uncertainty in dynamics. In *2010 IEEE International Conference on Robotics and Automation*, pages 960–965, May 2010.

[132] Arjun Menon, Benjamin Cohen, and Maxim Likhachev. Motion Planning for Smooth Pickup of Moving Objects. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[133] George Mesesan, Mximo Alejandro Roa Garzon, Esra er, and Matthias Althoff. Hierarchical path planner using workspace decomposition and parallel task-space rrts. 10 2018.

[134] J. Minguez and L. Montano. Nearness diagram navigation (nd): a new real time collision avoidance approach. In *Intelligent Robots and Systems, 2000. (IROS 2000). Proceedings. 2000 IEEE/RSJ International Conference on*, volume 3, pages 2094–2100 vol.3, 2000.

[135] Wasif Naeem, George W. Irwin, and Aolei Yang. Colregs-based collision avoidance strategies for unmanned surface vehicles. *Mechatronics*, 22(6):669 – 678, 2012.

[136] Jauwairia Nasir, Fahad Islam, Usman Malik, Yasar Ayaz, Osman Hasan, Mushtaq Khan, and Mannan Saeed Muhammad. Rrt*-smart: A rapid convergence implementation of rrt* regular paper. *International Journal of Advanced Robotic Systems*, 2013.

[137] Teck Chew Ng, J. Ibanez-Guzman, Jian Shen, Zhiming Gong, Han Wang, and Chen Cheng. Vehicle following with obstacle avoidance capabilities in natural environments. In *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, volume 5, pages 4283–4288 Vol.5, April 2004.

[138] Iram Noreen, Amna Khan, and Zulfiqar Habib. Optimal path planning using rrt* based approaches: a survey and future directions. *Int. J. Adv. Comput. Sci. Appl*, 7:97–107, 2016.

[139] Dinko Osmankovic and Bakir Lacevic. Rapidly exploring bur trees for optimal motion planning. pages 002085–002090, 2016.

[140] M. Otte, W. Silva, and E. Frew. Any-time path-planning: Time-varying wind field + moving obstacles. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2575–2582, May 2016.

[141] J. Pan, L. Zhang, and D. Manocha. Retraction-based rrt planner for articulated models. In *2010 IEEE International Conference on Robotics and Automation*, pages 2529–2536, May 2010.

[142] Zherong Pan, Chonhyon Park, and Dinesh Manocha. Robot Motion Planning for Pouring Liquids. *International Conference on Automated Planning and Scheduling*, (Icaps):518–526, 2016.

[143] J. Park and J. Kim. Predictive evaluation of ship collision risk using the concept of probability flow. *IEEE Journal of Oceanic Engineering*, 42(4):836–845, Oct 2017.

[144] Nicola Pedrocchi, Federico Vicentini, Malosio Matteo, and Lorenzo Molinari Tosatti. Safe human-robot cooperation in an industrial environment. *International Journal of Advanced Robotic Systems*, 10(1):27, 2013.

[145] Thierry Peynot, Sin-Ting Lui, Rowan McAllister, Robert Fitch, and Salah Sukkarieh. Learned stochastic mobility prediction for planning with control uncertainty on unstructured terrain. *J. Field Robotics*, 31:969–995, 2014.

[146] Mike Phillips, Benjamin Cohen, Sachin Chitta, and Maxim Likhachev. E-graphs: Bootstrapping planning with experience graphs. In *In Proceedings of the Robotics: Science and Systems Conference (RSS 2012)*, July 2012.

[147] Vinay Pilania and Kamal Gupta. A hierarchical and adaptive mobile manipulator planner. In *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 45–51. IEEE, 2014.

[148] Vinay Pilania and Kamal Gupta. A hierarchical and adaptive mobile manipulator planner with base pose uncertainty. *Autonomous Robots*, 39(1):65–85, 2015.

[149] Vinay Pilania and Kamal Gupta. Mobile manipulator planning under uncertainty in unknown environments. *The International Journal of Robotics Research*, 37(2-3):316–339, 2018.

[150] Erion Plaku, Lydia E Kavraki, and Moshe Y Vardi. Discrete Search Leading Continuous Exploration for Kinodynamic Motion Planning. In *Robotics: Science and Systems*, 2007.

[151] Oliver Purwin and Raffaello DAndrea. Trajectory generation and control for four wheeled omnidirectional vehicles. *Robotics and Autonomous Systems*, 54(1):13–22, 2006.

[152] Sean Quinlan. *Real-time modification of collision-free paths*. Number 1537. Stanford University Stanford, 1994.

[153] Ana Huamán Quispe and Mike Stilman. Deterministic motion planning for redundant robots along end-effector paths. In *Humanoid Robots (Humanoids), 2012 12th IEEE-RAS International Conference on*, pages 785–790. IEEE, 2012.

[154] Ahmed H. Qureshi and Michael C. Yip. Deeply informed neural sampling for robot motion planning. *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6582–6588, 2018.

[155] Mohammadhussein Rafieisakhaei, Amirhossein Tamjidi, Suman Chakravorty, and P. R. Kumar. Feedback motion planning under non-gaussian uncertainty and non-convex state constraints. *CoRR*, abs/1511.05186, 2015.

[156] Pradeep Rajendran, Brual C. Shah, and Satyandra K. Gupta. Dynamics-Aware Reactive Planning for Unmanned Ground Vehicles to Avoid Collisions with Dynamic Obstacles on Uneven Terrains. In *Workshop on Planning and Robotics (PlanRob), held at International Conference on Automated Planning and Scheduling (ICAPS' 17), Pittsburgh, PA, June 19 - 20, 2017*, 2017.

[157] J. H. Reif. Complexity of the mover's problem and generalizations. In *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*, pages 421–427, Oct 1979.

[158] M. Rickert, A. Sieverling, and O. Brock. Balancing exploration and exploitation in sampling-based motion planning. *IEEE Transactions on Robotics*, 30(6):1305–1317, Dec 2014.

[159] Markus Rickert, Oliver Brock, and Alois Knoll. Balancing exploration and exploitation in motion planning. In *IEEE International Conference on Robotics and Automation (ICRA).*, pages 2812–2817, 2008.

[160] Samuel Rodriguez, Shawna Thomas, Roger Pearce, and Nancy M Amato. Resampl: A region-sensitive adaptive motion planner. In *Algorithmic Foundation of Robotics VII*, pages 285–300. Springer, 2008.

[161] Nicholas Roy and Sebastian Thrun. Coastal navigation with mobile robots. In *Advances in Neural Information Processing Systems*, pages 1043–1049, 2000.

[162] Gildardo Sánchez and Jean-Claude Latombe. On delaying collision checking in prm planning: Application to multi-robot coordination. *The International Journal of Robotics Research*, 21(1):5–26, 2002.

[163] Edoardo I. Sarda, Huajin Qu, Ivan R. Bertaska, and Karl D. von Ellenrieder. Station-keeping control of an unmanned surface vehicle exposed to current and wind disturbances. *CoRR*, abs/1702.04941, 2017.

[164] Edward Schmerling and Marco Pavone. Evaluating trajectory collision probability through adaptive importance sampling for safe motion planning. *CoRR*, abs/1609.05399, 2016.

[165] Jonathan Scholz and Mike Stilman. Combining motion planning and optimization for flexible robot manipulation. *2010 10th IEEE-RAS International Conference on Humanoid Robots, Humanoids 2010*, 12 2010.

[166] Brual Shah and Satyandra K. Gupta. Speeding up A* search on visibility graphs defined over quadtrees to enable long distance path planning for unmanned surface vehicles. In *Internation Conference on Automated Planning and Scheduling (ICAPS' 16), London, UK, June 12 - 17, 2016*, 2016.

[167] Brual C Shah, Petr Švec, Ivan R Bertaska, Armando J Sinisterra, Wilhelm Klinger, Karl von Ellenrieder, Manhar Dhanak, and Satyandra K Gupta. Resolution-adaptive risk-aware trajectory planning for surface vehicles operating in congested civilian traffic. *Autonomous Robots*, pages 1–25, 2015.

[168] Brual C. Shah, Petr Švec, Ivan R. Bertaska, Armando J. Sinisterra, Wilhelm Klinger, Karl von Ellenrieder, Manhar Dhanak, and Satyandra K. Gupta. Resolution-adaptive risk-aware trajectory planning for surface vehicles operating in congested civilian traffic. *Autonomous Robots*, 40(7):1139–1163, Oct 2016.

[169] M. Sheckells, T. M. Caldwell, and M. Kobilarov. Fast approximate path coordinate motion primitives for autonomous driving. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 837–842, Dec 2017.

[170] K. Shi, J. Denny, and N. M. Amato. Spark prm: Using rrts within prms to efficiently explore narrow passages. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4659–4666, May 2014.

[171] Z. Shiller, F. Large, and S. Sekhavat. Motion planning in dynamic environments: obstacles moving along arbitrary trajectories. In *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, volume 4, pages 3716–3721 vol.4, 2001.

[172] S. Shimoda, Y. Kuroda, and K. Iagnemma. Potential field navigation of high speed unmanned ground vehicles on uneven terrain. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 2828–2833, April 2005.

[173] A. Shkolnik and R. Tedrake. Path planning in 1000+ dimensions using a task-space voronoi bias. In *2009 IEEE International Conference on Robotics and Automation*, pages 2061–2067, May 2009.

[174] Alexander Shkolnik and Russ Tedrake. Sample-based planning with volumes in configuration space. *arXiv preprint arXiv:1109.3145*, 2011.

[175] Shaurya Shriyam and Satyandra K. Gupta. Task assignment and scheduling for mobile robot teams. In *ASME International Design Engineering Technical Conferences Computers and Information in Engineering Conference (IDETC/CIE)*, Quebec City, Canada, Aug 2018.

[176] David Silver and Joel Veness. Monte-carlo planning in large pomdps. In *Advances in neural information processing systems*, pages 2164–2172, 2010.

[177] Rui Song, Yuanchang Liu, and Richard Bucknall. A multi-layered fast marching method for unmanned surface vehicle path planning in a time-variant maritime environment. *Ocean Engineering*, 129:301 – 317, 2017.

[178] Matthew Spenko, Yoji Kuroda, Steven Dubowsky, and Karl Iagnemma. Hazard avoidance for high-speed mobile robots in rough terrain. *Journal of Field Robotics*, 23(5):311–331, 2006.

[179] Matthew Spenko, Yoji Kuroda, Steven Dubowsky, and Karl Iagnemma. Hazard avoidance for high-speed mobile robots in rough terrain. *Journal of Field Robotics*, 23(5):311–331, 2006.

[180] Stephen B. Stancliff, John M. Dolan, and Ashitey Trebi-Ollennu. Towards a predictive model of mobile robot reliability. Technical Report CMU-RI-TR-05-38, Pittsburgh, PA, August 2005.

[181] M. Stilman. Global manipulation planning in robot joint space with task constraints. *IEEE Transactions on Robotics*, 26(3):576–584, June 2010.

[182] Mike Stilman. Task constrained motion planning in robot joint space. *IEEE International Conference on Intelligent Robots and Systems (ICRA)*, pages 3074–3081, 2007.

[183] Daniel Strawser and Brian Williams. Approximate Branch and Bound for Fast , Risk-Bound Stochastic Path Planning. pages 7047–7054, 2018.

[184] P. Svec, M. Schwartz, A. Thakur, and S. K. Gupta. Trajectory planning with look-ahead for unmanned sea surface vehicles to handle environmental disturbances. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1154–1159, Sept 2011.

[185] P. Svec, B. C. Shah, I. R. Bertaska, J. Alvarez, A. J. Sinisterra, K. von Ellenrieder, M. Dhanak, and S. K. Gupta. Dynamics-aware target following for an autonomous surface vehicle operating under colregs in civilian traffic. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3871–3878, Nov 2013.

[186] Petr Svec, Brual C. Shah, Ivan R. Bertaska, Wilhelm Klinger, Armando J. Sinisterra, Karl von Ellenrieder, Manhar Dhanak, and Satyandra K. Gupta. Adaptive sampling based colregs-compliant obstacle avoidance for autonomous surface vehicles. 2014.

[187] Adnan Tahirovic and Mina Ferizbegovic. Rapidly-exploring Random Vines (RRV) for Motion Planning in Configuration Spaces with Narrow Passages. pages 7055–7062, 2018.

[188] Michel Taïx, David Flavigné, and Etienne Ferré. Human interaction with motion planning algorithm. *Journal of Intelligent & Robotic Systems*, 67(3):285–306, Sep 2012.

[189] CheeKuang Tam and Richard Bucknall. Cooperative path planning algorithm for marine surface vessels. *Ocean Engineering*, 57:25 – 33, 2013.

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

[191] Shantanu Thakar, Liwei Fang, Brual Shah, and Satyandra Gupta. Towards time-optimal trajectory planning for pick-and-transport operation with a mobile manipulator. pages 981–987, 08 2018.

[192] Shantanu Thakar, Ariyan Kabir, Prahar Bhatt, Rishi Malhan, Pradeep Rajendran, Brual Shah, and Satyandra K. Gupta. Task assignment and motion planning for bi-manual mobile manipulation. In *IEEE International Conference on Automation Science and Engineering (CASE)*, Vancouver, Canada, August 2019.

[193] Shantanu Thakar, Pradeep Rajendran, Vivek Annem, Ariyan Kabir, and Satyandra K. Gupta. Accounting for part pose estimation uncertainties during trajectory generation for part pick-up using mobile manipulators. In *IEEE International Conference on Robotics and Automation (ICRA)*, Montreal, Canada, May 2019.

[194] Atul Thakur, Petr Svec, and Satyandra K. Gupta. GPU based generation of state transition models using simulations for unmanned surface vehicle trajectory planning. *Robotics and Autonomous Systems*, 60(12):1457 – 1471, 2012.

[195] M. Tiger and F. Heintz. Online sparse gaussian process regression for trajectory modeling. In *2015 18th International Conference on Information Fusion (Fusion)*, pages 782–791, July 2015.

[196] N. E. Du Toit and J. W. Burdick. Probabilistic collision checking with chance constraints. *IEEE Transactions on Robotics*, 27(4):809–815, Aug 2011.

[197] Rakshit Trivedi, Mehrdad Farajtbar, Prasenjeet Biswal, and Hongyuan Zha. Representation learning over dynamic graphs. *arXiv preprint arXiv:1803.04051*, 2018.

[198] E. Tu, G. Zhang, L. Rachmawati, E. Rajabally, and G. B. Huang. Exploiting ais data for intelligent maritime navigation: A comprehensive survey from data to methodology. *IEEE Transactions on Intelligent Transportation Systems*, PP(99):1–24, 2017.

[199] V. Turri, A. Carvalho, H. E. Tseng, K. H. Johansson, and F. Borrelli. Linear model predictive control for lane keeping and obstacle avoidance on low curvature roads. In *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*, pages 378–383, Oct 2013.

[200] V. V. Unhelkar, J. Perez, J. C. Boerkoel, J. Bix, S. Bartscher, and J. A. Shah. Towards control and sensing for an autonomous mobile robotic assistant navigating assembly lines. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4161–4167, May 2014.

[201] J. P. van den Berg and M. H. Overmars. Using workspace information as a guide to non-uniform sampling in probabilistic roadmap planners. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, volume 1, pages 453–460 Vol.1, April 2004.

[202] Jur Van Den Berg, Pieter Abbeel, and Ken Goldberg. Lqg-mp: Optimized path planning for robots with motion uncertainty and imperfect state information. *The International Journal of Robotics Research*, 30(7):895–913, 2011.

[203] Jur van den Berg, Sachin Patil, and Ron Alterovitz. *Motion Planning Under Uncertainty Using Differential Dynamic Programming in Belief Space*, volume 100, pages 473–490. 01 2017.

[204] C W Warren, J C Danos, and B W Mooring. An Approach To Manipulator Path Planning. *Van Nostrand Reinhold. Schutz, B. Geometrical Methods of Mathematical Physics*, 1981.

[205] Mike Vande Weghe, Dave Ferguson, and Siddhartha S Srinivasa. Randomized path planning for redundant manipulators without inverse kinematics. In *Humanoid Robots, 2007 7th IEEE-RAS International Conference on*, pages 477–482. IEEE, 2007.

[206] M. Werling, J. Ziegler, S. Kammel, and S. Thrun. Optimal trajectory generation for dynamic street scenarios in a frenet frame. In *2010 IEEE International Conference on Robotics and Automation*, pages 987–993, May 2010.

[207] Moritz Werling, Sören Kammel, Julius Ziegler, and Lutz Gröll. Optimal trajectories for time-critical street scenarios using discretized terminal manifolds. *The International Journal of Robotics Research*, 31(3):346–359, 2012.

[208] D. Wilkie, J. van den Berg, and D. Manocha. Generalized velocity obstacles. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 5573–5578, Oct 2009.

[209] D. Wilkie, J. van den Berg, and D. Manocha. Generalized velocity obstacles. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 5573–5578, Oct 2009.

[210] Wenda Xu, Junqing Wei, J. M. Dolan, Huijing Zhao, and Hongbin Zha. A real-time motion planner with trajectory optimization for autonomous vehicles. In *2012 IEEE International Conference on Robotics and Automation*, pages 2061–2067, May 2012.

[211] Zhenwang Yao and Kamal Gupta. Path planning with general end-effector constraints: Using task space to guide configuration space search. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2211–2216, 2005.

[212] Nan Ye, Adhiraj Somani, David Hsu, and Wee Sun Lee. DESPOT: online POMDP planning with regularization. *CoRR*, abs/1609.03250, 2016.

[213] Anna Yershova, Léonard Jaillet, Thierry Siméon, and Steven M LaValle. Dynamic-domain rrts: Efficient exploration by controlling the sampling domain. In *IEEE International Conference on Robotics and Automation (ICRA). Proceedings of the 2005*, pages 3856–3861, 2005.

[214] D. Yi, M. A. Goodrich, and K. D. Seppi. Homotopy-aware rrt*: Toward human-robot topological path-planning. In *2016 11th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 279–286, March 2016.

[215] Byunghyun Yoo and Jinwhan Kim. Path optimization for marine vehicles in ocean currents using reinforcement learning. *Journal of Marine Science and Technology*, 21(2):334–343, Jun 2016.

[216] Xiaowen Yu, Yu Zhao, Cong Wang, and Masayoshi Tomizuka. Trajectory planning for robot manipulators considering kinematic constraints using probabilistic roadmap approach. *Journal of Dynamic Systems, Measurement, and Control*, 139(2):021001, 2017.

[217] Fang Yuan, Jia-Hong Liang, Yue-Wen Fu, Han-Cheng Xu, and Ke Ma. A hybrid sampling strategy with optimized probabilistic roadmap method. In *12th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, pages 2298–2302, 2015.

[218] Liangjun Zhang and Dinesh Manocha. An efficient retraction-based rrt planner. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3743–3750, 2008.

[219] M. Zucker, J. Kuffner, and J. A. Bagnell. Adaptive workspace biasing for sampling-based planners. In *2008 IEEE International Conference on Robotics and Automation*, pages 3757–3762, May 2008.

[220] Matt Zucker, James Kuffner, and J Andrew Bagnell. Adaptive workspace biasing for sampling-based planners. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3757–3762, 2008.

[221] Matt Zucker, Nathan Ratliff, Anca D Dragan, Mihail Pivtoraiko, Matthew Klingensmith, Christopher M Dellin, J Andrew Bagnell, Siddhartha S Srinivasa, Pittsburgh Pa, and Philadelphia Pa. CHOMP: Covariant Hamiltonian Optimization for Motion Planning. *The International Journal of Robotics Research*, 2012.