Designing Integrated Strategies for Modularized Robotic Systems in Uncertain Environments

by

Jessica En Shiuan Leu

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Engineering - Mechanical Engineering

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Masayoshi Tomizuka, Chair
Professor Francesco Borrelli
Professor Claire Tomlin

Spring 2022

Designing Integrated Strategies for Modularized Robotic Systems in Uncertain Environments

Abstract

Designing Integrated Strategies for Modularized Robotic Systems in Uncertain Environments

by

Jessica En Shiuan Leu

Doctor of Philosophy in Engineering - Mechanical Engineering

University of California, Berkeley

Professor Masayoshi Tomizuka, Chair

Historically, robots have successfully performed various tasks in isolated areas by following preprogrammed commands. However, more and more potential robotic applications require robots to complete tasks alongside or in collaboration with other agents such as other robots and human workers. Such robots need algorithms that enable flexible behaviors in crowded and uncertain environments, e.g., to plan actions safely and efficiently in shared spaces where other agents are present. Consequently, researchers have proposed modularized robotic systems, which typically consist of perception, prediction, planning, and control modules. However, most of these works only emphasize the design of a single module and use off-the-shelf methods directly for other modules. Therefore, the improvements made by the re-designed module may have limited value to the overall robotic system; the off-the-shelf methods may not be able to provide adequate information or fully utilize the information provided by the re-designed module. Robot modules should be designed for better performance individually and collectively; thus, this dissertation aims to develop integrated designs and strategies for modularized robotic systems in uncertain environments. Furthermore, the computational efficiency of the robotic system is crucial to the robot's performance in uncertain environments; we address the improvement of the computational speed of the most time-consuming module, the planner.

This dissertation consists of two parts. Part I involves a fundamental exploration of robot motion planners and proposes hybrid motion planners that combine and utilize different planning methods for better computational speed and plan quality, such as travel distance. Chapter 2 combines a sampling-based algorithm, RRT*, and an optimization-based algorithm, the convex feasible set algorithm (CFS). Chapter 3 focuses on long-horizon planning problems; it combines RRT*, CFS, the Interior Point OPTimizer, and segmented trajectory optimization. Chapter 4 focuses on motion planning for articulated vehicles; it combines a search-based algorithm, improved A-search guided tree, and utilizes results from rein-

forcement learning to guide the search. Simulation results demonstrate the advantage of the proposed motion planners in terms of computational speed and plan quality in static and deterministic environments cluttered with obstacles. Chapter 5 studies motion planning in dynamic environments and presents a hierarchical receding horizon control (HRHC) framework. The HRHC coordinates a motion planner, such as the planners presented in Chapters 2-4, with a safety controller to achieve safe and efficient robot motion in uncertain environments in simulations and real-world experiments.

Part II presents application-oriented integrated robotic systems that coordinate the predictor and the planner to make an effective and safe plan. Chapter 6 discusses the close relationship between the prediction and planning modules and identifies several conditions for realizing safe model predictive control in dynamic and uncertain environments; we present a predictor designed for better closed-loop robot performance. Simulations and real-world experiments that involve a robot working alongside a human worker are conducted; the robot can navigate safely in the presence of unexpected human movements. In Chapter 7, simulations with a computer assembly setting that involve a robot collaborating with a human worker are conducted; the proposed robotic system coordinates the prediction and planning modules to utilize human motion prediction and uncertainty estimation for robust task planning. The robot can generate time-efficient task plans when the human worker performs inefficiently. In Chapter 8, simulations involving an autonomous vehicle navigating in a parking lot while avoiding collisions with static and moving obstacles are conducted; the proposed system includes a hybrid environment predictor that makes short-term and long-term predictions of the surroundings and a strategic motion planner that reacts to the environment according to the predictions. The robot demonstrates the effectiveness of the proposed method in terms of motion prediction, safe tracking, retreating in an emergency, and trajectory repairing.

To my family and friends

Contents

Contents **ii**

List of Figures **iv**

List of Tables **vii**

1 Introduction **1**
1.1 Background 1
1.2 Computational Efficiency and Robot Motion Planning 3
1.3 Integrated Strategies of Modularized Robotic Systems 5
1.4 Dissertation Contributions and Outline 6

I Robot Motion Planning **12**

2 Robot Motion Planning and Hybrid Motion Planning **13**
2.1 Introduction 13
2.2 Problem Formulation and Related Works 15
2.3 The RRT*-CFS Algorithm 17
2.4 Applications 19
2.5 Chapter Summary 26

3 Long-horizon Motion Planning **29**
3.1 Introduction 29
3.2 Problem Formulation and Related Works 31
3.3 The RRT*-sOpt Algorithm 32
3.4 Applications 36
3.5 Chapter Summary 48

4 Search-based Motion Planning for Articulated Vehicles **49**
4.1 Introduction 49
4.2 Related Works 50
4.3 Preliminaries 51

4.4 The Off-lattice Motion Planning Algorithm 53
4.5 Applications 64
4.6 Chapter Summary 66

5 Motion Planning in Dynamic Environments 69
5.1 Introduction 69
5.2 Problem Formulation 71
5.3 Hierarchical Receding Horizon Control 73
5.4 Applications 76
5.5 Chapter Summary 77

II Integrated Strategies of Modularized Robotic Systems 82

6 Environment Prediction and Motion Planning 83
6.1 Introduction 83
6.2 Stability of MPC-based Planning 85
6.3 Stability-enhanced Prediction and M-Convergence for Analysis 88
6.4 Applications 92
6.5 Chapter Summary 96

7 Application I: Human-Robot Collaboration for Assembly 100
7.1 Introduction 100
7.2 Related Works 101
7.3 Preliminaries 102
7.4 Robust Task Planning for HRC Applications 104
7.5 Applications 108
7.6 Chapter Summary 114

8 Application II: Autonomous Parking in Uncertain Environments 115
8.1 Introduction 115
8.2 Related Works 116
8.3 Problem Statement and Proposed Architecture 118
8.4 Prediction in Dynamic Parking Environments 119
8.5 Strategic Motion Planner 123
8.6 Applications 126
8.7 Chapter Summary 127

9 Concluding Remarks and Suggested Future Works 131

Bibliography 134

List of Figures

1.1 Applications of robotic systems in uncertain environments. A factory setting (left) where the manipulators, mobile robots, and mobile manipulators need to collaborate with human workers. A parking lot setting (right) where the ego vehicle (blue car) needs to plan for parking while avoiding a moving obstacle vehicle (gray car) and other static vehicles (white cars). The other moving agents' intents and future motion plans are unknown to the ego robot in both settings. To complete a task, the robot will need to perform safely and efficiently under these uncertainties. 3
1.2 Illustration of the modularized robotic system and the environment. 6
1.3 Dissertation outline. In Part I, Chapters 2-4 focus on motion planner developments in static and deterministic environments. Chapter 5 moves to motion planning strategies in uncertain environments. In Part II, Chapters 6-8 further incorporate the prediction module and propose integrated strategies for modularized robotic systems in uncertain environments, while Chapter 7 emphasizes task planning. 7

2.1 A manipulator navigating through the obstacles. 14
2.2 A mobile robot (left) and a manipulator (right). 20
2.3 Illustration of the distance function. 21
2.4 Simulation results of a 2D motion planning. 24
2.5 A simulation result of motion planning in the narrow passage (close to $y = 0.1$) scenario. 24
2.6 Planning results using RRT*-CFS in 5D manipulation planning problems. . . . 27

3.1 A manipulator navigating through a car frame in a factory. 30
3.2 The illustration of a segmented trajectory and the iterative optimization process. 34
3.3 An example of computation time reduced by segment merging in 2D planning. (Notice that we hard-coded this merge to generate this plot for more precise visualization. In the simulations, merging typically occurs in later iterations.) . 35
3.4 Illustration of segment merging. 36
3.5 A mobile manipulator (left) and the coordinate system of the mobile manipulator (right). 37
3.6 Path length reduction performance comparison between RRT* and RRT*-sOpt. 40

3.7 Simulation results of the 2D motion planning. 41
3.8 Simulation results of the 5-Dof manipulator motion planning. 44
3.9 Simulation results of a mobile manipulator motion planning. 47

4.1 Kinematics of a front-drive tractor with 3 trailers. (All trailers are on-axle, and all angles representing the orientation of tractor and trailers ($\theta_i, i = 0, \ldots, 3$) are w.r.t. the x-axis.) 52
4.2 (a) An example of the circular equilibrium configuration. (b) An example of a motion primitive in the original state space $\mathcal{X}$, but starts and ends at the plane of the reduced-state space $\bar{\mathcal{X}}$. 55
4.3 Motion primitive generation: (a) an example of parallel parking maneuver, (b) motion primitives from reversibility and symmetry. The lines indicate the trajectories of the tractor. 56
4.4 Simplification of motion primitive by rotational symmetry. 56
4.5 A trailer driving "forward" (left) and "backward" (right) to the goal configuration (marked in red). 58
4.6 9 poses of $\mathcal{M}_{off}$. 59
4.7 Reachability analysis of $\mathcal{M}_{on}$: (a) adjacency matrix of $\mathcal{M}_{on}$ (b) outdegree of $\mathcal{M}_{on}$. 59
4.8 Reachability analysis of $\mathcal{M}_{off}$: (a) adjacency matrix of $\mathcal{M}_{off}$, and (b) outdegree of $\mathcal{M}_{off}$. 60
4.9 The critic net (left) and actor net (right). 61
4.10 Example of the heuristic value for goal configuration at $[0, 0, 0, 0, 0, 0]^\top$ based on: (b) Euclidean distance, (c) RS path, and (d) SAC value function. 62
4.11 Combining the local heuristic function and the global heuristic function. 63
4.12 Simulation results of case 1-5. 67
4.13 Simulation results of case 6-11. 68

5.1 Distance function $\phi_{m,ij}(z(k))$ and $\phi_{p,j}(z(k))$. 73
5.2 Illustration of the slack variable. 74
5.3 The overall control system. 74
5.4 The hierarchical structure. 75
5.5 Results of collision avoidance in scenario I. 78
5.6 Results of collision avoidance in scenario II. 79
5.7 Result of end-effector position keeping. 80
5.8 Result of avoiding moving human worker. 81

6.1 The execution structure of MPC. 85
6.2 The overall system control design. 90
6.3 Illustration of M-convergence. 92
6.4 Scenario that has oscillating moving obstacles. 93
6.5 Comparison between MPC without stability-enhanced prediction and the proposed MPC with stability-enhanced prediction. 94

6.6 Path planned and implemented in the second scenario. 96
6.7 Path planned and implemented in the third scenario. 97
6.8 Experimental result with mobile robot passing by another robot. 98
6.9 Experimental result with mobile robot passing by a human worker that walks away from the robot after leaving the working area. 98
6.10 Experimental result with mobile robot passing by a human worker that walks along the same direction as the robot for a short period of time after leaving the working area. 99

7.1 The sequential/parallel task model for a desktop assembly task. 103
7.2 The overall system control design. 108
7.3 An illustration of a human worker and a robot performing an assembly task (left) and the simulation setup for the computer assembly. The areas enclosed by the dashed lines are the designated areas for the objects. The mouse controlled by the human is indicated by the green circle (right). 109
7.4 Experimental result of the robot with the baseline planner collaborating with an efficient human worker. (Case 1.) 110
7.5 Experimental result of the robot with the baseline planner (upper row) and the robust task planner (lower row) collaborating with a lazy human worker. (Case 2.) 111
7.6 Experimental results of the robot with the robust task planner collaborating with a slacking human worker. (Case 3.) 112

8.1 The simplified bicycle vehicle model. L is the distance between the axis of the rear wheels and the axis of the front wheels. 118
8.2 The integrated prediction and planning system. 119
8.3 The architecture of the hybrid environment predictor. 120
8.4 The architecture of the cascade motion estimator. 120
8.5 There are 2 routes (black dashed lines) and 2 main modes (red for "cruising" and blue for "maneuvering") in this example, resulting in a total of 4 modes for the OV: 1) cruise(exit) left; 2) maneuver left; 3) cruise(exit) right; 4) maneuver right. 122
8.6 The hybrid predictor predicts a short-term OV trajectory (green line) and use it with mode prediction to generate the safety margins for $h = 1$ and $h = H$ and the safety bound. 123
8.7 The ego AV following the retreating plans (blue star-lines). The light-blue lines illustrate the original trajectory, the red dashed-lines combining the blue star-lines will be the new reference $\mathcal{P}_{ref}$, and the collision field is illustrated by the contours. 125
8.8 The ego AV (red vehicle) following the repaired plan (green lines) that is calculated from the blocked original trajectory (light-blue lines). 126
8.9 Simulation results in one of the parking scenarios. (0.25 s/time step) 129
8.10 The simulation result of steering estimation, mode estimation, and safety margin. 130

List of Tables

2.1 Simulation comparison of 2D planning. (Average of 100 trials.) 23
2.2 Simulation comparison of 2D-narrow-passage planning. (Average of 20 trials.) . 23
2.3 Simulation comparison of 5D Motion planning. (Average of 20 trials.) 28

3.1 Simulation comparison of 2D planning with 5 to 20 obstacles. (The results are the average of 25 trials. The notion "7 $\rightarrow$ 3.9" in # **Segments** in "auto-merge segments" means that the RRT*-sOpt starting from 7 segments on average terminates at 3.9 segments. The computation time standard deviation for RRT*-sOpt only considered the time variation during the optimization stage.) 42
3.2 Simulation comparison of 5-Dof manipulator motion planning. (Average of 20 trials.) 45
3.3 Simulation comparison of mobile manipulator motion planning. (Average of 10 trials.) 46

4.1 Comparison of i-AGT performances with heuristic based on RS and SAC value functions. 65

7.1 Task completion time for each case. 109

Acknowledgments

The past five years at UC Berkeley have been a fantastic journey. To begin with, I want to express my deepest gratitude to my advisor, Professor Masayoshi Tomizuka, who has tremendously supported me throughout my Ph.D. study. His scientific insights helped direct my research pathway, and his enthusiasm for work shaped my attitudes as a student, a researcher, and an engineer. Professor Tomizuka respected my research ideas and inspired me to explore effective solutions for challenging problems. This dissertation would not have been possible without his help and guidance.

I also want to thank Professor Francesco Borrelli and Professor Claire Tomlin for serving on my dissertation committee. Meanwhile, I am thankful to Professor Kameshwar Poolla, Professor Laurent El Ghaoui, Professor Mark Mueller, and Professor Hannah Stuart for serving on my qualifying exam committee.

Special thanks go to National Science Foundation and Mitsubishi Electric Research Laboratories (MERL), the sponsors for the work in this dissertation. The great collaborations and valuable discussions with employees at MERL, including Dr. Yebin Wang and Dr. Stefano Di Cairano, enhance this dissertation by having more diversified robot platforms and more industrial perspectives.

I am extremely grateful to work with excellent graduate students in Mechanical Systems Control (MSC) Laboratory. I would like to especially thank Changliu Liu and Liting Sun for their helpful guidance and extraordinary research collaborations. I also want to express my gratitude to Daisuke Kaneishi, Rachel Lim, Michael Wang, and Ge Zhang for the great research collaborations. Many thanks to all the members that I have met at the MSC lab, including Chen-Yu Chan, Xiaowen Yu, Yu Zhao, Shuyang Li, Te Tang, Hsien-Chung Lin, Yongxiang Fan, Cheng Peng, Wei Zhan, Yu-Chu Huang, Zining Wang, Jianyu Chen, Kiwoo Shin, Taohan Wang, Shiyu Jin, Yujiao Cheng, Chen Tang, Zhuo Xu, Yeping Hu, Jiachen Li, Saman Fahandezhsaadi, Hengbo Ma, Changhao Wang, Xinghao Zhu, Ting Xu, Weiye Zhao, Lingfeng Sun, Yiyang Zhou, Huidong Gao, Xiang Zhang, Zheng Wu, Wu-Te Yang, Jinning Li, Catherine Weaver, Akio Kodaira, Chenfeng Xu, Ce Hao, Chenran Li, Ran Tian, Yichen Xie, Qiyang Qian, Keita Kobashi, Jen-Wei Wang, Wei-Jer Chang, Chiara Talignani Landi, Mayuko Mori, Daichi Kitagawa, Takanori Yamazaki, Motohiro Hirao, and all others.

Finally, I would like to express my deepest love to my parents, whose support and love are with me all the time. I would also like to thank my friends that I met in Berkeley, including friends in CalTTC, CFC Berkely Church, and BSF El Sobrante.

Chapter 1

Introduction

1.1 Background

Robotic systems have made a significant impact in our society in the last couple of decades, for example, in industrial and autonomous driving applications [131, 104]. In manufacturing, robotic automation has increased productivity by executing repetitive tasks in designated areas without human workers. Control policies can be developed for these applications that follow human-engineered, predetermined rules. However, these robotic systems lack flexibility and generalizability to different settings; additionally, many real-world tasks in factories and warehouses require robots to work safely alongside or collaborate with other agents such as human workers and other robots. Autonomous vehicles share similar challenges, where robotic systems must be aware of the environment to navigate roads and parking lots safely. Ideally, we want to grant both human and robotic systems maximum freedom in choosing their actions to achieve higher joint-flexibility in these complex human-robot applications. However, such freedom introduces uncertainty to the environment and collision risks to the agents involved. In particular, an environment is an uncertain environment to a robot when there exist unobservable states of the surrounding agents, which causes their unpredictable movements (Figure 1.1). For example, a robot cannot directly observe a human worker's intent if the person has the freedom of choosing actions instead of executing an assigned action. A misunderstanding of the worker's intent or wrong prediction of the worker's future movement may cause collisions when the robot follows an unsafe plan due to the wrong predictions. Therefore, robotic applications in these uncertain environments are generally challenging. In these uncertain environments, we need to develop robotic systems that utilize their prior knowledge of the application and their observation of the environment to appropriately analyze the unobservables, make predictions, and replan while completing their tasks.

There are two main approaches to designing a robotic system in uncertain environments: the modularized design approach [162, 76] and the end-to-end design approach [153, 25]. In the modularized design approach, the functions (or modules) that govern robot behav-

iors such as perception, prediction, planning, and control are individually designed. These systems offer clear structures and explainable behaviors that can be easily targeted and improved. On the other hand, end-to-end design approaches leverage advances in artificial intelligence to learn the end-to-end robot policies, i.e., complex functions represented by neural networks that receive sensor readings from the environment and output control commands directly. However, it is difficult to identify how different environmental changes or external agents' behaviors (e.g., an interacting human agent) affect the end-to-end policy's decisions; it can be challenging to make safety guarantees or generalize the end-to-end policy to other environments. Considering the trade-offs of these design approaches, modularized design approaches are widely adopted in industry and academia for robotic systems in uncertain environments with human interaction. This dissertation will focus on the design of such modularized robotic systems.

Depending on the specific robotic application, what constitutes "good" performance may vary; in this dissertation, we focus on three important aspects of the performance of modularized robotic systems and their challenges.

- **Computational efficiency:** In uncertain environments, the robot must react quickly to unexpected events, and hence the computational speed of each module is crucial. Individual algorithms in each module should be designed to suit the application and leverage parallel computing technology. In addition, the robot could use different algorithms for different scenarios to handle environmental changes more efficiently. Therefore, modules should have the ability to use different algorithms strategically.

- **Safety:** The ability to avoid collisions with static and dynamic obstacles directly links to the safety of the robot and its surroundings. In uncertain environments where motions of moving obstacles are hard to predict, safety is critical yet challenging. The realization of safety is a combined effort of the robot modules. The perception and prediction modules should be accurate, the planner should find a safe plan, and the controller should execute the plan accurately. Theoretically, the planning result will remain safe during execution if the perception and prediction are perfect. However, perfect perception and prediction are rarely possible; therefore, dedicated designs and strategies for modules to collaboratively ensure safety without perfect perception and prediction are needed. For example, a predictor should provide the prediction result with its prediction error history to the planner; the planner should account for the predictor's performance based on the prediction error history to avoid being overly confident with the prediction result.

- **Robot Motion Optimality:** We desire robots to perform optimally according to application-specific objectives. Task completion time, travel distance, energy consumption, and travel area coverage are typical metrics that a robot minimizes to achieve motion optimality. In static and deterministic environments, motion optimality relies on the planner to find an optimal plan given the objectives and the controller to execute the plan precisely. However, in dynamic and uncertain environments, the optimal

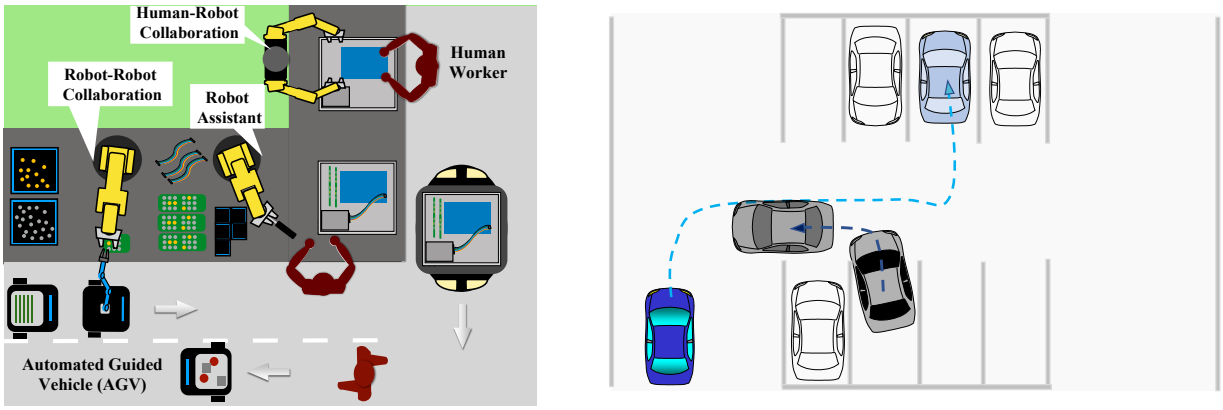


Figure 1.1: Applications of robotic systems in uncertain environments. A factory setting (left) where the manipulators, mobile robots, and mobile manipulators need to collaborate with human workers. A parking lot setting (right) where the ego vehicle (blue car) needs to plan for parking while avoiding a moving obstacle vehicle (gray car) and other static vehicles (white cars). The other moving agents' intents and future motion plans are unknown to the ego robot in both settings. To complete a task, the robot will need to perform safely and efficiently under these uncertainties.

plan may no longer be optimal once the environment changes are different from the prediction; a conservative planner may sometimes result in better robot motions than a greedy planner that outputs the "optimal" solution by fully trusting the partially incorrect predictions. In uncertain environments, the robot should be able to extract information about the uncertainties from observations and consider it properly during planning. Similar to ensuring safety, coordinated designs and strategies are needed for modules to optimize robot motions collaboratively.

Since planning is often the most time-consuming module among all modules, this dissertation devotes much effort to the algorithm design for improving the planner's computational efficiency, which directly improves the computational efficiency of the robotic system. Furthermore, we study the design and relationship between modules in the modularized robotic system and develop strategies for multiple robotic applications in uncertain environments. The proposed robotic systems and strategies aim to enable better safety and robot motion optimality.

1.2 Computational Efficiency and Robot Motion Planning

Since computational efficiency is important for robotic systems in uncertain environments, planners should be able to find feasible and locally-optimal solutions to various planning problems quickly. Hence, we evaluate the planner's performance based on 1) computa-

tion time and 2) plan quality, i.e., plan optimality given the application-specific objective. Naturally, there is a trade-off between the two performance metrics, i.e., a planner will need more time to find a better solution. Therefore, we need to consider both of them when we compare the overall performances of different planners. Three types of planning are considered in this dissertation:

- **Task Planning:** A *task-level* planning problem refers to planning a sequence of *actions*, which are tuples of $\{motion, object\}$, to complete a task. For example, gluing two pieces of paper together can be completed by executing five actions sequentially: 1) $\{grab, paper1\}$, 2) $\{release\ at\ location\ A, paper1\}$, 3) $\{glue, paper1\}$, 4) $\{grab, paper2\}$, and 5) $\{release\ at\ location\ A, paper2\}$. A task planner often tries to minimize task completion time [55]. If the robot collaborates with other human workers, human fatigue and mutual spatial interference are often considered [55, 107]. The specific *motion* generation relies on other planners.

- **Path Planning:** Path planning and motion planning problems are *action-level* planning problems. A path is a sequence of robot configurations that connects the initial configurations to the goal configurations. Without considering the full robot kinematic models which often involve velocity and acceleration terms, path planning problems can usually be solved quickly. It is a common strategy to first use a path planner to generate some "waypoints" and form a path, then use a tracking controller to generate control commands for the robot to track the path [17, 174]. Path planning is useful when there exists a tracking controller that can control the robot to follow the generated path closely. Such controller exists when the robot kinematic model can be approximated by simpler models such as a differential-drive, a bicycle model, or a point mass [26]. However, path planning results for robots with complicated models are sometimes useless because tracking these paths that do not consider the full robot model can be hard or even impossible.

- **Motion Planning:** In this dissertation, a motion planning problem refers to the problem of finding a sequence of robot input commands and its resulting trajectory that brings the robot from its initial state to its goal state to achieve the desired *motion*. The *motion* in the previous example contains many sub-motions. For example, *grab* contains $\{reach, grasp, retrieve\}$. While *grasp*, or robot grasping, is itself a major research field, in this dissertation, we focus on motion planning problems that target "moving from A to B," such as *reach* and *retrieve*. Unlike path planning, motion planning considers both the robot input commands and the resulting trajectory, i.e., the full robot kinematic model; therefore, this planning method ensures that the robot can follow the path by executing the planned commands. This guarantee is essential when the system's kinematics is complicated and developing a tracking controller is not trivial [90].

Planning at task-level and at action-level are both critical. Nevertheless, we view action-level planning as a more fundamental problem because all robot application requires action-

level planning while simple applications may not require task-level planning; therefore, we put more effort into solving the action-level problem (Chapters 2-5) and discuss task-level planning in Chapter 7. In particular, we focus on motion planning problems for the completeness of their planning results, i.e., planning results that consider both the robot input commands and the resulting trajectory. Unlike path planning, we ensure that the motion planning result is directly executable.

Motion planning becomes a challenging problem when: 1) the environment is cluttered with many obstacles, 2) the planning horizon is long, and 3) the robot kinematics is complicated. When solving these problems, planners often suffer a long computation time and a low success rate. Therefore, we desire planners that overcome these problems while generating solutions that achieve a certain level of optimality according to the application-specific objectives. This challenge motivates the development of motion planners presented in Chapters 2-4. On the other hand, robots may fail to react to rapid environmental changes by replanning with such planners only. Therefore, Chapter 5 introduces a hierarchical optimal planning strategy that augments the planner with a safety controller to react to changes in the environment. Furthermore, we note that different motion planning algorithms perform differently in different scenarios; therefore, a planner should not be restricted to a single planning algorithm but switch between different planning algorithms to computationally efficiently handle different situations. An example of such strategic motion planner is presented in Chapter 8.

The following section discusses our approach to enable better performance in safety and optimality for modularized robotic systems.

1.3 Integrated Strategies of Modularized Robotic Systems

We first give a closer look at the robotic system's structure. A modularized robot system often has four main modules (Fig. 1.2): perception, prediction, planning, and control. The perception module monitors the environment with sensors. It records the observable states of the environment (e.g., the observable states such as the locations of the human workers or robots nearby) and perform signal processing such as filtering. The prediction module takes the processed signals and makes predictions of the environments, such as future movements of the surrounding agents or estimations of the agents' unobservable states, such as their intentions. The planning module receives the prediction information and generates task-level and action-level plans to guide the robot safely and efficiently in completing a task. Finally, the control module executes control commands to carry out the plan.

The success of the robotic system depends on the modules' performance individually and collectively. However, most works in modularized robotic system design only emphasize the design of a single module and use off-the-shelf methods directly for other modules. This approach limits the collaboration between modules because the off-the-shelf methods may

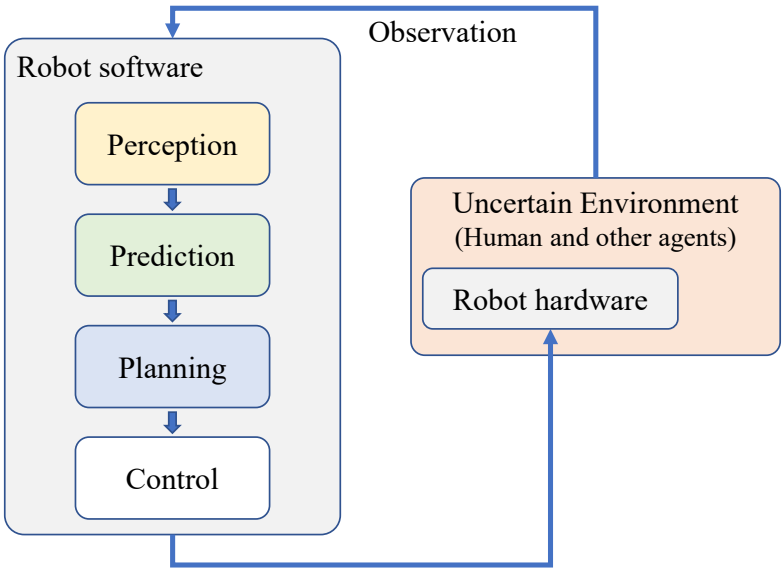


Figure 1.2: Illustration of the modularized robotic system and the environment.

not be able to provide adequate information or fully utilize the information provided by the re-designed module. While each module should calculate as quickly and accurately as possible, it is also essential that the modules upstream provide the outputs with a *bound* on how much the downstream modules should *trust* these outputs so that the robot can behave safely and optimally. For example, when a robot is bypassing a non-stationary human worker, if the predictor cannot accurately predict the worker's movement, the planner should detour the robot to make more space for the worker when the robot passes by; even though this will cost the robot to spend more time to complete its task. In contrast, if the predictor is certain about the human worker's future movement, the planner only needs to leave a small clearance away from the worker's predicted future path. The term *integrated strategies* refers to the communication, decision making, and planning that rely on multiple modules. This dissertation focuses on the design and communication of two of the four modules: prediction and planning. The main idea is to design a predictor that can estimate the uncertainty and represent it in an interpretable way that the planner can utilize. Chapter 6 considers the design of a predictor that facilitates the planner for better stability of the closed-loop robot motion. Chapters 7 and 8 present modularized robotic systems and their integrated strategy in an industrial setting and an autonomous parking setting, respectively.

1.4 Dissertation Contributions and Outline

This dissertation consists of two parts. Part I involves a fundamental exploration of robot motion planners and proposes hybrid motion planners that combine and utilize the

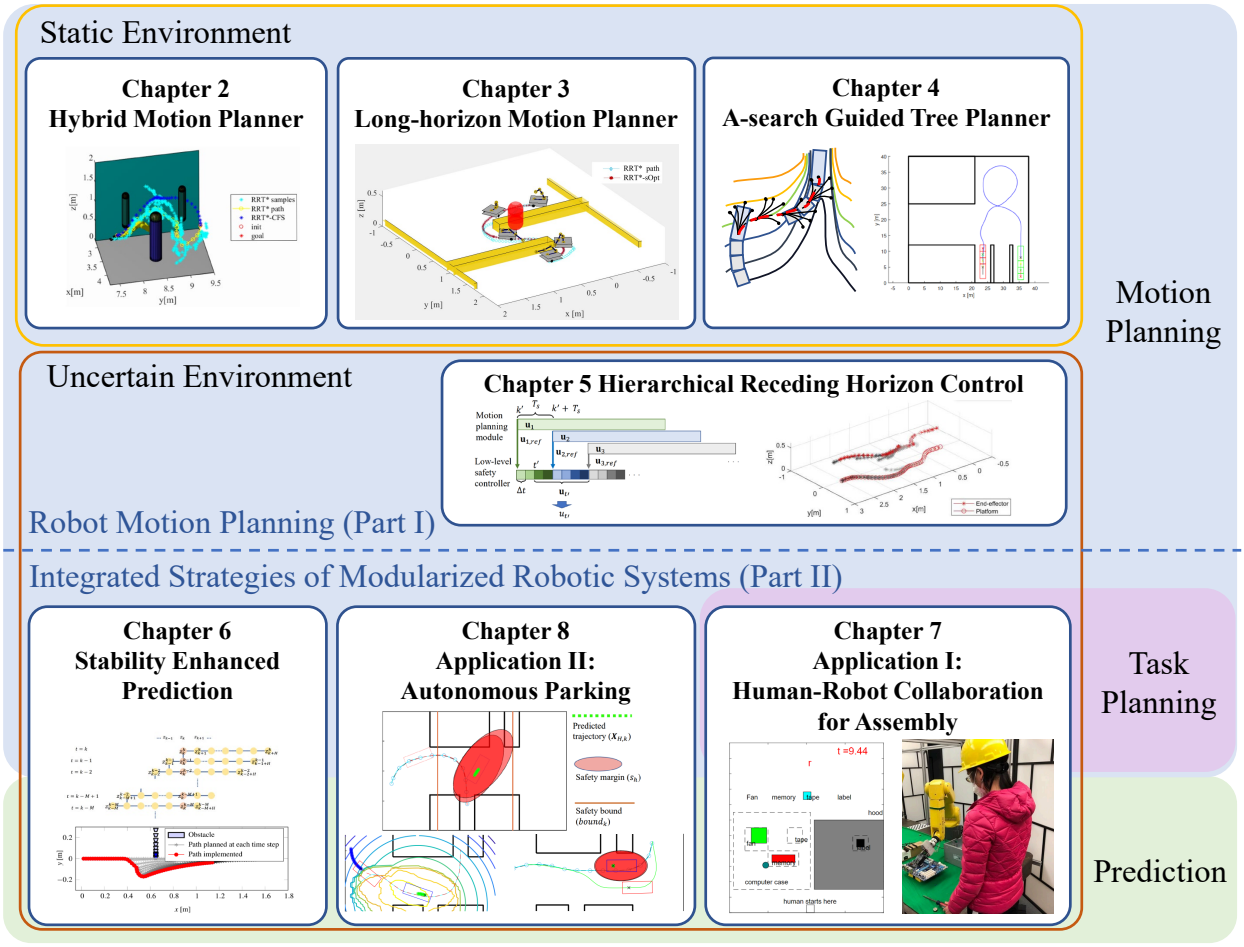


Figure 1.3: Dissertation outline. In Part I, Chapters 2-4 focus on motion planner developments in static and deterministic environments. Chapter 5 moves to motion planning strategies in uncertain environments. In Part II, Chapters 6-8 further incorporate the prediction module and propose integrated strategies for modularized robotic systems in uncertain environments, while Chapter 7 emphasizes task planning.

strengths of different planner types to achieve better computational speed and plan quality. Part II presents application-oriented integrated robotic systems that coordinate the predictor and the planner in order achieve better performance in safety and robot motion optimality. Figure 1.3 illustrates the outline of this dissertation.

1.4.1 Part I: Robot Motion Planning

Traditionally, motion planners generally fall into three types: Sampling-based, search-based, and optimization-based methods. On the other hand, data-driven learning techniques

provide us with a different approach to generating robot motion. To better handle the aforementioned challenges in Section 1.2, Part I studies these different types of motion planners and then develops hybrid planners that combine and utilize the strength of existing methods. Chapters 2∼4 present such planners that demonstrate better computational speed and plan quality in static and deterministic environments. To extend robotic application to uncertain and dynamic environments, Chapter 5 studies a realization of motion planning in dynamic environments. Notice that the framework proposed in Chapter 5 can also utilize the planners presented in Chapters 2-4 for robots in dynamic environments. Some of the work has been published in [97, 101, 99].

Chapter 2 Robot Motion Planning and Hybrid Motion Planning

In environments full of obstacles, it is always challenging to find a collision-free and dynamically-feasible trajectory between the robot's initial state and goal state. While many motion planning algorithms have been proposed in the past, each has its pros and cons. This chapter presents a benchmark that implements and compares existing planning algorithms on various problems with extensive simulation. Based on that, we also propose a hybrid planning algorithm, an optimal rapidly-exploring random tree with the convex feasible set algorithm (RRT*-CFS), that combines the merits of sampling-based planning and optimization-based planning. The first layer, RRT*, quickly samples a semi-optimal path, and the second layer, CFS, performs sequential convex optimization given the reference path from RRT*. The proposed RRT*-CFS has feasibility and convergence guarantees. Simulation results show that RRT*-CFS benefits from the hybrid structure and performs robustly in various scenarios, including the narrow passage problems.

Chapter 3 Long-Horizon Motion Planning

Many robotic applications require robots to travel long distance, however, planning for a long-horizon motion plan is challenging due to the unsatisfactory scaling ability of modern motion planners. We propose a hybrid planner, RRT* with segmented trajectory optimization (RRT*-sOpt), which can plan for long-horizon robot navigation in environments with obstacles. Like RRT*-CFS, RRT*-sOpt combines the merits of sampling-based planning, optimization-based planning, trajectory splitting to plan for a collision-free and dynamically-feasible motion plan quickly. Given the semi-optimal reference path from the RRT* layer, the sOpt layer splits the reference path and optimizes each segment. It then splits the new trajectory again and repeats the process until the whole trajectory converges. We also propose reducing the number of segments before convergence to reduce computation time further. Simulation results show that RRT*-sOpt benefits from the hybrid structure with trajectory splitting and performs robustly in various robot platforms and scenarios.

Chapter 4 Search-based Motion Planning for Articulated Vehicles

Planning for a robot with a complex kinematic system, e.g., a tractor-trailer system, is challenging because the solution needs to satisfy the highly nonlinear kinematic equations. This chapter presents a motion planning strategy that utilizes the improved A-search guided tree to enable autonomous parking of a general 3-trailer with a car-like tractor. Unlike state-of-the-art state-lattice-based methods where numerous motion primitives are necessary to ensure successful planning, our work allows quick off-lattice exploration to find a solution. Our treatment brings at least three advantages: fewer and lower design complexity of motion primitives, improved success rate, and increased path quality. Unlike on-lattice exploration, where the cost-to-go is obtained by querying a heuristic look-up table, off-lattice exploration entails a well-defined heuristic function at off-lattice nodes. It is challenging to manually design a heuristic that can capture the tractor-trailer kinematics and differentiate the different levels of maneuver difficulty given the tractor-trailer's configuration. Therefore, we exploit the power of the data-driven method and train a neural network through reinforcement learning to model the maneuver costs of the trailer and use it to obtain the heuristic value that approximates the cost-to-go. Simulations demonstrate the effectiveness of the proposed method in terms of computation time and path length.

Chapter 5 Motion Planning in Dynamic Environments

Motion planners need to strike a balance between plan quality and computation time. Computation time is safety-critical in dynamic environments where a motion planner needs to replan quickly and react to moving obstacles. However, as mentioned in Chapter 4, motion planning is naturally time-consuming when the robot model is complex, e.g., a mobile manipulator; therefore, solely putting efforts into reducing the computation time of the motion planner may not be enough. This chapter presents a hierarchical receding horizon control algorithm (HRHC) to ensure safety while reducing travel time and distance in robots surrounded by dynamic environments. HRHC contains an optimization-based motion planning module that utilize the robot's kinematic redundancy for better motion optimality and a low-level safety controller to deal with fast changes in the environment. With this method, we verify the performance through experiments. The results show that HRHC increases space efficiency and can guarantee local safety in dynamic and uncertain environments.

1.4.2 Part II: Integrated Strategies of Modularized Robotic Systems

This dissertation focuses on human-induced uncertainties and their effects on the space-domain and time-domain, which make both task-level planning and action-level planning challenging. Chapters 6 and 8 address the uncertainty's effect on space-domain with the focus on motion planning, while Chapter 8 addresses the uncertainty's effect on time-domain with the focus on task planning. When a robot is in a dynamic and uncertain environment,

its predictor makes predictions based on past observations and passes this information to the planner so that the planner can plan for the future. Due to the uncertainties, a predictor often needs to update its prediction as more observations come in. These updates can propagate to the planner and cause the motion plans to change. Without carefully designing the predictor-planner relationship, the robot motion's optimality and safety may be compromised. Chapter 6 discusses the relationship between prediction, motion planning, and the resulting stability of the closed-loop robot motion. Furthermore, we demonstrate the collective designs of the prediction and planning modules to ensure that sufficient information is passed from the predictor to the planner and that the planner will not plan "too greedily" under large uncertainty. Chapters 7 and 8 target human-robot collaboration for computer assembly and autonomous car parking, respectively. Both settings contain uncertainty caused by surrounding agents. The key to developing the proposed modularized robotic systems is to examine the uncertainties involved in the applications and model them properly so that the predictor can estimate them and make effective predictions, and the planner can plan appropriately according to the predictor's outputs. We design such predictors and planners to facilitate safe and optimal robot motion. Some of the work has been published in [98, 102, 100].

Chapter 6 Environment Prediction and Motion Planning

Real-time, safe, and stable motion planning in applications involving dynamic human-robot interaction remains challenging due to the time-varying nature of the problem. One of the biggest challenges is guaranteeing the planning algorithm's closed-loop stability in dynamic environments. Typically, this can be addressed if there exists a perfect predictor that precisely predicts the future motions of the obstacles. Unfortunately, a perfect predictor is rarely achievable. We discuss necessary conditions for the closed-loop stability of a planning problem in uncertain environments using the framework of model predictive control (MPC). It is concluded that the predictor needs to be able to detect the obstacles' movement *mode* change within a time delay allowance, and the MPC needs to have a sufficient prediction horizon and a proper cost function. If these conditions are satisfied, the MPC can ensure closed-loop stability, and it can still avoid collision when the obstacles' mode suddenly changes (i.e., when the *mode* change condition does not hold); therefore, safety is guaranteed. Also, the closed-loop performance is investigated using a notion of M-convergence, which guarantees finite local convergence (at least M steps ahead) of the open-loop trajectories toward the closed-loop trajectory. With this notion, we verify the performance of the proposed MPC with prediction through simulations and experiments. With the proposed method, the robot can better deal with dynamic environments and reduce the closed-loop cost.

Chapter 7 Application I: Human-Robot Collaboration for Assembly

In this chapter, we switch from action-level planning to task-level planning, which is needed when the robot collaborates with others. Nevertheless, efficient and robust task planning for a human-robot collaboration (HRC) system remains challenging. The human-aware task planner needs to assign jobs to both robots and human workers so that they can work collaboratively to achieve better time efficiency. However, the complexity of the tasks and the stochastic nature of the human collaborators bring challenges to such task planning. To reduce the complexity of the planning problem, we utilize the hierarchical task model, which explicitly captures the sequential and parallel relationships of the task. We model human movements with the sigma-lognormal functions to account for human-induced uncertainties. A human action model adaptation scheme is applied during run-time, and it provides a measure for modeling the human-induced uncertainties. We propose a sampling-based method to estimate human job completion time uncertainties. Next, we propose a robust task planner, which formulates the planning problem as a robust optimization problem by considering the task structure and the uncertainties. We conduct simulations of a robot arm collaborating with a human worker in an electronics assembly setting. The results show that our proposed planner can reduce task completion time when human-induced uncertainties occur compared to the baseline planner.

Chapter 8 Application II: Autonomous Parking in Uncertain Environments

This chapter presents an integrated motion planning system for autonomous vehicle parking in the presence of other moving vehicles. The proposed system includes 1) a hybrid environment predictor that predicts the motions of the surrounding vehicles and 2) a strategic motion planner that reacts to the predictions. The hybrid environment predictor performs short-term predictions via an extended Kalman filter and an adaptive observer. It also combines short-term predictions with a driver behavior cost-map to make long-term predictions, which contain information of the prediction uncertainties. Unlike Part I, where we combine different planning techniques to develop a hybrid planner, we focus on strategies for switching between different types of planners to deal with different scenarios. The strategic motion planner comprises 1) a model predictive control-based safety controller for trajectory tracking; 2) a search-based retreat planner for finding an evasion path in an emergency; 3) an optimization-based repair planner for planning a new path when the original path is invalidated. Simulation validation demonstrates the effectiveness of the proposed method in terms of initial planning, motion prediction, safe tracking, retreating in an emergency, and trajectory repairing.

Part I

Robot Motion Planning

Chapter 2

Robot Motion Planning and Hybrid Motion Planning

2.1 Introduction

Motion planning is one of the key challenges in robotics. It refers to the problem of finding a collision-free and dynamically-feasible trajectory between the initial configuration and the goal configuration in environments full of obstacles. Existing motion planning algorithms fall into two categories: planning-by-construction or planning-by-modification [112]. Search-based planning and sampling-based planning are two typical plan-by-construction algorithms. Algorithms such as A* and D* search [63, 155] belong to search-based algorithms, whereas rapidly-exploring random tree (RRT) [94] and probabilistic roadmap (PRM) [79] belong to sampling-based planning. Planning-by-modification refers to the algorithms that reshape a reference trajectory to obtain optimality regarding specific properties [111, 136, 97]. Optimization-based algorithms belong to this category.

Both types of motion planning algorithms have pros and cons. Planning-by-construction algorithms are guaranteed to generate collision-free trajectories and require short computation time. However, the constructed trajectories often do not consider dynamic constraints and are not smooth. Some modifications remedy these problems [166, 170], but require more computational resource. On the other hand, optimization-based algorithms often start with an initial trajectory that links the initial state and the goal state [111, 136]. Next, the algorithms iteratively improve the trajectory to satisfy the constraints (e.g., collision-free conditions) and minimize the cost. Note that the initial trajectory can be either feasible or infeasible depending on the requirements of the algorithms. The trajectories generated by optimization-based algorithms are usually smooth. However, optimization-based algorithms often fail to find a feasible solution if initial trajectories are naively chosen (e.g., a line segment from the initial to the goal in the configuration space). The reason is that most optimization-based algorithms rely on local gradient information, so the final trajectories and the computation time highly depend on the selection of the initial trajectory. This chapter

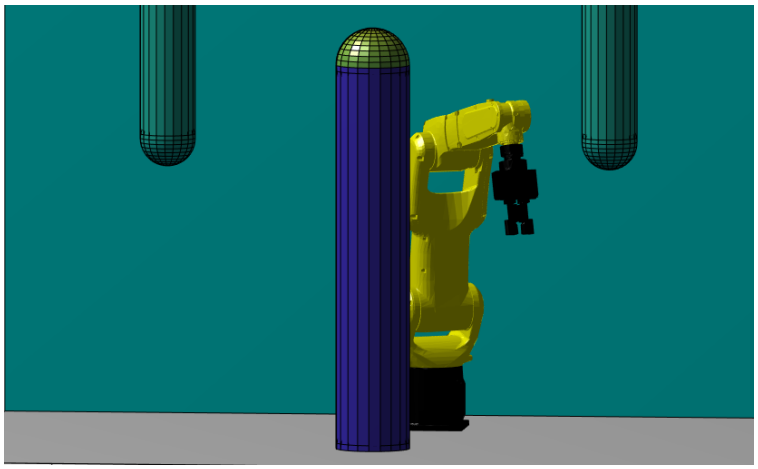


Figure 2.1: A manipulator navigating through the obstacles.

presents a benchmark that tests motion planning algorithms from different categories to see the effects of these pros and cons when solving motion planning problems. To the best of the authors knowledge, this is the first comprehensive benchmark that tests motion planning algorithms from different categories.

In addition to the benchmark, this chapter also presents a hybrid planning algorithm, RRT*-CFS, which combines the merits of planning-by-construction algorithms and planning-by-modification algorithms. The algorithm has two layers. We first use RRT* [78] to generate a feasible and semi-optimal path quickly. This path then serves as an initial trajectory for the optimization layer that uses the convex feasible set algorithm (CFS) [111] to solve the non-convex motion planning problem quickly. RRT*-CFS can find a globally-near-optimal solution in complex environments, even in scenarios with narrow passages [140], and has good performance in terms of optimality and computation time. The contributions of this chapter are threefold as follows:

- A comprehensive benchmark that compares motion planning algorithms from different categories is presented.
- The proposed RRT*-CFS algorithm can solve planning problems that cannot be solved by many optimization-based algorithms alone, has a short computation time, and has the lowest average cost of all algorithms we compare in this chapter.
- We implement RRT*-CFS and demonstrate its success with extensive simulation (video is publicly available here).

2.2 Problem Formulation and Related Works

2.2.1 Problem statement

In this section, we first introduce the general formulation of the motion planning problem that we consider.

Problem 1. *In many scenarios, robot motion planning can be performed by solving an optimization problem with the following form:*

$$\min_{\mathbf{x}\in\Gamma} f(\mathbf{x}), \tag{2.1}$$

where $\mathbf{x} \in \mathbb{R}^n$ *and* Γ *defines the feasible set:*

$$\Gamma = \bigcap_j \Gamma_j = \bigcap_j \{\mathbf{x} : h_j(\mathbf{x}) \geq 0\}. \tag{2.2}$$

An example of $\mathbf{x}$ in 2D planning would be $\mathbf{x} = [x_1, y_1, x_2, y_2, \cdots, x_h, y_h]^\top$, where (x, y) contains the x and y coordinate of the robot's location in 2-dimensional Cartesian space, also referred as waypoints; and Γ would be the collision-free set in $\mathbb{R}^n$. We assume that the constraint function $h_j(x)$ is a semi-convex function [111]. For example, $h_j(x)$ can be the distance between a robot and the jth obstacle. The cost function, $f : \mathbb{R}^n \to \mathbb{R}$, is strongly convex and smooth. Note that the motion planning problem is non-convex due to the obstacles and the non-linear robot dynamics. Also, the dimension of the planning problem, n, depends on both the robot's state space and the number of points needed from the initial state to the final state. The value of n is usually large in motion planning problems; therefore, solving motion planning problems is generally hard.

2.2.2 Sampling-based motion planning

In general, sampling-based algorithms are fast algorithms that can quickly find a path connecting the initial with the goal [94]. There are many sampling-based planning algorithms, such as RRT*, variations of PRM [7], [15] and variations of RRT [170, 78, 86, 72]. These variations modify the original methods for better planning performances, e.g., handling dynamic constraints, smoothing trajectories, short-cutting trajectories, etc. However, these modifications often require more computation time and resource and the requirements grow quickly as the robot system becomes more complex. As a result, these algorithms require longer computation time in complex systems. This weakens the computation advantage of sampling-based methods. Since our goal is to construct a strong hybrid algorithm rather than a strong stand-alone sampling-based algorithm, we value computation time highly and choose RRT*, which can provide a feasible and semi-optimal [78, 72] path with the shortest computation time.

2.2.3 Search-based motion planning

Search-based algorithms are another type of planning-by-construction algorithm that will solve for a deterministic result given the initial and the goal. Algorithms belonging to this type typically require a search graph or tree in the state space that connects the initial state and the goal state, then use search methods such as A* or D* search [63, 155] to find the best solution. Resolution completeness and optimality are often guaranteed. However, the construction of the graph can be time-consuming, and an effective heuristic can be hard to find; therefore, they may not be directly applicable for robot planning when the environment changes frequently.

2.2.4 Optimization-based motion planning

There are many optimization-based algorithms that can be used for motion planning such as SQP [154], CHOMP [136], TrajOpt [146], and CFS [111]. SQP uses the Lagrangian of the original problem to formulate a transformed problem that solves the Lagrange multipliers. The solution of these Lagrange multipliers is then used to update the decision variables of the original problem. CHOMP and TrajOpt formulate an unconstrained problem with a cost function that penalizes the trajectory's smoothness and proximity to the obstacles. However, the two have different approaches to collision detection. In addition, TrajOpt uses SQP to solve the problem, whereas CHOMP uses gradient descent.

Among these algorithms, CFS is a fast optimization-based motion planning algorithm that can handle infeasible initialization under some assumptions. Here we give a brief review of the CFS algorithm. We can rewrite the non-convex optimization problem as follows:

$$\mathbf{x}^* = \underset{\mathbf{x}\in\Gamma\subset\mathbb{R}^n}{\arg\max} \quad f(\mathbf{x}). \tag{2.3}$$

CFS solves the non-convex problem iteratively. The following information is required:

- *Initialization*: An initial value of the state $\mathbf{x}^{(0)}$, which does not necessarily satisfy $\mathbf{x}^{(0)} \in \Gamma$.
- *Safety index and disjoint convex obstacles* : Similar to (2.2), where $h_j(\mathbf{x})$ is the safety index for the jth disjoint obstacle.
- *Convex feasible set*: The convex feasible set, $\chi^{(k)} := \chi(\mathbf{x}^{(k)}) \in \Gamma$, is constructed corresponding to previous states $\mathbf{x}^{(k)}$.

The convex feasible set we consider is as follows:

$$\begin{aligned} \chi^{(k)} &= \bigcap_{j=1}^{n} \chi_j^{(k)}, \\ &= \bigcap_{j=1}^{n} \{\mathbf{x} : h_j(\mathbf{x}^{(k)}) + \nabla^\top h_j(\mathbf{x}^{(k)})(\mathbf{x} - \mathbf{x}^{(k)}) \geq 0\}. \end{aligned} \tag{2.4}$$

With CFS, a convex sub-optimization problem is formulated and solved for the optimal value of $\mathbf{x}^{(k+1)}$:

$$\mathbf{x}^{(k+1)} = \arg\max_{\mathbf{x}\in\chi^{(k)}} \quad f(\mathbf{x}). \tag{2.5}$$

The algorithm solves the problem iteratively and results in a sequence of $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(k)}, \ldots$. It is guaranteed in [111] that this sequence will converge to a local optimal, $\mathbf{x}^*$.

Notice that SQP and TrajOpt rely on the second-order information of the original problem, while CHOMP and CFS rely on the gradient information. As mentioned previously, optimization-based algorithms rely on local gradient information (or higher-order information); therefore, the final trajectories and the computation time highly depend on the choice of the initial trajectory.

2.2.5 Hybrid motion planners

There have been several works focusing on hybrid planners [112, 108, 129, 43]. Authors of [112] used Lattice A* Search to generate initial trajectories for CFS and illustrated its performance on a mobile robot. The sampling space of these experiments is 2D. Since the number of points in the lattice grid grows exponentially as the dimension grows in the configuration space, the effectiveness of this work for high dimensional applications (e.g., 6-DoF manipulators) is highly doubtful. Authors of [108] adopted the Bidirectional Rapidly-exploring Random Tree (BiRRT) [86] to generate an initial feasible guess for the TrajOpt trajectory optimizer and demonstrated the success of their approach on the Atlas robot. However, only a few testing scenarios were presented; the robots only needed to avoid no more than two obstacles. Authors of [129] presented a planning algorithm that combined a roadmap and TrajOpt. Besides generating a collision-free and dynamically-feasible trajectory, they focused on avoiding singularities in redundant manipulators and meeting Cartesian constraints. However, the algorithm required a long planning time. Authors of [43] combined a sparse roadmap with TrajOpt. However, their methods did not consider complex environments.

A key challenge in motion planning is the narrow passage problem, which refers to planning problems with a very narrow region between the initial and the goal in the feasible configuration space. Motion planning algorithms often take too much time or even cannot find a solution when encountering a narrow passage problem even though the solution does exit [140, 173]. This type of problem is one of the target scenarios of this work, which was not tested in [112, 108, 146, 129, 43].

2.3 The RRT*-CFS Algorithm

In this section, we introduce the proposed RRT*-CFS algorithm and its feasibility and global convergence guarantees. The proposed RRT*-CFS inherits the merits and avoids

the shortcomings of each of the two algorithms. The RRT*-CFS algorithm solves the non-convex motion planning problem first by quickly finding a feasible and semi-optimal path, then iteratively refining the solution using CFS. The RRT*-CFS has three main features.

- First, the RRT* layer can be implemented with multi-thread computation, which allows us to reduce the computation time significantly.
- Second, RRT*-CFS has stochasticity due to the random sampling process in RRT*. This helps RRT*-CFS avoid bad local optima that optimization-based algorithms may get stuck in.
- Finally, RRT*-CFS inherits the properties of CFS so that feasibility, smoothness, and convergence of the final solution are guaranteed if the problem satisfies the description in Section 2.2.1.

Denote the configuration of a d-degree-of-freedom (d-DoF) robot as $\theta \in \mathbb{R}^d$, the initial configuration as θ_0, the goal configuration as θ_{goal}, the maximum number of samples in one RRT* thread as $n_{samples}$, and the obstacles as $\mathcal{O}$. The RRT*-CFS algorithm is summarized as in Algorithm 1.

Algorithm 1: RRT*-CFS

input $\theta_0, \theta_{goal}, n_{samples}, \mathcal{O}$
while *!* $\exists\ \boldsymbol{\theta}^{RRT^*}$ **do**
  $\boldsymbol{\theta}^{RRT^*} \leftarrow$ `Multi_thread_RRT*`$(\theta_0, \theta_{goal}, n_{samples}, \mathcal{O})$
$\mathbf{x}^{(0)} \leftarrow$ `generate_reference`$(\boldsymbol{\theta}^{RRT^*})$
while *Stop criterion is not satisfied* **do**
  $\chi^{(k)} = \bigcap_{j=1}^{n} \{\mathbf{x} : h_j(\mathbf{x}^{(k)}) + \nabla^\top h_j(\mathbf{x}^{(k)})(\mathbf{x} - \mathbf{x}^{(k)}) \geq 0\}$
  $\mathbf{x}^{(k+1)} = \underset{\mathbf{x} \in \chi^{(k)}}{\arg\min}\ f(\mathbf{x})$
return $\mathbf{x}^{(k+1)}$

As shown in Algorithm 1, given the inputs, $\theta_0, \theta_{goal}, n_{samples}$, and $\mathcal{O}$, each thread of the multi-thread RRT* starts to find a feasible path that connects the initial configuration and the goal configuration. If more than one thread find a path, we choose the shortest path and set it as $\boldsymbol{\theta}^{RRT^*}$. If no thread finds a solution, we repeat the multi-thread RRT* until we find a path. By setting up the $n_{samples}$ properly, we can find a solution in the first batch almost every time. Let $x \in \mathbb{R}^n$ and the planning horizon be H, we generate the initial reference $\mathbf{x}^0 := [x_0^{0\top}, x_1^{0\top}, \cdots, x_H^{0\top}]^\top$ for CFS using the sampled path $\boldsymbol{\theta}^{RRT^*}$. This process can be done by feeding the $\boldsymbol{\theta}^{RRT^*}$ to a motion generator (e.g., iLQR [159]) that outputs a motion plan, $\mathbf{x}^{(0)}$, which is a trajectory that follows the RRT* path. Then, the convex feasible set, $\chi^{(k)}$, is generated by linearizing the constraints at the reference point $\mathbf{x}^{(k)}$ for $k = 0, 1, \ldots$. The algorithm terminates when the change of the cost at each iteration is smaller than a threshold, i.e., $\|f(\mathbf{x}^{(k+1)}) - f(\mathbf{x}^{(k)}))\| \leq \epsilon$.

2.3.1 Theoretical analysis

Both the feasibility and the global convergence of RRT*-CFS rely on the fact that the motion planning problem (Problem 1) satisfies the following assumption:

Assumption 1 (Problem formulation)**.** *The cost function $f(\mathbf{x})$ is strongly convex and smooth. The constraint function $h_j(\mathbf{x})$ is continuous, piece-wise smooth, and convex. The state constraint Γ is non-convex and its complement is a collection of disjoint convex sets, i.e., each of the obstacle-region is itself convex.*

Let $\mathbf{x}^r \in \mathbb{R}^n$ be a feasible reference point, i.e., $\mathbf{x}^r \in \Gamma$.

Lemma 1 (Feasibility)**.** *If $\mathbf{x}^r \in \Gamma$, then $\mathbf{x}^r \in \chi^r$ and $Int(\chi^r) \neq \emptyset$, where $Int(\chi^r)$ is the interior of the set χ^r.*

Proof. When $\mathbf{x}^r$ is feasible, $\mathbf{x}^r \in \chi_j^r$ for all j according to (2.4). Therefore, $\mathbf{x}^r \in \chi^r$. [111] proved that $\chi^{(0)}$ has nonempty interior if the assumption is satisfied. The problems studied in this chapter satisfy the assumption; therefore the proof holds true. □

With Lemma 1, we obtain the following theorem:

Theorem 1 (Feasibility of RRT*-CFS)**.** *Under Algorithm 1, the sequence $\{\mathbf{x}^{(k)}\}$ satisfies $\mathbf{x}^{(k)} \in \Gamma$ for $k = 0, 1, 2, ...$*

Proof. RRT* generates $\mathbf{x}^{(0)}$ such that $\mathbf{x}^{(0)} \in \Gamma$, i.e., feasibility holds when $k = 0$. According to Lemma 1, $\chi^{(0)}$ has nonempty interior, then $\mathbf{x}^{(1)} \in \Gamma$ can be attained by solving the convex optimization problem (2.5). By induction, we conclude that $\mathbf{x}^{(k)} \in \chi^{(k-1)} \subset \Gamma$ for $k = 1, 2, 3....$ □

The following shows the convergence of RRT*-CFS. Given Theorem 1, the remainder of the proof is similar to that in [111], Theorem 4.1; therefore, we have the following corollary.

Corollary 1 (Global convergence of RRT*-CFS)**.** *Under Algorithm 1, the sequence $\{\mathbf{x}^{(k)}\}$ will always converge to some $\mathbf{x}^* \in \Gamma$. $\mathbf{x}^*$ is a strong local optimum of (2.1) if the limit is reached. $\mathbf{x}^*$ is at least a weak local optimum of (2.1) if the limit is not reached.*

2.4 Applications

2.4.1 Robot models

We used two different robot platforms (Figure 2.2) to test the RRT*-CFS and other algorithms for comparison.

Figure 2.2: A mobile robot (left) and a manipulator (right).

Mobile robot

We model the mobile robot as a point mass on a $2D$-plan. Denote the states of the mobile robot at time step t as $z_t = [x_t, y_t]^\top$, the input velocity as $u_t = [v_{x,t}, v_{y,t}]^\top$, and the robot configuration as $\theta = [x, y]^\top$. The linear kinematic model, is

$$\begin{bmatrix} x_{t+1} \\ y_{t+1} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_t \\ y_t \end{bmatrix} + \begin{bmatrix} T_s & 0 \\ 0 & T_s \end{bmatrix} \begin{bmatrix} v_{x,t} \\ v_{y,t} \end{bmatrix}, \tag{2.6}$$

where T_s is the sampling time.

Manipulator

Denote the states of a 5-Dof manipulator as $z_t = [\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \omega_1, \omega_2, \omega_3, \omega_4, \omega_5]_t^\top$, where θ_i and ω_i, $i \in \{1, 2, 3, 4, 5\}$ are the angle and the angular velocity of the ith joint, respectively. The input contains the angular acceleration at each joint, denoted as $u_t = [\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5]_t^\top$. The robot configuration is $\theta = [\theta_1, \theta_2, \theta_3, \theta_4, \theta_5]^\top$. The linear kinematic model is as follows:

$$z_{t+1} = \mathbf{A}_5 z_t + \mathbf{B}_5 u_t, \tag{2.7}$$

where

$$\mathbf{A}_5 = \begin{bmatrix} \mathbf{I}_{5\times5} & T_s\mathbf{I}_{5\times5} \\ (\mathbf{0})_{5\times5} & \mathbf{I}_{5\times5} \end{bmatrix} \text{ and } \mathbf{B}_5 = \begin{bmatrix} 0.5T_s^2\mathbf{I}_{5\times5} \\ T_s\mathbf{I}_{5\times5} \end{bmatrix}.$$

2.4.2 The motion planning problem

In this chapter, the goal of the motion planning problem is to plan the commands that bring the robot to the goal state while avoiding obstacles. We first solve for a path

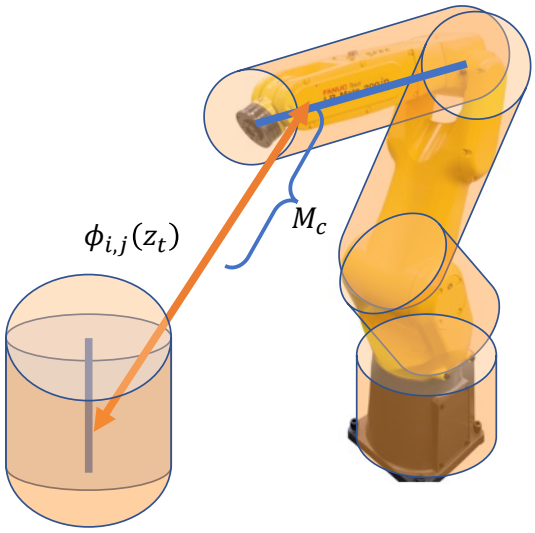


Figure 2.3: Illustration of the distance function.

using the multi-thread-RRT* with the configuration θ defined previously. After getting the path $\boldsymbol{\theta}^{RRT^*}$, an optimization problem can be formulated. The decision variable for each time step is u_t, and the input vector that the algorithm optimizes is denoted as $\mathbf{u} := [u_0^\top, u_1^\top, \cdots, u_H^\top]^\top$, where H is the planning horizon. Similarly, the resulting state vector is $\mathbf{z} := [z_1^\top, z_2^\top, \cdots, z_{H+1}^\top]^\top$. Given the initial state, z_0, we obtain $\mathbf{z} = f_{ki}(z_0, \mathbf{u})$ by concatenating the kinematic function ((2.6) or (2.7)) throughout the planning horizon. For simplicity, denote the kinematic function as $f_{ki,z_0}(\mathbf{u}) := f_{ki}(z_0, \mathbf{u})$.

Problem 2. *In order to obtain the optimal solution* $\mathbf{u}^*$ *given the constrained feasible set* Γ *and the input constraint* $\mathbf{u}_{max}$*, the following optimization problem needs to be solved:*

$$\begin{aligned} \mathbf{u}^* = \arg\min_{\mathbf{u}} \quad & f_{z_0}(\mathbf{u}), \\ s.t. \quad & f_{ki,z_0}(\mathbf{u}) \in \Gamma, \\ & -\mathbf{u}_{max} \leq \mathbf{u} \leq \mathbf{u}_{max}, \\ & z_{H+1} = z_{goal}. \end{aligned} \tag{2.8}$$

The cost function is quadratic that has the form: $f_{z_0}(\mathbf{u}) = \|f_{ki,z_0}(\mathbf{u}) - \mathbf{z}_{goal}\|_2^2 + \lambda\|\mathbf{u}\|_2^2$, which is convex and regular. The first term penalizes the deviation from the goal and the second term penalizes the input.

2.4.3 Implementation

CFS updates $\mathbf{z}^{(k)} = f_{ki,z_0}(\mathbf{u}^{(k-1)})$ at iteration $k = 2, 3 \ldots$. Notice that $\mathbf{z}^{(1)}$ is determined by $\boldsymbol{\theta}^{RRT^*}$ and $\mathbf{u}^{(0)}$ is initialize with a motion generator that commands the robot to track $\boldsymbol{\theta}^{RRT^*}$. In the optimization stage, a safety index function is needed. Here, the distance from the robot to the obstacle is chosen as the safety index function. As shown in Figure 2.3, each link of the manipulator and the obstacle can be captured with a capsule. (We enclose the mobile robot in the same way.) Distances between the capsules can be calculated by obtaining the distances between the center lines of the capsules $\phi_{i,j}(z_t)$, where i represents the ith manipulator link and j represents the jth obstacle. Thus, the safety index function $h_j(z_t) \geq 0$ is equivalent to the set of constraints $\{\phi_{i,j}(z_t) - M_c \geq 0, i = 1, \ldots, \#\text{ robot links}\}$ and M_c is some safety margin. The convex feasible set, $\chi^{(k)}$, is determined by $\mathbf{z}^{(k)}$. Given the feasible set $\Gamma = \bigcap_{i,j,t}\{\mathbf{z} : h_j(z_t) \geq 0\}$ (j numerates over obstacles and t numerates over time steps), the results of the previous iteration ($\mathbf{u}^{(k-1)}$ and $\mathbf{z}^{(k)}$), and the function $f_{ki,z_0}(\mathbf{u})$, we can construct the convex feasible set as:

$$\chi_{z_0}^{(k)} = \bigcap_{j,t}\{\mathbf{u} : h'_{j,t}(\mathbf{u}, z_t^{(k)}, \mathbf{u}^{(k-1)}) \geq 0\}, \tag{2.9}$$

where $h'_{j,t} = h_j(z_t^{(k)}) + \nabla^\top h_j(z_t^{(k)})\nabla f_{ki,z_0,t}(\mathbf{u}^{(k-1)})(\mathbf{u} - \mathbf{u}^{(k-1)})$. Therefore, the iterative sub-problem is as follows:

$$\begin{aligned} \mathbf{u}^{*(k)} = \arg\min_{\mathbf{u}} \quad & f_{z_0}(\mathbf{u}), \\ s.t. \quad & \mathbf{u} \in \chi_{z_0}^{(k)}(\mathbf{u}^{(k-1)}, \mathbf{z}^{(k)}), \\ & -\mathbf{u}_{max} \leq \mathbf{u} \leq \mathbf{u}_{max}. \end{aligned} \tag{2.10}$$

2.4.4 Simulation setup

We show the motion planning simulation results in the following sections. The simulation is conducted in `Matlab R2020a` on a desktop computer with a 3.2GHz Intel Core i7-8700 CPU. The stopping criteria for optimization-based methods (i.e., CFS, SQP, CHOMP, the SQP layer of RRT*-SQP, and the CFS layer of PRM-CFS and RRT*-CFS) are the same. In other words, if either (1) the algorithm reaches the maximum number of iterations (i.e., 40 iterations in this work) or (2) the change of the cost is smaller than the threshold (i.e., $\epsilon = 1 \times 10^{-3}$), the algorithm terminates.

2.4.5 2D motion planning benchmark

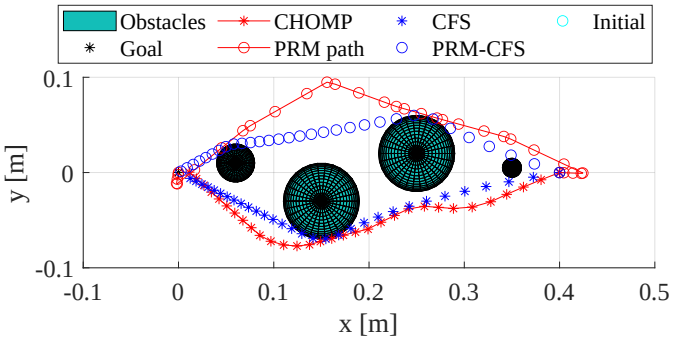
The 2D scenarios simulate general 2D motion planning scenes of mobile platforms. Notice that the obstacles in these scenarios are convex, and the robot kinematics is simplified as a point mass. Therefore, the motion planning problems satisfy the conditions mentioned in Section 2.2.1. The benchmark has two categories: collision avoidance planning in multiple obstacles and narrow passage scenes. In both cases, the planning horizons of optimization-based algorithms are set to be $H = 30$.

Table 2.1: Simulation comparison of 2D planning. (Average of 100 trials.)

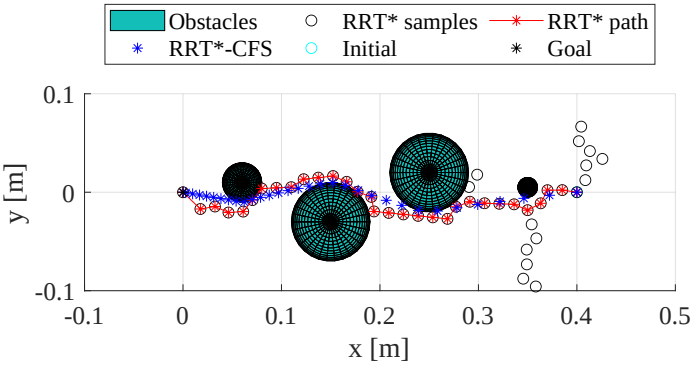
# obstacles: 1								
	Sampling		Optimization			Hybrid		
	RRT*	PRM	CFS	SQP	CHOMP	PRM-CFS	RRT*-SQP	RRT*-CFS
Time average [s]	0.01	1.19	0.04	1.73	0.05	1.26	1.00	0.04
Cost	1.27	1.36	1.01	1.01	1.10	1.01	**1.00**	**1.00**
# iterations	*N/A*	*N/A*	7.40	7.40	137.2	7.22	**7.01**	**7.01**
Success rate (%)	100	100	99	100	99	100	100	100
# obstacles: 4								
	Sampling		Optimization			Hybrid		
	RRT*	PRM	CFS	SQP	CHOMP	PRM-CFS	RRT*-SQP	RRT*-CFS
Time average [s]	0.03	4.24	0.12	4.29	0.23	4.35	2.63	0.15
Cost	1.27	1.38	1.05	1.03	1.11	1.10	**1.01**	**1.01**
# iterations	*N/A*	*N/A*	12.63	12.80	199.42	11.04	**10.58**	**10.58**
Success rate (%)	100	100	99	100	99	100	100	100
# obstacles: 10								
	RRT*		CFS			RRT*-CFS		
Time average [s]	0.04		0.70			0.51		
Cost	1.27		1.11			**1.06**		
# iterations	*N/A*		14.46			**12.69**		
Success rate (%)	100		87			100		

Table 2.2: Simulation comparison of 2D-narrow-passage planning. (Average of 20 trials.)

	Sampling		Hybrid	
	RRT*	PRM	PRM-CFS	RRT*-CFS
Time average [s]	0.27	20.05	20.64	0.43
Cost	1.22	1.22	1.02	**1.01**
# iterations	*N/A*	*N/A*	**6.6**	8.26



(a) Planning result using CFS, CHOMP, and PRM-CFS.



(b) Planning result using RRT*-CFS.

Figure 2.4: Simulation results of a 2D motion planning.

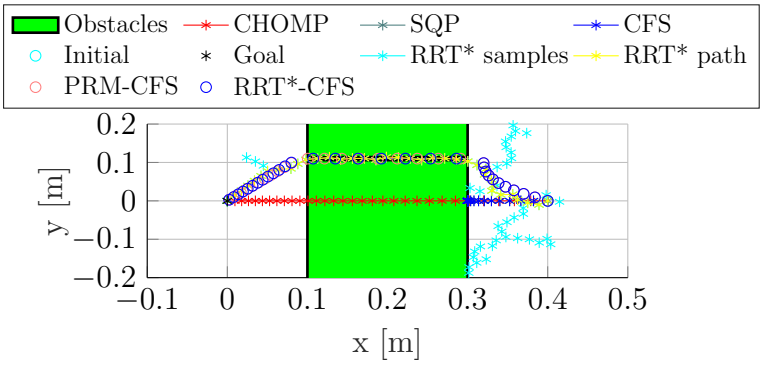


Figure 2.5: A simulation result of motion planning in the narrow passage (close to $y = 0.1$) scenario.

Multiple obstacles

One exemplar simulation environment is shown in Figure 2.4, where Figure 2.4(a) shows the planning result of CHOMP (red-star-line), PRM-CFS (blue-circle), and CFS (blue-star-line) and Figure 2.4(b) shows the planning result of RRT*-CFS (blue-line). By comparing the two figures, we see that all four trajectories are smooth and can successfully bring the robot to the goal (at $(0, 0)$). More importantly, RRT*-CFS converges to the global optimum, while others are stuck in local optima. Table 2.1 shows the result averaging over 100 trials (obstacles are randomly placed in each trial). Benefiting from the first layer, the proposed RRT*-CFS has a success rate of 100% and requires fewer iterations to converge compared to CFS, SQP, CHOMP, and PRM-CFS. RRT*-CFS also has the lowest average cost, which is contributed by the semi-optimal initial path given by RRT* and the optimization process by CFS. This also indicates that RRT*-CFS is more likely to converge to nearly-global optima. Comparing RRT*-CFS and RRT*-SQP, we see that RRT*-CFS requires less computation time. This is because CFS exploits the geometry of the motion planning problem. Also, RRT*-CFS is faster than PRM-CFS because RRT* is faster than PRM in these motion planning problems. Even though CFS is the quickest algorithm besides RRT* in both 4-obstacle and 1-obstacle cases, RRT*-CFS is faster than CFS in the 10-obstacle complex scenarios due to a better initialization and faster convergence rate. Also, CFS is more vulnerable to being trapped in bad local optima; therefore, it struggles to find a trajectory to reach the goal in complex scenarios. In contrast, RRT*-CFS can always find a trajectory if one exists (the failure cases indicate the resulting solutions do not bring the robot to the goal).

Narrow passages

We also test RRT*-CFS in the narrow passage scenario. The robot's goal is again to plan a trajectory to the goal point. However, the robot has to navigate through a narrow pathway to reach the goal point. Figure 2.5 shows a planning result in one of these scenarios. RRT*-CFS successfully explores the narrow passage and plans a feasible trajectory (blue-circle-line). On the other hand, CFS, SQP, and CHOMP failed to find a solution. The CFS trajectory (blue-star-line) and the SQP trajectory (gray-star-line, under the CFS trajectory) stop in front of the wall while the CHOMP trajectory (red-star-line) directly penetrates the wall. These two failures are due to the lack of local gradient information, which is crucial for optimization-based methods that solve each iteration by calculating the gradient using the result of the previous iteration. The 99% success rate in Table 2.1 for CFS and CHOMP is due to the same reason, where the initial point and the goal point are both lying on the symmetric axis of the obstacle. Even though using local higher-order information can enable the algorithms to deal with some of these scenarios (e.g., SQP), solving the narrow passage problem is still hard for optimization-based methods alone. In some cases, there is no information in the higher-order terms (e.g., when facing the wall, the second-order derivative of a line is zero). This observation again demonstrates the need for a sampling

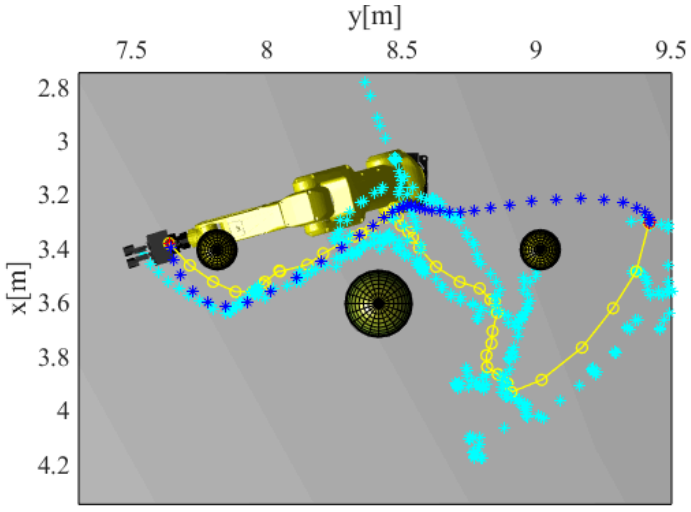
mechanism in the motion planning algorithm so that the algorithm can explore beyond the local gradient information. The simulation result of narrow passage environments is summarized in Table 2.2. We see that RRT*-CFS improves the cost without sacrificing too much computation time. Although PRM-CFS also finds a solution and requires fewer iterations, its computation time is two magnitudes larger than RRT*-CFS.

2.4.6 5D motion planning for a manipulator

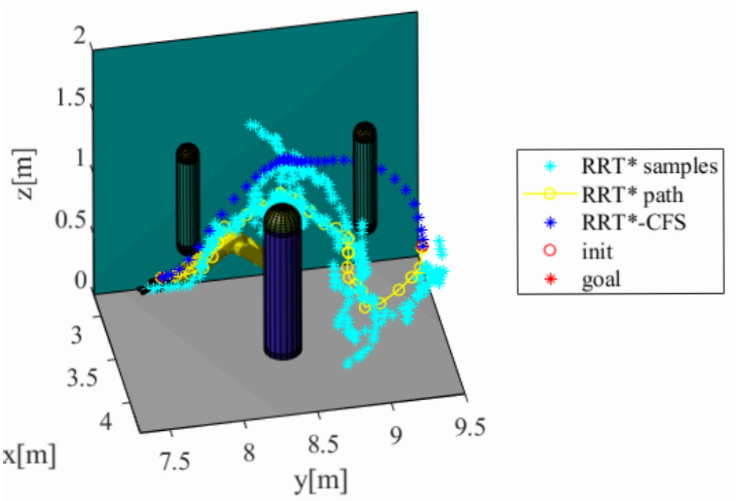
In 5D motion planning simulations, the manipulator's goal is to reach the goal configuration from the initial configuration while avoiding obstacles. Table 2.3 shows the result averaging over 20 trials. Similar to the trend in 2D cases, RRT*-CFS has a lower average cost compared to CFS. CFS performs better in scenarios with one or two obstacles in terms of computation time. RRT*-CFS shows its strength in complex environments and performs better in scenarios with three or four obstacles. Benefiting from the first layer, the proposed RRT*-CFS has a success rate higher than that of CFS. RRT*-CFS also requires fewer iterations to converge compared to CFS. Notice that these planning problems do not satisfy the descriptions in Section 2.2.1 due to the nonlinear transformation from the robot joint space to the safety distance calculated in Cartesian space; nevertheless, RRT*-CFS still converges well empirically. One of the simulation environments is shown in Figure 2.6, where the manipulator moves to the goal configuration without collision. Notice that CFS fails to plan a trajectory that brings the manipulator to the goal configuration in this scenario. With the initial path from RRT*, RRT*-CFS plans a smoother trajectory that navigates through the complex environment compared to the original RRT* trajectory. In general, RRT* has 100% success rate; however, the success rate of RRT*-CFS is 89%. This is due to the linearization error of the collision avoidance constraints during the CFS stage, which causes the convex feasible set to be an empty set in some iterations. Typically, the more obstacles the robot needs to avoid, the more constraints the optimization stage needs to consider; therefore, this problem will likely happen. Similarly, complicated robot kinematics can also increase linearization error and cause the convex feasible set to be empty. In addition, robots in these scenes require a longer plan, i.e., a plan that has a larger planning horizon. This problem motivates the development of the planner presented in Chapter 3.

2.5 Chapter Summary

This chapter presented a comprehensive benchmark that compares motion planning algorithms from different categories and introduced a fast motion planning algorithm, RRT*-CFS, that combined the merits of sampling-based planning methods and optimization-based planning methods. The RRT*-CFS quickly found a feasible and semi-optimal path using RRT* and iteratively refined the solution using CFS. RRT*-CFS had feasibility and global convergence guarantees inherited from CFS in scenarios where obstacles could be represented by disjoint convex objects. Simulation results showed that RRT*-CFS benefited from the



(a) Top view of the planning result.



(b) Side view of the planning result.

Figure 2.6: Planning results using RRT*-CFS in 5D manipulation planning problems.

Table 2.3: Simulation comparison of 5D Motion planning. (Average of 20 trials.)

# obstacles: 1 or 2			
	RRT*	CFS	RRT*-CFS
Time average [s]	5.7	4.74	12.42
Cost	N/A	1.94	**1.66**
# iterations	N/A	20	**15.56**
Success rate (%)	100	95	99
# obstacles: 3 or 4			
	RRT*	CFS	RRT*-CFS
Time average [s]	8.45	114.10	59.67
Cost	N/A	3.78	**3.39**
# iterations	N/A	20	**18.58**
Success rate (%)	100	85	89

hybrid structure. Compared to RRT*, PRM, CFS, SQP, CHOMP, and PRM-CFS, RRT*-CFS had the lowest cost and converged with less number of iterations. RRT*-CFS could also solve planning problems in complex scenarios such as the narrow passage problem, in which CHOMP, SQP, and CFS failed. Even though RRT*-CFS has two layers, the computation time is still competitive in simple scenarios and outperforms other algorithms (except RRT* in terms of time) in complex scenarios. We concluded that the hybrid structure indeed brought strong performance.

Chapter 3

Long-horizon Motion Planning

3.1 Introduction

Motivated by the failure cases mentioned in Chapter 2, this chapter presents a hybrid robot motion planner that generates long-horizon and collision-free robot motion plans. Among the different planning problems, long-horizon motion planning is especially challenging in terms of finding a feasible solution and improving the quality of the solution in environments full of obstacles (Figure 3.1). With only a naive initialization (e.g., a straight line from the initial to the goal in the state space), optimization-based planners struggle to find a solution that can travel long distances and make multiple big turns, which are often needed in long-horizon motion planning [110]. On the other hand, search-based methods require more memory space to store the pre-computed graph; and a well-designed heuristic must be provided to guide the search. In this light, search-based methods face more significant challenges when the robot has many degrees of freedom and when generating an effective heuristic is difficult due to the cluttered environment [42]. Also, the pre-computed graph and the heuristics are hard to reuse if the obstacle configuration in the environment is changed.

A strong candidate for long-horizon path planning is RRT*, which is known for its efficient and exploratory properties of finding a feasible and semi-optimal [78, 72] path in the shortest time. Although RRT* can converge to the optimal solution given infinite computation time, users often terminate the algorithm at a time limit, resulting in an unsmooth path. In addition, the constructed path often does not consider the robot dynamics since deliberately considering these constraints weakens the computation advantage. Most RRT* variations proposed lately [142] remain to be "path planners" rather than "motion planners." In contrast, planning-by-modification methods are very efficient given a good initialization. This motivates the development of hybrid motion planners [112, 108, 43, 129, 101], which use planning-by-construction methods to generate a feasible reference path and use optimization-based methods to polish the solution.

One of the computational bottlenecks of hybrid motion planners is the optimization stage.

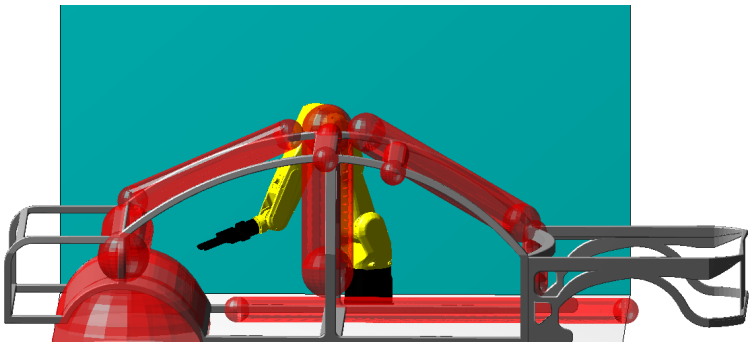


Figure 3.1: A manipulator navigating through a car frame in a factory.

An optimization-based motion planner that does not exploit the structure of the planning problem would typically scale in complexity with the cube or square of the planning horizon H, i.e., $\mathcal{O}(H^3)$ to $\mathcal{O}(H^2)$ [126, 124]. This is especially costly when solving long-horizon planning problems where the number H is large. To mitigate this problem, researchers propose splitting the problem into several sub-problems and developing an iterative update strategy to combine the distributed solutions and find the optimal trajectory [151, 152, 168]. This formulation also allows the solver to exploit parallel computation power.

This chapter incorporates the hybrid planner structure and proposes RRT*-sOpt, focusing on long-horizon motion planning scenarios. We develop a segmented trajectory optimization (sOpt) layer that segments the initial reference from the RRT* layer equally by time and performs optimization iteratively and efficiently. We also notice that the optimal number of segments can vary as the optimization process continues. In other words, "merging" segments can benefit the computation time because it enforces consensus between neighboring segments and may reduce the number of iterations needed for the overall trajectory to converge. Therefore, we identify the conditions for merging segments and demonstrate its effectiveness.

Compared to the previous work, RRT*-CFS [101], we improve the optimization layer to exploit the parallel computation power. Simulation results show that RRT*-sOpt significantly improves computation time and is more robust at successfully finding long-horizon motion plans in complex environments. Our main contributions are threefold as follows:

- We develop a segmented trajectory optimization strategy (sOpt) with a segment merging scheme.
- The proposed RRT*-sOpt algorithm improves upon RRT*-CFS in computation time and robustness.
- We implement RRT*-sOpt and demonstrate its success with extensive simulation on multiple robot platforms (video is publicly available here).

3.2 Problem Formulation and Related Works

The baseline problem formulation of the motion planning problems in this chapter is the same as that mentioned in Section 2.2.1. However, the dimension n is substantial in long-horizon motion planning problems. On top of that, we include a mobile manipulator as a testing platform, which has a state space nearly twice the size of a manipulator. This makes n even larger. Therefore, solving these motion planning problems is more challenging than solving those in Chapter 2.

3.2.1 Hybrid motion planning algorithms

Many works have focused on hybrid planners [112, 108, 43, 129, 101], as reviewed in Section 2.2.5. Methods such as lattice A* search, bidirectional RRT [86], or roadmaps are commonly used in the planning-by-construction stage; while methods such as SQP [154], CFS [111], and TrajOpt [146] are often used to polish the solution. Hybrid planners have better computation time efficiency than non-hybrid ones and can also solve harder problems, such as the narrow passage problem [101]. Chapter 2 presented RRT*-CFS and demonstrated its computational speed advantages over its counterparts. Although its performance is robust with most of the test cases in the chapter, it wanes when applied to long trajectories and higher-dimensional problems. This is also a common problem for most hybrid planners because the problem complexity normally scales with the cube or square of the planning horizon and the robot state space. Therefore, we develop RRT*-sOpt to mitigate this problem.

3.2.2 Segmented trajectory optimization algorithms

In recent years, researchers have proposed to develop planners that enable the exploitation of parallel computing with multi-core CPUs/GPUs. Many have utilized the alternating direction method of multipliers (ADMM) [18] to solve a highly non-linear and non-convex problem in a distributed manner. Authors of [151] proposed to split the problem into two subproblems that consider dynamic and collision avoidance constraints, respectively, and combine the two solutions with a consensus update. Nevertheless, these methods gained little speed-up since the number of waypoints (i.e., time steps) in the subproblems remained the same. In addition, this splitting method may not split the complexity evenly, which results in wait-time for the more complex process to finish [168]. Authors of [152] achieved a distributed structure by decomposing the mobile-manipulator trajectory optimization into a sequence of convex QPs. However, this work did not demonstrate collision avoidance. Authors of [168] proposed a similar distributed formulation for robot motion planning with collision avoidance. Yet, these works do not consider scenarios requiring long-horizon motion planning and may still suffer from naive initialization. In this work, we leverage the hybrid structure to obtain an initial reference and focus on exploiting parallel computation power using sOpt for long-horizon motion planning with collision avoidance.

3.3 The RRT*-sOpt Algorithm

The proposed RRT*-sOpt inherits the merits of hybrid planners and segmented trajectory optimization. The RRT*-sOpt algorithm solves the non-convex motion planning problem by quickly finding a feasible and semi-optimal path and then iteratively refining the solution using sOpt. The RRT*-sOpt has three main features.

- RRT*-sOpt has stochasticity due to the random sampling process in RRT*. This helps RRT*-sOpt find a feasible path and avoid bad local optima that optimization-based algorithms may suffer.
- Both the RRT* layer and the sOpt layer can be implemented with parallel computation, which allows us to reduce the computation time significantly.
- sOpt leverages parallel computation to mitigate the high dimensionality of long-horizon planning problems and implements a segment merging strategy to reduce computation time further.

The RRT*-sOpt algorithm is summarized in Algorithm 2. We introduce the details of the proposed method in the following sections.

3.3.1 The parallel RRT*

The parallel RRT* works in the same way as mentioned in Section 2.3, which results in a reference trajectory $\mathbf{x}^{(0)}$ that can serve as initialization in the optimization stage. In the following, we will introduce the segmented trajectory optimization process.

3.3.2 The segmented trajectory optimization

The second part of the algorithm is the sOpt layer, which solves the planning subproblems iteratively.

Trajectory segmentation

An illustration of the terminology for sOpt is shown in Figure 3.2(a). We denote a split of the H-horizon-trajectory as $\mathbf{x}_j$ (the purple line in Figure 3.2(a)). Given an integer number N, a trajectory with $2N$ splits is $\mathbf{x} := [\mathbf{x}_1, \ldots, \mathbf{x}_j, \ldots, \mathbf{x}_{2N}]$, each containing $(H-1)/2N+1$ waypoints. The indices of the split-points, i.e., the indices of the starting and ending point of each split are stored in a set $\mathbf{W} = \{w_1, w_2, \ldots, w_j, \ldots, w_{2N+1}\}$ where $w_j \in \{0, 1, \ldots, H\}$. A segment (the green line in Figure 3.2(a)) contains two splits, denoted as $\mathbf{x}_{[i,i+(H-1)/N]}$. To ensure connectivity between segments, the end point of the previous segment is set to be the same as the first point of the succeeding segment. As shown in Figure 3.2(b), in the odd iterations, i.e. kth iteration where $k = 1, 3, \ldots$, all the segments start with index in the odd entries of $\mathbf{W}$, i.e., $\mathbf{W}^{odd}$. In the even iterations, segments start with index in

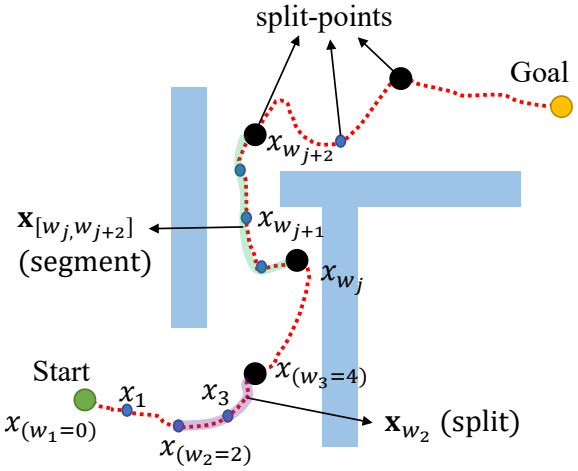
Algorithm 2: RRT*-sOpt

input $\theta_0, \theta_{goal}, n_{samples}, \mathcal{O}, N_{segments}$
while $! \exists\ \boldsymbol{\theta}^{RRT^*}$ **do**
  $\boldsymbol{\theta}^{RRT^*} \leftarrow$ `parallel_RRT*`$(\theta_0, \theta_{goal}, n_{samples}, \mathcal{O})$
$\mathbf{x}^{(0)} \leftarrow$ `generate_reference`$(\boldsymbol{\theta}^{RRT^*})$
$\mathbf{W}^{(0)} \leftarrow$ `split_reference`$(\mathbf{x}^{(0)}, N_{segments})$
while *termination conditions not met* **do**
  $mode \leftarrow$ `odd_or_even`$()$
  for $w_j^{(k)} \in \mathbf{W}^{(k),mode}$ **do**
    $\chi^{(k)}_{[w_j,w_{j+2}]} \leftarrow$ `Obs_select`$(\mathbf{x}^{(k)}_{[w_j,w_{j+2}]}, \mathcal{O})$
    $\mathbf{x}^{(k+1)}_{[w_j,w_{j+2}]} \leftarrow$ `OPT`$(\mathbf{x}^{(k)}_{[w_j,w_{j+2}]}, \chi^{(k)}_{[w_j,w_{j+2}]})$
  for $w_j^{(k+1)} \in \mathbf{W}^{(k),mode}$ **do**
    $d \leftarrow$ `iter_prog`$(\mathbf{x}^{(k)}_{[w_j,w_{j+4}]}, \mathbf{x}^{(k+1)}_{[w_j,w_{j+4}]})$
    if $d \leq \frac{2\epsilon H}{N_{seg}}$ **then**
      Remove $w^{(k)}_{j+2}$ **from** $\mathbf{W}^{(k)}$
  $\mathbf{x}^{(k+1)} \leftarrow$ `resample_traj`$(\mathbf{x}^{(k+1)})$
  $\mathbf{W}^{(k+1)} \leftarrow$ `split_reference`$(\mathbf{x}^{(k+1)}, \mathbf{W}^{(k)})$
  $k \leftarrow k+1$
return $\mathbf{x}^{(k)}$

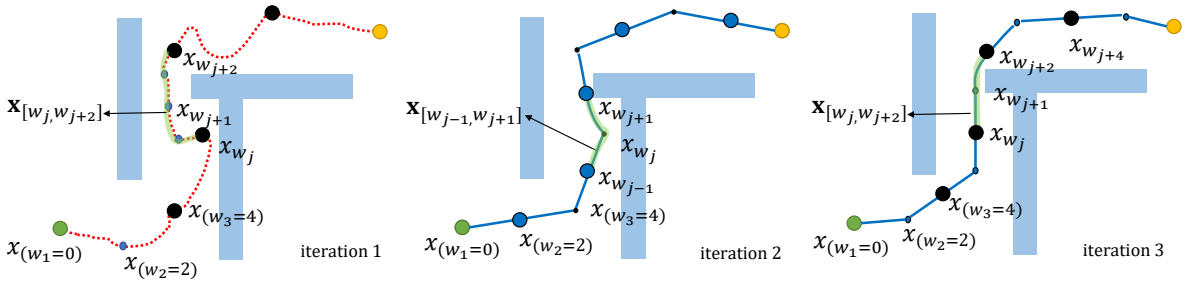
the even entries of $\mathbf{W}$, i.e., $\mathbf{W}^{even}$. For example, if $N = 3$, the first set of segments are $\{\mathbf{x}_{[w_1,w_3]}, \mathbf{x}_{[w_3,w_5]}, \mathbf{x}_{[w_5,w_7]}\}$; then, it becomes $\{\mathbf{x}_{[w_2,w_4]}, \mathbf{x}_{[w_4,w_6]}, \mathbf{x}_{w_1}, \mathbf{x}_{w_6}\}$ in the next iteration. The last two elements are the splits at the beginning and the tail.

Algorithm 3: Obstacle selection

input $\mathbf{x}^{(k)}_{[w_j,w_{j+2}]}, \mathcal{O}$
`Obs_getID`$(\cdot) \leftarrow$ `Watershed`$(\mathcal{O})$
for $x_i \in \mathbf{x}^{(k)}_{[w_j,w_{j+2}]}$ **do**
  $id_i \leftarrow$ `Obs_getID`(x_i)
$\chi^{(k)}_{[w_j,w_{j+2}]} = \bigcap_{id}\{\mathbf{x} : h_{id}(\mathbf{x}^{(k)}_j) + \nabla^\top h_{id}(\mathbf{x}^{(k)}_j)(\mathbf{x} - \mathbf{x}^{(k)}_j) \geq 0\}$
return $\chi^{(k)}_{[w_j,w_{j+2}]}$



(a) The illustration of a segmented trajectory.



(b) illustrations of the segment selection for the first three iterations during sOpt (the other tree figures on the right).

Figure 3.2: The illustration of a segmented trajectory and the iterative optimization process.

Segmented trajectory optimization

The obstacle avoidance constraints of the planning problem are also distributed to each segment. As shown in Algorithm 3, in the kth iteration, we utilize the function `watershed` from MATLAB to select nearby obstacles for each segment. The safety functions associated with these obstacles are linearized at the reference segment, $\mathbf{x}^{(k)}_{[w_j,w_{j+2}]}$, to formulate the constraints as a convex feasible set, $\chi^{(k)}_{[w_j,w_{j+2}]}$ [111]. The planning subproblem optimizes each segment according to a cost function that has the form: $f(\cdot) = \|\mathbf{x} - \mathbf{x}_{goal}\|^2_Q + \lambda\|\mathbf{x}\|^2_R$. By fixing the initial and the goal waypoints to stay at x_{w_j} and $x_{w_{j+2}}$ respectively, we formulate

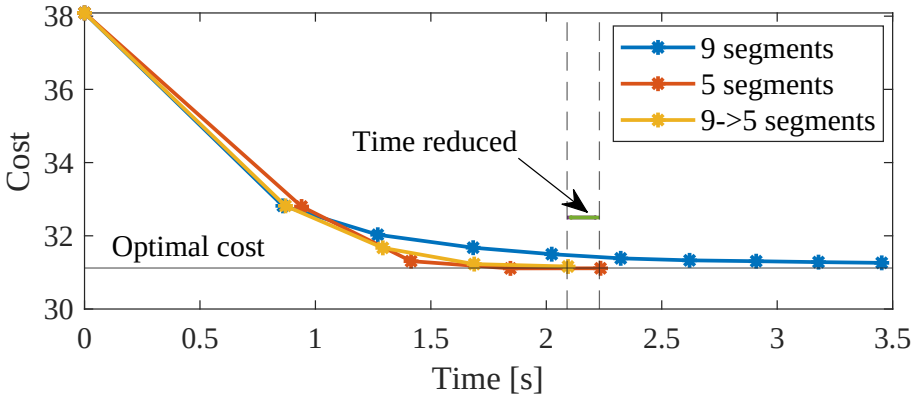


Figure 3.3: An example of computation time reduced by segment merging in 2D planning. (Notice that we hard-coded this merge to generate this plot for more precise visualization. In the simulations, merging typically occurs in later iterations.)

the subproblem as follows:

$$\begin{aligned} \mathbf{x}^{(k+1)}_{[w_j,w_{j+2}]} = \arg\min_{\mathbf{x}} \quad & f(\mathbf{x}), \\ s.t. \quad & f_{ki}(\mathbf{x}) \in \chi^{(k)}_{[w_j,w_{j+2}]}, \\ & x_0 = x_{w_j}, \\ & x_{(H-1)/N} = x_{w_{j+2}}, \end{aligned} \tag{3.1}$$

where $f_{ki}(\cdot)$ is the robot kinematic model. Note that the selection of the starting and ending points alternates in each iteration so that the waypoints fixed in the present iteration will be optimized in the next iteration. Ultimately, the full trajectory can be optimized iteratively.

Merging segments

We observe that performances of sOpt with different numbers of segments are different at different stages. As shown in Figure 3.3, a sOpt with more segments reduces the cost quickly at the beginning but converges slowly later on (blue line) compared to a sOpt with fewer segments, which performs oppositely (orange line). Therefore, we propose to merge neighboring segments in later iterations when quick convergence is desired (Figure 3.4). As shown in Algorithm 2 line 11 to 14, the function `iter_prog` calculates the cost of two neighboring segments and compares it with the cost of the previous iteration to quantify the progress made by the optimization in that iteration, i.e.,

$$iter\_prog = \|f(\mathbf{x}^{(k)}_{[w_j,w_{j+4}]}) - f(\mathbf{x}^{(k+1)}_{[w_j,w_{j+4}]})\|. \tag{3.2}$$

By selecting a threshold ϵ, merging happens when `iter_prog` $\leq 2\epsilon H/N$.

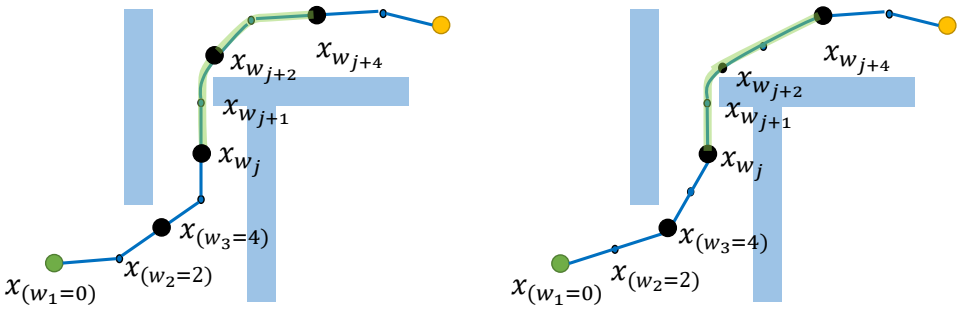


Figure 3.4: Illustration of segment merging.

Trajectory resampling

Since the path length will be reduced after every iteration, we use `resample_traj` to find the new planning horizon $H^{(k)}$ according to the desired robot operating speed. We record the new set of split-point indices with `split_reference`.

Termination conditions

The algorithm terminates if: (1) the algorithm reaches the maximum number of iterations (i.e., 20 iterations) or (2) the cost of the entire trajectory between iterations is less than the threshold ϵ, i.e., $\|f(\mathbf{x}^{(k)}) - f(\mathbf{x}^{(k+1)})\| \leq \epsilon$.

3.4 Applications

We use three different robot platforms to test RRT*-sOpt: a mobile robot (Figure 2.2 (left)), a manipulator (Figure 2.2 (right)), and a mobile manipulator (Figure 3.5). The kinematic models of the mobile robot and the manipulator are the same as (2.6) and (2.7), respectively. Here, we introduce the kinematic model of the mobile manipulator.

3.4.1 The mobile manipulator model

The mobile manipulator used in this work, TurtleBot3 with OpenManipulator (TB3O), is composed by a 2-DoF mobile platform, TurtleBot3, and a 4-DoF manipulator, OpenManipulator, which are developed by ROBOTIS. As shown in Figure 3.5 (right), the world frame is defined as $F_w - X_wY_wZ_w$. Each link, $link_i$, has an associated body-fixed frame $F_i - X_iY_iZ_i$, $i \in \{1, 2, 3, 4, 5\}$. In addition to the original four joints of the manipulator, we added a virtual link, $link_1$, to the system for the convenience of constructing the motion planning optimization problem.

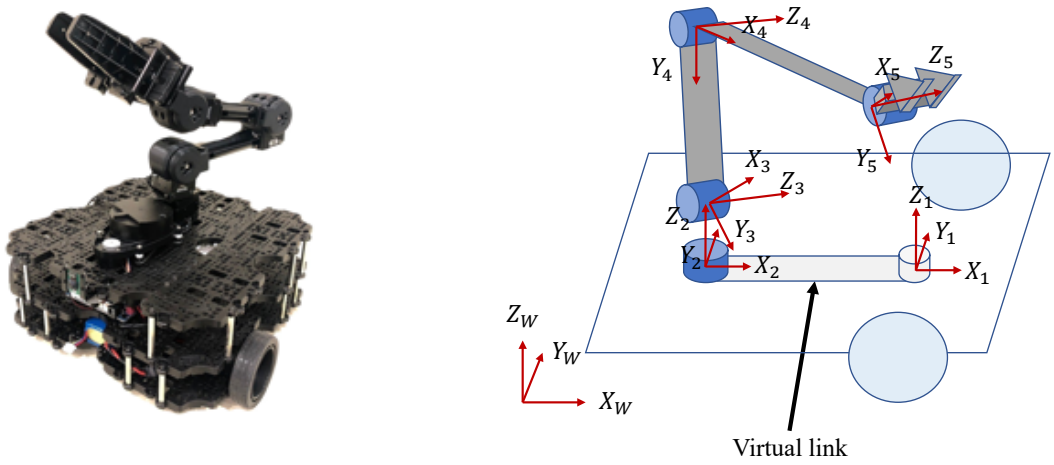


Figure 3.5: A mobile manipulator (left) and the coordinate system of the mobile manipulator (right).

To establish the kinematic model of the mobile manipulator, we first examine the non-holonomic mobile platform and the manipulator separately.

Model of the mobile platform

Denote the states of the mobile platform as $z_{p,k} = [x_k, y_k, \theta_{p,k}, v_k, \omega_{p,k}]^\top$, where (x_k, y_k) and v_k are the location and speed of the origin of the body-fixed frame $F_1 - X_1Y_1Z_1$ relative to the world frame $F_w - X_wY_wZ_w$, respectively. $\theta_{p,k}$ and $\omega_{p,k}$ are the angle and angular velocity between X_w (the x-axis of the frame $F_w - X_wY_wZ_w$) and X_1 (the x-axis of the frame $F_1 - X_1Y_1Z_1$), respectively. The inputs are the linear and angular acceleration denoted as $u_{p,k} = [a_{p,k}, \alpha_{p,k}]^\top$. k denotes time. The nonlinear kinematic model is:

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ \theta_{p,k+1} \\ v_{k+1} \\ \omega_{p,k+1} \end{bmatrix} = \begin{bmatrix} x_k \\ y_k \\ \theta_{p,k} \\ v_k \\ \omega_{p,k} \end{bmatrix} + \begin{bmatrix} v_k T_s \cos(\theta_{p,k} + 0.5 T_s \omega_{p,k}) \\ v_k T_s \sin(\theta_{p,k} + 0.5 T_s \omega_{p,k}) \\ T_s \omega_{p,k} \\ T_s a_{p,k} \\ T_s \alpha_{p,k} \end{bmatrix}, \tag{3.3}$$

where T_s is the sampling time.

Model of the manipulator

Denote the states of the manipulator as $z_{m,k} = [\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \omega_1, \omega_2, \omega_3, \omega_4, \omega_5]_k^\top$, where θ_i and ω_i, $i \in \{1, 2, 3, 4, 5\}$, are the angle position and the angular velocity of ith joint,

respectively. The inputs are the angular acceleration at each joint, denoted as $u_{m,k} = [\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5]_k^\top$.

The homogeneous transformation matrix of the frame $F_1 - X_1Y_1Z_1$ with respect to the frame $F_w - X_wY_wZ_w$ at the time step k is describe as:

$$
{}^{F_w}_{F_1}\mathbf{T}_k = \begin{bmatrix} {}^{F_w}_{F_1}\mathbf{R}_x(\gamma_1) {}^{F_w}_{F_1}\mathbf{R}_z(\theta_{1,k}) & o_{1,k} \\ 0_{1\times 3} & 1 \end{bmatrix}, \tag{3.4}
$$

where,

$$
{}^{F_w}_{F_1}\mathbf{R}_x(\gamma_1) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\gamma_1) & -\sin(\gamma_1) \\ 0 & \sin(\gamma_1) & \cos(\gamma_1) \end{bmatrix},
$$

$$
{}^{F_w}_{F_1}\mathbf{R}_z(\theta_{1,k}) = \begin{bmatrix} \cos(\theta_{1,k}) & -\sin(\theta_{1,k}) & 0 \\ \sin(\theta_{1,k}) & \cos(\theta_{1,k}) & 0 \\ 0 & 0 & 1 \end{bmatrix},
$$

$\gamma_1 = 0$, and $o_{1,k} = [x_{1,k}, y_{1,k}, z_{1,k}]^\top$, which is the origin of the frame $F_1 - X_1Y_1Z_1$ with respect to the world frame. The homogeneous transformation matrix of $F_i - X_iY_iZ_i$ with respect to $F_{i-1} - X_{i-1}Y_{i-1}Z_{i-1}$ for $i \in \{2, 3, 4, 5\}$ is:

$$
{}^{F_{i-1}}_{F_i}\mathbf{T}_k = \begin{bmatrix} {}^{F_{i-1}}_{F_i}\mathbf{R}_x(\gamma_i) {}^{F_{i-1}}_{F_i}\mathbf{R}_z(\theta_{i,k}) & o_{i,k} \\ 0_{1\times 3} & 1 \end{bmatrix}, \tag{3.5}
$$

where $[\gamma_2, \gamma_3, \gamma_4, \gamma_5] = [0, -0.5\pi, 0, 0]$, and $o_{i,k}$ is the origin of the coordinate $F_i - X_iY_iZ_i$ with respect to $F_{i-1} - X_{i-1}Y_{i-1}Z_{i-1}$. The derivation of the transformation matrices is important to find the relationship between the robot state and the robot end-effector's position and orientation, i.e., end-effector pose. In real-world applications, one can either set the goal to be a tuple of the end-effector pose and the mobile base pose. However, this will introduce more nonlinear equality constraints due to these transformations. Instead, one can use inverse-kinematics to first find the target joint angles and base pose, then conduct planning in the joint space. When working in the joint space, the kinematic model of the manipulator is a linear system that is the same as (2.7) with slight changes on the subscript:

$$
z_{m,k+1} = \mathbf{A}_m z_{m,k} + \mathbf{B}_m u_{m,k}. \tag{3.6}
$$

Connection of the platform and the manipulator

The two systems, the mobile platform and the manipulator, are connected with three equality constraints, $\theta_{p,k} = \theta_{1,k}$, $\omega_{p,k} = \omega_{1,k}$, and $\alpha_{p,k} = \alpha_{1,k}$. The full state of the mobile manipulator is denoted as $z_k = [z_{p,k}^\top, z_{m,k}^\top]^\top$, and the input is denoted as $u_k = [u_{p,k}^\top, u_{m,k}^\top]^\top$. The full model is a non-linear model:

$$
z_{t+1} = g(z_t, u_t). \tag{3.7}
$$

3.4.2 The motion planning problem

Given the initial state, z_0, we obtain $\mathbf{z} = f_{ki}(z_0, \mathbf{u})$ by concatenating the kinematic function ((2.6) or (2.7) or (3.7)) throughout the planning horizon. The motion planning problem has the same form as Problem 2. Note that the dimensions of $\mathbf{x}$ and $\mathbf{u}$ are much larger in this chapter.

3.4.3 Implementation of sOpt

We denote the segment that starts at split-point z_{w_j} as $\mathbf{z}^{(k)}_{[w_j,w_{j+2}]}$, the associating input vector as $\mathbf{u}^{(k-1)}_{[w_j,w_{j+2}]}$, and the convex feasible set as $\chi^{(k)}_{[w_j,w_{j+2}]}$. For simplicity, we drop the subscript in this section. For each segment in each iteration, we update $\mathbf{z}^{(k)} = f_{ki,z_{w_j}}(\mathbf{u}^{(k-1)})$ at iteration $k = 2, 3 \ldots$. The states $\mathbf{z}^{(1)}$ and the inputs $\mathbf{u}^{(0)}$ are initialized in the same way as in Section 2.4.3. The convex feasible set, $\chi^{(k)}_{[w_j,w_{j+2}]}$, is determined by $\mathbf{z}^{(k)}_{[w_j,w_{j+2}]}$. Given the feasible set $\Gamma = \bigcap_{\bar{j},t}\{\mathbf{z} : h_{\bar{j}}(z_t) \geq 0\}$ ($\bar{j}$ numerates over obstacles and t numerates over time steps in a segment, i.e., $t = 1, \ldots, \bar{H}$. The function $h_{\bar{j}}(z_t)$ if defined similarly to $h_j(z_t)$ in Section 2.4.3.), the results of the previous iteration ($\mathbf{u}^{(k-1)}$ and $\mathbf{z}^{(k)}$), and the function $f_{ki,z_{w_j}}(\mathbf{u})$, we can construct the convex feasible set as:

$$\chi^{(k)}_{[w_j,w_{j+2}]} = \bigcap_{\bar{j},t}\{\mathbf{u} : h'_{\bar{j},t}(\mathbf{u}, z^{(k)}_t, \mathbf{u}^{(k-1)}) \geq 0\}, \tag{3.8}$$

where $h'_{\bar{j},t} = h_{\bar{j}}(z^{(k)}_t) + \nabla^\top h_{\bar{j}}(z^{(k)}_t)\nabla f_{ki,z_{w_j},t}(\mathbf{u}^{(k-1)})(\mathbf{u} - \mathbf{u}^{(k-1)})$. Therefore, the iterative sub-problem for each segment with full notation is as follows:

$$\begin{aligned} \mathbf{u}^{*(k)}_{[w_j,w_{j+2}]} = \arg\min_{\mathbf{u}} \quad & f_{z_{w_j}}(\mathbf{u}), \\ s.t. \quad & \mathbf{u} \in \chi^{(k)}(\mathbf{u}^{(k-1)}_{[w_j,w_{j+2}]}, \mathbf{z}^{(k)}_{[w_j,w_{j+2}]}), \\ & z_{\bar{H}} = z_{w_{j+2}}, \end{aligned} \tag{3.9}$$

where $f_{z_{w_j}}(\mathbf{u}) = \|f_{ki,z_{w_j}}(\mathbf{u}) - \mathbf{z}_{w_{j+2}}\|^2_Q + \lambda\|\mathbf{u}\|^2_R$.

3.4.4 Simulation setup

We show the simulation results in the following sections. The simulation is conducted in `Matlab R2021a` on a desktop with a 3.7 GHz Intel Core i9-10900K CPU. Parallel computation can be realized by using the function `parfor`. The stopping criteria for sOpt are the same. The threshold is set at $\epsilon = H \times 10^{-3}$. Obstacles in these scenarios are either convex or wrapped around with convex geometries.

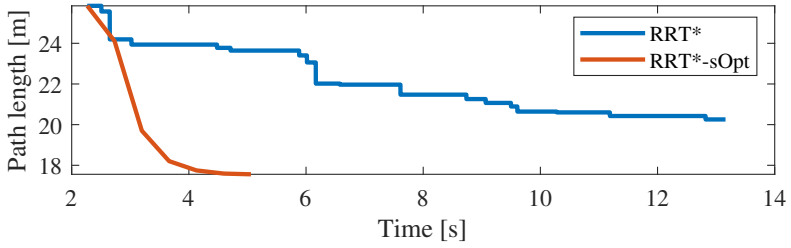


Figure 3.6: Path length reduction performance comparison between RRT* and RRT*-sOpt.

3.4.5 Simulation results

Concept verification

One may argue that RRT* alone can also find a near-optimal solution in a short amount of time. Note that RRT* can optimize against different costs. We choose "path length" as the cost, the most straightforward cost function in 2D planning. To justify the need for having the two-stage strategy, we run RRT* for roughly 13 seconds and compare its path length reduction performance with RRT*-sOpt. As shown in Fig. 3.6, RRT* converges slowly while RRT*-sOpt, using the first returned RRT* solution for initialization, converges quickly to a better local optimal. This confirms the effectiveness of having an optimization solver to improve the solution when the global optimality guarantee isn't necessary. Instead of RRT*, an RRT solution can also serve as an initialization; however, we notice that RRT* can better utilize the samples by rewiring the path without increasing much computation time. Therefore, we adopt RRT* and terminate it once a solution is found. The "optimization" of RRT* only happens while RRT* is sampling for the first solution.

2D motion planning

The scenarios here are designed to simulate mobile platforms' long-distance 2D motion planning scenes. Some of the planning results are shown in Figure 3.7. Note that most of the test cases require the robot to travel a long distance, i.e., more than 100 waypoints are needed for a motion plan. The performance comparison of RRT*-sOpt with different numbers of segments is shown in Table 3.1. (The computation time for RRT*-sOpt includes both the time spent during the RRT* and optimization stages. The sOpt time can be obtained by subtracting the RRT* time from the total time.) First, we observe that all RRT*-sOpt can smooth the RRT* reference and achieve similar final costs, which are noticeably smaller than the costs of the original RRT* solutions. This empirically shows that the algorithm can converge to a local optimum given the RRT* reference. Second, all of the RRT*-sOpt are much quicker than RRT*-Opt. This verifies our claim that trajectory splitting reduces computation time by leveraging parallel computation power. Without merging, $s = 7$ has the

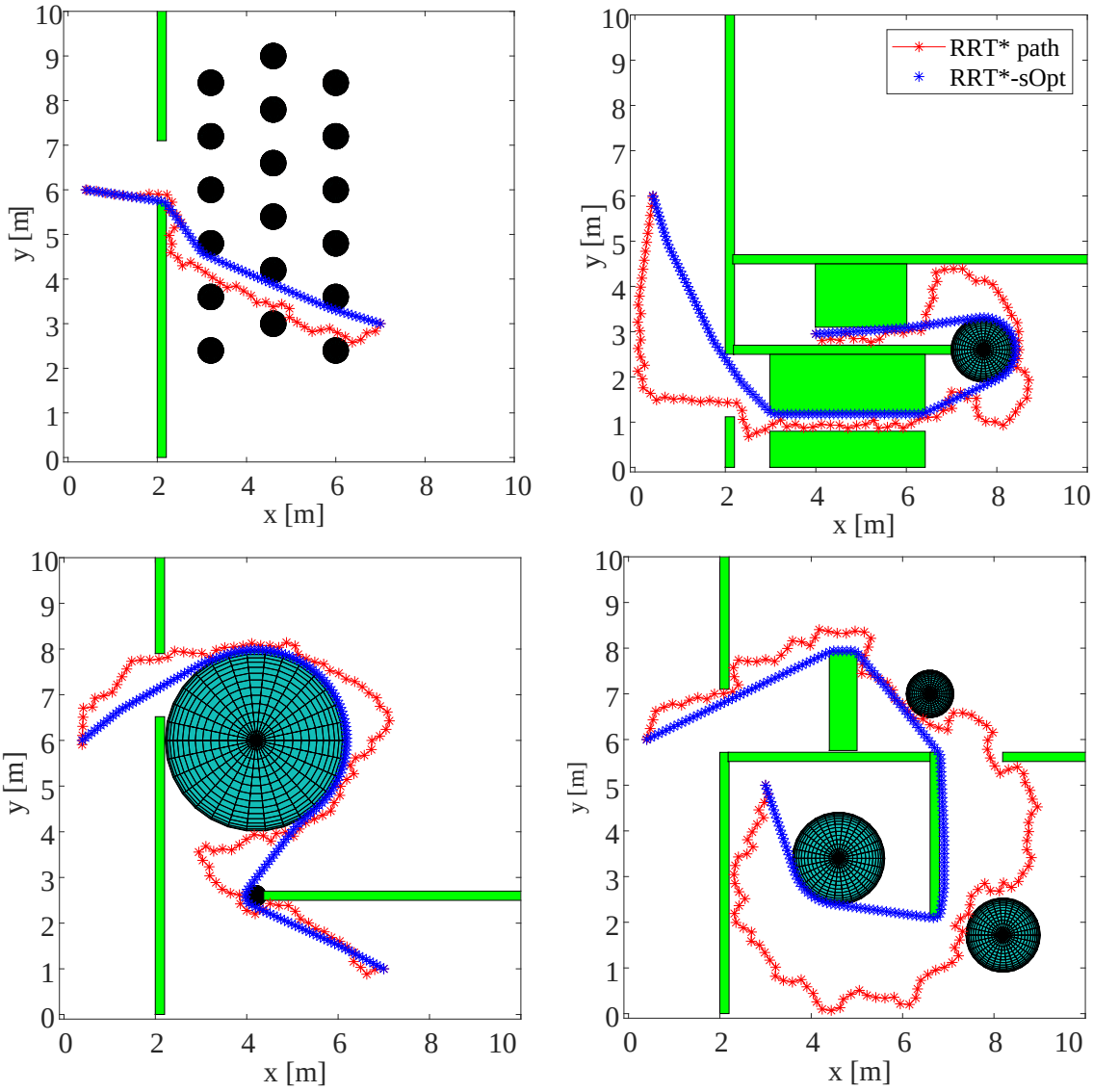


Figure 3.7: Simulation results of the 2D motion planning.

Table 3.1: Simulation comparison of 2D planning with 5 to 20 obstacles. (The results are the average of 25 trials. The notion "$7 \rightarrow 3.9$" in # **Segments** in "auto-merge segments" means that the RRT*-sOpt starting from 7 segments on average terminates at 3.9 segments. The computation time standard deviation for RRT*-sOpt only considered the time variation during the optimization stage.)

	RRT*	RRT*-sOpt						
		# Segments fixed					auto-merge segments	
# Segments		1	3	5	7	9	$7 \rightarrow 3.9$	$9 \rightarrow 5.8$
Computation time average [s]	19.82	80.32	22.32	21.63	21.43	22.01	**21.42**	21.81
Computation time standard deviation [s]		36.93	1.39	1.01	0.98	1.18	0.93	1.03
Cost	37.69	31.24	31.10	30.82	31.08	30.93	31.10	31.16
# iterations		13.8	6.5	5.9	5.5	7.7	5.5	7.1
Success rate (%)	100	100	100	100	100	100	100	100

shortest computation time and number of iterations. On the other hand, the computation time is further reduced when merging is allowed. (The notion $7 \rightarrow 3.9$ means that the RRT*-sOpt starting from 7 segments on average terminates at 3.9 segments.)

Motion planning for a 5-Dof manipulator

The scenarios here are designed to simulate general 3D motion planning scenes of manipulators operating in factories. Two categories of settings are created: one with only two obstacles; the other with 7 to 9 obstacles. Some of the planning results are shown in Figure 3.8. The performance comparison is shown in Table 3.2. Similar to the 2D case, RRT*-sOpt is much quicker than RRT*-Opt. Note that RRT*-Opt fails in some of the test cases with two obstacles and fails in most cases with more obstacles. This is mainly due to the linearization and accumulating approximation errors when solving optimization problems. Without segmenting the trajectory, the optimization solver will need to handle all the constraints at once, resulting in a larger chance of failure. On the other hand, RRT*-sOpt mitigates the problem caused by accumulating linearization errors by distributing the environment (safety) constraints to multiple segments. Also, the computation time of RRT*-sOpt is less sensitive to the number of obstacles than that of RRT*-Opt, i.e., RRT*-sOpt performs more robustly against different configurations of the environment. The reduction in computation time due to merging is more evident in manipulator motion planning problems, especially in the second category of settings. In the cases with s going from $7 \rightarrow 4.7$ (on average), the computation speed is 12% faster than the fastest RRT*-sOpt without merging in the optimization stage.

Motion planning for a mobile manipulator

The scenarios here are designed to simulate 3D motion planning scenes of mobile manipulators traveling through hallways while moving the arm to avoid obstacles. One of the planning results is shown in Figure 3.9. The performance comparison is shown in Table 3.3. Since the mobile manipulator kinematic model is more complicated, we formulate the non-linear constraints and use an open-source solver, CasADi [8] with IPOPT [167], to solve the nonlinear planning problem directly. In other words, we use $\Gamma^{(k)}_{[w_j,w_{j+2}]} = \bigcap_{\bar{j},t}\{\mathbf{u} : h_{\bar{j},t}(\mathbf{u}, z_t^{(k)}, \mathbf{u}^{(k-1)}) \geq 0\}$ directly instead of (3.8). Nevertheless, RRT*-Opt still fails in most cases because CasADi also requires approximations during the solving process. The solver is still likely to fail when too many constraints are included in one optimization problem. On the other hand, this problem can be mitigated with segmentation. The computation advantage increases as the number of segments increases, similar to what we observe in the 2D and manipulator cases. This also indicates that sOpt can reduce computation time when working with other solvers. Notice that, in this case, RRT*-sOpt with merging does not reduce the computation time. This is mainly due to the overhead required to set up CasADi when merging occurs. In the future, we will improve the implementation (by choosing a

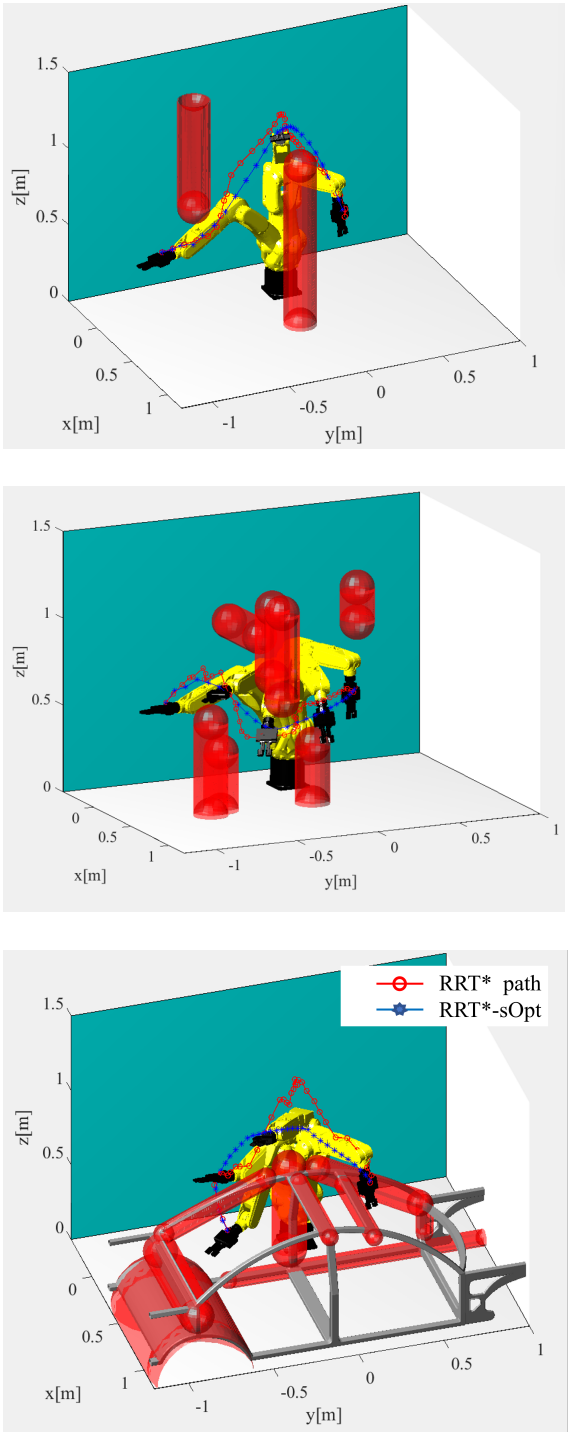


Figure 3.8: Simulation results of the 5-Dof manipulator motion planning.

Table 3.2: Simulation comparison of 5-Dof manipulator motion planning. (Average of 20 trials.)

	# obstacles: 2								
	RRT*	RRT*-sOpt							
		# Segments fixed					Auto-merge segments		
# Segments		1	3	5	7	9	$5 \rightarrow 3.6$	$7 \rightarrow 6.2$	$9 \rightarrow 8.4$
Computation time average [s]	1.49	98.87	4.66	4.16	3.80	3.65	4.14	3.78	**3.57**
Computation time standard deviation [s]		84.23	1.29	0.90	0.93	0.76	1.06	0.76	1.10
Cost	8.40	10.4	7.76	7.76	7.73	7.77	7.75	7.74	7.77
# iterations		6.2	5.6	6.4	7.2	6	6.4	6.8	6
Success rate (%)	100	60	100	100	100	100	100	100	100
	# obstacles: 7 to 9								
	RRT*	RRT*-sOpt							
		# Segments fixed					Auto-merge segments		
# Segments		1	3	5	7	9	$5 \rightarrow 3$	$7 \rightarrow 4.7$	$9 \rightarrow 6.3$
Computation time average [s]	23.26	170.61	28.42	27.48	26.56	26.11	25.77	**25.76**	25.79
Computation time standard deviation [s]		48.51	2.23	1.04	0.56	0.12	0.59	0.10	1.00
Cost	9.35	7.80	7.86	7.57	7.58	7.53	7.69	7.66	7.63
# iterations		6.4	6	7.9	7.3	7.6	6.3	7	7.2
Success rate (%)	100	10	100	100	100	100	100	100	100

Table 3.3: Simulation comparison of mobile manipulator motion planning. (Average of 10 trials.)

	RRT*	RRT*-sOpt				
		# Segments fixed				Auto-merge
# Segments		3	5	7	9	$7 \rightarrow 4$
Computation time average [s]	16.45	32.32	22.64	**21.67**	21.84	24.16
Computation time standard deviation [s]		3.73	2.01	1.71	1.99	3.37
Cost	39.73	34.91	22.27	22.65	23.14	22.46
# iterations		9	7.8	8	9.2	7.6
Success rate (%)	100	60	100	100	100	100

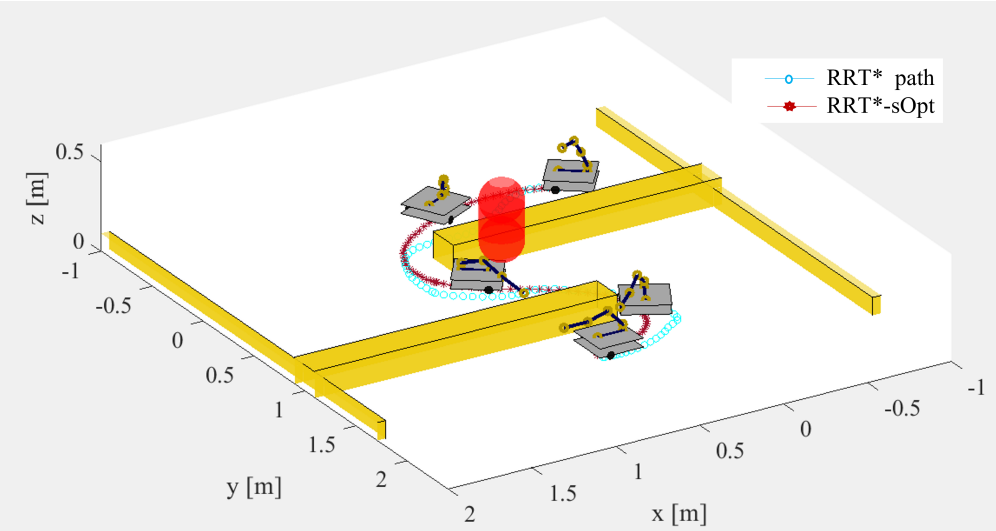


Figure 3.9: Simulation results of a mobile manipulator motion planning.

different optimization solver or coding language) to verify the performance of RRT*-sOpt with merging.

In summary, simulation results show that RRT*-sOpt can successfully plan long-horizon motion plans for mobile robots, manipulators, and mobile manipulators. With the trajectory segmentation, the computation time is significantly reduced and is relatively robust to the different number of obstacles. The novel idea of segment merging is also tested in these settings and has demonstrated the potential to reduce the computation time further. Based on the results of the three models, we suggest using auto-merge-segment RRT*-sOpt for 2D and manipulator planning and fixed-segment RRT*-sOpt for mobile manipulator planning. (The number of segments can be determined based on the planning horizon. Empirically, we suggest that each segment should not handle more than 30 time steps.) The computation time standard deviation of RRT*-sOpt is also small compared to RRT*-Opt. It is worth noticing that the setups with a short average time also tend to have a smaller standard deviation.

3.4.6 Discussion and suggested future works

Though the current implementation of RRT*-sOpt has shortened the computation time substantially, some areas remain to be addressed.

- A method of selecting the initial number of segments is required. One way of determining such a number is to choose a horizon for the initial segments. Then, by calculating the path length of the RRT* reference, we can determine the number of segments needed. However, this selection method does not consider the configuration

of the environment (e.g., obstacles' relative locations). In the future, we aim to develop a method that determines the initial number of segments based on both the reference path length and the environment configuration.

- The merging condition can be improved. The current implementation uses the cost reduction trend to determine when the merging occurs. However, we observed that the optimization problem with tight space usually converges in fewer iterations. This indicates that the environment configuration should be taken into account when designing the merging conditions.
- Although the algorithm converges empirically, the theoretical properties of RRT*-sOpt can be further investigated.

3.5 Chapter Summary

This chapter presented a fast long-horizon motion planning algorithm, RRT*-sOpt, that inherits the computation advantages of its predecessor, RRT*-CFS, and further improves it by incorporating the idea of segmented trajectory optimization. The RRT*-sOpt quickly found a feasible and semi-optimal path using RRT* and iteratively refined the solution using sOpt. Simulation results showed that RRT*-sOpt benefits from the hybrid structure and the ability to distribute the problem complexity to leverage the power of parallel computation. RRT*-sOpt can solve problems that are extremely challenging to stand-alone optimization-based planners, has better final cost than pure sampling-based planners, and has significantly shorter computation time than previous hybrid planners. The novel idea of segment merging was also tested and has shown the potential to reduce the computation time further. We concluded that the hybrid structure with trajectory segmentation had brought the strong performance to RRT*-sOpt for general long-horizon robot motion planning problems.

Chapter 4

Search-based Motion Planning for Articulated Vehicles

4.1 Introduction

In Chapter 2 and 3, we explored methods that only rely on online computation for motion planning. However, as computation power and memory storage improve, search-based motion planners become more popular for robot motion planning because of their ability to utilize off-line computation results, especially for robots with complicated kinematic or dynamic models. This chapter presents a motion planning strategy that utilizes the improved A-search guided tree and data-driven methods to enable autonomous parking of a general 3-trailer with a car-like tractor.

Autonomous tractor-trailer systems have attracted strong interest from industry and academia due to their high cargo transportation efficiency. However, their complicated kinematics pose significant challenges in both control and planning, particularly when reversing maneuvers and collision avoidance are needed.

Early works proposed flatness-based trajectory generation for unconstrained environments [123, 163], while others proposed hierarchical motion planners [148, 91] that first plan a collision-free holonomic path and then iteratively modify it to a kinematically feasible trajectory. Recent works resorted to the state-lattice framework to accomplish kinodynamic planning [37, 35, 115], where the dynamic feasibility and collision avoidance are addressed simultaneously. These algorithms search a graph, where the vertices are discrete states and the edges are from a set of pre-computed motion primitives (MPs). Since the MPs can be generated offline by solving optimal control problems (OCP), the difficulty incurred by the complex kinematics is handled offline. These state-lattice-based planners guarantee resolution optimality and completeness.

A major limitation with state-lattice-based methods is the curse of dimensionality. Works [115, 164] proposed to restrict the MPs to those admitting transition between circular equilibrium configurations. This lowers the dimension of the search space that planning algorithms

explore. However, a large amount of MPs is necessary. The trade-off between the number of MPs and planning success rate/planning accuracy needs to be managed carefully so that the planner can find a path that ends close to the goal within a reasonable time period. A well-informed heuristic function that correctly approximates the true cost-to-go from a state to the goal state is needed to maintain real-time performance [37, 114]. Work [114] combines Euclidean distance and a free-space heuristic look-up table (HLUT) to calculate the heuristic value. Nevertheless, the memory burden of storing the HLUT may be of concern for certain applications.

A remedy to lattice-based limitations is an improved A-search guided tree (i-AGT) [169], which is constructed on-the-fly and accepts off-lattice exploration. This work adopts and extends i-AGT to tractor-trailer systems since moving away from state-lattice-based methods enables using a smaller set of MPs with a similar planning success rate and yields better path quality. We propose a construction process to generate the set of MPs for off-lattice use and analyze its connectivity to achieve good performance. On top of these advantages, i-AGT groups MPs into various modes with their associated priorities, allowing a mode selection process to improve the computation efficiency of node expansion. Unlike lattice-based methods where the HLUT can readily construct the cost-to-go, i-AGT with off-lattice nodes requires a more sophisticated design to estimate the cost-to-go. We reckon that the cost-to-go often largely depends on the "level of maneuver difficulty." In this light, we learn the maneuver costs of the complicated trailer kinematics by reinforcement learning and use the learned value function to obtain the heuristic value.

Thus, this work presents a motion planning method that utilizes i-AGT with a data-driven heuristic to plan for autonomous tractor-trailer systems. The main contributions are:

- A simple set of MPs is generated for i-AGT with reachability guarantees.
- A data-driven heuristic is proposed to increase the search efficiency of i-AGT.
- Extensive simulation is performed to show the effectiveness of the proposed system (video is publicly available here).

4.2 Related Works

Here we review works related to trailer planning. There are two main methodologies.

4.2.1 Optimization-based algorithms

Works in this category usually seek a numerical solution of an OCP [106, 127, 105], which requires a non-trivial initialization to converge, especially in cluttered environments [135, 175, 13, 101]. Sampling-based planners with simplified vehicle models [175, 101] have been used to provide an initial path for the OCP. This treatment still does not guarantee the

feasibility of the OCP because 1) the initial path may not be homotopic to the true solution of the OCP [157]; 2) the objective function in the sampling stage may not represent that in the optimization stage [14].

4.2.2 Search-based and state-lattice-based algorithms

Search-based planning is another popular method for tractor-trailer systems, which abstracts the configuration space as a graph with nodes and edges [34, 115]. Search operations are often done by A* [62, 71, 35]. State-lattice-based planners are deterministic planners which use a finite set of pre-computed MPs online to find a resolution-optimal solution [132, 114]. However, due to the discretized search space, the graph resolution will affect the optimality [14].

4.3 Preliminaries

4.3.1 Problem statement

This chapter considers the motion planning problem as finding a trajectory that connects the initial state to the goal state. Consider a system with the following dynamics

$$\dot{X} = f(X) + g(X, u), \tag{4.1}$$

where $X \in \mathcal{X} \subset \mathbb{R}^{n_x}$ is the state, $u \in \mathcal{U} \subset \mathbb{R}^m$ is the control input, f and g are smooth vector fields. A *configuration* of system (4.1) is a complete specification of the position of every points of that system. The *configuration space* $\mathcal{C} \subset \mathbb{R}^{n_c}$ is a compact set representing all possible configurations; and $\mathcal{C}_{free}$ denotes a collision-free configuration space. This work assumes $\mathcal{C} = \mathcal{X}$ and the motion planning problem as follows:

Problem 3. *Given an initial state $X_I \in \mathcal{C}_{free}$, a goal state $X_G \in \mathcal{C}_{free}$, and system* (4.1)*, find a feasible trajectory $\mathcal{P}_t$ which*

(I) starts at X_I and ends at X_G, while satisfying (4.1)*; and*

(II) lies in the collision-free configuration space $\mathcal{P}_t \subset \mathcal{C}_{free}$.

Although optimality is not directly considered in the problem formulation, as opposed to the formulation in Section 2.2.1, the search method still results in a solution with some level of optimality. The optimality is directly related to the definition of the arrival cost and the heuristic function used during the search.

4.3.2 Trailer modeling

Consider a front-wheel drive *standard trailer system* [141, 5] as shown in Figure 4.1, where $[x, y]^\top$ are the coordinates of the midpoint of the tractor's rear-wheel axis, θ_0 is the

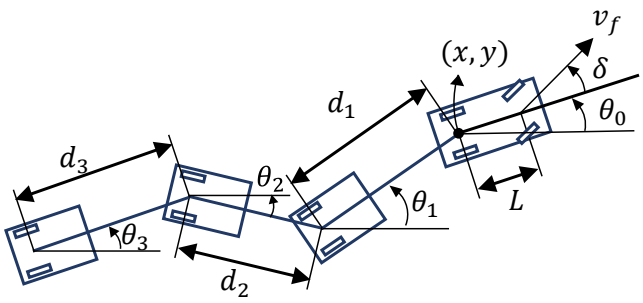


Figure 4.1: Kinematics of a front-drive tractor with 3 trailers. (All trailers are on-axle, and all angles representing the orientation of tractor and trailers ($\theta_i, i = 0, \ldots, 3$) are w.r.t. the x-axis.)

tractor orientation, $\theta_1, \theta_2, \theta_3$ are the orientations of trailers, v_f is the front-wheel velocity of the tractor, δ is the steering angle of the tractor, and L is the distance between (x, y) and the midpoint of the front wheel axis. The control inputs are v_f and δ. A mechanical constraint $|\delta| \leq \delta_{\max}$ limits the minimum turning radius R of a path. Provided that v_f and δ can be independently controlled, we introduce new control variables: $u = [v, s]^\top = [\cos(\delta)v_f, \frac{\tan(\delta)}{\tan(\delta_{\max})}]^\top$, where $\tan(\delta_{\max}) = \frac{L}{R}$, and $s \in [-1, 1]$ is the normalized steering angle. It is beneficial to represent the kinematic model in the coordinates $\xi = [x, y, \theta_0, \theta_1 - \theta_0, \theta_2 - \theta_1, \theta_3 - \theta_2]^\top$, where:

$$\begin{aligned}
\dot{x} &= \cos(\theta_0)v, \\
\dot{y} &= \sin(\theta_0)v, \\
\dot{\theta}_0 &= \frac{vs}{R}, \\
\dot{\xi}_4 &= -v\frac{d_1 s + \sin(\xi_4)R}{Rd_1}, \\
\dot{\xi}_5 &= -v\frac{d_1\cos(\xi_4)\sin(\xi_5) - d_2\sin(\xi_4)}{d_1 d_2}, \\
\dot{\xi}_6 &= -v\cos(\xi_4)\frac{d_2\cos(\xi_5)\sin(\xi_6) - d_3\sin(\xi_5)}{d_2 d_3}.
\end{aligned} \tag{4.2}$$

In ξ-coordinates, the constraints to prevent a jack-knife configuration are

$$|\xi_i| \leq \xi_{\max}, \quad 4 \leq i \leq 6, \tag{4.3}$$

where $\xi_{\max}$ must be less than $\frac{\pi}{2}$. The trailer system is subject to additional state and control constraints:

$$0 \leq \theta_0 < 2\pi, \ |v| \leq v_{\max}, \ |s| \leq 1. \tag{4.4}$$

Remark 2. *Although this work is demonstrated on a standard trailer system, the proposed method can be readily generalized to other trailer systems.*

4.3.3 i-AGT algorithm

i-AGT constructs a tree $\mathcal{T}$, with a root node $X_{init} \in \{X_I, X_G\}$ and a goal $X_{goal} \in \{X_I, X_G\} \backslash \{X_{init}\}$, which reaches a neighbor $\mathcal{B}_\epsilon(X_{goal}) \triangleq \{X \in \mathcal{X} | d(X, X_{goal}) \leq \epsilon\}$. Specifically, $d(\cdot, \cdot)$ is a distance function, i.e., a weighted 2-norm: $\|X_i - X_j\|_P = ((X_i - X_j)^\top P(X_i - X_j))^{1/2}, X_i, X_j \in \mathcal{C}$. Similar to A*, each node X is assigned an F-value calculated as follows:

$$F(X) = g(X_{init}, X) + h(X, X_{goal}), \tag{4.5}$$

where $g(X_{init}, X)$ represents the cost-to-come, or the arrival cost, from X_{init} to X, and $h(X, X_{goal})$ denotes the estimated cost-to-go, or the heuristic value, from X to X_{goal}. i-AGT maintains a priority queue Q, which contains nodes to be expanded. All nodes in Q are ordered according to their F-values.

The trade-off between planning time and maneuver resolution, i.e., the cardinality of the set of MPs $\mathcal{M}$, has to be made when a search algorithm is involved [14]. i-AGT pivots on a concept 'mode' which divides $\mathcal{M}$ into m unique subsets of motion primitives $M_i \subset \mathcal{M}, i = 1, \ldots, m$. During node expansion of the best candidate node X_{best}, a current mode M_c, which has the highest priority among untried modes, is determined. Then, X_{best} is expanded by applying primitives in M_c one by one. Applying each MP gives a new node X_k and the connecting trajectory $\mathcal{P}_k$ from X_{best} to X_k. If X_k is δ-distant away from $\mathcal{T}$ and $\mathcal{P}_k$ is collision free, then the algorithm

(I) updates priority $P^{M_c}_{X_{\text{best}}}$ according to $F(X_{\text{best}})$ and $F(X_k)$;

(II) copies the mode priorities of X_{best} to X_k;

(III) adds node X_k and edge $E(X_{\text{best}}, X_k)$ to $\mathcal{T}$;

(IV) inserts node X_k into Q.

Details of i-AGT can be found in [169].

4.4 The Off-lattice Motion Planning Algorithm

i-AGT entails three components: the MPs that govern node expansion, the mode priority that governs mode selection, and the heuristics that govern node selection. A naive approach is to directly apply the i-AGT used in [169] to the trailer planning setting, i.e., use the same heuristics and mode definition while defining the MPs as a tuple of constant velocity and constant steering angle over a certain period. In Case 7 (Figure 4.13(a)), the i-AGT constructs a tree with 2617 nodes using 7 sec. However, it undergoes heavy computation because the MPs result in exploring the 6-dim state space and land too many collision-free but kinematically undesirable nodes. With the state space being 6-dimensional, the number of elements in the state-lattice covering a compact set of the state space could be large, yielding an explosive number of motion primitives - known as the curse of dimensionality.

The following sections introduce the proposed methods to enable trailer planning using i-AGT, mainly the construction of the MPs set and obtaining informative heuristics for a trailer system.

4.4.1 Dimension reduction

To find a better set of MPs, we follow the well-established idea in [5, 114] to circumvent the curse of dimensionality by restricting X to meet the condition that the tractor and all trailers move in circles (trailers and tractor have the same yaw rate C), as shown in Figure 4.2(a). Note that given v, the yaw rate of the tractor and the headings of all trailers are uniquely determined by the steering action s. Particularly, for tractor $\dot{\theta}_0 = vs/R = C(s)$, whereas the headings of trailers can be uniquely determined from

$$\dot{\xi}_k = 0, \quad k = \{4, 5, 6\},$$

which admit the solutions

$$\begin{aligned} \xi_4(s) &= -\arcsin(\frac{sd_1}{R}), \\ \xi_5(s) &= -\arcsin(\frac{sd_2}{Rc_1}), \\ \xi_6(s) &= -\arcsin(\frac{sd_3}{Rc_1c_2}), \end{aligned} \tag{4.6}$$

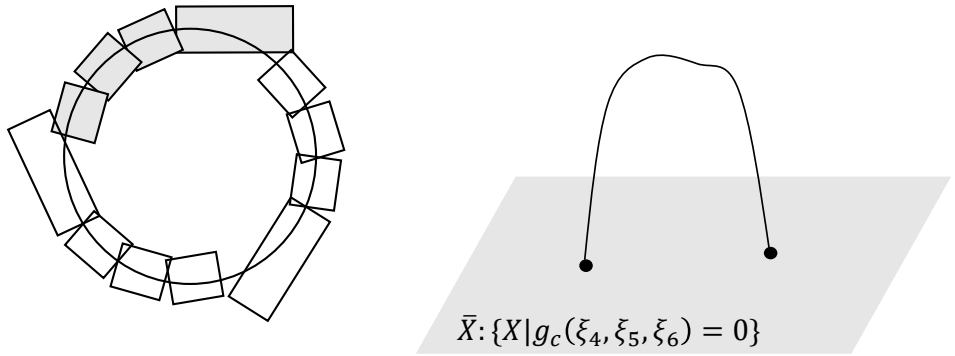
where $c_1 = \sqrt{1-(sd_1/R)^2}, c_2 = \sqrt{1-(sd_2/(Rc_1))^2}$. We use $g_c(\xi_4, \xi_5, \xi_6) = 0$ to simplify (4.6). As a result, planning is carried out over the 4-dim space $\bar{\mathcal{X}} \subset \mathbb{R}^4$ (also called the reduced-state space) with $\bar{X} = [x, y, \theta_0, s]^\top \in \bar{\mathcal{X}}$, i.e., the tree has 4-dim nodes. Note that the trailer system state still evolves in $\mathcal{X} \subset \mathbb{R}^6$ during the transition between nodes, as shown in Figure 4.2(b).

4.4.2 Motion primitives

An MP can be viewed as a function $mp(\cdot)$ that transforms a node $\bar{X}_i$ to a new node $\bar{X}_j$, i.e., $\bar{X}_j = mp(\bar{X}_i)$. To ensure all nodes in i-AGT remains in the 4-dim space $\bar{\mathcal{X}}$, one should design $\mathcal{M}$ so that $\bar{\mathcal{X}}$ is $\mathcal{M}$-invariant, while satisfying (4.2). That is $mp(\bar{X}) \in \bar{\mathcal{X}}, \forall \bar{X} \in \bar{\mathcal{X}}, mp \in \mathcal{M}$. Below illustrates how to obtain such $\mathcal{M}$ for on-lattice exploration and for off-lattice exploration, respectively, by solving a multitude of steering problems.

The steering problem

Steering refers to connecting two states with a kinematically or dynamically feasible trajectory. It can be posed as an open-loop OCP. We denote the initial state as X_0, the target state as X_f, an admissible control set as $\mathcal{U}$, an objective function as $c(X, u)$, the control input as u, and final time as t_f. If the final time is free, the steering problem is cast



(a) (b)

Figure 4.2: (a) An example of the circular equilibrium configuration. (b) An example of a motion primitive in the original state space $\mathcal{X}$, but starts and ends at the plane of the reduced-state space $\bar{\mathcal{X}}$.

into a fixed final time OCP via time-scale transformation $\gamma = \frac{t}{t_f}$. The time-scaled OCP has a final time 1:

where $\bar{t}_f$ is the upper bound of t_f. The OCPs (4.7) are formulated and solved by using CasADi [8] and IPOPT [167].

$$\begin{aligned} \min_{u,t_f} \quad & c(X,u), \\ s.t. \quad & \frac{\partial X}{\partial \gamma} = t_f f(X,u) \;\&\; (4.3) \;\&\; (4.4), \\ & X(0) = X_0,\ X(1) = X_f,\ u \in \mathcal{U},\ t_f \in [0, \bar{t}_f]. \end{aligned} \tag{4.7}$$

When solving (4.7) numerically in ξ-coordinates, we choose 50 time steps over $[0, 1]$, and the bounds

$$\begin{aligned} & |x| \le 10,\ |y| \le 10,\ |\theta_0| \le 2\pi, \\ & |\xi_i| \le \frac{\pi}{2}, \quad 4 \le i \le 6, \\ & |v| \le v_{\max} = 5R,\ |s| \le 1. \end{aligned} \tag{4.8}$$

Figure 4.3(a) exemplifies a parallel parking MP which goes from $X_0 = [0, 0, 0, 0, 0, 0]^\top$ to $X_f = [8, 1, 0, 0, 0, 0]^\top$. The key of constructing $\mathcal{M}$ is to determine *what* are the MPs we need and design the state-lattice accordingly, so that we can obtain the underlying MPs by solving the steering OCPs from one state-lattice node to the other.

State-lattice and simplification

The main concern when designing the state-lattice is to ensure discrepancy and dispersion of the underlying MPs. Here we employ uniform discretization of a compact set: $\mathcal{D}_0 \triangleq$

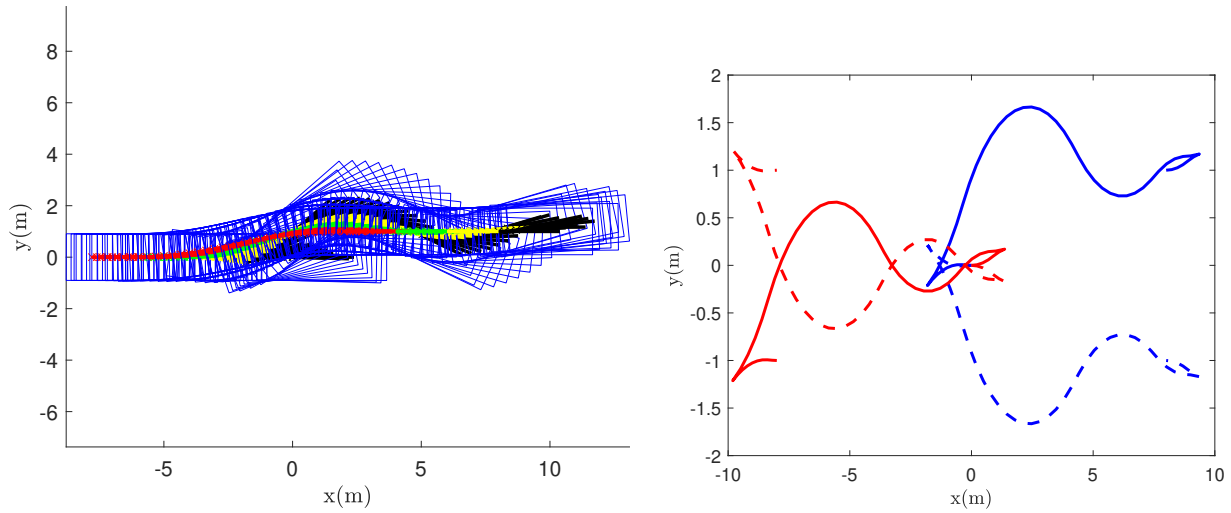


Figure 4.3: Motion primitive generation: (a) an example of parallel parking maneuver, (b) motion primitives from reversibility and symmetry. The lines indicate the trajectories of the tractor.

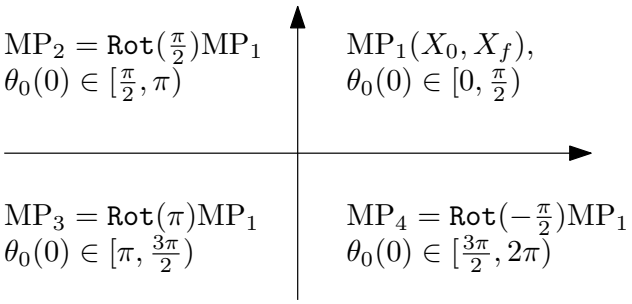


Figure 4.4: Simplification of motion primitive by rotational symmetry.

$[-L_x, L_x] \times [-L_y, L_y] \times [-\pi, \pi] \times [-1, 1] \subset \bar{\mathcal{X}}$. Denote the lattice set $\mathcal{S}$. Steering problems are defined with $(\bar{X}_0, \bar{X}_f) \in \mathcal{S} \times \mathcal{S}$.

The number of elements in $\mathcal{S}$ could be huge. For example, with $L_x = L_y = 2m$ and the size of the state-lattice being $\Delta x = \Delta y = 1m, \Delta\theta = \pi/8$, and $\Delta s = 0.5$, we obtain a total of 2125 state-lattice nodes. Solving steering problems by trying all possible combinations of $(\bar{X}_0, \bar{X}_f)$ will lead to solving millions of steering problems, which could be computationally prohibitive. We simplify the MP generation process by exploiting the properties of the steering problems: invariance w.r.t. (x, y), symmetry against x-axis, $\pi/2$ rotation, and reversibility over time. In the following, we treat the x, y positions, the heading θ, and steering s as functions of time duration within the MP, i.e., $[x(t), y(t), \theta_0(t), s(t)]^\top,\ t \in [0, 1]$.

- **Simplify by invariance w.r.t.** (x, y)**:** Consider the steering problem from $\bar{X}_0 = [x(0), y(0), \theta_0(0), s(0)]^\top$ to $\bar{X}_1 = [x(1), y(1), \theta_0((1), s(1)]^\top$ and assume the solution trajectory is a function of time duration, i.e., $\bar{X}(t),\ t \in [0, 1]$. One can construct the steering solution trajectory from $\bar{X}_i = [x(0) + l_x, y(0) + l_y, \theta_0(0), s(0)]^\top$ to $\bar{X}_j = [x(t)+l_x, y(t)+l_y, \theta_0((1), s(1)]^\top$ as $\bar{X}(t)+[l_x, l_y, 0, 0]^\top$. Hence, we only need to consider $\bar{X}_0$ with $x(0) = y(0) = 0$.

- **Simplify by symmetry against** x**-axis:** If there exists a steering solution between $\bar{X}_0 = [0, 0, \theta_0(0), s(0)]^\top$ and $\bar{X}_1 = [x(1), y(1), \theta_0(1), s(1)]^\top$, e.g., the blue solid in Figure 4.3(b), so does the solution between $\bar{X}_i = [0, 0, -\theta_0(0), -s(0)]^\top$ and $\bar{X}_j = [x(1), -y(1), -\theta_0(1), -s(1)]^\top$, the dash blue line in Figure 4.3(b). Hence one only needs to solve the steering problems $\theta_0 \in [0, \pi)$, i.e., $\bar{X}_0 \in \{0\} \times \{0\} \times [0, \pi) \times [-1, 1]$.

- **Simplify by** $\pi/2$ **rotation:** Next we exploit the symmetry w.r.t. to rotations with angles $k(\pi/2), k \in \mathbb{Z}$. That is if there exists a solution from $\bar{X}_0$ to $\bar{X}_1$, so does the solution from $\bar{X}_i = [0, 0, \theta_0(0) + \pi/2, s(0)]^\top$ and $\bar{X}_j = [\texttt{Rot}(\pi/2)(x(1), y(1)), \theta_0((1) + \pi/2, s(1)]^\top$, where $\texttt{Rot}(\pi/2) \in \mathcal{SO}(2)$ with the angle $\pi/2$. That is to say: we only need to solve steering problems with $\bar{X}_0 \in \mathcal{D}_2 \triangleq \{0\} \times \{0\} \times [0, \pi/2) \times [-1, 1]$ to obtain MP_1, and the MP with $\theta_0 \in [\frac{\pi}{2}, 2\pi)$ can be constructed from MP_1, see Figure 4.4.

- **Simplify by reversibility over time:** Reversibility over time [114] means that a solution from $\bar{X}_0$ to $\bar{X}_1$ (the solid blue in Figure 4.3(b)) implies the solution from $\bar{X}_1$ to $\bar{X}_0$ (the solid red line in Figure 4.3(b)), by reversing time. Combining reversibility with the assumption that forward and backward MPs lead to $\bar{X}_f$ with $x_f \geq 0$ and $x_f < 0$, respectively, one only needs solving steering problems with $x_f \in [0, L_x]$ for forward MPs, and then constructing backward MPs by reversing forward MPs.

Remark 3. *The idea "forward" and "backward" is relative to the trailer's heading that we try to keep roughly in the same direction, if possible. Therefore, forward MPs and backward MPs are associated with $\bar{X}_f$ located in the right half-plane (RHP) and the left half-plane (LHP), respectively. Note that driving "forward" to the LHP is essentially making a U-turn (Figure 4.5(left)), which is not preferred because it is hard to perform in environments full of obstacles.*

Solving steering problems from $\bar{X}_0 \in \mathcal{D}_2 \rightarrow \bar{X}_f \in \mathcal{D}_0$ gives a group of MPs: $\mathcal{M}_{on}$. Similar to [132, 37, 114], one can categorize all MPs according to $\bar{X}_0$, i.e., each class of MPs is associated with a unique 2-dim pose $q = [\theta_0, s]$. During node expansion, one first performs mod $(\theta_0, \frac{\pi}{2})$ to map $\bar{X}$ into $\mathcal{D}_2$, then retrieves the corresponding MPs as $\mathcal{M}_{on,q}$, and finally applies $\mathcal{M}_{on,q}$ at $\bar{X}$. In such a way, all nodes explored during planning remains on lattice.

Off-lattice motion primitives

Applying MPs in an on-lattice manner is beneficial to maintaining the tree sparsity and high computational efficiency. However, restricting all nodes to the lattice introduces

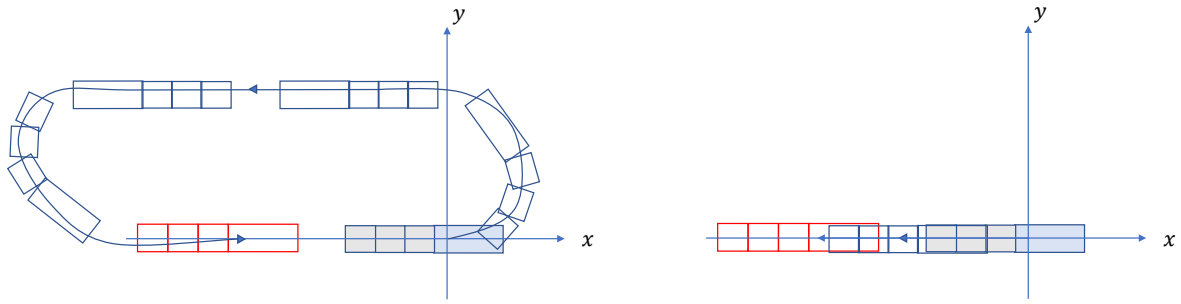


Figure 4.5: A trailer driving "forward" (left) and "backward" (right) to the goal configuration (marked in red).

significant limitations that bring various detrimental impacts: compromised feasibility in tight environments and often unsatisfactory planning quality, such as unnecessarily long path length. One can remedy these shortcomings by adopting many MPs for better resolution completeness and potentially better planning quality. However, ensuring the quality and connectivity of the MPs requires an even more complicated process.

To overcome the curse of dimensionality, the planner should plan over a 4-dim space, not over the 4-dim lattice. MPs should be classified and applied so that off-lattice nodes can be generated. This is readily achievable by grouping MPs according to steering s: given $\bar{X}$, one fetches and applies the MPs beginning with s at $\bar{X}$. In this case, each class of MPs should associate with a unique 1-dim pose $q = s$. We construct $\mathcal{M}_{off}$ from $\mathcal{M}_{on}$ by merging all MPs beginning with the same s to one class, and pruning MPs that ends close. In this work, we pick $L_x = L_y = 8m$, $\Delta x = \Delta y = 1m, \Delta\theta = \pi/12$, and $\Delta s = 0.25$, and follow the above process to obtain $\mathcal{M}_{on}$ containing 2904 MPs and $\mathcal{M}_{off}$ containing 594 MPs. Figure 4.6 illustrates the 9 poses used in $\mathcal{M}_{off}$. In the following section we will show the analysis of these, particularly to show that $\mathcal{M}_{off}$ contains enough MPs.

Remark 4. *Generating $\mathcal{M}_{off}$ from $\mathcal{M}_{on}$ does not alleviate the computation complexity of MP generation. However, the off-lattice application allows us to apply MPs generated with $\theta_0 = 0$ to other configurations with the same pose. Therefore, one can directly solve much less steering problems by further restricting $\bar{X}_0$ to the set: $\{0\} \times \{0\} \times \{0\} \times [-1, 1]$.*

Remark 5. *To keep the element in $\mathcal{M}_{off}$ simple enough to be followed by the underlying control system, we remove the MPs that contain cusps.*

Analysis

Both $\mathcal{M}_{on}$ and $\mathcal{M}_{off}$ have to meet specific criteria for reasonable planning performance. We analyze them in the graph framework. Given an MP set $\mathcal{M}$, one can construct a directed

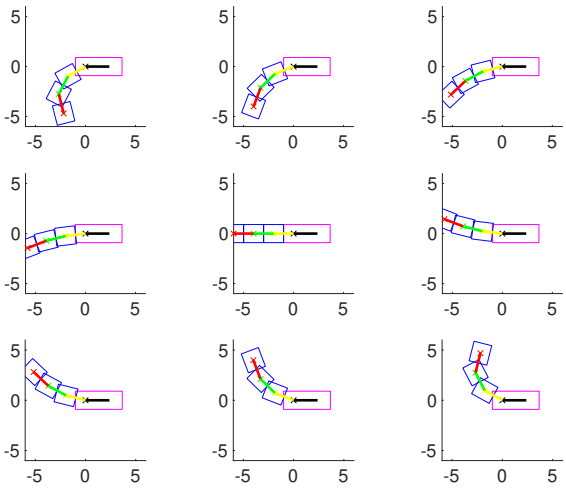
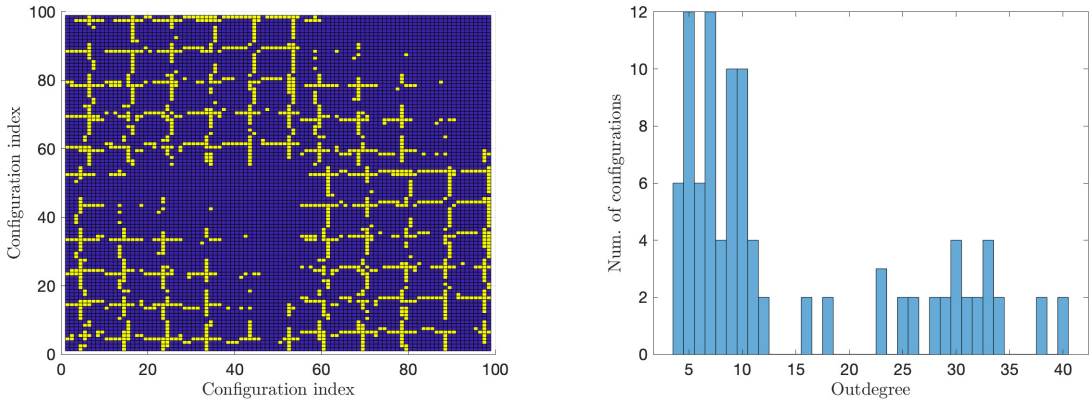


Figure 4.6: 9 poses of $\mathcal{M}_{off}$.

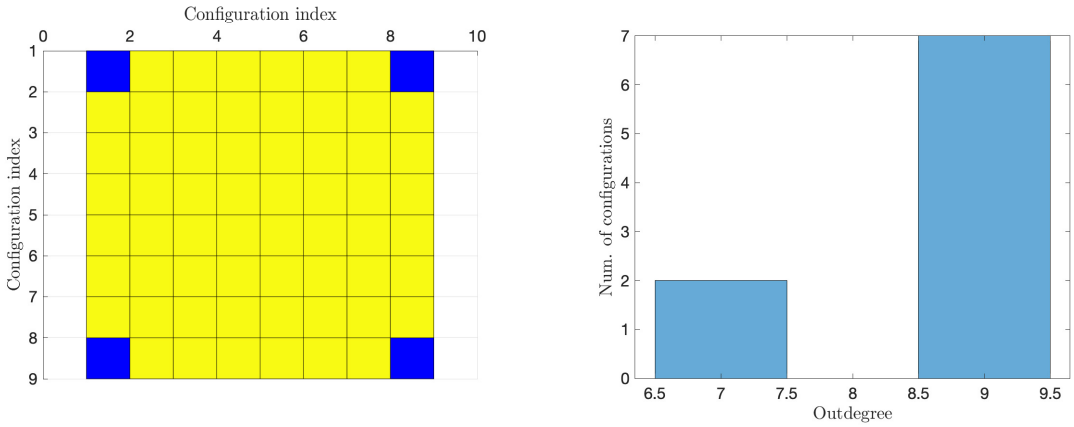


(a) (b)

Figure 4.7: Reachability analysis of $\mathcal{M}_{on}$: (a) adjacency matrix of $\mathcal{M}_{on}$ (b) outdegree of $\mathcal{M}_{on}$.

graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ where the node-set $\mathcal{V}$ represents all possible poses q of $\bar{X}_0$. $\mathcal{E}$ is a collection of directed edges from q_i to q_j. Each represents the existence of an MP, allowing the uni-directional transition from q_i to q_j.

The graph $\mathcal{G}$ constructed from $\mathcal{M}$ shall satisfy the following properties: 1) the graph is connected, meaning the trailer can transform from one pose to another pose in finite steps; 2) all nodes have similar in-degree and out-degree, meaning the transformation from pose to pose can potentially be done easily; and 3) all q have a similar amount of backward and forward MPs, meaning the trailer can potentially move easily in 2D space. We construct $\mathcal{G}_{on}, \mathcal{G}_{off}$ from $\mathcal{M}_{on}, \mathcal{M}_{off}$, respectively. Figure 4.7 shows the analysis results, when θ_0 and s are discretized over $[-\frac{\pi}{2}, \frac{\pi}{2}]$ and $[-1, 1]$ with resolution $\frac{\pi}{12}$ and $\frac{1}{4}$, respectively. As a result,



(a) (b)

Figure 4.8: Reachability analysis of $\mathcal{M}_{off}$: (a) adjacency matrix of $\mathcal{M}_{off}$, and (b) outdegree of $\mathcal{M}_{off}$.

$\mathcal{M}_{on}$ ends up with 99 2-dim poses q, and $\mathcal{M}_{off}$ contains 9 1-dim poses. Figure 4.7(a) and 4.8(a) plot the adjacency matrices of $\mathcal{M}_{on}$ and $\mathcal{M}_{off}$, respectively where yellow in (i,j) means the transition from q_i to q_j exists. We can see that the yellow area in Figure 4.8(a) covers almost the full matrix, meaning that without considering obstacles, the trailer can transform from one pose to another pose by applying at most two MPs. One easily confirms the connectivity of $\mathcal{M}_{on}, \mathcal{M}_{off}$ by checking the indegree and outdegree of all nodes. Figure 4.8(b) and 4.7(b) show the outdegree of $\mathcal{M}_{on}$ and $\mathcal{M}_{off}$, respectively, where $\mathcal{M}_{on}$ exhibits much more non-uniformity than $\mathcal{M}_{off}$. It is noteworthy that $\mathcal{M}_{off}$ is better in the sense that it is fully connected with much fewer (5x) MPs.

4.4.3 Mode and estimated cost-to-go

The planning efficiency of i-AGT is highly dependent on the mode definition/selection to find the proper subset of MPs and the estimation accuracy of the cost-to-go to find the right node to apply these MPs. In this work, we use the idea in [169] and classify the MPs into two modes: *forward mode* and *backward mode*. A child node generated following 4.3.3 is likely to explore the forward MPs if the parent node was also exploring forward so that the planner won't easily "undo" its previous exploration. A more involved mode definition/selection can be developed, which may be future work. On the other hand, constructing a heuristic function to approximate the cost-to-go for trailers is much more challenging and time-consuming than for cars because the steering problem does not admit an analytical solution.

Estimating the cost-to-go function is based on the realization that $h(X, X_{goal})$ is exactly the same as $h(\mathbf{T}_{goal}(X), \mathbf{T}_{goal}(X_{goal}))$, where $\mathbf{T}_{goal}$ is an SE transformation such that $\mathbf{T}_{goal}(X_{goal})$ has the tractor configuration being $[x, y, \theta_0]^\top = [0, 0, 0]^\top$ and the trailer configuration $[\xi_4, \xi_5, \xi_6]^\top$ remains unchanged and $\mathbf{T}_{goal}(X)$ has the tractor configuration being the

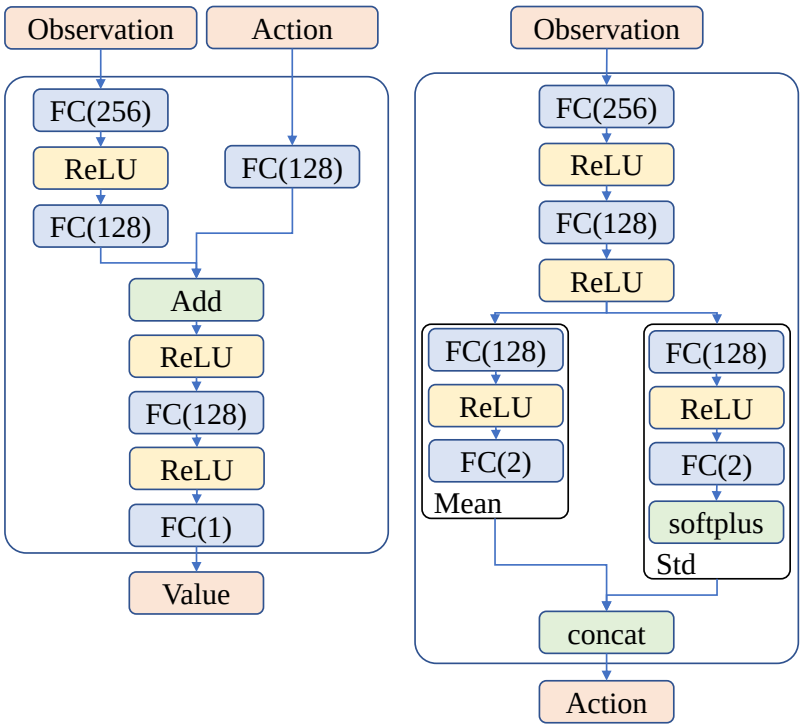


Figure 4.9: The critic net (left) and actor net (right).

value relative to X_{goal} and the trailer configuration remains unchanged. Particularly, the SE transformation is uniquely defined as follows

$$\mathbf{T}_{goal}(X) = \begin{pmatrix} x\cos\theta_{0,goal} + y\sin\theta_{0,goal} - x_{goal} \\ -x\sin\theta_{0,goal} + y\cos\theta_{0,goal} - y_{goal} \\ \theta_0 - \theta_{0,goal} \\ \xi_{4,goal} \\ \xi_{5,goal} \\ \xi_{6,goal} \end{pmatrix}.$$

We propose to learn $h(X, X_{goal})$ by learning the navigation policy and value function in free space with the soft actor-critic (SAC) algorithm [60], which is a model-free, online, off-policy, actor-critic reinforcement learning method. Previous works have considered learning policies for similar systems with deep deterministic policy gradient (DDPG) [11] or deep Q-Network (DQN) [64]. Since our goal is to obtain a value function to approximate the cost-to-go, we favor the exploration property and the continuous action space and choose SAC over DDPG or DQN.

To train the SAC agent, we initialize the trailer state randomly in $\mathcal{X}$. The network structure is shown in Figure 4.9, where observation refers to the state X and action refers to the input u. "FC(n)" refers to a fully connected layer with output size n. "Add" refers

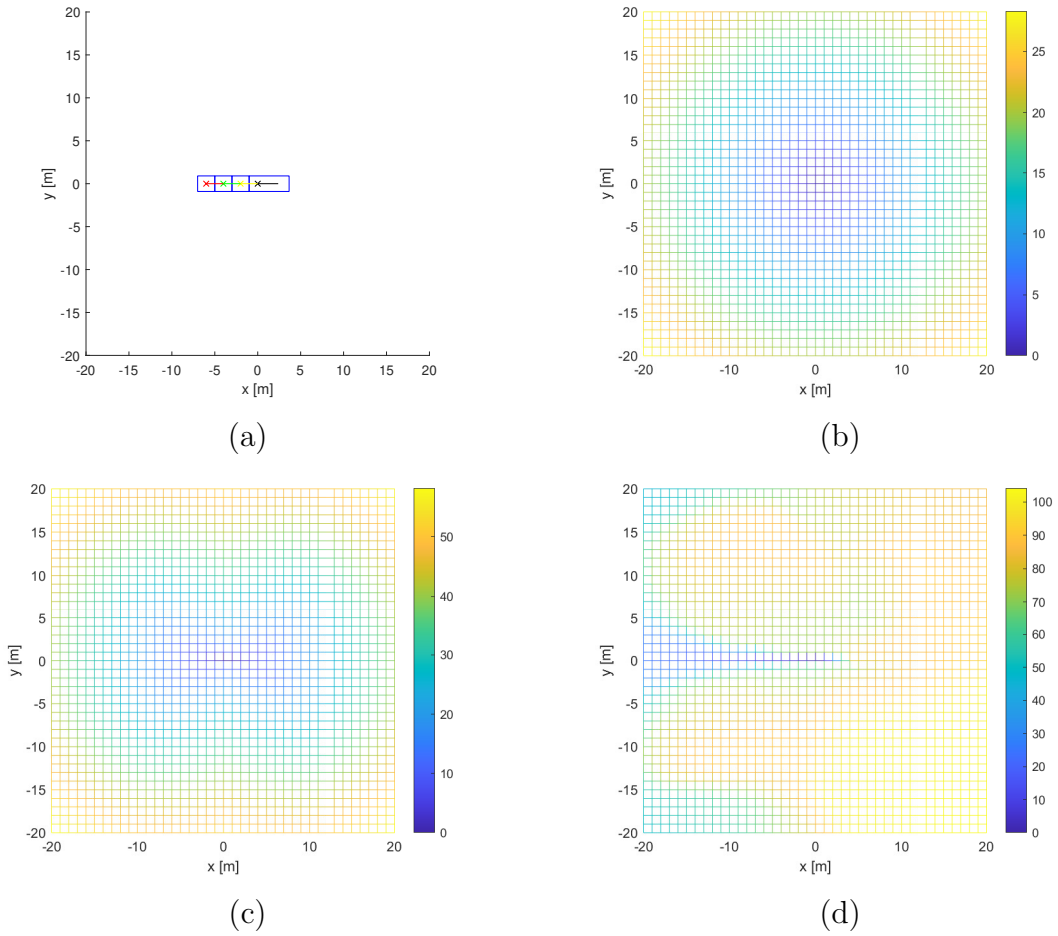


Figure 4.10: Example of the heuristic value for goal configuration at $[0, 0, 0, 0, 0, 0]^\top$ based on: (b) Euclidean distance, (c) RS path, and (d) SAC value function.

to an addition layer, which adds inputs from multiple neural network layers element-wise. The "softplus" layer applies the softplus activation function $q = \log(1 + e^p)$, $p, q \in \mathbb{R}$, which ensures the positiveness of outputs. The "concat" refers to the concatenation layer, which takes inputs and concatenates them along a specified dimension. Since the zero heading and steering configuration is often a preferred trailer parking pose, a reward is given when the trailer is closed to $X_{train,goal} = [0, 0, 0, 0, 0, 0]^\top$. Each episode terminates when the maximum steps per episode $MaxSteps$ is reached when the trailer goes out of a $35[m]$x$35[m]$ window centered at the goal, or when the trailer runs into a jackknife configuration. The design of the reward function and the training process is crucial to the performance of the trained policy. The two objectives of the reward function are:

- encouraging the trailer to go to $X_{train,goal}$ as quickly as possible;
- not discouraging exploration of the complicated trailer kinematics and the control

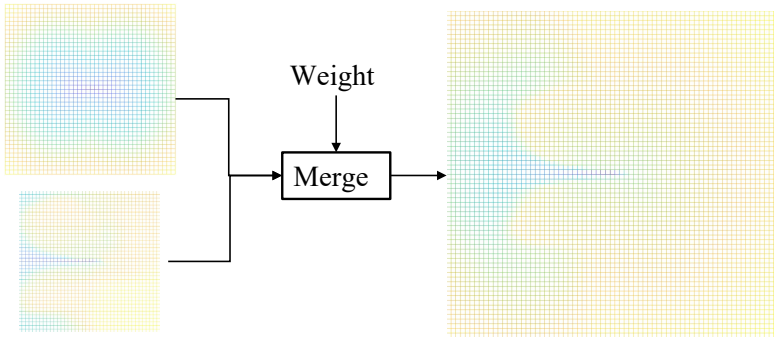


Figure 4.11: Combining the local heuristic function and the global heuristic function.

policy.

We design a sparse reward function. While $MaxSteps$ is not reached, the reward at time k with observation X_k is:

$$r(X_k) = \begin{cases} 1 & \text{if } \|X_k - X_{train,goal}\|_{W_1} \leq \epsilon \\ 0 & \text{otherwise} \end{cases}, \tag{4.9}$$

where $W_1 = \text{diag}([4.83, 4.25, 0.33, 0.15, 0.07, 0.03])$ is a weight matrix and ϵ determines the size of the "goal region." Since the reward is sparse, it is hard to receive a reward at the beginning if ϵ is too small. Therefore, we start the training with $\epsilon = 1$ until the average reward converges, then shrink ϵ to half of its previous value, and repeat the process until $\epsilon = 0.25$.

To obtain the approximated cost-to-go from the learned critic value function, we use a zero vector for the action input and use the resulting critic value function as the heuristic value. We compare the proposed heuristic with heuristic functions based on Euclidean distance and Reeds-Shepp (RS) path [139]. Figure 4.10(b), (c), and (d) show the heuristic value to reach $X_{train,goal} = [0, 0, 0, 0, 0, 0]^\top$ starting from $X = [x(0), y(0), 0, 0, 0, 0]^\top$ as an example. The color indicates the heuristic value, where the locations with lower cost-to-go are colored in blue and higher cost-to-go are colored towards yellow. The heuristic value provided by Euclidean distance is shown in Figure 4.10(b), where the value change is the same in all directions from the origin. This indicates that this heuristic cannot reflect the different levels of difficulty in maneuvering tasks when the trailer heading and steering angles are different. The heuristic value provided by RS path length is shown in Figure 4.10(c), which has smaller values along the x-axis and larger values with locations close to y-axis. This reflects the difficulty in steering for a car model and similarly for the trailer. However, this heuristic cannot represent the steering angle, which largely affects the maneuvering difficulty. The heuristic based on the value function learned by SAC is shown in Figure 4.10(d), which has lower values along the x-axis because the trailer only needs to drive straight to the

goal. It also captures the fact that the trailer may need a larger space for maneuvers even though the Euclidean distance to the goal and the RS path length are small. This is reflected, for example, by the fact that $(x, y) = (-20, -5)$ has a smaller value than $(0, -5)$ in Figure 4.10(d), which is not captured by the other two heuristics. We conclude that the learned value function can reflect the true cost-to-goal more accurately. Note that the learned heuristic also has a higher value when the trailer needs to go backward. Although there is no preference between forward and backward motions during planning, in reality, forward motions are preferred because they are easier to execute.

Since the data-driven value function generally only works with states visited during the training process, it is only considered a local heuristic function. To get the global heuristic function, we combine the local and global heuristic functions. As shown in Figure 4.11, the final heuristic function is a weighted sum of the local heuristic function and global heuristic function given the weight.

4.5 Applications

Simulations benchmark the i-AGT with the proposed heuristic (i-AGT-NN) against the baseline i-AGTs that use RS as a heuristic, one with on-lattice (i-AGT-RS-on) MPs and the other with off-lattice (i-AGT-RS-off) MPs. The simulation environments mimic tracker-trailers moving materials in a large factory area where it needs to navigate through narrow aisles and park in narrow spaces. This section presents several of the simulation results as examples. The tractor-trailer parameters used $L = 2.396[m]$ and $d_1 = d_2 = d_3 = 2[m]$. The MPs are processed following the proposed method, which leads us to an $\mathcal{M}_{on}$ containing 2904 MPs and an $\mathcal{M}_{off}$ containing 594 MPs. Simulation is conducted on a 10-core Intel i9 3.7GHz desktop with Matlab R2021a. Figure 4.12, 4.13 shows the simulation results with i-AGT-NN in 11 cases, where the initial position of the trailer is colored in green, and the goal position is in red. The blue line indicates the trajectory of the tractor. Table 4.1 shows the detailed planner performance.

We first compare i-AGT-NN and i-AGT-RS-off to verify the effectiveness of the proposed heuristic. Figure 4.12(a)-(e) show the cases for trailer parking where several narrow parking spots are in the bottom right, and an open space is on the top right allowing for maneuvering. Cases 1 and 2 require the trailer to perform a right turn parking and parallel parking, respectively. In both cases, i-AGT-NN outperforms i-AGT-RS-off in planning time and requires less node exploration with similar resulting path lengths. Case 3 moves the trailer from one parking spot to another. To change the heading angle by 180 degrees, the trailer must utilize the open space, which makes the planning problem very challenging because moving the trailer to the upper boundary for successful planning is against the direction suggested by the heuristic of both heuristics. It takes both i-AGT-NN and i-AGT-RS-off a longer time to solve the problem than in other test cases. However, i-AGT-NN requires 50% less time than i-AGT-RS-off. Cases 4 and 5 show the importance of having a heuristic that accurately captures the system kinematics. Both cases require similar navigation skills as

Table 4.1: Comparison of i-AGT performances with heuristic based on RS and SAC value functions.

Case No.	i-AGT-RS-on			i-AGT-RS-off			i-AGT-NN		
	Planning Time [s]	# of Nodes Explored	Path length [m]	Planning Time [s]	# of Nodes Explored	Path length [m]	Planning Time [s]	# of Nodes Explored	Path length [m]
1	N/A*	N/A	N/A	14.44	2094 (18475)	35.67	2.02	319 (972)	38.65
2	0.50	67 (410)	37.68	4.08	585 (6666)	29.37	0.63	79 (317)	33.03
3	N/A	N/A	N/A	274.22	17381 (514137)	71.62	130.42	9400 (265622)	78.00
4	2.44	274 (5714)	36.27	103.88	7625 (198423)	58.33	0.53	68 (335)	28.79
5	2.98	179 (4812)	90.27	324.78	19551 (623083)	68.58	2.028	300 (912)	47.32
6	2.80	166 (5722)	88.15	5.49	383 (8598)	40.93	8.31	581 (13588)	69.71
7	1.54	98 (2536)	57.19	5.57	413 (5770)	45.19	2.46	182 (2239)	45.19
8	N/A	N/A	N/A	49.67	2285 (93292)	71.03	14.71	1043 (19528)	75.00
9	N/A	N/A	N/A	N/A	N/A	N/A	66.52	2859 (134787)	79.05
10	2.89	2094 (4812)	90.27	2.49	172 (2531)	52.09	4.54	325 (4739)	52.09
11	N/A	N/A	N/A	6.78	466 (8285)	69.52	1.59	124 (1883)	61.87

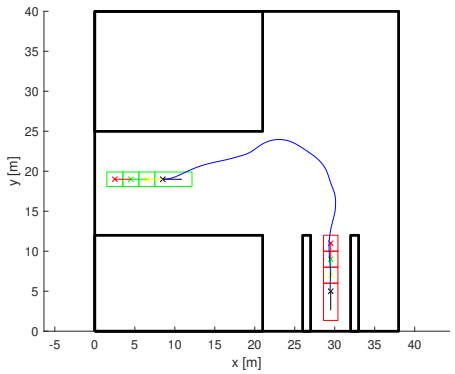
* Solver failed or time out. The maximum computation time is 350 seconds.

cases 2 and 1, respectively. While i-AGT-NN performs similarly to cases 2 and 1, i-AGT-RS-off suffers a longer planning time and path length because the heuristic cannot capture the steering error. In previous cases 1 and 2, the narrow space around the goal forced the steering to be closed to zero. However, when the goal is in relatively free space, the results of cases 4 and 5 show that it is important to have an accurate heuristic to guide the search efficiently. Figure 4.13(a)-(f) show the cases of a trailer navigating around narrow aisles. All planners perform similarly in cases 6, 7, 10, and 11, while the planning time is largely decreased by i-AGT-NN in cases 8 and 9. The path length from both planners is also similar in cases 7-11.

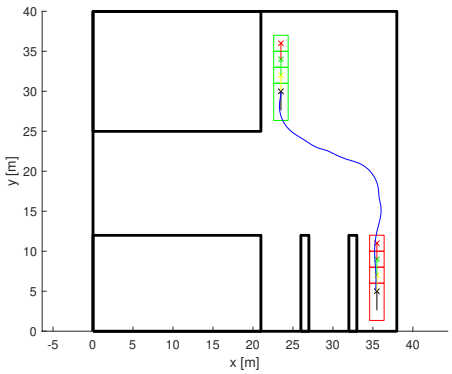
i-AGT-RS-on has a short planning time in the successful cases, but it fails to find a solution in nearly half of the cases due to insufficient resolution and results in long paths in cases 5,7,10. Using the NN heuristic may help reduce path length but will not improve the success rate. Note that $\mathcal{M}_{off}$ is derived from $\mathcal{M}_{on}$, and these results show that allowing off-lattice exploration indeed requires fewer MPs to reach the same level of (or even improve) planning success rate. In summary, i-AGT-NN outperforms i-AGT-RS-off in terms of average planning time while keeping the path length at the same level, and it outperforms i-AGT-RS-on in terms of success rate and path length.

4.6 Chapter Summary

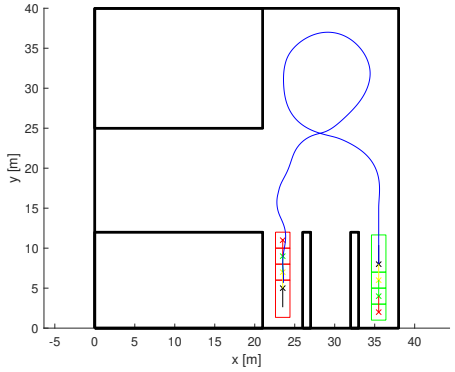
This chapter presented a motion planning strategy that utilized an improved A-Search Guided Tree to enable autonomous parking of a standard 3-trailer system with a car-like tractor. While exploiting the well-established lattice idea to circumvent the curse of dimensionality, we proposed to perform planning over a 4-dim space instead of planning over a 4-dim lattice by allowing the planner to explore outside of the lattice. We constructed motion primitives dependent only on the steering angle, which drastically lowers the complexity in the generation and selection of motion primitives, leads to a better success rate, and typically results in improved paths. To further increase search efficiency, we described a data-driven heuristic modeling the maneuver cost of the trailer to capture the cost-to-go by training a neural network through reinforcement learning. Simulations demonstrated the effectiveness of the proposed method in terms of success rate, planning speed, and path length.



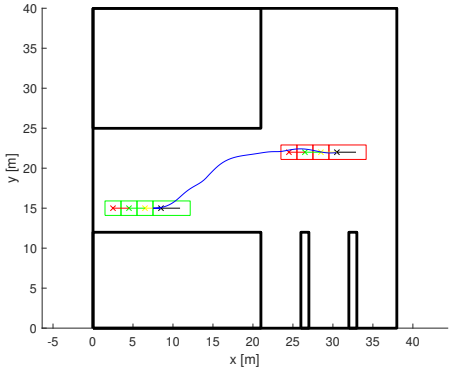
(a) Case 1.



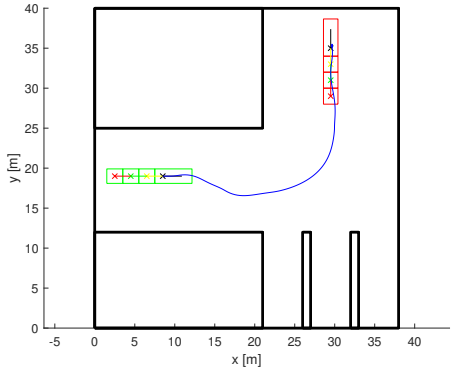
(b) Case 2.



(c) Case 3.

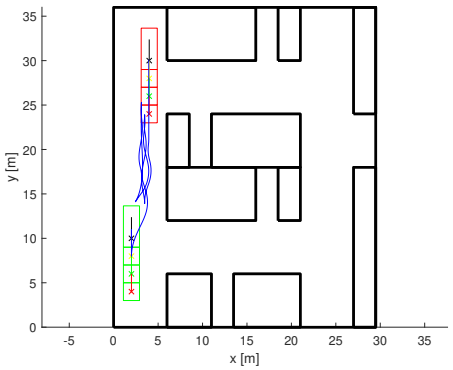


(d) Case 4.

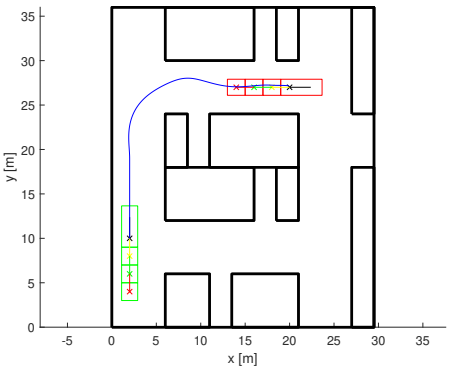


(e) Case 5.

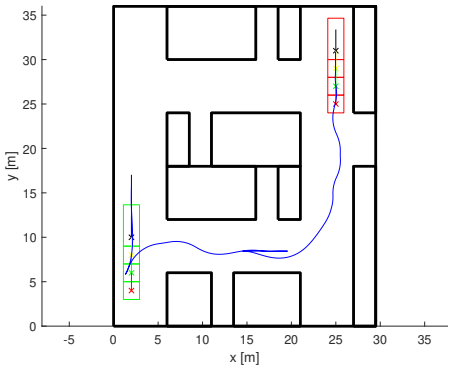
Figure 4.12: Simulation results of case 1-5.



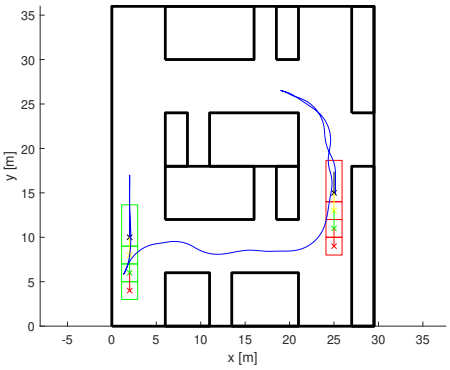
(a) Case 6.



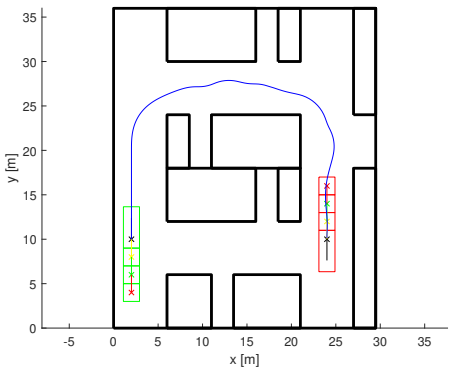
(b) Case 7.



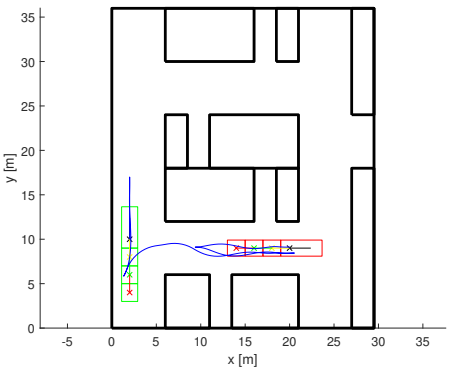
(c) Case 8.



(d) Case 9.



(e) Case 10.



(f) Case 11.

Figure 4.13: Simulation results of case 6-11.

Chapter 5

Motion Planning in Dynamic Environments

5.1 Introduction

To prepare the robots for industrial application in future factories, where human workers and robots work collaboratively, we move from planning in static environments to dynamic environments in this chapter. In modern factories, robots are already playing increasingly essential roles [131]. While most industrial robots are either fixed base robotic arms or mobile platforms [70], future factories can employ robots that have agile manipulation and mobility. Mobile manipulators (Figure 3.9) are typically composed of a mobile platform and one or more manipulators. The mobility from the platform and agility of the manipulators enlarge reachable space and capability to assist human workers on the factory floor [69]. It is crucial to have real-time, safe, and reactive motion planning for mobile manipulators in dynamic environments to utilize these advantages of a mobile manipulator. For example, an autonomous mobile manipulator in industrial human-robot interaction (HRI) system [69] needs to detect changes in the environment and re-plan in real-time to bypass multiple human workers and a set of obstacles to approach its target efficiently and safely [32, 130]. These planning problems are different from those mentioned in Chapter 3 because the planner needs to be able to react to environmental changes. Thus, the re-plan rate has to be higher. In industrial settings, the surrounding agents, such as human workers or other robots, normally will not move faster than 3 m/s; therefore, a robot system with a re-plan rate 5 to 10 Hz will be sufficient.

Mobile manipulators have high degrees of freedom and kinematic redundancy, making it hard to plan motions for the platform and the manipulator simultaneously, i.e., coordinated motion planning. In [142], an extensive review of motion planning methods for mobile robots is given. Motion planning algorithms can be classified to three categories: graph-search based algorithm [39], sampling-based algorithm [93, 51], and optimization-based algorithm [4, 54]. In [39], ARA* search is used to solve a planning problem for a mobile manipulator in cluttered

spaces. In [51], a sampling-based method, adaptive simulated annealing combined with torque minimization, is used to solve the motion planning problem and guarantees global optimality. In [77], a stochastic optimization method is used to solve a motion planning problem for mobile manipulators in static cluttered environments. In [161], the authors proposed a sampling-based method that first samples for the mobile base, then sample for the arm. Although these motion planning methods show promising results in static testing environments, they cannot be directly used in time-varying environments.

The time-varying environment requires the robot to re-plan and adapt its motion in reacting to the changes in the surroundings. This usually is difficult for graph-search-based algorithms and sampling-based algorithms due to the high dimensionality of the mobile manipulator that requires considerable computation time with these methods. Thus, some prior knowledge of motion primitives and limitations is often needed [38, 20] to shorten the computation time. However, it is challenging to ensure that the prior knowledge is always suitable or sufficient for the time-varying environment. In addition, these "planning-by-construction" planners often require an optimizer to convert the path plan to a motion plan. Thus, additional computation time is needed. Therefore, these methods may not be the best choice to solve a motion planning problem for a mobile manipulator in a dynamic time-varying environment. On the other hand, optimization methods can be solved within a relatively short period if appropriately formulated. Therefore, they may have more potential in solving these time-varying motion planning problems.

As mentioned previously, optimization methods need to be properly formulated to have a shorter computation time. Motion planning problems are often non-convex due to the nonlinear robot model and the obstacles in the environment. Solving these non-convex optimization problems is challenging. The feasible set that satisfies the constraints of the planning problem needs to be constructed carefully so that the planning result will not be over-conservative. Previous works proposed optimization-based planners but have restrictions on the range of scenarios that planners can handle. In [54], the authors propose a method using constrained sequential linear quadratic optimal control in a receding horizon control framework and claim that it allows a planning rate up to 100Hz. However, the motion planning controller can only deal with convex optimization problems and relies on reference trajectory generated beforehand. In [177], covariant hamiltonian optimization is presented with promising results avoiding dynamic obstacles, but it requires offline pre-computation of the distance field. In [4], a motion planning method for collaborative omnidirectional multi-robot manipulation is presented. While the update rate is shown to be at the order of 10Hz and can deal with non-convex state constraints, the constraints are overly simplified, making the motion plan conservative. Also, the time horizon is assumed to be short. Similar limitations are also in [9, 145], where short time horizon (horizon= 6) and conservative problem formulation appear, respectively.

The work in this chapter is motivated to achieve two objectives: make the computation time for planning small to achieve a higher update rate of planning and broaden the range of scenarios that the method can handle. The convex feasible set (CFS) algorithm [111] has been proposed to solve non-convex motion planning problems and obtain safe open-loop

trajectories in cluttered scenarios. In [113], a parallel planning-and-control architecture is introduced to solve motion planning problems for manipulators in dynamic environments while assuring safety. Inspired by these methods, this work presents an effective control strategy for mobile manipulators.

The proposed method, hierarchical receding horizon control (HRHC), solves the motion planning problem for mobile manipulators in time-varying dynamic environments. A high-level motion planning module utilizes the environment information and solves the non-convex motion planning problem for the mobile manipulator. In addition, a low-level safety controller running at a higher sampling rate detects rapid changes in the environment and modifies the commands to assure safety locally. The main contributions are:

- The HRHC is proposed for mobile manipulator planning in dynamic environments.
- Experiments are conducted to verify the performance of the proposed control method (video is publicly available here).

5.2 Problem Formulation

5.2.1 Kinematic system modeling

The model of the mobile manipulator is similar to (3.7). However, to solve the planning problem quickly, we further linearize the model (3.3) and obtain:

$$z_{p,k+1} = \mathbf{A}_{p,k} z_{p,k} + \mathbf{B}_{p,k} u_{p,k}. \tag{5.1}$$

By connecting (5.1) and (3.6), the linearlized mobile manipulator model is obtained:

$$z_k = g_k(u_k, z(k)). \tag{5.2}$$

5.2.2 Formulation of the motion planning optimization problem

In this work, motion planning is done by solving an optimization problem. The planning problem is similar to Problem 2, except that now we plan in every time step. At time step k, current states are recorded in $z(k)$, the decision variables for each time step is u_k and the input vector that the problem optimizes is denoted as $\mathbf{u}_k := [u_k^\top, u_{k+1}^\top, u_{k+2}^\top, \cdots, u_{k+H-1}^\top]^\top$, where H is the prediction horizon. The state vector is $\mathbf{z}_{k+1} := [z_{k+1}^\top, z_{k+2}^\top, \cdots, z_{k+H}^\top]^\top$. Denote the kinematic relation $\mathbf{z}_{k+1} = f_k(\mathbf{u}_k, z(k))$ by concatenating the kinematic function (5.2) throughout the planning horizon. The following problem needs to be solved.

$$\begin{aligned} \min_{\mathbf{u}_k} \quad & J(\mathbf{u}_k, z(k)), \\ s.t. \quad & f_k(\mathbf{u}_k, z(k)) \in \Gamma_k, \\ & -\mathbf{u}_{max} \leq \mathbf{u}_k \leq \mathbf{u}_{max}. \end{aligned} \tag{5.3}$$

The planning horizon is usually no more than 30 time steps to ensure the computation speed. Notice that the main differences between this problem and Problem 2 are that the last state $z_{k+H}^{\top}$ in the plans during early stages may not reach the goal due to the velocity limit; a well-constructed initialization such as the one provided by RRT^* in Chapter 2 and 3 may not be provided; thus, the cost functions are also different. There are two assumptions in this formulation:

Assumption 2 (Cost)**.** *The cost function is convex and regular, and has the following form:*

$$J(\mathbf{u}_k, z(k)) = C_1\|\mathbf{D}\mathbf{u}_k - \mathbf{d}\|_2^2 + C_2\|\mathbf{V}\mathbf{u}_k - \mathbf{v}_{ref}\|_2^2 + C_3\|\mathbf{A}\mathbf{u}_k\|_2^2. \tag{5.4}$$

Here, C_1, the coefficient of the first term, penalizes the robot's deviation from the desired path and the desired manipulator pose so that the robot output trajectory is not too irregular. Matrix $\mathbf{D}$ is a transformation matrix that converts the decision variables, $\mathbf{u}_k$, to the states $\mathbf{z}_{k+1}$. Vector $\mathbf{d}$ contains the desired states of the mobile manipulator, i.e., the goal state vector. C_2, the coefficient of the second term, penalizes the speed profile of the planned trajectory relative to a constant speed reference $\mathbf{v}_{ref}$ so that the robot will reach the goal close to the desired timing. $\mathbf{V}\mathbf{u}_k$ is the velocity vector. C_3, the coefficient of the third term, penalizes the acceleration and angular acceleration of the mobile platform and the joints' angular acceleration so that the motion will be smooth. $\mathbf{A}\mathbf{u}_k$ is the acceleration/angular acceleration vector.

Assumption 3 (Constraint)**.** *The state constraint Γ_k is non-convex and its complement is a collection of disjoint convex sets, i.e., each of the obstacle-region is itself convex.*

In order to solve this non-convex problem fast enough for real-time implementation, the CFS algorithm [111] will be used.

5.2.3 Constraints formulation

To enable collision avoidance with human workers, we need an obstacle representation that satisfies Assumption 3. As shown in Figure 5.1, each link of the manipulator, as well as the links of the human worker, can be captured with a capsule. The function $\phi_{m,ij}(z(k))$ calculates the distance between ith manipulator link and the jth obstacle link. A disk captures the mobile platform and the obstacle projection area (starting from 10 cm away from the floor) on the floor. Therefore the distance $\phi_{p,j}(z(k))$ can be measured by calculating the distance between the center of the robot to the center of the jth obstacle projection. In this work, two cases are considered, collision avoidance and end-effector position keeping.

In collision avoidance, the convex feasible set corresponding to the jth obstacle-region during time step k at the rth iteration is:

$$F_j(\mathbf{u}_k^{(r)}) = \left\{\mathbf{u} : \phi_j(\mathbf{u}_k^{(r)}) + \nabla\phi_j(\mathbf{u}_k^{(r)})(\mathbf{u} - \mathbf{u}_k^{(r)}) \geq M_c\right\}. \tag{5.5}$$

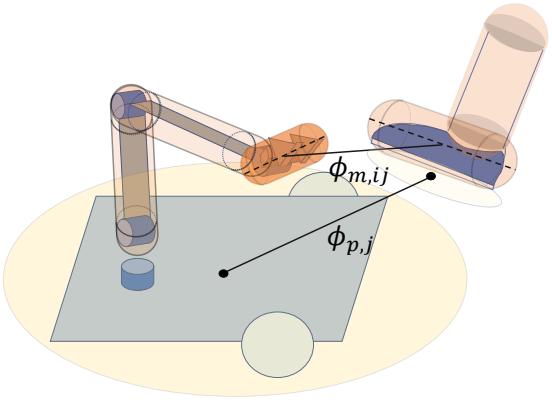


Figure 5.1: Distance function $\phi_{m,ij}(z(k))$ and $\phi_{p,j}(z(k))$.

The inequality constraints prevent collision and require the mobile manipulator to keep a margin, M_c, away from the obstacle.

On the other hand, for end-effector position keeping, obstacle-region is instead named as "lingering-target" and set corresponding to the lingering-target:

$$F(\mathbf{u}_k^{(r)}) = \left\{ \mathbf{u} : 0 \leq \phi(\mathbf{u}_k^{(r)}) + \nabla\phi(\mathbf{u}_k^{(r)})(\mathbf{u} - \mathbf{u}_k^{(r)}) \leq M_l \right\}. \tag{5.6}$$

The inequality constraints keeps the end-effector in the lingering-target within a margin M_l.

5.3 Hierarchical Receding Horizon Control

The following section introduces soft constraints in the motion planning problem and a low-level safety controller to realize closed-loop control for the mobile manipulator.

5.3.1 Soft constraints for implementation

The planning problem is solved in a receding horizon control (RHC) framework, and the robot implements the results accordingly. Although the planned trajectory from RHC is feasible, tracking errors will occur in real-world experiments, causing the robot to violate the margin boundary of the obstacles. The robot will then re-plan and try to immediately move away from the obstacle to maintain the margin, which results in a violent motion [26]. In order to avoid such problem, we introduce $\mathbf{S}^k = [s_{k+1}, s_{k+2}, \ldots, s_{k+H}]$, the slack variable vector (Figure 5.2). Introducing slack variables allows the states to violate the original constraint, i.e., the margin boundary. However, these slack variables are also added to the cost function to penalize the violation. The new problem is shown in the following:

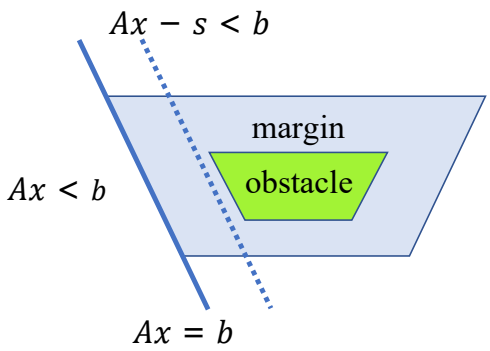


Figure 5.2: Illustration of the slack variable.

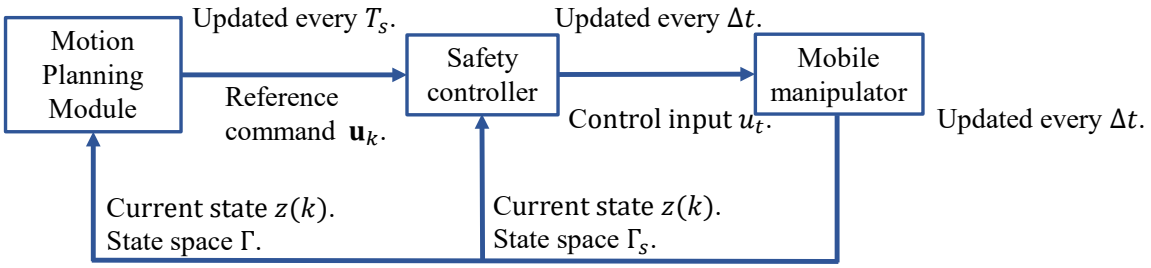


Figure 5.3: The overall control system.

Problem 4. *(Optimization problem with soft constraints):*

$$\begin{aligned} \min_{\mathbf{u}_k, \mathbf{S}^k} \quad & J(\mathbf{u}_k, z(k)) + \|\mathbf{S^k}\|_2^2, \\ s.t. \quad & f_k(\mathbf{u}_k, z(k)) \in \Gamma_k(\mathbf{S}^k), \\ & g_1(\mathbf{u}_k) = 0. \end{aligned} \tag{5.7}$$

5.3.2 Low-level safety controller

As mentioned in the introduction section, we should enable the motion planning algorithm to cope with many scenarios while maintaining a sufficient sampling rate. Here we introduce a low-level safety controller, which runs at a higher sampling rate, in the proposed HRHC.

The proposed overall control system is as shown in Figure 5.3. The low-level safety controller utilizes the reference command given by the motion planning module every T_s. The environmental detection, i.e., obstacle detection and prediction, and the current states, $z(t)$, are updated every Δt. Here, $T_s = c\Delta t$ and c is an even number. We consider the case where there is only one obstacle and it is moving at constant speed. The safety controller

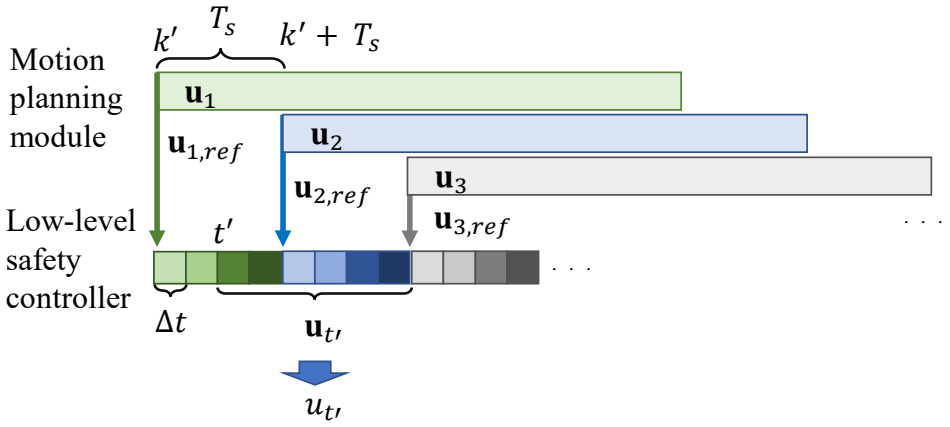


Figure 5.4: The hierarchical structure.

considers possible collisions in the future $1.5T_s$ time duration by checking the distances, $\phi_p(t)$, between future mobile manipulator positions and future obstacle positions. At a specific time step, $t = t'$ and $k' \leq t' \leq k' + T_s$, the inputs during t' to $t' + 1.5T_s$ is denoted as $\mathbf{u}_{t'}$ (Fig 5.4).

While the motion planning module aims for safety and efficiency, the low-level controller mainly focuses on safety. Therefore, to lower computation time, the low-level controller simplifies the problem and treats the mobile manipulator as a single capsule covering the whole robot. Thus, the entire system can have a higher sampling rate of the environment. At every Δt, the safety controller examines the original plan from the motion planning module given the new change in the environment. If the original plan is invalidated, the safety controller modifies the trajectory of the mobile platform by solving an optimization problem as stated below:

Problem 5. *(Optimization problem for the safety controller):*

$$\begin{aligned} \arg\min_{\mathbf{u}_{t'}} \quad & \|\mathbf{u}_{t'} - \mathbf{u}_{t',ref}\|_2^2, \\ s.t. \quad & f_{s,t'}(\mathbf{u}_{t'}) \in \Gamma_{s,t'}, \end{aligned} \tag{5.8}$$

where $\mathbf{u}_{t',ref}$ *is the original command,* $f_{s,t'}$ *is the kinematic function, and the collision-free set is*

$$\Gamma_{s,t'} = \{u_i : C_i f_{p,i}(u_i) - d_i \geq M_s \forall i = 0, \ldots, 1.5c - 1\}. \tag{5.9}$$

Here, M_s is a margin and $C_i f_{p,i}(u_i) = d_i$ is the hyperplane tangent to the obstacle that is the closest to the mobile manipulator at time $t' + i$. The optimization problem solves for a series of local commands to push the mobile manipulator into the collision-free set. With HRHC, as long as the collision-free set is reachable to the robot hardware, the mobile manipulator can always react to dynamic changes in the environment locally; thus, safety is guaranteed.

5.4 Applications

5.4.1 Experimental setup

To verify the performance, the HRHC (with $H = 15$) controller is tested on TB3O. The HRHC controller runs in MATLAB and python on a separate laptop with a 2.8 GHz Intel Core i7-7700HQ. An iterative LQR (ILQR) controller [159] is used for better tracking performance.

5.4.2 Experimental results

Four experimental setups are selected to verify the performance of the proposed HRHC controller. Experimental results are shown in Figure 5.5-5.8.

In the first and second scenarios, the mobile manipulator is expected to move along a line, $y = 0$, toward the positive x-axis direction while maintaining a neutral pose and constant speed (75% of the suggested maximum speed), which also points along the positive x-axis. The initial reference trajectory is a line segment in the state space. In the first scenario, the mobile manipulator tries to achieve the goal while avoiding an obstacle on the floor. In the second scenario, the mobile manipulator avoids obstacles both on the ground and hanging from above. In Figure 5.5(c) and Figure 5.6(c), the mobile manipulator avoids the obstacles successfully in both scenarios. In Figure 5.5(a) and Figure 5.6(a), the gray lines are the planned open-loop trajectories at each time step (colored from light to dark as time goes on). Although the open-loop trajectories may be different from each other, the overall trajectory is still smooth. Output angles are shown in Figure 5.5(b) and Figure 5.6(b), respectively.

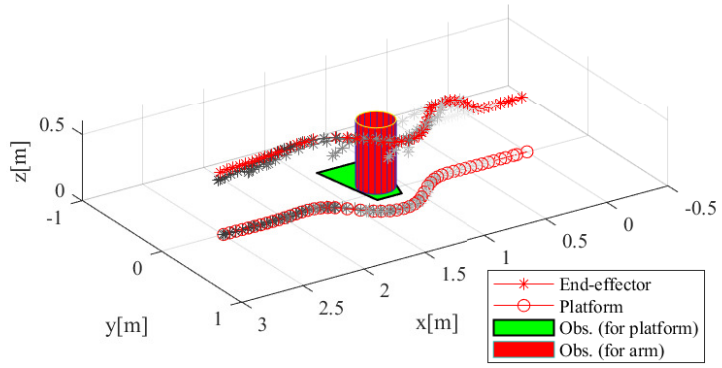
In the third scenario, a mobile manipulator is holding an object in place. However, a human worker is pushing a chair down the aisle while the mobile manipulator's platform appears to block the way. Because the worker has higher priority, the mobile manipulator should move to avoid collisions. Note that the mobile manipulator is expected to maintain its end-effector to stay close to a certain location. In Figure 5.7(c), it is shown that the mobile manipulator can keep the end-effector position within a small range and move the platform away from the aisle. Figure 5.7(b) shows the performance of end-effector position keeping.

The fourth and first scenarios are similar, except the obstacle here is a human worker standing on a ladder. In the beginning, the mobile manipulator can only see the ladder and plans to avoid it. When the worker comes down to the floor and walks away, it appears to the mobile manipulator that there is another obstacle appearing close to itself suddenly. The low-level safety controller will react to the change while bypassing the worker. In Figure 5.8(c), it is shown that the mobile manipulator performed successfully and can deal with the dynamic environment. The performance of the safety controller is shown in Figure 5.8(b). Around time step $k = 20$, it is shown that the platform angles changes ($k = 21$) before the manipulator starts to react ($k = 23$) to the new obstacle because the safety controller has

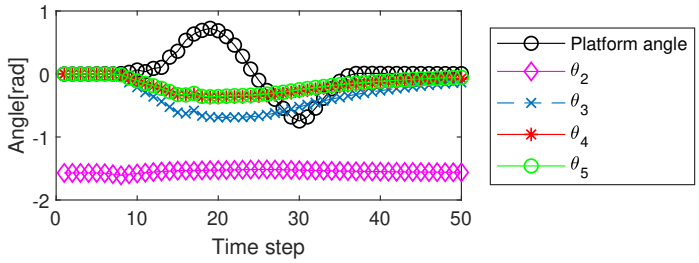
kicked in ($k = 21$) to make the platform avoid the worker. And after the planning module solves the new plan ($k = 23$), the manipulator will also retract to avoid the worker. With the safety controller, the overall system is sampling and reacting at 10Hz.

5.5 Chapter Summary

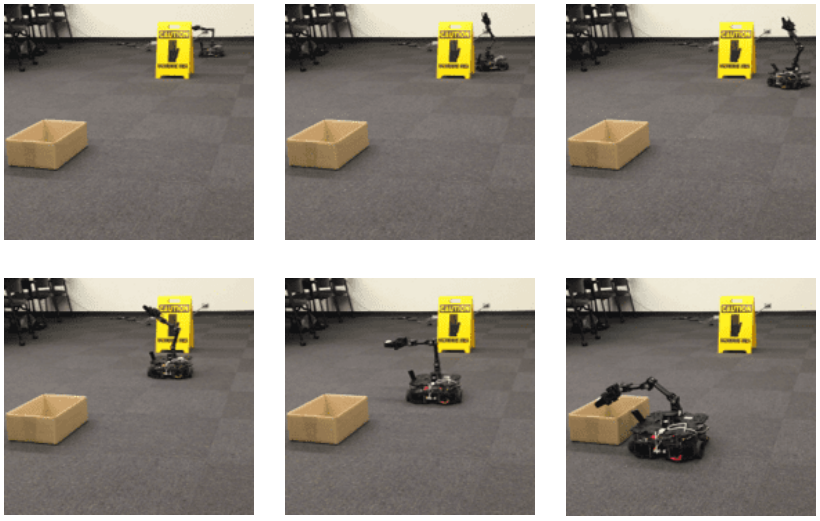
This chapter presented a motion planning method, HRHC (hierarchical receding horizon control), for mobile manipulators to handle time-varying and uncertain environments in industrial HRI systems. A high-level motion planning module considered the environment information and the mobile manipulator kinematic redundancy to better utilize the shared space and achieve higher efficiency. Combined with a low-level safety controller that modified commands locally, HRHC was developed to perform coordinated motion planning and guarantee safe maneuvers for mobile manipulators in dynamic environments. Experiments were conducted to evaluate HRHC. The results showed that the HRHC enabled the mobile manipulator to perform collision avoidance while completing its tasks successfully in both static and dynamic environments.



(a) Open-loop plans and the closed-loop result (scenario I).

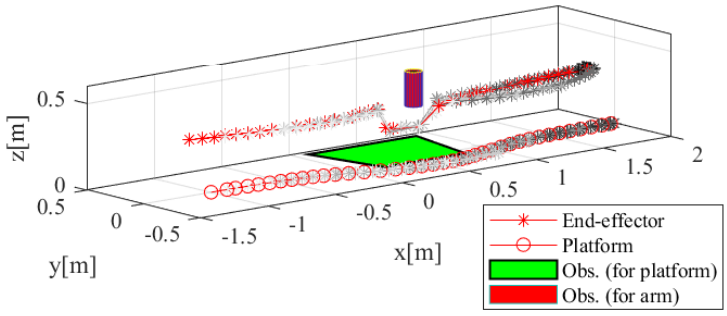


(b) Output angles of the platform and each joint (scenario I).

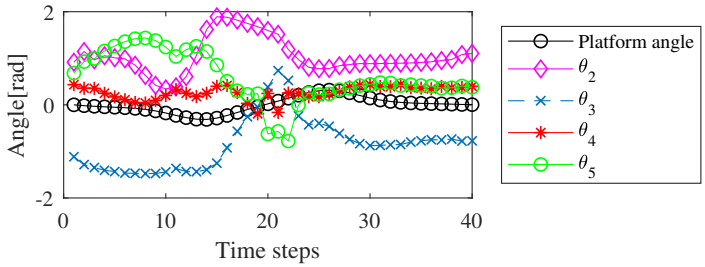


(c) Experimental result with mobile manipulator performing collision avoidance with an obstacle on the floor in the front.

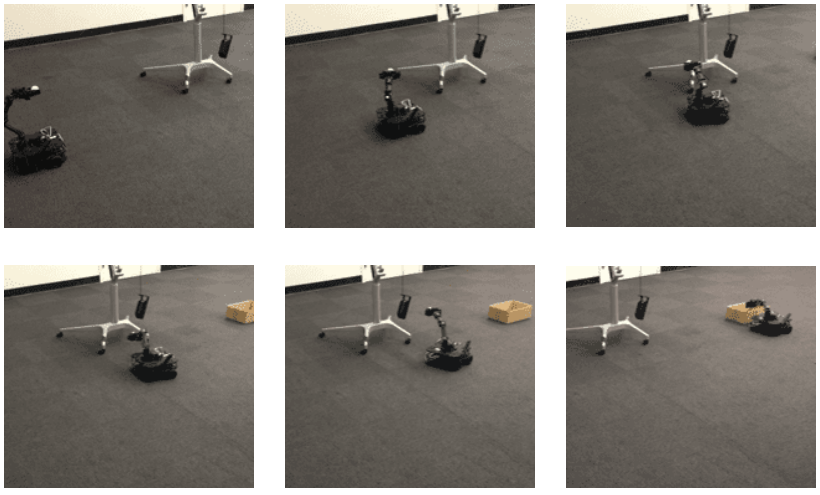
Figure 5.5: Results of collision avoidance in scenario I.



(a) Open-loop plans and the closed-loop result (scenario II).

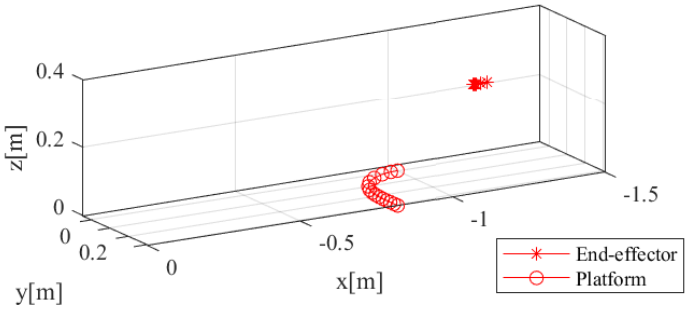


(b) Output angles of the platform and each joint (scenario II).

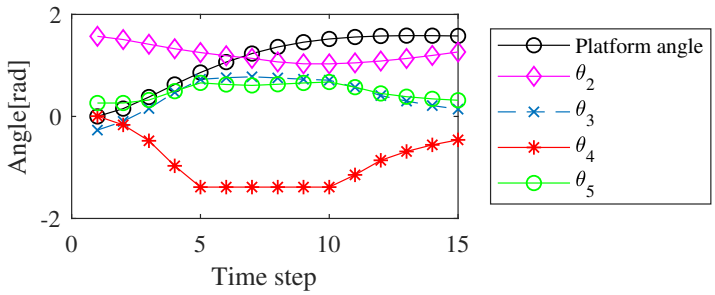


(c) Experimental result with mobile manipulator performing collision avoidance with one obstacle on the floor and one hanging from above.

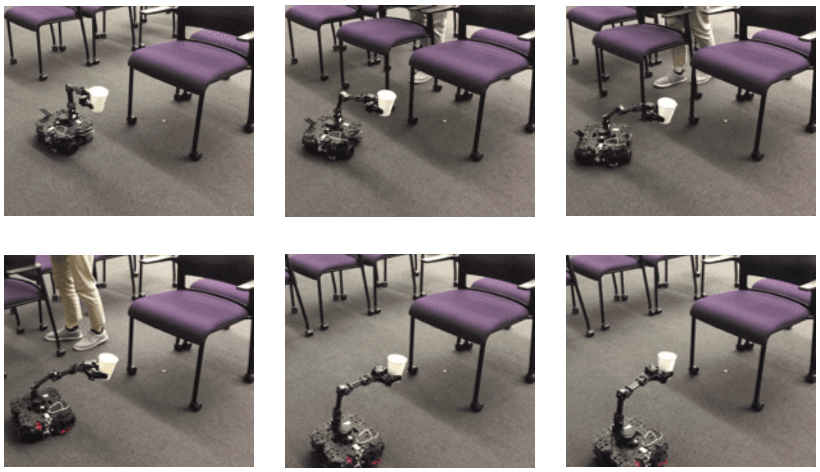
Figure 5.6: Results of collision avoidance in scenario II.



(a) Closed-loop result.

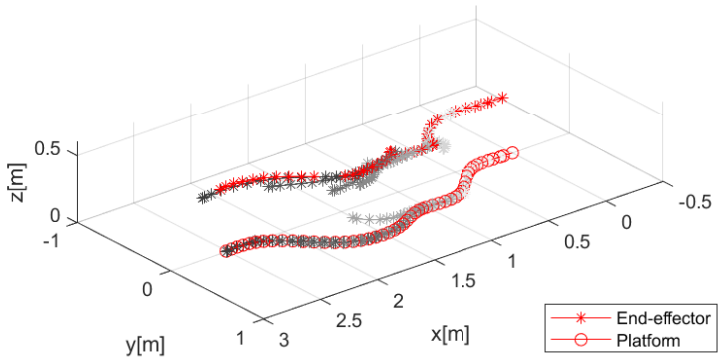


(b) Output angles of the platform and each joint.

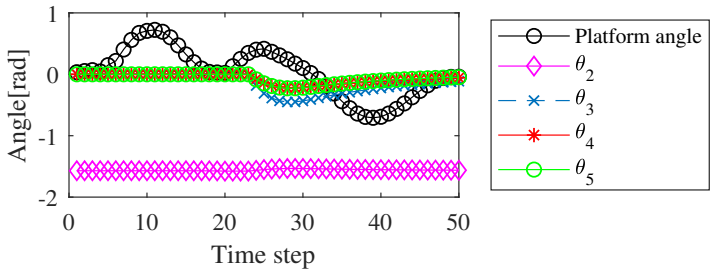


(c) Experimental result with mobile manipulator performing collision avoidance while keeping the end-effector position in the Cartesian space.

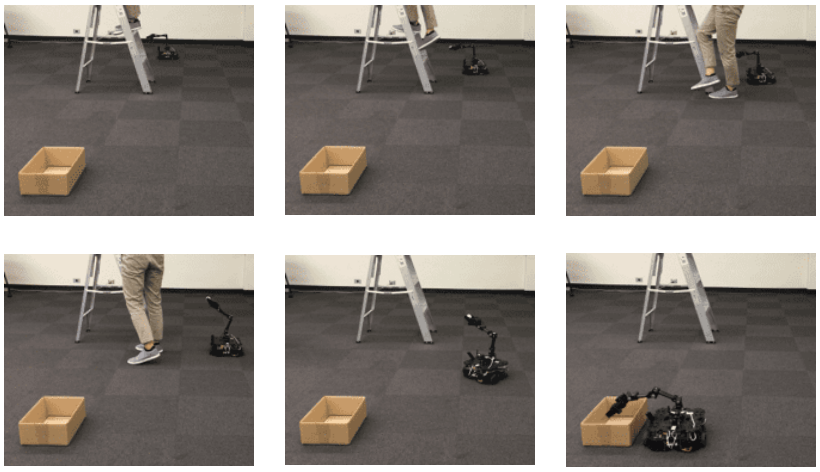
Figure 5.7: Result of end-effector position keeping.



(a) Open-loop plans (gray-star lines) and the closed-loop result (red lines).



(b) Output angles of the platform and each joint.



(c) Experimental results with mobile manipulator avoiding a moving human worker.

Figure 5.8: Result of avoiding moving human worker.

Part II

Integrated Strategies of Modularized Robotic Systems

Chapter 6

Environment Prediction and Motion Planning

6.1 Introduction

The second part of this dissertation moves from motion planning to a broader scope that considers the prediction module, the planning module, and their interactions. Looking from a motion planner point of view, we develop a predictor that can facilitate the motion planner for better robot performance in this chapter.

This chapter focuses on motion planners that are based on model predictive control (MPC) [137], an optimization-based algorithm that belongs to the category of receding horizon control. The fact that MPC observes the environment every time before solving the optimization problem allows the system to adjust and re-plan according to the changes in the environment; thus, it is a popular method for addressing robot motion planning problems. A typical MPC approach involves a sequence of receding horizon optimization. The sequential nature of such MPC formulation introduces a closed-loop with respect to the performance index for optimization, i.e., the cost function of the optimization problem. The closed-loop stability problem is a major issue when the environment, including the motions of obstacles, is dynamic. If the MPC problem is convex and has time-invariant constraints, the stability of the closed loop system can be guaranteed by modifying planning horizon, adding terminal cost, adding terminal equality constraints, or using terminal set constraints instead [118]. These methods ensure that the calculated commands and the resulting states are bounded in the presence of bounded disturbances. One common application of motion planning MPC problems is autonomous vehicles [16], where stability can be guaranteed by setting stability boundary for the control system, i.e., stability is quantified at several vehicle speeds. Another common application is industrial robots [58], where stability can be guaranteed by using Lyapunov-like functions and the terminal-state controller when the state space is convex.

However, two natures of motion planning in dynamic environments have contaminated

the stability properties of MPC. First, the existence of obstacles in the environment introduces non-convex state constraints, which result in time-varying and non-convex MPC problems. This makes the closed-loop stability of MPC hard to analyze. Second, environmental changes induce changes of the constraints of the optimization problem. The fact that optimization problems are sensitive to the constraints makes the accuracy of the prediction extremely crucial because the prediction determines the state constraints of the problem. Commonly seen assumptions in MPC, such as obstacles are moving in constant velocity or acceleration (the method used in Chapter 5), may not suit the environment well and cause the open-loop trajectories at each time step to vary largely. This might result in dynamically unreasonable closed-loop trajectories. Dynamically unreasonable closed-loop trajectories will cause the robot to execute violent or zigzagging movements that harm the robot's motors and scare other workers in the environment. Fortunately, with some assumptions, it is possible to consider the uncertainties in the MPC problem properly. Some works have been focusing on dealing with uncertainties in MPC problems [33, 89]. In these works, systems under persistent disturbance can be controlled by robust MPC. However, a more common scenario in motion planning problems is to have uncertainties in the environment, e.g., an environment with obstacles moving at varying speeds. Therefore, a probabilistic model of the obstacles' future movement is needed. In [120], stochastic MPC deals with probability constraints to handle uncertainties. Another approach is to directly predict the object's future motion to achieve even higher efficiency. In [56, 85], the authors focus on the industrial HRI environment, where human workers usually cause uncertainties in the environment. It is clear that although the workers do not explicitly reveal their intention, it is still possible to predict their future movements based on past observations [30, 21]. In [87], a Bayesian filter is used for motion prediction given observations in the past and the state transition model.

To achieve better robot performance, we identify conditions for the predictor and the MPC so that closed-loop stability is guaranteed when the environment satisfies some assumptions, and collision avoidance can be achieved even though the assumptions are not satisfied. With the conditions, we proposed an example of MPC with stability-enhanced prediction, which utilizes the past and current observations of the environment to predict a worker's intention and construct the state space for the optimization problem at each MPC time step. It is assured that the proposed method is stable theoretically in the sense of Lyapunov in a set of common factory scenarios that satisfies the assumptions. The proposed method can also better deal with environment uncertainties which addresses the limitation of MPC without stability-enhanced prediction, i.e., MPC that only uses current observations and always assumes constant velocity or constant acceleration. The contributions of this chapter are:

- We discuss and identify the conditions to enable closed-loop stability.
- We provide an example of a motion planning method with a closed-loop stability-enhanced predictor in an MPC framework.

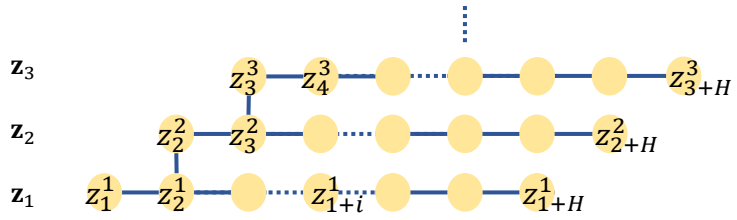


Figure 6.1: The execution structure of MPC.

- A new notion of open-loop prediction convergence property called M-convergence is proposed serves as an indicator of the closed-loop performance.
- Simulation studies and experiments are carried out to verify the proposed method (video is publicly available here).

6.2 Stability of MPC-based Planning

6.2.1 Traditional non-convex MPC and notations

In MPC (Figure 6.1), at each time step t, a future trajectory will be planned by solving an optimization problem. This trajectory is denoted as $\mathbf{z}_k := [z_k, z_{k+1}, z_{k+2}, \cdots, z_{k+H}]$ where the state, z_k, is called the k^{th} action location. H is the planning horizon. In this chapter, $z_k = [x(k), y(k)]^\top$ contains the x and y coordinate of the robot's location in 2-dimensional Cartesian space at time step $t = k$. Note that z_k corresponds to the current position, denoted as $z_0(t = k) = (x, y)|_{t=k}$. The trajectory will then be executed and the robot will go to z_{k+1}, the $k+1^{th}$ action location. The sampling time between two time steps is a constant, denoted as Δt. After reaching the next action location, the robot will again solve the optimization problem and plan a new trajectory and repeat the process. To avoid confusion, the current time step can also be marked as superscript in some cases , e.g., z_{k+1}^k means the planned action location z_{k+1} at time step $t = k$.

At time step $t = k$, given the current state, the following optimization needs to be solved to obtain $\mathbf{z}_k$,

$$\begin{aligned} \min_{\mathbf{z}_k} \quad & J(\mathbf{z}_k), \\ s.t. \quad & \mathbf{z}_k \in \Gamma_m^k(z_0(k), \mathbf{O}_m^k). \end{aligned} \tag{6.1}$$

This optimization is performed at every MPC time step k as time evolves. There are two assumptions in this formulation:

Assumption 4 (Cost)**.** *The cost function, containing both the stage cost and the terminal cost, is convex, regular, and time-invariant, and has the following form*

$$J(\mathbf{z}_k) = C_1\|\mathbf{D}\mathbf{z}_k - \mathbf{z}_{ref}\|_2^2 + C_2\|\mathbf{V}\mathbf{z}_k - \mathbf{v}_{ref}\|_2^2 + C_3\|\mathbf{A}\mathbf{z}_k\|_2^2. \tag{6.2}$$

Similar to Assumption 2, C_1, C_2, and C_3 are the coefficients. The first term penalizes the robot's deviation from a reference line. Matrix $\mathbf{D}$ is a projection matrix that extract all y's from $\mathbf{z}_k$. The second term penalizes the speed profile of the planned trajectory with regard to a constant speed that goes along the positive x-axis. The third term penalizes the acceleration of the output trajectory. $\mathbf{A}\mathbf{z}_k$ is the acceleration vector.

Assumption 5 (Constraint)**.** *The state constraint* Γ *is non-convex and its complement,* $\mathbf{O}$*, is a collection of disjoint convex sets.*

Based on the current observation, assuming that the obstacles are moving in constant speed (indicated by the subscript m), the obstacle region, $\mathbf{O}_m^k$, can be obtained. The set of state constraints, Γ_m^k, satisfies Assumption 5. The current state, $z_0(k)$, is measured and assigned to the first entry of $\mathbf{z}_k$. The final trajectory is $[z_k^k, z_{k+1}^{k+1}, \ldots]$, which is equivalent to $[z_k^{k-1}, z_{k+1}^k, \ldots]$ (Figure 6.1) if the tracking controller is perfect.

6.2.2 Closed-loop stability of MPC-based planning problems

As discussed previously, uncertainties in the environment are usually caused by human workers or other robots. Here, we identify the necessary conditions that enable closed-loop stability. First, the scenario where the worker's motion is predictable needs to be identified. Assuming that the worker's movements can be captured by a predictor with finite modes, $m \in \{m_1, m_2, \ldots, m_d, m_{up}\}$, where m_i, $i = 1, \ldots, d$ are the modes where the worker's movements are predictable and m_{up} captures all the other unpredictable movements. An assumption is made in the predictable modes:

Assumption 6 (Obstacle motion)**.** *Assuming that the acceleration and velocity of the obstacle are bounded, the velocity,* v_{obs}*, of the obstacle during* m_i *mode is also bounded, i.e.,* $\|v_{obs}\| \leq v_{m_i}^u$.

Therefore, we can define the predictable mode as the following:

Definition 1 (Predictable mode)**.** *During predictable mode* m_i*, there exists an obstacle region* $\mathbf{O}_i^k$ *that covers the obstacle movements for the coming* T_{ci} *time duration, and thus, the movements in this mode are predictable. Here,* T_{ci} *is called the time delay allowance.*

With the Definition 1, we can define closed-loop stability of MPC in time-varying environments when the obstacle's movement mode is predictable as the following:

Definition 2 (Closed-loop stability of MPC)**.** *Let* $J^*(k)$ *be the optimal cost at time step* k*. MPC is stable in closed-loop if the optimal cost function is a Lyapunov function, i.e.,* $0 \leq J^*(k) \leq J^*(k-1)$ *for all* k *during predictable mode* m_i.

Denote k_s to be the time when the obstacle movement switches in between different modes, and k_{ps} to be the time when the predictor detects that change. Denote T_d to be the time delay between k_s and k_{ps}, i.e., $T_d = k_{ps} - k_s$. With Assumption 6, the first condition for the overall system to be stable and safe is:

Condition 1 (Condition for the predictor)**.** *The time delay, T_d, needs to satisfy $T_d < T_{ci}$ for all $i = 1, \ldots, d$.*

This is important for safety reason especially when the worker is switching from predictable modes to unpredictable mode. This condition guarantees that the obstacle region can cover the worker's movements even though the predictor hasn't detected the mode change. Therefore, the motion planner can still plan for a solution that avoids collision. A predictor that can recognise mode defined in Definition 1 and satisfies Condition 1 is called a stability-enhanced predictor. With the above assumptions and condition, we have the first main result:

Main result 1 (Convergence of the state constraints)**.** *A series of obstacle region, $\mathbf{O}_i^k$, can be chosen by the predictor and $\mathbf{O}_i^k$ satisfies the relationship $\mathbf{O}_i^k \subseteq \mathbf{O}_i^{k-1}$ during predictable mode m_i. The set of state constraints, Γ_i^k, is the complement of $\mathbf{O}_i^k$ such that $\mathbf{z}_k \in \Gamma_i^k$ and $\Gamma_i^{k-1} \subseteq \Gamma_i^k$ during mode m_i.*

The MPC problem during mode m_i is:

$$\begin{aligned} \mathbf{z}_k^* = \arg\min_{\mathbf{z}_k} \quad & J(\mathbf{z}_k), \\ s.t. \quad & \mathbf{z}_k \in \Gamma_i^k(z_0(k), \mathbf{O}_i^k). \end{aligned} \tag{6.3}$$

Here, Γ_i^k also satisfies Assumption 5. Denote $J^*(k) = J(\mathbf{z}_k^*)$.

A condition on the MPC controller is also needed to guarantee a sufficient knowledge of the obstacle size in the planning problem:

Condition 2 (Planning horizon and terminal cost)**.** *Both the planning horizon H and the penalty on the terminal cost are sufficient that during mode m_i, the open-loop problem always solves for a solution that can bring the robot to completely pass by the obstacle.*

With Assumption 4, the cost function is time-invariant and quadratic. With Condition 2 and Main result 1, the number of future way points affected by the obstacle is non-increasing. Therefore, $\Gamma_i^{k-1} \subseteq \Gamma_i^k$ implies that the open-loop planning problem is finding a solution in a larger and larger space throughout time with regard to a time-invariant cost function during mode m_i.

Main result 2 (Closed-loop stability of MPC)**.** *The cost of the optimal open-loop trajectory follows $0 \leq J^*(k) \leq J^*(k-1)$ when Assumption 4 to 6 and Condition 1 and 2 are satisfied during mode m_i. Therefore, the proposed MPC with stability-enhanced prediction is stable in the sense of Lyapunov during mode m_i before encountering a new obstacle or the worker begins to leave the obstacle region, i.e., m switches back to m_{up}.*

Regarding the closed-loop stability, the main results may seem to be relatively straight forward to obtain under the assumptions. However, it is important that with Condition 1, the proposed method can react to situations when the condition of a Lyapunov function is violated, i.e., collision is avoided and thus, safety is guaranteed at all times.

6.3 Stability-enhanced Prediction and M-Convergence for Analysis

6.3.1 An example of a stability-enhanced predictor

The following example presents a preliminary closed-loop stability-enhanced predictor, which provides needed information to improve closed-loop stability. First, the scenario where the workers' motions are predictable needs to be identified. In a factory setting, human workers and mobile robots usually move toward a place and stop to complete some tasks. Typically, they will stay in an area for an amount of time which is likely to be long enough for the ego robot to consider them static objects with margins. The robot can then plan a path to go around them so that the shared space is better utilized, i.e., space efficiency is increased. We divided the worker's movement into two modes, moving mode and working (stop) mode. Denote the mode as $m \in \{moving, working\}$. Here $moving$ is the unpredictable mode and $working$ is the predictable mode. Therefore, it is important to know whether the worker is going to stop, i.e., switching to predictable mode. To do such prediction, denote the past position of the obstacle at time $t = k$ be $z_{obs}(k)$, the past path of the obstacle is $\mathbf{h}^k_{obs} = [z_{obs}(0), z_{obs}(1), \ldots, z_{obs}(k)]^\top$. Denote the probability for the obstacle to continue moving and to stop be $p_{go}(k)$ and $p_{stop}(k)$, respectively ($p_{go}(k) + p_{stop}(k) = 1$). Let r_1 and r_2 be the update rates ($r_1 > 1$ and $0 < r_2 < 1$) and b be the bias term. Notice that T_d is determined by the choices of these constants. We have the probability-update algorithm as shown in Algorithm 4.

Although the worker/robot has stopped, it is still possible that he/she will have some movements within the working area (i.e., obstacle region). Therefore, the ego robot needs to know potentially how big the area is by observing the movements of the worker/robot so that its motion will be less affected by their movement uncertainty. A naive Bayesian filtering-based method is presented to track the obstacle (e.g., other moving robot or human worker) and give a safety margin according to the past movements that the robot should keep away from the obstacle.

Each past position has an associated weight $p(z_{obs}(k))$, and the weight vector for $\mathbf{h}^k_{obs}$ is $\mathbf{p}^k_{obs} = [p(z_{obs}(0)), p(z_{obs}(1)), \ldots, p(z_{obs}(k))]^\top$. In each MPC time step, $\mathbf{p}^k_{obs}$ is updated:

$$\begin{aligned} \hat{\mathbf{p}}^k_{obs} =& \left[\left[\gamma \mathbf{p}^{k-1}_{obs} \right]^\top 1 \right]^\top, \\ \mathbf{p}^k_{obs} =& \frac{\hat{\mathbf{p}}^k_{obs}}{\|\hat{\mathbf{p}}^k_{obs}\|_1}, \end{aligned} \tag{6.4}$$

Algorithm 4: The probability-update algorithm.

input $\mathbf{h}^k_{obs}, p_{go}(k-1), p_{stop}(k-1)$
$Acc \leftarrow$`getAcc`$(\mathbf{h}^k_{obs})$
$endVel \leftarrow$`PredictEndVel`(Acc)
if `abs`$(endVel) \geq threshold$ **then**
    $w_{go} \leftarrow r_1 p_{go}(k-1)$
    $w_{stop} \leftarrow r_2 p_{stop}(k-1) + b$
else
    $w_{go} \leftarrow r_2 p_{go}(k-1) + b$
    $w_{stop} \leftarrow r_1 p_{stop}(k-1)$
$p_{go}(k), p_{stop}(k) \leftarrow$`normalize`$(w_{go}, w_{stop})$
return $p_{go}(k), p_{stop}(k)$

where γ is the discount constant ($0 < \gamma < 1$). Let $\mathbf{h'}^k_{obs}$ and $\mathbf{p'}^k_{obs}$ be the last l elements of $\mathbf{h}^k_{obs}$ and $\mathbf{p}^k_{obs}$, respectively. To determine the obstacle region in the motion planning problem at each MPC time step, denote the space uncertainty index as $\sigma(k) = std(\mathbf{p'}^k_{obs} \circ \mathbf{h'}^k_{obs})$, where $\circ$ denotes the entry-wise multiplication. A robot that travels a larger distance in the last several time steps will result in larger $\sigma(k)$, which serves as an indicator of position uncertainty in the future under the assumption that the future movement of the obstacle is related to the current and past movements.

6.3.2 State space formulation and stability

As denoted previously, k_{ps} is the time when the predictor detects that the worker/robot has switched to working mode ($m = working$), i.e., $p_{go}(k_{ps}) \leq c p_{stop}(k_{ps})$ with some $c \in (0, 1)$. In the following, we construct the state space during working mode ($k > k_{ps}$) with the information presented in the previous section. Denote the high-dimensional obstacle region in the trajectory space at each MPC time step as $\mathbf{O}^k_w(\sigma(k), \mathbf{h'}^k_{obs})$. We consider the scenario where there is an obstacle coming into the scene and staying within a working area (obstacle region) while moving back and forth to complete a task in the area.

With Assumption 6, it can be shown that $\sigma(k)$ has an upper bound, $\sigma(k) \leq \mathcal{O}(l^2 v^u_{working})$, where $l = 10$ in this work case. Therefore, the motion is predictable and it is possible to find $\mathbf{O}^k_w$ that can cover the obstacle movements as stated in Definition 1. With a proper choice of the update rates, the Condition 1 holds and Main result 1 can be applied.

Again, the set of state constraints, Γ^k_w, is the complement of $\mathbf{O}^k_w$ and satisfies Assumption 5. The MPC problem during mode *working* is:

$$\begin{aligned} \mathbf{z}^*_k = \arg\min_{\mathbf{z}_k} \quad & J(\mathbf{z}_k), \\ s.t. \quad & \mathbf{z}_k \in \Gamma^k_w(z_0(k), \mathbf{O}^k_w). \end{aligned} \tag{6.5}$$

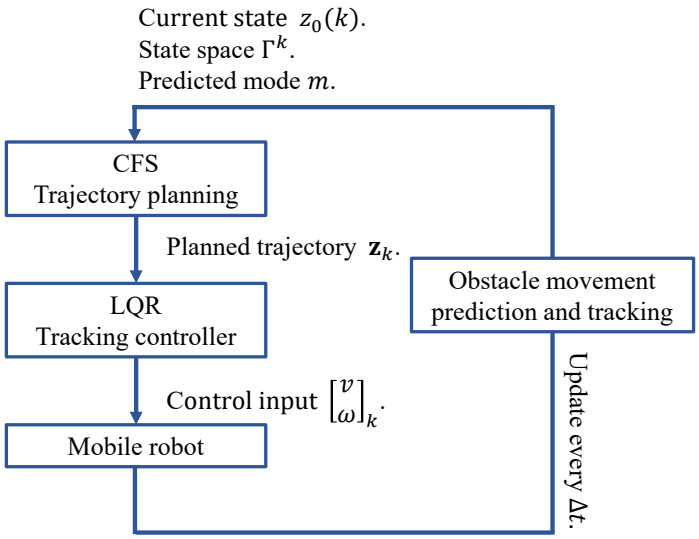


Figure 6.2: The overall system control design.

Denote $J^*(k) = J(\mathbf{z}_k^*)$. The cost of the optimal open-loop trajectory follows $0 \leq J^*(k) \leq J^*(k-1)$ when Assumption 4 to 6 and Condition 1 and 2 are satisfied. Therefore, the proposed MPC with stability-enhanced prediction is stable in the sense of Lyapunov for all k during mode *working*.

6.3.3 Overall system

The overall system is shown in Figure 6.2. Similar to Section 5.3.1, we introduce the slack variable vector $\mathbf{S}^k = [s_{k+1}^k, s_{k+2}^k, \ldots, s_{k+H}^k]$ and modifies the cost function accordingly for better real-time robot performance. The new problem is shown in the following:

$$\min_{\mathbf{z}_k} \quad J(\mathbf{z}_k) + \|\mathbf{S}^k\|_2^2, \tag{6.6}$$

$$s.t. \quad \mathbf{z}_k \in \Gamma^k, \tag{6.7}$$

where Γ^k is:

$$\Gamma^k = \begin{cases} \Gamma_w^k(z_0(k), \mathbf{O}_w^k, \mathbf{S}^k), & \text{if } m = working \\ \Gamma_m^k(z_0(k), \mathbf{O}_m^k, \mathbf{S}^k), & \text{otherwise} \end{cases}. \tag{6.8}$$

An iterative LQR (ILQR) controller [159] is used for a better tracking performance, which outputs the control commands $[v, \omega]_k^\top$. The overall system goes through a process as shown in Algorithm 5 at ever MPC time step.

Algorithm 5: The overall algorithm.

input $M_{map}, M_{route}, X_{goal}$
while *MPC* **do**
  `GetInformation()`
  `Predictor()`
  if $p_{go}(k_w) \leq cp_{stop}(k_w)$ **then**
    $m = working$
  else
    $m = moving$
  $\mathbf{z}_k \leftarrow$`CFS`$(z_0(k), \Gamma^k, m)$
  $[v,\ \omega]_k^\top \leftarrow$`ILQR`$(\mathbf{z}_k)$
  ego robot executes the control commands

6.3.4 M-Convergence for analyzing closed-loop performance

In this chapter, we propose a new notion, M-convergence, to analyze the transient of the planned open-loop trajectory from one planning instance to the next.

Definition 3 (M-convergence)**.** *We say that an action location, z_k, is M-converging if for $k - M < t \leq k$, the last M predictions for z_k satisfy: (i) $\|z_k^t - z_k^{t-1}\| \leq \|z_k^{t-1} - z_k^{t-2}\|$, or (ii) $\left|\|z_k^t - z_k^{t-1}\| - \|z_k^{t-1} - z_k^{t-2}\|\right| \leq \delta$, when $\|z_k^t - z_k^{t-1}\| \leq \epsilon$, or both. Here δ and ϵ are small thresholds.*

In Figure 6.3, the pink-orange-color circles are the action location z_k planned at different time steps that are tested for M-convergence of z_k. The conditions imply that the planned state z_k would have smaller and smaller change (i) or sufficiently small change (ii) on the predicted action location between consecutive time steps after time step $k - M$. This notion allows us to examine the local convergence property of the open-loop trajectories to the closed-loop trajectory at each action location.

Having such property benefits the robot's performance. M-convergence guarantees that the action location will not change much from plan to plan, therefore, guaranteeing smoothness of the output trajectory. If M is sufficiently large, the robot system will not experience sudden changes. Moreover, since the planned trajectory is usually tracked by a low-level tracking controller, the larger the M is, the smoother the control command would be, which is important for the longevity of the robot hardware. We use M-convergence as one of the metrics to evaluate the robot's closed-loop performance.

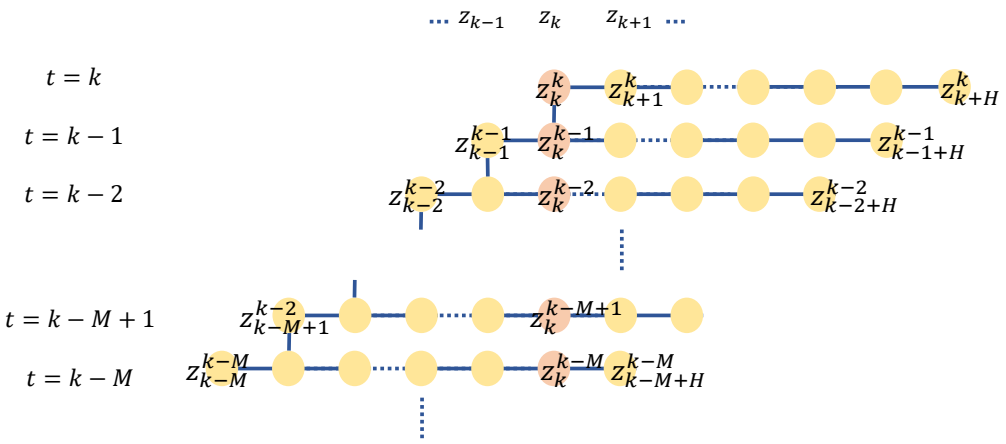


Figure 6.3: Illustration of M-convergence.

6.4 Applications

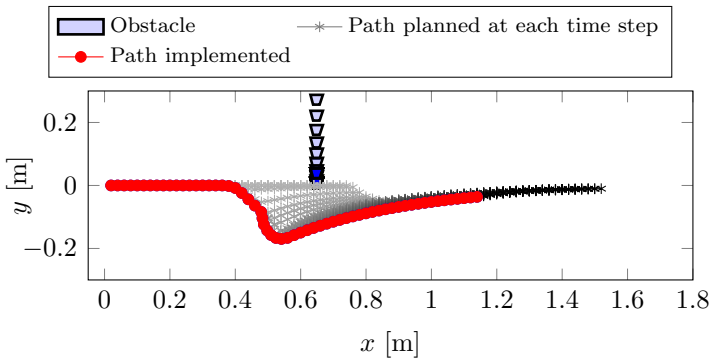
6.4.1 Results and discussion

The simulation scenario replicates factory settings that involve mobile robot operations. To test the proposed algorithm and see the improvement of the proposed method, a typical industrial environment with a human-robot shared space is considered. The goal for the robot is to move along $y = 0$, while maintaining a constant speed that points along the positive x-axis.

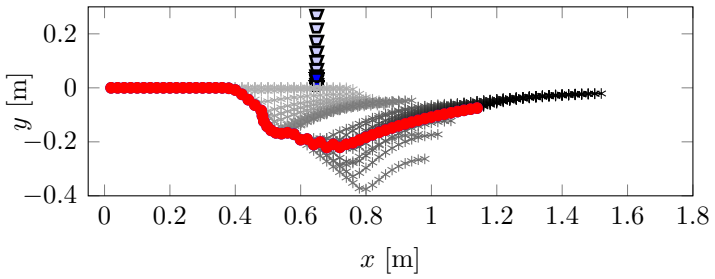
Simulation Results of The Scenario With shared Working Area

At the beginning of the simulation, the human worker walks (moving mode) in constant acceleration and stops in front of the ego robot. The worker then starts to work in the area and moves back and forth in a small range (working mode). The robot needs to determine the size of the working area and not overreact to the worker's small but inconsistent movements while bypassing the working area.

Simulation results are shown in Figure 6.4(a) and Figure 6.4(b), where the blue blocks represent the worker's positions, and the gray star-lines represent the planned trajectories at each time step (darker colors represent time steps that are closer to the current). Here, $k_{ps} = 31$, i.e., the worker switches from moving mode to working mode at time step 31. We can see that the proposed method has a smooth closed-loop trajectory, while MPC without stability-enhanced prediction experiences oscillation on the open-loop prediction due to the worker's small movements in the working mode. Correspondingly, Figure 6.5(a)



(a) Simulation result of the proposed MPC with stability-enhanced prediction.



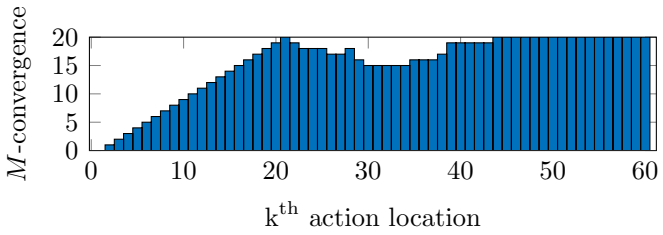
(b) Simulation result of MPC without stability-enhanced prediction.

Figure 6.4: Scenario that has oscillating moving obstacles.

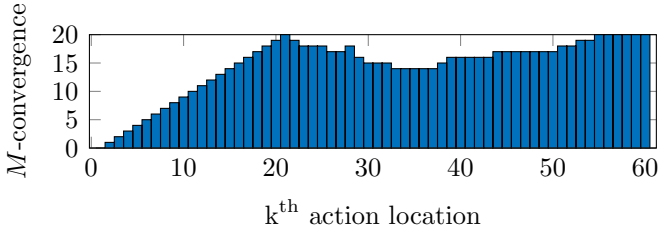
and Figure 6.5(b) also show that the proposed method recovers faster in M-convergence after the worker switches to working mode. As shown in Figure 6.5(c), the proposed method can plan safely around the worker while not overreacting to the worker's small movements. On the other hand, the red peaks indicate that MPC without stability-enhanced prediction reacts strongly to the speed change and results in open-loop trajectories with high cost. In Figure 6.5(d), the cost with the proposed method is the same as MPC without stability-enhanced prediction in the first 12 steps. However, it is noticeably smaller afterward because the proposed method can better utilize the share-space and has better space efficiency and overall performance.

Discussion

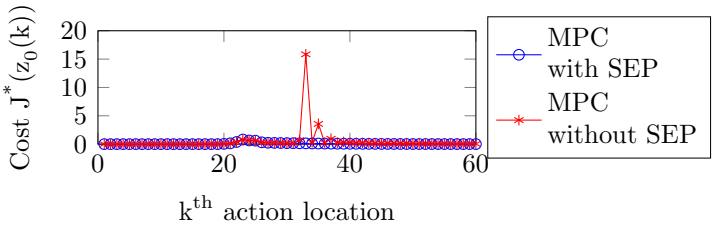
The proposed method is also tested in similar scenarios with different factors, e.g., the worker's acceleration and range of movement during the working mode. It is shown that the ego robot can always complete its task without colliding with an obstacle. We also observed that the performance depends on the parameters of the predictor. Thus, a better stability-enhanced predictor can improve the robot's performance. The preliminary predictor



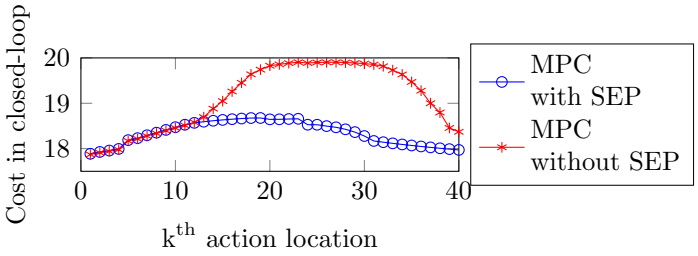
(a) M-convergence for simulation result of the proposed MPC with stability-enhanced prediction.



(b) M-convergence for simulation result of MPC without stability-enhanced prediction .



(c) Comparison for open-loop cost of MPC with/without stability-enhanced prediction.



(d) Comparison for closed-loop cost of MPC with/without stability-enhanced prediction.

Figure 6.5: Comparison between MPC without stability-enhanced prediction and the proposed MPC with stability-enhanced prediction.

introduced can be replaced by more advanced ones in the future.

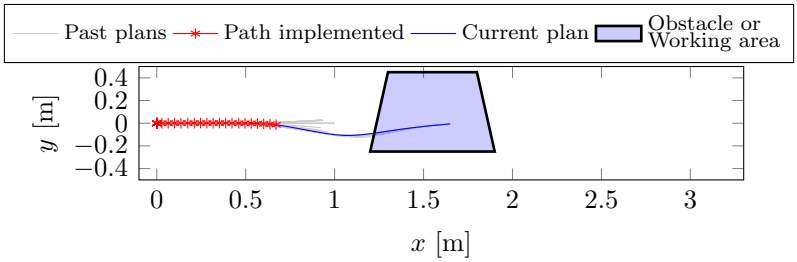
6.4.2 Experimental results

To verify the performance, the proposed MPC (with $H = 20$) controller with stability-enhanced prediction is tested on TurtleBot3 (Figure 3.5). The MPC controller runs in MATLAB on a separate laptop with a 2.8GHz Intel Core i7-7700HQ and communicates with the TurtleBot3 through ROS. Three experiments are conducted.

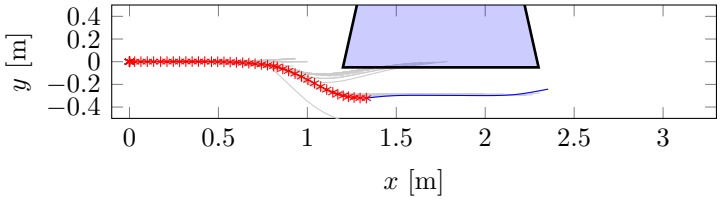
The first scenario is similar to the simulation scenario. The ego robot runs the proposed MPC with stability-enhanced prediction, and another remote-controlled robot (obstacle robot) represents a worker. The experimental result is shown in Figure 6.8, where the robot successfully avoids the obstacle and merges back to the original route without overreacting to the other robot's movements in the working area.

In the second scenario (Figure 6.9), a human worker, having a higher priority, i.e., doesn't need to yield to the robot, comes toward the table to perform some tasks and leaves the working area in the opposite direction of the robot after finishing the tasks. The robot running the proposed MPC with stability-enhanced prediction avoids the human worker while tracking a straight line. From Figure 6.6(a), we can see that at the beginning, the robot detects the worker's movement, and the prediction assumes constant speed. After the worker switches to the working mode, the robot predicts the working area and plans accordingly (Figure 6.6(b)). Finally, after the worker leaves the working area and starts to walk away from the robot, the robot detects that movement, switches m back to $m = moving$, and merges back to the line $y = 0$ before the turning points of the old plan when $m = working$. Figure 6.6(c) shows that the gray lines turn earlier and earlier compared to the blue line in Figure 6.6(b) as the worker leaves.

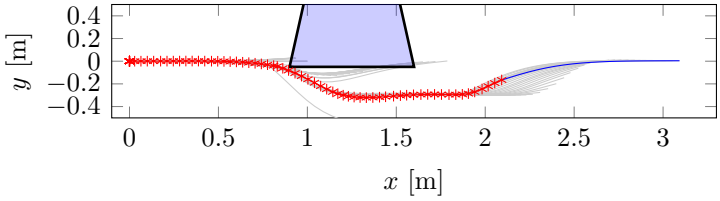
The third and the second scenarios are similar, except that the worker leaves from the other side of the table. Therefore, the worker walks in the same direction as the robot for a short period after leaving the working area. This experiment shows that the proposed method can react and avoid collisions in situations when the worker is switching back to unpredictable mode. In other words, safety is still maintained even though the cost of the optimal open-loop is no longer a Lyapunov function. From Figure 6.7(a), the robot behaves the same as in scenario 2 shown in Figure 6.6(b). After the worker switches back to moving mode, the robot detects that movement, switches m back to $m = moving$, and can successfully re-plan and continue to avoid collision (Figure 6.7(b)) instead of merging back to $y = 0$ as planned previously (Figure 6.7(a)). Also, because the working area is sufficient to capture the obstacle movement when the predictor is detecting the change, which fulfills Condition 1, the path implemented is smooth even though there is a change in the worker's mode. Finally, after the worker starts to leave the scene along the positive y direction, the robot begins to merge back to the line $y = 0$ (Figure 6.7(c)). The experimental results show that with the proposed method, the prediction and the motion planning modules can work collaboratively so that the robot can deal with a time-varying environment and efficiently utilize the shared working space.



(a) Experimental result when $m = moving$.



(b) Experimental result when $m = working$.

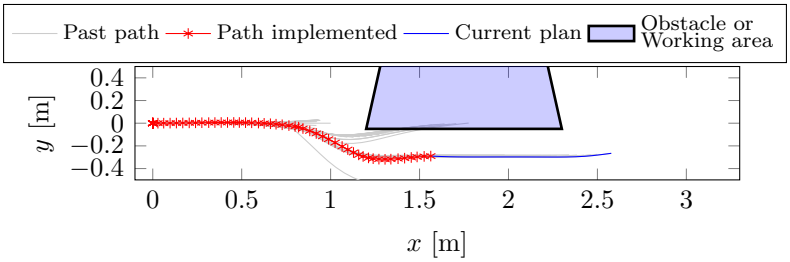


(c) Experimental result when m is switched back to $m = moving$ (obstacle is moving away from the robot).

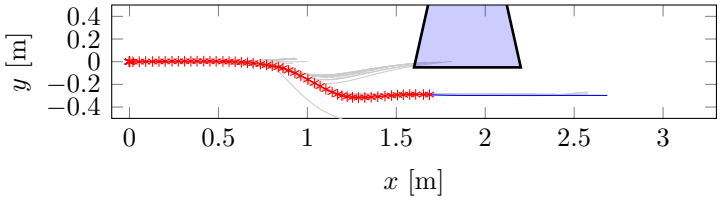
Figure 6.6: Path planned and implemented in the second scenario.

6.5 Chapter Summary

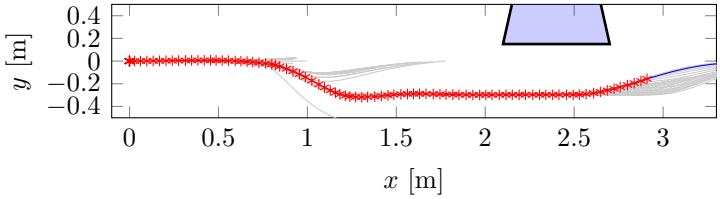
This chapter extended our focus from the motion planning module to the prediction module and the interaction between the two. We discussed the conditions to enable closed-loop stability of a motion planning MPC that handles common time-varying scenarios in co-robot systems involving dynamic HRI. Under the listed assumptions, the predictor needed to detect the workers' movement mode change within a time delay allowance to guarantee such property. The MPC needed to have a sufficient planning horizon and a proper cost function. An example of an MPC with a closed-loop stability-enhanced predictor was presented. Satisfying the conditions, the proposed MPC with stability-enhanced prediction observed the environment and constructed proper state constraints for the optimization problem. This guaranteed the robot to have closed-loop stability theoretically when all assumptions were satisfied. The conditions also allowed the proposed method to avoid collisions even though the environment did not satisfy all the assumptions. To evaluate the proposed method's



(a) Experimental result when $m = working$.



(b) Experimental result right after m is switched back to $m = moving$.



(c) Experimental result when the worker leaves the scene.

Figure 6.7: Path planned and implemented in the third scenario.

performance in practice, a new notion, M-convergence, was used, and simulation results showed that a robot with the proposed method could deal with dynamic environments, had a better convergence property, and thus, resulted in a smooth closed-loop trajectory. The results showed that the cost was reduced sufficiently for both open-loop and closed-loop costs. Finally, experiments on Turtlebot3 showed that the proposed method performed well in time-varying environments.

Figure 6.8: Experimental result with mobile robot passing by another robot.

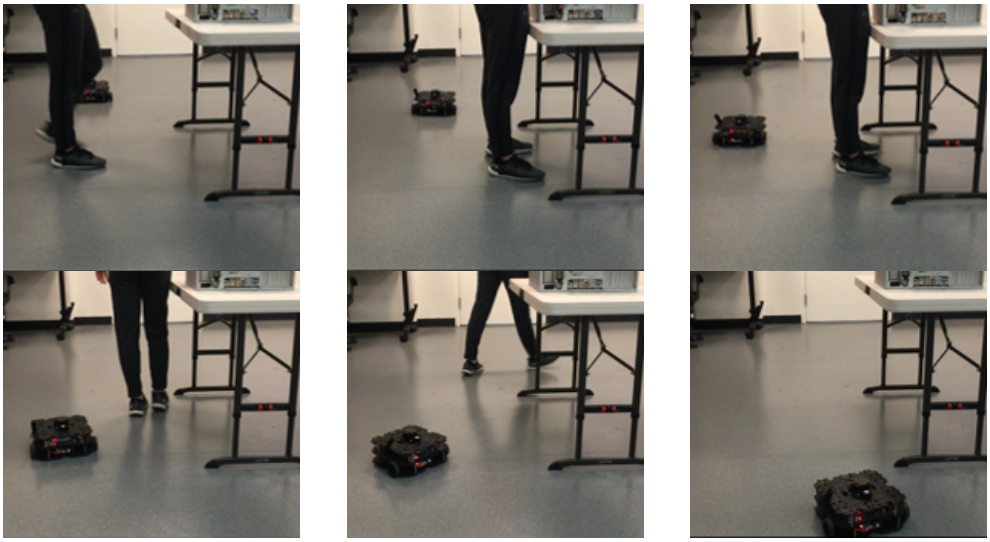


Figure 6.9: Experimental result with mobile robot passing by a human worker that walks away from the robot after leaving the working area.

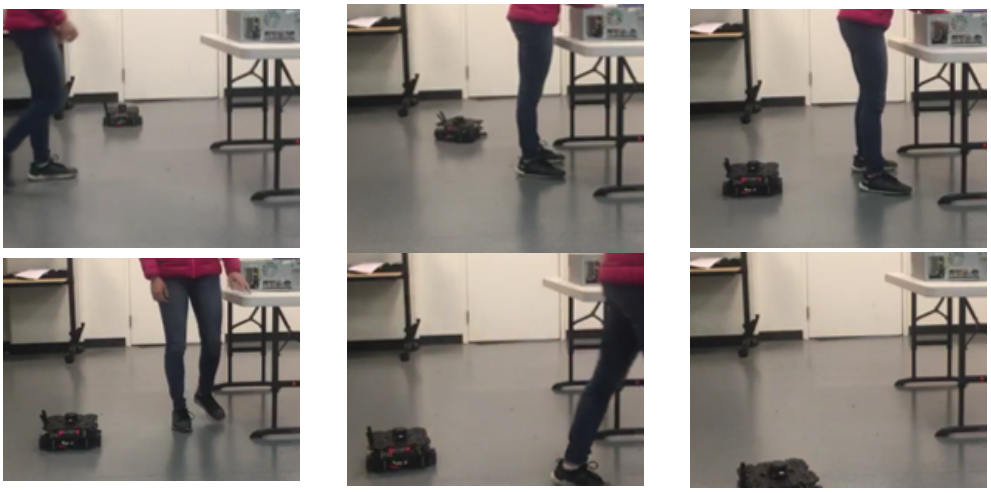


Figure 6.10: Experimental result with mobile robot passing by a human worker that walks along the same direction as the robot for a short period of time after leaving the working area.

Chapter 7

Application I: Human-Robot Collaboration for Assembly

7.1 Introduction

In previous chapters, we focused on robot "maneuvering" tasks, which can be easily defined by specifying the robot's goal. However, we also want to enable the robots to perform more complicated tasks or collaborate with human workers. Therefore, efficient, reactive, and robust task planning in dynamic environments is essential to developing intelligent industrial robots. For example, task planners for human-robot collaboration (HRC) systems in assembly lines [125, 41] need to decide in real-time how to assign different jobs to both human workers and robots to minimize task completion time. These HRC applications, however, raise three challenges for robotic systems: 1) predicting humans' actions and intentions [30, 98, 31], 2) taking humans' actions into account in the planning problem [28], and 3) designing a computationally efficient planner. While we mainly focus on motion planning with collision avoidance in the presence of *space-uncertainties* caused by the surrounding agents in previous chapters, here, we move to task planning and design a predictor to cope with the *time-uncertainty* and facilitate the task planner.

It is important to predict the duration of human actions in HRC environments to make a collaborative task plan. We use the sigma-lognormal function to model the human worker's movements [29]. According to the new observations, an online adaptation process adapts the human action model. This method enables us to predict the human trajectory and estimate the duration of the human's current and future actions. More importantly, we reckon that the adaptation process provides us with information to estimate the uncertainty of the duration prediction. Intuitively, the more the human action model needs to be adapted, the less we can trust the duration prediction before the adaptation converges to a new human action model. We propose a sampling process to estimate the human action duration uncertainty.

The goal of the human-aware task planning is to assign the robot action from all possible actions to minimize the collaborative costs, including factors such as completion time [55],

human fatigue [107], and spatial interfaces [55]. To plan efficiently, these planners require some task knowledge for constructing the task model. The two popular task models include flat models, such as a plan network [103], or hierarchical models, such as and/or graphs [82, 19]. Hierarchical models have shown superiority over flat models when used for predicting human actions and planning predictable actions for the robot. In this work, we adopt the sequential/parallel task model to facilitate task-level prediction and optimization-based planning [28]. This framework also allows us to incorporate human-induced uncertainties into the planning problem.

Previous works on HRC task planning mainly address uncertainties for safety considerations [75, 117]. However, one should not overlook the importance of accounting for human motion uncertainties for the time efficiency of the task plan. With the task model and the duration uncertainty estimation, we proposed a robust optimization-based formulation, in which the objective is to minimize the completion time of the planning horizon. This problem can be formulated as a mixed-integer linear programming problem, which can be solved efficiently. Simulations of a computer assembly task with different human behaviors are conducted to verify the effectiveness of the proposed robust task planner, and the performance is compared with that of the baseline planner. The key contributions of this work are:

- We propose a sampling-based estimator that utilizes information from the human action model adaptation process to estimate human action duration uncertainty.
- We formulate the robust task planning problem as a robust optimization problem.
- Simulations are conducted to verify the performance of the robust task planner.

7.2 Related Works

The goal of the online task planner considered in this work is to allocate actions to both robots and human workers. Many previous works address these problems by constructing either a tree [28, 36] or a graph [103, 75] and use search-based methods to find the best plan according to the given objectives. Human observations are often used to prune unpromising edges in the tree or graph for better planning efficiency and quality [36, 103, 53]. Some other works formulate the task planning problem as a multi-agent planning problem and apply reinforcement learning to learn a cooperative policy [84, 52]. While human motion is taken into account for safety and ensuring plan validation, to the best of the authors knowledge, previous works have not explicitly considered the effect of uncertainty in human action duration and effect on task completion time. Previously, most works treat the estimated human action duration, or human action completion time, of each individual action as a given constant [36, 53, 75]. However, the original plan based on the estimated time information may no longer be time-optimal due to the varying human action completion time during execution. Since the objective of most task planners is to minimize task completion time,

we claim that it is important to address the uncertainty in human action duration in the planning, which is the main focus of this work.

7.3 Preliminaries

7.3.1 Hierarchical task model

In this work, we target the computer assembly task [28]. Figure 7.1 shows an example of the sequential/parallel task model for the target task. The root node represents the *task*, the leaf nodes (colored in gray) are *actions*, and all the other nodes represents *subtasks*. The indicators below the root and the subtasks nodes indicates the relationship among their child nodes. The three types of relationships are:
Sequential nodes: their child nodes must be executed in the order from left to right, which is denoted by the operator $\rightarrow$. For example, the subtask *assemble main body* is a sequential node, and its child *install motherboard* must be done before *close hood*.
Parallel nodes: their child nodes can be executed in parallel, which is denoted by $\|$. For example, the subtask *install motherboard* is a parallel node, its children *install CPU fan*, *install memory*, and *install memory* can be executed simultaneously.
Independent nodes: their child nodes can be executed in any orders, which is denoted by $\perp$. Parallel nodes are special case of independent nodes. For example, root node is an independent node but not a parallel node. Its child nodes *applying labels to hood* and *assemble main body* have no fixed order, but they cannot be executed in parallel if *close hood* is in progress.

Following [28], actions can be defined as $a = [\{motion, object\}, attribute]$, where *motion* indicates the types of the movement, *object* indicates the object of interaction, and *attribute* contains information such as completion time and energy consumption, which are useful in the planning process.

7.3.2 Sigma-lognormal model

Sigma–lognormal model can explain most of the fundamental phenomena of human motor control [133] and can be used to model human motions in the assembly setting [29]. The human action model takes the following form:

$$\begin{aligned} \hat{\vec{v}}(t) &= \sum_{i=1}^{N} \hat{\vec{v}}_i(t) = \sum_{i=1}^{N} \vec{D}_i(t)\Lambda_i(t; t_{0i}, \mu_i, \sigma_i^2), \\ \Lambda_i(t; t_{0i}, \mu_i, \sigma_i^2) &= \frac{1}{\sigma_i\sqrt{2\pi}(t - t_{0i})} exp(\frac{-(\ln(t - t_{0i}) - \mu_i)^2}{2\sigma^2}), \end{aligned} \tag{7.1}$$

where $\hat{\vec{v}}(t)$ is the velocity of the human hand at time t, $\Lambda(t, t_0, \mu, \sigma^2)$ is a lognormal distribution with the time shift t_0, μ is the expected value of the t's natural logarithm, and

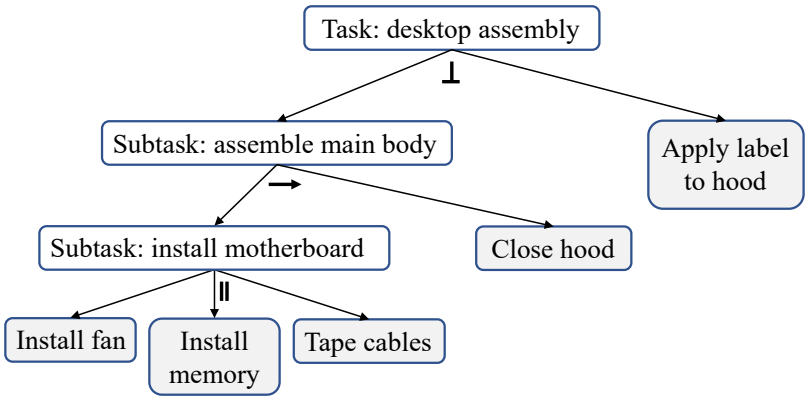


Figure 7.1: The sequential/parallel task model for a desktop assembly task.

σ is the standard deviation of the t's natural logarithm. The velocity $\hat{\vec{v}}(t)$ is composed of N lognormal distributions Λ_i, each scaled by variable $\vec{D}_i, i = 1, ..., N$. Each action in the task can be modeled by its own sigma-lognormal human action model. In this work, we use $N = 2$ to capture motions in 2D space, i.e., $i \in x, y$.

To obtain the parameters $\vec{\alpha}_i = \{\vec{D}_i, t_{0i}, \mu_i, \sigma_i^2\}, i = 1, ..., N$, of the nominal human action model, we collect training data of similar action motions and use the Levenberg–Marquardt algorithm [147] to solve the following problem: given a set of m data points $(t_{p,j}, \vec{v}_{p,j})$ from the p-th collected trajectory, find $\vec{\alpha}$ such that the sum of the squares of the deviations for all q collected trajectories $S(\vec{\alpha})$ is minimized:

$$\vec{\alpha}^* = \arg\min_{\vec{\alpha}} S(\vec{\alpha}) = \arg\min_{\vec{\alpha}} \sum_{p=1}^{q} \sum_{j=1}^{m} ||\vec{v}_{j,p} - \hat{\vec{v}}(t_{j,p}, \vec{\alpha})||_2^2. \tag{7.2}$$

7.3.3 Human action model adaptation

It is necessary to adapt the offline learned human action models to accommodate different human motion styles during action executions. Authors of [171] proposed to adapt the human action model by time scaling and shifting, i.e., modifying μ_i and t_{0i} in (7.1) through scaling factor $S_{t,i}$ and $s_{t,i}$. In addition, [29] proposed to scale D_i with $S_{D,i}$ to account for more variations.

$$\begin{aligned} t_{0s,i} &= S_{t,i} t_{0i} + s_{t,i}, \\ \mu_{s,i} &= \mu_i + ln(S_{t,i}), \\ D_{s,i} &= S_{D,i} D_i. \end{aligned} \tag{7.3}$$

Therefore, the sigma-lognormal human action model becomes $\hat{\vec{v}}(t; \vec{\alpha}^*, \vec{\beta})$, where $\vec{\beta}_i = \{S_{t,i}, s_{t,i}, S_{D,i}\}$, $i = 1, ..., N$. To adapt $\vec{\beta}$ when a new data point is available, the human

action model is first updated by using Levenberg–Marquardt algorithm that minimizes the prediction error, which gives the update rule:

$$\vec{\beta}_i = \vec{\beta}_i - (\hat{v}_{i,t_k} - v_{i,t_k})\nabla_{\vec{\beta}_i}\hat{v}_{i,t_k}./((\nabla_{\vec{\beta}_i}\hat{v}_{i,t_k})^{.2} + \vec{\lambda}_i),\ i = 1, \ldots, N, \tag{7.4}$$

where v_{i,t_k} is the measurement and $\hat{v}_{i,t_k}$ is the model predictions on either the x or the y direction at time t_k. $\nabla_{\vec{\beta}_i}\hat{v}_{x,t_k}$ is the gradients of $\hat{v}_{x,t_k}$ with respect to $\vec{\beta}_i$, and $\vec{\lambda}_i$ is the non-negative damping factors. Note that the operator "." indicates the element-wise operation. After this model update process, the zero-crossing time will be used as the estimated final time $\hat{t}_f$, also called the human action duration or the human action completion time. Second, to take advantage of the scene information, $\vec{\beta}$ is updated to minimize an objective function with three terms: 1) the difference between the current distance to the goal and the predicted travel distance to the goal; 2) the prediction error at the current time t_k; 3) velocity at the final time $\hat{t}_f$. Therefore, to optimize the objective functions, the two ends of the future velocity profile are fixed, and the velocities are modified in between to best fit the scene information. The objective function is a weighted sum of the three terms: $K(\hat{t}_f, \vec{\beta}) = \gamma_1 J_1(\hat{t}_f, \vec{\beta}) + \gamma_2 J_2(\vec{\beta}) + \gamma_3 J_3(\hat{t}_f, \vec{\beta})$, and the update rule is:

$$\vec{\beta}_i = \vec{\beta}_i + K(\hat{t}_f, \vec{\beta})\nabla_{\beta_i}K./((\nabla_{\beta_i}K)^{.2} + \vec{\lambda}'_i),\ i = 1, \ldots, N. \tag{7.5}$$

The parameter $\vec{\beta}$ can be viewed as the characteristics of the human worker, such as the worker's tendency to execute every action quickly or slowly. Assuming that the worker carries the same characteristics when performing all the actions, we can directly apply $\vec{\beta}$ from one model to scale the other models for a potentially more accurate prediction of those action completion times, as suggested in [29]. In this work, we proposed a sampling based method that utilizes the adaptation terms

$$\begin{aligned} \delta_{\hat{v},i} &= (\hat{v}_{i,t_k} - v_{i,t_k})\nabla_{\vec{\beta}_i}\hat{v}_{i,t_k}./((\nabla_{\vec{\beta}_i}\hat{v}_{i,t_k})^{.2} + \vec{\lambda}_i),\ i = 1, \ldots, N, \\ \delta_{K,i} &= K(\hat{t}_f, \vec{\beta})\nabla_{\beta_i}K./((\nabla_{\beta_i}K)^{.2} + \vec{\lambda}'_i),\ i = 1, \ldots, N, \end{aligned} \tag{7.6}$$

to estimate the human-induced uncertainty.

7.4 Robust Task Planning for HRC Applications

7.4.1 Baseline problem formulation

The human-aware robot task planning problem considers action allocation problems for a robot and a human worker in this work. Notice that the proposed formulation and methods can be easily extended to the multi-robot case. The number of actions considered in one planning problem is determined by the number of the remaining actions parallel to the human's current action. Thus, the planning horizon k is part of the remaining actions, and it varies as the task proceeds. During run-time, the planner identifies the human's

current action, then finds the parallel actions according to the hierarchical task model. The planning problem can then be formulated for assigning those parallel actions to the human and the robot such that the completion time for the planning horizon t is minimized. Denote the k-dimensional action assignment vectors $x_r, x_h \in \{1,0\}^k$ for the robot and the human, respectively, where the entries are either 0 (not assigned) or 1 (assigned). For example, $x_r = [1,0,1]^\top$ and $x_h = [0,1,0]^\top$ together means that there are three actions in the planning horizon. The first and third actions are assigned to the robot while the second action is assigned to the human worker. The inputs of the algorithm are T, C, C_r, C_h, t_0, t_r, and t_h, where T is the sequential/parallel task model, $C = \{1, \ldots, k\}$ is the set of action indices, C_h and C_r are the action indices that human worker and the robot are capable of executing, respectively, $t_o \in \mathbb{Z}_+$ is the remaining time before the human's current action is completed, and $t_h, t_r \in \mathbb{Z}_+^k$ are the empirical completion time of a human and a robot for each action, respectively. If t_r, t_0, and t_h are time invariant, the optimization problem is formulated as follows:

$$\begin{aligned} \min_{x_h, x_r, t} \quad & t, \\ \text{s.t.} \quad & x_h^\top t_h + t_o \le t, \quad x_r^\top t_r \le t, \\ & x_h + x_r = \mathbf{1}, \quad x_{h,i}, x_{r,i} \in \{0,1\}, \quad i = 1, \ldots, k, \\ & x_{h,\{C-C_h\}} = \mathbf{0}, \quad x_{r,\{C-C_r\}} = \mathbf{0}. \end{aligned} \tag{7.7}$$

Decision variables x_h and $x_r \in \{0,1\}^k$ are binary vectors for the assignment of actions, where k is the number of actions in the planning horizon. The objective function t is the completion time for the planning horizon, which is the upper bound of the human's completion time and the robot's completion time. $\{C - C_h\}$ and $\{C - C_r\}$ denote the indices of the actions that the human and the robot are unable to do, respectively. During run-time, the planner optimizes the planning problem, sends the selected action with the shortest completion time among all actions assigned to the robot to the controller for execution, then plans repeatedly.

The optimality of the plan holds only when t_0 and t_h are the true action completion time. However, this is seldom the case. Since action completion time of a group of workers often stay in a range, one can use the time upper bound for solving all planning problems. However, the planner could be unnecessarily conservative while still cannot account for abnormal human behaviors, leading to a task completion time far longer than the time truly needed. Therefore, we propose to include such human uncertainty into the planning problem by extracting the human uncertainty information during the human action model adaptation process, then incorporating the completion time uncertainty into the planning problem.

7.4.2 Task planning with human-induced uncertainties

Human-induced uncertainties affect HRC systems mainly from the safety and time efficiency aspects. While many works have addressed these uncertainties for safety [30, 98, 97], few have attempted to address the impacts on time efficiency caused by these uncertainties. To address this problem, we first reckon that human-induced uncertainty mainly affects t_0

and t_h. In this light, the task planning problem should be rewritten as:

$$\begin{aligned} \min_{x_h, x_r, t} \quad & t, \\ \text{s.t.} \quad & x_h^\top t_h + t_o \leq t, \quad x_r^\top t_r \leq t, \\ & x_h + x_r = \mathbf{1}, \quad x_{h,i}, x_{r,i} \in \{0,1\}, \quad i = 1, \ldots, k, \\ & x_{h,\{C-C_h\}} = \mathbf{0}, \quad x_{r,\{C-C_r\}} = \mathbf{0}, \\ & t_0 = \bar{t}_0 + u_0, \quad u_0 \in \mathcal{U}_0, \\ & t_h = \bar{t}_h + u_1, \quad u_1 \in \mathcal{U}_1, \end{aligned} \tag{7.8}$$

where $\bar{t}_0$ and $\bar{t}_h$ are the originally proposed action completion time; u_o and u_1 are added to account for the uncertainty on human action completion time; $\mathcal{U}_0$ and $\mathcal{U}_1$ are the uncertainty sets. A more solvable form of the problem is needed to solve this optimization problem efficiently. Two questions are to be addressed:

1. To what extent should the plan be immune to the uncertainty?
2. What form should $\mathcal{U}_0$ and $\mathcal{U}_1$ take to model the action completion time uncertainties?

To answer the first question, as mentioned in the literature [12], it is sufficient to find a solution plan that is immune to all disturbances from $\mathcal{U}$ so that it is also immune to "nearly" all real-world disturbances, i.e., up to $(1-\epsilon)$ of the total probability mass, where ϵ is a small number. This can be achieved by finding a solution of a chance constraint problem, in which we need to choose $\mathcal{U}$ as a computationally tractable convex set that "$(1-\epsilon)$-supports" all real-world disturbances. In other words, we answer the first question by formulating the first constraint as a chance constrain, i.e., $Prob(x_h^\top t_h + t_o \leq t) \geq (1-\epsilon)$ and choose a small $\epsilon = 0.005$, so that we can say the solution plan guarantees that the completion time for the planning horizon will be smaller or equal to t for 99.5% of the time. The second question can be answered by following the robust optimization formulation described in [12], which requires an estimate of the human-induced uncertainty to construct the uncertainty set.

7.4.3 Human-induced uncertainty estimation

This section proposed a sampling-based method to estimate human-induced uncertainty. While human-induced uncertainty may be an abstract concept, we can focus on the "correctness" of the current human action model and use it to construct the uncertainty set. The basic idea behind the proposed method is that the less correct the action model is, the more adaptation it needs since it deviates from the observation and the task scene. Therefore, the less we should trust the human action model and the initially proposed action completion time $\bar{t}_0$ and $\bar{t}_h$. In other words, larger uncertainty sets should be applied to $\bar{t}_0$ and $\bar{t}_h$ when the action model is adapting. Recall that in section 7.3.3, a parameter update process based on adaptation information $\delta_{\hat{v},i}$ and $\delta_{K,i}$ is introduced. We propose using this adaptation information to model the "human action model parameter uncertainty," where we assume

that the true action model parameter lies in a parameter distribution $\vec{\beta}_i \sim \mathcal{N}(\hat{\vec{\beta}}_i, C)$. Here, $\hat{\vec{\beta}}_i$ is the current action model parameter and $C = diag(w^\top abs(\delta_{\hat{v},i} + \delta_{K,i}))$ where w is a weighting vector.

To estimate the action completion time, we draw n samples of $\vec{\beta}_i$ from the distribution $\vec{\beta}_i \sim \mathcal{N}(\hat{\vec{\beta}}_i, C)$ and calculate n human completion time for each action according to these sampled parameters. This gives us a distribution of the predicted completion time and allows us to calculate the standard deviation σ_{t_0} and σ_{t_h} of the sampled completion time. We assume that the human action completion time can be described by a normal distribution, i.e., $t_0 \sim \mathcal{N}(\bar{t}_0, \sigma_{t_0})$ and $t_h \sim \mathcal{N}(\bar{t}_h, \sigma_{t_h})$. Since the 3-sigma bound covers most uncertainty mass, it is acceptable to model t_0 and t_h as random variables taking values in the range $[\bar{t}_0 - 3\sigma_{t_0}, \bar{t}_0 + 3\sigma_{t_0}]$ and $[\bar{t}_h - 3\sigma_{t_h}, \bar{t}_h + 3\sigma_{t_h}]$, respectively.

To construct the uncertainty set, we first considered the budget uncertainty [12]:

$$\mathcal{Z} = \{\xi \in \mathbb{R}^L : -1 \leq \xi_l \leq 1,\ l = 1, \ldots, L,\ \sum_{l=1}^{L} |\xi_l| \leq \gamma\}, \tag{7.9}$$

where $\gamma = \sqrt{2ln(1/\epsilon)L}$. Notice that L in this work equals to one plus the number of parallel actions that the human worker can execute, i.e., $L = 1 + |C_h|$ where $|\cdot|$ denotes the cardinality of a set. Let $\sigma_0 = 3\sigma_{t_0}$ and $\sigma_h = 3\sigma_{t_h}$, we can rewrite the human action completion time t_0 and t_h in (7.8) as $t_0 = \bar{t}_0 + \sigma_0\xi_1$ and $t_{h,i} = \bar{t}_{h,i} + \sigma_{h,i}\xi_{i+1}, i = 1, \ldots, k$, respectively, where $\xi \in \mathcal{Z}$.

7.4.4 Robust task planning

With the new representation of the human action completion time, we can formulate the following robust optimization problem [12] by introducing additional variables $z, w \in \mathbb{R}^L$.

$$\begin{aligned}
\min_{x_h, x_r, t, z, w} \quad & t, \\
\text{s.t.} \quad & x_r^\top t_r \leq t, \\
& \sum_{l}^{L} |z_l| + \gamma \max_l |w_l| + x_h^\top \bar{t}_h + \bar{t}_0 \leq t, \\
& z_1 + w_1 = \sigma_0, \\
& z_{i+1} + w_{i+1} = -\sigma_{h,i} x_{h,i}, \qquad i = 1, \ldots, |C_h|, \\
& x_h + x_r = \mathbf{1}, \quad x_{h,i}, x_{r,i} \in \{0, 1\}, \qquad i = 1, \ldots, k, \\
& x_{h,\{C - C_h\}} = \mathbf{0}, \quad x_{r,\{C - C_r\}} = \mathbf{0}.
\end{aligned} \tag{7.10}$$

Notice that this optimization problem is a mixed-integer linear programming problem that can be solved efficiently by commercial solvers, e.g., `intlinprog` from `MATLAB`, which gives the proposed method a computational advantage. As opposed to search-based task planning

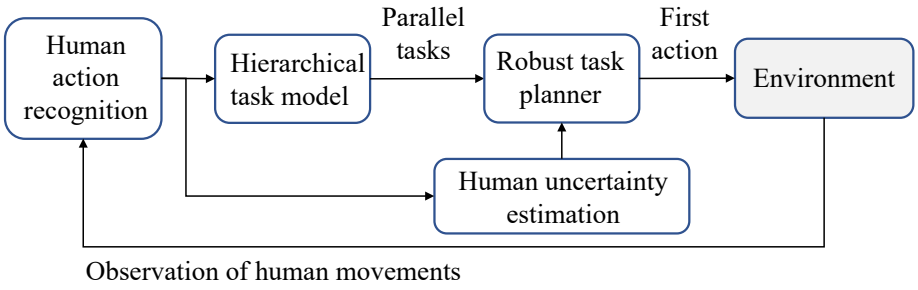


Figure 7.2: The overall system control design.

methods, the optimization formulation allows the solver to approximate the problem as a standard linear programming (LP) problem and only resort to enumerating all possible combinations when the approximation fails. During the simulation experiments, the solver can always solve the problem by solving the approximated LP problems in 0.01 seconds on average, without enumerating all possible combinations. The computational efficiency shows another advantage of such a robust task planning formulation.

The overall system is shown in Figure 7.2. During run-time, the robot first collects human motion observations and then conducts human action recognition. Given the human's current action, the hierarchical task model finds the parallel actions and sends them to the robust task planner. On the other hand, the human uncertainty estimation module, containing the off-line trained human action models, monitors the human uncertainty and updates its models as well as σ_0 and σ_h in the planner. After the plan is generated, the action that has the shortest completion time among all actions assigned to the robot will be executed. This process repeats every time the robot completes its action until there is no remaining action that the robot can execute.

7.5 Applications

The simulation scenario in this work is similar to that of a human and a robot collaborating on a computer assembly task. We test the proposed algorithm on a simulator built-in `MATLAB`. Human subjects manipulate the objects using the computer mouse to drag and release the objects. Complex actions such as inserting and wrapping in the desktop assembly scenario are simplified by the release. As shown in Figure 7.3(left), five actions are needed to move the fan, the memory, the tape, the label, and the hood to their designated areas. The capability of both the human and the robot is set to the whole action space except that only the human can close the hood. The simulation is conducted in `Matlab R2020a` on a desktop with a 3.2GHz Intel Core i7-8700 CPU. Three case studies with different human worker behaviors are presented in the following sections. Two task planners are considered, the proposed robust task planner and the baseline task planner that treats human action

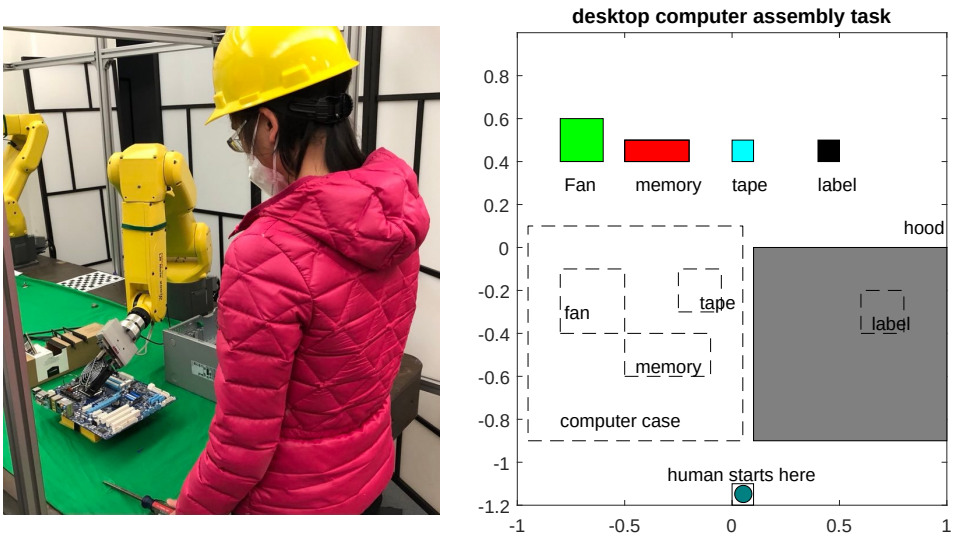


Figure 7.3: An illustration of a human worker and a robot performing an assembly task (left) and the simulation setup for the computer assembly. The areas enclosed by the dashed lines are the designated areas for the objects. The mouse controlled by the human is indicated by the green circle (right).

Table 7.1: Task completion time for each case.

Planner	Case 1	Case 1	Case 2
Robust task planner	7.88 [s]	16.81 [s]	17.06 [s]
Baseline planner	7.94 [s]	21.52 [s]	19.52 [s]

completion time as constants [28]. During run-time, the task operation time is shown on the top right. The indicator on top of each object indicates the planner's assignment for the object, where "h" means that the associated action is assigned to the human worker and "r" indicates that the associated action is assigned to the robot (Figures 7.4-7.6). The task completion time for each case is shown in Table 7.1.

7.5.1 Case 1: collaborating with an efficient human worker

The word "efficient" means that the worker performs each action using the amount of time similar to the prior knowledge that the planner was given. As shown in Figure 7.4, once the planner detects the human's current action, the planning is conducted using the given human completion time. The resulting plan lets the human worker place the memory and the tape while assigning the robot to place the label. After the robot finishes putting the label, the planner replans and realizes that the robot should move the tape since the

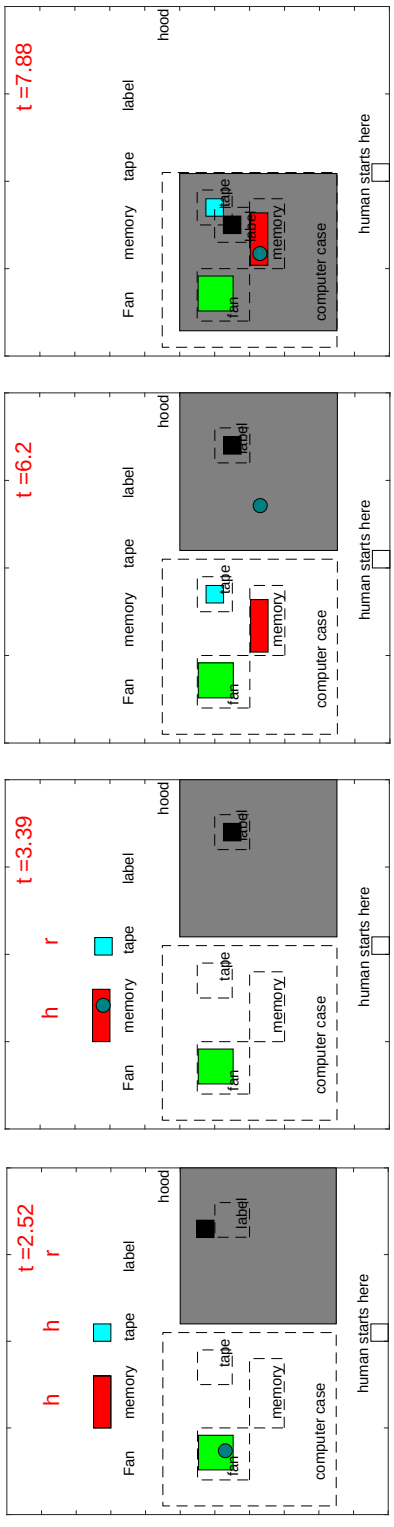


Figure 7.4: Experimental result of the robot with the baseline planner collaborating with an efficient human worker. (Case 1.)

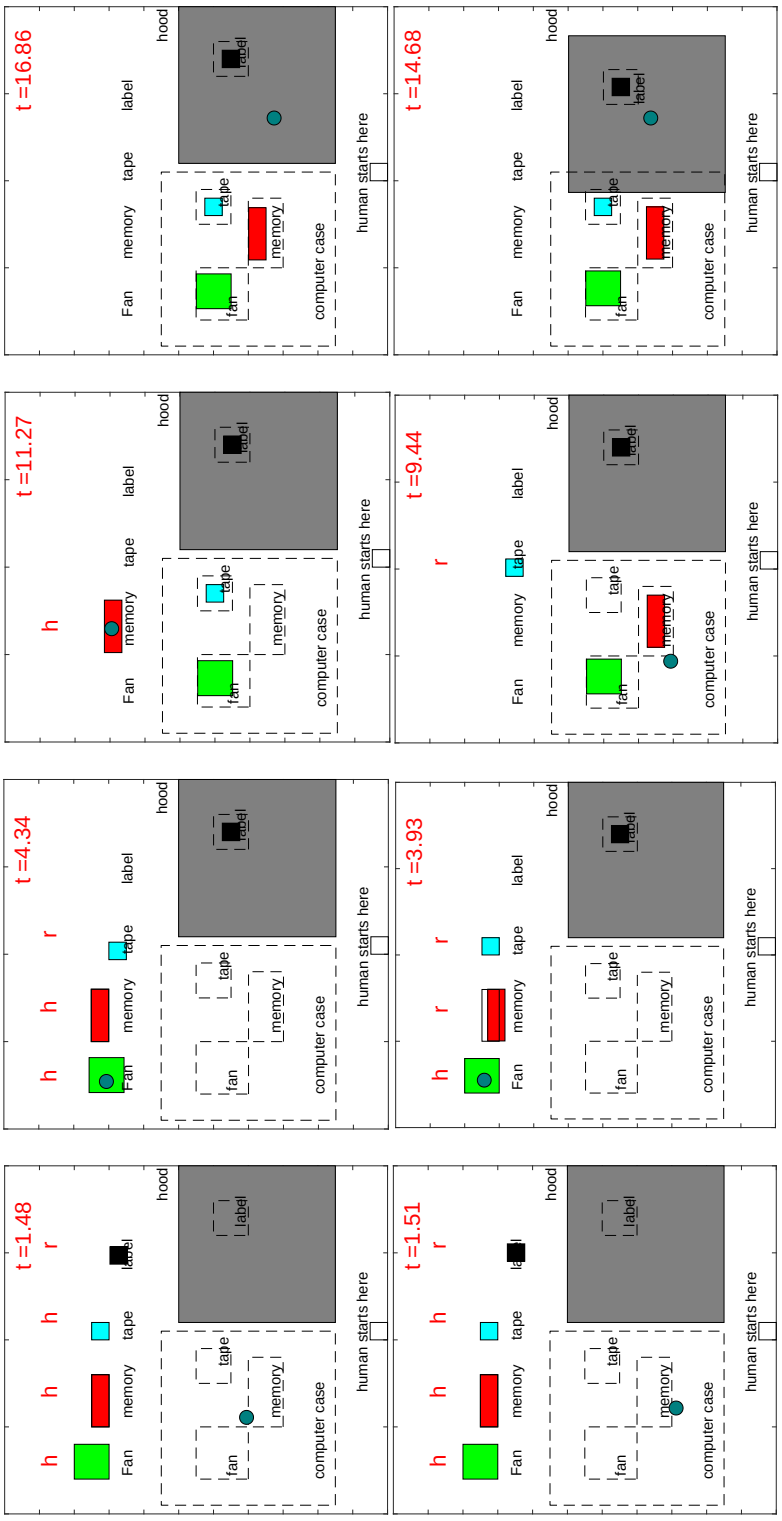


Figure 7.5: Experimental result of the robot with the baseline planner (upper row) and the robust task planner (lower row) collaborating with a lazy human worker. (Case 2.)

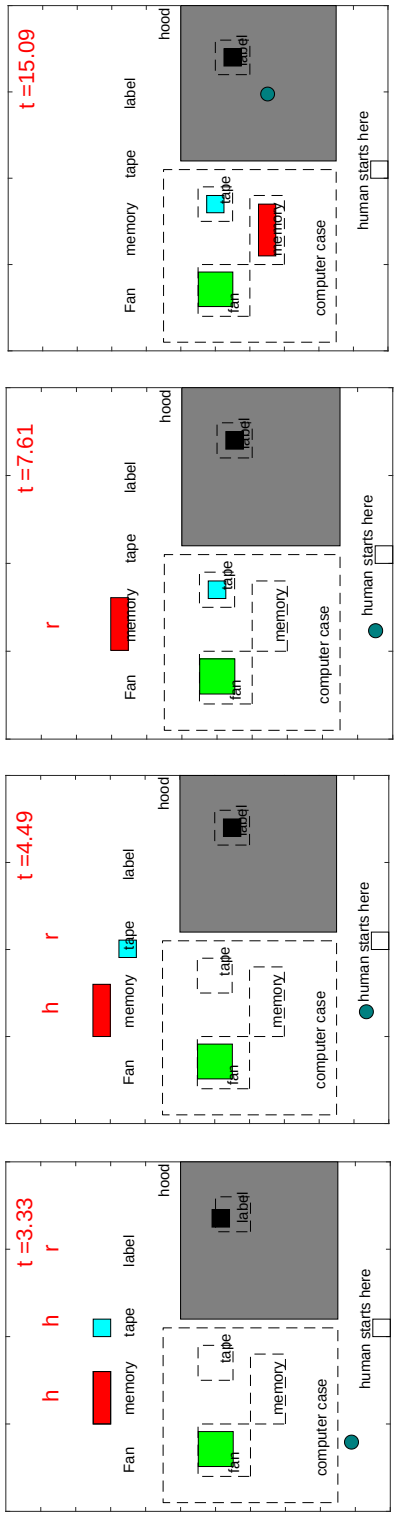


Figure 7.6: Experimental results of the robot with the robust task planner collaborating with a slacking human worker. (Case 3.)

human has just started to move the memory. In the end, the human worker closes the hood and finishes the task. We can see that the baseline planner collaborated with the worker efficiently. Similar behaviors can be seen when the robot runs the proposed robust task planner.

7.5.2 Case 2: collaborating with a lazy human worker

The word "lazy" means that the worker performs each action using 2x to 3x the amount of time compared to the prior knowledge that the planner was given. The upper row in Figure 7.5 shows the result of the robot running with the baseline planner. Since the completion time is treated as a constant, the planner continues to assign the memory to the human because it believes the human worker can complete that task quicker than the robot. This causes the robot to idle while the human slowly completes the current action, then completes the memory placement. On the other hand, the proposed method detects the deviation of the human movements from the model; therefore, it replans and assigns both the tape and the memory to the robot, as shown in the lower row in Figure 7.5. This indicates that the proposed planner can account for human-induced uncertainties and, as a result, reduce the damage of these uncertainties in terms of task completion time.

7.5.3 Case 3: collaborating with a slacking human worker

The word "slacking" means that the worker stops doing the task for some time in the middle. The worker can either be an efficient worker or a lazy worker. As shown Figure 7.6, the mouse idles around the bottom area of the interaction window for sometime ($t = 4$ to 13s). When using the baseline planner, similar to case 2, the planner continues to assign the memory to the human worker after the fan, tape, and label placements are completed. With the proposed planner, the robot will complete the memory placement instead to achieve a shorter execution time. Figure 7.6 shows the result where the human worker works "efficiently" most of the time but slacks off sometimes during the task. Since the human moves fast (completes the fan placement in 3 seconds), the planner assigns memory placement to the human when the robot is placing the tape. However, later on, the planner assigns this action to the robot because the human worker is idling and causes larger human uncertainties. This demonstrates the ability of the proposed task planner to handle abnormal human behaviors.

In summary, the proposed robust task planner can take in the human uncertainty estimation and replan accordingly, resulting in a shorter task completion time when human action deviates from the prior knowledge.

7.6 Chapter Summary

This chapter presented a robust task planner for assembly lines with human-robot interaction. Based on the adaptation process of the off-line-learned human action models, we sampled the model parameters and estimated the distribution of the human action completion time to quantify the human-induced uncertainty. The robust task planner took in this information alongside the hierarchical task model and solved the task planning problem as a robust optimization problem. Simulation results showed that the proposed robust task planner could handle the human uncertainty and replan accordingly, resulting in a shorter task completion time than the baseline planner when human action deviated from prior knowledge.

Chapter 8

Application II: Autonomous Parking in Uncertain Environments

8.1 Introduction

Aside from industrial applications, autonomous cars also demand integrated strategies to drive on roads or perform parking while reacting to the environment. Fully autonomous parking [40, 121] remains challenging, especially in a dynamic and uncertain environment with multiple independent agents. A parking task requires autonomous vehicles (AVs) to plan in a tight space and intelligently react to the surrounding vehicles, i.e., obstacle vehicles (OVs). In contrast to driving on roads or highways, vehicle motions in parking areas do not have a clear set of rules to follow and largely depend on the driver's intention or even skill level. Hence an autonomous parking system that integrates prediction and planning is possibly necessary. In this chapter, we develop an integrated strategy and demo the modularized robotic system with autonomous parking in dynamic.

Motion prediction is crucial because it determines the safety constraints of the planning modules and thus the feasibility and smoothness of the motion plan [98]. In particular, an accurate short-term motion prediction enables the AV to plan and react safely to the OVs, whereas long-term plan/mode prediction allows the AV to plan more efficiently and smoothly. This work proposes a model-based hybrid predictor to perform both short-term motion and long-term mode predictions by observing the poses of OVs. A major challenge in short-term prediction is to estimate the OV's steering angle. We use an extended Kalman filter (EKF) to reconstruct OV's velocity and then resort to an adaptive observer for the steering estimation. Despite the difficulty in predicting exact long-term motions, we observe that a driver generally follows some routes due to driving conventions (e.g., cars should stay on their left-hand side in Japan). Also, vehicles' motion throughout parking/leaving can be captured by several "modes" (e.g., maneuvering into/out of tight spaces and cruising on aisles). Based on these two priors, we use a cost-map [49, 67] to capture these routes, combine the short-term predictions to determine OVs' modes, and make long-term predictions.

Motion planning for AVs is another challenge in parking scenarios. General motion planning algorithms [48, 119, 116, 71, 2, 27] are not directly suitable for parking in the presence of OVs, which requires rapid replanning for complicated driving maneuvers. On the other hand, motion planners specializing in autonomous parking either fail to integrate short-horizon planning with long-horizon planning or cannot incorporate online path repairing upon new obstacles in uncertain environments [68, 88, 61, 121, 81, 160].

A scenario-aware planner that implements multiple strategies can be effective in computation time, leading to a high replanning rate, and possibly safety guarantees. In this work, we first generate a long-term motion reference with Bi-Directional A-Search Guided Tree (BIAGT) [169]. Then, a strategic motion planner, based on the results of the hybrid environment predictor, implements three strategies: 1) model predictive control (MPC)-based safety controller [138] for trajectory tracking if the reference remains valid regarding the environment, 2) search-based retreat-planning that quickly finds an evasion path in an emergency, and 3) optimization-based repair-planning when the reference is invalidated.

This chapter presents an integrated system that combines the hybrid environment predictor and the strategic motion planner for autonomous vehicle parking in dynamic and uncertain environments. Simulation evaluation of the proposed system on a variety of parking tasks demonstrates its advantages in terms of initial planning, motion prediction, safe tracking, retreating in an emergency, and trajectory repairing. Main contributions are threefold as follows:

- A model-based hybrid environment predictor predicts short-term motions and long-term modes.
- A strategic motion planner is presented to plan under different situations efficiently.
- Simulation is performed to show the effectiveness of the proposed system (video is publicly available here).

8.2 Related Works

8.2.1 Predictor

Research in vehicle motion prediction has attracted a lot of interests, and results in numerous contributions, e.g., short-term motion prediction methods [144, 66, 6] and long-term plans/modes prediction methods [66, 6, 149, 143, 150, 44, 92] with different types of observation (e.g. car pose, RGBD-camera, Lidar) were proposed. It is arguably true that most research in this area is related to road driving. Interested readers are referred to an extensive survey in [96]. In contrast, vehicle motion prediction in parking is less explored. In [74], an interacting multiple model (IMM) filter is used to predict short-term trajectories in parking. Focusing on long-term prediction, [57] first trains a trajectory cluster classifier, and then acquires the mean-value trajectory of the classified cluster. In [150], the

classified driver's intent and the vehicle's pose history are used to generate short-term motion predictions with a Long Short-Term Memory network. Purely data driven methods are not desirable for two reasons: 1) the lack of guarantees; 2) their performance depends largely on the training data set, and they may present a larger prediction error if the data set is chosen poorly. Prediction network over fitting may also be a concern. To the best of our knowledge, there are not extensive studies on a predictor fusing both short-term and long-term predictions for parking.

8.2.2 Planning

Prevailing motion planning approaches fall into three categories: search-based [63, 156, 109], sampling-based [79, 95, 78] and optimization-based [177, 146, 59, 175]. Sampling-based planners could raise concerns in risk-sensitive tasks due to their non-deterministic nature, while optimization-based planners are only locally optimal and often need to work with global planners [176, 172, 43, 175, 101]. Various search-based motion planners are widely adopted by autonomous vehicles for their computation efficiency with well-chosen motion primitives and heuristics [165, 48, 122, 47, 23, 3, 24, 80].

If the parking environment changes, the initial long-term trajectory may need to be repaired. Real-time trajectory repairing methods include online heuristic update [83, 158, 128], pruning and reconnecting sampling-based search structures [50, 22, 1, 134], and spline-based kinodynamic search [46, 45]. The heuristic update method is not directly applicable to the tree-based search structure in BIAGT, and pruning is less efficient for parking scenarios. Spline-based kinodynamic search is relatively efficient, but the original trajectory is not utilized. On the other hand, we observe that alternative feasible solutions in the repairing scenarios are often in the same homotopy class as the original trajectory. Therefore, optimization-based methods [174, 146, 46] become suitable candidates for repairing an existing path.

8.2.3 Parking system

As for system-level strategies for AV parking, [174] and [73] present autonomous parking systems to park in static environments. In [74], IMM is used for prediction with a sampling-based method for planning. The method in [149] first predicts the strategy of OVs and then selects the navigation strategy of the ego AV. However, the AV only traverses the roads of the parking lot but does not perform parking maneuvers. Instead, this work aims at the more complete approach of an integrated parking system that makes short-term and long-term predictions of the environment and utilizes them in the planning for parking.

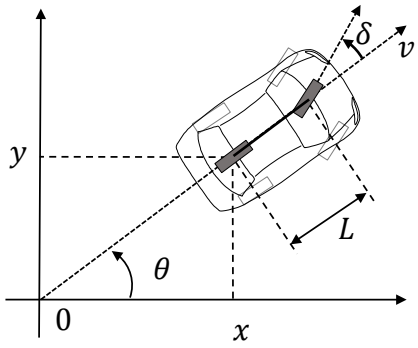


Figure 8.1: The simplified bicycle vehicle model. L is the distance between the axis of the rear wheels and the axis of the front wheels.

8.3 Problem Statement and Proposed Architecture

Consider the planning problem with vehicle dynamics:

$$\dot{X} = f(X) + g(X, u), \tag{8.1}$$

where $X = [x, y, \theta]^\top$ denotes the 2D coordinates and the vehicle heading, and $u = [\delta, v]^\top$ is the control input that includes longitudinal velocity and steering angle. A collision-free configuration space $\mathcal{C}_{free} \subset \mathbb{R}^{n_c}$ is the set of configurations at which the vehicle has no intersection with the obstacles. The motion planning problem considered in this chapter is the same as Problem 3.

We use the bicycle model, illustrated by Figure 8.1, to represent the vehicle motion. The discrete-time model is obtained through Euler discretization as follows:

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ \theta_{k+1} \end{bmatrix} = \begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix} + \begin{bmatrix} v_k T_s \cos(\theta_k) \\ v_k T_s \sin(\theta_k) \\ v_k T_s \frac{\tan \delta_k}{L} \end{bmatrix}, \tag{8.2}$$

where T_s is the sampling time.

Figure 8.2 shows the architecture of the proposed system. The two main modules are the hybrid environment predictor (Section 8.4) and the strategic motion planner (Section 8.5). The information generated by the predictor is designed to better facilitate the planning module, which receives this information and conducts planning. During run-time, the central control first processes the parking lot map M_{map} and generates an initial long-term trajectory $\mathcal{P}_{ref}$. We use BIAGT since it is guaranteed to plan a trajectory that brings the ego AV precisely to the goal, a vital feature for parking in a tight space. Note that the planners presented in Chapters 3 and 4 can also provide such an initial long-term trajectory. The hybrid environment predictor monitors the environment and predicts the movements of the OVs. The strategic motion planner first checks if the ego AV is violating the safety margin based on the prediction. If not, it checks if $\mathcal{P}_{ref}$ needs to be repaired due to the OV. If any of

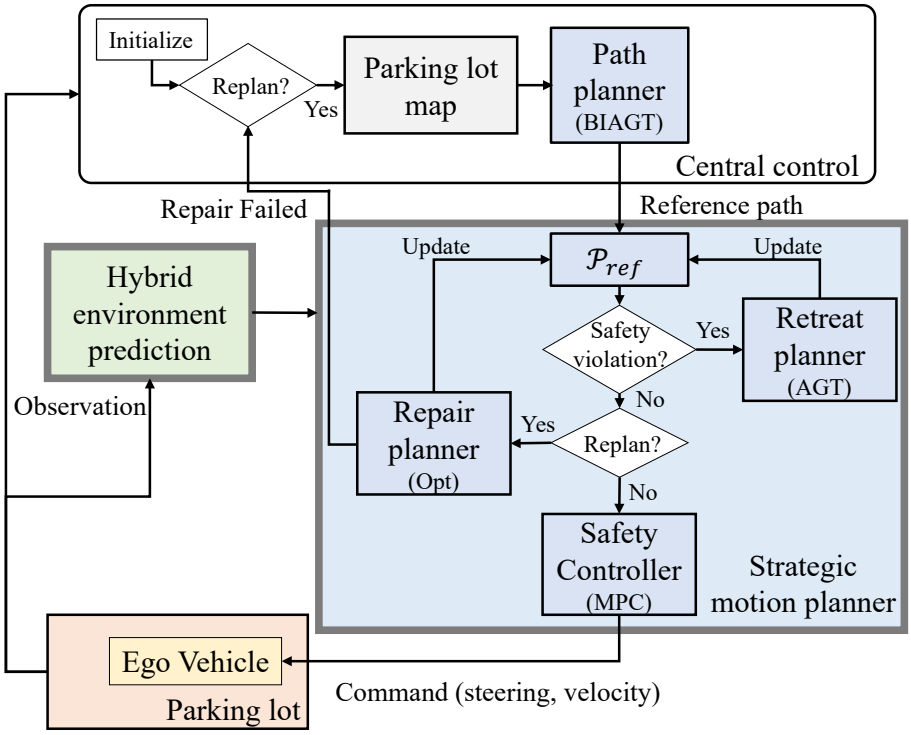


Figure 8.2: The integrated prediction and planning system.

these situations occur, $\mathcal{P}_{ref}$ will be updated. Finally, an MPC-based safety controller plans a collision-free motion that tracks the latest $\mathcal{P}_{ref}$ in the uncertain environment. If the repair planner cannot succeed, it requests the central control to update the map and regenerate a reference trajectory.

8.4 Prediction in Dynamic Parking Environments

The hybrid environment predictor (illustrated in Figure 8.3 and summarized in Algorithm 6) contains three main parts: motion estimation, motion prediction, and mode estimation of OVs.

8.4.1 Cascaded motion estimation

Many previous works studied motion estimation[150, 74], which reconstructed the state X from the measurement of (x, y) based on the unicycle model. Such a treatment is insufficient for vehicle parking, where vehicles frequently change their moving direction and steering actions. To accurately predict the short-term motion of the OV, it is advantageous to reconstruct the control input u. One can either pose it as an unknown input estimation

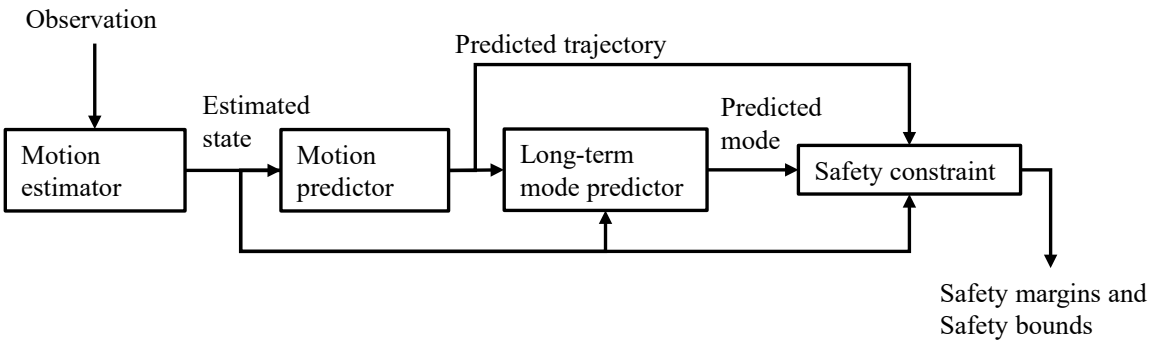


Figure 8.3: The architecture of the hybrid environment predictor.

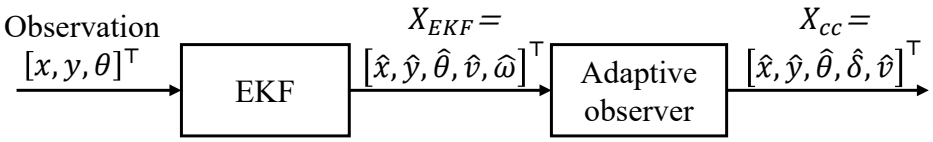


Figure 8.4: The architecture of the cascade motion estimator.

problem [65] or augment the OV's system state with the control input and solve a state estimation problem. Assume that the OV motion evolves according to the model (8.2). We obtain the augmented model of the OV by assuming that control input (δ, v) are piecewise constant, and estimate the augment state $[x, y, \theta, \delta, v]^\top$.

Given the nonlinear augmented model, it is natural to apply well-established nonlinear state estimators such as EKF or particle filter for state estimation. We observe that it is not straightforward to tune EKF to estimate the steering angle accurately due to the term $v\tan(\delta)$, which involves the multiplication of unmeasured states. Meanwhile, the heavy computation presents a hurdle for adopting particle filter. However, estimating steering is crucial to short-term and long-term predictions. Therefore, a cascaded motion estimator (Figure 8.4) is proposed to estimate the OV motion. The velocity estimation resorts to EKF and is based on the following discrete-time model:

$$\begin{aligned} \begin{bmatrix} x_{k+1} \\ y_{k+1} \\ \theta_{k+1} \\ v_{k+1} \\ \omega_{k+1} \end{bmatrix} &= \begin{bmatrix} x_k + v_k T_s \cos(\theta_k) \\ y_k + v_k T_s \sin(\theta_k) \\ \theta_k + T_s \omega_k \\ v_k \\ \omega_k \end{bmatrix} + q_k,\ q_k \sim \mathcal{N}(0, Q), \\ z_k &= X_{OV,k} + r_k, \end{aligned} \tag{8.3}$$

where ω is the yaw rate, q is the disturbance, $z_k = [z_{x,k}, z_{y,k}, z_{\theta,k}]^\top$ is the measurement, $X_{OV,k} = [x_k, y_k, \theta_k]^\top$, and $r_k \sim \mathcal{N}(0, R)$. The EKF outputs $X_{EKF,k} = [\hat{x}_k, \hat{y}_k, \hat{\theta}_k, \hat{v}_k, \hat{\omega}_k]^\top$.

The estimation of the steering action is based on the following model

$$\omega = vs, \quad z_\theta = \theta, \tag{8.4}$$

where z_θ is the measurement and $s = \frac{\tan\delta}{L}$. We construct an adaptive observer as follows:

$$\begin{aligned} M_{k+1} &= M_k - T_s(gM_k + \hat{v}_k), \\ \hat{s}_{k+1} &= \hat{s}_k + T_s(\lambda M_k(z_\theta - \hat{\theta}_{a,k})), \\ \hat{\theta}_{a,k+1} &= \hat{\theta}_{a,k} + T_s(\hat{s}_k\hat{v}_k + g(z_\theta - \hat{\theta}_{a,k}) + M_k(\lambda M_k(z_\theta - \hat{\theta}_{a,k})), \end{aligned} \tag{8.5}$$

where M is an auxiliary signal, $g, \lambda \in \mathcal{R}$ are observer gains, $\hat{\theta}_a$ is the estimated heading angle, and $\hat{v}$ is the velocity estimated by EKF. The estimated steering angle can be calculated with $\hat{s}$, i.e., $\hat{\delta} = \tan^{-1}(\hat{s}L)$. It is not hard to verify that as long as $\hat{v} \to v$ and v is non-zero, the steering angle estimate is guaranteed to converge to its true value as $t \to \infty$. The output of the cascade motion estimator is denoted as $X_{cc,k} = [\hat{x}_k, \hat{y}_k, \hat{\theta}_k, \hat{\delta}_k, \hat{v}_k]^\top$.

8.4.2 Short-term motion prediction

For the sake of computation efficiency, we assume the short-term motion of an OV is fully captured by the mean value of the state X_{cc} and its covariance. For the mean value, we propagate the estimated states $X_{cc,k}$ forward and obtain a short-term prediction $\mathbf{X}_{H,k} = [X_{1,k}^\top, \ldots, X_{H,k}^\top]^\top$ for the future H steps of the time horizon. Similarly, forward propagation is carried out to obtain the covariance matrices $Pm_{H,k} = \{Pm_{k+1}, \ldots, Pm_{k+H}\}$ according to EKF's forward prediction formula. These information will facilitate long-term prediction and be used to determine the safety margin for each future time step.

8.4.3 Long-term mode prediction

OV's long-term motion is dependent on the history of its state, the dynamic model, and its relative movement against the environment, where the first two factors are captured to some extent by the short-term motion prediction. The OV's relative movement to the environment is important for long-term prediction. To exploit the relative movement against the environment, [49] and [67] introduced a cost map to capture an OV's possible long-term movements. We adopt the same idea and construct a cost map, M_{route} using a route planner [42], where the cost map contains possible routes that the OV will take (8.6). Also, we recognize that a vehicle in the parking lot normally runs in two modes, "maneuvering" and "cruising." Vehicles in maneuvering mode change the steering frequently and deviate from the routes (black dashed line in Figure 8.5) in the cost map to park or leave the narrow parking spot. Vehicles in cruising mode have small or steady steering angles and generally follow one of the routes. Vehicles are in this mode when they first enter the parking lot and are approaching a parking spot or when they get out of the parking spot and are leaving the parking lot. Including the route information, an OV that has n routes to follow will have $2n$ possible modes (Figure 8.5). To determine the mode m at time step k, i.e., m_k, Bayesian framework is employed

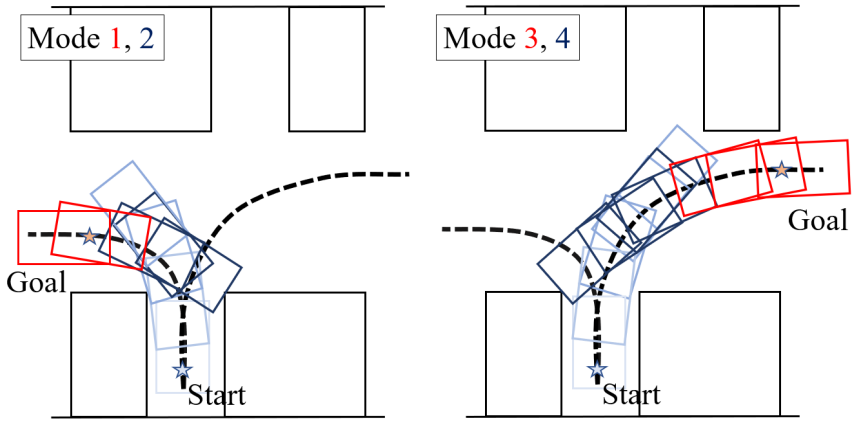


Figure 8.5: There are 2 routes (black dashed lines) and 2 main modes (red for "cruising" and blue for "maneuvering") in this example, resulting in a total of 4 modes for the OV: 1) cruise(exit) left; 2) maneuver left; 3) cruise(exit) right; 4) maneuver right.

to keep track of the belief of each mode, i.e., $b(m_k)$. The process is described in Algorithm 6, lines 5-8. We perform the prior believe update $p(m_k) = T_b b(m_{k-1})$ based on the previous belief, $b(m_{k-1})$. The posterior $p(m_k|X_{cc,k}, \mathbf{X}_{H,k})$ is proportional to the prior multiplied with the conditional probability of the motion estimation and prediction given the mode, i.e., $p(m_k)p(X_{cc,k}, \mathbf{X}_{H,k}|m_k)$. The Boltzmann policy is one common way to design this conditional probability [10], i.e., $p(X_{cc,k}, \mathbf{X}_{H,k}|m_k) \propto \exp(-M_{route}(m_k, X_{cc,k}, \mathbf{X}_{H,k}))f(m_k, X_{cc,k})$. The function $M_{route}(m_k, X_{cc,k}, \mathbf{X}_{H,k})$ compares the OV states and predictions with the routes:

$$\begin{aligned} M_{route}(m_k, X_{cc,k}, \mathbf{X}_{H,k}) = &\min_i \|X_{m_k,i} - X_{cc,k}\|^2_{W_1} \\ &+ \sum_h \min_i \|X_{m_k,i} - X_{h,k}\|^2_{W_2}, \end{aligned} \tag{8.6}$$

where $X_{m_k,i}$ is the ith waypoint of the route in mode $m_k = m$, $m \in \{1, \ldots, 2n\}$, $\|\mathrm{v}\|^2_W = \mathrm{v}^\top W \mathrm{v}$, and W_1 and W_2 are weighting matrices. The function $f(m_k, X_{cc,k})$ is proportional to the magnitude of the OV's steering angle and the deviation of the OV's heading angle from the final heading angle of the route. Finally, we normalize $p(m_k|X_{cc,k}, \mathbf{X}_{H,k})$ to obtain $b(m_k)$ and take the value of m_k with the largest belief to be $\hat{m}_k$.

8.4.4 Safety margin and safety bound

When the OV is in cruising mode, the predictor calculates the safety margins $\mathbf{s}_{H,k} = [s^\top_{k+1}, \ldots, s^\top_{k+H}]^\top$ (red areas in Figure 8.6) according to Algorithm 6, lines 9 and 10. The safety margin of the hth future time step is an ellipsoid and the length of the principal semi-axes s_{k+h} $(h = 1, \ldots, H)$ are proportional to the differential entropy of the mode belief and the covariance from the motion estimation, i.e., $s_{k+h} \propto (b_{uncertainty} C^\top P m_{k+h} C)$.

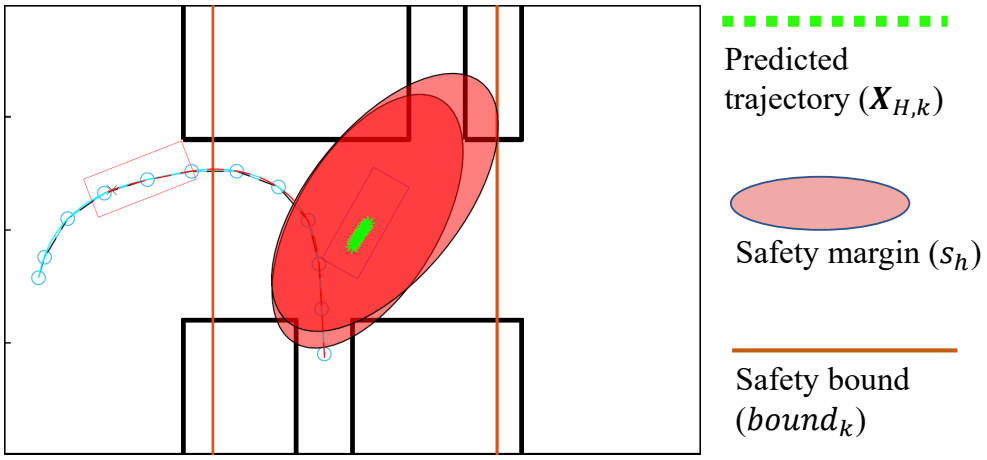


Figure 8.6: The hybrid predictor predicts a short-term OV trajectory (green line) and use it with mode prediction to generate the safety margins for $h = 1$ and $h = H$ and the safety bound.

The movements of the OV in maneuver mode (colored in blue in Figure 8.5) are hard to predict precisely and thus causing uncertainty in the prediction result. To handle such uncertainty, the predictor generates a safety bound (the bound of a convex hall of the OV's pose history, orange lines in Figure 8.6) to enclose the area that may be affected by the motion uncertainties. To comply with the safety bound, the planner behaves more conservatively and keeps the ego AV away from the hardly predictable OVs. This design is inspired by the work presented in Chapter 6, which is to encourage or force conservative behaviors when uncertainty occurs by enlarging the infeasible area. Notice that the safety margin and safety bound can also be applied to other moving obstacles such as pedestrians or motorbikes, given their kinematic models and routes information.

8.5 Strategic Motion Planner

With the reference trajectory $\mathcal{P}_{ref}$, the strategic motion planner runs the main module, the MPC-based safety controller, and two supporting modules: the retreat planner and the repair planner (Figure 8.2). These planners are activated if the ego AV's current location and reference trajectory is invalidated by the OV's movements, respectively. The strategic motion planner is summarized in Algorithm 7.

8.5.1 MPC-based safety controller

The safety controller is designed similarly to that in Chapter 5, which tracks the reference trajectory $\mathcal{P}_{ref}$ given the safety margin and the safety bound. With $\mathcal{P}_{ref}$, we use an

Algorithm 6: Hybrid Environment Predictor

input z_k, M_{route}
$X_{EKF,k} \leftarrow \texttt{EKF}_{\texttt{unicycle}}(z_k)$
$X_{cc,k} \leftarrow \texttt{steeringEST}(X_{EKF,k}, z_k)$
$\mathbf{X}_{H,k}, Pm_H \leftarrow \texttt{propagate}(X_{cc,k})$
$p(m_k) \leftarrow T_p b(m_{k-1})$
$p(m_k|X_{cc,k}, \mathbf{X}_{H,k}) \propto p(X_{cc,k}, \mathbf{X}_{H,k}|m_k)p(m_k)$
$b(m_k) \leftarrow \texttt{normalize}(p(m_k|X_{cc,k}, \mathbf{X}_{H,k}))$
$\hat{m}_k \leftarrow \arg\max_{m_k} b(m_k)$
$b_{uncertainty} \propto -\sum(b(m_k)\log(b(m_k)))$
$\mathbf{s}_{H,k} \leftarrow \texttt{getSafetyMargin}(b_{uncertainty}, Pm_{H,k})$
if m **is** `maneuver` **then**
  $bound_k \leftarrow \texttt{getConvexHall}(X_{cc,k}, bound_{k-1})$
return $X_{cc,k}, \mathbf{X}_{H,k}, \mathbf{s}_{H,k}, b(m_k), \hat{m}_k, bound_k$

optimization-based planner to compute tracking motions in an MPC framework. Let $\mathbf{X}_{ref,k}$ be the segment of $\mathcal{P}_{ref}$ to track at time step k. $\mathbf{X}_{ref,k}$ is selected and trimmed so that it will not violate the safety margin (in all modes) nor the safety bound (in "maneuver" modes). The trajectory tracking problem is formulated as follows:

Problem 6. *Given the planning horizon H, the vehicle model* (8.2)*, and the reference segment $\mathbf{X}_{ref,k}$, the optimization planner solves the problem*

$$\begin{aligned} \mathbf{u}_k^* = \arg\min_{\mathbf{u}_k} \quad & \|F(X_k) + G(X_k, \mathbf{X}_k, \mathbf{u}_k) - \mathbf{X}_{ref,k}\|_{W_3}^2, \\ s.t. \quad & \mathbf{X}_k = F(X_k) + G(X_k, \mathbf{X}_k, \mathbf{u}_k), \\ & (\mathbf{X}_k, \mathbf{u}_k) \in \Gamma_k, \end{aligned} \tag{8.7}$$

where $\mathbf{X}_k = [X_{k+1}^\top, X_{k+2}^\top, \ldots, X_{k+H}^\top]^\top$, $\mathbf{u}_k = [u_k^\top, u_{k+2}^\top, \ldots, u_{k+H-1}^\top]^\top$, and Γ_k defines the feasible set, $\Gamma_k = \{(\mathbf{X}_k, \mathbf{u}_k) : X_{k+h} \in \mathcal{C}_{free,k},\ u_{k+h-1} \in [-u_{max}, u_{max}], \forall h = 1, \ldots, H\}$.

Problem 6 can be readily formulated as a non-convex optimization problem using various software tools, e.g., CasADi [8], and solved using nonlinear programming solvers, e.g., IPOPT. With the reference path serving as a warm start, the average solving time is around 0.06 second. Details are omitted here.

8.5.2 Retreat planner

The retreat planner deals with scenarios when stopping or staying on the original reference is deemed unsafe. This can happen when the OV drives toward the ego AV, and its motion vastly differs from the previous prediction - possibly violating the safety margin and causing a safety threat. Therefore, the ego AV needs to find a path and retreat from the emergency.

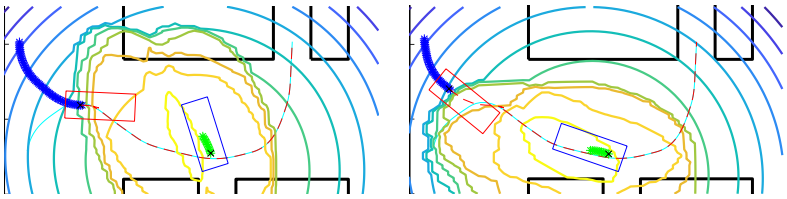


Figure 8.7: The ego AV following the retreating plans (blue star-lines). The light-blue lines illustrate the original trajectory, the red dashed-lines combining the blue star-lines will be the new reference $\mathcal{P}_{ref}$, and the collision field is illustrated by the contours.

The retreating movement is not a standard navigation problem because the ego AV hasn't had a safe goal. Instead, it needs to explore the environment to find the best goal, and thus we propose a search-based retreat planner which explores the space and quickly finds a retreating trajectory. It also allows us to represent the environment without too many relaxations that optimization planners often require.

The retreat planning algorithm is similar to that presented in Chapter 4 but simplified for the simpler car kinematics, allowing it to serve as a real-time planner for safety-critical scenarios. As a variant of A*-based algorithms, the retreat planner constructs a tree $\mathcal{T} = (\mathcal{V}, \mathcal{E})$ composed of a node set $\mathcal{V} \subset \mathcal{C}_{free}$ and an edge set $\mathcal{E}$, where $E(X_i, X_j) \in \mathcal{E}$ represents a feasible short path between X_i and X_j. $\mathcal{C}_{free}$ is implicitly obtained by checking collisions with obstacles in the parking lot map M_{map}. Let $\mathcal{M}$ denote a finite set of motion primitives pre-computed through available control actions, and V_{max} denotes the maximum number of nodes allowed. The retreat planner constructs a tree $\mathcal{T}$ from X_k (the configuration when the retreat planning starts) and expands it according to a cost function $\mathcal{F}(\cdot)$ which sums up the heuristic value $h(\cdot)$ and the arrival cost $g(\cdot)$. The heuristic (8.8) is calculated based on a collision field as shown in Figure 8.7. The field is a weighted sum of Gaussian distributions centered at waypoints of both the predicted trajectory $\mathbf{X}_{H,k}$, i.e., $X_{h,k}$, and the routes on the cost map, i.e., $X_{m_k,i}$, and

$$h(X) = \sum_{m_k,i} b(m_k) e^{-\|X - X_{m_k,i}\|^2_{W_4}} + \sum_{h} c e^{-\|X - X_{h,k}\|^2_{W_5}}, \tag{8.8}$$

where c is a weighting constant. The planning is completed if the ego AV finds a trajectory that keeps it away from the OV at a safe clearance. If the number of nodes in $\mathcal{T}$ reaches $V_{\max}$, the trajectory giving the maximum clearance is chosen.

8.5.3 Trajectory repair planner

OV's movements could invalidate the ego AV's reference trajectory. Figure 8.8 illustrates one such case, where an OV, represented by the blue box, stops on the AV's reference trajectory, represented as the light-blue line. The safety controller will command the ego AV to stop on the reference trajectory when the area in front is infeasible. Unless receiving

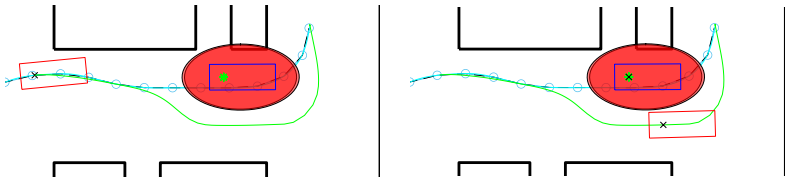


Figure 8.8: The ego AV (red vehicle) following the repaired plan (green lines) that is calculated from the blocked original trajectory (light-blue lines).

a new reference trajectory, the safety controller will stop the ego AV and wait for the OV to clear - not efficient if the OV stops for a long time. It is reasonable to update $\mathcal{P}_{ref}$, so that the safety planner can command the ego AV to go around the OV and merge back to the original path. The repaired trajectory usually lies in the same homotopy class as the original one, which makes an optimization-based repairing strategy a viable solution. To obtain a repaired path quickly, we conduct repair path planning over 2D space, i.e., $X_{repair} = [x, y]^\top$, and modify the constraints accordingly. It is understood that the resultant path, despite being collision-free, cannot always be followed accurately, causing the AV to collide into obstacles. We, therefore, verify the path and accept the repaired trajectory (as shown in Figure 8.8) only if it passes the kinematic feasibility check. If the repairing fails, the central control will be notified to take over the repairing task.

8.6 Applications

The proposed system is tested by simulation conducted on a 6-core Intel i7 3.7GHz desktop with Matlab R2020a. The prediction horizon is 10, i.e., $H = 10$. The integrated system is set to run at a rate of 4 Hz (all calculation in each time step is finished within 0.1 seconds). This section presents one of the simulation results as an example. More simulation results can be found in the video available here.

As shown in Figure 8.9(a), the ego AV (red box) first performs parking by tracking the reference trajectory $\mathcal{P}_{ref}$ (light-blue line) while avoiding collision with the OV (blue box). The safety margins are illustrated by red shaded areas, where the one with the black edge is for the current time step and the one without the edge is for the Hth future time step (the value is shown in Figure 8.10). As the OV comes out from its parking spot, it moves towards the ego AV. The ego AV needs to retreat temporarily to make space for the OV (Figure 8.9(b)-(d)). Although the ego AV might be able to avoid collision by backing up along the original reference, this movement may be dangerous since that is the direction where the OV is heading and will potentially block the OV. This showcases the importance of considering the long-term mode to construct a collision field (as shown in Figure 8.7) of the OV in retreat planning. In Figure 8.9(b)-8.9(c), the retreat planner is activated to update the $\mathcal{P}_{ref}$. When violation happens again (Figure 8.9(d)), the retreat planner updates $\mathcal{P}_{ref}$ once more. As the OV completely leaves the parking spot, the estimator detects the

Algorithm 7: The Strategic Motion Planner

input $M_{map}, M_{route}, X_{goal}$
`RequestCentralPathPlanning`(X_{goal})
while X_{goal} **not reached do**
  if `ReceiveCentralMsg` **then**
    $\mathcal{P}_{ref} \leftarrow$ `Update`$(\mathcal{P}_{ref})$
  $z_k \leftarrow$ `GetMeasurements`
  $X_{cc,k}, \mathbf{X}_{H,k}, \mathbf{s}_{H,k}, b(m_k), \hat{m}_k, bound_k \leftarrow$ `ENVPredict`(z_k, M_{route})
  $flag_{retreat} \leftarrow$ `SafetyCheck`$(X_{cc,k}, s_k, bound_k, \hat{m}_k)$
  if $flag_{retreat}$ **then**
    $\mathcal{P}_{ref} \leftarrow$ `PlanRetreat`$(M_{map}, M_{route}, X_{cc,k}, \mathbf{X}_{H,k}, b(m_k))$
  $\mathbf{X}_{OV,history} \leftarrow$ `OV_MotionHistory`$(X_{cc,k})$
  $flag_{repair} \leftarrow$ `BlockerCheck`$(\mathbf{X}_{OV,history}, \mathcal{P}_{ref})$
  if $flag_{repair}$ **then**
    $\mathcal{P}_{ref}, flag_{fail} \leftarrow$ `PlanRepair`$(P_{ref}, X_{cc,k})$
  if $flag_{fail}$ **then**
    `RequestCentralPathPlanning`$(X_{goal}, X_{cc,k})$
  $\mathbf{X}_{ref,k} \leftarrow$ `setXref`$(\mathcal{P}_{ref}, \mathbf{X}_{H,k}, \mathbf{s}_{H,k}, \hat{m}_k, bound_k)$
  $u_k \leftarrow$ `SafetyController`$(\mathbf{X}_{ref,k}, \mathbf{X}_{H,k}, \mathbf{s}_{H,k})$
return u_k

mode switch (from "maneuver left" to "cruise (exit) left" in Figure 8.10), therefore the safety bound is removed (Figure 8.9(e)) and the safety controller continues to follow the reference path (Figure 8.9(e)-8.9(f)). Later, the OV stops at the road entrance and blocks the original reference trajectory of the ego AV. Therefore, the ego AV tries repairing the trajectory (Figure 8.9(g)). However, the repairing fails (the solver converges to an infeasible red path). Because of the narrow space, cusps are required in the maneuver, which is generally hard for an optimization-based planner to generate. Therefore, a new trajectory from the central control is needed. Once the updated trajectory $\mathcal{P}_{ref}$ (Figure 8.9(h)) is received, the safety controller will start following it until the ego AV reaches the goal. In the video, we show a successful path repair in demo 1 and the effectiveness of the safety bound (so that the retreat planner won't be triggered unnecessarily) in demo 3.

8.7 Chapter Summary

This chapter presented an integrated motion planning strategy for an AV to park in uncertain environments. A hybrid environment predictor incorporated the model-based short-term motion prediction and a driver behavior cost-map to make long-term predictions of an

OV. A strategic motion planner was composed of an MPC-based safety controller, a search-based retreat planner, and an optimization-based repair planner. It struck a good balance between safety, plan feasibility, and smooth maneuver by leveraging optimization-based and search-based methods. The strategic motion planner generated safe and smooth trajectories based on the predictor's results and brought the AV to the target directly or through an intermediate safe spot to yield to the OV. Simulation results demonstrated that the proposed integrated robotic system and strategies enabled the ego AV to plan safely and smoothly in complicated dynamic parking environments.

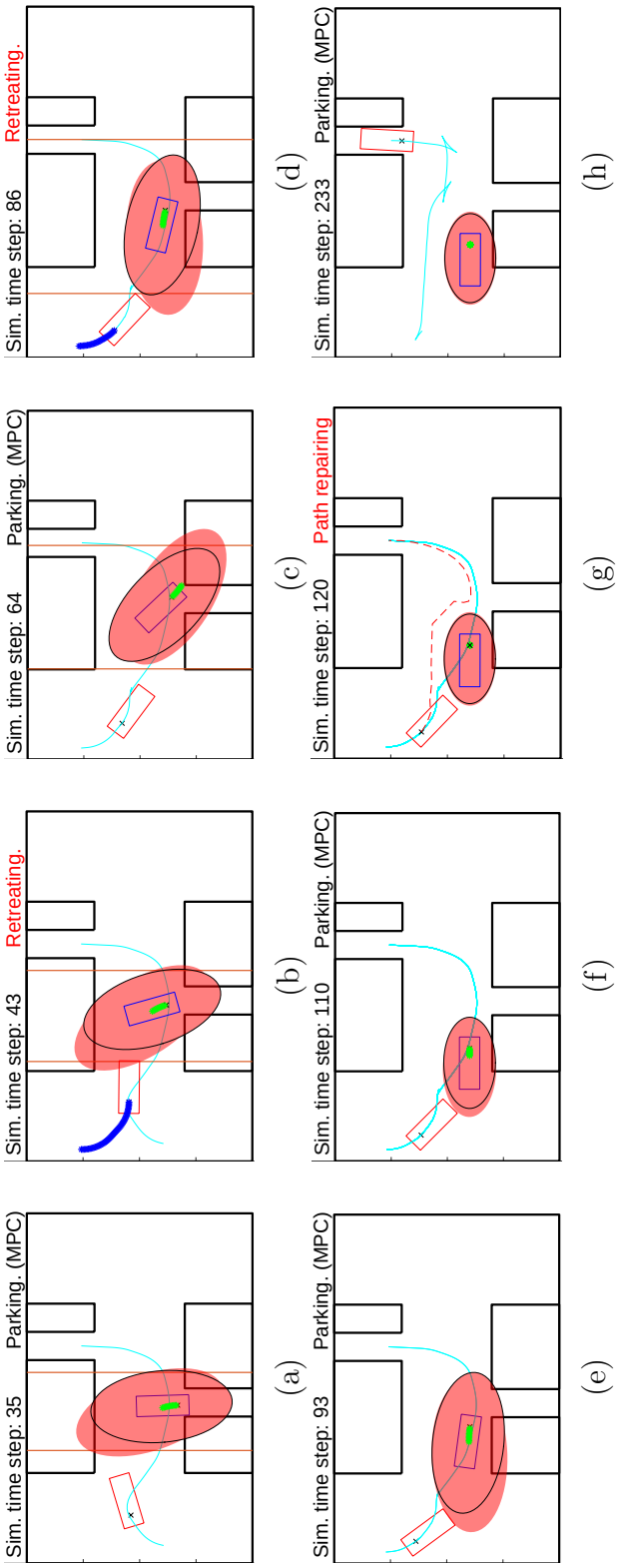


Figure 8.9: Simulation results in one of the parking scenarios. (0.25 s/time step)

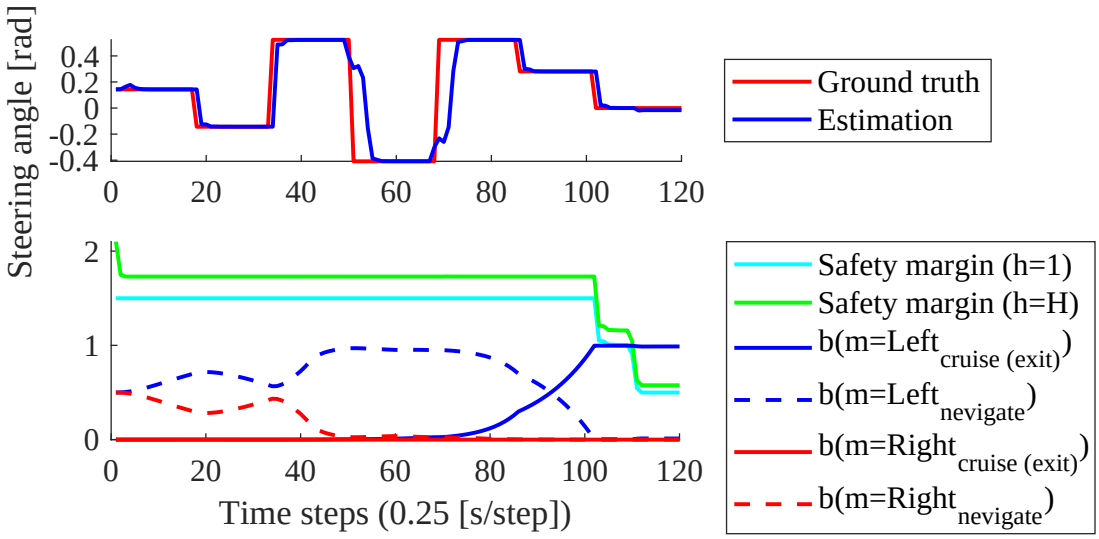


Figure 8.10: The simulation result of steering estimation, mode estimation, and safety margin.

Chapter 9

Concluding Remarks and Suggested Future Works

This dissertation explored the methods of designing integrated strategies for modularized robotic systems in uncertain environments. Integrated systems that featured the prediction module, the planning module, and the collaboration of the two were presented in Chapters 7 and 8. In Chapter 7, simulations with a computer assembly setting that involved a robot collaborating with one human worker were conducted; the proposed robotic system coordinated the prediction and planning modules to utilize human motion prediction and uncertainty estimation for robust task planning. The robot could generate time-efficient task plans when the human worker performed inefficiently. In Chapter 8, simulations involving an autonomous vehicle navigating in a parking lot while avoiding collision with static and moving obstacles were conducted; the proposed system included a hybrid environment predictor that made short-term and long-term predictions of the surrounding vehicles and a strategic motion planner that reacted to the environment according to the predictions. The robot demonstrated the effectiveness of the proposed method in terms of motion prediction, safe tracking, retreating in an emergency, and trajectory repairing. Chapter 6 discussed the close relationship between the prediction and planning modules and identified several conditions for realizing safe MPC in dynamic and uncertain environments; we presented a predictor designed for better closed-loop robot performance. Simulations and real-world experiments that involved a robot working alongside a human worker were conducted; the robot could navigate safely in the presence of unexpected human movements. The proposed systems in Chapters 6-8 utilized human-identified, application-specific knowledge to estimate the uncertainties, i.e., the predefined modes and human actions. Similar knowledge is required when extending these systems to different applications, e.g., applications in domestic environments.

Chapters 2-5 discussed the design of the motion planners for maneuvering tasks. Their performances were validated in simulation. Planners presented in Chapters 2-4 focused on combining different planning methods to develop hybrid planners for static and deterministic environments cluttered with obstacles. Chapter 2 combined RRT* and CFS to plan

for robots in cluttered environments. Chapter 3 focused on long-horizon planning problems; the proposed RRT*-sOpt combined RRT*, CFS, IPOPT, and segmented trajectory optimization. Simulation results with various environment settings and robot platforms demonstrated the advantage of the proposed motion planners in terms of computational speed. Chapter 4 focused on motion planning for articulated vehicles; it combined i-AGT and a SAC reinforcement learning agent and demonstrated its advantages in computational speed and plan quality. Chapter 5 targeted dynamic and uncertain environments and presented a hierarchical planning and control framework, HRHC. The HRHC demonstrated its ability to coordinate a motion planner, such as the planners presented in Chapters 2-4, with a safety controller to achieve safe and efficient robot motion in uncertain environments in simulations and real-world experiments.

In summary, the works presented in Part I provided better motion planning tools for robot applications to systems such as factories, warehouses, and autonomous vehicles; the works presented in Part II served as examples of applying modularized robotic systems in uncertain environments. There are many directions for future works, which are listed below.

Develop an integrated system with prediction, task planning, and motion planning

We presented an integrated system that combined prediction and task planning in Chapter 7 and an integrated system that combined prediction and motion planning in Chapter 8. While the autonomous parking application may not require a task planner, in the assembly task presented in Chapter 7, considering the influence of the robot's motion on the spatial interference will benefit the task planner's performance. For example, without considering its motion, a robot may execute an action that forces the workers to deviate from their original plan to avoid collisions, thus causing a longer action completion time. With the motion planning knowledge, the robot may plan for another action for the human workers or for itself so that they can have less spatial interference.

Further investigation on agents' modes

Human workers' and drivers' *modes* have been introduced in Chapters 6 and 8, respectively. Unlike short-term *motion* predictions that predict states of the surrounding agents, these modes are identified to predict the *behavior types* of the agent. Identifying these modes enabled us to make better long-term predictions and planning. In Chapter 4, modes can also help the search-based planner plan more efficiently. In this dissertation, we empirically selected the modes of the agents. Note that an intelligent agent will choose its mode based on its characteristics, the geometry of the surroundings, and other agents' characteristics and motions. It will be a promising approach to use data-driven methods and develop standardized approaches for identifying the agents' modes.

Environment uncertainty modeling

While Chapters 6-8 introduced different ways to estimate the uncertainties in the environment, environment uncertainty modeling remains challenging. In these chapters, we focused on uncertainties introduced by other agents. On the other hand, a robot that has physical contact with the environment, for example, a robot pushing open a door, can also experience uncertainties induced by the unknown physical properties of the objects in contact, for example, the weight of the door. It will be a promising approach to develop a generalizable environment uncertainty model so that the robot can plan an effective plan even in the presence of different kinds of uncertainties.

Improve the modules of the robotic system

The computation efficiency of the algorithms in each module is crucial to the overall robot performance, and improvements can be made by better algorithm implementation. On the other hand, a strategic planner, such as the one presented in Chapter 8, uses different algorithms for different scenarios to improve computational efficiency. It is suggested to extend this strategic idea to other modules.

Analysis and evaluation of the integrated robotic systems

While Chapter 6 introduced M-convergence to evaluate the closed-loop system performance, and Chapter 7 evaluated the system with task completion time, the analysis of robot systems in uncertain environments remains challenging. Standardized platforms and benchmarks are crucial for system evaluation. One of the major challenges is to unify the input/output of each module across different robotic systems. For example, we need to have the same type of input (e.g., RGB images) and output (e.g., motor angular velocity command) to compare the performances of different robotic systems and each individual module. Developing such standardized evaluation platforms for modularized robotic systems is essential so that researchers from different groups can better compare, combine, and improve their designs for modularized robotic systems.

With the unstopping improvement of technology and the continuing determination of researchers, achieving safe and efficient robotic system in uncertain environments to enable more robotic applications in our society is no longer just a dream but a future to pursue with hope and excitement.

Bibliography

[1] Olzhas Adiyatov and Huseyin Atakan Varol. "A novel RRT*-based algorithm for motion planning in Dynamic environments". In: *2017 IEEE International Conference on Mechatronics and Automation (ICMA)*. IEEE. 2017, pp. 1416–1421.

[2] Sandip Aine et al. "Multi-heuristic a". In: *The International Journal of Robotics Research* 35.1-3 (2016), pp. 224–243.

[3] Zlatan Ajanovic et al. "Search-based optimal motion planning for automated driving". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 4523–4530.

[4] Javier Alonso-Mora et al. "Local motion planning for collaborative multi-robot manipulation of deformable objects". In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2015, pp. 5495–5502.

[5] Claudio Altafini, Alberto Speranzon, and Bo Wahlberg. "A feedback control scheme for reversing a truck and trailer vehicle". In: *IEEE Transactions on robotics and automation* 17.6 (2001), pp. 915–922.

[6] Matthias Althoff, Olaf Stursberg, and Martin Buss. "Model-based probabilistic collision detection in autonomous driving". In: *IEEE Transactions on Intelligent Transportation Systems* 10.2 (2009), pp. 299–310.

[7] Nancy M Amato et al. "OBPRM: An obstacle-based PRM for 3D workspaces". In: *Robotics: The Algorithmic Perspective: 1998 Workshop on the Algorithmic Foundations of Robotics*. 1998, pp. 155–168.

[8] Joel A E Andersson et al. "CasADi – A software framework for nonlinear optimization and optimal control". In: *Mathematical Programming Computation* 11.1 (2019), pp. 1–36. DOI: 10.1007/s12532-018-0139-4.

[9] Giovanni Buizza Avanzini, Andrea Maria Zanchettin, and Paolo Rocco. "Constraint-based model predictive control for holonomic mobile manipulators". In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2015, pp. 1473–1479.

[10] Chris L Baker, Rebecca Saxe, and Joshua B Tenenbaum. "Action understanding as inverse planning". In: *Cognition* 113.3 (2009), pp. 329–349.

[11] Eduardo Bejar and Antonio Moran. "A preview neuro-fuzzy controller based on deep reinforcement learning for backing up a truck-trailer vehicle". In: *2019 IEEE Canadian Conference of Electrical and Computer Engineering (CCECE)*. IEEE. 2019, pp. 1–4.

[12] Aharon Ben-Tal, Laurent El Ghaoui, and Arkadi Nemirovski. *Robust optimization*. Princeton university press, 2009.

[13] Kristoffer Bergman and Daniel Axehill. "Combining homotopy methods and numerical optimal control to solve motion planning problems". In: *2018 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2018, pp. 347–354.

[14] Kristoffer Bergman, Oskar Ljungqvist, and Daniel Axehill. "Improved path planning by tightly combining lattice-based path planning and optimal control". In: *IEEE Transactions on Intelligent Vehicles* 6.1 (2020), pp. 57–66.

[15] Robert Bohlin and Lydia E Kavraki. "Path planning using lazy PRM". In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*. Vol. 1. IEEE. 2000, pp. 521–528.

[16] Francesco Borrelli et al. "MPC-based approach to active steering for autonomous vehicle systems". In: *International Journal of Vehicle Autonomous Systems* 3.2-4 (2005), pp. 265–291.

[17] Scott A Bortoff. "Path planning for UAVs". In: *Proceedings of the 2000 American Control Conference. ACC (IEEE Cat. No. 00CH36334)*. Vol. 1. 6. IEEE. 2000, pp. 364–368.

[18] Stephen Boyd et al. "Distributed optimization and statistical learning via the alternating direction method of multipliers". In: *Foundations and Trends® in Machine learning* 3.1 (2011), pp. 1–122.

[19] Taylor Boyd et al. "Hierarchical Task Learning Through Human Demonstration". In: (2019).

[20] Felix Burget, Maren Bennewitz, and Wolfram Burgard. "BI 2 RRT*: An efficient sampling-based path planning framework for task-constrained mobile manipulation". In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2016, pp. 3714–3721.

[21] Judith Bütepage, Hedvig Kjellström, and Danica Kragic. "Anticipating many futures: Online human motion prediction and synthesis for human-robot collaboration". In: *arXiv preprint arXiv:1702.08212* (2017).

[22] Bryant Chandler and Michael A Goodrich. "Online RRT* and online FMT*: Rapid replanning with dynamic cost". In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 6313–6318.

[23] Chao Chen, Markus Rickert, and Alois Knoll. "Kinodynamic motion planning with space-time exploration guided heuristic search for car-like robots in dynamic environments". In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2015, pp. 2666–2671.

[24] Chao Chen, Markus Rickert, and Alois Knoll. "Path planning with orientation-aware space exploration guided heuristic search for autonomous parking and maneuvering". In: *2015 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2015, pp. 1148–1153.

[25] Jianyu Chen, Shengbo Eben Li, and Masayoshi Tomizuka. "Interpretable end-to-end urban autonomous driving with latent deep reinforcement learning". In: *IEEE Transactions on Intelligent Transportation Systems* (2021).

[26] Jianyu Chen, Changliu Liu, and Masayoshi Tomizuka. "Foad: Fast optimization-based autonomous driving motion planner". In: *2018 Annual American Control Conference (ACC)*. IEEE. 2018, pp. 4725–4732.

[27] Jianyu Chen, Wei Zhan, and Masayoshi Tomizuka. "Constrained iterative lqr for on-road autonomous driving motion planning". In: *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE. 2017, pp. 1–7.

[28] Yujiao Cheng, Liting Sun, and Masayoshi Tomizuka. "Human-Aware Robot Task Planning Based on a Hierarchical Task Model". In: *IEEE Robotics and Automation Letters* 6.2 (2021), pp. 1136–1143.

[29] Yujiao Cheng and Masayoshi Tomizuka. "Long-Term Trajectory Prediction of the Human Hand and Duration Estimation of the Human Action". In: *IEEE Robotics and Automation Letters* 7.1 (2021), pp. 247–254.

[30] Yujiao Cheng et al. "Human motion prediction using semi-adaptable neural networks". In: *2019 American Control Conference (ACC)*. IEEE. 2019, pp. 4884–4890.

[31] Yujiao Cheng et al. "Towards Efficient Human-Robot Collaboration With Robust Plan Recognition and Trajectory Prediction". In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 2602–2609.

[32] T Chettibi et al. "Minimum cost trajectory planning for industrial robots". In: *European Journal of Mechanics-A/Solids* 23.4 (2004), pp. 703–715.

[33] Luigi Chisci, J Anthony Rossiter, and Giovanni Zappa. "Systems with persistent disturbances: predictive control with restricted constraints". In: *Automatica* 37.7 (2001), pp. 1019–1028.

[34] Ji-wung Choi and Kalevi Huhtala. "Constrained global path optimization for articulated steering vehicles". In: *IEEE Transactions on Vehicular Technology* 65.4 (2015), pp. 1868–1879.

[35] Marcello Cirillo. "From videogames to autonomous trucks: A new algorithm for lattice-based motion planning". In: *2017 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2017, pp. 148–153.

[36] Marcello Cirillo, Lars Karlsson, and Alessandro Saffiotti. "Human-aware task planning for mobile robots". In: *2009 International Conference on Advanced Robotics*. IEEE. 2009, pp. 1–7.

[37] Marcello Cirillo, Tansel Uras, and Sven Koenig. "A lattice-based approach to multi-robot motion planning for non-holonomic vehicles". In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2014, pp. 232–239.

[38] Benjamin Cohen, Sachin Chitta, and Maxim Likhachev. "Single-and dual-arm motion planning with heuristic search". In: *The International Journal of Robotics Research* 33.2 (2014), pp. 305–320.

[39] Benjamin J Cohen, Sachin Chitta, and Maxim Likhachev. "Search-based planning for manipulation with motion primitives". In: *2010 IEEE International Conference on Robotics and Automation*. IEEE. 2010, pp. 2902–2908.

[40] David C Conner et al. "Valet parking without a valet". In: *2007 IEEE/RSJ international conference on intelligent robots and systems*. IEEE. 2007, pp. 572–577.

[41] Eva Coupeté, Fabien Moutarde, and Sotiris Manitsaris. "Gesture recognition using a depth camera for human robot collaboration on assembly line". In: *Procedia Manufacturing* 3 (2015), pp. 518–525.

[42] Siyu Dai and Yebin Wang. "Long-Horizon Motion Planning for Autonomous Vehicle Parking Incorporating Incomplete Map Information". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 8135–8142.

[43] Siyu Dai et al. "Improving trajectory optimization using a roadmap framework". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 8674–8681.

[44] Nachiket Deo, Akshay Rangesh, and Mohan M Trivedi. "How would surround vehicles move? a unified framework for maneuver classification and motion prediction". In: *IEEE Transactions on Intelligent Vehicles* 3.2 (2018), pp. 129–140.

[45] Wenchao Ding et al. "An efficient b-spline-based kinodynamic replanning framework for quadrotors". In: *IEEE Transactions on Robotics* 35.6 (2019), pp. 1287–1306.

[46] Wenchao Ding et al. "Trajectory replanning for quadrotors using kinodynamic search and elastic optimization". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 7595–7602.

[47] Dmitri Dolgov et al. "Path planning for autonomous vehicles in unknown semi-structured environments". In: *The International Journal of Robotics Research* 29.5 (2010), pp. 485–501.

[48] Dmitri Dolgov et al. "Practical search techniques in path planning for autonomous driving". In: *Ann Arbor* 1001.48105 (2008), pp. 18–80.

[49] Dave Ferguson, Thomas M. Howard, and Maxim Likhachev. "Motion planning in urban environments: Part II". In: *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2008, pp. 1070–1076. DOI: 10.1109/IROS.2008.4651124.

[50] Dave Ferguson, Nidhi Kalra, and Anthony Stentz. "Replanning with rrts". In: *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.* IEEE. 2006, pp. 1243–1248.

[51] Devendra P Garg and Manish Kumar. "Optimization techniques applied to multiple manipulators for path planning and torque minimization". In: *Engineering applications of artificial intelligence* 15.3-4 (2002), pp. 241–252.

[52] Ali Ghadirzadeh et al. "Human-centered collaborative robots with deep reinforcement learning". In: *IEEE Robotics and Automation Letters* 6.2 (2020), pp. 566–571.

[53] Tinka RA Giele et al. "Dynamic Task Allocation for Human-robot Teams." In: *ICAART (1)* 1 (2015), pp. 117–124.

[54] Markus Giftthaler et al. "Efficient kinematic planning for mobile manipulators with non-holonomic constraints using optimal control". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 3411–3417.

[55] Matthew C Gombolay, Ronald J Wilcox, and Julie A Shah. "Fast scheduling of robot teams performing tasks with temporospatial constraints". In: *IEEE Transactions on Robotics* 34.1 (2018), pp. 220–239.

[56] Michael A Goodrich, Alan C Schultz, et al. "Human–robot interaction: a survey". In: *Foundations and Trends® in Human–Computer Interaction* 1.3 (2008), pp. 203–275.

[57] Dizan Alejandro Vasquez Govea et al. "Moving Obstacles' Motion Prediction for Autonomous Navigation". In: *Proc. of the Int. Conf. on Control, Automation, Robotics and Vision*. 2004.

[58] Dongbing Gu and Huosheng Hu. "A stabilizing receding horizon regulator for nonholonomic mobile robots". In: *IEEE Transactions on Robotics* 21.5 (2005), pp. 1022–1028.

[59] Benjamin Gutjahr, Lutz Gröll, and Moritz Werling. "Lateral vehicle trajectory optimization using constrained linear time-varying MPC". In: *IEEE Transactions on Intelligent Transportation Systems* 18.6 (2016), pp. 1586–1595.

[60] Tuomas Haarnoja et al. "Soft actor-critic algorithms and applications". In: *arXiv preprint arXiv:1812.05905* (2018).

[61] Long Han, Quoc Huy Do, and Seiichi Mita. "Unified path planner for parking an autonomous vehicle based on RRT". In: *2011 IEEE International Conference on Robotics and Automation*. IEEE. 2011, pp. 5622–5627.

[62] P. E. Hart, N. J. Nilsson, and B. Raphael. "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". In: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107.

[63] Peter E Hart, Nils J Nilsson, and Bertram Raphael. "A formal basis for the heuristic determination of minimum cost paths". In: *IEEE transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107.

[64] Carl-Johan Hoel, Krister Wolff, and Leo Laine. "Automated speed and lane change decision making using deep reinforcement learning". In: *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE. 2018, pp. 2148–2155.

[65] Ming Hou and Ron J Patton. "Input observability and input reconstruction". In: *Automatica* 34.6 (1998), pp. 789–794.

[66] Adam Houenou et al. "Vehicle trajectory prediction based on motion model and maneuver recognition". In: *2013 IEEE/RSJ international conference on intelligent robots and systems*. IEEE. 2013, pp. 4363–4369.

[67] Thomas Howard et al. "Model-Predictive Motion Planning: Several Key Developments for Autonomous Mobile Robots". In: *IEEE Robotics Automation Magazine* 21.1 (2014), pp. 64–73. DOI: 10.1109/MRA.2013.2294914.

[68] Ming Feng Hsieh and Umit Ozguner. "A parking algorithm for an autonomous vehicle". In: *2008 IEEE Intelligent Vehicles Symposium*. IEEE. 2008, pp. 1155–1160.

[69] Mads Hvilshøj et al. "Autonomous industrial mobile manipulation (AIMM): past, present and future". In: *Industrial Robot: An International Journal* 39.2 (2012), pp. 120–135.

[70] Mads Hvilshøj et al. "Calibration techniques for industrial mobile manipulators: Theoretical configurations and best practices". In: *Robotics (ISR), 2010 41st International Symposium on and 2010 6th German Conference on Robotics (ROBOTIK)*. VDE. 2010, pp. 1–7.

[71] Fahad Islam, Venkatraman Narayanan, and Maxim Likhachev. "A*-connect: Bounded suboptimal bidirectional heuristic search". In: *2016 IEEE International Conference On Robotics and Automation (ICRA)*. IEEE. 2016, pp. 2752–2758.

[72] Fahad Islam et al. "Rrt*-smart: Rapid convergence implementation of rrt* towards optimal solution". In: *2012 IEEE International Conference on Mechatronics and Automation*. IEEE. 2012, pp. 1651–1656.

[73] Chulhoon Jang et al. "Re-plannable automated parking system with a standalone around view monitor for narrow parking lots". In: *IEEE Transactions on Intelligent Transportation Systems* 21.2 (2019), pp. 777–790.

[74] Yonghwan Jeong et al. "Sampling Based Vehicle Motion Planning for Autonomous Valet Parking with Moving Obstacles". In: *International Journal of Automotive Engineering* 9.4 (2018), pp. 215–222.

[75] Lars Johannsmeier and Sami Haddadin. "A hierarchical human-robot interaction-planning framework for task allocation in collaborative industrial assembly processes". In: *IEEE Robotics and Automation Letters* 2.1 (2016), pp. 41–48.

[76] Matthew Johnson et al. "Team IHMC's lessons learned from the DARPA robotics challenge trials". In: *Journal of Field Robotics* 32.2 (2015), pp. 192–208.

[77] Mrinal Kalakrishnan et al. "STOMP: Stochastic trajectory optimization for motion planning". In: *2011 IEEE international conference on robotics and automation*. IEEE. 2011, pp. 4569–4574.

[78] Sertac Karaman and Emilio Frazzoli. "Sampling-based algorithms for optimal motion planning". In: *The international journal of robotics research* 30.7 (2011), pp. 846–894.

[79] Lydia E Kavraki et al. "Probabilistic roadmaps for path planning in high-dimensional configuration spaces". In: *IEEE transactions on Robotics and Automation* 12.4 (1996), pp. 566–580.

[80] Sebastian Klaudt, Adrian Zlocki, and Lutz Eckstein. "A-priori map information and path planning for automated valet-parking". In: *2017 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2017, pp. 1770–1775.

[81] Sebastian Klemm et al. "Autonomous multi-story navigation for valet parking". In: *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE. 2016, pp. 1126–1133.

[82] Ross A Knepper et al. "Distributed assembly with and/or graphs". In: *Workshop on AI Robotics at the Int. Conf. on Intelligent Robots and Systems (IROS)*. 2014.

[83] Sven Koenig and Maxim Likhachev. "Dˆ* lite". In: *AAAI* 15 (2002).

[84] Hema S Koppula, Ashesh Jain, and Ashutosh Saxena. "Anticipatory planning for human-robot teams". In: *Experimental Robotics*. Springer. 2016, pp. 453–470.

[85] Thibault Kruse et al. "Human-aware robot navigation: A survey". In: *Robotics and Autonomous Systems* 61.12 (2013), pp. 1726–1743.

[86] James J Kuffner and Steven M LaValle. "RRT-connect: An efficient approach to single-query path planning". In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*. Vol. 2. IEEE. 2000, pp. 995–1001.

[87] Puneet Kumar et al. "Learning-based approach for online lane change intention prediction". In: *Intelligent Vehicles Symposium (IV), 2013 IEEE*. IEEE. 2013, pp. 797–802.

[88] Rainer Kummerle et al. "Autonomous driving in a multi-level parking structure". In: *2009 IEEE International Conference on Robotics and Automation*. IEEE. 2009, pp. 3395–3400.

[89] Yoshiaki Kuwata et al. "Distributed robust receding horizon control for multivehicle guidance". In: *IEEE Transactions on Control Systems Technology* 15.4 (2007), pp. 627–641.

[90] Steven M La Valle. "Motion planning". In: *IEEE Robotics & Automation Magazine* 18.2 (2011), pp. 108–118.

[91] Florent Lamiraux, Sepanta Sekhavat, and J-P Laumond. "Motion planning and control for Hilare pulling a trailer". In: *IEEE Transactions on robotics and automation* 15.4 (1999), pp. 640–652.

[92] Christian Laugier et al. "Probabilistic analysis of dynamic scenes and collision risks assessment to improve driving safety". In: *IEEE Intelligent Transportation Systems Magazine* 3.4 (2011), pp. 4–19.

[93] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

[94] Steven M LaValle et al. "Rapidly-exploring random trees: A new tool for path planning". In: (1998).

[95] Steven M LaValle and James J Kuffner Jr. "Randomized kinodynamic planning". In: *The international journal of robotics research* 20.5 (2001), pp. 378–400.

[96] Stéphanie Lefèvre, Dizan Vasquez, and Christian Laugier. "A survey on motion prediction and risk assessment for intelligent vehicles". In: *ROBOMECH journal* 1.1 (2014), pp. 1–14.

[97] Jessica Leu, Rachel Lim, and Masayoshi Tomizuka. "Safe and Coordinated Hierarchical Receding Horizon Control for Mobile Manipulators". In: *2020 American Control Conference (ACC)*. IEEE. 2020, pp. 2143–2149.

[98] Jessica Leu and Masayoshi Tomizuka. "Motion Planning for Industrial Mobile Robots With Closed-Loop Stability Enhanced Prediction". In: *Dynamic Systems and Control Conference*. Vol. 59162. American Society of Mechanical Engineers. 2019, V003T19A009.

[99] Jessica Leu, Michael Wang, and Masayoshi Tomizuka. *Long-Horizon Motion Planning via Sampling and Segmented Trajectory Optimization*. 2022. arXiv: 2204.07939 [cs.RO].

[100] Jessica Leu et al. *Autonomous Vehicle Parking in Dynamic Environments: An Integrated System with Prediction and Motion Planning*. 2022. DOI: 10.48550/ARXIV. 2204.10383. URL: https://arxiv.org/abs/2204.10383.

[101] Jessica Leu et al. "Efficient Robot Motion Planning via Sampling and Optimization". In: *2021 American Control Conference (ACC)*. IEEE. 2021, pp. 4196–4202.

[102] Jessica Leu et al. *Robust Task Planning for Assembly Lines with Human-Robot Collaboration*. 2022. arXiv: 2204.07936 [cs.RO].

[103] Steven James Levine and Brian Charles Williams. "Concurrent plan recognition and execution for human-robot teams". In: *Twenty-Fourth International Conference on Automated Planning and Scheduling*. 2014.

[104] Jesse Levinson et al. "Towards fully autonomous driving: Systems and algorithms". In: *2011 IEEE intelligent vehicles symposium (IV)*. IEEE. 2011, pp. 163–168.

[105] Bai Li et al. "Optimization-based maneuver planning for a tractor-trailer vehicle in a curvy tunnel: A weak reliance on sampling and search". In: *IEEE Robotics and Automation Letters* 7.2 (2021), pp. 706–713.

[106] Bai Li et al. "Trajectory planning for a tractor with multiple trailers in extremely narrow environments: A unified approach". In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 8557–8562.

[107] Kai Li et al. "Sequence planning considering human fatigue for human-robot collaboration in disassembly". In: *Procedia CIRP* 83 (2019), pp. 95–104.

[108] Lening Li, Xianchao Long, and Michael A Gennert. "BiRRTOpt: A combined sampling and optimizing motion planner for humanoid robots". In: *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*. IEEE. 2016, pp. 469–476.

[109] M. Likhachev et al. "Anytime Dynamic A*: An Anytime, Replanning Algorithm". In: *Proc. 2005 ICAPS*. 2005, pp. 262–271.

[110] Maxim Likhachev and Dave Ferguson. "Planning long dynamically feasible maneuvers for autonomous vehicles". In: *The International Journal of Robotics Research* 28.8 (2009), pp. 933–945.

[111] Changliu Liu, Chung-Yen Lin, and Masayoshi Tomizuka. "The convex feasible set algorithm for real time optimization in motion planning". In: *SIAM Journal on Control and Optimization* 56.4 (2018), pp. 2712–2733.

[112] Changliu Liu et al. "Convex feasible set algorithm for constrained trajectory smoothing". In: *2017 American Control Conference (ACC)*. IEEE. 2017, pp. 4177–4182.

[113] Changliu Liu et al. "Serocs: Safe and efficient robot collaborative systems for next generation intelligent industrial co-robots". In: *arXiv preprint arXiv:1809.08215* (2018).

[114] Oskar Ljungqvist et al. "A path planning and path-following control framework for a general 2-trailer with a car-like tractor". In: *Journal of field robotics* 36.8 (2019), pp. 1345–1377.

[115] Oskar Ljungqvist et al. "Lattice-based motion planning for a general 2-trailer system". In: *2017 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2017, pp. 819–824.

[116] Liang Ma et al. "Efficient sampling-based motion planning for on-road autonomous driving". In: *IEEE Transactions on Intelligent Transportation Systems* 16.4 (2015), pp. 1961–1976.

[117] Jim Mainprice and Dmitry Berenson. "Human-robot collaborative manipulation planning using early prediction of human motion". In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2013, pp. 299–306.

[118] David Q Mayne et al. "Constrained model predictive control: Stability and optimality". In: *Automatica* 36.6 (2000), pp. 789–814.

[119] Matthew McNaughton et al. "Motion planning for autonomous driving with a conformal spatiotemporal lattice". In: *2011 IEEE International Conference on Robotics and Automation*. IEEE. 2011, pp. 4889–4895.

[120] Ali Mesbah. "Stochastic model predictive control: An overview and perspectives for future research". In: *IEEE Control Systems Magazine* 36.6 (2016), pp. 30–44.

[121] Kyoung-Wook Min and Jeong-Dan Choi. "Design and implementation of autonomous vehicle valet parking system". In: *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*. IEEE. 2013, pp. 2082–2087.

[122] Michael Montemerlo et al. "Junior: The stanford entry in the urban challenge". In: *Journal of field Robotics* 25.9 (2008), pp. 569–597.

[123] Richard M Murray and S Shankar Sastry. "Steering nonholonomic systems in chained form". In: (1991).

[124] Isak Nielsen and Daniel Axehill. "A parallel structure exploiting factorization algorithm with applications to model predictive control". In: *2015 54th IEEE Conference on Decision and Control (CDC)*. IEEE. 2015, pp. 3932–3938.

[125] Stefanos Nikolaidis and Julie Shah. "Human-robot teaming using shared mental models". In: *ACM/IEEE HRI* (2012).

[126] Brendan O'Donoghue, Giorgos Stathopoulos, and Stephen Boyd. "A splitting method for optimal control". In: *IEEE Transactions on Control Systems Technology* 21.6 (2013), pp. 2432–2442.

[127] Rui Oliveira et al. "Optimization-based on-road path planning for articulated vehicles". In: *IFAC-PapersOnLine* 53.2 (2020), pp. 15572–15579.

[128] Tugcem Oral and Faruk Polat. "MOD* Lite: an incremental path planning algorithm taking care of multiple objectives". In: *IEEE Transactions on Cybernetics* 46.1 (2015), pp. 245–257.

[129] Chonhyon Park et al. "Parallel cartesian planning in dynamic environments using constrained trajectory planning". In: *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*. IEEE. 2015, pp. 983–990.

[130] Mikkel Rath Pedersen et al. "Robot skills for manufacturing: From concept to industrial deployment". In: *Robotics and Computer-Integrated Manufacturing* 37 (2016), pp. 282–291.

[131] J Norberto Pires. *Industrial robots programming: building applications for the factories of the future*. Springer Science & Business Media, 2007.

[132] Mihail Pivtoraiko, Ross A Knepper, and Alonzo Kelly. "Differentially constrained mobile robot motion planning in state lattices". In: *Journal of Field Robotics* 26.3 (2009), pp. 308–333.

[133] Rejean Plamondon, Chunhua Feng, and Anna Woch. "A kinematic theory of rapid human movement. Part IV: a formal mathematical proof and new insights". In: *Biological cybernetics* 89.2 (2003), pp. 126–138.

[134] Jie Qi, Hui Yang, and Haixin Sun. "MOD-RRT*: A Sampling-based algorithm for robot path planning in dynamic environment". In: *IEEE Transactions on Industrial Electronics* (2020).

[135] Anil V Rao. "A survey of numerical methods for optimal control". In: *Advances in the Astronautical Sciences* 135.1 (2009), pp. 497–528.

[136] Nathan Ratliff et al. "CHOMP: Gradient optimization techniques for efficient motion planning". In: *2009 IEEE International Conference on Robotics and Automation*. IEEE. 2009, pp. 489–494.

[137] James B Rawlings. "Tutorial: Model predictive control technology". In: *American Control Conference, 1999. Proceedings of the 1999*. Vol. 1. IEEE. 1999, pp. 662–676.

[138] James Blake Rawlings, David Q Mayne, and Moritz Diehl. *Model predictive control: theory, computation, and design*. Vol. 2. Nob Hill Publishing Madison, WI, 2017.

[139] James Reeds and Lawrence Shepp. "Optimal paths for a car that goes both forwards and backwards". In: *Pacific journal of mathematics* 145.2 (1990), pp. 367–393.

[140] S. Rodríguez et al. "An obstacle-based rapidly-exploring random tree". In: *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.* (2006), pp. 895–900.

[141] Pierre Rouchon et al. "Flatness and motion planning: the car with n trailers". In: *Proc. ECC'93, Groningen*. 1993, pp. 1518–1522.

[142] Thushara Sandakalum and Marcelo H Ang Jr. "Motion Planning for Mobile Manipulators—A Systematic Review". In: *Machines* 10.2 (2022), p. 97.

[143] Julian Schlechtriemen et al. "When will it change the lane? A probabilistic regression approach for rarely occurring events". In: *2015 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2015, pp. 1373–1379.

[144] Robin Schubert, Eric Richter, and Gerd Wanielik. "Comparison and evaluation of advanced motion models for vehicle tracking". In: *2008 11th international conference on information fusion*. IEEE. 2008, pp. 1–6.

[145] John Schulman et al. "Finding Locally Optimal, Collision-Free Trajectories with Sequential Convex Optimization." In: *Robotics: science and systems*. Vol. 9. 1. Citeseer. 2013, pp. 1–10.

[146] John Schulman et al. "Motion planning with sequential convex optimization and convex collision checking". In: *The International Journal of Robotics Research* 33.9 (2014), pp. 1251–1270.

[147] George AF Seber and Christopher John Wild. "Nonlinear regression. hoboken". In: *New Jersey: John Wiley & Sons* 62 (2003), p. 63.

[148] Sepanta Sekhavat et al. "Multilevel path planning for nonholonomic robots using semiholonomic subsystems". In: *The international journal of robotics research* 17.8 (1998), pp. 840–857.

[149] Xu Shen et al. "Collision avoidance in tightly-constrained environments without coordination: a hierarchical control approach". In: *arXiv preprint arXiv:2011.00413* (2020).

[150] Xu Shen et al. "Parkpredict: Motion and intent prediction of vehicles in parking lots". In: *2020 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2020, pp. 1170–1175.

[151] Vikas Sindhwani, Rebecca Roelofs, and Mrinal Kalakrishnan. "Sequential operator splitting for constrained nonlinear optimal control". In: *2017 American Control Conference (ACC)*. IEEE. 2017, pp. 4864–4871.

[152] Arun Kumar Singh et al. *Inducing Multi-Convexity in Path Constrained Trajectory Optimization for Mobile Manipulators*. 2019. arXiv: 1904.09780 [cs.RO].

[153] Avi Singh et al. "End-to-end robotic reinforcement learning without reward engineering". In: *arXiv preprint arXiv:1904.07854* (2019).

[154] P. Spellucci. "A new technique for inconsistent QP problems in the SQP method". In: *Mathematical Methods of Operations Research* 47 (1998), pp. 355–400.

[155] Anthony Stentz. "Optimal and Efficient Path Planning for Partially-Known Environments". In: *IN PROCEEDINGS OF THE IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION*. 1994, pp. 3310–3317.

[156] Anthony Stentz. *Optimal and efficient path planning for unknown and dynamic environments*. Tech. rep. CARNEGIE-MELLON UNIV PITTSBURGH PA ROBOTICS INST, 1993.

[157] Samantha Stoneman and Roberto Lampariello. "Embedding nonlinear optimization in RRT* for optimal kinodynamic planning". In: *53rd IEEE Conference on Decision and Control*. IEEE. 2014, pp. 3737–3744.

[158] Xiaoxun Sun, William Yeoh, and Sven Koenig. "Moving target D* lite". In: *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*. 2010, pp. 67–74.

[159] Yuval Tassa, Tom Erez, and Emanuel Todorov. "Synthesis and stabilization of complex behaviors through online trajectory optimization". In: *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE. 2012, pp. 4906–4913.

[160] Yuichi Tazaki, Hiroyuki Okuda, and Tatsuya Suzuki. "Parking trajectory planning using multiresolution state roadmaps". In: *IEEE Transactions on Intelligent Vehicles* 2.4 (2017), pp. 298–307.

[161] Shantanu Thakar et al. "Accelerating bi-directional sampling-based search for motion planning of non-holonomic mobile manipulators". In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2020, pp. 6711–6717.

[162] Sebastian Thrun et al. "Stanley: The robot that won the DARPA Grand Challenge". In: *Journal of field Robotics* 23.9 (2006), pp. 661–692.

[163] Dawn Tilbury, Richard M Murray, and S Shankar Sastry. "Trajectory generation for the n-trailer problem using Goursat normal form". In: *IEEE Transactions on Automatic Control* 40.5 (1995), pp. 802–819.

[164] Philipp Töws and Dieter Zöbel. "Reversing General 2-Trailer Vehicles Using a 2D Steering Function Model and a Novel Mesh Search Algorithm". In: *2021 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2021, pp. 1274–1281.

[165] Chris Urmson et al. "Autonomous driving in urban environments: Boss and the urban challenge". In: *Journal of field Robotics* 25.8 (2008), pp. 425–466.

[166] Jur Van Den Berg and Mark Overmars. "Kinodynamic motion planning on roadmaps in dynamic environments". In: *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2007, pp. 4253–4258.

[167] Andreas Wachter. "An interior point algorithm for large-scale nonlinear optimization with applications in process engineering". PhD thesis. Carnegie Mellon University, 2002.

[168] Changhao Wang, Jeffrey Bingham, and Masayoshi Tomizuka. *Trajectory Splitting: A Distributed Formulation for Collision Avoiding Trajectory Optimization*. 2021. arXiv: 2111.01899 [cs.RO].

[169] Yebin Wang. "Improved A-search guided tree construction for kinodynamic planning". In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 5530–5536.

[170] Dustin J Webb and Jur Van Den Berg. "Kinodynamic RRT*: Asymptotically optimal motion planning for robots with linear dynamics". In: *2013 IEEE International Conference on Robotics and Automation*. IEEE. 2013, pp. 5054–5061.

[171] A Woch, R Plamondon, and C O'Reilly. "Kinematic characteristics of successful movement primitives in young and older subjects: a delta-lognormal comparison". In: *Hum. Mov. Sci* 30.1 (2011), pp. 1–17.

[172] Wenda Xu et al. "A real-time motion planner with trajectory optimization for autonomous vehicles". In: *2012 IEEE International Conference on Robotics and Automation*. IEEE. 2012, pp. 2061–2067.

[173] Liangjun Zhang and Dinesh Manocha. "An efficient retraction-based RRT planner". In: *2008 IEEE International Conference on Robotics and Automation*. IEEE. 2008, pp. 3743–3750.

[174] Xiaojing Zhang, Alexander Liniger, and Francesco Borrelli. "Optimization-based collision avoidance". In: *IEEE Transactions on Control Systems Technology* 29.3 (2020), pp. 972–983.

[175] Xiaojing Zhang et al. "Autonomous parking using optimization-based collision avoidance". In: *2018 IEEE Conference on Decision and Control (CDC)*. IEEE. 2018, pp. 4327–4332.

[176] Matt Zucker et al. "An optimization approach to rough terrain locomotion". In: *2010 IEEE International Conference on Robotics and Automation*. IEEE. 2010, pp. 3589–3595.

[177] Matt Zucker et al. "Chomp: Covariant hamiltonian optimization for motion planning". In: *The International Journal of Robotics Research* 32.9-10 (2013), pp. 1164–1193.