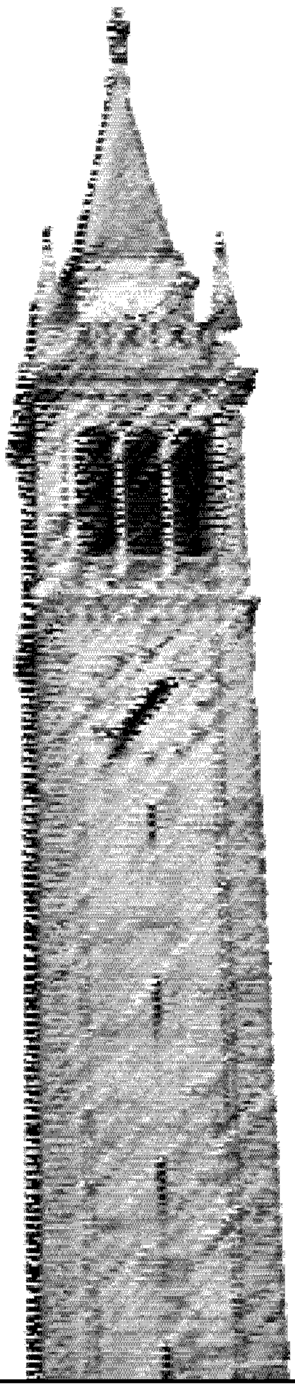
Safe and Data-Efficient Learning for Robotics: A Control Theoretic Approach



Somil Bansal

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2020-186
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2020/EECS-2020-186.html

November 24, 2020

Safe and Data-Efficient Learning for Robotics: A Control Theoretic Approach

by

Somil Bansal

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Electrical Engineering and Computer Sciences

in the

Graduate Division

of the

University of California, Berkeley

Fall 2020

Safe and Data-Efficient Learning for Robotics: A Control Theoretic Approach

Abstract

Safe and Data-Efficient Learning for Robotics: A Control Theoretic Approach

by

Somil Bansal

Doctor of Philosophy in Electrical Engineering and Computer Sciences

University of California, Berkeley

Professor Claire Tomlin, Chair

For successful integration of autonomous systems such as drones and self-driving cars in our day-to-day life, they must be able to quickly adapt to ever changing environments, and actively reason about their safety and that of other users and autonomous systems around them. Even though control theoretic approaches have been used for decades now for the control and safety analysis of autonomous systems, these approaches typically operate under the assumption of a *known* system dynamics model and the environment in which the system is operating. To overcome these challenges, machine learning approaches have been explored to operate autonomous systems intelligently and reliably in unpredictable environments based on prior data. However, learning techniques widely used today are extremely data inefficient, making it challenging to apply them to real-world physical systems. Moreover, unlike control theoretic approaches, these techniques lack the necessary mathematical framework to provide guarantees on correctness, causing safety concerns as data-driven physical systems are integrated in our society.

This dissertation aims to combine the control theoretic perspective with the modern learning approaches to enable autonomous systems to safely adapt to unknown environments. It first introduces a suite of core tools from dynamical system theory and robust control that permits the safety analysis of single and multi-agent autonomous systems under the assumption of known system model and environment. In the remainder of the dissertation, we discuss how we can go past these assumptions with the help of machine learning, while minimizing the data requirement for learning and maintaining the safety guarantees for the autonomous system.

To that end, we first discuss how we can learn inaccuracies in the dynamics model of the system, such as unknown ground effects for quadrotors, and use the learned model along with optimal control tools to improve the control performance. Our particular focus is on learning task-specific models that allow for a quick adaptation for the task at hand; these techniques are showcased on physical quadrotors and robotic arms. We next present modular architectures that use a learning-based perception module for the environment level reasoning and a dynamics model-based module for system level reasoning to solve challeng-

ing perception and control problems in *a priori unknown* environments in a data-efficient fashion. Moreover, due to their modularity, these architectures are amenable to simulation-to-real transfer, and can be used for different robotic systems without any retraining. These approaches are demonstrated on a variety of ground robots navigating in unknown buildings around humans based only on onboard visual sensors.

Next, we discuss how we can use dynamics models not only for data-efficient learning, but also to monitor and recognize the learning system's failures, and to provide online corrective safe actions when necessary. This allows us to provide *safety* assurances for learning-enabled systems in unknown and human-centric environments, which has remained a challenge to date. Together these techniques enable autonomous systems to learn to operate in unknown environments, but do so in a data-efficient and safe fashion. The dissertation ends with a discussion of future challenges and opportunities at the intersection of learning and control, including the safety analysis in online learning settings and the need to close the loop between the design of learning systems and their safety analysis for developing resilient and continually improving autonomous systems.

To my parents, Sh. Raman and Punam Bansal, and to my grandma Sh. Kasturi Mahipal.

Contents

List of Figures

List of Tables

Acknowledgments

The research work presented in this dissertation is far from just being my achievement, but rather that of all the amazing collaborators, teachers and professors, mentors, friends, and family members that I am fortunate to have in my life.

First and foremost, I am grateful to my wonderful PhD advisor, Prof. Claire Tomlin, for her mentorship. I can write pages and pages on how lucky I am to have Claire as my advisor, but unfortunately, I am constrained to express my gratitude in just a few sentences here. Claire kept me going through the highs and the lows of my PhD journey – she always supported me during my research adventures whether it is trying to steer through the unknown territory of learning and control, or setting up a hardware testbed from scratch for an entire year. Claire is the kind of advisor who makes you believe in yourself and sees your strengths that you yourself can't see. She is not just an excellent mentor, but also an excellent teacher and above all a wonderful human being. As I start my own journey as an assistant professor, I can only hope to be an advisor like Claire one day, and help others grow in a similar fashion.

I also owe gratitude to the other two members of my dissertation committee: Sanjit, for teaching me how to always keep the bigger picture in mind; and Koushil, for nurturing my fascination with control theory and answering my million questions about Lyapunov theory.

Achieving the objectives of this dissertation has required bridging different areas of research. This goes beyond understanding the fundamentals of control theory to its intersection with computer vision, formal methods, reinforcement learning, and robotics. This would not have been possible without the support of Shankar, for always encouraged me to think outside the box; Jitendra, for nurturing my excitement with computer vision; Sanjit and Alberto, for helping me understand the beautiful world of formal methods; Sergey, Pieter, and Anca, for being patient with me as I pick their brain about robotics.

The insights that I have obtained during my research are the results of the phenomenal collaborations that I have had over years. I am thankful to Mo and Sylvia for exploring the world of Hamilton-Jacobi reachability with me, for proving and disproving each other's conjectures, for countless lunches together on Euclid Avenue, for all the conference travels together, for their genuine friendship that I can always count on, and for our virtual tea parties during these strange time of COVID-19 pandemic. Thanks to Andrea Bajcsy for some of the best collaborations I have ever had, for the energy that she brings to a project, for the stimulating walks around the campus, for her aesthetic touches to my research presentations, and last but not the least, for introducing me to the fine bakeries of Berkeley. I am grateful to the *robot whisperers*, Andrea Bajcsy, Ellis Ratner, Varun Tolani, Forrest Laine, and Frank Jiang who made working on real robots so much fun; to Kene Akametalu, for exploring the intersection of learning and control with me, for all the late night philosophical discussions, and for always having my back; to Shromona Ghosh, for teaching me the basics of formal methods and for helping me understand the core safety problems in AI; to Jaime Fisac, for the discussions about multi-agent systems, learning, safety, humans, and how to bring them all together; to Saurabh Gupta, for being patient with me and answering my questions about the basics of computer vision; and to Roberto Calandra, for helping me getting started with the intersection of learning and control. Thanks also to Thomas Beckers for coping with my spotty presence in the last year and yet making me feel like an equal contributor, and for introducing me to the nice beer gardens of Freiburg. I am also thankful to the younger students who coped with my crazy ideas and very far from perfect research guidance: Varun Tolani, Anjian Li, Ted Xiao, Eli Bronstein, Georgios Giovanis, Khalil Sarwari, Lucas Medino, Nathan Blair, Jonathan Lee, Frank Jiang, Chuck Tang, and Jingqi Li – it was a wonderful experience to work with these extremely talented and smart people.

I have been fortunate to have wonderful mentors during my PhD years: Adam Bry, Katie Driggs-Campbell, Aleksandra Faust, Hayk Martiros, Nikolai Matni, Ian Mitchell, Lillian Ratliff, Dorsa Sadigh, and Insoon Yang have all held my hand through the ups and downs of the graduate school life, academic job applications, and research direction uncertainties. It is such a relief to know that I can still count on the guidance of these wonderful people even after my graduation. Special thanks to my mentor and collaborator, Melanie Zeilinger, whose guidance is one of the primary reasons for why I even considered applying to the graduate school.

I cannot even imagine how hard my graduate student life would have been without the three angels – Shirley Salanio, Jessica Gamble, and Angie Abbatecola. They have made my life so much easier, often at the expense of making their own life harder, be it helping me with my countless petitions and getting them approved, helping me with the immigration stuff, setting up meetings, filing reimbursements, or managing my recommendation letters. And the most remarkable thing about them is that they always did all these things with a smile on their face. Thank you so much for all your support in my PhD journey.

As I leave Berkeley for my next steps, I will dearly miss these amazing friends that I am fortunate to share my graduate school journey with – Eric Mazumdar, Roel Dobbe, Anusha Nagabandi, Carlos Florensa, Carolyn Matl, Matthew Matl, Nick Antipa, Esther Rolf, Ashish Kumar, David Fouhey, Angjoo Kanazawa, Amir Zamir, Pulkit Agrawal, Deepak Pathak, Niladri Chatterji, Patricia Hidalgo-Gonzalez, David Fridovich-Keil, Vicenç Rubies-Royo, Joe Menke, Greg Kahn, and the rest of the TRUST, SDH7, and Cory 301 inhabitants; thanks to all of them for great discussions about life, research, Berkeley, and really just about everything. Also, thanks to the members of Hybrid Systems group, Semiautonomous group, and Compvision group – our weekly research discussions was one of the things that I looked forward to most during my PhD. A special shout-out to our *Indian gang* who has been my family away from home over these past eight years – Abhinav Prateek, Varun Mishra, Sakshi Jain, Vivek Mishra, Abhishek Kar, Sonam Gupta, Garvit Juniwal, Tanvi Lall, Julia Nee, Smeet Bhatt, Varsha Padhee, Avani Goyal, and Shubham Tulsiani. It is even hard for me to imagine my life here without you all.

And this journey wouldn't have even started without the support of my family. I am from a background in India where higher education is an exception rather than a norm – I am ever grateful to my parents Raman and Poonam Bansal, and my uncle Niraj Khadaria for their unconditional support towards my dreams. To my four grandparents for their selfless love. To my brother Ritesh Bansal for always helping me think through my problems; talking to him constantly reminds me that he probably understands me better than myself. To my sister-in-law Shanaya Bansal for never failing to cheer me up whenever I am feeling down. To Vijay Bansal (*Ramu Bhaiya*) for working tirelessly since my middle school to make sure that I have access to the best education. Without the selfless love of these people and their constant encouragement, I wouldn't have even dreamt of pursuing a PhD.

Chapter 1

Introduction

The era of autonomous systems will transform our lives in more ways than one – autonomous cars will drive us from our homes to offices; autonomous drones will deliver our Amazon packages; assistive robots will clean our houses, and the list goes on and on. Other than these civil applications that will increase human efficiency and productivity, autonomous systems can drastically improve safety for employees in high-risk work environments. For example, an autonomous drone can easily inspect bridges and nuclear reactors, whereas the same tasks might pose a high risk for humans.

Even though autonomous systems present a vast range of opportunities for our society, their successful integration in our day-to-day life pose some key challenges. First, future autonomous systems will need to constantly operate in and adapt to *a priori* unknown environments. For example, an autonomous car will need to drive around construction sites and blocked lanes in the city, in crowded city downtowns around humans, and in different types of weathers such as sunny, rainy, and snowy. And all these environment conditions cannot be known beforehand. Second, a lot of these autonomous systems are safety-critical systems, and deploying them in the real-world without a diligent safety analysis can lead to catastrophic failures. So an important question to answer is *"how can autonomous systems quickly and safely adapt to a priori unknown environments to achieve their goal?"*

If we think from a historical perspective, robots and autonomous systems are not any new concepts for humans – thermostats in our homes all the way to the car assembly lines and highly safety-critical systems such as aeroplanes are all autonomous systems. In fact, control theoretic approaches have been used for decades now for control and safety analysis of such autonomous systems. However, to develop these control and safety algorithms, we often need to have a very good understanding of the environment and conditions in which these systems will operate. Once these conditions are known, control theoretic approaches make sure that the system objective is safely achieved under those conditions. But these approaches are challenged when the autonomous system must operate outside of the assumptions of *known* system model and environment and need to adapt to their environments, as is the case for future autonomous systems.

One way to enable this adaptation in an autonomous system is based on its prior data and experience in similar environments. Data-driven adaptation, studied primarily under the banner of machine learning, has led to tremendous progress in domains such as computer vision, speech recognition, and natural language processing. Fueled by these advances, machine learning approaches are now being explored to develop autonomous systems that can operate intelligently and reliably in new, unknown environments. However, learning techniques widely used today are extremely data inefficient, making it challenging to apply them to real-world physical systems. Moreover, they lack the necessary mathematical framework to provide guarantees on correctness, causing safety concerns as data-driven physical systems are integrated in our society.

The key philosophical foundation of this thesis is that *we should certainly leverage the adaptability of the modern data-driven approaches, but marry them with the classical, control theoretic approaches that have been used for decades to control autonomous systems reliably in controlled environments.* In this dissertation, we demonstrate how we can combine tools from robust optimal control theory with machine learning and computer vision to develop data-efficient and provably safe learning-based control algorithms for physical robotic systems. The presence of learning in our algorithms will enable adaptability in *a priori* unknown and hard-to-model environments; the presence of control theoretic tools will allow us to reduce the sample complexity of the learning component and to actively reason about safety of the system.

Part 1: Safety Analysis for Robotic Systems. The first part of this dissertation focuses on developing robust control algorithms for scalable safety analysis of autonomous systems. This safety analysis is primarily based on Hamilton-Jacobi-Isaacs (HJI) reachability analysis. The HJI reachability analysis is an important formal verification method for guaranteeing safe operation of dynamical systems that provides both the set of safe states and the corresponding safe controller for general nonlinear system dynamics. The main challenge is to scale the HJI analysis to real-world autonomous systems because of its exponential computational complexity with respect to the number of state variables. In the first part, we will address this challenge on multiple fronts by leveraging (a) the structure in dynamics and control strategy, (b) smart offline computations, and (c) modern computational tools to perform the HJI analysis tractably for single and multi-agent autonomous systems. For example, on the computation front, we introduce the Berkeley Efficient API in C++ for Level Set methods (BEACLS), a C++-based reachability toolbox that can leverage modern computational tools such as GPUs to improve computation speed of HJI reachability by nearly 100 times compared to existing implementations (Chapter 3). On the algorithmic front, rather than restarting the safety analysis from scratch, we propose a method of "warm-start" reachability, which uses a user-defined initialization (typically a previously computed solution). By warm-starting an HJI value function, convergence may take significantly fewer iterations (Chapter 3). We demonstrate the potential of these advances for safe, large-scale, multi-vehicle trajectory planning problems, through a problem in autonomous drone delivery (Chapter 4).

Part 2: Going Beyond Known Dynamics Models and Environments: Learning-Based Control for *Unknown* Models and Environments. The safety analysis algorithms in the first part are developed under the assumption of known system dynamics models and known environments in which this system is operating. In this part of the thesis, we focus on developing mathematical tools and algorithms that combine learning and control theoretic methods for controlling autonomous systems when a system dynamics model is not fully known and/or when the system is operating in an unknown environment and needs to make decisions based on onboard perception sensors.

To capture inaccuracies in the dynamics model, we discuss indirect and direct learning-based control algorithms, very much motivated by direct and indirect adaptive control algorithms. Indirect learning-based control approaches use machine learning tools, such as Deep Neural Networks (DNNs) and Gaussian Processes (GPs), to learn a residual dynamics model of the system directly based on the data collected on the system (Chapter 5). This dynamics model is then used with optimal control schemes to improve the control performance. Direct learning-based approaches take a task-specific approach to dynamics modeling, wherein we combine Bayesian optimization and optimal-control in a closed-loop to develop aDOBO (Dynamics Optimization via Bayesian Optimization), a framework for learning unmodeled effects that are specific to control task at hand (Chapter 6). Unlike traditional system identification approaches, aDOBO does not necessarily find the most accurate dynamics model; instead, it learns a "coarse" model that can be learned with a small amount of data, and yet yields the best closed-loop controller performance when provided to the optimal control method used. We demonstrate aDOBO on 3-DoF robotic arm, which is tasked to stir jelly in a given pattern. Rather than accurately modeling the complex nonlinear fluid dynamics, which can be quite challenging, aDOBO leverages learning and optimal control for efficiently completing the task with a very high accuracy. We finally combine direct and indirect learning-based control approaches to obtain a high-performing and robust dynamics model for the system.

In many applications of interest, simple and well understood dynamics models are sufficient for control, and it is rather the vision and perception components that require learning, such as to navigate in *a priori* unseen environments. Typically, a geometric map of the environment is used for navigation; however, real-time map generation can be challenging in texture-less environments or in the presence of transparent, shiny objects, or strong ambient lighting. In contrast, pure learning approaches, such as end-to-end learning, side-step this explicit map estimation step, but suffer from data inefficiency and lack of robustness. We take a factorized approach to robot navigation that uses a deep learning-based perception module for environment level reasoning and a dynamics model-based module for system level reasoning. More specifically, we train a Convolutional Neural Network (CNN) that uses the RGB image observations obtained from the onboard camera to produce a sequence of intermediate waypoints, which are used as targets for a model-based optimal controller to generate smooth, dynamically feasible, and collision-free trajectories to be executed on the robot (Chapter 7). Leveraging learning allows the robot to navigate in completely new buildings based only on the onboard RGB camera. Leveraging underlying dynamics and feedback-based control not only accelerates learning, but also leads to trajectories that are robust to variations in physical properties and noise in actuation. Through simulations and experiments on a mobile robot, we demonstrate that this modular approach is *better* (more successful at reaching the goals), more *efficient* at reaching the goals (takes less time), and results in *smoother* trajectories (less jerk), as compared to end-to-end learning. Due to the real-world imperfections in depth measurements, the proposed approach is more *reliable* (more successful) than geometric mapping-based approaches, as it does not explicitly rely on a map. Thanks to the presence of the visual and model-based feedback in the closed-loop, this modular approach can be *directly* transferred from simulation to unseen, real-world, human-centric environments without any finetuning or data collection in the real-world.

Part 3: Safety for Learning-Enabled Control Systems. The previous part of this dissertation focuses on the “control aspect” of autonomous systems that are operating in unknown environments. By combining learning and control approaches, we design data-efficient learning-based control algorithms for autonomous systems. In the last part of this dissertation, we bring back safety into the picture.

As the autonomous system evolves via learning-in-the-loop, the safety assurances need to be *evaluated* and *updated* at operation-time. This becomes particularly challenging when the system is operating in an unknown environment where even the unsafe states (such as obstacles) are not known *a priori*, such as navigation in an unseen environment. In such cases, rather than verifying the learning-enabled perception component explicitly, which can be quite challenging, we develop a HJI reachability-based framework to monitor the output of the perception module to recognize any failure and provide a corrective safe action when necessary (Chapter 9). Building on the scalable HJI analysis tools that we developed in the first part of this dissertation, we update both the monitor as well as the corrective actions in real-time as the system is operating in the environment. We deploy this framework on the mobile robot with the learning-based perception-action loop presented in the previous part, but now also actively ensure safety of the system.

The safety analysis of the system is still only as accurate as the dynamics model of the system. To close the gap between the system and the model, we also propose a data-driven verification approach that can provide both probabilistic safety guarantees and a safety controller for the actual system based on that for the model and the collected data (Chapter 8).

Together, these three parts present contributions that enable autonomous systems to adapt to unknown environments, but do so in a data-efficient and safe fashion. The research outlined in this dissertation has also opened many promising future research directions. I conclude this dissertation with a discussion on some of these directions, including how we can bridge the model-based and statistical verification techniques for scalable safety analysis of data-driven systems; how we can ensure the safety of data-driven systems in online learning settings where the control-loop itself might change over time; and finally, how we can close the loop between the design and analysis of learning-enabled systems by using model-based control not only for the analysis, but also for actively gathering safety-critical data samples.

Chapter 2

Background and Preliminaries

The goal of this chapter is to set up the mathematical preliminaries and definitions that will form the basis for combining learning and control. My hope is to present the relevant background content from the learning and control literature in a unifying light to further highlight the connections between the two. Throughout the chapter, I mark some sections with a star (*). These can be skipped based on reader's interest without affecting the overall flow.

2.1 System Dynamics and Feedback Control

At its core, control theory is the theory of manipulating a system to achieve a desired behavior out of the system. Let's try to understand the basic idea behind control theory through a very simple system that we all use on a regular basis: a room heater. Here our *system* that we want to manipulate is the heater and the *desired behavior* is to maintain a given temperature in the room. To achieve the objective, a thermostat (*controller*) employs a very simple *control strategy* – turn on the heater if the temperature drops below the desired temperature and turn it off if the temperature in the room is higher than the desired. Even though a thermostat employs a very simple switching control strategy to achieve the desired behavior in this case, these strategies become more and more complex as the system and the situations they operate in become more complex. For example, another control system that we might have experienced at some point is adaptive cruise control in a car. The objective of a cruise control system is to maintain a desired speed. To do this, the cruise system constantly adjusts the speed of the car depending on the curves or bumps in the road, such as accelerating when going uphill to counter gravity and decelerating when going downhill.

Over decades, control theorists have developed general representations to describe such systems, designing control strategies for manipulating these systems, as well as tools to analyze the resulting control strategies. One such representation for dynamical systems that is particularly conducive for describing robotic systems is the *state-space representation.*

2.1.1 State-Space Representation for Dynamical Systems

In state-space form, a dynamical system consists of a state $x \in \mathbb{R}^{n_x}$, a control input $u \in \mathbb{R}^{n_u}$, and a disturbance input $d \in \mathbb{R}^{n_d}$. Here, $\mathbb{R}^{n_x}$ is called the state-space of the system and n_x is the dimension of the state-space. In general, the state could include the position and orientation of a ground vehicle or the joint angles of a human torso and robot manipulator. The evolution of these states over time can be described by an ordinary differential equation:

$$\dot{x} = f(x, u, d), \tag{2.1}$$

where f is a map from $\mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_d} \rightarrow \mathbb{R}^{n_x}$. $\dot{x}$ represent the time derivative of state, $u \in \mathbb{R}^{n_u}$ is system's control input, and $d \in \mathbb{R}^{n_d}$ is disturbance in the state evolution. In general, d can either represent actual external disturbances, such as winds in case of aerial vehicles, or any unmodeled physical effects that are not explicitly taken into account in f. For example, a simple model of a ground robot may not model friction between its tires and the ground. Such effects can then be taken into account by modeling them as disturbances in system dynamics. When no disturbances are present in system dynamics, we simply omit d from the arguments of f in (2.1).

A note on time and trajectories. Note that state, control, and disturbance in (2.1) are all functions of time. To make the dependence on time explicit, we sometime denote the state at time t as $x(t)$ or x_t. The former is typically more common for continuous time systems and the latter for discrete time systems; however, they are often used interchangeably as well. $u(t)$ (or u_t) and $d(t)$ (or d_t) are similarly defined. Whenever the dependence on time is clear, we omit the time argument for brevity purposes. Often times, we will be interested in defining the state and control trajectories over time. In such cases, we use $x(\cdot)$ (respectively $u(\cdot)$) to denote a general state (respectively control) trajectory over time. More formally, the trajectory of a system over time is represented as $\xi(\tau; x_0, t, u(\cdot), d(\cdot))$. This notation can be read as the state achieved at time τ by starting at initial state x_0 and initial time t, and applying control input $u(\cdot)$ and disturbance input $d(\cdot)$ over the time interval $[t, \tau]$. For compactness we will refer to trajectories using $\xi_{x,t}^{u,d}(\tau)$. Note that as per the above notation $x(\tau)$, $\xi(\tau; x_0, t, u(\cdot), d(\cdot))$, and $\xi_{x,t}^{u,d}(\tau)$ are all equivalent ways of denoting the system state at time τ; the latter simply contain more information on the initial state, control input, and the disturbance experienced by the system while getting to the state $x(\tau)$.

Existence and uniqueness of the state trajectory*. There are some important theoretical questions that beg our attention at this point. For example, does there always exist a solution to the differential equation in (2.1)? If not, what does it even mean to define the state trajectory? In other words, when is the model in (2.1) well posed?

To address these questions recall that f is a map from $\mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_d} \rightarrow \mathbb{R}^{n_x}$. It can be shown that if f is uniformly continuous in its arguments and Lipschitz continuous in x for any given u and d, then there exists a unique solution of these system dynamics for a given control and disturbance trajectory $u(\cdot), d(\cdot)$ [85]. In other words, the system (state) trajectory is well defined under these assumptions. Moreover, the resultant state trajectory $\xi(\cdot; x_0, t, u(\cdot), d(\cdot)) : \mathbb{R} \to \mathbb{R}^{n_x}$ is differentiable and satisfies (2.1) almost everywhere. Mathematically, the following conditions are satisfied by the state trajectory:

$$\frac{d}{ds}\xi(s; x_0, t, u(\cdot), d(\cdot)) = f\left(\xi(s; x_0, t, u(\cdot), d(\cdot)), u(s), d(s)\right), \tag{2.2}$$

$$\xi(t; x_0, t, u(\cdot), d(\cdot)) = x_0 \tag{2.3}$$

From here on, we assume that these well posedness assumptions are satisfied by f, unless stated otherwise.

2.1.2 Discrete-Time Dynamical Systems

In Sec. 2.1.1, we discussed the state-space representation of dynamical systems in continuous time. However, discrete time notation is more widely used in reinforcement learning (RL) and robotics communities. The dynamics in discrete time can be represented as:

$$x_{k+1} = f(x_k, u_k, d_k), \quad k \in \mathbb{N}. \tag{2.4}$$

As before, x_k, u_k, and d_k represent the state, control, and disturbance of the system at timestep k.

It is also a common practice in RL/robotics to obtain the discrete time dynamics using an Euler approximation of the continuous time dynamics in (2.1) with some time step ΔT. In particular, the discretized dynamics $f_{discrete}$ can be obtained as:

$$f_{discrete}(x_k, u_k, d_k) = x_k + \Delta T.f_{continuous}(x_k, u_k, d_k). \tag{2.5}$$

This process is often referred to as "discretization of dynamics" colloquially. The discretized system in this case is sometimes also referred to as *sampled data system*. In (2.5), we have obtained a first order discretization of system dynamics; however, higher order discretizations can also be used for more accuracy.

In discrete time, it is more common to use the word state and control *sequences* rather than *trajectories*. In particular, we use $\mathbf{x}_k^T$ to denote the state sequence starting from timestep k through T, i.e., $\mathbf{x}_k^T = [x_k, x_{k+1}, \ldots, x_T]$. $\mathbf{u}_k^T$ is similarly defined. When the final timestep is clear, we omit T from $\mathbf{x}_k^T$.

2.1.3 The Magic of Feedback

Feedback control is one of the foundational concepts in control theory. A dynamics model of the system allows us to find a controller (a control sequence or trajectory) to achieve a desired system response. There are several ways to design such a controller, some of which we will discuss later in this chapter and throughout this dissertation. However, more often than not, when this controller is applied on the actual system, one may not necessarily get the desired response, simply because of inevitable modeling errors or presence of external disturbances. Feedback is a mechanism to counter the effect of these inaccuracies in the dynamics model to still achieve the desired system performance.

The concept of feedback can probably be best understood through an example. Imagine walking from point A to point B through a cluttered living room in your house without bumping into anything, except that now you can only see the room once at point A and then you need to walk with your eyes closed. Since you can see the room at point A, in theory, you can plan a path (or "state" trajectory) that is collision free. For simplicity, let's assume that the planned path is represented as steps, i.e., 5 steps forward, 2 steps left, 6 steps forward, and so on. However, once you start walking along this path with your eyes closed, it is very likely that you will not take the desired steps exactly and thus you will deviate from the planned path over time, and may ultimately collide into something or reach some other position. This control mechanism is called *open-loop* control, wherein a control trajectory or sequence is determined at the initial time and followed for the rest of the duration, completely ignoring the actual system state during the controller execution.

Now imagine doing the same task with your eyes open. Why are we more likely to succeed in this case? It is simply because if we take the first 5 steps and observe that we are not at the location that we should be at, we take additional steps to get to that desired location. In control theory language, we *observe* (through eyes in this case) our *state* (position in this case), and if we are not at the desired state, we apply *control* (steps in this case) based on this state feedback. This control mechanism is called *feedback* or *closed-loop* control, wherein a control trajectory is adapted during the execution based on the received observations. Feedback control is what making it possible to fly our aircraft or drones in sky, where they will inevitably experience unmodeled wind flows and turbulence that would otherwise deviate the vehicle from its desired path. How often we need to apply feedback depends heavily on the system and the inaccuracies in the dynamics model; for example, a faster feedback is often required for unstable systems such as bipedal robots and helicopters, whereas a slower feedback is often sufficient for wheeled robots.

There are two different feedback mechanisms that are popular in control community: state feedback and output feedback. In output feedback, the observations (also called *system output*) are directly used to adapt the control commands. In contrast, in state feedback, the observations are first used to estimate the state of the system (also called *state estimation*) and then the control commands are adapted based on the estimated state. In some cases, however, it is non-trivial to estimate the state from the observations, particularly when observations are high-dimensional such as camera images or videos. In fact, developing algorithms for processing these high-dimensional perceptual observations for feedback control is one of the major contributions of this dissertation; we will discuss this further in Chapter 7.

2.2 Optimal Control and Dynamic Games

The optimal control problem refers to the problem of optimal decision making for a dynamical system. When there are multiple decision makers (also called *players*) in the system, the problem of optimal decision making takes the form of a dynamic game (Section 2.2.4).

Optimal control and dynamic games formulations can be posed both in continuous time as well as in discrete time. Here, we provide their overview primarily in continuous time, but arguably, it is more common in the robotics literature to formulate and solve optimal control problems in discrete time. In fact, we also heavily leverage discrete-time optimal control throughout this work. We will comment on discrete-time optimal control problem as well as a popular approach to solve this problem called Model Predictive Control (MPC) in Section 2.2.3.1.

2.2.1 The Optimal Control Problem

The problem of optimal decision making, or *optimal control problem*, arises naturally in a variety of control and robotic settings. For example, the problem of a robot navigating from point A to point B as quickly as possible, finding minimum energy walking gaits for a bipedal robot, or performing an acrobatic maneuver on an autonomous helicopter can all be (and have been) formulated as optimal control problems. Mathematically, the optimal control problem consists of minimizing a cost function subject to the dynamics, state, and control constraints. To understand this, let's go back to the problem of a robot navigating from point A to point B as quickly as possible. Here, the cost function is the time taken by the robot trajectory. The robot has some physical constraints that it needs to satisfy; these constraints define its dynamics. The motors that are controlling the robot have limited power so there are some control input constraints. Finally, there might be some obstacles in the environment that the robot should avoid; these obstacles define some position constraints (or state constraints more generally) for the robot.[1] We now formally define an optimal control problem using the state-space framework discussed in Section 2.1.1. For the discussion of optimal control in this section, we assume that there are no disturbances in the system dynamics. We will add the disturbance back in the dynamics in Section 2.2.4.

As we discussed, the goal of optimal control problem is to minimize a cost function, subject to the dynamics and the control constraints. Mathematically, we are interested in solving the following optimization problem:

$$\inf_{u(\cdot)} J_t(x_0, u(\cdot)) \equiv \int_t^T L(x(s), u(s))ds + M(x(T)), \tag{2.6}$$

$$\text{subject to} \quad \dot{x}(s) = f(x(s), u(s)), \quad \forall s \in [t, T] \tag{2.7}$$

$$u(s) \in \mathcal{U} \tag{2.8}$$

[1]In fact, this particular problem is so common in robotics that it has a special name and research subfield dedicated to it – the trajectory planning problem.

Here, J_t is the cost accumulated over the time horizon $[t, T]$ starting from state x_0 at time t and applying the control trajectory $u(\cdot)$. $L(\cdot)$ is called running cost and is accumulated over the entire time horizon. $M(\cdot)$ is referred to as terminal cost, as it is only dependent on the terminal state. When there is no terminal cost, $M(\cdot)$ can be set to zero. Similarly, if there is no running cost, $L(\cdot)$ can be set to zero. The cost function should be minimized subject to the dynamics constraints (Equation (2.7)). Additionally, there might be control bounds or constraints imposing a constraint of the form (2.8), where $\mathcal{U}$ is the set of feasible controls.

Remark 1 *Note that even though we have an additive structure to the cost function in* (2.6)*, many optimal control problems of interest, including Hamilton-Jacobi reachability problem, do not adhere to that structure. In other cases, the control bounds can be time and state dependent time well. Finally, there might be some state constraints. We are ignoring these generalizations for brevity. Nevertheless, the discussion to follow is quite general and can be applied to the "non-standard" optimal control problems as well.*

2.2.2 Dynamic Programming and Hamilton-Jacobi-Bellman PDE

Many different methodologies have been developed over years to solve optimal control problems. I cannot possibly even attempt to provide a comprehensive overview of these approaches here; however, intuitively, there are two predominant views to solve optimal control problems: the calculus of variations or "optimization" viewpoint and the dynamic programming viewpoint.

The calculus of variations treats optimal control problem as an optimization problem. Intuitively, this process is as follows: first, the constraints are used to form a Lagrangian, which then results in an unconstrained optimization problem over control trajectory. Next, the first order necessary conditions are obtained for this unconstrained optimization problem. The Lagrange multipliers as well as the optimal control trajectories can be obtained by solving these first order conditions. The advantage of the calculus of variations is that it allows us to use tools from the optimization community to solve optimal control problems. However, the solution obtained by the calculus of variations is only locally optimal, unless the optimal control problem is convex[2] and the duality gap is zero.

We now discuss another approach to solving optimal control problems, dynamic programming, which overcomes this limitation of calculus of variations at the expense of a higher computational complexity. In dynamic programming, the optimal control trajectory (and the corresponding optimal cost function) is obtained recursively starting from the terminal time T to the initial time t. The dynamic programming approach is based on the following *principle of optimality* which was first observed by Richard Bellman in 1952 in his seminal paper *On the Theory of Dynamic Programming* [52]:

[2]Note that since the dynamics appear as equality constraints in the optimal control problem, the problem can be convex only if the dynamics are linear.

In an optimal sequence of decisions or choices, each subsequence must also be optimal. Thus, if we take any state along the optimal state trajectory, then the remaining subtrajectory is also optimal.

In other words, the principle of optimality allows us to find the optimal solution to a problem using a combination of optimal solutions to some of its subproblems. To understand this mathematically, we define the *value function*, which is the optimal cost-to-go starting at state x at time t under dynamics and control constraints:

$$V(t,x) = \inf_{u(\cdot)} \int_t^T L(x(s),u(s))ds + M(x(T)), \tag{2.9}$$

Note that solving the optimal control problem in (2.6) is thus equivalent to finding the value function and the corresponding optimal control trajectory. Using the principle of optimality, we can divide the value function at time t into two subcomponents: choosing optimal control over the time interval $[t, t+\delta)$ and the optimal solution over the time interval $[t+\delta, T]$:

$$V(t,x) = \inf_{u(\cdot)} \left[\int_t^{t+\delta} L(x(s),u(s))ds + V(t+\delta, x(t+\delta)) \right], \tag{2.10}$$

where $x(t+\delta)$ is the system state at time $t+\delta$ when the control input $u(\cdot)$ is applied to the system. Even though this integral form doesn't seem particularly useful at first glance, using this form it can be shown the $V(t,x)$ in (2.10) is the *viscosity solution* to the Hamilton-Jacobi-Bellman (HJB) partial differential equation (PDE):

$$\frac{\partial V}{\partial t}(t,x) + H(t,x,V(t,x)) = 0 \qquad V(T,x) = M(x), \tag{2.11}$$

where $H(t,x,V(t,x))$ is called *Hamiltonian*. The Hamiltonian encodes the role of dynamics in the value function and is given as

$$H(t,x,V(t,x)) = \inf_{u \in \mathcal{U}} \left(L(x,u) + \langle \frac{\partial V}{\partial x}(t,x), f(x,u) \rangle \right), \tag{2.12}$$

where $\langle \cdot \rangle$ represent the inner product operation (dot product in $\mathbb{R}^n$). Using D_t and D_x to represent the partial derivatives of $V(t,x)$ with respect to time and state respectively, we can write the HJB PDE compactly as:

$$D_t V(t,x) + H(t,x,V(t,x)) = 0 \qquad V(T,x) = M(x), \tag{2.13}$$

The HJB PDE is a final-value PDE which can be solved backwards starting from the terminal time to obtain the value function at any time and state.[3] Once the value function is obtained, the optimal control at any state and time is given as:

$$u^*(t,x) = \arg\inf_{u \in \mathcal{U}} \left(L(x,u) + \langle \frac{\partial V}{\partial x}(t,x), f(x,u) \rangle \right), \tag{2.14}$$

[3]We will go further in the computational aspects of the HJB PDE in Section 2.3.3.

which is a state feedback policy that minimizes the cost function.

Intuitively, the HJB PDE can be thought of as the continuous-time version of the Bellman equation in discrete-time. To gain intuition, here we present an informal derivation of the HJB PDE based on finite element analysis of Equation (2.10)[4]. For very small $\delta > 0$, Equation (2.10) can be written as

$$V(t, x(t)) \approx \inf_{u(t)\in\mathcal{U}} \left[L(x(t), u(t))\delta + V(t+\delta, x(t+\delta))\right]. \tag{2.15}$$

On the other hand, the Taylor expansion of $V(t+\delta, x(t+\delta))$ can be written as:

$$V(t+\delta, x(t+\delta)) = V(t, x(t)) + D_t V(t, x(t))\delta + D_x V(t, x(t))\delta x + \text{h.o.t}, \tag{2.16}$$

where δx is change in the state and can be approximated as $f(x,u)\delta$. Ignoring the higher order terms in (2.16), we have

$$V(t+\delta, x(t+\delta)) \approx V(t, x(t)) + D_t V(t, x(t))\delta + D_x V(t, x(t)) \cdot f(x,u)\delta. \tag{2.17}$$

Finally, plugging in this Taylor approximation in Equation (2.15), we have

$$V(t, x(t)) \approx V(t, x(t)) + D_t V(t, x(t))\delta + \inf_{u(t)\in\mathcal{U}} \left[L(x(t), u(t))\delta + D_x V(t, x(t)) \cdot f(x,u)\delta\right], \tag{2.18}$$

where we have separated terms that do not depend on u. Cancelling out the redundant terms and noting that $\delta > 0$, we achieve the HJB PDE:

$$D_t V(t, x(t)) + \inf_{u(t)\in\mathcal{U}} \left[L(x(t), u(t)) + D_x V(t, x(t)) \cdot f(x,u)\right] \approx 0, \tag{2.19}$$

which allows propagating the value function backwards in time from the terminal condition $V(T, x) = M(x)$.

Existence of The Value Function - A Note on Viscosity Solution*. Equipped with the HJB PDE, we can compute the value function and hence solve the optimal control problem. But an important question to ask at this point is that does there always exist a solution to the PDE in (2.11). If so, what can we say about the properties of the resulting value function? For example, is value function always differentiable? If not, is it at least continuous in x and t? These questions are not only interesting from a theoretical viewpoint, but the answers to these questions immediately influence the plausible numerical methods to solve the HJB PDE. For example, if we cannot expect the value function to be differentiable, we should not expect to be able to compute the value function using a numerical method that relies on gradients and smoothness. In fact, it can be shown that the value function in general may not be differentiable everywhere in the state space. Thus, there may not exist a classical solution to the HJB PDE. However, the good news is that it can be shown that

[4] I thank Jaime F. Fisac for sharing this intuitive proof of the HJB PDE from the integral form.

there exists a *unique* viscosity solution to the HJB PDE [88] which is uniformly continuous in x and t, as long as $L(\cdot)$, $M(\cdot)$, and f are all Lipschitz continuous in x, u, and t (see Chapter 10 in [106]. In fact, the non-existence of a classical solution to the HJB PDE was one of the motivations to develop a *level set approach* for computing approximations to the viscosity solution of the HJB PDE. In the level set approach, the value function is represented as an implicit surface function. Sophisticated methods, that do not require differentiability of the underlying surface function, have been developed to propagate this entire surface function in time. We will further discuss the advantages and mathematical formulation of level set approach in Section 2.3.1.

2.2.3 Discrete Time Optimal Control Problem

So far we have discussed the optimal control problem in continuous time; however, as we discussed earlier, the discrete time formulations are more popular in RL and robotic communities. In discrete time optimal control problem, we are interested in solving the following optimization problem:

$$\inf_{\mathbf{u}} J_t(x_0, \mathbf{u}) \equiv \sum_{k=t}^{T} L(x_k, u_k) + M(x_T), \tag{2.20}$$

$$\text{subject to} \quad x_{k+1} = f(x_k, u_k), \quad \forall k \in \{t, t+1, \dots, T\} \tag{2.21}$$

$$u_k \in \mathcal{U} \tag{2.22}$$

Similar to continuous time optimal control problems, we can compute the optimal value function in discrete time using the principle of dynamic programming

$$V_k(x) = \inf_{u} \left[L(x, u) + V_{k+1}(x_{k+1}) \right], \quad V_T(x) = M(x), \tag{2.23}$$

where x_{k+1} is the state of the system when the control input u is applied to the system at state x, i.e., $x_{k+1} = f(x, u)$. $V_T(x)$ is the terminal value function. Unlike continuous time, the dynamic programming for discrete time does not result into a PDE, but rather a value function "backup" or iteration. Consequently, the equation (2.23) is referred as *Bellman equation*, *Bellman backup*, or *value iteration* equation. One particularly popular method to solve optimal control problem in discrete time is receding horizon Model Predictive Control (MPC), which is what we discuss next.

2.2.3.1 Model Predictive Control (MPC) Approach to Optimal Control

Model Predictive Control (MPC) is an optimization-based approach to design a state feedback controller for a dynamical system so as to minimize a cost function of the form (2.20). MPC is a rather broad and an active area of research and I won't be able to do a justice to this approach in the few pages that I have; however, I will attempt to discuss a specific MPC approach called *receding horizon* control that is widely used in the robotics community to solve optimal control problems. Consider the optimal control problem in (2.20) over the time horizon $\{0, 1, \ldots, T\}$. In receding horizon MPC, at time step t, the following optimization problem is solved starting from the system's current state x_t:

$$\begin{aligned}
&\inf_{\mathbf{u}} J_t(x_t, \mathbf{u}) \equiv \sum_{k=t}^{t+H_p} L(x_k, u_k), && (2.24)\\
&\text{subject to} \quad x_{k+1} = f(x_k, u_k), \quad \forall k \in \{t, t+1, \ldots, t+H_p-1\} && (2.25)\\
&\qquad\qquad\quad\; u_k \in \mathcal{U} && (2.26)
\end{aligned}$$

The optimal control problem in (2.24) is for a smaller horizon of H_p (also referred to as *planning horizon*). The intuition here is that the full optimal control problem might be very challenging to solve; on the other hand, the problem in (2.24) can be a much smaller optimization problem depending on the ratio $\frac{H_p}{T}$, and can be solved with significantly fewer computational resources. In fact, there are a variety of computational tools available for solving receding horizon MPC problems, such as YALMIP [201], that makes it a popular approach to solve optimal control problem.

Once the optimal control sequence $\mathbf{u}_t^{t+H_p}$ is obtained, it is applied on the system for a horizon of $H < H_p$, and the new system state is obtained. The rest of the control sequence is discarded and the MPC problem is solved again (called *replanning*) starting from the new state. Typically, $H = 1$, i.e., only the first control is applied from the sequence $\mathbf{u}_t^{t+H_p}$, and then the control sequence is replanned.

More general formulations of receding horizon MPC also include a special cost function, as well as a terminal state constraint set at the timestep $(t + H_p)$. We omitted those details here for simplicity and refer the interested readers to [202] for further discussion on the receding horizon MPC.

2.2.4 From Optimal Control to Robust Optimal Control: Zero-Sum Dynamic Games

So far we have considered optimal decision-making in the absence of disturbance. Now we will add the disturbance back in to discuss what is called a *robust optimal control* problem. Recall that the disturbance can represent actual environmental effects or model uncertainty. Robust optimal control aims to find a control strategy that leads to a desirable behavior (offer measured in terms of a cost function) despite the worst-case disturbance or model uncertainty. Mathematically, we are interested in solving the following optimization problem:

$$\begin{aligned}
&\inf_{u(\cdot)} \sup_{d(\cdot)} J_t(x_0, u(\cdot), d(\cdot)) \equiv \int_t^T L(x(s), u(s), d(s))ds + M(x(T)), && (2.27)\\
&\text{subject to} \quad \dot{x}(s) = f(x(s), u(s), d(s)), \quad \forall s \in [t, T] && (2.28)\\
&\qquad\qquad\quad\; u(s) \in \mathcal{U}, \qquad d(s) \in \mathcal{D}, && (2.29)
\end{aligned}$$

Thus, we want to find the control strategy that minimizes the worst-case cost function, when the disturbance tries its best to maximize the cost function.

The notion of a game. Note that the decision making process in (2.27) is a *game* between control (Player 1) and disturbance (Player 2) – Player 1 (or control) tries its best to minimize the cost function whereas Player 2 (or disturbance) tries it best to maximize the cost function. This game can be alternatively thought as of two different optimization problems, one for each player. Player 1 tries to minimize the cost function $J_t(x_0, u(\cdot), d(\cdot))$ subject to dynamics and control constraints. On the other hand, Player 2 tries to minimize the negative of the cost function subject to dynamics and disturbance bounds. However, unlike a traditional optimal control problem, these two optimization problems are coupled through dynamics, thus coupling their decision making process. Such games are also referred to as *minimax* or *zero-sum* games[5]. Moreover, here the decision making process evolves over time, hence it is a *dynamic* zero-sum game. Finally, this game evolves over time in the context of a dynamical system, where the state variables evolve over time according to a differential equation; such dynamic games are referred to as *differential* games.

Information pattern and non-anticipative strategies. In a differential game setting, it is important to address what information the players know about each other's decisions which directly affects their strategies, and consequently, the outcome of the game. As written in Equation (2.27), the control needs to declare its entire trajectory first and the disturbance can declare its own trajectory taking into account the control trajectory. This open-loop information structure is particularly conservative for control since control cannot adapt its decisions at all while the system is evolving. On the other hand, the disturbance has all the information it needs to make an optimal decision beforehand. Such strategies are overly conservative for dynamical systems, except when state feedback is not possible during the system evolution.

To allow adaptability in decision making during the system evolution, we assume that Player 2 (or disturbance) uses only non-anticipative strategies $\Gamma(\cdot)$ [298], defined as follows:

$$\begin{aligned}\gamma \in \Gamma_t^s :=& \{\mathcal{N} : \mathbb{U}_t^s \to \mathbb{D}_t^s : u(r) = \hat{u}(r) \text{ a. e. } r \in [t, s] \\ & \Rightarrow \mathcal{N}[u](r) = \mathcal{N}[\hat{u}](r) \text{ a. e. } r \in [t, s] \forall u(\cdot), \hat{u}(\cdot)\},\end{aligned} \tag{2.30}$$

where $\mathbb{U}_t^s$ is the space of all control trajectories over the time interval $[t, s]$. $\mathbb{D}_t^s$ is similarly defined. Intuitively, Equation (2.30) states that Player 2 cannot respond differently to two Player 1 controls until they become different. Thus, Player 2 cannot change its strategy in the anticipation of a change in $u(\cdot)$ until that change actually begins. Yet, in this setting, Player

[5]When the cost of one player is not the same as the negative of the cost of the other player, we refer to the game between the players as a general sum game. In this work, we are primarily interested in zero-sum games; however, general sum games are also a popular choice for modeling several multi-agent robotic systems, particularly in human-centric environments. We refer the interested readers to [42] to learn more about general sum games.

2 has the advantage of factoring in Player 1's choice of input at every instant t and adapting its own accordingly. Thus, Player 2 has an *instantaneous informational advantage*, which allows us to obtain the robust control (Player 1) with respect to the worst-case disturbance (Player 2) (not because this disturbance is in fact reacting to the controller's input, but rather, because out of all possible disturbances there will be one that will happen to be the worst possible given the chosen control).

With non-anticipative information pattern, the zero-sum differential game can be more formally written as:

$$\sup_{d(\cdot)\in\Gamma_t^T} \inf_{u(\cdot)} J_t(x_0, u(\cdot), d(\cdot)) \equiv \int_t^T L(x(s), u(s), d(s))ds + M(x(T)), \tag{2.31}$$

$$\text{subject to} \quad \dot{x}(s) = f(x(s), u(s), d(s)), \quad \forall s \in [t, T] \tag{2.32}$$

$$u(s) \in \mathcal{U}, \qquad d(s) \in \mathcal{D}, \tag{2.33}$$

that is, disturbance first chooses a strategy from the set of non-anticipative strategies[6] and then the control optimizes for its strategy.

Value of the game. We can now solve the robust optimal control problem using the dynamic programming principle, much like optimal control problem. In particular, the value function is given by

$$V(t,x) = \sup_{d(\cdot)\in\Gamma_t^T} \inf_{u(\cdot)} J_t(x_0, u(\cdot), d(\cdot)), \tag{2.34}$$

which can be shown to be the unique viscosity solution to the Hamilton-Jacobi-Isaacs (HJI) partial differential equation (PDE)[7]:

$$D_t V(t,x) + H(t,x,V(t,x)) = 0 \qquad V(T,x) = M(x). \tag{2.35}$$

The Hamiltonian is given by

$$H(t,x,V(t,x)) = \inf_{u\in\mathcal{U}} \sup_{d\in\mathcal{D}} \left(L(x,u,d) + \langle D_x V(t,x), f(x,u,d)\rangle\right) \tag{2.36}$$

Given the value function, optimal control and disturbance can be obtained as:

$$u^*(t,x) = \arg\inf_{u\in\mathcal{U}} \sup_{d\in\mathcal{D}} \left(L(x,u,d) + \langle D_x V(t,x), f(x,u,d)\rangle\right), \tag{2.37}$$

$$d^*(t,x) = \inf_{u\in\mathcal{U}} \arg\sup_{d\in\mathcal{D}} \left(L(x,u,d) + \langle D_x V(t,x), f(x,u,d)\rangle\right), \tag{2.38}$$

[6]Here, the disturbance is only committing to a non-anticipative strategy; however, it can still adapt its action based on the applied control.

[7]Note that the PDE in (2.35) is called Hamilton-Jacobi-Isaacs and not Hamilton-Jacobi-Bellman to honor Rufus Isaacs who studied pursuit-evasion games and proposed a principle of optimality for dynamic games around the same time Richard Bellman proposed it for optimal control problems. In fact, they both were working in RAND corporation at that time.

2.3 Hamilton-Jacobi Reachability

Hamilton-Jacobi (HJ) reachability analysis is a verification method for guaranteeing performance and safety properties of systems. In reachability theory, we are often interested in computing the *backward reachable set (BRS)* of a dynamical system. This is the set of states such that the trajectories that start from this set can reach some given target set. If the target set consists of those states that are known to be unsafe, then the BRS contains states which are potentially unsafe and should therefore be avoided. As an example, consider collision avoidance protocols for two aircraft in En-Route airspace. The target set would contain those states that are already "in loss of separation," such as those states in which the aircraft are within the five mile horizontal separation distance mandated by the Federal Aviation Administration. The backward reachable set contains those states which could lead to a collision, despite the best possible control actions. We typically formulate such safety-critical scenarios in terms of a two-player game, with Player 1 and Player 2 being control inputs. For example, Player 1 could represent one aircraft, Player 2 another, with Player 1's control input being treated as the control input of the joint system, and with Player 2's control input being treated as the disturbance.

Consider the dynamics in Equation 2.1. Mathematically, a BRS represents the set of states $x \in \mathbb{R}^{n_x}$ from which the system can be driven into some set $\mathcal{L} \subseteq \mathbb{R}^{n_x}$ at the *end* of a time horizon, despite the best control efforts. We call $\mathcal{L}$ the "target set". We assume that Player 1 (or control) will try to steer the system away from the target with her input, and Player 2 (or disturbance) will try to steer the system toward the target with her input. Consequently, we want to compute the following set:

$$\mathcal{V}(t) = \{x : \forall u(\cdot) \in \mathbb{U}_t^T, \exists d(\cdot) \in \Gamma_t^T, \xi(T; x, t, u(\cdot), d(\cdot)) \in \mathcal{L}\}, \tag{2.39}$$

where $\Gamma(\cdot)$ denotes the set of non-anticipative strategies as defined in (2.30), and $\mathbb{U}_t^T$ is the space of all control trajectories over the time interval $[t, T]$. Intuitively, Equation (2.39) computes the set of all states starting from which no matter what control does, there exists a non-anticipative disturbance strategy that will drive the system to the target set at the end of the time horizon.

2.3.1 The Level Set Approach: From Games of Kind to Games of Degree

The computation of the BRS in (2.39) requires solving a differential game between Player 1 and Player 2. However, this differential game is a "game of kind" rather than a "game of degree", i.e., games in which the outcome is Boolean: the system either reaches the target set or not under specified constraints at any time within the duration of the game. On the other hand, our discussion on differential games in Section 2.2.4 was limited to the game of degree, where the outcome of the game is continuous defined by the value function. The good news is that an approach known as the *level set method* can transform these games of kind into games of degree in an analytically sound and computationally tractable way. For example, if we consider $J_t(\cdot)$ as the distance between the system state and the target region at the terminal state of the system, it is easy to determine whether the system reached the target by comparing this distance to some threshold value (simply 0 in this case). This allows us to find the solution to a game of kind by posing an auxiliary game of degree whose solution encodes that of the original problem: this is, in essence, the level set approach.

In particular, one can always find a Lipschitz function $l(x)$ such that $\mathcal{L}$ (the target set) is equal to the subzero level set of l, that is, $x \in \mathcal{L} \Leftrightarrow l(x) \leq 0$. The Lipschitz function l can always be found, since one can always choose the signed distance to the respective sets. If we define the cost function to be

$$J_t(x, u(\cdot), d(\cdot)) = l(x(T)), \tag{2.40}$$

then the system reaches the target set under controls u and d if and only if $J_t(x, u(\cdot), d(\cdot)) \leq 0$. Since Player 2 (or disturbance) wants to drive the system to the target, it wants to minimize the cost in (2.40), and Player 1 (or control) wants to maximize this cost. Comparing, (2.40) with (2.27), the reachability problem can be formulated as a robust control problem by setting the terminal cost $M(x) \equiv l(x)$ and the running cost $L \equiv 0$.

We can now compute the value function $V(t, x)$ for this differential game in a similar fashion to Section 2.2.4. Consequently, the BRS can be obtained as

$$\mathcal{V}(t) = \{x : V(t, x) \leq 0\}, \tag{2.41}$$

where $V(t, x)$ satisfies the following HJI PDE:

$$D_t V(t, x) + H(t, x, V(t, x)) = 0 \qquad V(T, x) = l(x), \tag{2.42}$$

and the Hamiltonian is given by

$$H(t, x, V(t, x)) = \sup_{u \in \mathcal{U}} \inf_{d \in \mathcal{D}} \langle D_x V(t, x), f(x, u, d) \rangle. \tag{2.43}$$

Note the change of the role of supremum and infimum in the above equation. Since control is now trying to keep the system away from the target set, it is maximizing the cost function and the disturbance is minimizing the cost function.

The interpretation of $\mathcal{V}(t)$ is that if $x(t) \in \mathcal{V}(t)$, then Player 2 has a control sequence that will drive the system to the target at time T starting from the state $x(t)$ at time t, irrespective of the control of Player 1. If $x(t) \in \partial\mathcal{V}(t)$, where $\partial\mathcal{V}(t)$ denotes the boundary of $\mathcal{V}(t)$, then Player 1 will *barely* miss the target at time T if it applies the optimal control

$$u^*(t, x) = \arg \sup_{u \in \mathcal{U}} \inf_{d \in \mathcal{D}} \langle D_x V(t, x), f(x, u, d) \rangle. \tag{2.44}$$

Finally, if $x(t) \in \mathcal{V}(t)^C$[8], then Player 1 has a control sequence (given by (2.44)) that will keep the system out of the target set, irrespective of the control applied by Player 2. In particular, when the target set $\mathcal{L}$ represents unsafe/undesired states of the system and Player 2

[8] $\mathcal{V}(t)^C$ represents the complement of the set $\mathcal{V}(t)$.

represents the disturbances in the system, then $\mathcal{V}(t)$ represents the *effective* unsafe set, i.e., the set of states from which the disturbance can drive the system to the *actual* unsafe set despite the best control efforts. Thus, reachability analysis gives us the safe set (in this case $\mathcal{V}(t)^C$) as well as a controller (in this case $u^*(t, x)$) that will keep the system in the safe set, given that the system starts in the safe set.

Advantages of Level Set Approach. We conclude this section by discussing some of the advantages of using level set approach to compute the BRS, i.e., recovering the BRS as a subzero level of a value function as in (2.41). First, the level set approach allows us to cast the reachability problem as a differential game which can then be solved using the tools developed for such games in the robust control community. A second advantage of the level set method is that there are well developed numerical methods that can propagate the level functions on a fixed Cartesian grid without having to parameterize these objects (this is called the Eulerian approach). Finally, the level-set method makes it very easy to follow shapes that change topology, for example, when a reachable set splits in two, develops holes, or the reverse of these operations [240]. All these make the level-set method a great tool for modeling time-varying value functions. Nevertheless, several other approach have been proposed in literature to compute the BRS. We discuss the relative advantages and limitations of these different approaches in Chapter 3.

2.3.2 Variations of Reachability

So far, we have presented the computation of BRSs, but reachability analysis is not limited to BRSs. One can compute various other kinds of sets that may be more useful, depending on the verification problem at hand. In this section, we provide a brief overview of some of these sets.

2.3.2.1 Roles of the Control and Disturbance.

Depending on the role of Player 1 and Player 2, we may need to use different inf-sup combinations in Equation (2.43). For example, suppose $\mathcal{L}$ represents the set of destination states for an aircraft that it wants to reach despite the disturbances, such as wind. Consequently, we want to compute the following BRS:

$$\mathcal{V}(t) = \{x : \forall d(\cdot) \in \Gamma_t^T, \exists u(\cdot) \in \mathbb{U}_t^T, \xi(T; x, t, u(\cdot), d(\cdot)) \in \mathcal{L}\}, \tag{2.45}$$

In this case, the control is trying to steer the system towards the target, and the disturbance can be modeled as steering the system away from the target to obtain a robust controller for the system. Thus, the control will minimize the Hamiltonian and the disturbance will maximize it.

As a rule of thumb, whenever the existence of a control ("$\exists u$") is sought, the optimization is a minimum over the set of controls in the corresponding Hamiltonian. Whenever a set/tube characterizes the behavior of the system for all controls ("$\forall u$"), the optimization is a maximum. For example, for the BRS in (2.39), we seek the *existence* of a Player 2 controller *for all* Player 1 controls, so we use minimum for Player 2 and maximum for Player 1 in the Hamiltonian (see (2.43)). When the target set represents the set of the desired states that we want the system to reach and Player 2's control represents the disturbance, then we are interested in verifying if there exists a control of Player 1 such that the system reaches its target despite the worst-case disturbance. In this case, we should use maximum for Player 2's control and minimum for Player 1's control in the corresponding Hamiltonian.

2.3.2.2 Reachable Sets vs. Tubes.

Another important aspect in reachability is that of reachable tubes. The reachable set is the set of states from which the system can reach a target at *exactly* time T. Perhaps a more useful notion is to compute the set of states from which the system can reach a target *within* the time interval $[t, T]$. For example, for safety analysis, we are interested in verifying if a disturbance can drive the system to the unsafe states *ever* within a horizon, and not just at the end of the horizon. This notion is captured by reachable tubes. Formally, the backward reachable tube (BRT) can be defined as:

$$\mathcal{V}(t) = \{x : \forall u(\cdot) \in \mathbb{U}_t^T, \exists d(\cdot) \in \Gamma_t^T, \exists s \in [t, T], \xi(s; x, t, u(\cdot), d(\cdot)) \in \mathcal{L}\}. \tag{2.46}$$

Intuitively, to compute the BRT, we need to compute the distance between the system state and the target set not only at the end of the time horizon, rather we need to keep track of the minimum distance between them over the *entire* time horizon. In particular, if we define the cost function to be

$$J_t(x, u(\cdot), d(\cdot)) = \inf_{s \in [t,T]} l(x(s)), \tag{2.47}$$

then a negative cost implies that the system state must have entered the target set at some point during the time horizon.

Note that the cost function in (2.47) does not satisfy the standard form in (2.31) as the cost seeks the minimum of a function over the time horizon, rather than its integral. Nevertheless, we can still use the principle of dynamic programming to compute the optimal value function, which is given as the viscosity solution of the following final-value HJI variational inequality (VI):

$$\min\{D_t V(t, x) + H(t, x, V(t, x)),\ l(x) - V(t, x)\} = 0\ \forall x, t \qquad V(T, x) = l(x), \tag{2.48}$$

where the Hamiltonian is given by

$$H(t, x, V(t, x)) = \sup_{u \in \mathcal{U}} \inf_{d \in \mathcal{D}} \langle D_x V(t, x), f(x, u, d) \rangle. \tag{2.49}$$

Finally, the BRT is given by:

$$\mathcal{V}(t) = \{x : V(t, x) \leq 0\}, \tag{2.50}$$

To gain intuition, let's again go through an informal derivation of the HJI VI based on finite element analysis. For very small $\delta > 0$, the dynamic programming principle for the cost function in (2.47) implies that

$$V(t, x(t)) \approx \sup_{u \in \mathcal{U}} \inf_{d \in \mathcal{D}} \min\{l(x(t)), V(t+\delta, x(t+\delta))\}, \tag{2.51}$$

$$= \min\{l(x(t)), \sup_{u \in \mathcal{U}} \inf_{d \in \mathcal{D}} V(t+\delta, x(t+\delta))\}, \tag{2.52}$$

On the other hand, the Taylor expansion of $V(t+\delta, x(t+\delta))$ can be written as:

$$V(t+\delta, x(t+\delta)) = V(t, x(t)) + D_t V(t, x(t))\delta + D_x V(t, x(t))\delta x + \text{h.o.t}, \tag{2.53}$$

where δx is change in the state and can be approximated as $f(x, u, d)\delta$. Ignoring the higher order terms in (2.53), we have

$$V(t+\delta, x(t+\delta)) \approx V(t, x(t)) + D_t V(t, x(t))\delta + D_x V(t, x(t)) \cdot f(x, u, d)\delta. \tag{2.54}$$

Finally, plugging in this Taylor approximation in Equation (2.51), we have

$$V(t, x(t)) \approx \min\{l(x(t)), V(t, x(t)) + D_t V(t, x(t))\delta + \sup_{u \in \mathcal{U}} \inf_{d \in \mathcal{D}} D_x V(t, x(t)) \cdot f(x, u, d)\delta\}, \tag{2.55}$$

where we have separated terms that do not depend on u and d. Subtracting $V(t, x(t))$ on both sides, we get

$$\min\{l(x(t)) - V(t, x(t)), \delta\left[D_t V(t, x(t)) + \sup_{u \in \mathcal{U}} \inf_{d \in \mathcal{D}} D_x V(t, x(t)) \cdot f(x, u, d)\right]\} \approx 0, \tag{2.56}$$

Since (2.56) holds for all $\delta > 0$, we must have that

$$\min\{l(x(t)) - V(t, x(t)), D_t V(t, x(t)) + \sup_{u \in \mathcal{U}} \inf_{d \in \mathcal{D}} D_x V(t, x(t)) \cdot f(x, u, d)\} \approx 0, \tag{2.57}$$

which results into the desired HJI VI.

2.3.2.3 Reachable Tubes vs. Viability Kernel.

Reachability theory is also strongly connected with viability theory. The basic problem of viability theory is to find the "viability kernel" of a system: the subset of initial states of the system such that there exists at least one "viable" evolution of the system, in the sense that at each time, the state of the evolution remains confined to a given set $\mathcal{L}$. This is important when for example there are obstacles in the environment and we want the system to remain clear of all the obstacles at each time step. Formally, the viability kernel can be defined as:

$$\mathcal{K}(t) = \{x : \forall d(\cdot) \in \Gamma_t^T, \exists u(\cdot) \in \mathbb{U}_t^T, \forall s \in [t, T], \xi(s; x, t, u(\cdot), d(\cdot)) \in \mathcal{L}\}, \tag{2.58}$$

Note that unlike a typical BRS or BRT computation, here we are interested in keeping the system state within $\mathcal{L}$ at *all times* in the horizon $[t, T]$. Thus, the viability kernel can at most be as big as $\mathcal{L}$ (otherwise the viability kernel requirement will be violated at the initial time).

Interestingly, the viability kernel computation can be formulated as a BRT computation. In particular, consider the complement of the set in Equation (2.58):

$$\mathcal{K}(t)^C = \{x : \forall u(\cdot) \in \mathbb{U}_t^T, \exists d(\cdot) \in \Gamma_t^T, \exists s \in [t,T], \xi(s; x, t, u(\cdot), d(\cdot)) \in \mathcal{L}^C\}, \tag{2.59}$$

Comparing equations (2.59) and (2.46), it is easy to note that $\mathcal{K}(t)^C$ is same as the BRT $\mathcal{V}(t)$ of $\mathcal{L}^C$. This BRT can be computed by solving the following HJI VI:

$$\min\{D_t V(t,x) + H(t,x,V(t,x)), \ -l(x) - V(t,x)\} = 0 \ \forall x, t \qquad V(T,x) = -l(x), \tag{2.60}$$

where the Hamiltonian is given by Equation (2.49). As before, $\mathcal{L}$ is given as the subzero level of $l(x)$. Note that instead of $l(x)$ we are using $-l(x)$ in (2.60) since we are computing the BRT of $\mathcal{L}^C$. Consequently, the BRT is given by:

$$\mathcal{K}(t)^C \equiv \mathcal{V}(t) = \{x : V(t,x) \le 0\} \tag{2.61}$$

Thus, the viability kernel is given by:

$$\mathcal{K}(t) \equiv \mathcal{V}(t)^C = \{x : V(t,x) > 0\}, \tag{2.62}$$

that is, the superzero level of the value function.

2.3.2.4 Forward vs. Backward Reachable Set.

In some cases, we might be interested in computing a forward reachable set (FRS): the set of all states that a system can reach from a given initial set of states after a given time duration. Formally, we want to compute the following set:

$$\mathcal{W}(t) = \{y : \forall u(\cdot) \in \mathbb{U}_t^T, \exists d(\cdot) \in \Gamma_t^T, \xi(T; x_0, t, u(\cdot), d(\cdot)) = y, x_0 \in \mathcal{L}\}, \tag{2.63}$$

Here, $\mathcal{L}$ represents the set of initial states of the system. $\mathcal{W}(t)$ is the set of all states that the system can reach at time T when it starts at state $x \in \mathcal{L}$ at time t while Player 1 applies the control to keep the system in $\mathcal{L}$ and Player 2 applies the control to drive the system out of $\mathcal{L}$. To avoid confusion between a FRS and a BRS, we will denote the FRS by $\mathcal{W}(t)$ and the BRS by $\mathcal{V}(t)$. The corresponding value functions are denoted by $W(t,x)$ and $V(t,x)$ respectively.

The FRS can be computed in a similar fashion as the BRS. The only difference is that an initial value HJI PDE needs to be solved instead of a final value PDE:

$$D_t W(t,x) + H(t,x,W(t,x)) = 0 \qquad W(0,x) = l(x), \tag{2.64}$$

and the Hamiltonian is given by

$$H(t, x, W(t, x)) = \sup_{u \in \mathcal{U}} \inf_{d \in \mathcal{D}} \langle D_x W(t, x), f(x, u, d) \rangle. \tag{2.65}$$

Here, 0 represents the initial time. Once the value function is computed, the FRS can be obtained as the subzero level set of the value function:

$$\mathcal{W}(t) = \{x : W(t, x) \leq 0\}, \tag{2.66}$$

One can similarly define forward reachable tubes (FRT), which can be computed by solving the following initial value HJI VI:

$$\max\{D_t W(t, x) + H(t, x, W(t, x)),\ W(t, x) - l(x)\} = 0\ \forall x, t \qquad W(0, x) = l(x), \tag{2.67}$$

Note that the initial value PDE in (2.64) can be converted into an equivalent final value PDE by change of time variables [106, 221]. This conversion can also be seen by drawing a parallel between a FRS and a BRS – the FRS $W(T, x)$ starting at the initial time t is same as the BRS $V(t, x)$ obtained using the dynamics $-f$ instead of f. Intuitively, $-f$ just reverses the flow field of f so the system is "evolving" backwards in time instead of forward in time. The corresponding final value HJI PDE for the BRS is given by:

$$D_t V(t, x) + \inf_{u \in \mathcal{U}} \sup_{d \in \mathcal{D}} \langle D_x V(t, x), -f(x, u, d) \rangle = 0 \qquad V(T, x) = l(x), \tag{2.68}$$

where an infimum is taken over control in Hamiltonian since it is trying to keep the system inside $\mathcal{L}$. In other words, it is trying to minimize the growth of the BRS. The above equation can be rewritten as:

$$D_t V(t, x) - \sup_{u \in \mathcal{U}} \inf_{d \in \mathcal{D}} \langle D_x V(t, x), f(x, u, d) \rangle = 0 \qquad V(T, x) = l(x), \tag{2.69}$$

By substituting $s = T - t$ in the above PDE, we get:

$$-D_s V(T - s, x) - \sup_{u \in \mathcal{U}} \inf_{d \in \mathcal{D}} \langle D_x V(T - s, x), f(x, u, d) \rangle = 0 \qquad V(0, x) = l(x), \tag{2.70}$$

Denoting $W(s, x) \equiv V(T - s, x)$, we get the following PDE:

$$D_s W(s, x) + \sup_{u \in \mathcal{U}} \inf_{d \in \mathcal{D}} \langle D_x W(s, x), f(x, u, d) \rangle = 0 \qquad W(0, x) = l(x), \tag{2.71}$$

which is same as the initial value PDE for computing FRS in (2.64).

2.3.2.5 Presence of State Constraints and Time-Varying Target Sets.

Another interesting problem that arises in verification is the reachability to and from a target set subject to some state constraints. For example, an aircraft may want to reach a destination while avoiding any tall buildings on its way. Avoiding these obstacles (buildings in this case) can be written as state constraints, i.e., the system state should never be in any obstacle at any time. In such scenarios, we are interested in computing what is called a *backward reach-avoid set*. Intuitively, the BRS represents the set of states from which a vehicle can reach the target set $\mathcal{L}$ at the *end* of a time horizon while avoiding the obstacles $\mathcal{G}(t)$ at all times t. The reach-avoid set can be formally defined as:

$$\mathcal{V}(t) = \{x : \forall d(\cdot) \in \Gamma_t^T, \exists u(\cdot) \in \mathbb{U}_t^T, \forall s \in [t,T], \xi(s;x,t,u(\cdot),d(\cdot)) \notin \mathcal{G}(s), \tag{2.72}$$
$$\xi(T;x,t,u(\cdot),d(\cdot)) \in \mathcal{L}\}. \tag{2.73}$$

To compute this BRS, we define implicit surface functions for representing both the target set and the time-varying obstacles. As before, the function $l(x)$ is the implicit surface function representing the target set $\mathcal{L} = \{x : l(x) \leq 0\}$. Similarly, the function $g(t,x)$ is the implicit surface function representing the time-varying obstacles $\mathcal{G}(t) = \{x : g(t,x) \leq 0\}$. The corresponding value function can be obtained as a solution to the following final-value HJI VI:

$$\max\{D_t V(t,x) + H(t,x,V(t,x)), \ -g(t,x) - V(t,x)\} = 0 \ \forall x,t \tag{2.74}$$
$$V(T,x) = \max\{l(x), -g(T,x)\}, \tag{2.75}$$

where the Hamiltonian is same as in equation (2.43), and the backward reach-avoid set is given as the subzero level set of the value function. One can similarly compute reach-avoid tubes, which is the set of states from which a vehicle can reach the target set $\mathcal{L}$ *within* a time horizon while avoiding the obstacles $\mathcal{G}(t)$ at all times t. The corresponding value function can be obtained by solving the following final value HJI VI:

$$\max\{\min\{D_t V(t,x) + H(t,x,V(t,x)), \ l(x) - V(t,x)\}, \ -g(t,x) - V(t,x)\} = 0 \ \forall x,t \tag{2.76}$$
$$V(T,x) = \max\{l(x), -g(T,x)\}, \tag{2.77}$$

These reach-avoid sets can be computed efficiently for even time-dependent target sets [209, 115]. The HJI VI now has a time argument for $l(x)$. For example, the HJI VI in Equation (2.76) is now given by:

$$\max\{\min\{D_t V(t,x) + H(t,x,V(t,x)), \ l(t,x) - V(t,x)\}, \ -g(t,x) - V(t,x)\} = 0 \ \forall x,t \tag{2.78}$$
$$V(T,x) = \max\{l(T,x), -g(T,x)\}, \tag{2.79}$$

The HJI VI for a reach-avoid FRS and a reach-avoid BRS can be similarly obtained by adding a time argument in $l(x)$ and adding the outer min in the HJI VI in the presence of an obstacle.

Remark 2 *Viability kernel of a target set $\mathcal{L}$ can also be computed by treating $\mathcal{L}^C$ as an obstacle (i.e., the state constraint of keeping the system state within $\mathcal{L}$). We leave it to the reader to show that the HJI VI for the negative of the value function in* (2.60) *results in the HJI VI in* (2.74) *with $g(x) = -l(x)$. This is not surprising because the problem of viability kernel is exactly same as the state constraint of keeping the state within $\mathcal{L}$ at all times, or equivalently, never let the system state be in $\mathcal{L}^C$ at any time.*

Remark 3 *In general, any combination of the four variants discussed in this section can be solved using the HJ reachability formulation. Partially, it is this flexibility of the reachability framework that has facilitated its use in various safety-critical applications, some of which we will discuss more in this work.*

2.3.3 Level Set Toolbox

A major practical appeal of Hamilton-Jacobi reachability stems from the availability of modern numerical tools, which can compute various definitions of reachable sets [275, 239, 220, 219]. For example, these numerical tools have been successfully used to solve a variety of differential games, path planning problems, and optimal control problems. Concrete practical applications include aircraft auto-landing [43], automated aerial refueling [97], model predictive control (MPC) of quadrotors [59, 25], multiplayer reach-avoid games [158], large-scale multiple-vehicle path planning [73, 77], and real-time safe motion planning [151].

One such computational tool is the *level set toolbox (ToolboxLS)* which was developed by Professor Ian Mitchell [219] to solve partial differential equations using level set methods, and is the foundation of the HJ reachability code. The toolbox is implemented in MATLAB and is equipped to solve any final-value HJ PDE. Since different reachable set computations can be ultimately posed as solving a final-value HJ PDE (see Section 2.3.2), the level set toolbox is fully equipped to compute various types of reachable sets. Information on how to install and use ToolboxLS can be found on Prof. Ian Mitchell's website[9]. This toolbox can be further augmented by the Hamilton-Jacobi optimal control toolbox (or *helperOC*). A quick-start guide to using toolboxLS and helperOC is presented in [34] and is also available at helperOC website[10].

However, traditionally, reachable set computations using level set toolbox involve solving an HJ partial differential equation (PDE) on a grid representing a discretization of the state space, resulting in an *exponential* scaling of computational complexity with respect to system dimensionality; this is often referred to as the "curse of dimensionality." Thus, HJ reachability becomes computationally intractable as the state space dimension increases. In Part 1 of this dissertation, we will present various ways to overcoming curse of dimensionality using algorithmic and computational advances.

[9] ToolboxLS Website: https://www.cs.ubc.ca/ mitchell/ToolboxLS/
[10] helperOC Website: http://www.github.com/HJReachability/helperOC

2.4 Reinforcement Learning

At its core, RL and optimal control are very similar in that they both aim to solve the following optimization problem (also referred to as *optimal control problem* hereon):

$$\inf_{\mathbf{u}} J_t(x_0, \mathbf{u}) \equiv \sum_{k=t}^{T} L(x_k, u_k) + M(x_T), \tag{2.80}$$

$$\text{subject to} \quad x_{k+1} = f(x_k, u_k), \quad \forall k \in \{t, t+1, \ldots, T\} \tag{2.81}$$

$$u_k \in \mathcal{U} \tag{2.82}$$

However, there are three key differences between RL and optimal control: first, in RL one is interested in maximizing a reward function whereas in control one is interested in minimizing a cost function; second, in RL one is interested in solving the above optimization problem when the dynamics model f is not available; and third, typically a *discount factor* $\gamma < 1$ is involved in the cost function, i.e., the cost function is of the form

$$\inf_{\mathbf{u}} J_t(x_0, \mathbf{u}) \equiv \sum_{k=t}^{T} \gamma^t L(x_k, u_k) + \gamma^T M(x_T) \tag{2.83}$$

The first difference is not that significant because minimizing a cost function can always be thought of as maximizing the negative of the cost function. Without loss of generality, from here on we will assume that in RL we are minimizing a cost function. However, the other two differences between RL and optimal control have huge implications for the kind of approaches taken into the two communities (RL community and control community) to solve the optimal control problem, some of which we discuss here.

Since RL approaches do not require a dynamics model of the system, they can be applied in scenarios where constructing a model can be quite challenging. Imagine for example a controller for tying your shoe laces. It is non-trivial to even define a reasonable state space for this problem, let alone come up with a dynamics model. Since the dynamics model is unknown, the optimal control problem in RL is solved in a *data-driven fashion.* These RL approaches can be broadly classified as: model-based RL and model-free RL. In model-based RL, first a system dynamics model is learned using the data collected on the system. Typically, the model is optimized so as to maximize the log likelihood of the observed data. For forward dynamics model, the model is optimized to minimize the prediction error of the next state given the current state and input. The learned model is then used to design the optimal controller for the system using traditional optimal control tools, such as MPC. In contrast, model-free RL approaches do not explicitly maintain a system dynamics model and directly optimize the controller (typically parameterized) to improve the control performance. Intuitively, this optimization process starts with trial and error and over time the trials that lead to good performance (i.e., low cost or high reward) are *positively reinforced* to further improve the control performance over time, hence the name reinforcement learning. Of course, this is a very simplistic view at RL; over the past few years, very sophisticated algorithms have been developed both for model-based and model-free RL that can solve complex optimal control problems even when the system state is not directly observable, and only high-dimensional observations are available. I should mention that data-driven control is also studied in the control community; in fact, the entire field of adaptive control is dedicated to this endeavor. Nevertheless, there are some key differences between RL and adaptive control. We discuss the relative advantages and limitations of model-based and model-free approaches further, as well as their connection to adaptive control and system identification approaches in Chapters 5 and 6.

As we discussed earlier, the second key difference is the presence of a discount factor in RL. Even though it might seem like a insignificant difference at first, it is actually of vital importance. To understand this, recall that the dynamic programming principle for discrete-time optimal control problem leads to following value iteration equation:

$$V_k(x) = \inf_u \left[\gamma^k L(x, u) + V_{k+1}(x_{k+1})\right], \tag{2.84}$$

This value iteration equation is typically solved backwards in time starting from the terminal value function. The presence of a discount factor gives the value function a *contraction* property, wherein at any timestep, future costs matter exponentially less compared to the current cost. Thus, even when one starts with an inaccurate value function V_T at the terminal time, the value iteration equation above will lead to the correct value function after sufficient iterations. This is because the error in the terminal value function decays exponentially due to the presence of a discount factor. This is particularly important for model-free RL approaches that do not maintain a dynamics model of the system, and try to directly optimize the control performance through learning a value function either implicitly and explicitly. The above contraction property ensures that even when we start with a wrong estimate of the value function, which is inevitable when we do not have a model of the system, over time we will be able to recover the correct value function. This contraction property also means that the value function eventually converges and hence we can now define *time-independent* value functions. A corollary of this property is that we can learn value functions from data without necessarily any time synchronization (e.g., a method that simply minimize the Bellman error between a state pair), which in general can be very challenging.

As we discussed earlier, there are a lot of similarities between RL and optimal control; however, different terminologies are used to represent the same quantities in the two communities which sometimes make these similarities rather opaque. In the hope to bridge this gap, we conclude this section with Table 2.1, which provides a summary of some relevant concepts as denoted in different communities.

2.5 Neural Networks

As more and more data is being produced, and more and more computational power continues to become available, an important opportunity lies in harnessing data towards auton-

Table 2.1: A summary of some relevant concepts, as denoted in reinforcement learning and control communities.

Control	**Reinforcement Learning**
System	Environment
Controller	Agent
Dynamics	Transition
State (x)	State (s)
Control input (u)	Action (a)
Control law ($u(\cdot)$)	Policy ($\pi(\cdot)$)
Output (y)	Observation (O)
Cost (minimize J)	Reward (maximize R)

omy. In recent years, the fields of computer vision and speech processing have not only made significant leaps forward, but also rapidly increased their rate of progress, largely thanks to developments in deep learning [179, 192]. The success of deep learning is primarily attributed to the reemergence of Deep Neural Networks (DNN) as the key function approximator in machine learning. DNNs are known to be universal function approximators; their structure allows them to model highly nonlinear functions directly from the observed data. Here we discuss some of the DNN architectures that are widely popular in the machine learning community.

2.5.1 Feed-Forward Neural Networks

A feed-forward neural network model is a multi-layer NN, consisting of an input layer, multiple hidden layers, and an output layer. Given input $x \in \mathbb{R}^n$, the output of the NN model can be written as:

$$f(x;\theta) := \mathbf{\Phi}_N \circ \mathbf{\Phi}_{N-1} \circ \dots \mathbf{\Phi}_0(x), \tag{2.85}$$

where $\mathbf{\Phi}_0$ represents the output of the input layer, $\mathbf{\Phi}_N$ represent the output of the output layer, and $\mathbf{\Phi}_1, \mathbf{\Phi}_2, \dots, \mathbf{\Phi}_{N-1}$ represent the outputs of $N-1$ hidden layers of the NN. Each hidden layer i typically consists of M_i *hidden units* with weight matrix $W_i \in \mathbb{R}^{M_{i-1} \times M_i}$, a bias vector $b_i \in \mathbb{R}^{M_i}$, and a non-linearity (also called transfer or activation function) $\phi(\cdot)$ at each hidden unit. Thus, the output of the ith hidden layer, $\mathbf{\Phi}_i$, is given as:

$$\mathbf{\Phi}_i(z) := \phi(W_i^T z + b_i), \tag{2.86}$$

where $z \in \mathbb{R}^{M_{i-1}}$ is the input to the ith layer. The non-linearity ϕ in (2.86) is applied componentwise. A popular choice for non-linearity is to use the rectified-linear (ReLU) transfer function in the hidden layer. The corresponding NN is called *ReLU network*. The ReLU activation function is given by $\phi(z) = max(0, z)$, where maximum is taken componentwise.

It is also a common practice to not use any nonlinearity and weights at the input layer, and to not use any nonlinearity at the output layer, i.e.,

$$\mathbf{\Phi}_0(z) := z, \quad \mathbf{\Phi}_N(z) := W_N^T z + b_N \tag{2.87}$$

The weights and biases of all the layers together form the parameters of the neural network, i.e., $\theta = (W_0, W_1, \ldots, W_N, b_0, b_1, \ldots, n_N)$.

Example 1 *With the notation above, the output of a 3-layer feedforward neural network, consisting of one hidden layer is given as:*

$$f(\beta; \theta) := w^T \phi(W^T \beta + B) + b. \tag{2.88}$$

The input to the NN is given by $x \equiv \beta \in \mathbb{R}^{|\beta|}$. The NN has a hidden layer (i.e., $N = 2$) with M_1 units with weight matrix $W_1 \equiv W \in \mathbb{R}^{|\beta| \times M_1}$ and bias vector $b_1 \equiv B \in \mathbb{R}^{M_1}$, and a linear output layer of M_2 units with a weight matrix $W_2 \equiv w \in \mathbb{R}^{N \times M_2}$ and a bias vector $b_2 \equiv b \in \mathbb{R}^3$. ϕ represents the ReLU activation (or transfer) function of hidden units.
The architecture of the NN is presented in Figure 2.1, which can be interpreted as follows: the input layer takes in the input of the system. Each of M_1 hidden units computes the inner product of β and one of the columns of W. The hidden units add a bias B to the inner product and rectify this value at zero. The output layer is a linear combination of the hidden units, plus a final bias b. Intuitively, each hidden unit linearly partitions the input space into two parts based on W and B. In one part the unit is inactive with zero output, while in the other it is active with positive output. Together, all hidden units partition the state space into polytopes. In each of these polytopes, the NN model has flexibility to learn the local function structure.

The goal of the training process is to determine (or "learn", "train") the parameters θ that minimize a given objective function (also called *loss function*). One popular learning procedure to train NNs is through supervised learning, which is what we discuss next.

2.5.2 Supervised Learning

Thus far the impact of deep learning has largely been in supervised learning. In supervised learning, each (training) example is a pair consisting of an input value (e.g., images) and a desired output value (e.g., 'cat', 'dog', etc. depending on what is in the image). After learning on the training data, the system is expected to make correct predictions for future (unseen) inputs. Supervised learning can thus also be thought as a direct high dimensional regression (or classification).

In supervised learning, the optimal parameters θ^* can be obtained by solving the following

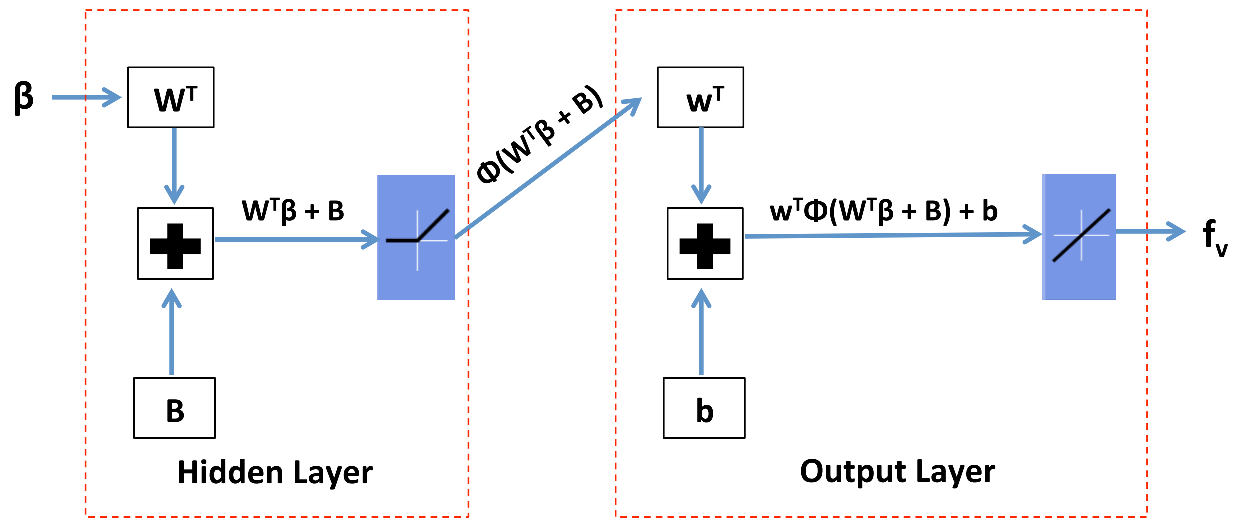


Figure 2.1: The neural network architecture for a 3 layer feed-forward ReLU network. The NN consists of three layers: an input layer, a hidden ReLU layer, and an output layer. The parameters to be learned during the training process are $\theta = (W, w, B, b)$.

optimization problem:

$$\theta^* = \min_{\theta} \sum_{i=1}^{|\mathcal{D}|} l\left(f(x_i;\theta), y_i\right), \tag{2.89}$$

given the dataset $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_{|\mathcal{D}|}, y_{|\mathcal{D}|})\}$. Here x_i represent the input vector, y_i represent the corresponding true output (also called *labels*), and $|\mathcal{D}|$ is the number of datapoints in the dataset. A popular choice of loss function in supervised learning is mean squared error (MSE), which is given as:

$$\theta^* = \min_{\theta} \sum_{i=1}^{|\mathcal{D}|} \|f(x_i;\theta) - y_i\|^2, \tag{2.90}$$

2.6 Gaussian Processes and Bayesian Optimization

2.6.1 Gaussian Process (GP)

Gaussian Processes (GPs) are a state-of-the-art probabilistic non-parametric regression methods [253]. Similar to a typical regression task, the goal is to find a nonlinear map, $f(x) : \mathbb{R}^n \to \mathbb{R}$, from an input vector $x \in \mathbb{R}^n$ to the function value $f(x)$.

In GP regression, we assume that function values $f(x)$, associated with different values of x, are random variables and that any finite number of these random variables have a joint

Gaussian distribution dependent on the values of x [253]. Thus, a GP is a distribution over functions

$$f \sim \mathcal{GP}\left(m, k\right), \tag{2.91}$$

fully defined by a prior mean function m and a covariance function (or kernel) k. The covariance function, $k(x_i, x_j)$, defines the covariance between any two function values, $f(x_i)$ and $f(x_j)$. The prior mean function is typically assumed to be zero without loss of generality.

The choice of kernel is often problem-dependent and encodes general assumptions such as smoothness of the unknown function. For example, one popular choice is squared exponential kernel, given by

$$k(x_i, x_j) = \sigma_f^2 \exp\left(-\tfrac{1}{2}(x_i - x_j)^T \mathbf{\Lambda}^{-1}(x_i - x_j)\right)\,, \tag{2.92}$$

with $\mathbf{\Lambda} = \text{diag}([l_1^2, ..., l_D^2])$. Both the characteristic length-scales l_i, and the variance σ_f^2 are open hyperparameters $\varphi = [l_1, ..., l_D, \sigma_f^2]$ to be selected. This selection is typically performed by minimizing the negative log marginal likelihood (NLML) of the observed data. In particular, given observations $\mathcal{D} = \{x_i, f(x_i)\}_{i=1}^n$, the NLML can be written as:

$$\begin{aligned} \text{NLML}(\varphi) &= -\log p(\boldsymbol{f}|\boldsymbol{X}, \varphi) \\ &\doteq \tfrac{1}{2}\boldsymbol{f}^T(\boldsymbol{K} + \sigma_w^2\boldsymbol{I})^{-1}\boldsymbol{f} + \tfrac{1}{2}\log|\boldsymbol{K} + \sigma_w^2\boldsymbol{I}| - \tfrac{n}{2}\log 2\pi\,. \end{aligned} \tag{2.93}$$

where $\boldsymbol{X}$ is the input matrix consisting of data points $\{x_i\}_{i=1}^n$, $\boldsymbol{K}$ is the kernel matrix with $\boldsymbol{K}_{ij} = k(x_i, x_j)$, and $\boldsymbol{f} = [f(x_1), \ldots, f(x_n)]$. Using the chain-rule, the gradients of the NLML can be computed analytically,

$$\frac{\partial \text{NLML}(\varphi)}{\partial \varphi} = \frac{\partial \text{NLML}(\varphi)}{\partial \boldsymbol{K}} \frac{\partial \boldsymbol{K}}{\partial \varphi}\,, \tag{2.94}$$

which allows to optimize the hyperparameters using standard gradient descent optimization [253]. Note that $\boldsymbol{K}$ depends on hyperparameters φ through k.

The GP framework can be used to predict the distribution of the performance function $f(x^*)$ at an arbitrary input x^* based on the past observations. Conditioned on $\mathcal{D}$, the mean and variance of the prediction are

$$\mu(x^*) = \boldsymbol{k}\boldsymbol{K}^{-1}\boldsymbol{f}; \quad \sigma^2(x^*) = k(x^*, x^*) - \boldsymbol{k}\boldsymbol{K}^{-1}\boldsymbol{k}^T\,, \tag{2.95}$$

where $\boldsymbol{K}$ is the kernel matrix as before, and $\boldsymbol{k} = [k(x_1, x^*), \ldots, k(x_n, x^*)]$. Thus, the GP provides both the expected value of the performance function at any arbitrary point x^* as well as a notion of the uncertainty of this estimate.

2.6.2 Bayesian Optimization (BO)

Bayesian optimization aims to find the global minimum of an unknown function $f(x)$ [184, 238, 276]. BO is particularly suitable for the scenarios where evaluating the unknown function is expensive, which is often the case in robotics as evaluating a function in real-world experiments is often time consuming. At each iteration, BO uses the past observations $\mathcal{D}$ to model the objective function, and uses this model to determine informative sample locations.

A common model used in BO for the underlying objective is Gaussian process (see Sec. 2.6.1). Using the mean and variance predictions of the GP from (2.95), BO computes the next sample location by optimizing the so-called acquisition function, $\alpha(\cdot)$. Different acquisition functions are used in the literature to trade off between exploration and exploitation during the optimization process [276]. For example, the next evaluation for expected improvement (EI) acquisition function [226] is given by $x^* = \arg\min_x \alpha(x)$ where

$$\alpha(x) = \sigma(x)[u\Phi(u) + \phi(u)]; \quad u = (\mu(x) - T)/\sigma(x). \tag{2.96}$$

$\Phi(\cdot)$ and $\phi(\cdot)$ in (2.96), respectively, are the standard normal cumulative distribution and probability density functions. The target value T is the minimum of all explored data so far. $\mu(x)$ and $\sigma(x)$ are the mean and variance of the GP that *models* the objective function $f(x)$ based on the data obtained so far. Intuitively, EI selects the next parameter point where the expected improvement over T is maximal. Repeatedly evaluating the system at points given by (2.96) thus improves the observed performance.

Note that optimizing $\alpha(x)$ in (2.96) does not require physical interactions with the system, but only evaluation of the GP model. When a new set of optimal parameters x^* is determined, they are finally evaluated on the real objective function $f(x)$.

Part I

Safety Analysis for Robotic Systems

Chapter 3

Scaling Safety Analysis: Algorithmic and Computational Fronts

This chapter is based on the papers "Hamilton-Jacobi Reachability: A Brief Overview and Recent Advances" [34], "Decomposition of Reachable Sets and Tubes for a Class of Nonlinear Systems" [75], "FaSTrack: a Modular Framework for Fast and Guaranteed Safe Motion Planning" [151], "Reachability-Based Safety Guarantees using Efficient Initializations" [150], "An Efficient Reachability-Based Framework for Provably Safe Autonomous Navigation in Unknown Environments" [29], and "Provably Safe and Scalable Multi-Vehicle Trajectory Planning" [35] written in collaboration with Mo Chen, Sylvia Herbert, Jaime Fisac, Andrea Bajcsy, Shromona Ghosh, Varun Tolani, Eli Bronstein, Mahesh Vashishtha, SooJean Han, and Claire Tomlin.

As the robotic systems we design grow more complex, determining whether they work according to specification and within given constraints is important. This is particularly important for safety-critical systems such as autonomous aircrafts, self-driving cars, drones. Consequently, we need algorithms that can actively reason about the safety of these systems as they work alongside other robots and humans.

Verification or safety analysis of robotic systems is challenging for many reasons. First, all possible system behaviors must be accounted for. This makes most simulation-based approaches insufficient, and thus formal verification methods are needed. Second, many practical systems are affected by disturbances in the environment, which can be unpredictable, and may even contain adversarial agents. In addition, these systems often have high dimensional state spaces and evolve in continuous time with complex, nonlinear dynamics. Third, during the operation time, the autonomous system may experience changes in system dynamics, external disturbances, and/or the surrounding environment, requiring updated safety guarantees.

Hamilton-Jacobi (HJ) reachability analysis is a verification method for guaranteeing performance and safety properties of systems, overcoming some of the above challenges. However, as we discussed in Section 2.3.3, HJ reachability becomes computationally intractable as the state space dimension increases (referred to as *curse of dimensionality*). In particular, its computational complexity scales exponentially with respect to the number of state variables. This makes it particularly challenging to run reachability computations online even for low-dimensional systems. **In this chapter, we will discuss our work on overcoming the curse of dimensionality in reachability analysis on multiple fronts by leveraging (a) the structure in dynamics to decompose a high-dimensional reachability problem into a number of smaller reachability problems [75], making computations of reachable sets orders of magnitude faster; (b) smart offline computations (or precomputations) that allow us to compute reachable sets efficiently in real time [151, 150, 29]; and (c) modern computational tools such as GPUs to parallelize the reachability computations, achieving a 100 fold speedup compared to existing implementations [35].**

3.1 Related Work

In this section, we discuss the related work on computing an approximation of reachable sets in an efficient fashion, both using HJ reachability as well as other verification approaches.

Verification of Dynamical Systems. An extensive body of research deals with verification of dynamical systems. We cannot hope to summarize all these works here, but we attempt to discuss several of the representative approaches.

In particular, satisfaction of properties such as safety, liveness, and fairness in computer software and in discrete-time dynamical systems can be verified by checking whether runs of a transition system, or words of a finite automaton satisfy certain desired properties [28, 53]. These properties may be specified by a variety of logical formalisms such as linear temporal logic. For specifications of properties of interest in autonomous robots, richer formalisms have been proposed. For example, propositional temporal logic over the reals [255, 108] allows specification of properties such as time in terms of real numbers, and chance-constrained temporal logic [160] allows specification of requirements in the presence of uncertainty. Besides autonomous cars and robots, verification approaches based on discrete models have also been successfully used in the context of intelligent transportation systems [87] and human-automation interaction [58].

For continuous and hybrid systems, safety properties can be verified by checking whether the forward reachable set or an over-approximation of it intersects with a set of undesirable states, akin to checking runs of transition systems. Numerous tools such as SpaceEx [119], Flow* [79], CORA [15], C2E2 [105, 109], and dReach [176] have been developed for this purpose; the authors in [104] present a tutorial on combining different tools for hybrid systems verification. In addition, methods that utilize semidefinite programming to search for Lyapunov functions can be used to verify safety [246, 287]. This is done, for example, by constructing barrier certificates [41] or funnels [205, 206] with Lyapunov properties.

Safety Analysis Using Backward Reachable Set. Outside of the realm of checking whether the set of possible future states of a system includes undesirable states, safety can also be verified by starting from known unsafe conditions and computing backward reachable sets, which the system should avoid. In general, the challenges facing verification methods include computational tractability, generality of system dynamics, existence of control and disturbance variables, and representation of sets [40, 222, 57, 115]. For example, the methods presented in [141, 119, 182, 183, 204, 135, 18] have had success in analyzing relatively high-dimensional affine systems using sets of pre-specified shapes, such as polytopes, hyperplanes, ellipsoids, and zonotopes. Other potentially less scalable methods are able to handle systems with the more complex dynamics [79, 15, 119, 207, 99, 149]. [207, 99, 149] are particularly well-suited to systems with polynomial dynamics. Computational scalability varies among these different methods, with the most scalable methods requiring that the system dynamics do not involve control and disturbance variables. The work in [236] accounts for both control and disturbances, but is only applicable to linear systems. Methods that can account for general nonlinear systems such as [16] also sometimes represent sets using simple shapes such as polytopes, potentially sacrificing representation fidelity in favor of the other aspects mentioned earlier. Hamilton-Jacobi (HJ) formulations [40, 222, 209, 57] excel in handling general nonlinear dynamics, control and disturbance variables, and flexible set representations via a grid-based approach; however, these methods are the least computationally scalable. Still other methods make a variety of other assumptions to make desirable trade offs [90, 145, 86]. In addition, under some special scenarios, it may be possible to obtain small computational benefits while minimizing trade offs in other axes of consideration by exploiting system or controller structure [217, 115, 223, 74, 171, 170].

Efficient Update of Backward Reachable Set. For real-time update of the reachable set in dynamic environment conditions, such as changes in system dynamics, external disturbances, and/or the surrounding environment, researchers have looked into warm-starting the reachable set computation. Warm-starting in the optimization community involves using an initialization that acts as a "best guess" of the solution, and therefore may converge in fewer iterations (if convergence can be achieved). Recent work applied this warm-starting idea to create a "discounted reachability" formulation for infinite-time horizon problems [13, 116]. By using a discount factor, this formulation *guarantees* convergence regardless of the initialization. However, convergence rates using this discount factor can be very slow, and in practice the analysis may not converge numerically when convergence thresholds are too tight, or may converge incorrectly when convergence thresholds are too lenient. In addition, parameter tuning of the discount factor can be time-intensive. These issues reduce the computational benefit of warm-start reachability.

Contributions and Chapter Organization

In Section 3.2, we present a system decomposition method for computing backward reachable sets (BRSs) and tubes (BRTs) of a class of nonlinear systems. Our method first computes BRSs for lower-dimensional subsystems, and then reconstructs the full-dimensional BRS without incurring additional approximation errors other than those arising from the lower dimensional computations. The computation of a BRS and a BRT now becomes orders of magnitude faster. Crucially, the subsystems can be coupled through common states, controls, and disturbances. The treatment of this coupling distinguishes our method from others which consider completely decoupled subsystems, potentially obtained through transformations, or achieve approximate reachable sets. Furthermore, we also provide the conditions under which the BRT computation can be decomposed into smaller subproblems.

To enable reachability computations online, we propose a method of “warm-start” reachability in Section 3.3. The proposed method uses a user-defined initialization (typically the previously computed solution) while computing the value function. By starting with a value function that is closer to the solution than the standard initialization, convergence takes significantly fewer iterations. Equipped with the warm starting technique, we then discuss how we can reduce computation time further by only locally updating the value function rather than in the entire state space.

In Section 3.4, we present BEACLS, a C++-based toolbox that parallelizes the numerical algorithms for solving the HJI PDE using graphics processing units (GPUs). This speeds up the computation time by nearly 100 times compared to state-of-the-art MATLAB implementations. GPU parallelization allows us to tractably solve many HJI PDEs, especially when they are required to be solved in real-time.

3.2 Reachability Decomposition

In this section, we seek to obtain the BRSs and the BRTs via computations in lower-dimensional subspaces under the assumption that the system dynamics $\dot{x} = f(x, u, d)$ in Equation (2.1) can be decomposed into *self-contained subsystems (SCS)* (3.2). Intuitively in self-contained subsystems, the evolution of each subsystem depends only on the subsystem states. Such a decomposition is common, since many systems involve components that are loosely coupled. In particular, the evolution of position variables in vehicle dynamics is often weakly coupled though other variables such as heading. We show that for SCS one can independently compute the reachable set for each subsystem and then “stitch” them together to obtain the reachable set for the original high-dimensional system. Since the reachability computation scales exponentially with the number of state variables, this decomposition allows us to compute a high-dimensional reachable set at a significantly lower complexity. In fact, we present several examples, such as a 10D quadrotor system, where it remains intractable to compute the exact high-dimensional reachable set without using a decomposition.

We will particularly focus on computing minimal and maximal BRSs and BRTs in this section. Recall, that a BRS represents the set of states $x \in \mathbb{R}^{n_x}$ from which the system can be driven into some set $\mathcal{L} \subseteq \mathbb{R}^{n_x}$ at the *end* of a time horizon, despite the best control efforts. Mathematically, the BRS is given by (Equation (2.39)):

$$\mathcal{A}(t) = \{x : \forall u(\cdot) \in \mathbb{U}_t^T, \exists d(\cdot) \in \Gamma_t^T, \xi(T; x, t, u(\cdot), d(\cdot)) \in \mathcal{L}\}, \qquad (\textbf{Minimal BRS})$$

The above BRS is sometimes also called *minimal BRS*. We are using $\mathcal{A}(t)$ to denote the minimal BRS instead of our standard notation of $\mathcal{V}(t)$ to distinguish it from what is called a *maximal BRS*:

$$\mathcal{R}(t) = \{x : \forall d(\cdot) \in \Gamma_t^T, \exists u(\cdot) \in \mathbb{U}_t^T, \xi(T; x, t, u(\cdot), d(\cdot)) \in \mathcal{L}\}, \qquad (\textbf{Maximal BRS})$$

The minimal BRS is typically useful when the target set represents some undesirable states that the system should avoid at all times (such as obstacles) and we want to compute from which states the system has a "chance" of avoiding these states. On the other hand, the maximal BRS is useful when the target set represents some desirable states, such as the goal states, and we are interested in computing from which states we can reach those goal states. Unsurprisingly, the complement of the minimal BRS of $\mathcal{L}$ is same as the maximal BRS of $\mathcal{L}^C$. Thus, the tools for computing a minimal BRS can be used to compute a maximal BRS as well. We refer the readers to Section 2.3.2 for more details on the computation of minimal and maximal BRSs.

One can similarly define minimal and maximal BRTs.

$$\bar{\mathcal{A}}(t) = \{x : \forall u(\cdot) \in \mathbb{U}_t^T, \exists d(\cdot) \in \Gamma_t^T, \exists s \in [t, T], \xi(s; x, t, u(\cdot), d(\cdot)) \in \mathcal{L}\}, \quad (\textbf{Minimal BRT})$$

$$\bar{\mathcal{R}}(t) = \{x : \forall d(\cdot) \in \Gamma_t^T, \exists u(\cdot) \in \mathbb{U}_t^T, \exists s \in [t, T], \xi(s; x, t, u(\cdot), d(\cdot)) \in \mathcal{L}\}. \quad (\textbf{Maximal BRT})$$

Note that unlike a BRS, the complement of the minimal BRT of $\mathcal{L}$ is *not* same as the maximal BRT of $\mathcal{L}^{C}$[1].

We now proceed with formal definitions of self-contained subsystems that are required to precisely state our main results regarding the computation of minimal and maximal BRS/BRT of a high-dimensional system.

3.2.1 Definitions

3.2.1.1 Subsystem Dynamics

Let the state $x \in \mathbb{R}^{n_x}$ be partitioned as $x = (x_1, x_2, x_c)$, with $x_1 \in \mathbb{R}^{n_1}, x_2 \in \mathbb{R}^{n_2}, x_c \in \mathbb{R}^{n_c}, n_1, n_2 > 0, n_c \geq 0$, $n_1 + n_2 + n_c = n_x$. Note that n_c could be zero. We call x_1, x_2, x_c "state partitions" of the system. Intuitively, x_1 and x_2, are states belonging to subsystems

[1]In fact, the complement of the minimal BRT of $\mathcal{L}$ is same as the viability kernel of $\mathcal{L}^C$ (see Section 2.3.2).

1 and 2, respectively, and x_c states belong to both subsystems. Under the above notation, the system dynamics (2.1) become

$$\begin{aligned}\dot{x}_1 &= f_1(x_1, x_2, x_c, u, d)\\ \dot{x}_2 &= f_2(x_1, x_2, x_c, u, d)\\ \dot{x}_c &= f_c(x_1, x_2, x_c, u, d)\end{aligned} \tag{3.1}$$

In general, depending on how the dynamics f depend on u and d, some state partitions may be independent of the control and/or disturbance.

We group these states into two subsystems by defining states $z_1 = (x_1, x_c) \in \mathbb{R}^{n_1+n_c}$ and $z_2 = (x_2, x_c) \in \mathbb{R}^{n_2+n_c}$, where z_1 and z_2 in general share the "common" states in x_c. Note that our theory is applicable to any finite number of subsystems defined in the analogous way, with $z_i = (x_i, x_c)$; however, without loss of generality (WLOG), we assume that there are just two subsystems.

Definition 1 ***Self-contained subsystem***. *Consider the following special case of* (3.1)*:*

$$\begin{aligned}\dot{x}_1 &= f_1(x_1, x_c, u, d)\\ \dot{x}_2 &= f_2(x_2, x_c, u, d)\\ \dot{x}_c &= f_c(x_c, u, d)\end{aligned} \tag{3.2}$$

We call each of the subsystems with states defined as $z_i = (x_i, x_c)$ a "self-contained subsystem" (SCS), or just "subsystem" for short.

Intuitively (3.2) means that the evolution of each subsystem depends only on the subsystem states: $\dot{z}_i$ depends only on $z_i = (x_i, x_c)$. Explicitly, the dynamics of the two subsystems are as follows:

$$\begin{array}{ll}\dot{x}_1 = f_1(x_1, x_c, u, d) & \dot{x}_2 = f_2(x_2, x_c, u, d)\\ \dot{x}_c = f_c(x_c, u, d) & \dot{x}_c = f_c(x_c, u, d)\\ \text{(Subsystem 1)} & \text{(Subsystem 2)}\end{array}$$

Note that the two subsystems are coupled through the common state partition x_c, control u, and disturbance d. When the subsystems are coupled through u, we say that the subsystems have "shared control". Similarly, when the subsystems are coupled through d, we say that the subsystems have "shared disturbance".

Example 2 *An example of a system that can be decomposed into SCSs is the Dubins Car with constant speed v:*

$$\begin{bmatrix} \dot{p}_x \\ \dot{p}_y \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos\theta + d_x \\ v\sin\theta + d_y \\ \omega \end{bmatrix}, \qquad \omega \in \mathcal{U}, \quad (d_x, d_y) \in \mathcal{D} \tag{3.3}$$

with state $x = (p_x, p_y, \theta)$ representing the $x - y$ position and heading, control $u = \omega$ representing the turn rate, and $d = (d_x, d_y)$ representing the disturbance. The state partitions are simply the system states: $x_1 = p_x, x_2 = p_y, x_c = \theta$. The subsystem states z_i and the subsystem dynamics are

$$\begin{aligned} \dot{z}_1 &= \begin{bmatrix} \dot{x}_1 \\ \dot{x}_c \end{bmatrix} = \begin{bmatrix} \dot{p}_x \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos\theta + d_x \\ \omega \end{bmatrix} \\ \dot{z}_2 &= \begin{bmatrix} \dot{x}_2 \\ \dot{x}_c \end{bmatrix} = \begin{bmatrix} \dot{p}_y \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\sin\theta + d_y \\ \omega \end{bmatrix} \\ u &= \omega \end{aligned} \tag{3.4}$$

where the overlapping state is θ, and the subsystem controls and their shared component is the control u itself. In this example, there is no shared disturbance between the two subsystems.

Although there may be common or overlapping states in z_1 and z_2, the evolution of each subsystem does not depend on the other explicitly. In fact, if we for example entirely ignore the subsystem z_2, the evolution of the subsystem z_1 is well-defined and can be considered a full system on its own; hence, each subsystem is self-contained.

3.2.1.2 Projection Operators

For the projection operators, it will be helpful to refer to Fig. 3.1. Define the projection of a state $x = (x_1, x_2, x_c)$ onto a subsystem state space $\mathbb{R}^{n_i+n_c}$ as

$$\mathcal{P}_i(x) = z_i = (x_i, x_c) \tag{3.5}$$

This projects a point in the full dimensional state space onto a point in the subsystem state space. Also define the back-projection operator to be

$$\mathcal{P}^{-1}(z_i) = \{x \in \mathbb{R}^{n_x} : (x_i, x_c) = z_i\} \tag{3.6}$$

This back-projection lifts a point from the subsystem state space to a set in the full dimensional state space. We will also need the ability to apply the back-projection operator on sets. In this case, we overload the back-projection operator:

$$\mathcal{P}^{-1}(\mathcal{S}_i) = \{x \in \mathbb{R}^{n_x} : \exists z_i \in \mathcal{S}_i, (x_i, x_c) = z_i\} \tag{3.7}$$

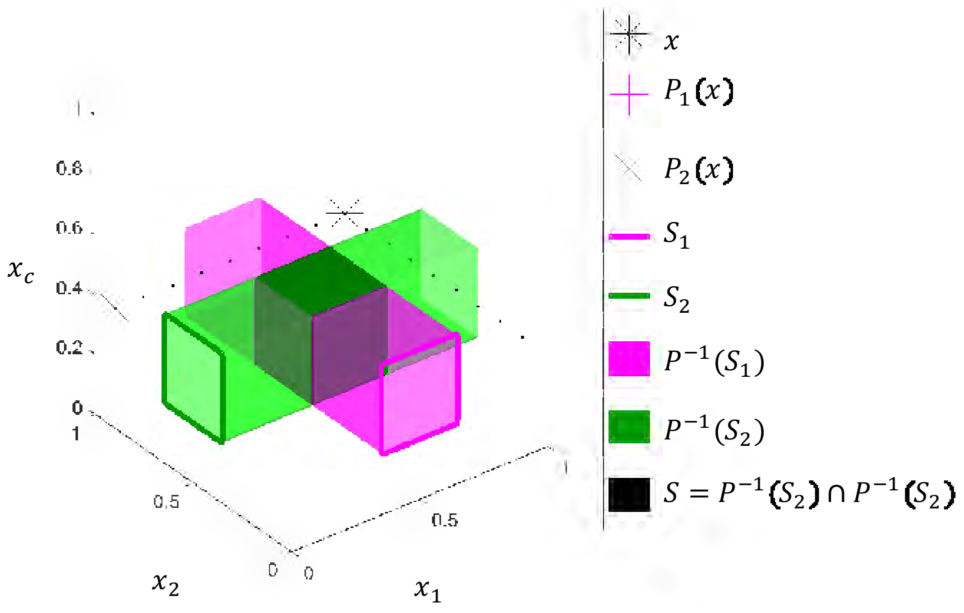


Figure 3.1: This figure shows the back-projection of sets in the x_1-x_c plane $\mathcal{S}_1$ and the x_2-x_c plane ($\mathcal{S}_2$) to the 3D space to form the intersection shown as the black cube ($\mathcal{S}$). The figure also shows projection of a point x onto the lower-dimensional subspaces in the x_1-x_c and x_2-x_c planes.

3.2.2 Decomposition of Backward Reachable Sets Using Self-Contained Subsystems

We are now ready to formalize the conditions under which we can compute the BRS of a high-dimensional system using self-contained subsystems, and how to actually obtain the BRS of the high-dimensional system from the BRSs of SCS. We start with the following two assumptions that are required to decompose the reachability problem.

Assumption 1 *The system dynamics $f(x, u, d)$ can be decomposed into self-contained subsystems as defined in* (3.2) *with subsystem states $z_1 \in \mathbb{R}^{n_1+n_c}$ and $z_2 \in \mathbb{R}^{n_2+n_c}$.*

Assumption 2 *We assume that the full system target set $\mathcal{L} \in \mathbb{R}^{n_x}$ can be written in terms of the subsystem target sets $\mathcal{L}_1 \subseteq \mathbb{R}^{n_1+n_c}, \mathcal{L}_2 \subseteq \mathbb{R}^{n_1+n_c}$ in one of the following ways:*

$$\mathcal{L} = \mathcal{P}^{-1}(\mathcal{L}_1) \cap \mathcal{P}^{-1}(\mathcal{L}_2) \tag{3.8}$$

where the full target set is the intersection of the back-projections of subsystem target sets, or

$$\mathcal{L} = \mathcal{P}^{-1}(\mathcal{L}_1) \cup \mathcal{P}^{-1}(\mathcal{L}_2) \tag{3.9}$$

where the full target set is the union of the back-projections of subsystem target sets.

Fig. 3.1 helps provide intuition for Assumption 2: applying (3.8) to $\mathcal{S}_1$ and $\mathcal{S}_2$ results in the black cube. Applying (3.9) would result in the cross-shaped set encompassing both $\mathcal{P}^{-1}(\mathcal{S}_1)$ and $\mathcal{P}^{-1}(\mathcal{S}_2)$.

Equipped with Assumptions 1 and 2, we now discuss what the BRS reconstruction process looks like depending on whether the target set is decomposed as a union or a intersection, whether there is a shared control among SCS or not, and whether there is a shared disturbance. We will discuss all possible combinations of the above three factors. The key results of this section are summarized in Table 3.1.

Table 3.1: Summary of possible decompositions of the BRS and the BRT, whether they are possible, and if so whether they are exact or conservative. Exact means that no additional approximation errors are introduced. Note that in the cases marked "no" for shared control (or shared disturbance), the results hold for both decoupled control (or disturbance) and for no control (or disturbance). All cases shown are for scenarios with shared states, with the shared states being x_c in (3.2); in the case that there are no shared states this becomes a straightforward decoupled system.

Shared Controls	**Yes**		**No**		**Yes**		**No**	
Shared Disturbance	**No**		**No**		**Yes**		**Yes**	
Target	**Intersection**	**Union**	**Intersection**	**Union**	**Intersection**	**Union**	**Intersection**	**Union**
Recover Maximal BRS?	No	Yes, exact	Yes, exact	Yes, exact	No	Yes, cons	Yes, cons	Yes, cons
Result: Maximal BRS	N/A	Thm 1	Prop 1	Thm 1	N/A	Prop 3	Prop 5	Prop 3
Recover Minimal BRS?	Yes, exact	No	Yes, exact	Yes, exact	Yes, cons	No	Yes, cons	Yes, cons
Result: Minimal BRS	Thm 2	N/A	Thm 2	Prop 2	Prop 4	N/A	Prop 4	Prop 6
Recover Maximal BRT?[2]	No	Yes, cons	Yes, cons	Yes, cons	No	Yes, cons	Yes, cons	Yes, cons
Result: Maximal BRT	N/A	Thm 1, Prop 7	Prop 1, 7	Thm 1, Prop 7	N/A	Prop 3, 7	Prop 5, 7	Prop 3, 7
Recover Minimal BRT?[3]	Yes, exact	No	Yes, exact	Yes, exact	Yes, cons	No	Yes, cons	Yes, cons
Result: Minimal BRT	Thm 2, 3	N/A	Thm 2, 3	Prop 2, Thm 3	Prop 4, Thm 3	N/A	Prop 4, Thm 3	Prop 6, Thm 3

3.2.2.1 Shared Control, No Shared Disturbance

When the SCS share control but not disturbance, the dynamics in (3.2) can be specialized as:

$$\begin{aligned} \dot{x}_1 &= f_1(x_1, x_c, u, d_1) \\ \dot{x}_2 &= f_2(x_2, x_c, u, d_2) \\ \dot{x}_c &= f_c(x_c, u) \end{aligned} \tag{3.10}$$

where the subsystem disturbance do not have any shared components, so that we have $d = (d_1, d_2)$. In this scenario, the full-dimensional BRS can be reconstructed without incurring additional approximation errors from lower-dimensional BRSs in the situations stated in Theorem 1 and 2.

If $\mathcal{L}$ represents states the system aims to reach, then $\mathcal{R}(t)$ represents the set of states from which $\mathcal{L}$ can be reached. If the system goal states are the union of subsystem goal

[2]If additionally the disturbance is zero in the cases when the subsystems do not have a shared disturbance components, the conservative results become exact.

[3]The solution here can be found only if the minimum BRSs are non-empty for the entire time period.

states, then it suffices for any subsystem to reach its subsystem goal states, regardless of any coupling that exists between the subsystems. Theorem 1 states this intuitive result.

Theorem 1 *Suppose that the full system dynamics $\dot{x} = f(x, u, d)$ can be decomposed into the form of* (3.10)*. Moreover, suppose that the target set $\mathcal{L}$ can be written as a union of the subsystem target sets then the maximal BRS can be written as a union of subsystems' maximal BRSs, i.e.,*

$$\mathcal{L} = \mathcal{P}^{-1}(\mathcal{L}_1) \cup \mathcal{P}^{-1}(\mathcal{L}_2) \quad \Rightarrow \quad \mathcal{R}(t) = \mathcal{P}^{-1}(\mathcal{R}_1(t)) \cup \mathcal{P}^{-1}(\mathcal{R}_2(t)) \tag{3.11}$$

Proof: *We will prove the following equivalent statement:*

$$\bar{x} \in \mathcal{R}(t) \Leftrightarrow \bar{x} \in \mathcal{P}^{-1}(\mathcal{R}_1(t)) \cup \mathcal{P}^{-1}(\mathcal{R}_2(t)) \quad \forall \bar{x} \tag{3.12}$$

Define $\bar{z}_i = \mathcal{P}_i(\bar{x})$ and $\xi_i(T; \bar{z}_i, t, u(\cdot), d_i(\cdot)) = \mathcal{P}_i(\xi(T; \bar{x}, t, u(\cdot), d(\cdot))$. We first prove the backward direction.

If $\bar{x} \in \mathcal{P}^{-1}(\mathcal{R}_1(t)) \cup \mathcal{P}^{-1}(\mathcal{R}_2(t))$, we must have

$$\bar{z}_1 \in \mathcal{R}_1(t) \vee \bar{z}_2 \in \mathcal{R}_2(t),$$

where $\vee$ represents an or operator. Without loss of generality, assume $\bar{z}_1 \in \mathcal{R}_1(t)$. By the subsystem maximal BRS definition, this is equivalent to

$$\forall d_1(\cdot), \exists u(\cdot), \xi_1(T; \bar{z}_1, t, u(\cdot), d_1(\cdot)) \in \mathcal{L}_1$$

Since $d_1(\cdot)$ and $d_2(\cdot)$ have no common components, the above statement is equivalent to

$$\forall d(\cdot), \exists u(\cdot), \xi_1(T; \bar{z}_1, t, u(\cdot), d_1(\cdot)) \in \mathcal{L}_1$$

Considering the backprojection operator, we have:

$$\forall d(\cdot), \exists u(\cdot), \mathcal{P}^{-1}\left(\xi_1(T; \bar{z}_1, t, u(\cdot), d_1(\cdot))\right) \in \mathcal{P}^{-1}(\mathcal{L}_1),$$

which can be equivalently written as

$$\forall d(\cdot), \exists u(\cdot), \xi(T; \bar{x}, t, u(\cdot), d(\cdot)) \in \mathcal{P}^{-1}(\mathcal{L}_1), \tag{3.13}$$

Noting that $\mathcal{L} \subseteq \mathcal{P}^{-1}(\mathcal{L}_1)$, (3.13) *is equivalent to $\bar{x} \in \mathcal{R}(t)$). This proves the backward direction.*

For the forward direction, we begin with $\bar{x} \in \mathcal{R}(t)$, which by the definition of a maximal BRS is equivalent to

$$\forall d(\cdot), \exists u(\cdot), \xi(T; \bar{x}, t, u(\cdot), d_1(\cdot)) \in \mathcal{L}, \tag{3.14}$$

which in turn can be written as

$$\forall d_1(\cdot), d_2(\cdot), \exists u(\cdot), \xi_1(T; \bar{z}_1, t, u(\cdot), d_1(\cdot)) \in \mathcal{L}_1 \vee \xi_2(T; \bar{z}_2, t, u(\cdot), d_2(\cdot)) \in \mathcal{L}_2, \tag{3.15}$$

Finally, distributing "$\exists u(\cdot)$" gives $\bar{x} \in \mathcal{P}^{-1}(\mathcal{R}_1(t)) \cup \mathcal{P}^{-1}(\mathcal{R}_2(t))$. ■

Now, we turn our attention to the minimal BRS. If $\mathcal{L}$ represents the set of unsafe states, then $\mathcal{A}(t)$ is the set of states from which the system will be driven into danger. Thus outside of $\mathcal{A}(t)$, there exists a control for the system to avoid the unsafe states. For the system to avoid $\mathcal{L}$, it suffices to avoid the unsafe states in either subsystem, regardless of any coupling that exists between the subsystems. Theorem 2 formally states this intuitive result.

Theorem 2 *Suppose that the full system dynamics $\dot{x} = f(x, u, d)$ can be decomposed into the form of* (3.10)*. Moreover, suppose that the target set $\mathcal{L}$ can be written as an intersection of the subsystem target sets then the minimal BRS can be written as an intersection of subsystems' minimal BRSs, i.e.,*

$$\mathcal{L} = \mathcal{P}^{-1}(\mathcal{L}_1) \cap \mathcal{P}^{-1}(\mathcal{L}_2) \quad \Rightarrow \quad \mathcal{A}(t) = \mathcal{P}^{-1}(\mathcal{A}_1(t)) \cap \mathcal{P}^{-1}(\mathcal{A}_2(t)) \tag{3.16}$$

Proof: *The proof of Theorem 2 follows that of Theorem 1. In particular, we will prove the following equivalent statement:*

$$\bar{x} \notin \mathcal{A}(t) \Leftrightarrow \bar{x} \notin \mathcal{P}^{-1}(\mathcal{A}_1(t)) \cap \mathcal{P}^{-1}(\mathcal{A}_2(t)) \;\; \forall \bar{x} \tag{3.17}$$

The above statement is equivalent to

$$\bar{x} \in \mathcal{A}^C(t) \Leftrightarrow \bar{x} \in \left[\mathcal{P}^{-1}(\mathcal{A}_1(t))\right]^C \cup \left[\mathcal{P}^{-1}(\mathcal{A}_2(t))\right]^C \tag{3.18}$$

By noting that the complement of the minimal BRS is same as the maximal BRS of $\mathcal{L}^C$, we can proceed in the same fashion as the proof of Theorem 1, with "$\mathcal{R}(t)$" replaced with "$\mathcal{A}^C(t)$", "$\mathcal{P}^{-1}(\mathcal{R}_i(t))$" replaced with "$\left[\mathcal{P}^{-1}(\mathcal{A}_i(t))\right]^C$", and "$\mathcal{L}$", "$\mathcal{L}_i$" replaced with "$\mathcal{L}^C$", "$\mathcal{L}_i^C$", respectively in Equation (3.18). ■

The conditions for reconstruction of the maximal BRS for an intersection of targets, as well as the minimal BRS for a union of targets, are more complicated. Consider for example the case where we are interested in computing the minimal BRS using that of subsystems' minimal BRSs. By computing the minimal BRS of each subsystem, we can obtain the optimal control such that the subsystem state will avoid the corresponding target set. If we are interested in avoiding the intersection of the target sets, we can simply apply optimal control obtained from any of the subsystem BRS because avoiding any of the target sets is sufficient. However, when we want to avoid the union of the target states, we need to avoid both target sets. Consequently, there might be the states of the actual system where applying optimal control from one subsystem push the system towards the target set of the other. Such states are called "leaky corners", which make the reconstruction results non-trivial. We defer studying the effects of leaky corners on the BRS reconstruction to future work. Table 3.2 summarizes the results from this section.

Table 3.2: BRS reconstruction from self-contained subsystems when they have shared control but no shared disturbance (Section 3.2.2.1)

Shared Controls	**Yes**	
Shared Disturbance	**No**	
Target	**Intersection**	**Union**
Recover Maximal BRS?	No	Yes, exact
Recover Minimal BRS?	Yes, exact	No
Result	Theorem 2	Theorem 1

Remark 4 *Even though we focus on computing the BRS in this section, since the FRS computation can be posed as a BRS computation (see Section 2.3.2), the ideas presented in this section can also be applied to compute FRS.*

Numerical Example: The Dubins Car The Dubins Car is a well-known system whose dynamics are given by (3.3). This system is only 3D, and its BRS can be tractably computed in the full-dimensional space, so we use it to compare the full formulation with our decomposition method. The Dubins Car dynamics can be decomposed according to (3.4). For illustration purposes, we assume that there are no disturbance in dynamics, i.e., $d_x = 0, d_y = 0$. For this example, we computed the BRS from the target set representing positions near the origin in both the p_x and p_y dimensions:

$$\mathcal{L} = \{(p_x, p_y, \theta) : |p_x|, |p_y| \leq 0.5\} \tag{3.19}$$

Such a target set $\mathcal{L}$ can be used to model an obstacle that the vehicle must avoid. Given $\mathcal{L}$, the interpretation of the BRS $\mathcal{A}(t)$ is the set of states from which a collision with the obstacle may occur after a duration of $|T - t|$. Without loss of generality, we set $T = 0$. From $\mathcal{L}$, we computed the BRS $\mathcal{A}(t)$ at $t = -0.5$. The resulting full BRS is shown in Fig. 3.2 as the red surface which appears in the bottom subplots. To compute the BRS using our decomposition method, we write the unsafe set $\mathcal{L}$ as

$$\begin{aligned} &\mathcal{L}_1 = \{(p_x, \theta) : |p_x| \leq 0.5\}, \mathcal{L}_2 = \{(p_y, \theta) : |p_y| \leq 0.5\} \\ &\mathcal{L} = \mathcal{P}^{-1}(\mathcal{L}_1) \cap \mathcal{P}^{-1}(\mathcal{L}_2) \end{aligned} \tag{3.20}$$

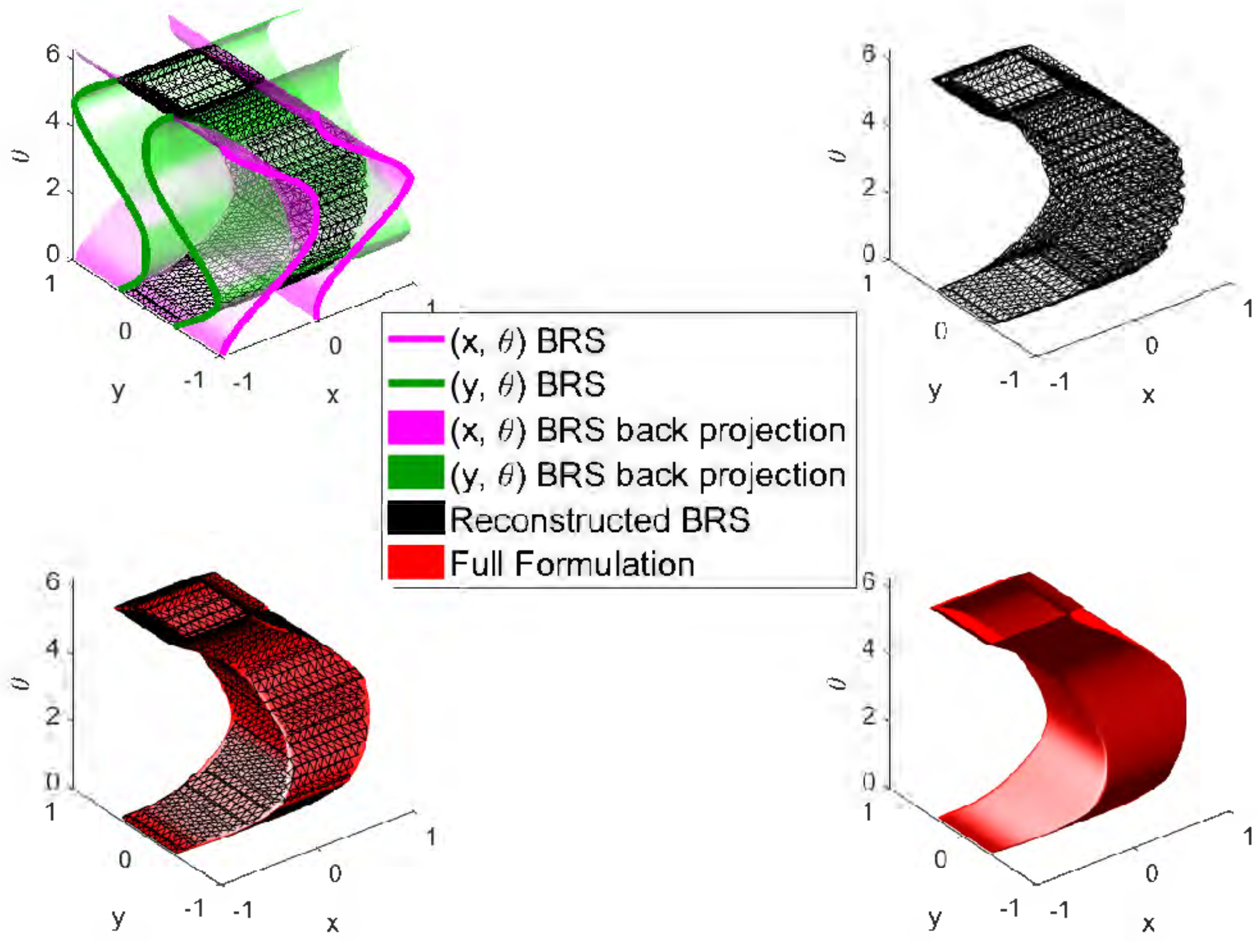


Figure 3.2: Comparison of the Dubins Car BRS $\mathcal{A}(t = -0.5)$ computed using the full formulation and via decomposition. Left top: The BRSs in the lower-dimensional subspaces and how they are combined to form the full-dimensional BRS. Top right: The BRS computed via decomposition. Bottom left: The BRSs computed using both methods, superimposed, showing that they are indistinguishable. Bottom right: The BRS computed using the full formulation.

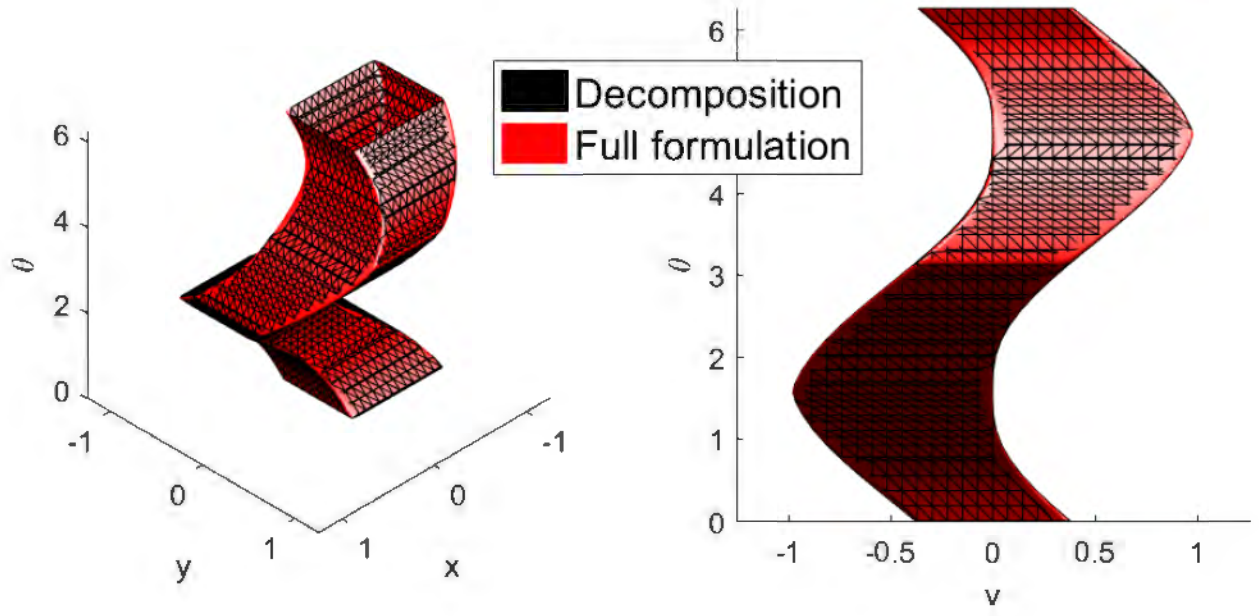


Figure 3.3: The Dubins Car BRS $\mathcal{A}(t = -0.5)$ computed using the full formulation and via decomposition, other view angles.

From $\mathcal{L}_1$ and $\mathcal{L}_2$, we computed the lower-dimensional BRSs $\mathcal{A}_1(t)$ and $\mathcal{A}_2(t)$, and then reconstructed the full-dimensional BRS $\mathcal{A}(t)$ using Theorem 2: $\mathcal{A}(t) = \mathcal{P}^{-1}(\mathcal{A}_1(t)) \cap \mathcal{P}^{-1}(\mathcal{A}_2(t))$. The subsystem BRSs and their back-projections are shown in magenta and green in the top left subplot of Fig. 3.2. The reconstructed BRS is shown in the top left, top right, and bottom left subplots of Fig. 3.2 (black mesh). In the bottom left subplot of Fig. 3.2, we superimpose the full-dimensional BRS computed using the two methods. We show the comparison of the computation results viewed from two different angles in Fig. 3.3. The results are indistinguishable.

Theorem 2 allows the computation to be performed in lower-dimensional subspaces, which is significantly faster. Another benefit of the decomposition method is that in the numerical methods for solving the HJI PDE, the amount of numerical dissipation increases with the number of state dimensions. Thus, computations in lower-dimensional subspaces lead to a slightly more accurate numerical solution.

The computation benefits of using our decomposition method can be seen from Fig. 3.4. The plot shows, in log-log scale, the computation time in seconds versus the number of grid points per dimension in the numerical computation. One can see that the direct computation of the BRS in 3D becomes very time-consuming as the number of grid points per dimension is increased, while the computation via decomposition hardly takes any time in comparison. Directly computing the BRS with 251 grid points per dimension in 3D took approximately 80 minutes, while computing the BRS via decomposition in 2D only took approximately 30 seconds! The computations were timed on a computer with an Intel Core i7-2600K processor and 16GB of random-access memory.

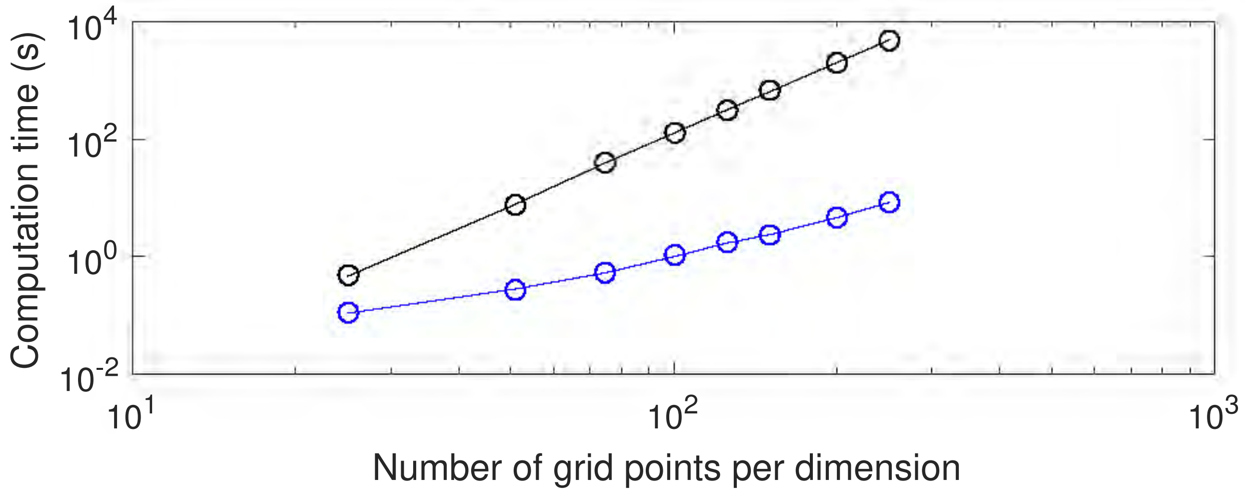


Figure 3.4: Computation times of the two methods in log scale for the Dubins Car. The time of the direct computation in 3D increases rapidly with the number of grid points per dimension. In contrast, computation times in 2D with decomposition are negligible in comparison.

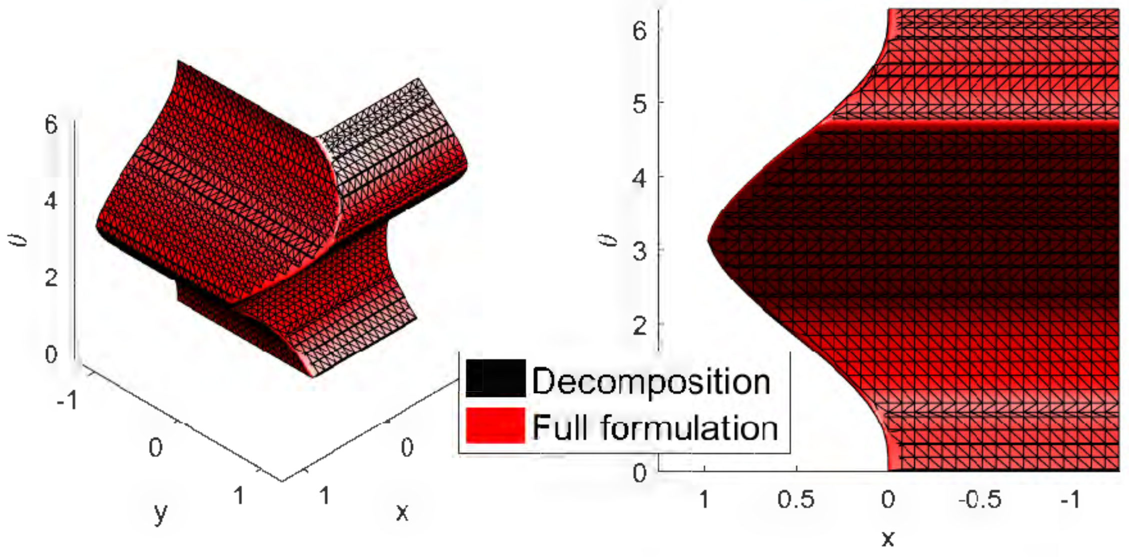


Figure 3.5: Comparison of the $\mathcal{R}(t)$ computed using our decomposition method and the full formulation. The computation results are indistinguishable. Note that the surface shows the boundary of the set; the set itself is on the "near" side of the left subplot, and the left side of the right subplot.

Fig. 3.5 illustrates Theorem 1. We chose the target set to be $\mathcal{L} = \{(p_x, p_y, \theta) : p_x \leq 0.5 \vee p_y \leq 0.5\}$, and computed the BRS $\mathcal{R}(t), t = -0.5$ via decomposition. No additional approximation error is incurred in the reconstruction process. The target set can be written as $\mathcal{L} = \mathcal{P}^{-1}(\mathcal{L}_1) \cup \mathcal{P}^{-1}(\mathcal{L}_1)$ where $\mathcal{L}_1 = \{(p_x, \theta) : p_x \leq 0.5\}, \mathcal{L}_2 = \{(p_y, \theta) : p_y \leq 0.5\}$.

3.2.2.2 No Shared Control, No Shared Disturbance

When the SCS do not share control or disturbance, the dynamics in (3.2) can be specialized as:

$$\begin{aligned}\dot{x}_1 &= f_1(x_1, x_c, u_1, d_1)\\ \dot{x}_2 &= f_2(x_2, x_c, u_2, d_2)\\ \dot{x}_c &= f_c(x_c)\end{aligned} \tag{3.21}$$

where in addition to disturbance the subsystem control do not have any shared components, so that we have $u = (u_1, u_2)$.

In addition to the results of Sec. 3.2.2.1, we can additionally obtain the maximal BRS even when the target set is the intersection of subsystem target sets, as well as the minimal BRS when the target set is the union of subsystem target sets. The following propositions formalize these results:

Proposition 1 *Suppose that the full system dynamics $\dot{x} = f(x, u, d)$ can be decomposed into the form of (3.21). Moreover, suppose that the target set $\mathcal{L}$ can be written as an intersection of the subsystem target sets then the maximal BRS can be written as an intersection of subsystems' maximal BRSs, i.e.,*

$$\mathcal{L} = \mathcal{P}^{-1}(\mathcal{L}_1) \cap \mathcal{P}^{-1}(\mathcal{L}_2) \quad \Rightarrow \quad \mathcal{R}(t) = \mathcal{P}^{-1}(\mathcal{R}_1(t)) \cap \mathcal{P}^{-1}(\mathcal{R}_2(t)) \tag{3.22}$$

Proof: *We will prove the following equivalent statement:*

$$\bar{x} \in \mathcal{R}(t) \Leftrightarrow \bar{x} \in \mathcal{P}^{-1}(\mathcal{R}_1(t)) \cap \mathcal{P}^{-1}(\mathcal{R}_2(t)) \quad \forall \bar{x} \tag{3.23}$$

By the definition of a maximal BRS, we have

$$\bar{x} \in \mathcal{R}(t) \Leftrightarrow \forall d(\cdot), \exists u(\cdot), \xi(T; \bar{x}, t, u(\cdot), d(\cdot)) \in \mathcal{L},$$

or equivalently

$$\Leftrightarrow \forall (d_1(\cdot), d_2(\cdot)), \exists (u_1(\cdot), u_2(\cdot)), \xi(T; \bar{x}, t, u(\cdot), d(\cdot)) \in \mathcal{L}.$$

Since $\mathcal{L} = \mathcal{P}^{-1}(\mathcal{L}_1) \cap \mathcal{P}^{-1}(\mathcal{L}_2)$, the above statement can equivalently be written as

$$\Leftrightarrow \forall (d_1(\cdot), d_2(\cdot)), \exists (u_1(\cdot), u_2(\cdot)), \xi_1(T; \bar{z}_1, t, u_1(\cdot), d_1(\cdot)) \in \mathcal{L}_1 \wedge \xi_2(T; \bar{z}_2, t, u_2(\cdot), d_2(\cdot)) \in \mathcal{L}_2.$$

Finally, noting that since $(u_1(\cdot), u_2(\cdot))$ and $(d_1(\cdot), d_2(\cdot))$ have no shared components, we have

$$\Leftrightarrow \forall d_1(\cdot), \exists u_1(\cdot), \xi_1(T; \bar{z}_1, t, u_1(\cdot), d_1(\cdot)) \in \mathcal{L}_1 \wedge \forall d_2(\cdot), \exists u_2(\cdot), \xi_2(T; \bar{z}_2, t, u_2(\cdot), d_2(\cdot)) \in \mathcal{L}_2$$

$$\Leftrightarrow \bar{z}_1 \in \mathcal{R}_1(t) \wedge \bar{z}_2 \in \mathcal{R}_2(t),$$
$$\Leftrightarrow \bar{x} \in \mathcal{P}^{-1}(\mathcal{R}_1(t)) \cap \mathcal{P}^{-1}(\mathcal{R}_2(t)).$$

■

Proposition 2 *Suppose that the full system dynamics $\dot{x} = f(x, u, d)$ can be decomposed into the form of (3.21). Moreover, suppose that the target set $\mathcal{L}$ can be written as a union of the subsystem target sets then the minimal BRS can be written as a union of subsystems' minimal BRSs, i.e.,*

$$\mathcal{L} = \mathcal{P}^{-1}(\mathcal{L}_1) \cup \mathcal{P}^{-1}(\mathcal{L}_2) \quad \Rightarrow \quad \mathcal{A}(t) = \mathcal{P}^{-1}(\mathcal{A}_1(t)) \cup \mathcal{P}^{-1}(\mathcal{A}_2(t)) \tag{3.24}$$

Proof: *This proof follows the same arguments as Proposition 1, but with "$\mathcal{R}$", "$\cap$", "$\exists$" replaced with "$\mathcal{A}$", "$\cup$", "$\forall$", respectively.* ■

Intuitively, when the subsystem controls have shared components, the control chosen by each subsystem may not agree with the other. This is the intuition behind why the results of Propositions 1 and 2 only hold true when there are no shared components in the subsystem controls. Note that the theorems hold despite the state coupling between the subsystems. The results from this section are summarized in Table 3.3.

Table 3.3: The BRS reconstruction from self-contained subsystems when they do not have shared control or disturbance (Section 3.2.2.2).

Shared Controls	**No**	
Shared Disturbance	**No**	
Target	**Intersection**	**Union**
Recover Maximal BRS?	Yes, exact	Yes, exact
Recover Minimal BRS?	Yes, exact	Yes, exact
Result	Proposition 1 Theorem 2	Theorem 1 Proposition 2

Numerical Example: The 10D Near-Hover Quadrotor The 10D Near-Hover Quadrotor was used for experiments involving learning-based MPC [59]. Its dynamics are

$$\begin{bmatrix} \dot{p}_x \\ \dot{v}_x \\ \dot{\theta}_x \\ \dot{\omega}_x \\ \dot{p}_y \\ \dot{v}_y \\ \dot{\theta}_y \\ \dot{\omega}_y \\ \dot{p}_z \\ \dot{v}_z \end{bmatrix} = \begin{bmatrix} v_x + d_x \\ g \tan \theta_x \\ -d_1\theta_x + \omega_x \\ -d_0\theta_x + n_0 S_x \\ v_y + d_y \\ g \tan \theta_y \\ -d_1\theta_y + \omega_y \\ -d_0\theta_y + n_0 S_y \\ v_z + d_z \\ k_T T_z - g \end{bmatrix} \tag{3.25}$$

where (p_x, p_y, p_z) denotes the position, (v_x, v_y, v_z) denotes the velocity, (θ_x, θ_y) denotes the pitch and roll, and (ω_x, ω_y) denotes the pitch and roll rates. The controls of the system are (S_x, S_y), which respectively represent the desired pitch and roll angle, and T_z, which represents the vertical thrust. The system experiences the disturbance (d_x, d_y, d_z) which represents wind in the three axes. g denotes the acceleration due to gravity. The parameters d_0, d_1, n_0, k_T, as well as the control bounds $\mathcal{U}$, that we used were $d_0 = 10, d_1 = 8, n_0 = 10, k_T = 0.91, |u_x|, |u_y| \leq 10$ degrees$, 0 \leq u_z \leq 2g, |d_x|, d_y \leq 0.5$ m/s$, |d_z| \leq 1$ m/s. The system can be fully decoupled into three subsystems of 4D, 4D, and 2D, respectively:

$$z_1 = (p_x, v_x, \theta_x, \omega_x), z_2 = (p_y, v_y, \theta_y, \omega_y), z_3 = (p_z, v_z) \tag{3.26}$$

The target set is chosen to be

$$\begin{aligned} \mathcal{L} = \{(p_x, v_x, \theta_x, \omega_x, p_y, v_y, \theta_y, \omega_y, p_z, v_z) :& \\ |p_x|, |p_y| \leq 1, |p_z| \leq 2.5\}& \end{aligned} \tag{3.27}$$

This target set can be written as $\mathcal{L} = \bigcap_{i=1}^{3} \mathcal{P}^{-1}(\mathcal{L}_{z_i})$, where $\mathcal{P}^{-1}(\mathcal{L}_{z_i}), i = 1, 2, 3$ are given by

$$\begin{aligned} \mathcal{L}_1 &= \{(p_x, v_x, \theta_x, \omega_x) : |p_x| \leq 1\} \\ \mathcal{L}_2 &= \{(p_y, v_y, \theta_y, \omega_y) : |p_y| \leq 1\} \\ \mathcal{L}_3 &= \{(p_z, v_z) : |p_z| \leq 2.5\} \end{aligned} \tag{3.28}$$

Since the subsystems do not have any common controls or disturbances, and $\mathcal{L} = \bigcap_{i=1}^{3} \mathcal{P}^{-1}(\mathcal{L}_{z_i})$, we can compute the full-dimensional $\mathcal{R}(t)$ and $\bar{\mathcal{R}}(t)$ by reconstructing lower-dimensional BRSs and BRTs. Again, without loss of generality, we set $T = 0$.

From the target set, we computed the 10D BRS and the BRT, $\mathcal{R}(s), \bar{\mathcal{R}}(s), s \in [-1, 0]$. In the left subplot of Fig. 3.6, we show a 3D slice of the BRS and the BRT sliced at $(v_x, v_y, v_z) = (-1.5, -1.8, 1.2), \theta_x = \theta_y = \omega_x = \omega_y = 0$. The colored sets show the slice of the BRSs $\mathcal{R}(s), s \in [-1, 0]$, with the times color-coded according to the legend. The slice of the BRT is shown as the black surface; the BRT is the union of BRSs by Proposition 7.

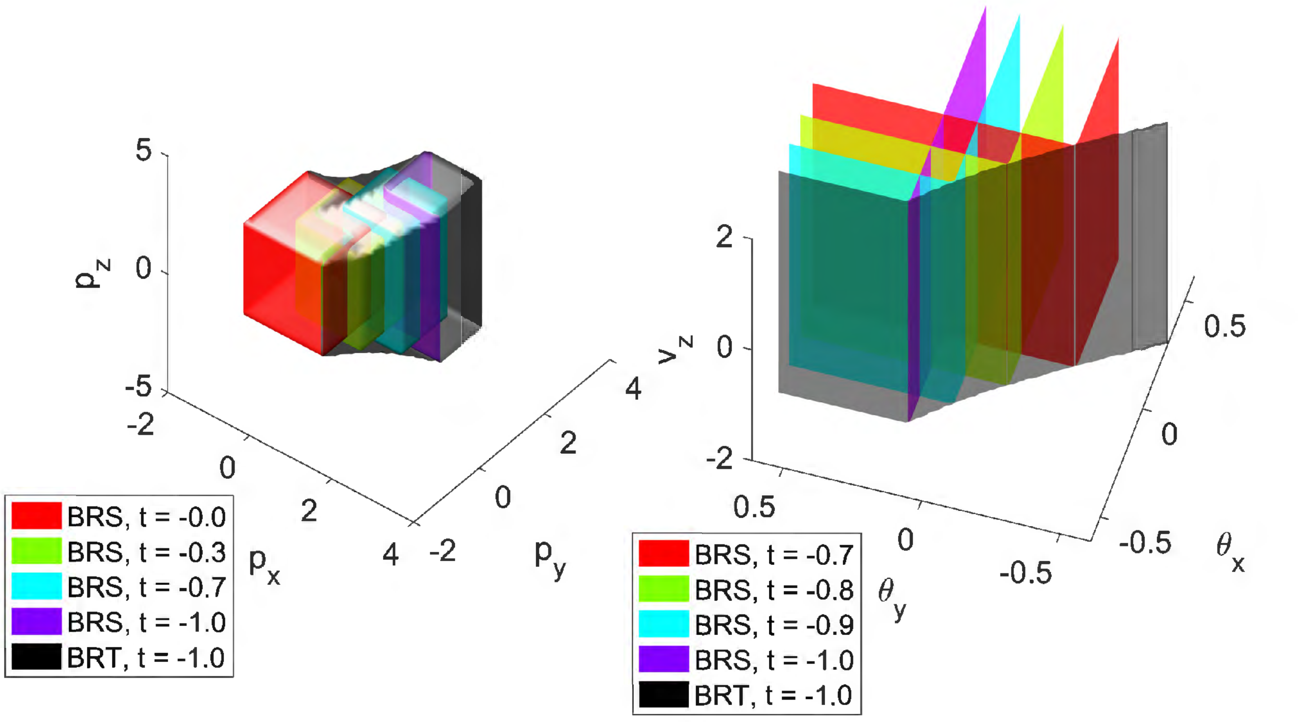


Figure 3.6: 3D slices of the 10D BRSs over time (colored surfaces) and the BRT (black surface) for the Near-Hover Quadrotor. The slices are taken at the 7D points at $(v_x, v_y, v_z) = (-1.5, -1.8, 1.2), \theta_x = \theta_y = \omega_x = \omega_y = 0$ (Left) and $(p_x, p_y, p_z) = (-1.5, 0, 1), (v_x, v_y) = (1.2, -0.6), \omega_x = \omega_y = -0.5$ (Right).

The right subplot of Fig. 3.6 shows the BRS and the BRT in $(\theta_x, \theta_y, v_z)$ space, sliced at $(p_x, p_y, p_z) = (-1.5, 0, 1), (v_x, v_y) = (1.2, -0.6), \omega_x = \omega_y = -0.5$. To the best of our knowledge, such a slice of the exact BRS and the BRT is not possible to obtain using previous methods, since a high-dimensional system model like (3.25) is needed for analyzing the angular behavior of the system.

3.2.2.3 Shared Control, Shared Disturbance

The system dynamics in this case can be written as:

$$\begin{aligned}\dot{x}_1 &= f_1(x_1, x_c, u, d)\\ \dot{x}_2 &= f_2(x_2, x_c, u, d)\\ \dot{x}_c &= f_c(x_c, u, d)\end{aligned} \tag{3.29}$$

When the SCS share control and disturbance components, the results from Section 3.2.2.1 carry over with some modifications. Theorems 1 and 2 need to be changed slightly, and the reconstructed BRS is now an approximation conservative in the right direction.

Proposition 3 *Suppose that the full system dynamics $\dot{x} = f(x, u, d)$ can be decomposed into the form of (3.29). Moreover, suppose that the target set $\mathcal{L}$ can be written as a union of the subsystem target sets then the maximal BRS is a superset of the union of subsystems' maximal BRSs, i.e.,*

$$\mathcal{L} = \mathcal{P}^{-1}(\mathcal{L}_1) \cup \mathcal{P}^{-1}(\mathcal{L}_2) \quad \Rightarrow \quad \mathcal{R}(t) \supseteq \mathcal{P}^{-1}(\mathcal{R}_1(t)) \cup \mathcal{P}^{-1}(\mathcal{R}_2(t)) \tag{3.30}$$

Proof: *The proof follows similarly to that of Theorem 1, except that we can no longer distribute disturbance in (3.15). This make the reconstructed BRS a conservative approximation of the true BRS in the right direction. By conservative in the right direction, we mean that a state x in the reconstructed BRS is guaranteed to be able to reach the target.* ■

Proposition 4 *Suppose that the full system dynamics $\dot{x} = f(x, u, d)$ can be decomposed into the form of (3.29). Moreover, suppose that the target set $\mathcal{L}$ can be written as an intersection of the subsystem target sets then the minimal BRS is a subset of the intersection of subsystems' minimal BRSs, i.e.,*

$$\mathcal{L} = \mathcal{P}^{-1}(\mathcal{L}_1) \cap \mathcal{P}^{-1}(\mathcal{L}_2) \quad \Rightarrow \quad \mathcal{A}(t) \subseteq \mathcal{P}^{-1}(\mathcal{A}_1(t)) \cap \mathcal{P}^{-1}(\mathcal{A}_2(t)) \tag{3.31}$$

Proof: *The proof of Proposition 4 follows that of Theorem 1, except that it involves complements of sets instead. Again, the reconstructed BRS is a conservative approximation of the true BRS in the right direction, meaning that a state x outside of the reconstructed BRS is guaranteed to be able to avoid the target.* ■

Table 3.4 summarizes the results from this section.

Table 3.4: The BRS reconstruction from self-contained subsystems when they have shared control and shared disturbance (Section 3.2.2.3)

Shared Controls	**Yes**	
Shared Disturbance	**Yes**	
Target	**Intersection**	**Union**
Recover Maximal BRS?	No	Yes, conservative
Recover Minimal BRS?	Yes, conservative	No
Result	Proposition 4	Proposition 3

3.2.2.4 No Shared Control, Shared Disturbance

The system dynamics in this case can be written as:

$$\begin{aligned}\dot{x}_1 &= f_1(x_1, x_c, u_1, d)\\ \dot{x}_2 &= f_2(x_2, x_c, u_2, d)\\ \dot{x}_c &= f_c(x_c, d)\end{aligned} \tag{3.32}$$

When the SCS do not share control but share disturbance components, the results from Section 3.2.2.3 still hold since the system dynamics structure is a special case of that in Section 3.2.2.3. In addition, results from Section 3.2.2.2 hold with some modifications. Propositions 1 and 2 need to be modified, and again the reconstructed BRS is now an approximation conservative in the right direction.

Proposition 5 *Suppose that the full system dynamics $\dot{x} = f(x, u, d)$ can be decomposed into the form of* (3.32)*. Moreover, suppose that the target set $\mathcal{L}$ can be written as an intersection of the subsystem target sets then the maximal BRS is a superset of the intersection of subsystems' maximal BRSs, i.e.,*

$$\mathcal{L} = \mathcal{P}^{-1}(\mathcal{L}_1) \cap \mathcal{P}^{-1}(\mathcal{L}_2) \quad \Rightarrow \quad \mathcal{R}(t) \supseteq \mathcal{P}^{-1}(\mathcal{R}_1(t)) \cap \mathcal{P}^{-1}(\mathcal{R}_2(t)) \tag{3.33}$$

Proposition 6 *Suppose that the full system dynamics $\dot{x} = f(x, u, d)$ can be decomposed into the form of* (3.29)*. Moreover, suppose that the target set $\mathcal{L}$ can be written as a union of the subsystem target sets then the minimal BRS is a subset of the union of subsystems' minimal BRSs, i.e.,*

$$\mathcal{L} = \mathcal{P}^{-1}(\mathcal{L}_1) \cup \mathcal{P}^{-1}(\mathcal{L}_2) \quad \Rightarrow \quad \mathcal{A}(t) \subseteq \mathcal{P}^{-1}(\mathcal{A}_1(t)) \cup \mathcal{P}^{-1}(\mathcal{A}_2(t)) \tag{3.34}$$

The proofs of Propositions 5 and 6 follow that of Propositions 1 and 2 respectively, except noting that we can no longer distribute disturbances in trajectories. Table 3.5 summarizes the results from this section.

Table 3.5: The BRS reconstruction from self-contained subsystems when they have no shared control but shared disturbance components (Section 3.2.2.4)

Shared Controls	**No**	
Shared Disturbance	**Yes**	
Target	**Intersection**	**Union**
Recover Maximal BRS?	Yes, conservative	Yes, conservative
Recover Minimal BRS?	Yes, conservative	Yes, conservative
Result	Proposition 4, 5	Proposition 3, 6

3.2.2.5 Reconstruction of BRT

We now turn our attention to computing minimal and maximal BRTs. Intuitively, it may seem like the results related to BRSs outlined in previous sections trivially carry over to BRTs, and the relationship between BRSs and BRTs are relatively simple; however, this is only partially true. The results related to BRSs presented so far in this paper only easily carry over for BRTs if $\mathcal{L} = \mathcal{P}^{-1}(\mathcal{L}_1) \cup \mathcal{P}^{-1}(\mathcal{L}_2)$ (see [75] for details).

However, if $\mathcal{L} = \mathcal{P}^{-1}(\mathcal{L}_1) \cap \mathcal{P}^{-1}(\mathcal{L}_2)$, the BRT cannot be directly reconstructed from lower-dimensional BRTs because when computing with BRTs, we lose information about the exact time that a trajectory enters a set. For the intersection scenario, *both* subsystem trajectories must be in the corresponding subsystem target sets *at the same time*. The time intervals during which each subsystem trajectory is in $\mathcal{L}_i$ may not overlap for the different subsystems, making it harder to reconstruct the high-dimensional BRT from that of the subsystems'. Instead, we provide a general method of obtaining the BRT that can *also* be used for the intersection scenarios: we first compute the BRSs, and then take their union to obtain the BRT. This allows us to reconstruct the BRT from subsystem decomposition results developed for BRS in Sections 3.2.2.1 through 3.2.2.4.

In particular, we show that when the system dynamics do not have disturbance components, $\bar{\mathcal{R}}(t) = \bigcup_{s\in[t,T]} \mathcal{R}(s)$, and $\bar{\mathcal{A}}(t) = \bigcup_{s\in[t,T]} \mathcal{A}(s)$ when $\mathcal{A}(s) \neq \emptyset\ \forall s \in [t,T]$. On the other hand, a conservative approximation is obtained when the disturbance is present in the system. These results related to the indirect reconstruction of BRTs are given in Proposition 7 and Theorem 3

Proposition 7 *Consider the full system dynamics $\dot{x} = f(x,u,d)$. Then, the union of maximal BRSs of the system is a superset of the BRT, i.e.,*

$$\bigcup_{s\in[t,T]} \mathcal{R}(s) \subseteq \bar{\mathcal{R}}(t) \tag{3.35}$$

In addition, if there is no disturbance in dynamics, i.e., $d = 0$, then the relation in (3.35) *holds with equality:*

$$\bigcup_{s\in[t,T]} \mathcal{R}(s) = \bar{\mathcal{R}}(t) \tag{3.36}$$

Proof: *We will first show the result in* (3.36)*, i.e., when there is no disturbance in dynamics. As per the definition of maximal BRS:*

$$\mathcal{R}(t) = \{x : \exists u(\cdot) \in \mathbb{U}_t^T, \xi(T; x, t, u(\cdot)) \in \mathcal{L}\}$$

If some state x is in the union $\bigcup_{s\in[t,T]} \mathcal{R}(s)$, then there is some $s \in [t,T]$ such that

$x \in \mathcal{R}(s)$. Therefore, the union can be written as

$$\bigcup_{s\in[t,T]} \mathcal{R}(s) = \{x : \exists s \in [t,T], \exists u(\cdot), \xi(T; x, s, u(\cdot)) \in \mathcal{L}\} \tag{3.37}$$

Suppose $x \in \bigcup_{s\in[t,T]} \mathcal{R}(s)$, then equivalently

$$\exists s \in [t,T], \exists u(\cdot) \in \mathbb{U}, \xi(T; x, s, u(\cdot)) \in \mathcal{L} \tag{3.38}$$

Using the time-invariance of the system $\dot{x} = f(x,u,d)$, we can shift the trajectory time arguments by $t - s$ to get

$$\exists s \in [t,T], \exists u(\cdot) \in \mathbb{U}, \xi(T + t - s; x, t, u(\cdot)) \in \mathcal{L} \tag{3.39}$$

Since $s \in [t,T] \Leftrightarrow T + t - s \in [t,T]$, we can equivalently write

$$\exists s \in [t,T], \exists u(\cdot) \in \mathbb{U}, \xi(s; x, t, u(\cdot)) \in \mathcal{L} \tag{3.40}$$

We can swap the expressions $\exists s \in [t,T]$ and $\exists u(\cdot) \in \mathbb{U}$ without changing meaning since both quantifiers are the same:

$$\exists u(\cdot) \in \mathbb{U}, \exists s \in [t,T], \xi(s; x, t, u(\cdot)) \in \mathcal{L} \tag{3.41}$$

which is equivalent to $x \in \bar{\mathcal{R}}(t)$ by the definition of a maximal BRT.

When the disturbance is present in the dynamics, the union of the BRSs becomes an under-approximation of the BRT in general. To show this, all arguments in the proof for zero disturbance remain the same, except (3.37) no longer implies (3.41). Instead, the implication is unidirectional:

$$\begin{aligned} &\exists s \in [t,T], \forall d(\cdot) \in \Gamma_t^T, \exists u(\cdot), \xi(T; x, s, u(\cdot), d(\cdot)) \in \mathcal{L} \\ &\Rightarrow \forall d(\cdot) \in \Gamma_t^T, \exists u(\cdot), \exists s \in [t,T], \xi(s; x, t, u(\cdot), d(\cdot)) \in \mathcal{L} \end{aligned} \tag{3.42}$$

This is due to the switching of the order of the expressions "$\exists s \in [t,T]$" and "$d(\cdot) \in \Gamma_t^T$". Therefore, the union of the BRSs becomes an under-approximation of the BRT, a conservatism in the right direction: a state in the under-approximated BRT is still guaranteed to be able to reach the target. ■

Theorem 3 *Consider the full system dynamics $\dot{x} = f(x,u,d)$. Then, the union of minimal BRSs of the system is a subset of the BRT, i.e.,*

$$\bigcup_{s\in[t,T]} \mathcal{A}(s) \subseteq \bar{\mathcal{A}}(t) \tag{3.43}$$

In addition, if $\forall s \in [t,T], \mathcal{A}(s) \neq \emptyset$, *then*

$$\bigcup_{s\in[t,T]} \mathcal{A}(s) = \bar{\mathcal{A}}(t). \tag{3.44}$$

Proof: *We first establish* $\bigcup_{s\in[t,T]} \mathcal{A}(s) \subseteq \bar{\mathcal{A}}(t)$. *As per the definition of a minimal BRS, we have:*

$$\mathcal{A}(t) = \{x : \forall u(\cdot), \exists d(\cdot) \in \Gamma_t^T, \xi(T; x, t, u(\cdot), d(\cdot)) \in \mathcal{L}\}$$

If some state x *is in the union* $\bigcup_{s\in[t,T]} \mathcal{A}(s)$, *then* $\exists s \in [t,T]$ *such that* $x \in \mathcal{A}(s)$. *Thus, the union can be written as*

$$\bigcup_{s\in[t,T]} \mathcal{A}(s) = \{x : \exists s \in [t,T], \forall u(\cdot), \exists d(\cdot) \in \Gamma_t^T, \xi(T; x, s, u(\cdot), d(\cdot)) \in \mathcal{L}\} \tag{3.45}$$

Suppose $x \in \bigcup_{s\in[t,T]} \mathcal{A}(s)$, *then*

$$\exists s \in [t,T], \forall u(\cdot), \exists d(\cdot) \in \Gamma_t^T, \xi(T; x, s, u(\cdot), d(\cdot)) \in \mathcal{L} \tag{3.46}$$

Using the time-invariance of the system $\dot{x} = f(x,u,d)$, *we can shift the trajectory time arguments by* $t - s$ *to get*

$$\exists s \in [t,T], \forall u(\cdot), \exists d(\cdot) \in \Gamma_t^T, \xi(T + t - s; x, t, u(\cdot), d(\cdot)) \in \mathcal{L} \tag{3.47}$$

Since $s \in [t,T] \Leftrightarrow T + t - s \in [t,T]$, *we can equivalently write*

$$\exists s \in [t,T], \forall u(\cdot), \exists d(\cdot) \in \Gamma_t^T, \xi(s; x, t, u(\cdot), d(\cdot)) \in \mathcal{L} \tag{3.48}$$

Let such an $s \in [t,T]$ *be denoted* $\bar{s}$, *then*

$$\begin{aligned} &\forall u(\cdot), \exists d(\cdot) \in \Gamma_t^T, \xi(\bar{s}; x, t, u(\cdot), d(\cdot)) \in \mathcal{L} \\ &\Rightarrow \forall u(\cdot), \exists d(\cdot) \in \Gamma_t^T, \exists s \in [t,T], \xi(s; x, t, u(\cdot), d(\cdot)) \in \mathcal{L} \end{aligned} \tag{3.49}$$

By the definition of a minimal BRT, we have $x \in \bar{\mathcal{A}}(t)$.

Next, given $\forall s \in [t,T], \mathcal{A}(s) \neq \emptyset$, *we show* $\bigcup_{s\in[t,T]} \mathcal{A}(s) \supseteq \bar{\mathcal{A}}(t)$. *Equivalently, we show*

$$x \notin \bigcup_{s\in[t,T]} \mathcal{A}(s) \Rightarrow x \notin \bar{\mathcal{A}}(t) \tag{3.50}$$

Given $x \notin \bigcup_{s\in[t,T]} \mathcal{A}(s)$, *for all* $d(\cdot) \in \Gamma_t^T$, *there exists some control* $\bar{u}(\cdot)$ *such that* $\forall s \in [t,T], \xi(T; x, s, \bar{u}(\cdot)) \notin \mathcal{L}$. *Since otherwise, for all* $u(\cdot)$ *we would have a* $\hat{s} \in [t,T]$ *and* $\bar{d}(\cdot) \in \Gamma_t^T$ *such that*

$$\xi(T; x, \hat{s}, u(\cdot), \bar{d}(\cdot)) \in \mathcal{L} \Rightarrow x \in \mathcal{A}(\hat{s}) \tag{3.51}$$

which contradicts $x \notin \bigcup_{s\in[t,T]} \mathcal{A}(s)$. Using time-invariance of the system dynamics, we have

$$\forall d(\cdot) \in \Gamma_t^T, \exists \bar{u}(\cdot), \forall s \in [t,T], \xi(T+t-s; x, t, \bar{u}(\cdot), d(\cdot)) \notin \mathcal{L},$$

which is equivalent to

$$\forall d(\cdot) \in \Gamma_t^T, \exists \bar{u}(\cdot), \forall s \in [t,T], \xi(s; x, t, \bar{u}(\cdot)) \notin \mathcal{L},$$

which in turn implies that $x \notin \bar{\mathcal{A}}(t)$. ■

Remark 5 *We note that Proposition 7 and the first part of Theorem 3 are known [219], but we present them here in greater detail for clarity and completeness. The second part of Theorem 3 is the main new result related to obtaining the BRT from BRSs.*

Equipped with Proposition 7 and Theorem 3, we can now reconstruct the BRT of a high-dimensional system using the BRSs of subsystems. The conservativeness or exactness of the reconstructed BRT now depends on (a) whether the high-dimensional BRS itself can be reconstructed exactly or not, and (b) whether the BRT can be reconstructed exactly from the BRS or not. Table 3.6 summarizes these results.

Table 3.6: The BRT reconstruction from self-contained subsystems.

Shared Controls	**Yes**		**No**		**Yes**		**No**	
Shared Disturbance	**No**		**No**		**Yes**		**Yes**	
Target	**Intersection**	**Union**	**Intersection**	**Union**	**Intersection**	**Union**	**Intersection**	**Union**
Recover Maximal BRT?[4]	No	Yes, cons	Yes, cons	Yes, cons	No	Yes, cons	Yes, cons	Yes, cons
Result: Maximal BRT	N/A	Thm 1, Prop 7	Prop 1, 7	Thm 1, Prop 7	N/A	Prop 3, 7	Prop 5, 7	Prop 3, 7
Recover Minimal BRT?[5]	Yes, exact	No	Yes, exact	Yes, exact	Yes, cons	No	Yes, cons	Yes, cons
Result: Minimal BRT	Thm 2, 3	N/A	Thm 2, 3	Prop 2, Thm 3	Prop 4, Thm 3	N/A	Prop 4, Thm 3	Prop 6, Thm 3

Numerical Example: Dubins Car With Disturbance. Under disturbances, the Dubins Car dynamics are given by

$$\begin{bmatrix} \dot{p}_x \\ \dot{p}_y \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos\theta + d_x \\ v\sin\theta + d_y \\ \omega + d_\theta \end{bmatrix} \qquad (3.52)$$

$$\omega \in \mathcal{U}, \quad (d_x, d_y, d_\theta) \in \mathcal{D}$$

with state $x = (p_x, p_y, \theta)$, control $u = \omega$, and disturbances $d = (d_x, d_y, d_\theta)$. The state partitions are $x_1 = p_x, x_2 = p_y, x_c = \theta$. The subsystems states z_i, controls, and disturbances

[4]If additionally the disturbance is zero in the cases when the subsystems do not have a shared disturbance components, the conservative results become exact.

[5]The solution here can be found only if the minimum BRSs are non-empty for the entire time period.

are

$$\begin{aligned}
\dot{z}_1 &= \begin{bmatrix} \dot{x}_1 \\ \dot{x}_c \end{bmatrix} = \begin{bmatrix} \dot{p}_x \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos\theta + d_x \\ \omega + d_\theta \end{bmatrix} \\
\dot{z}_2 &= \begin{bmatrix} \dot{x}_2 \\ \dot{x}_c \end{bmatrix} = \begin{bmatrix} \dot{p}_y \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\sin\theta + d_y \\ \omega + d_\theta \end{bmatrix} \\
u_c &= \omega = u \\
d_1 &= (d_x, d_\theta), d_2 = (d_y, d_\theta)
\end{aligned} \tag{3.53}$$

where the overlapping state is $\theta = x_c$. We assume that each component of disturbance is bounded in some interval centered at zero: $|d_x| \leq \bar{d}_x, |d_y| \leq \bar{d}_y, |d_\theta| \leq \bar{d}_\theta$. The subsystem disturbances d_1 and d_2 have the shared component d_θ.

We compute the BRT of Dubins car dynamics assuming $T = 0$. Fig. 3.7 compares the BRT $\bar{\mathcal{A}}(t), t = -0.5$ computed directly from the target set in (3.19), and using our decomposition technique from the subsystem target sets in (3.20). For this computation, we chose $\bar{d}_x, \bar{d}_y = 1, \bar{d}_\theta = 5$.

Since there is a shared component in the disturbances, the BRT computed using our decomposition technique becomes an over-approximation of the true BRT. One can see the over-approximation by noting that the black set is not flush against the red set, as marked by the arrows in Fig. 3.7.

Fig. 3.8 shows the same computation with $\bar{d}_\theta = 0$, so that subsystem disturbances effectively have no shared components. In this case, one can see that the BRTs computed directly in 3D and via decomposition in 2D are the same.

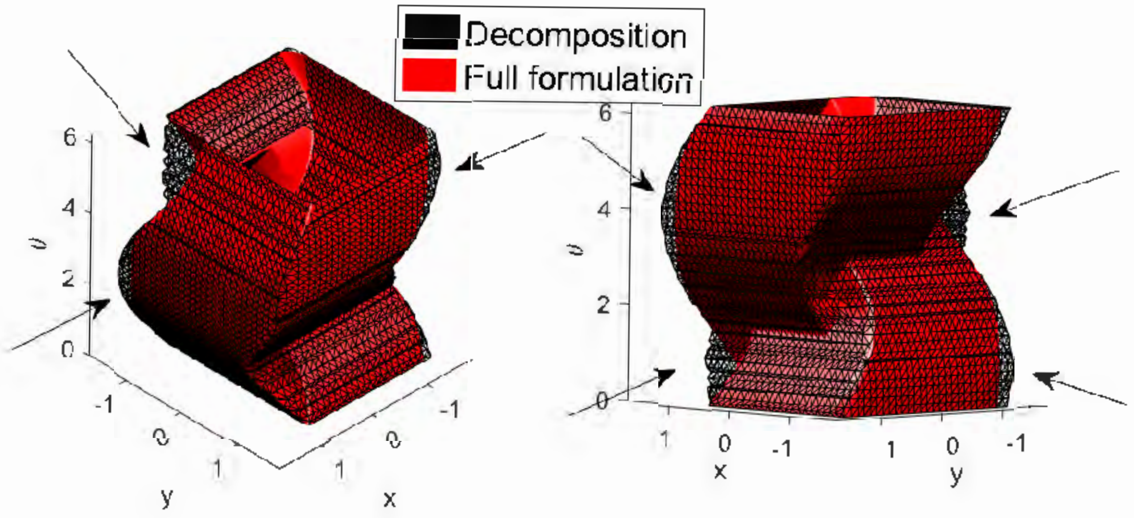


Figure 3.7: Minimal BRTs computed directly in 3D and via decomposition in 2D for the Dubins Car under disturbances with shared components. The reconstructed BRT is an over-approximation of the true BRT. The over-approximated regions of the reconstruction are indicated by the arrows.

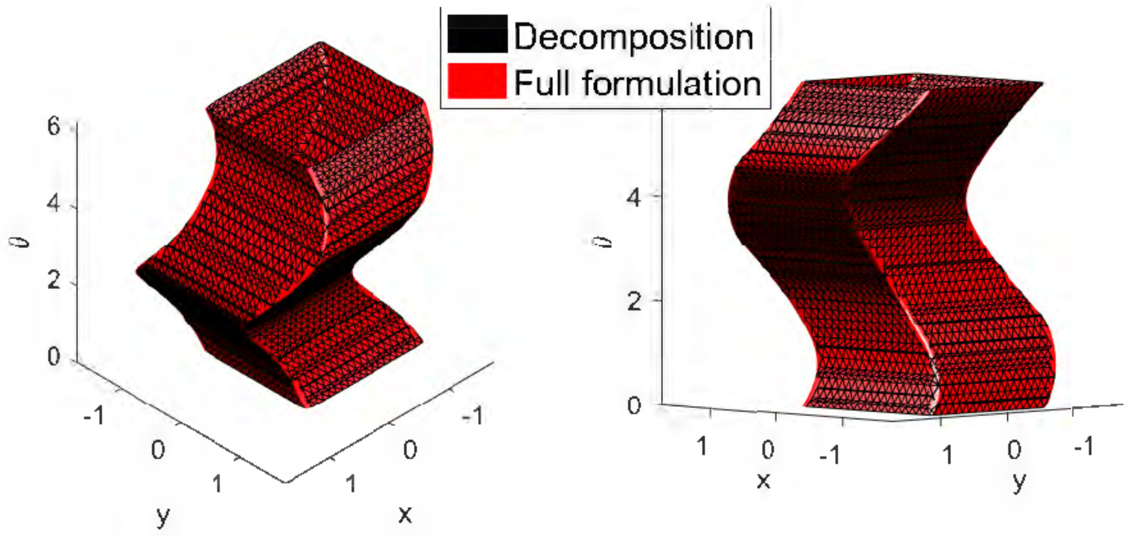


Figure 3.8: Minimal BRTs computed directly in 3D and via decomposition in 2D for the Dubins Car under disturbances *without* shared components. In this case, the BRT computed using decomposition matches the true BRT.

Numerical Example: The 10D Near-Hover Quadrotor. We now show numerical results for the 6D Acrobatic Quadrotor, a system whose exact BRSs and BRTs were intractable to compute with previous methods to the best of our knowledge. In [134], a 6D quadrotor model used to perform backflips was simplified into a series of smaller models linked together in a hybrid system. The Quadrotor has state $x = (p_x, v_x, p_y, v_y, \phi, \omega)$, and dynamics

$$\begin{bmatrix} \dot{p}_x \\ \dot{v}_x \\ \dot{p}_y \\ \dot{v}_y \\ \dot{\phi} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} v_x \\ -\frac{1}{m}C_D^v v_x - \frac{T_1}{m}\sin\phi - \frac{T_2}{m}\sin\phi \\ v_y \\ -\frac{1}{m}(mg + C_D^v v_y) + \frac{T_1}{m}\cos\phi + \frac{T_2}{m}\cos\phi \\ \omega \\ -\frac{1}{I_{yy}}C_D^\phi \omega - \frac{l}{I_{yy}}T_1 + \frac{l}{I_{yy}}T_2 \end{bmatrix} \tag{3.54}$$

where x, y, and ϕ represent the quadrotor's horizontal, vertical, and rotational positions, respectively. Their derivatives represent the velocity with respect to each state. The control inputs T_1 and T_2 represent the thrust exerted on either end of the quadrotor, and the constant system parameters are m for mass, C_D^v for translational drag, C_D^ϕ for rotational drag, g for acceleration due to gravity, l for the length from the quadrotor's center to an edge, and I_{yy} for moment of inertia. There are no disturbance in the dynamics.

We decompose the system into the following subsystems:

$$z_1 = (p_x, v_x, \phi, \omega), \qquad z_2 = (p_y, v_y, \phi, \omega) \tag{3.55}$$

For this example we will compute $\mathcal{A}(t)$ and $\bar{\mathcal{A}}(t)$, which describe the set of initial conditions from which the system may enter the target set despite the best possible control to avoid the

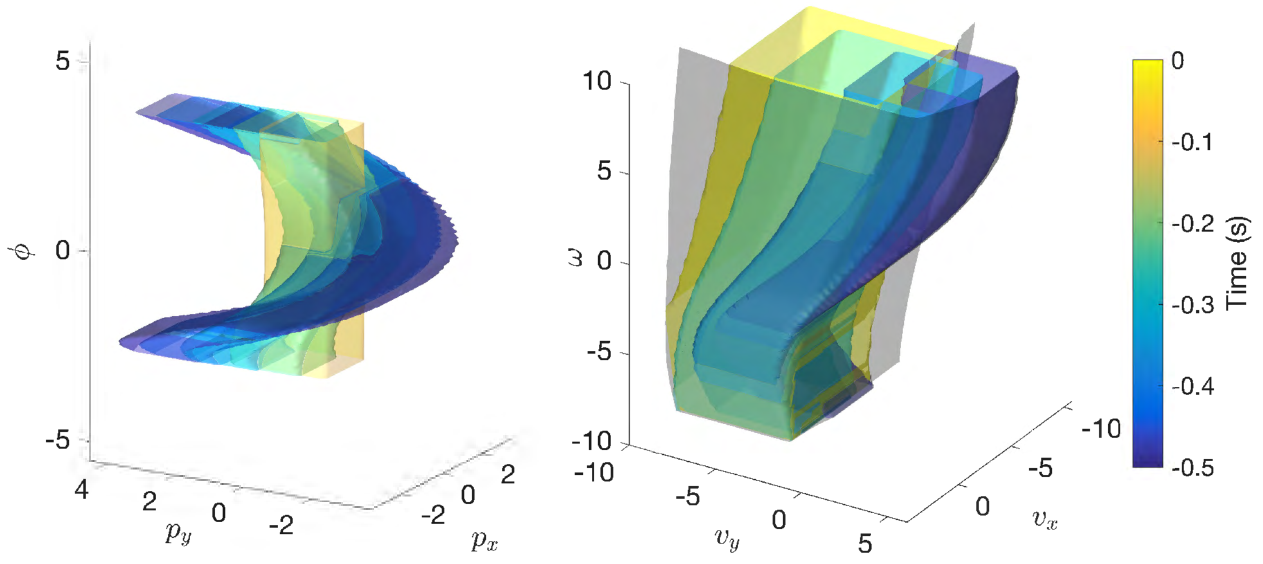


Figure 3.9: Left: 3D positional slices of the reconstructed 6D BRSs at $v_x = v_y = 1$, $\omega = 0$ at different points in time. The BRT cannot be seen in this image because it encompasses the entire union of BRSs. Right: 3D velocity slices of the reconstructed 6D BRSs at $x, y = 1.5$, $\phi = 1.5$ at different points in time. The BRT can be seen as the transparent gray surface that encompasses the sets.

target. We define the target set as a square of length 2 centered at $(p_x, p_y) = (0, 0)$ described by $\mathcal{L} = \{(p_x, v_x, p_y, v_y, \phi, \omega) : |p_x|, |p_y| \leq 1\}$. This can be interpreted as a positional box centered at the origin that must be avoided for all angles and velocities. From the target set, we define $l(x)$ such that $l(x) \leq 0 \Leftrightarrow x \in \mathcal{L}$. This target set is then decomposed as follows:

$$\mathcal{L}_1 = \{(p_x, v_x, \phi, \omega) : |p_x| \leq 1\}$$
$$\mathcal{L}_2 = \{(p_y, v_y, \phi, \omega) : |p_y| \leq 1\}$$

The BRS of each 4D subsystem is computed and then recombined into the 6D BRS. To visually depict the 6D BRS, 3D slices of the BRS along the positional and velocity axes were computed. The left image in Fig. 3.9 shows a 3D slice in (p_x, p_y, ϕ) space at $v_x = v_y = 1, \omega = 0$. The yellow set represents the target set $\mathcal{L}$, with the BRS in other colors. Shown on the right in Fig. 3.9 are 3D slices in (v_x, v_y, ω) space at $p_x, p_y = 1.5, \phi = 1.5$ through different points in time. The sets grow darker as time propagates backward. The union of the BRSs is the BRT, shown as the gray surface.

3.3 Run-Time Reachability in Dynamic Environments: Warm Start and Local Updates

Decomposition methods discussed in Section 3.2 allows us to significantly alleviate the computational complexity of Hamilton-Jacobi reachability analysis. The HJI reachability analysis is based on assumptions about system dynamics, external disturbances, and the surrounding environment. However in reality the dynamics, the disturbance bounds, or the environment may differ from the assumptions, or they may simply change online as the system is operating in its environment. In these situations the safety analysis must be updated. Restarting the safety analysis, even with the decomposition methods, is typically prohibitive for such run-time updates. In this section, we propose a method of "warm-start" reachability, which uses a user-defined initialization (typically the previously computed solution). By starting with a value function that is closer to the solution than the standard initialization, the analysis may take fewer iterations. We additionally prove that warm-start reachability will in general result in *guaranteed conservative* safety analyses and controllers (i.e. the analysis over-approximates the set of states that are unsafe to enter). Moreover, if the initialization is over-optimistic and therefore dangerous (i.e. the initialization underestimates the set of states that are unsafe to enter), we prove that warm-starting is *guaranteed to converge exactly* to the true solution (here we use "exact" to mean numerically convergent [219]). In addition to these proofs, we provide several common problem classes for which we can prove this exact convergence, and illustrate these ideas on some numerical examples.

Equipped with our results from warm-start reachability, we propose a novel, real-time algorithm that only locally updates the warm-start value function in the parts of the environment where new information has been obtained, rather than everywhere in the state space. This further reduces the computational complexity of the warm-start reachability, which as we will show later, can enable updating the reachable sets during runtime on real autonomous systems (Chapter 9).

3.3.1 Warm-Start Reachability

When there are minor changes to the problem formulation, such as changes to the model parameters, external disturbances, or target sets, computing $V(t,x)$ requires recomputing the entire value function starting with the target function ($V(T,x) = l(x)$). Instead, we initialize with a previous computed (converged) value function.

The theory in this section applies to BRTs with infinite-time horizons. Typically, for the safety analysis of robotic systems, we are more interested in the avoid case (or minimal BRT), where the system seeks to avoid an unsafe set of states forever, and will therefore be the focus of this section.

We define this warm-starting function as $k(x)$, with subzero level set $\mathcal{K} = \{x : k(x) \leq 0\}$[6]. To develop the theory, we revisit the cost function (2.47) for the differential game

[6]Note that we are overloading the notation $\mathcal{K}$ in this section. It no longer represents the viability kernel,

corresponding to computing the BRT:

$$J_t(x, u(\cdot), d(\cdot)) = \inf_{\tau \in [t,T]} l(x(\tau)),$$

To make the dependence on l explicit, we add the time as an argument to the cost function and use l as subscript. We also make dependence on the trajectory explicit, which will later be convenient in stating our key results. In particular, we rewrite the above cost function as

$$J_l(x, t, u(\cdot), d(\cdot)) = \min\Big\{ \inf_{\tau \in [t,0)} l(\xi_{x,t}^{u,d}(\tau)), l(\xi_{x,t}^{u,d}(0))\Big\}.$$

Recall that $\xi_{x,t}^{u,d}(\tau)$ represent the state achieved at time τ by starting at initial state x and initial time t, and applying input functions $u(\cdot)$ and $d(\cdot)$ over $[t, \tau]$. We similarly define the value function corresponding to the cost function J_l as $V_l(t, x)$. From the definition of the value function, we have:

$$\begin{aligned} V_l(t, x) &= \inf_{d(\cdot)\in\Gamma_t^T} \sup_{u(\cdot)} J_l(x, t, u(\cdot), d(\cdot)) \\ &= \inf_{d(\cdot)\in\Gamma_t^T} \sup_{u(\cdot)} \min\Big\{ \inf_{\tau \in [t,T)} l(\xi_{x,t}^{u,d}(\tau)), V_l(T, \xi_{x,t}^{u,d}(T))\Big\}, \\ &= \inf_{d(\cdot)\in\Gamma_t^T} \sup_{u(\cdot)} \min\Big\{ \inf_{\tau \in [t,T)} l(\xi_{x,t}^{u,d}(\tau)), l(\xi_{x,t}^{u,d}(T))\Big\}, \end{aligned} \tag{3.56}$$

$V_l(t, x)$ is the solution to the following HJI VI,

$$\begin{aligned} &\min\{D_t V_l(t, x) + H_l(t, x, , V_l(t, x)),\ l(x) - V_l(t, x)\} = 0\ \forall x, t, \\ &\quad H_l(t, x, , V_l(t, x)) = \max_u \min_d \langle D_x V_l(t, x), f(x, u, d)\rangle, \\ &\quad V_l(T, x) = l(x). \end{aligned} \tag{3.57}$$

The converged value function is defined as $V_l^*(x) = \lim_{t\to-\infty} V_l(t, x)$. When we warm-start the computation of value function using k, the cost function is given by:

$$J_k(x, t, u(\cdot), d(\cdot)) = \min\Big\{ \inf_{\tau \in [t,T)} l(\xi_{x,t}^{u,d}(\tau)), k(\xi_{x,t}^{u,d}(T))\Big\}. \tag{3.58}$$

V_k can be defined as in (3.56) with $J = J_k$, i.e

$$\begin{aligned} V_k(x, t) &= \inf_{d(\cdot)\in\Gamma_t^T} \sup_{u(\cdot)} J_k(x, t, u(\cdot), d(\cdot)) \\ &= \inf_{d(\cdot)\in\Gamma_t^T} \sup_{u(\cdot)} \min\Big\{ \inf_{\tau \in [t,T)} l(\xi_{x,t}^{u,d}(\tau)), k(\xi_{x,t}^{u,d}(T))\Big\}. \end{aligned} \tag{3.59}$$

V_k is the solution to the HJI VI defined similarly as in (3.57) with $V_k(T, x) = k(x)$. The converged value function is defined as $V_k^*(x) = \lim_{t\to-\infty} V_k(t, x)$.

but rather the warm-starting target set.

In this section we prove that the converged value function $V_k^*(x)$ that is initialized as above $V_k(T,x) = k(x)$ will always be more negative than the value function $V_l^*(x)$ achieved by standard reachability (i.e. initialized as $V_l(T,x) = l(x)$). For the case of avoiding an unsafe set, this means that the relationship between the functions' BRTs (i.e. subzero level sets) is $\mathcal{V}_k^* \supseteq \mathcal{V}_l^*$. In other words, $\mathcal{V}_k^*$ is a conservative over-approximation of $\mathcal{V}_l^*$. We will prove that for certain conditions (i.e. when $k(x) \geq V_l^*(x)$), we can guarantee the resulting value function and BRT will be exact.

3.3.1.1 Conservative Warm-Start Reachability

If $[V_k(T,x) = k(x)] \leq V_l^*(x)$, a contraction mechanism is required to raise $V_k^*(x)$ towards the true solution $V_l^*(x)$. Recall the HJI VI from (3.57). Contraction may happen naturally, when the left hand side of the minimization (the HJI PDE) "pulls the system up" due to the Hamiltonian. However, there are no guarantees that this contraction will happen, and the new value function may get stuck in a local solution, $V_k^*(x) \leq V_l^*(x)$. This will result in a conservative BRT.

Theorem 4 *For all initializations of $V_k(T,x) = k(x)$, the resultant value function will be a conservative approximation of the true value function,i.e.,*

$$V_k(t,x) \leq V_l(t,x), \quad \forall x, t < T$$

Proof: *We prove that $V_k(t,x) \leq V_l(t,x)$ for two cases, (a) $k(x) < l(x)$ and (b) $k(x) \geq l(x)$.*

$\underline{\text{(a) } k(x) < l(x)}$ For $\forall x, t < T$, let $V_l(t,x)$ be defined as (3.56) *and $V_k(t,x)$ be defined as* (3.59). *At $t = T$, we have $\Big[V_k(T,x) = k(x)\Big] < \Big[l(x) = V_l(T,x)\Big] \Rightarrow V_k(T,x) < V_l(T,x)$. For any $t < T$:*

$$\begin{aligned} V_k(x,t) &= \inf_{d(\cdot)\in\Gamma_t^T} \sup_{u(\cdot)} \min\Big\{ \inf_{\tau\in[t,T)} l(\xi_{x,t}^{u,d}(\tau)), k(\xi_{x,t}^{u,d}(T))\Big\}, \\ &\leq \inf_{d(\cdot)\in\Gamma_t^T} \sup_{u(\cdot)} \min\Big\{ \inf_{\tau\in[t,T)} l(\xi_{x,t}^{u,d}(\tau)), l(\xi_{x,t}^{u,d}(T))\Big\}, \\ &= V_l(t,x). \end{aligned} \tag{3.60}$$

The second inequality follows from the fact that $k(x) < l(x)$ $\forall x \in \mathbb{R}^{n_x}$. Hence, $\forall t, x$, we have $V_k(t,x) \leq V_l(t,x)$. Finally, $t \to -\infty$, we have $V_k^*(x) \leq V_l^*(x)$.

$\underline{\text{(b) } k(x) \geq l(x)}$ When $t = T$, $\Big[V_k(T,x) = k(x)\Big] \geq \Big[l(x) = V_l(T,x)\Big]$. For a time instance $t = T^-$,

$$\begin{aligned} V_k(t,x) &= \inf_{d(\cdot)\in\Gamma_t^T} \sup_{u(\cdot)} \min\left\{ \inf_{\tau\in[t,T)} l(\xi_{x,t}^{u,d}(\tau)), k(\xi_{x,t}^{u,d}(T))\right\} \\ &= \min\{l(\xi_{x,t}^{u,d}(T^-)), k(\xi_{x,t}^{u,d}(T)\} \\ &= l(\xi_{x,t}^{u,d}(T^-)) = V_l(t,x). \end{aligned} \tag{3.61}$$

We can re-write (3.56) *and* (3.59) *by replacing T by T^-. The rest follows from proof of case (a). Here T^- implies an infinitesimally small change in time and we are effectively computing $V_k(T,x^-) = \min(k(x), l(x))$ and treating $V_k(T,x) = V_k(T,x^-)$. One could derive the same result by considering $V_k(T,x) = \min(k(x), l(x))$.* ■

In other words, the converged warm-starting solution will never be *more* conservative than the initialization, and at least as conservative as the exact solution. We now present a numerical example that uses the result in Theorem 4 to efficiently update the reachable set as the problem parameters change.

Numerical Example: Double Integrator. Double integrator is a canonical benchmarking example in control theory. Its system dynamics are:

$$\dot{x} = \begin{bmatrix} \dot{p} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} v + d \\ ub \end{bmatrix}, \tag{3.62}$$

with states position p and velocity v, where $u \in [-1, 1]$ is acceleration. By default the disturbance is $d = 0$, and there is a default model parameter of $b = 1$. In later examples we will change the disturbance bound and model parameter.

The target set is $\mathcal{L} = \{(p, v) : |p| \leq 2, |v| < \infty\}$. Without loss of generality, we set $T = 0$. This set and its corresponding target function $l(x)$ can be seen in Fig. 3.10 in green. The initial spatial gradients for $V(0, x) = l(x)$ can be seen as black arrows. The Hamiltonian will optimize over the inner product between these gradients and the flow field of the dynamics $f(x, u, d)$, seen as blue arrows. The converged BRT $\mathcal{V}^*$ and value function $V^*(x)$ are in cyan. If the system starts inside $\mathcal{V}^*$, it will eventually enter the unsafe target set even while applying the optimal control (i.e. decelerating/accelerating as much as possible).

We now change the problem parameters and apply warm-start reachability to get the updated BRT. In practice we find that frequently the value function converges to the exact solution even when initialized below the converged value function, i.e. when $k(x) < V_l^*(x)$. Fig. 3.11a demonstrates one such example. The warm-start function $k(x)$ (seen in blue) is initialized to be the original value function acquired when $u \in [-.7, .7]$. If the control authority increases to $u \in [-1, 1]$, standard reachability converges to the cyan value function $V_l^*(x)$. In black is the value function under $V_k^*(x)$ that was initialized by $k(x)$ instead of $l(x)$. Convergence occurs due to the Hamiltonian in (3.57) contracting the value function until the solution has been reached.

To find a result that does not converge exactly and instead results in a conservative solution, we initialize with $k(x) < V_l^*(x)$ that has incorrect gradients everywhere, as shown in blue in Fig 3.11b. This is a fairly unrealistic initial estimate for the true value function,

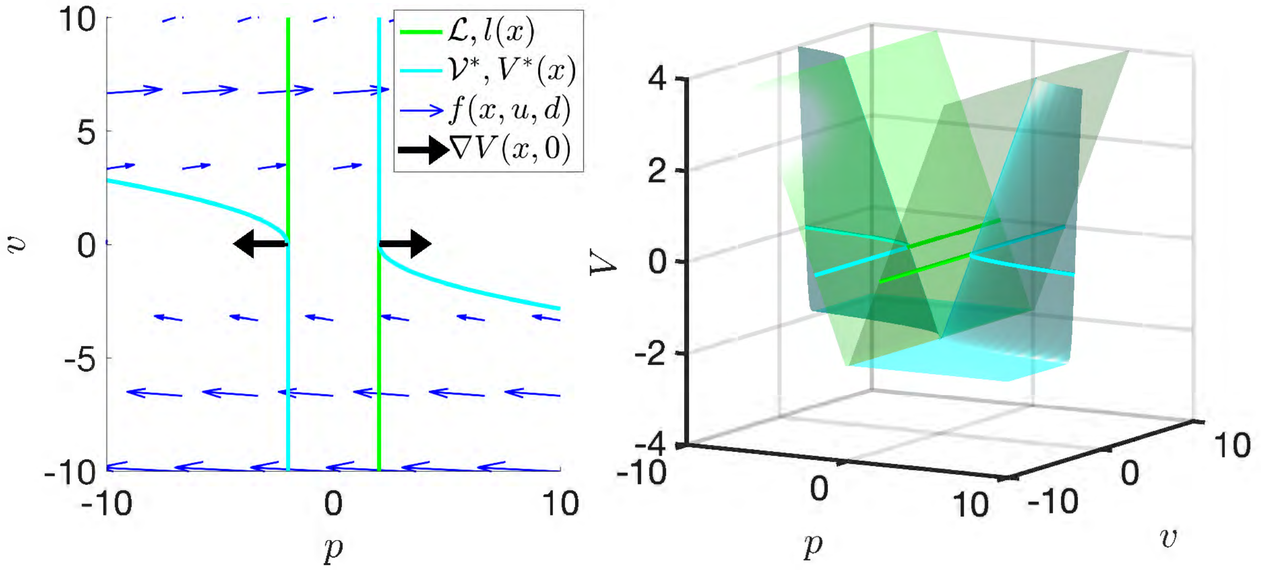


Figure 3.10: Visualization of the numerical example using a double integrator model. The target set $\mathcal{L}$ and corresponding function $l(x)$ are in green. We initialize $V(0,x) = l(x)$, and update the function using (3.57) by optimizing over the inner product between the spatial gradients (seen for $V(0,x)$ as black arrows) and the system dynamics (whose flow field is seen as blue arrows). The converged BRT $\mathcal{V}^*$ and value function $V^*(x)$ are in cyan.

as as the subzero level set $\mathcal{K}$ is the entire state space. As the Hamiltonian contracts the function, convergence occurs at a local solution when the gradients of the value function approach zero. In black we see that $V_k^*(x) < V_l^*(x)$, and the BRT $\mathcal{V}_k^*$ is the entire state space.

In Fig. 3.11c we initialize the warm-starting function as $[V_k(x,0) = k(x)] = 0$ (blue) so that $k(x) \geq V_l^*(x)$ for a subset of the state space. Where $k(x) \geq V_l^*(x)$ convergence is nearly exact (black), with slight conservativeness introduced at the boundary where $k(x) = V_l^*(x)$. Where $k(x) < V_l^*(x)$ the warm-start solution remains flat at $V_k^*(x) = 0$. The resulting BRT $\mathcal{V}_k^*$ is a slight over-approximation of $\mathcal{V}_l^*$.

Though we are able to find cases that lead to conservative results, these cases are hard to come by. In almost all initializations the correct value function was achieved exactly. Fig. 3.11d demonstrates this by initializing $\mathcal{V}_k$ with randomly spaced and sized circles. Similar exact results were found for a variety of system dynamics and problem formulations.

Numerical Example: The 10D Near-Hover Quadrotor. The strength of warm-starting in reducing computation time is best seen in high-dimensional examples. In this example we perform reachability analysis to provide safety guarantees for a 10D nonlinear near-hover quadcopter model. When the quadcopter experiences changes to its constraints or dynamics (e.g. changes in mass or disturbances), it must update its safety guarantees appropriately.

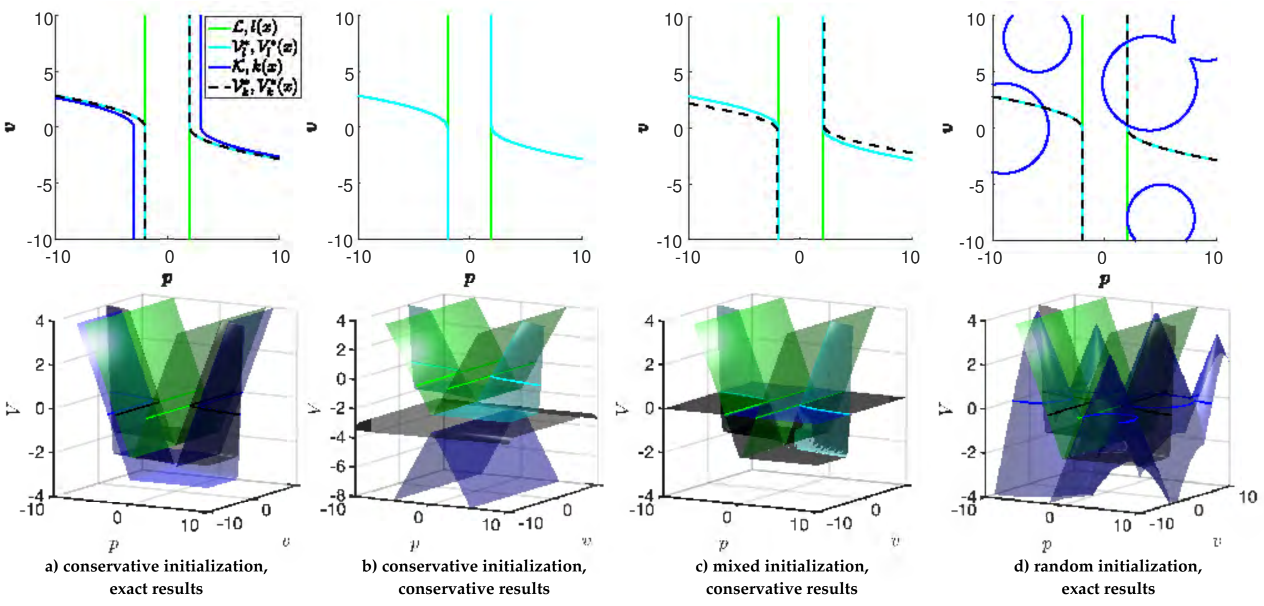


Figure 3.11: The top row shows the target sets and backward reachable tubes, which are the subzero level sets of the target and value functions (bottom row). For all examples shown, green is the target set and function, cyan is the true BRT and converged value function, blue is the warm-start initialization, and black is the warm-start converged value function. (a) conservative warm-start initialization that converges exactly. (b) somewhat unrealistic conservative warm-start initialization that gets stuck in a local solution and results in a conservative value function ($\mathcal{K}$ and $\mathcal{V}_k^*$ are not visualized because they include the entire state space). (c) initializing at zero everywhere ($\mathcal{K}$ not visualized because it includes the entire state space) results in a slightly conservative BRT. (d) to demonstrate how well this algorithm works in practice, we initialize with the complement of random circles, resulting in exact convergence.

The dynamics of the 10D quadrotor are given by (3.25). The parameters d_0, d_1, n_0, k_T, as well as the control bounds $\mathcal{U}$ that we used were $d_0 = 10, d_1 = 8, n_0 = 10, k_T = 4.55, |u_x|, |u_y| \leq 10$ degrees, $0 \leq u_z \leq 2g$. As discussed in Section 3.2.2.2, we can decompose this system into two 4D systems and one 2D system.

In this example the initial mass is $m = 5$ and initial disturbances are $|d_x|, |d_y| \leq 1, |d_z| \leq 1$. As the quadcopter is flying, the mass increases to $m = 5.25$ (say, due to rain accumulation or picking up a package), effectively decreasing the control bounds. In addition, disturbance bounds go up: $|d_x|, |d_y| \leq 1.5$. In this scenario we can warm-start from the previously computed value function to update the safety guarantees exactly. The value function converges to the true solution (max error of 0.189 in p_x, p_y and 0.003 in p_z) in 66 steps (2.8 hours) instead of 87 steps (3.65 hours) for standard reachability. An important aspect of warm-starting is that even before convergence, every iteration of dynamic programming will provide a safer and more accurate value function, continuously bringing the system closer to the true safety guarantees.

If the mass and disturbances instead go down (say, to $m = 4.8, |d_x|, |d_y| \leq .95$), we can guarantee that the warm-start solution will at best be exactly the new solution, and at worst will be a conservative solution. As demonstrated in the previous example, in practice we almost always converge to the correct solution, and this 10D example converges correctly as well (max error of $2.7e-05$ in the p_x, p_y subsystems and .074 in the p_z subsystem). Our warm-starting method took 48 steps, compared to 50 for standard reachability. Though warm-starting does not provide much computational benefit in this case, every iteration toward convergence provides a guaranteed safe over-approximation of the BRT, which is not true for standard reachability.

3.3.1.2 Exact Warm-Start Reachability

In the case in which $k(x) \geq V_l^*(x)$, we are additionally guaranteed to recover the *exact* solution.

Theorem 5 *If we warm-start with $V_k(T,x) = k(x)$, such that $\forall x\ k(x) \geq V_l^*(x)$, then the resultant value function will converge to the true value function, i.e.,*

$$V_k^*(x) = V_l^*(x)$$

Proof: *We will first show that if $\forall x\ k(x) \geq V_l^*(x)$, we have,*

$$V_k(t,x) \geq V_l^*(x) \quad \forall x, t < T \tag{3.63}$$

To prove this result, let us consider $k'(x) = V_l^*(x)$. For $t < T$, using dynamic programming we have,

$$\begin{aligned} V_{k'}(t,x) &= \inf_{d(\cdot)\in\Gamma_t^T} \sup_{u(\cdot)} \min\Big\{ \inf_{\tau\in[t,T)} l(\xi_{x,t}^{u,d}(\tau)), V_{k'}(T, \xi_{x,t}^{u,d}(T))\Big\}, \\ &= \inf_{d(\cdot)\in\Gamma_t^T} \sup_{u(\cdot)} \min\Big\{ \inf_{\tau\in[t,T)} l(\xi_{x,t}^{u,d}(\tau)), k'(\xi_{x,t}^{u,d}(T))\Big\}. \end{aligned} \tag{3.64}$$

For any $t < T$, we can follow the same logic as in (3.60). Hence, $\forall x, t$, we have $V_{k'}(t,x) \leq V_k(t,x)$. Moreover, since $V_{k'}(T,x) = V_l^*(x)$, we know that $V_{k'}(t,x) = V_l^*(x) \quad \forall t$ since $V_l^*(x)$ is the converged value function corresponding to $l(x)$. Hence, $V_k(t,x) \geq V_l^*(x) \quad \forall x, t < T$. Since this holds for all time, it also holds for $t \to -\infty$: $V_k^*(x) \geq V_l^*(x)$.

To prove Theorem 5, we have $\forall x$,

$$\begin{aligned} V_k^*(x) &\leq V_l^*(x) \quad \textit{(Theorem 4)} \\ V_k^*(x) &\geq V_l^*(x) \quad \textit{(Equation (3.63))} \\ \Rightarrow V_k^*(x) &= V_l^*(x) \end{aligned} \tag{3.65}$$

■

Numerical Example: Double Integrator. Though in general we may not know if $k(x) \geq V_l^*(x)$, there are some cases in which this can be proved, and therefore the exact solution can be recovered. For all following examples, $V_l^*(x)$ is the original value function and $\mathcal{V}_l^*$ is the corresponding BRT acquired from standard reachability using the double integrator example introduced in Section 3.3.1.1. We introduce several changes to the problem formulations, resulting in a new $V_{l'}^*(x), \mathcal{V}_{l'}^*$ acquired from standard reachability. Finally, $V_k^*(x), \mathcal{V}_k^*$ are the value function and BRT acquired by warm-starting with $k(x) = V_l^*(x)$ with the changed problem formulation. We further show what happens when the conditions that lead to exact results are reversed. In these cases we cannot guarantee exact convergence, but can guarantee that in each iteration the function will either reduce conservativeness or remain in a local solution (i.e. $k \leq V_k(t,x) \leq V_l^*(x)\ \forall x, t$). We show in Table 3.7 a time comparison for each example to standard and discounted reachability, shown both in runtime and number of iteration steps. For the exact cases we find that warm-starting is consistently faster. For comparison to discounted reachability, we used a discount factor of 0.999 and annealed to a discount factor of 1 once convergence was reached (see [13] for details).

(a) Changing Target Set: When the target set increases ($\mathcal{L}' \supseteq \mathcal{L}$), setting the initialization to the previously converged value results in $[k(x) = V_l^*(x)] \geq V_{l'}^*(x)$ and therefore exact convergence is guaranteed. We demonstrate this in Fig. 3.12a, where the target sets are in green (solid for $\mathcal{L}$, dashed for $\mathcal{L}'$). When warm-starting from the original BRT $\mathcal{V}_l^*$ (cyan), we are able to recover the new BRT $\mathcal{V}_{l'}^*$ (red) exactly, resulting in $\mathcal{V}_k^*$ (black). We similarly show the reverse case for a decreasing target set in Fig. 3.12b.

(b) Changing Control Authority: In many applications the control authority can change over time. This can happen because of several reasons; for example, increasing the mass of a quadrotor leads to a reduction in its effective control authority. We can explicitly modify $\mathcal{U}$ when there is a change in the control bounds or in a model parameter which updates the effective control authority. When the control space is decreased, i.e. $\mathcal{U}' \subseteq \mathcal{U}$, initializing with the previously converged value function will lead to $[k(x) = V_l^*(x)] \geq V_{l'}^*(x)$ and therefore exact convergence is guaranteed.

To demonstrate this case of reduced control authority we vary the parameter b in the system model (3.62). When b decreases, the effective control authority decreases. In Fig. 3.12c we compute the value function for $b = 1$ (cyan). We then compute the value function for $b = .8$ (red). Finally, we warm-start from the original cyan value function and reach the

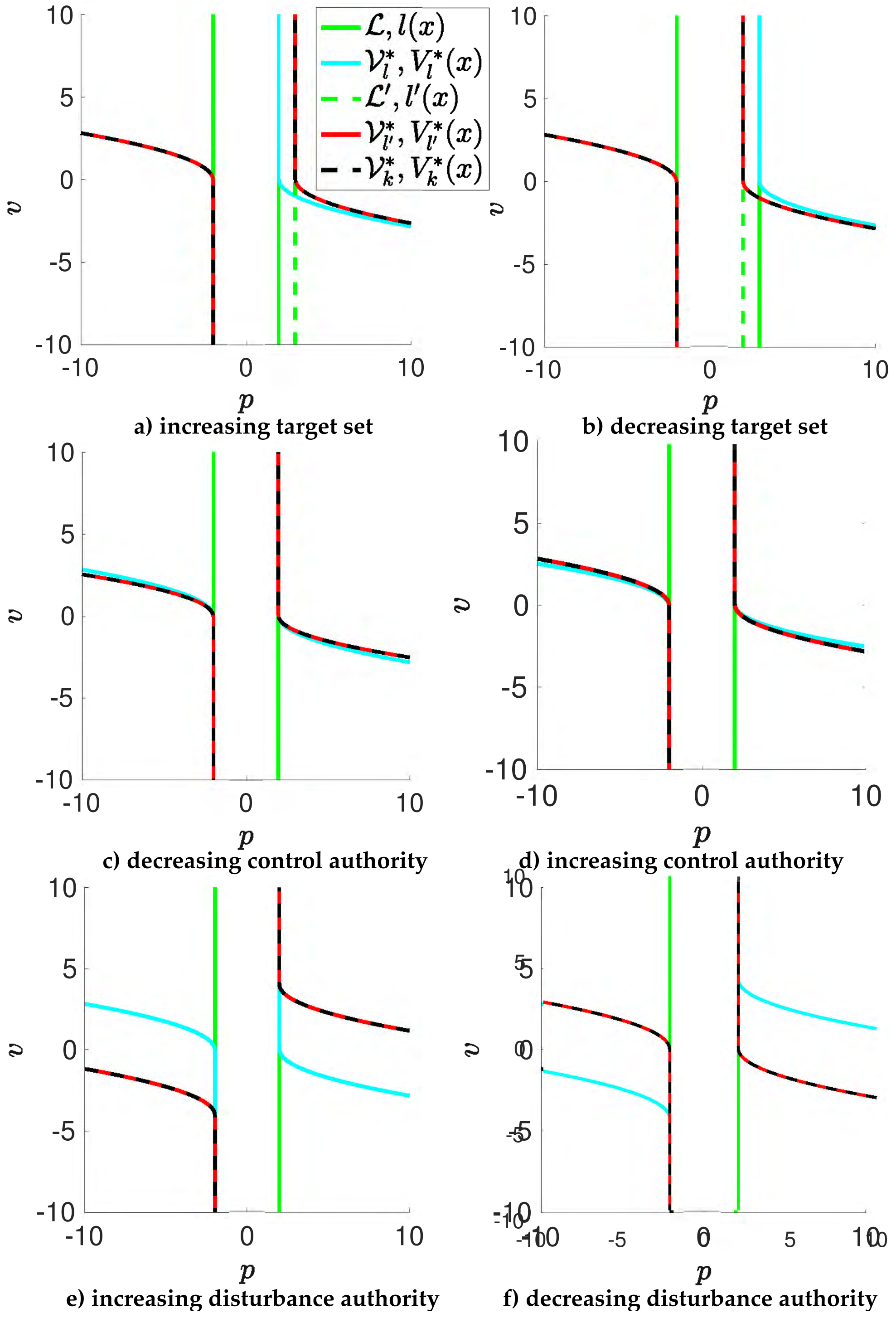


Figure 3.12: For all examples shown, the region between the green lines is the target set. Similarly cyan marks the boundary of the original BRT, red marks the BRT based on new conditions, and black is the boundary of the warm-start converged BRT. The left column shows cases in which the exact solution (red) can be achieved by warm-starting (black) from a previous solution (cyan). The right column shows cases in which warm-starting (black) is guaranteed to at worst remain at the initialization (cyan) or at best will achieve the exact solution (red). In practice we generally achieve the exact solution.

new red value function exactly, as shown in black. We similarly show the reverse case for an increasing control authority in Fig. 3.12d.

Table 3.7: Runtime analysis for reachability methods as the problem parameters change. We compare standard reachability, warm-start formulation, and discounted reachability formulation [13]. We compare both the runtime and the number of iteration steps.

	Standard	Warm-Start	Discounted
a) Increasing $\mathcal{L}$ (exact)	6.4s, 115 steps	6.0s, 109 steps	12.5s, 231 steps
b) Decreasing $\mathcal{L}$ (conserv)	6.0s 110 steps	9.3s, 169 steps	21.2s, 385 steps
d) Decreasing $\mathcal{U}$ (exact)	6.5s, 124 steps	6.0s, 111 steps	20.3s, 374 steps
e) Increasing $\mathcal{U}$ (conserv)	6.8s, 124 steps	6.1s, 111 steps	20.5s, 374 steps
c) Increasing $\mathcal{D}$ (exact)	21.4s, 311 steps	7.7s, 112 steps	13.3s, 195 steps
d) Decreasing $\mathcal{D}$ (conserv)	6.0s, 110 steps	11.7s, 213 steps	19.0s, 346 steps
e) 10D quad, increasing m, $\mathcal{D}$ (exact)	3.7hr, 86 steps	2.8hr, 65 steps	>18 hr, >401 steps
f) 10D quad, decreasing m, $\mathcal{D}$ (conserv)	.68hr, 50 steps	.67hr, 48 steps	1.12hr, 82 steps

(c) Changing Disturbance Authority: Following similar logic to the previous example, we find that increasing $\mathcal{D}$ to a larger $\mathcal{D}'$ has the same effect on the value function as decreasing $\mathcal{U}$ to $\mathcal{U}'$. To demonstrate this, we change the disturbance bounds in our model (3.62). In Fig. 3.12e we compute the value function for $d \in [0, 0]$, shown in blue. We then compute the value function for $d \in [-4, 4]$. Finally, we warm-start from the original cyan value function and reach the new red value function exactly, as shown in black. We similarly show the reverse case for a decreasing disturbance authority in Fig. 3.12f.

3.3.2 Local Update of Backward Reachable Tubes

In the last section, we discussed how warm-starting the value function computation might lead to a faster convergence of the value function; however, the value function is still computed over the entire state space. In this section, we present a more practical algorithm that leverages the advantages of warm-starting by computing and updating the value function only locally at the states for which new information has been obtained since the last value function computation.

Similar to the last section, we will focus on computing infinite-horizon minimal BRTs, which are often desirable in safety-critical applications. Additionally, we will specifically focus on the scenarios, where the target set changes during the runtime and we are interested in computing the updated minimal BRT. This scenario is particularly common in robotic applications, where for example, the robot might have discovered a new obstacle in the environment and hence may want to update its safe set.

Suppose that the target set of the system changes from $\mathcal{L}$ to $\mathcal{L}'$, and we warm-start the value function computation with $k(x)$. Now, instead of updating the value function using the HJI VI everywhere, at each step of the HJI VI iteration, we maintain a list of states $\mathcal{Q}$ at which the value function needs to be updated in light of the new environment observations. $\mathcal{Q}$ is initialized to be the set of states which are recently included/excluded from the target set, i.e., $\mathcal{Q} = \mathcal{L} - \mathcal{L}'$. Here, $-$ represents the set difference between $\mathcal{L}$ and $\mathcal{L}'$. Since the change in the value of the states in $\mathcal{Q}$ (compared to $V_l^*(x)$) would also cause a change in the value of the neighboring states, $\mathcal{N}(\mathcal{Q})$, we also add them to $\mathcal{Q}$. Thus, $\mathcal{Q} = \mathcal{Q} \cup \mathcal{N}(\mathcal{Q})$. Typically, the value function in the HJI VI is computed by discretizing the state-space into a grid and solving the VI over that grid (see Sec. 2.3.3 for example). In such cases, the spatial derivative of the value function (required to compute the Hamiltonian in the HJI VI in (3.57)) is computed numerically using the neighboring grid points. This spatial derivative is precisely responsible for the propagation of the change in the value function at a state to its neighboring states. In such cases, $\mathcal{N}(\mathcal{Q})$ might represent the neighboring grid points used to compute the spatial derivative of the value function for the states in $\mathcal{Q}$; however, other neighboring criteria can be used.

Once the neighbors are added to $\mathcal{Q}$, the value for all the states in $\mathcal{Q}$ is initialized using $V_l^*(x)$. There are multiple good candidates for warm-starting the value function here. One possibility is to initialize the computation with the previously converged value function, i.e., $k(x) = V_l^*(x)$. Another good candidate for initialization is:

$$k(x) = \begin{cases} l'(x), & \text{if } x \in \mathcal{L} - \mathcal{L}' \\ V_l^*(x), & \text{otherwise} \end{cases} \tag{3.66}$$

where $\mathcal{L}' = \{x : l'(x) \leq 0\}$. Intuitively, instead of initializing the value function with $V_l^*(x)$ or $l'(x)$ everywhere in the state space, (3.66) initializes it with the last computed value function for the states where no new information has been obtained since the last computation, and with $l'(x)$ only at the states which were previously assumed to be unsafe but are actually safe, or vice versa. This leads to a much faster computation of the BRT because the value function needs to be updated only for a much smaller number of states that are newly found out to be safe (or unsafe). At all the other states, the value function is already almost accurate and only small refinements are required. Given $k(x)$, the value of the states in $\mathcal{Q}$ is updated using the HJI VI in (3.57) for some time step ΔT. Again, this computation is much faster than a classical HJI VI computation since it is typically performed for many fewer states. Next, we remove all those states from $\mathcal{Q}$ whose value function hasn't changed significantly over this ΔT, as these states won't cause any change in the value function for any other state. The neighbors of the remaining states are next added to $\mathcal{Q}$ and the entire procedure is repeated until the value function is converged for all states. We outline this procedure in Algorithm 1.

Note that Algorithm 1 still maintains the conservatism of the safe set since it is just a different computational procedure for computing the warm-started value function. Hence the results in Theorem 4 and Theorem 5 can still be used to establish the conservatism (or exactness) of the value function obtained using local update procedure. The key intuition behind the local update algorithm is that if we warm-start with a previously converged value function in the parts of the state space where the target function has not changed, then the

Algorithm 1: Local update methods to compute the Minimal BRT

```
1  $\mathcal{Q} \longleftarrow \mathcal{L} - \mathcal{L}'$: Initialize list of states for which the value function should be updated
2  $\mathcal{Q} \longleftarrow \mathcal{Q} \cup \mathcal{N}(\mathcal{Q})$: Add neighboring states to $\mathcal{Q}$
3  Warm-start the value for states in $\mathcal{Q}$, $V(0, \mathcal{Q})$, using $k(x)$
4  $V_{\text{old}} \longleftarrow V(0, \mathcal{Q})$: The last computed value function for states in $\mathcal{Q}$
5  while $\mathcal{Q}$ is not empty do
6      $V_{\text{updated}} \longleftarrow$ Update the value function $V_{\text{old}}$ for a time step $\Delta T$
7      $\Delta V = \|V_{\text{updated}} - V_{\text{old}}\|$: Change in the value function
8      $\mathcal{Q}_{\text{remove}} \longleftarrow \{x \in \mathcal{Q} : \Delta V = 0\}$: States with unchanged value
9      $\mathcal{Q} \longleftarrow \mathcal{Q} - \mathcal{Q}_{\text{remove}}$: Remove states with unchanged value
10     $\mathcal{Q} \longleftarrow \mathcal{Q} \cup \mathcal{N}(\mathcal{Q})$: Add neighboring states to $\mathcal{Q}$
11     $V_{\text{old}} \longleftarrow V_{\text{updated}}$
```

HJI VI would not induce any change in the value function at a state until the value function has been changed for some state in the neighborhood of that state. Thus, it is sufficient to start the warm-start computation only at the states which have recently been included in/excluded from the target set. This, somewhat straightforward, modification in the warm-start reachability leads to significant savings in computation. Table 3.8 summarizes the computation time for the BRT for the 3D Dubins car model with dynamics (3.52). Warm-start reachability is able to compute the BRT 3 times faster than a standard HJI VI. The computed solution is almost exact with only 0.41% conservative volume. Local update method is able to reduce the computation time of the warm-start method by a factor of 13, leading to a 40 times faster computation of the BRT compared to the standard reachability initialization.

Table 3.8: Numerical comparison of average compute time and relative volume of over-conservative states for different BRS update methods. Local updates compute an almost exact BRS in ≈ 1 second, and are significantly faster than both the standard HJI VI and the warm-start reachability.

Metric	**Standard HJI VI**	**Warm-Start Reachability**	**Local Update**
Average Compute Time (s)	35.97	12.16	0.94
% Over-conservative States	0.0	0.41	0.39

3.3.2.1 Practical Considerations

When implementing a safety framework on real systems, there are many practical considerations that should be acknowledged. Below we discuss some of the main practical considerations we think are worth noting when using our proposed framework:

- Since the value function is computed over a discretized state space, it might incur some numerical inaccuracies. Using a finer discretization and a higher order approximation for the spatial derivative is often helpful in alleviating these issues; however, the computation time also increases consequently. In our experience, we have found the 3rd or higher order approximations schemes to work pretty well.

- Since we are interested only in the zero level of the value function for the BRT computation, we can obtain further computational saving by leveraging narrow-band level-set methods [9] within the local update method proposed here. Narrow-band methods update the value function only around the zero level set boundary, thus reducing the computational complexity of the value function. One can use the local update method only around the zero-level, thus combining the advantages of both methods.

3.4 The Berkeley Efficient API in C++ for Level Set Methods (BEACLS)

In the previous sections, we have seen how warm-starting the reachability computation and/or locally updating the reachable set can lead to a significant decrease in the computation time. However, even with these algorithms, reachability analysis is not suitable for real-time applications. For example, even for a 3D system, we still need ≈ 1 second to update the reachable set (Table 3.8), which is still not enough for highly dynamic autonomous systems which might require us to update the reachable sets at the order of milliseconds. To meet these requirements, algorithmic advancements alone are not sufficient, we also have to leverage computational advances. In this section, we introduce BEACLS, a C++-based reachability toolbox that can leverage GPU parallelization to improve computation speed of HJ reachability by nearly 100 times compared to existing MATLAB implementations. BEACLS combined with local update allows us to update the reachable set of 3D Dubins car within 10ms for instance! As we will see later, this enables us to use HJ reachability to maintain safety guarantees in dynamic environments, which otherwise is very challenging. Similarly, we need efficient computational tools for HJ reachability for large-scale autonomous systems, involving, for example hundreds to thousands of robots (see Chapter 4 for instance).

BEACLS is an implementation of existing Matlab toolboxes (Level set toolbox [219] and helperOC library [286]) in C++ and CUDA. However, it has two key features that allow enable parallel computation of reachable sets: (a) approximate computation of the dissipation term in Lax-Friedrichs scheme and (b) splitting of multi-dimensional grids into smaller chunks.

As with the level set toolbox [219], the Hamiltonian is approximated using the Lax-Friedrichs scheme while solving PDEs and VIs. The Lax-Friedrichs scheme stabilizes the numerical integration by adding a dissipation term into the approximation of the Hamiltonian

[239]. For example, in a 2D state space, this is written as follows:

$$\hat{H} = H(\frac{\hat{V}_x^- + \hat{V}_x^+}{2}, \frac{\hat{V}_y^- + \hat{V}_y^+}{2}) - \alpha^x(\frac{\hat{V}_x^+ - \hat{V}_x^-}{2}) - \alpha^y(\frac{\hat{V}_y^+ - \hat{V}_y^-}{2}) \tag{3.67}$$

where $\hat{V}_x^-, \hat{V}_x^+, \hat{V}_y^-, \hat{V}_y^+$ are respectively left and right finite difference approximations of partial derivatives of V, and α^x, α^y are dissipation coefficients, given by

$$\alpha^x = \max_\lambda |\frac{\partial H(x,\lambda)}{\lambda_x}|, \quad \alpha^y = \max_\lambda |\frac{\partial H(x,\lambda)}{\lambda_y}|, \tag{3.68}$$

The above formulas trivially generalize to higher-dimensional state spaces. Note that dissipation coefficients α require maximization of the costate λ (or gradient of the value function) over the entire grid. Previously, in the Matlab implementation, the costate is stored over the entire grid to take advantage of Matlab's accelerated vectorized computations at the cost of memory usage. Thus, previously maximizing over λ was trivial.

In BEACLS, we chose to only store the value function during computations to greatly improve space efficiency, since each component of the costate would require the same amount of memory as the value function itself. This optimization is important for computations using CPU due to the use of lower-level cache and for computations using GPU due to the lax of video random-access memory (VRAM) as described later. However, since λ is not stored over the entire grid, maximizing over λ would require a second pass over the entire grid, which would drastically reduce computation speed. To address this, we observed that α typically does not change quickly, which allows us to use the value of α from the previous integration time step. We have observed no stability issues in our computations. In the worst case, instabilities can typically be resolved by scaling α by a constant less than 1.

In place of the native MATLAB functionalities and toolboxes used by the level set toolbox and helperOC library, BEACLS uses several open source C++ libraries. These libraries include MATIO for loading and saving .mat files which store value functions and trajectories, OpenCV for visualizations, and several others for computation, as depicted in Fig. 3.13.

The another key feature that allows parallelized computation is the splitting of multi-dimensional grids into smaller chunks. These smaller chunks fit into the cache of multi-core CPUs, in which each CPU core only has access to its own cache. The chunks are much larger when computation is done on multiple GPUs; in this case, the chunks need to fit into the VRAM of each GPU. Fig. 3.14 depicts the general procedure of splitting up a grid into appropriate chunks, computing partial derivatives, and recombining the chunks. A few other notable features are as follows:

- To allow parallel computation, the chunks have sufficient overlap to allow correct gradient computations at the boundaries of each chunk.
- Unlike the MATLAB libraries, the computational grid is not explicitly stored in a multi-dimensional array, an implementation that is necessary to improve computation

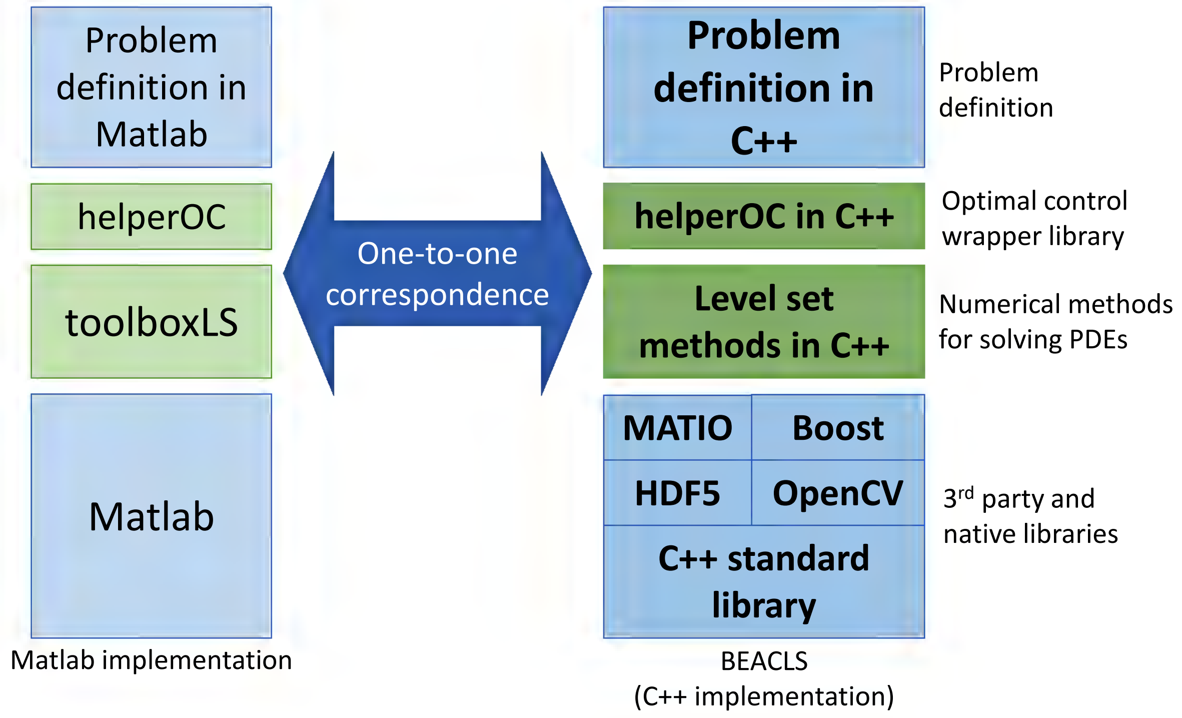

Problem definition in Matlab
helperOC
toolboxLS
Matlab
Matlab implementation
One-to-one correspondence
Problem definition in C++
helperOC in C++
Level set methods in C++
MATIO
Boost
HDF5
OpenCV
C++ standard library
BEACLS (C++ implementation)
Problem definition
Optimal control wrapper library
Numerical methods for solving PDEs
3rd party and native libraries

Figure 3.13: Correspondence between the Matlab and BEACLS implementations of reachability toolbox.

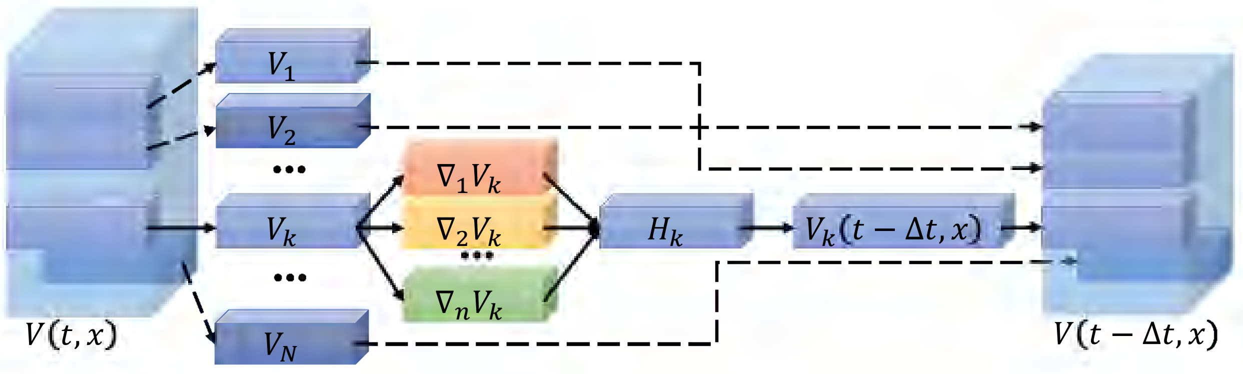


Figure 3.14: BEACLS splits multi-dimensional arrays representing the value function $V(t,x)$ into appropriate overlapping chunks according to processor configuration. Numerical gradients and the Hamiltonian values of each chunk are computed in parallel to produce the updated value function at the previous time step (in the case of backward reachability). Finally, the chunks are combined together to form the updated value function over the entire computational domain.

time through vectorized computations in MATLAB. Instead, BEACLS stores only the value function (and not the grid) in multi-dimensional arrays, which greatly reduces memory usage.

Figure 3.15 shows at a glance the computation times for implementations of the level set methods used to solve a benchmark 3D HJI VI (different than the one presented in Sec. 3.3.2), which provides the BRS and the optimal trajectory for the system. The software implementations are the MATLAB level set toolbox in [219], a C++ implementation of [219] in BEACLS, and [286], and a Compute Unified Device Architecture (CUDA) implementation of [219] and [286] in BEACLS with the C++ interface. The MATLAB and C++ implementations are run on a Core i7-5820K CPU, and the CUDA implementation was run on one or two Geforce GTX Titan X GPUs. When run on two GPUs, BEACLS is approximately 100 times faster compared to the MATLAB implementation.

This improvement in computation time greatly facilitates case studies such as city-level airspace design, which may involve testing different initial and goal positions at different vehicle densities and wind speeds, as we demonstrate in Chapter 4. Trajectory planning with 50 vehicles takes less than 2 minutes, compared to 27 minutes for a non-CUDA C++ implementation or 2.8 hours for the MATLAB implementation. For even larger-scale case studies, a GPU-parallelized implementation such as BEACLS may be necessary to keep the computation tractable. For example, in the study presented in Chapter 4 involving multiple cities and 200 vehicles, computation for each vehicle was approximately 4 minutes on average using two GPUs. This is because a much finer grid was needed to maintain positional accuracy over a much larger geographical area. The entire simulation took approximately 13.3 hours. Extrapolating the computation time comparison in Fig. 3.15, even a non-CUDA C++ implementation would take prohibitively long – approximately 10 days.

BEACLS, along with more detailed documentation, is available at the BEACLS website[7].

3.5 Chapter Summary

This chapter introduce several algorithmic and computational frameworks to combat the exponentially scaling computational complexity of Hamilton-Jacobi reachability analysis. We first present a method that decomposes a high-dimensional reachability problem into a number of small-dimensional reachability problems. These small dimensional reachable sets are then projected back to the full state space and combined together to get the reachable set for high-dimensional systems. We also present the conditions under which such decomposition can be performed. Using the proposed method, we are able to compute the reachable sets for some high-dimensional systems for the first time.

We then turn our attention to the scenarios where these reachable sets must be updated during run-time due to potential changes in the problem parameters or due to new environment information. We propose a method of warm-start reachability that can update

[7] BEACLS website: https://github.com/HJReachability/beacls

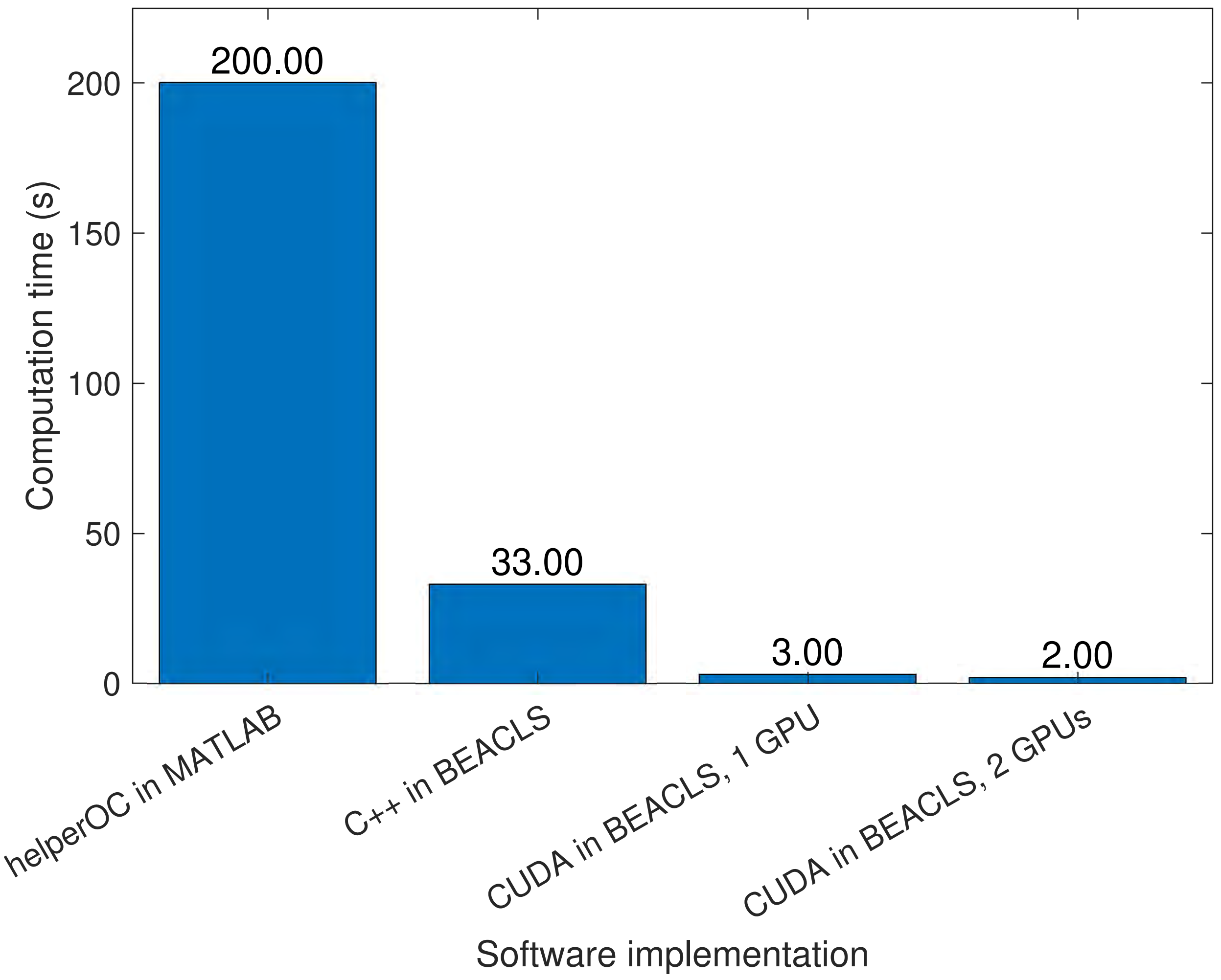


Figure 3.15: Summary of computation times for a benchmark 3D HJI VI. A 50-UAV simulation takes less than 2 minutes with the CUDA implementation in BEACLS using two GPUs, compared to 27 minutes with a non-CUDA C++ implementation in BEACLS or 2.8 hours with helperOC and the level set toolbox in MATLAB.

reachable set faster compared to recomputing the reachable set from scratch by using a smart initialization while still maintaining safety guarantees. Building upon the warm-start reachability, we propose a method to locally update the reachable set in the regions of the state space where new environment information has been acquired. As we will see in Chapter 9, these methods will be crucial in maintaining safety guarantees when an autonomous system operates in *a priori* unknown environment.

Finally, we present BEACLS, a C++-based computational tool for computing reachable sets. BEACLS can leverage modern computational tools such as GPUs to parallelize the reachable set computation, achieving a 100 fold speedup compared to state-of-the-art reachability libraries. As we will see in Chapter 4, this allows us to develop highly scalable schemes for provably safe trajectory planning for large scale, multi-vehicle systems.

Chapter 4

Provably Safe and Scalable Multi-Vehicle Trajectory Planning

This chapter is based on the papers "Safe Sequential Path Planning of Multi-Vehicle Systems Under Disturbances and Imperfect Information" [33], "Robust Sequential Path Planning Under Disturbances and Adversarial Intruder " [73], and "Provably Safe and Scalable Multi-Vehicle Trajectory Planning" [35] written in collaboration with Mo Chen, Jaime Fisac, and Claire Tomlin.

Recently, there has been an immense surge of interest in the use of unmanned aerial systems (UASs) in urban environments. UASs have great potential in civil applications such as package delivery, aerial surveillance, disaster response, among many others [289, 92, 20, 27, 44]. Unlike previous uses of UASs for military purposes, these civil applications will involve unmanned aerial vehicles (UAVs) flying in urban environments, potentially in close proximity of humans, other UAVs, and other important assets. As a result, government agencies such as the Federal Aviation Administration (FAA) and National Aeronautics and Space Administration (NASA) of the United States are urgently trying to develop new scalable ways to organize an airspace in which potentially thousands of UAVs can fly simultaneously in the same region [161, 250]. One essential question that needs to be answered for enabling such applications is that of multi-vehicle trajectory planning: how a group of vehicles in the same vicinity can reach their destinations while avoiding situations which are considered dangerous, such as collisions? Providing an answer to the above question is the focus of this chapter.

Trajectory planning is one of the core problems in robotics, and thanks to the ubiquity of autonomous systems in industrial applications, a significant progress has been made on this problem in the past two decades. Traditionally, the problem of trajectory planning has been studied in the context of a single robot operating in a static environment. However trajectory planning problem becomes significantly more complex for multi-robot systems, since the robot's success in safely completing its task is not only determined by its own decisions, but also by the actions of other robots or agents present in its environment. Safe decision making only becomes more complex when external disturbances and potential adversarial agents are present in the environment.

One way to solve multi-vehicle trajectory planning problem is to assume no knowledge about the intent or behavior of the other agents, and safeguard against all possible behaviors of other agents. However, this leads to an overly pessimistic robot behavior and often results into a "frozen" robot. In this chapter, we will focus on the cooperative trajectory planning case, in which the multiple agents navigating the environment have the ability to share information and compute a coordinated solution. Thus, individual agents (such as autonomous mobile robots or vehicles) may forgo part or all of their decision authority in exchange for the safety assurances obtained from coordination. **We propose a Hamilton-Jacobi reachability-based approach for safe cooperative trajectory planning for multi-vehicle systems. Our approach guarantees both goal satisfaction and safety for vehicles with general nonlinear dynamics while taking into account disturbances and potential adversarial agents.** Towards the end of this chapter, we will discuss how to perform safe multi-vehicle trajectory planning when the vehicles are not necessarily cooperative.

4.1 Related Work

Reactive Collision Avoidance Approaches. One popular class of approaches for multi-vehicle trajectory planning is reactive feedback-based collision avoidance. Reactive methods seek to achieve collision avoidance by implementing real-time decision rules for the different agents based on their current state (or observations). These methods include those using *virtual potential fields* [237, 83], which discourage collisions by modifying vehicle control actions so that there is an effective repulsive force between the vehicles. Other methods use *velocity obstacles* [113, 295, 305, 133], which anticipate the future positions of vehicles based on their current speeds. Collision avoidance among vehicles with kinematics Dubins Car dynamics has also been analyzed in [241], in which the authors propose reserving local regions around each vehicle to allow for predetermined maneuvers; this framework is formalized using a hybrid automaton. The authors of [310] proposed a Voronoi-based method that guarantees collision avoidance under decentralized planning for holonomic vehicles. While reactive methods provide computationally cheap feedback-based rules to resolve local conflicts, it is not usually possible to derive safety guarantees from them, at least without strong assumptions on vehicle dynamics or control strategy.

Planning-Based Approaches. In contrast to reactive approaches, planning-based approaches explicitly reason about the future motion of all vehicles in the environment, which is then used to plan collision-free trajectory for the vehicle. Many existing methods for path planning come from the robotics literature; these methods are primarily based on building probabilistic and sampling-based roadmaps [190], or leveraging differential flatness to plan in a reduced state space [300, 213]. We will provide a more detailed overview of path planning literature in robotics in Chapter 9.

Path planning literature for multi-vehicle systems include methods for real-time trajectory generation [197], for path planning for vehicles with linear dynamics in the presence of obstacles with known motion [12], and for cooperative path planning via waypoints which do not account for vehicle dynamics [51] or which indirectly account for dynamics via smoothing through a mixed integer program [214]. Other related work include those which consider only the collision avoidance problem without path planning. These results include those that assume the system has a linear model [46, 268, 281], rely on a linearization of the system model [212, 17], assume a simple positional state space [198], and many others [187, 156, 78]. However, scalable methods to flexibly plan provably safe and dynamically feasible trajectories without making strong assumptions on the vehicles' dynamics and other vehicles' motion are still lacking.

The problem of trajectory planning and collision avoidance for safety-critical systems that overcome some of the above limitations has been studied using Hamilton-Jacobi (HJ) reachability analysis. In particular, a reach-avoid set is computed in the joint state-space of all vehicles with the target and the obstacle sets being the intersection of individual target and obstacle sets (projected to the full state space) respectively. Reachability analysis has been successfully used in applications involving systems with no more than two vehicles [222, 97, 158, 43]. One of the main challenges of managing the next generation of airspace is the density of vehicles that needs to be accommodated [250]. Such a large-scale system has a high-dimensional joint state space, making a direct application of dynamic programming-based approaches such as reachability analysis intractable.

Contributions and Chapter Organization

In this chapter, we propose the *sequential trajectory planning* (STP) method to tackle the cooperative, multi-vehicle trajectory planning problem. Our approach can be best characterized as a planning-based approach; however, it provides hard guarantees on both the goal satisfaction and safety of all vehicles even in the presence of disturbances and a single intruder vehicle that could potentially be adversarial. In addition, our method scales only *linearly* with the number of vehicles when there is no intruder, and *quadratically* with the number of vehicles when there is a single intruder. On a high level, STP assigns a strict priority ordering to vehicles under consideration. Higher-priority vehicles plan trajectories without accounting for lower-priority vehicles. Lower-priority vehicles treat higher-priority vehicles as time-varying (or moving) obstacles. Under this assumption, time-varying HJ reachability [57, 115] (also see Sec. 2.3.2) can be used to obtain optimal and provably safe trajectories for each vehicle, starting from the highest-priority vehicle.

In a sense, the STP method reserves a portion of "space-time" in the airspace for each vehicle. The reserved space-time portion is recorded so that lower-priority vehicles can take it into account. Besides planning around the reserved space-time portions of higher-priority vehicles, no other communication between the vehicles is needed at execution time, even when disturbances and an intruder are present. Almost all computation is done *offline* to produce a value function, corresponding to an appropriate differential game, for each vehicle. The gradient of the value function can be stored in a look-up table, which is used *online* to synthesize the optimal controller. Controller synthesis amounts to evaluating an analytic expression, is the only online computation, and therefore can be done in real-time.

We formally state the problem of multi-vehicle trajectory planning in Section 4.2. We discuss a HJ reachability-based algorithm for sequential trajectory planning in the presence of disturbances but in the absence of an intruder in Section 4.3. Under the STP algorithm, each vehicle declares a nominal trajectory which can be robustly tracked under disturbances.

In scenarios where there could potentially be single, possibly adversarial intruder in the airspace, each vehicle needs extra space around other vehicles in order to be able to perform avoidance maneuvers. Assuming the intruder may be present for some maximum duration, we use use reachability analysis to determine precisely the amount of space-time needed for each vehicle to be able to avoid the intruder under the presence of disturbances, making our proposed method sufficiently robust to most practical scenarios. STP in the presence of a single intruder is formally presented in Section 4.4.

4.2 Safe Multi-Vehicle Trajectory Planning

Consider N vehicles which participate in the STP process and denote these vehicles as the *STP vehicles* $Q_i, i = 1, \dots, N$. We assume their dynamics are given by

$$\begin{aligned} \dot{x}_i &= f_i(x_i, u_i, d_i), t \le t_i^{\text{STA}} \\ u_i &\in \mathcal{U}_i, d_i \in \mathcal{D}_i, i = 1 \dots, N \end{aligned} \tag{4.1}$$

where $x_i \in \mathbb{R}^{n_{x_i}}$ represents the state of vehicle Q_i, $u_i \in \mathcal{U}_i$ the control of Q_i, and $d_i \in \mathcal{D}_i$ the disturbance experienced by Q_i. For convenience, we partition the state x_i into the position component $p_i \in \mathbb{R}^{n_p}$ and the non-position component $h_i \in \mathbb{R}^{n_{x_i} - n_p}$: $x_i = (p_i, h_i)$. Note that here we are overloading the notation x_i – it represents the state of Q_i and not the state at time i.

Each vehicle Q_i has initial state x_i^0, and aims to reach its target $\mathcal{L}_i$ by some scheduled time of arrival t_i^{STA}. The target in general represents some set of desirable states, for example the destination of Q_i. In some situations, we may find that it is infeasible for Q_i to get to $\mathcal{L}_i$ at or before t_i^{STA}. Whenever unsure, we may first determine the earliest feasible t_i^{STA} for the vehicle.

On its way to $\mathcal{L}_i$, Q_i must avoid a set of static obstacles $\mathcal{O}_i^{\text{static}} \subset \mathbb{R}^{n_{x_i}}$. The interpretation of $\mathcal{O}_i^{\text{static}}$ could be a tall building, a region around an airport, or any set of states that are forbidden for each STP vehicle. In addition to the static obstacles, each vehicle Q_i must also avoid the danger zones with respect to every other vehicle $Q_j, j \neq i$. The danger zones in general can represent any joint configurations between Q_i and Q_j that are considered to be unsafe. In this paper, we define the danger zone of Q_i with respect to Q_j to be

$$\mathcal{Z}_{ij} = \{(x_i, x_j) : \|p_i - p_j\|_2 \le R_c\} \tag{4.2}$$

whose interpretation is that Q_i and Q_j are considered to be in an unsafe configuration when they are within a distance of R_c of each other. For concreteness, we will call $\mathcal{Z}_{ij}$ the collision set, and if $(x_i, x_j) \in \mathcal{Z}_{ij}$, then Q_i and Q_j are said to have collided. Note that there may exist an inevitable danger zone given the position-based danger zone $\mathcal{Z}_{ij}$ (for example the states from which wind can push an aerial vehicle into $\mathcal{Z}_{ij}$ irrespective of control applied by the vehicle). Reachability analysis presented in this chapter by definition guarantees that vehicles avoid such inevitable danger zones. Finally, if the vehicles are not particles (as often is the case), the danger zone $\mathcal{Z}_{ij}$ can be defined to include the effective sizes of the vehicles.

Given the set of STP vehicles, their targets $\mathcal{L}_i$, the static obstacles $\mathcal{O}_i^{\text{static}}$, and the vehicles' danger zones with respect to each other $\mathcal{Z}_{ij}$, we would like, for each vehicle Q_i, to synthesize a controller which guarantees that Q_i reaches its target $\mathcal{L}_i$ at or before the scheduled time of arrival t_i^{STA}, while avoiding the static obstacles $\mathcal{O}_i^{\text{static}}$ as well as the danger zones with respect to all other vehicles $\mathcal{Z}_{ij}, j \neq i$. In addition, we would like to obtain the latest departure time t_i^{LDT} such that Q_i can still arrive at $\mathcal{L}_i$ on time.

Note that as long as it is feasible a vehicle to reach its target in the *absence* of all other vehicles, producing a safe and timely trajectory is *always* feasible using our proposed algorithms, since the latest departure time t_i^{LDT} is obtained. Indeed, if the environment is expected to be crowded, a vehicle can simply depart early enough (and potentially arrive very early) to "avoid traffic". In practice, if departing at or before the latest departure time t_i^{LDT} is not possible, then arriving on time is infeasible.

4.3 Sequential Trajectory Planning

In general, the optimal trajectory planning problem posed in Sec. 4.2 must be solved in the joint space of all N STP vehicles. However, due to the high joint dimensionality, a direct dynamic programming-based solution is intractable. Therefore, we propose to assign a priority to each vehicle, and perform STP given the assigned priorities. Without loss of generality, let Q_j have a higher priority than Q_i if $j < i$. Under the STP scheme, higher-priority vehicles can ignore the presence of lower-priority vehicles, and perform trajectory planning without taking into account the lower-priority vehicles' danger zones. A lower-priority vehicle Q_i, on the other hand, must ensure that it does not enter the danger zones of the higher-priority vehicles $Q_j, j < i$; each higher-priority vehicle Q_j induces a set of time-varying obstacles $\mathcal{O}_i^j(t)$, which represents the possible states of Q_i such that a collision between Q_i and Q_j could occur.

It is straight-forward to see that if each vehicle Q_i is able to plan a trajectory that takes it to $\mathcal{L}_i$ while avoiding the static obstacles $\mathcal{O}_i^{\text{static}}$ and the danger zones of *higher-priority vehicles* $Q_j, j < i$, then the set of STP vehicles $Q_i, i = 1, \dots, N$ would all be able to reach their targets safely. With the STP scheme, the additional structure provided by the vehicle priorities allows us to reduce the complexity of the joint trajectory planning problem. As we will see, under the STP scheme, trajectory planning can be done sequentially in descending order of vehicle priority in the state space of only a single vehicle. Thus, STP provides a solution whose complexity scales linearly with the number of vehicles in the presence of disturbances, as opposed to exponentially with a direct application of dynamic programming approaches. In the presence of a single intruder, the computation complexity scaling becomes quadratic. In general, the priorities of vehicles may be assigned in a first-come-first-serve basis through, for example, an air navigation service provider [250] which manages a region of the low altitude airspace.

In the following sections, we will explore STP under different assumptions. We begin with the basic STP algorithm in which disturbances are ignored and perfect information of vehicles' positions is assumed. This simplification allows us to clearly establish the basic STP algorithm. Next, we show how the basic STP approach can be made robust to disturbances. Finally, we further robustify the STP approach by considering how the set of STP vehicles may respond to the presence of an intruder vehicle which may be adversarial. All of our methods use time-varying reachability analysis (Sec. 2.3.2) to provide goal satisfaction and safety guarantees.

4.3.1 Sequential Trajectory Planning Without Disturbances

In this section, we introduce the basic STP algorithm assuming that there is no disturbance affecting the vehicles, and that each vehicle knows the exact position of higher-priority vehicles. Although in practice, such assumptions do not hold, the description of the basic STP algorithm will introduce the notation needed for describing the subsequent, more realistic versions of STP. We also show simulation results for the basic STP algorithm.

Recall that the STP vehicles $Q_i, i = 1, \dots, N$, are each assigned a strict priority, with Q_j having a higher priority than Q_i if $j < i$. In the absence of disturbances, we can write the dynamics of the STP vehicles as

$$\dot{x}_i = f_i(x_i, u_i), \quad t \leq t_i^{\text{STA}}, \quad u_i \in \mathcal{U}_i, \quad i = 1 \dots, N \tag{4.3}$$

In STP, each vehicle Q_i plans the trajectory to its target set $\mathcal{L}_i$ while avoiding static obstacles $\mathcal{O}_i^{\text{static}}$ and the obstacles $\mathcal{O}_i^j(t)$ induced by higher-priority vehicles $Q_j, j < i$. Path planning is done sequentially starting from the first vehicle and proceeding in descending priority, $Q_1, Q_2, \dots, Q_N$ so that each of the trajectory planning problems can be done in the state space of only one vehicle. During its trajectory planning process, Q_i ignores the presence of lower-priority vehicles $Q_k, k > i$, and induces the obstacles $\mathcal{O}_k^i(t)$ for $Q_k, k > i$.

From the perspective of Q_i, each higher-priority vehicles $Q_j, j < i$ induces a time-varying obstacle denoted $\mathcal{O}_i^j(t)$ that Q_i needs to avoid[1]. Therefore, each vehicle Q_i must plan its trajectory to $\mathcal{L}_i$ while avoiding the union of all the induced obstacles as well as the static obstacles. Let $\mathcal{G}_i(t)$ be the union of all the obstacles that Q_i must avoid on its way to $\mathcal{L}_i$:

$$\mathcal{G}_i(t) = \mathcal{O}_i^{\text{static}} \cup \bigcup_{j=1}^{i-1} \mathcal{O}_i^j(t) \tag{4.4}$$

[1]Note that the index k in $\mathcal{O}_k^i$ denotes vehicles with lower priority than Q_i, and the index j in $\mathcal{O}_i^j(t)$ denotes vehicles with higher priority than Q_i.

With full position information of higher priority vehicles, the obstacle induced for Q_i by Q_j is simply

$$\mathcal{O}_i^j(t) = \{x_i : \|p_i - p_j(t)\|_2 \leq R_c\} \tag{4.5}$$

Each higher priority vehicle Q_j plans its trajectory while ignoring Q_i. Since trajectory planning is done sequentially in descending order or priority, the vehicles $Q_j, j < i$ would have planned their trajectories before Q_i does. Thus, in the absence of disturbances, $p_j(t)$ is *a priori* known, and therefore $\mathcal{O}_i^j(t), j < i$ are known, deterministic moving obstacles, which means that $\mathcal{G}_i(t)$ is also known and deterministic. Therefore, the trajectory planning problem for Q_i can be solved by first computing the reach-avoid BRT, $\mathcal{V}_i^{\text{basic}}(t)$, defined as follows:

$$\begin{aligned}\mathcal{V}_i^{\text{basic}}(t) = \{x : \exists u_i(\cdot) \in \mathbb{U}_t^{t_i^{\text{STA}}}, &\forall s \in [t, t_i^{\text{STA}}], \xi(s; x, t, u_i(\cdot)) \notin \mathcal{G}_i(s), \\ &\exists s \in [t, t_i^{\text{STA}}], \xi(s; x, t, u_i(\cdot)) \in \mathcal{L}_i\}.\end{aligned} \tag{4.6}$$

Here, $\xi(\cdot)$ represent the system trajectory corresponding to the dynamics in (4.3). The BRS $\mathcal{V}_i^{\text{basic}}(t)$ can be obtained by solving the following HJI VI

$$\max\{\min\{D_t V(t,x) + H(t,x,V(t,x)),\ l_i(x) - V(t,x)\},\ -g(t,x) - V(t,x)\} = 0\ \forall x, t \tag{4.7}$$

$$V(t_i^{\text{STA}}, x) = \max\left\{l_i(x), -g(t_i^{\text{STA}}, x)\right\}, \tag{4.8}$$

where $\mathcal{L}_i$ and $\mathcal{G}_i(t)$ are given by the subzero level sets of functions $l_i(x)$ and $g(t,x)$ respectively. The Hamiltonian is given by

$$H(t,x,V(t,x)) = \inf_{u_i \in \mathcal{U}_i} \langle D_x V(t,x), f_i(x,u_i)\rangle \tag{4.9}$$

Given the value function $V(t,x)$, the BRT $\mathcal{V}_i^{\text{basic}}(t)$ can be obtained as a subzero level set of the value function. Note that $\mathcal{V}_i^{\text{basic}}(t)$, by definition, does not contain any states from which it is inevitable to avoid the danger zone $\mathcal{Z}_{ij}$ (and $\mathcal{G}_i$ in general). Given $\mathcal{V}_i^{\text{basic}}(t)$, the optimal control for reaching $\mathcal{L}_i$ while avoiding $\mathcal{G}_i(t)$ is then given by

$$u_i^{\text{basic}}(t,x) = \arg\inf_{u_i \in \mathcal{U}_i} \langle D_x V(t,x), f_i(x,u_i)\rangle \tag{4.10}$$

from which the trajectory $x_i(\cdot)$ can be computed by integrating the system dynamics, which in this case are given by (4.3). In addition, the latest departure time t_i^{LDT} can be obtained from the BRT $\mathcal{V}_i^{\text{basic}}(t)$ as $t_i^{\text{LDT}} = \arg\sup_t\{x_i^0 \in \mathcal{V}_i^{\text{basic}}(t)\}$. The basic STP algorithm is summarized in Algorithm 2.

Note that Step 1, which determines the total obstacle set, can be updated recursively by adding a new set of induced obstacles for each next vehicle: $\mathcal{G}_{i+1}(t) = \mathcal{G}_i(t) \cup \mathcal{O}_{i+1}^i(t)$. As previously mentioned, the basic STP algorithm, as well as all subsequent variants of STP algorithms, will *always* return a feasible trajectory that arrives at the target on time, as long as a feasible trajectory exists in the *absence* of other vehicles. This is because a vehicle can simply depart early enough to avoid being blocked by higher-priority vehicles. In fact, the latest departure time t_i^{LDT} quantifies exactly when each vehicle needs to depart to arrive on time.

Algorithm 2: STP algorithm in the absence of disturbances and intruders.

```
input : STP vehicles $Q_i$, their dynamics (4.3), initial states $x_i^0$, destinations $\mathcal{L}_i$,
        static obstacles $\mathcal{O}_i^{\text{static}}$
output: Provably safe trajectories to destinations, goal-satisfaction controllers
        $u_i^{\text{basic}}(\cdot)$, and the latest departure time $t_i^{\text{LDT}}$
1  for $i = 1 : N$ do
2      Trajectory planning for $Q_i$
3      compute the total obstacle set $\mathcal{G}_i(t)$ given by (4.4). If $i = 1$, $\mathcal{G}_i(t) = \mathcal{O}_i^{\text{static}}\ \forall t$;
4      compute the BRT $\mathcal{V}_i^{\text{basic}}(t)$ defined in (4.6);
5      Latest departure time for $Q_i$
6      the latest departure time $t_i^{\text{LDT}}$ is given by $\arg\sup_t \{x_i^0 \in \mathcal{V}_i^{\text{basic}}(t)\}$;
7      Trajectory and controller of $Q_i$
8      compute the optimal controller $u_i^{\text{basic}}(\cdot)$ given by (4.10);
9      determine the trajectory $x_i(\cdot)$ using vehicle dynamics (4.3) and the control
        $u_i^{\text{basic}}(\cdot)$;
10     output the trajectory and optimal controller for $Q_i$.
11     Obstacles induced by $Q_i$
12     given the trajectory $x_i(\cdot)$, compute the induced obstacles $\mathcal{O}_k^i(t)$ given by (4.5) for
        all $k > i$.
```

4.3.2 Sequential Trajectory Planning With Disturbances: Robust Trajectory Tracking (RTT)

Disturbances and incomplete information significantly complicate STP. The main difference is that the vehicle dynamics satisfy (4.1) as opposed to (4.3). Committing to exact trajectories is therefore no longer possible, since the disturbance $d_i(\cdot)$ is *a priori* unknown. Thus, the induced obstacles $\mathcal{O}_i^j(t)$ are no longer just the danger zones centered around positions.

Even though it is impossible to commit to and track an exact trajectory in the presence of disturbances, it may still be possible to *robustly* track a feasible *nominal* trajectory with a bounded error at all times. The tracking error bound can be used to determine the induced obstacles. Here, computation is done in two phases: the *planning phase* and the *disturbance rejection phase.* In the planning phase, we compute a nominal trajectory $x_{r,j}(\cdot)$ that is feasible in the absence of disturbances. In the disturbance rejection phase, we compute a bound on the maximum deviation from the reference trajectory due to the disturbances (also referred to as a *bound on the tracking error* here on). Thus, instead of directly accounting for disturbances when planning trajectories, one first augments the time-varying obstacle by this bound to obtain the augmented obstacles, which ensures that Q_i will not collide with the obstacles *in the presence of disturbances.* One then reduces the size of $\mathcal{L}_i$ by the same amount to obtain $\tilde{\mathcal{L}}_i$, which ensures that the vehicle safely reaches its goal despite any trajectory tracking errors resulting from disturbances.

It is important to note that the planning phase does not make full use of a vehicle's control authority, as some margin is needed to reject unexpected disturbances while tracking the nominal trajectory. Therefore, in this method, planning is done for a reduced control set $\mathcal{U}_i^p \subset \mathcal{U}_i$. The resulting trajectory reference will not utilize the vehicle's full control capability; additional maneuverability is available at execution time to counteract external disturbances.

In the disturbance rejection phase, we determine the error bound independently of the nominal trajectory by finding a robust controlled-invariant set in the joint state space of the vehicle and a tracking reference that may "maneuver" arbitrarily in the presence of an unknown bounded disturbance. Taking a worst-case approach, the tracking reference can be viewed as a virtual evader vehicle that is optimally avoiding the actual vehicle to enlarge the tracking error. We therefore can model trajectory tracking as a pursuit-evasion game in which the actual vehicle is playing against the coordinated worst-case action of the virtual vehicle and the disturbance.

Disturbance rejection phase: Computation of tracking error bound. Let x_j and $x_{r,j}$ denote the states of the actual vehicle Q_j and the virtual evader, respectively. Following [278], we define the relative state between x_j and $x_{r,j}$, which puts the virtual evader at the origin: $e_j = \phi(x_j, x_{r,j})(x_j - x_{r,j})$. For many dynamical systems, differentiating e_j with respect to time and choosing the appropriate $\phi(\cdot,\cdot)$ leads to the following relative dynamics, which is defined in the same dimensional space as the state space of the vehicle:

$$\begin{aligned} &\dot{e_j} = f_{e_j}(e_j, u_j, u_{r,j}, d_j), \\ &u_j \in \mathcal{U}_j, u_{r,j} \in \mathcal{U}_j^p, d_j \in \mathcal{D}_j, \quad t \leq 0 \end{aligned} \tag{4.11}$$

In the case that both vehicles have dynamics of the form in (4.24), $\phi(x_j, x_{r,j}) = R(-\theta_j)$. In general, reference [278] presents examples of coordinate transforms for obtaining relative dynamics between pairs of systems.

To obtain bounds on the tracking error, we first conservatively estimate the error bound around any reference state $x_{r,j}$, denoted $\mathcal{E}_j$:

$$\mathcal{E}_j = \{e_j : \|p_{e_j}\|_2 \leq R_{\text{EB}}\}, \tag{4.12}$$

where p_{e_j} denotes the position coordinates of e_j and R_{EB} is a design parameter. We next compute the viability kernel of the set $\mathcal{E}_j$ – the set of relative states from which the tracking error bound is not violated at any time. This set is formally defined as:

$$\begin{aligned} \mathcal{K}_j^{\text{EB}}(t) = &\{e_j : \forall d_j(\cdot) \in \Gamma, \forall u_{r,j}(\cdot) \in \mathbb{U}_j^p, \exists u_j(\cdot) \in \mathbb{U}_j, \\ &\forall s \in [t, 0], \xi(s; e_j, t, u_j(\cdot), u_{r,j}(\cdot), d_j(\cdot)) \in \mathcal{E}_j\}, \end{aligned} \tag{4.13}$$

Here, $\xi(\cdot)$ represent the system trajectory corresponding to the dynamics in (4.11). We also set the time horizon $T = 0$ without loss of generality. The viability kernel in (4.13) can be computed by the superzero level set of the value function obtained by solving the following HJI VI:

$$\min\{D_t V(t,x) + H(t,x,V(t,x)), \ -l'_j(x) - V(t,x)\} = 0 \ \forall x,t \qquad V(0,x) = -l'_j(x), \quad (4.14)$$

where $\mathcal{E}_j$ is given by the subzero level set of l'_j. The Hamiltonian to compute the value function is given by:

$$H(t,x,V(t,x)) = \sup_{u_j \in \mathcal{U}_j} \inf_{u_{r,j} \in \mathcal{U}^p_j, d_j \in \mathcal{D}_j} \langle D_x V(t,x), f_{e_j}(x,u_j,u_{r,j},d_j)\rangle. \quad (4.15)$$

Finally, the viability kernel is given by:

$$\mathcal{K}^{\text{EB}}_j(t) = \{x : V(t,x) > 0\}. \quad (4.16)$$

For more details on the computation of the viability kernel, please refer to Section 2.3.2.

Letting $t \to -\infty$, we obtain the infinite-horizon control-invariant set $\Omega_j := \lim_{t\to-\infty} \mathcal{K}^{\text{EB}}_j(t)$. If Ω_j is nonempty, then the tracking error e_j at flight time is guaranteed to remain within $\Omega_j \subseteq \mathcal{E}_j$ provided that the vehicle starts inside Ω_j and subsequently applies the feedback control law

$$u_j(x) = \arg \sup_{u_j \in \mathcal{U}_j} \inf_{u_{r,j} \in \mathcal{U}^p_j, d_j \in \mathcal{D}_j} \langle D_x V(-\infty,x), f_{e_j}(x,u_j,u_{r,j},d_j)\rangle. \quad (4.17)$$

Planning phase: Computation of nominal/reference trajectory. Given the tracking error bound, the induced obstacles by each higher-priority vehicle Q_j can be obtained by

$$\begin{aligned} \mathcal{O}^j_i(t) &= \{x_i : \exists y \in \mathcal{P}_j(t), \|p_i - y\|_2 \le R_c\} \\ \mathcal{P}_j(t) &= \{p_j : \exists h_j, (p_j,h_j) \in \mathcal{M}_j(t)\} \\ \mathcal{M}_j(t) &= \Omega_j + x_{r,j}(t), \end{aligned} \quad (4.18)$$

where the "+" in (4.18) denotes the Minkowski sum[2]. Intuitively, if Q_j is tracking $x_{r,j}(t)$, then it will remain within the error bound Ω_j around $x_{r,j}(t)$ $\forall t$. This set is mathematically given by the "tube" obtained by augmenting the error bound Ω_j around the reference trajectory $x_{r,j}(\cdot)$. This is precisely the set $\mathcal{M}_j(t)$ (and $\mathcal{P}_j(t)$ for the position states). The induced obstacles can then be obtained by augmenting a danger zone around this set. Finally, we can obtain the total obstacle set $\mathcal{G}_i(t)$ using (4.4).

Since each vehicle Q_j, $j < i$, can only be guaranteed to stay within Ω_j, we must make sure during the trajectory planning of Q_i that at any given time, the error bounds of Q_i and Q_j, Ω_i and Ω_j, do not intersect. This can be done by augmenting the total obstacle set by Ω_i:

$$\tilde{\mathcal{G}}_i(t) = \mathcal{G}_i(t) + \Omega_i. \quad (4.19)$$

[2]The Minkowski sum of sets A and B is the set of all points that are the sum of any point in A and B.

Finally, given Ω_i, we can guarantee that Q_i will reach its target $\mathcal{L}_i$ if $\Omega_i \subseteq \mathcal{L}_i$. Thus, in the trajectory planning phase, we modify $\mathcal{L}_i$ to be $\tilde{\mathcal{L}}_i := \{x_i : \Omega_i + x_i \subseteq \mathcal{L}_i\}$, and compute a BRT, with the control authority $\mathcal{U}_i^p$, that contains the initial state of the vehicle. Mathematically,

$$\begin{aligned}\mathcal{V}_i^{\text{rtt}}(t) = \{x : \exists u_i(\cdot) \in \mathbb{U}_i^p, \forall s \in [t, t_i^{\text{STA}}], \xi(s; x, t, u_i(\cdot)) \notin \tilde{\mathcal{G}}_i(s), \\ \exists s \in [t, t_i^{\text{STA}}], \xi(s; x, t, u_i(\cdot)) \in \tilde{\mathcal{L}}_i\}.\end{aligned} \tag{4.20}$$

Here, $\xi(\cdot)$ represent the system trajectory corresponding to the disturbance-free dynamics in (4.3), except the control authority is changed to $\mathcal{U}_i^p$. The BRT $\mathcal{V}_i^{\text{rtt}}(t)$ can be computed in an analogous fashion to $\mathcal{V}_i^{\text{basic}}(t)$ using the HJI VI in (4.7). The corresponding optimal control for reaching $\tilde{\mathcal{L}}_i$ is

$$u_i^{\text{rtt}}(t, x) = \arg \inf_{u_i \in \mathcal{U}_i^p} \langle D_x V(t, x), f_i(x, u_i) \rangle. \tag{4.21}$$

The nominal trajectory $x_{r,i}(\cdot)$ can thus be obtained by using vehicle dynamics (4.3), with the optimal control $u_i^{\text{rtt}}(\cdot)$ given by (4.21). From the resulting nominal trajectory $x_{r,i}(\cdot)$, the overall control policy to reach $\mathcal{L}_i$ can be obtained via (4.17). The robust trajectory tracking method-based STP is summarized in Algorithm 3.

A note on the computation of tracking error bound. In this work, the reduced control authority $\mathcal{U}^p$ and the parameter R_{EB} are selected by trial and error such that the the tracking error bound Ω_j is nonempty. However, the value function obtained by solving the HJI VI in (4.14) can be used to compute the tracking error bound even when the Ω_j is empty, provided that the value function converges. When the value function converges, an empty Ω_j will often mean that the radius R_{EB} was not chosen conservatively enough to have a non-empty positive level. Since the value function is invariant to additive constants, one can simply find any nonempty level of the value function (say $\alpha < 0$) and then increase the value of R_{EB} to $R_{\text{EB}} + |\alpha|$. The super-α set of the value function corresponding to radius R_{EB} now becomes a non-empty super-zero level set corresponding to radius $R_{\text{EB}} + |\alpha|$, and can serve as tracking error bound. This procedure also highlights a mechanism to choose the smallest possible tracking error bound. In particular, one can pick $\alpha = \max_x V(-\infty, x)$ to obtain the smallest error bound Ω_j.

Finally, it might be useful to use simple shapes like circles, sphere to bound the tracking error for computational reasons during the planning phase. For example, it is much easier to do an obstacle augmentation by a fixed radius. Thus, one may want to project Ω_j in the position space and find a circle that circumscribe the projected set. The radius of this circle (sphere in 3D) is given by

$$R_{\text{EB}}^* = \max_{x \in \Omega_j} \|p(x)\|, \tag{4.22}$$

where $p(x)$ is the position component of the state x.

Algorithm 3: Robust trajectory tracking algorithm for STP in the presence of disturbances but in the absence of intruders.

```
input  : STP vehicles $Q_i$, their dynamics (4.1), initial states $x_i^0$, destinations $\mathcal{L}_i$,
         static obstacles $\mathcal{O}_i^{\text{static}}$
output: Provably safe trajectories to destinations, goal-satisfaction controllers
         $u_i^{\text{rtt}}(\cdot)$, and the latest departure time $t_i^{\text{LDT}}$.
1  for $i = 1 : N$ do
2      Tracking error bound computation for $Q_i$
3      decide on a reduced control authority $\mathcal{U}_i^p$ for the planning phase, and choose a
         parameter $R_{\text{EB}}$ to conservatively bound the tracking error;
4      compute the viability kernel $\mathcal{K}_i^{\text{EB}}(t)$ using (4.13) and compute the infinite
         horizon control invariant set $\Omega_i := \lim_{t \to -\infty} \mathcal{K}_i^{\text{EB}}(t)$;
5      the error bound on the tracking error is given by $\Omega_i$;
6      Trajectory planning for $Q_i$
7      compute the total obstacle set $\mathcal{G}_i(t)$ given by (4.4). If $i = 1$, $\mathcal{G}_i(t) = \mathcal{O}_i^{\text{static}}\ \forall t$;
8      using $\Omega_i$, determine the augmented obstacle set $\tilde{\mathcal{G}}_i(t)$, given in (4.19);
9      compute the BRS $\mathcal{V}_i^{\text{rtt}}(t)$ defined in (4.20);
10     Latest departure time for $Q_i$
11     the latest departure time $t_i^{\text{LDT}}$ is given by $\arg\sup_t \{x_i^0 \in \mathcal{V}_i^{\text{rtt}}(t)\}$;
12     Trajectory and controller of $Q_i$
13     compute the optimal controller $u_i^{\text{rtt}}(\cdot)$ given by (4.21);
14     compute the nominal trajectory $x_{r,i}(\cdot)$ using vehicle dynamics in (4.3) and
         optimal control given in (4.21);
15     Obstacles induced by $Q_i$
16     compute the induced obstacles $\mathcal{O}_k^i(t)$ via (4.18) for all $k > i$.
```

4.3.3 Large-Scale Multiple UAV Simulations: City Environment

We now combine BEACLS (Sec. 3.4) with the STP algorithm for the safe trajectory planning for a 50-UAV system in which the vehicles are flying over the city of San Francisco. This setup can be representative of many UAV applications, such as package delivery, aerial surveillance, etc. Using this simulation, we investigate the resulting trajectories of vehicles as a function of the amount of traffic and wind speed. Videos of the city environment simulation with various vehicle densities and wind speeds can be found on YouTube[3].

Simulation setup. We grid the City of San Francisco (SF) in California, USA, and use it as our position space, as shown in Fig. 4.1. Each box in Fig. 4.1 represents a 1 km^2 area of SF. The origin point for the vehicles is denoted by the blue star. This origin point may represent an exit of an air highway connecting SF to other cities in the Bay Area [76]. In general there

[3] Video link: https://youtu.be/1ocaBGZqSAE

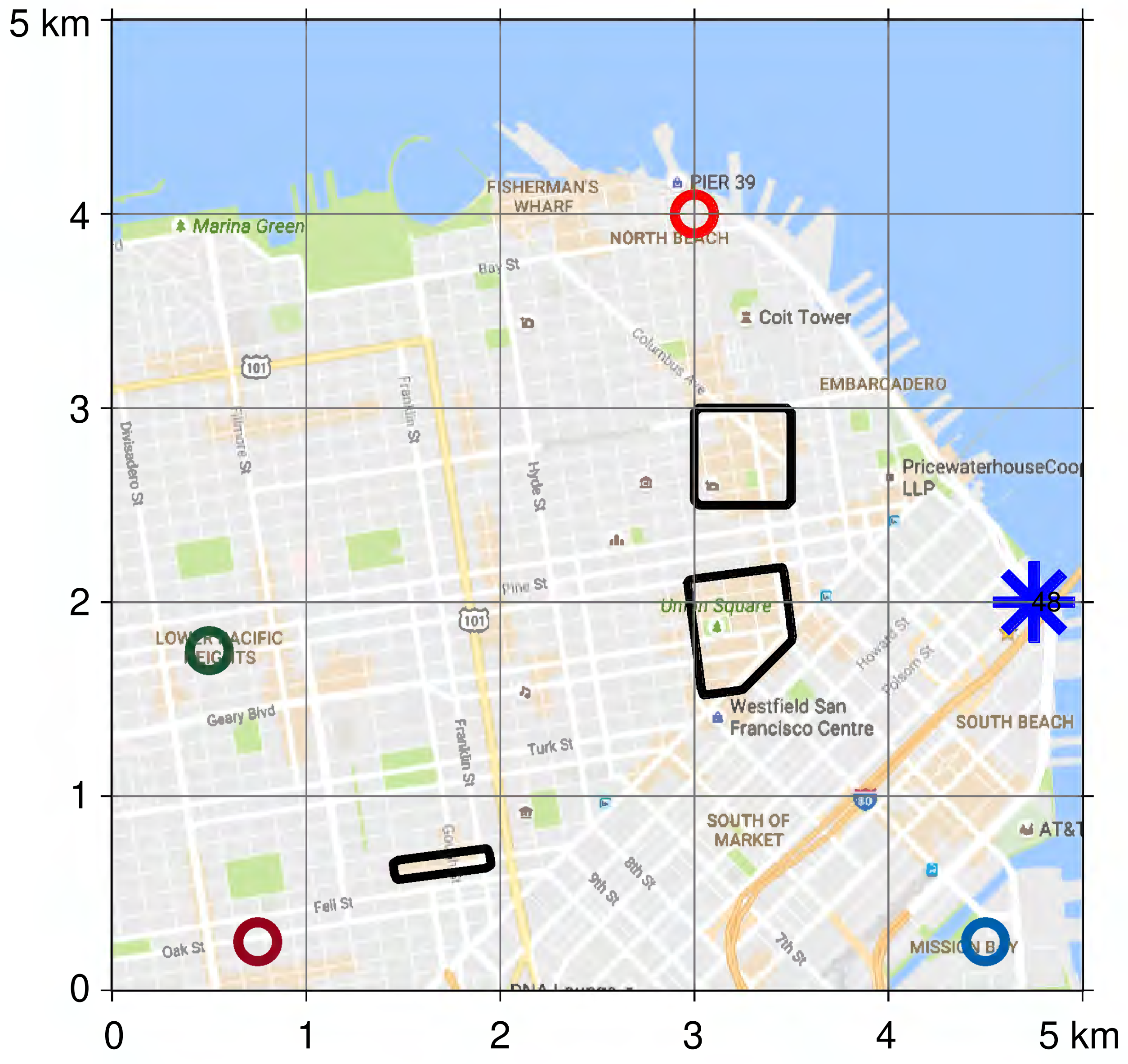


Figure 4.1: City environment multiple UAV simulation setup. A 25 km^2 area in the City of San Francisco is used as the space for the 50-vehicle simulation. Vehicles originate from the blue star and go to one of the four destinations, denoted by circles. Tall buildings in the downtown area are used as static obstacles, represented by the black contours.

may be multiple origin points; we will demonstrate this other case in Section 4.3.4. Four different areas in the city are chosen as the destinations for the vehicles. Mathematically, the target sets $\mathcal{L}_i$ of the vehicles are circles of radius r in the position space, i.e. each vehicle is trying to reach some desired set of positions. In terms of the state space x_i, the target sets are defined as

$$\mathcal{L}_i = \{x_i : \|p_i - c_i\|_2 \leq r\} \tag{4.23}$$

where c_i are centers of the target circles. In this simulation, we use $r = 100$ meters. The four targets are represented by four circles in Fig. 4.1. The destination of each vehicle is chosen randomly from these four destinations. Finally, some areas in downtown SF and the city hall are used as representative static obstacles for the STP vehicles, denoted by black contours in Fig. 4.1.

For this simulation, we use the following Dubins car dynamics for each vehicle:

$$\begin{aligned}
\dot{p}_{x,i} &= v_i \cos\theta_i + d_{x,i} \\
\dot{p}_{y,i} &= v_i \sin\theta_i + d_{y,i} \\
\dot{\theta}_i &= \omega_i, \\
\underline{v} &\leq v_i \leq \bar{v}, \ |\omega_i| \leq \bar{\omega}, \\
\|(d_{x,i}&, d_{y,i})\|_2 \leq d_r
\end{aligned} \tag{4.24}$$

where $x_i = (p_{x,i}, p_{y,i}, \theta_i)$ is the state of vehicle Q_i, $p_i = (p_{x,i}, p_{y,i})$ is the position, θ_i is the heading, and $d = (d_{x,i}, d_{y,i})$ represents Q_i's disturbances, for example wind, that affect its position evolution. The control of Q_i is $u_i = (v_i, \omega_i)$, where v_i is the speed of Q_i and ω_i is the turn rate; both controls have a lower and upper bound. To make our simulations as close as possible to real scenarios, we choose velocity and turn-rate bounds as $\underline{v} = 0$ m/s, $\bar{v} = 25$ m/s, $\bar{\omega} = 2$ rad/s, aligned with the modern UAV specifications [1, 232].

For planning, we choose the reduced control authority to be $\mathcal{U}_j^p = \{(v_{r,j}, \omega_{r,j}) : 11 \text{ m/s} \leq v_{r,j} \leq 13 \text{ m/s}, |\omega_{r,j}| \leq 1.2 \text{ rad/s}\}$. Given this reduced control authority, we obtain a tracking error bound as well as a disturbance rejection controller using the robust trajectory tracking method (see Section 4.3.2). In the case that both vehicles have dynamics of the form in (4.24), $\phi(x_j, x_{r,j}) = R(-\theta_j)$.

The disturbance bounds are chosen to be either $d_r = 6$ m/s or $d_r = 11$ m/s. These conditions correspond to *moderate breeze* and *strong breeze* respectively on the Beaufort scale [303]. In our simulations, we use random wind speeds (within above bounds) and directions which can constantly and discontinuously change. We also repeated the simulation using the worst-case disturbance. The results are qualitatively the same; here, we only present the results for random disturbances.

The scheduled times of arrival for all vehicles are chosen to be same for all vehicles (0 without loss of generality) for a high UAV density condition. For medium and low density conditions, we separated the arrival times by 5 seconds and 10 seconds respectively, with the latest t_i^{STA} being 0. Note that we have used same dynamics and input bounds across all vehicles for clarity of illustration; however, STP can easily handle more general systems of the form in which the vehicles have different control bounds, t_i^{STA} and dynamics.

The goal of the vehicles is to reach their destinations while avoiding a collision with the other vehicles or the static obstacles. The joint state space of this 50-vehicle system is 150-dimensional (150D), making the joint trajectory planning and collision avoidance problem intractable for direct analysis. Therefore, we assign a priority ordering to vehicles and solve the trajectory planning problem sequentially. For this simulation, we assign a random priority order to fifty vehicles.

High UAV Density with 6 m/s Wind. We start with Q_1 and sequentially compute the optimal control policy and the latest departure time t_j^{LDT} for each vehicle. In presence of moderate winds, the obtained tracking error bound is 5 meters. This means that given any trajectory (which is a sequence of states over time) of vehicle, winds can at most cause a deviation of 5 meters from this trajectory at all times. Consequently, the vehicle will be within a distance of 5 meters from the planned trajectory. Note that since all vehicles have same dynamics, the error bound is also same for all vehicles. This error bound is used to obtain the augmented obstacles and the reduced target set $\tilde{\mathcal{L}}_i$. Note that since disturbance directly impacts the computation of tracking error bound, in general the size of augmented obstacles increases as disturbance magnitude increases. We will illustrate the effect of disturbance magnitude on the trajectories of vehicles later in this Section.

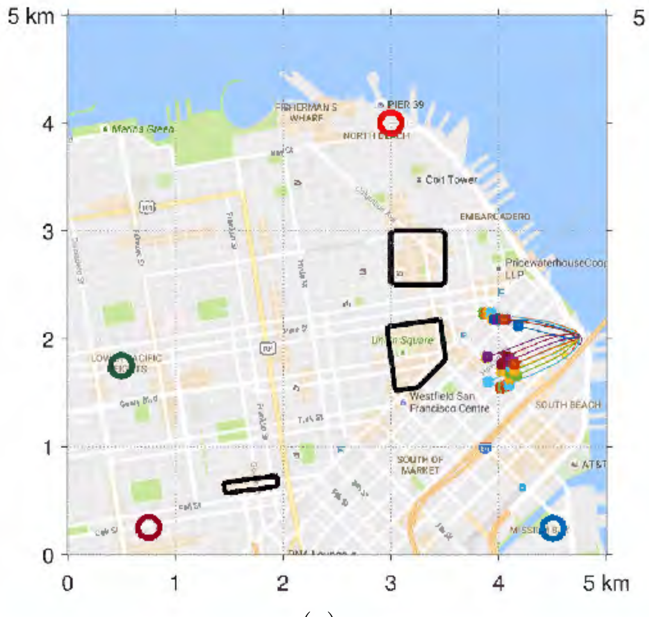
The nominal trajectory can now be obtained using a reduced control authority starting from the initial state x_j^0. The resulting trajectories of vehicles for $d_r = 6$ m/s and $t_j^{\text{STA}} = 0 \; \forall j$ at different times are shown in Fig. 4.2. As is evident from the figures, the vehicles remain clear of all the static obstacles (the black contours) and make progress towards reaching their destinations, according to their planned trajectories. The vehicles whose destinations are relatively close need less time to travel to their destinations and thus they depart later.

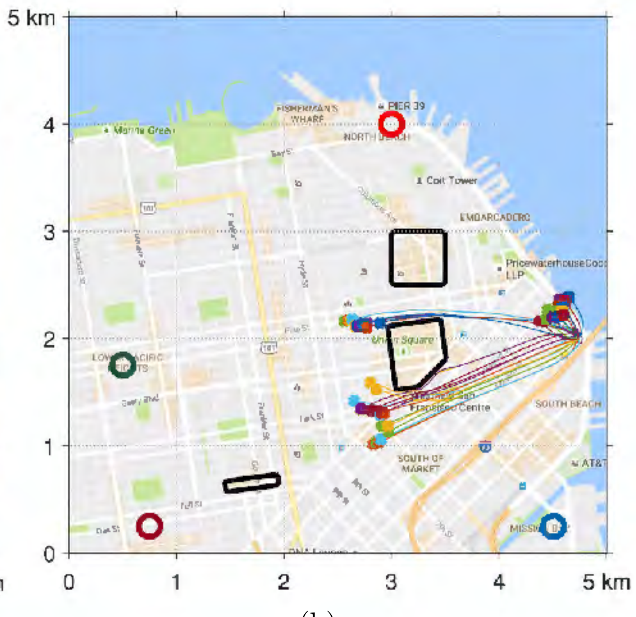
The full trajectories of vehicles from their departure to arrival are shown in Fig. 4.3a. All vehicles reach their respective destinations. A zoomed-in version of Fig. 4.3a near the red target (Fig. 4.4) illustrates that vehicles are also outside each other's danger zones (circles around the vehicles) as required.

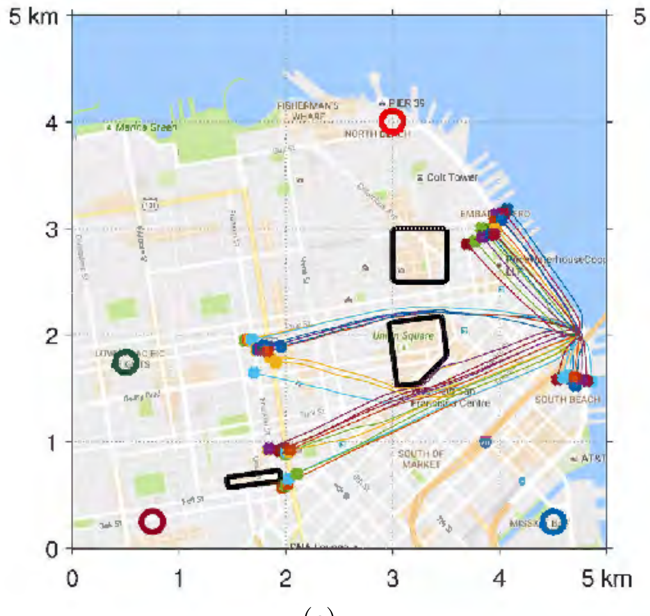
It is interesting to note that the vehicles going to the same destination take different trajectories. This is because all vehicles have the same scheduled time of arrival, and hence the lower-priority vehicles do not have the flexibility to wait for the higher-priority vehicles. In order to ensure that they reach their destinations on time, they must depart earlier and take alternative trajectories to their destinations, forming different "traffic lanes". Thus, the vehicles' trajectories are *state-separated* trajectories, i.e., they follow different state trajectories but at the same time.

The average trajectory computation time per vehicle is 2 seconds using the CUDA implementation of the level set method in BEACLS. Computations were run on a desktop computer with a Core i7 5820K processor and two GeForce GTX Titan X graphics processing units. Computations were done on a $101 \times 101 \times 15$ grid, which corresponds to a spatial resolution of 50 meters and an angular resolution of 24 degrees over a 5 kilometer $\times$ 5 kilometer area over the city.

Recall that all of the computation is done offline and the resulting BRS and corresponding


(a)


(b)


(c)

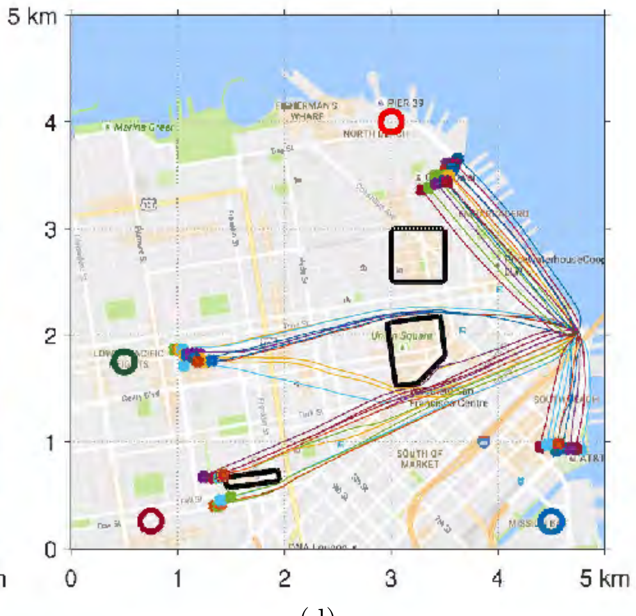
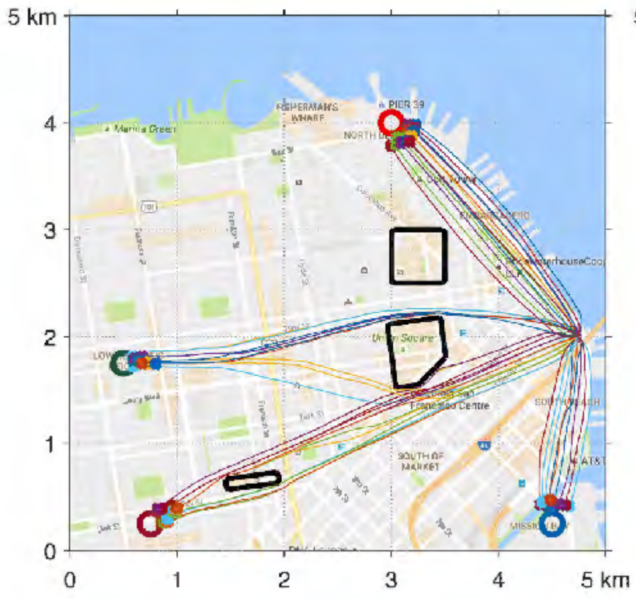
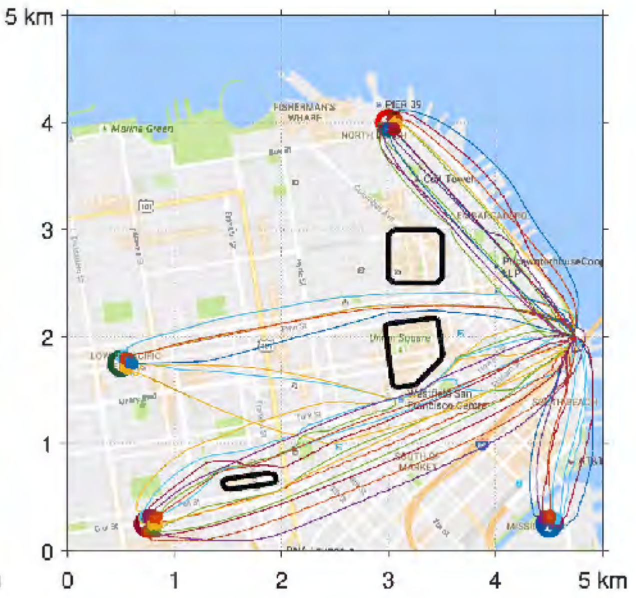
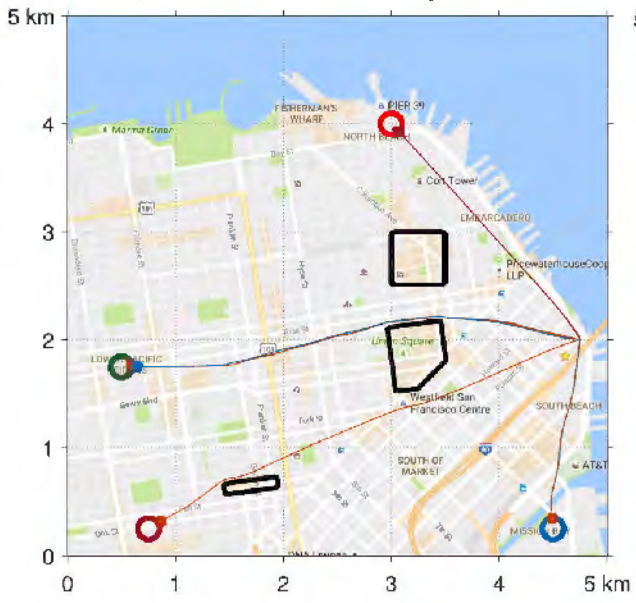

(d)

Figure 4.2: Snapshots of vehicle trajectories at approximately a) 1 minute, b) 3 minutes, c) 4 minutes, and d) 5 minutes after the first vehicle departs. The wind speed is uniformly random with a bound of $d_r = 6$ m/s. The vehicles remain clear of all static obstacles and of each other despite the disturbance in the dynamics.

control policy are obtained as lookup tables. In real time, computation and communication between vehicles is not required. Only a lookup table query is required, and this can be


(a) Case 0: $d_r = 6m/s$, $t_i^{\text{STA}} = 0$


(b) Case 1: $d_r = 11m/s$, $t_i^{\text{STA}} = 0$


(c) Case 2: $d_r = 6m/s$, $t_i^{\text{STA}} = 5(i-1)$

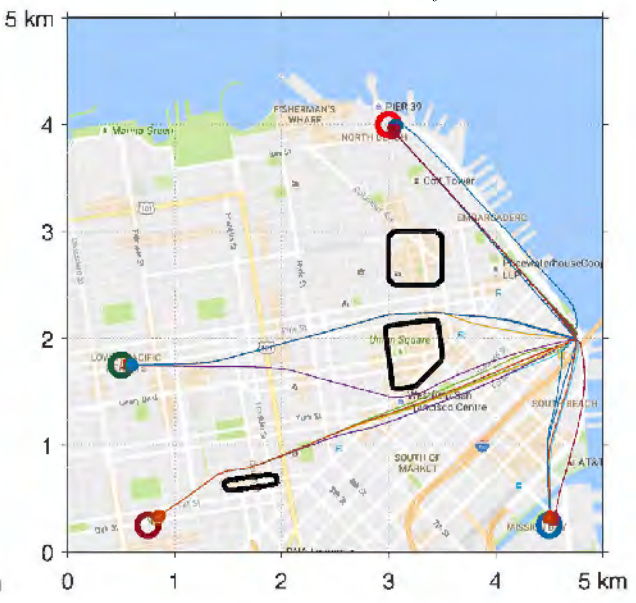

(d) Case 3: $d_r = 11m/s$, $t_i^{\text{STA}} = 5(i-1)$

Figure 4.3: Effect of the disturbance magnitude and the scheduled times of arrival on vehicle trajectories. All trajectories are simulated under uniformly random disturbance. The relative separation in the scheduled times of arrival of vehicles determines the number of lanes between a pair of origin and destination, and more and more trajectories become *time-separated* as this relative separation increases. The disturbance magnitude determines the relative separation between different lanes, and more and more trajectories become *state-separated* as the disturbance increases.

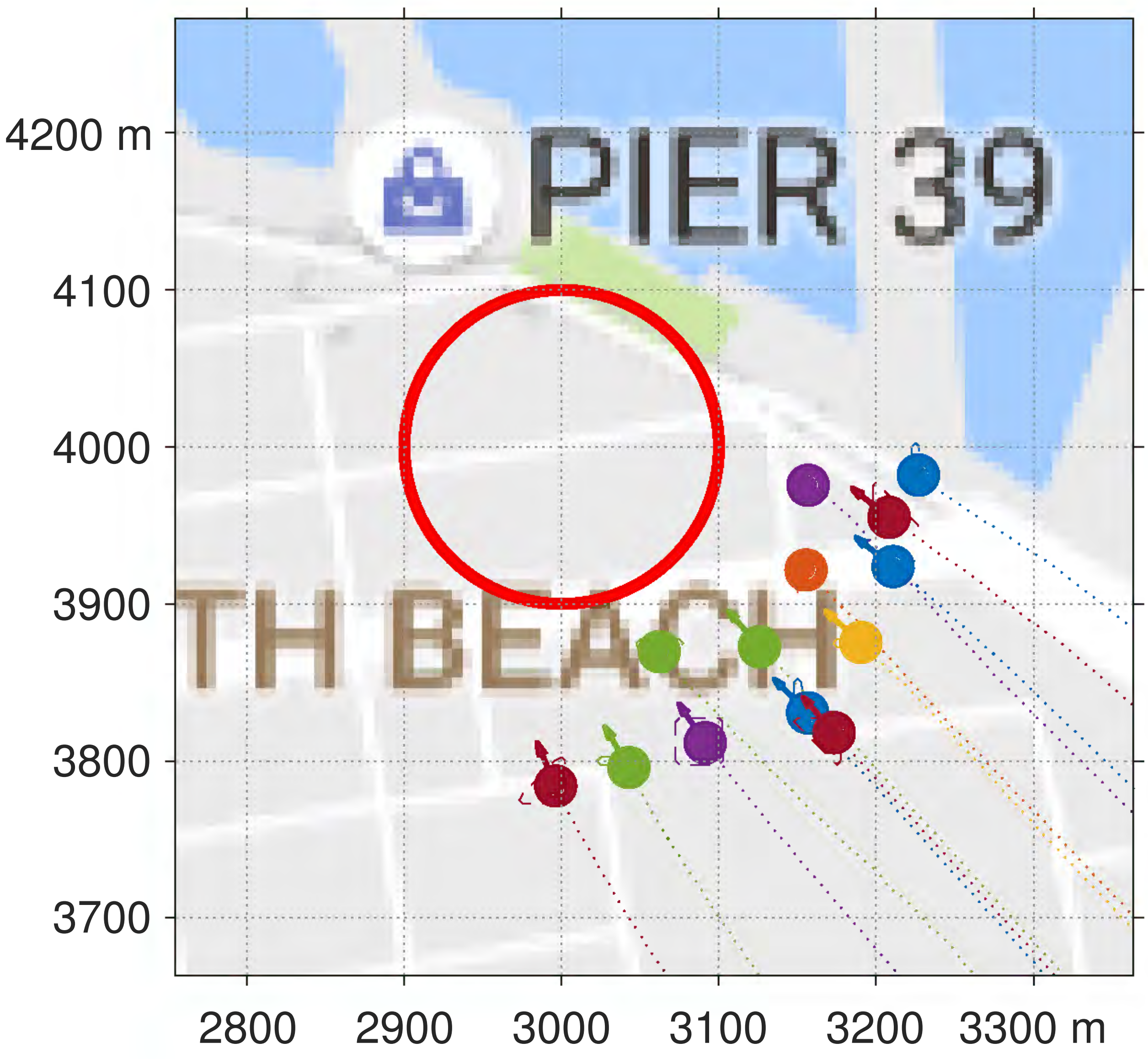


Figure 4.4: Zoomed-in version of vehicle trajectories near the red target in Fig. 4.3a. The STP algorithm ensures that the vehicles are outside each other's danger zones, i.e., the centers of any two intersecting circles are not within the same circle.
Here the smallest distance between vehicles is just over 100 meters (blue and red vehicles below the letter "H").

performed very quickly in real time. This illustrates the capability of STP as a provably safe trajectory planning algorithm for large multi-vehicle systems.

Without the CUDA implementation in BEACLS, the approximate computation time per vehicle is 33 seconds using the C++ implementation of the level set toolbox without CUDA, and 200 seconds using the level set toolbox and helperOC in MATLAB. A comparison of the *total* computation time for all 50 vehicles is shown in Figure 3.15. Trajectory planning with 50 vehicles takes less than 2 minutes, compared to 27 minutes for a non-CUDA C++ implementation or 2.8 hours for the MATLAB implementation. This improvement in computation time greatly facilitates case studies such as city-level airspace design presented in this section, which more generally may involve testing different initial and goal positions at different vehicle densities and wind speeds.

For even larger-scale case studies, a GPU-parallelized implementation such as BEACLS may be necessary to keep the computation tractable. For example, in the study presented in Section 4.3.4 involving multiple cities and 200 vehicles, computation for each vehicle was approximately 4 minutes on average using two GPUs. This is because a much finer grid was needed to maintain positional accuracy over a much larger geographical area. The entire simulation took approximately 13.3 hours. Extrapolating the computation time comparison in Fig. 3.15, even a non-CUDA C++ implementation would take prohibitively long – approximately 10 days.

Effects of Disturbance and Scheduled Time of Arrival. We now illustrate how the disturbance bound d_r in (4.24) and the relative t^{STA}'s of vehicles affect the vehicle trajectories. For this purpose, we simulate the STP algorithm in five scenarios:

- Case 0: $d_r = 6$ m/s, $t_i^{\text{STA}} = 0\ \forall i$ (moderate breeze, high UAV density; the simulation also shown in Figure 4.2)
- Case 1: $d_r = 11$ m/s, $t_i^{\text{STA}} = 0\ \forall i$ (strong breeze, high UAV density)
- Case 2: $d_r = 6$ m/s, $t_i^{\text{STA}} = 5(i-1)\ \forall i$ (moderate breeze, medium UAV density)
- Case 3: $d_r = 11$ m/s, $t_i^{\text{STA}} = 5(i-1)\ \forall i$ (strong breeze, medium UAV density)
- Case 4: $d_r = 11$ m/s, $t_i^{\text{STA}} = 10(i-1)\ \forall i$ (strong breeze, low UAV density)

The interpretation of $t_i^{\text{STA}} = 5(i-1)$ is that the scheduled time of arrival of any two consecutive vehicles is separated by 5 seconds, which represents a medium vehicle density scenario; a separation of 10 seconds represents a low vehicle density scenario. $d_r = 6$ m/s and $d_r = 11$ m/s correspond to the moderate breeze and strong breeze respectively on Beaufort wind force scale [303].

Intuitively, as d_r increases, it is harder for a vehicle to closely track a particular nominal trajectory, which results in a higher tracking error bound. As mentioned previously, with a 6 m/s wind speed, the tracking error bound is 5 meters; however, with an 11 m/s wind speed, the tracking error bound becomes 35 meters. Thus, the vehicles need to be separated more from each other in space, compared to with a 6 m/s wind speed, to ensure that they do not enter each other's danger zones. This is also evident from comparing the results corresponding to Case 0 (Fig. 4.3a) and Case 1 (Fig. 4.3b). As the disturbance magnitude increases from $d_r = 6$ m/s (moderate breeze) to $d_r = 11$ m/s (strong breeze), the vehicles' trajectories get farther apart from each other. Since t^{STA} is same for all vehicles, the vehicles' trajectories are still predominately *state-separated*.

We next compare Case 0 (Fig. 4.3a) and Case 2 (Fig. 4.3c). The difference between these two cases is that vehicles have a 5-second separation in their schedule times of arrival in Case 2. When vehicles Q_i and Q_j ($j > i$) have same scheduled time of arrival as in Case 0, and are going to the same destination, they are constrained to travel at the same time to make sure they reach the destination by the designated t^{STA}. However, since Q_i is high-priority, it is able to take an optimal trajectory (in terms of the total time of travel to destination) and Q_j has to settle for a relatively sub-optimal trajectory. Thus, all vehicles going to a particular destination take different trajectories, creating a "band" of trajectories between the origin and the destination, as shown in Fig. 4.3a; the high-priority vehicles take a relatively straight trajectory between the origin and the destination whereas the low-priority vehicles take a (relatively sub-optimal) curved trajectory. If we think of an air highway between the origin and the destination, then vehicles take different lanes of that highway to reach the destination in Case 0. Thus, the trajectories of vehicles in this case are *state-separated*. However, when $t_j^{\mathrm{STA}} > t_i^{\mathrm{STA}}$, then Q_j is not bound to travel at the same time as Q_i; it can wait for Q_i to depart and take a shorter trajectory later on. Thus, vehicles travel in a single lane in this case, as shown in Fig. 4.3c. In other words, they take the same trajectory to the destination, but at different times. Thus, the trajectories of vehicles in this case are *time-separated*.

Note that the exact number of lanes depends on *both* the disturbance (wind speed) and separation of scheduled times of arrival (vehicle density). As the disturbance increases, vehicles need to be separated more from each other to ensure safety. A larger arrival time difference between vehicles is also able to ensure this separation even if the vehicles were to take the same lane. As shown in Fig. 4.3d, a difference of 5 seconds in the t^{STA}'s is not sufficient to achieve a single lane behavior for stronger 11 m/s wind conditions. However, the number of lanes is significantly fewer than that in Case 1 (Fig. 4.3b). Finally, a separation of 10 seconds in t^{STA}'s ensure that we get the single lane behavior even in the presence of 11 m/s winds, leading to *time-separated* trajectories, as shown in Fig. 4.5. Videos of the simulations can be found on YouTube[4].

Given our observations about the simulations presented, one can conclude, more generally, that the relative magnitude of disturbance and scheduled times of arrival separation determines the number of lanes and type of trajectories that emerge out of the STP algorithm. For a fixed disturbance magnitude, as the separation in the scheduled times of arrival of vehicles increases, the number of lanes between a pair of origin and destination decreases, and more and more trajectories become *time-separated*. On the other hand, for a fixed separation in the scheduled times of arrival of vehicles, as the disturbance magnitude increases, the number of lanes between a pair of origin and destination increases, and more and more trajectories become *state-separated*.

[4] Video link: https://youtu.be/1ocaBGZqSAE

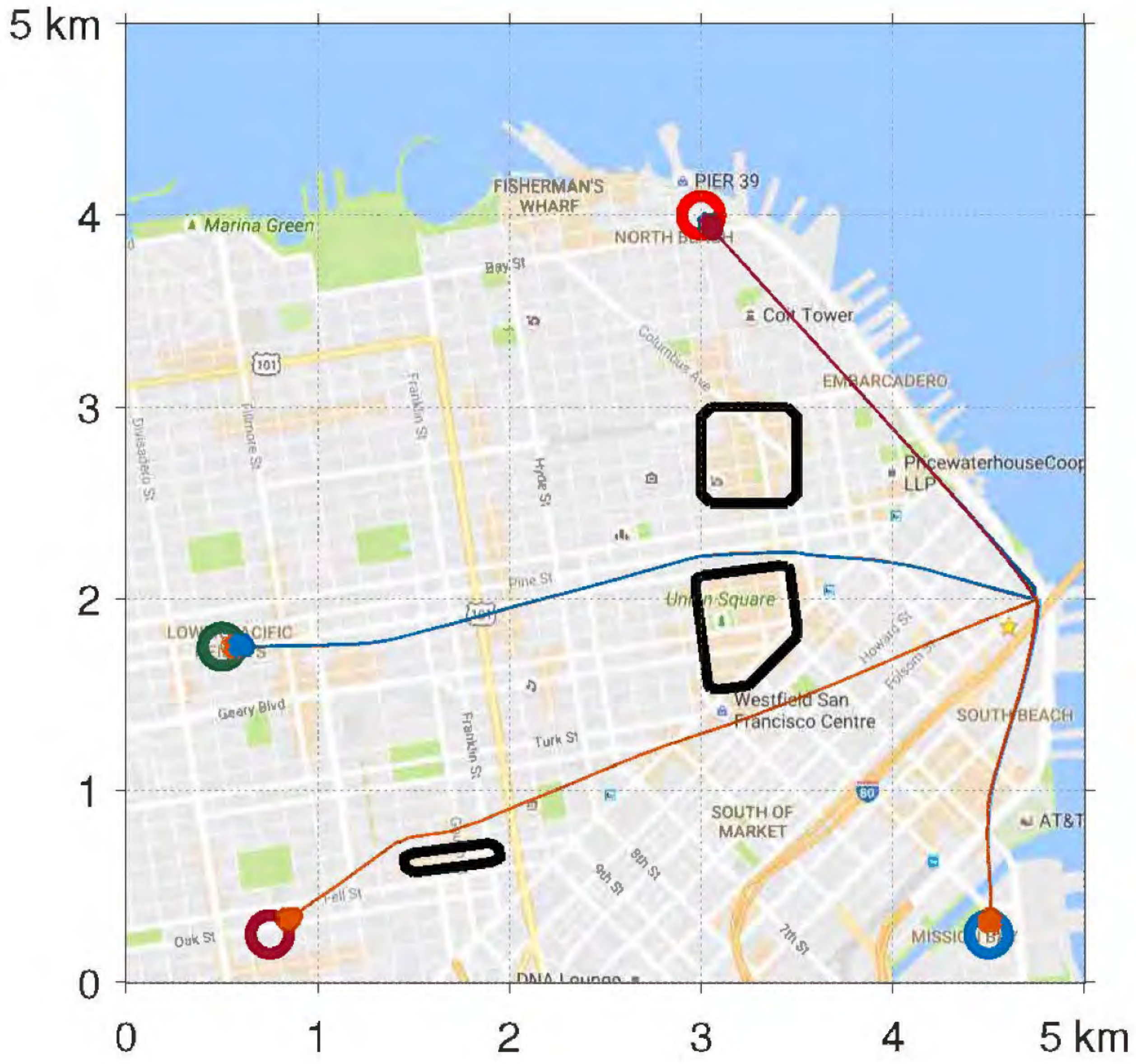


Figure 4.5: Trajectories of 50 vehicles for Case 4: $d_r = 11$ m/s, $t_i^{\text{STA}} = 10(i-1)$. Since different vehicles have different scheduled times of arrival, there is a single lane between every origin-destination pair.

4.3.4 Large-Scale Multiple UAV Simulations: Multi-City Environment

We next use STP to design trajectories for a 200-UAV system where UAVs are flying through a multi-city region.

Simulation setup. We use a part of the San Francisco Bay Area in California, USA as our position space, as shown in Fig. 4.6. We consider the UAVs flying to and from four cities: Richmond, Berkeley, Oakland, and San Francisco. The blue region in Fig. 4.6 represents bay (water). This environment is different from the city environment in the previous section in that here the UAVs need to fly for longer distances and through a high-density vehicle environment with strong winds, but have no static obstacles like tall buildings. Due to the much larger number of vehicles and longer trajectory time horizons, many more reachable sets need to be computed, and, even more crucially, each computation must be done on a much larger computational domain. Therefore, the use of GPU parallelization is essential for making this simulation possible. Each box in Fig. 4.6 represents a 25 km^2 area. The vehicles are flying to and from the four cities indicated by the four circles. The origin and destination of each vehicle is chosen randomly from these four cities. The vehicle dynamics are given by (4.24). We choose velocity and turn-rate bounds as $\underline{v} = 0$ m/s, $\bar{v} = 25$ m/s, $\bar{\omega} = 2$ rad/s. The disturbance bound is chosen as $d_r = 11$ m/s, which corresponds to "strong breeze" on Beaufort wind force scale [303]. The scheduled time of arrival t^{STA} for vehicles are chosen as $5(i - 1)$ seconds.

The goal of the vehicles is to reach their destinations while avoiding a collision with the other vehicles. The joint state space of this 200-vehicle system is 600-dimensional, making the joint trajectory planning and collision avoidance problem intractable for direct analysis. Therefore, we again use STP and assign a priority order to vehicles to solve the trajectory planning problem sequentially.

Results. The resulting trajectories of vehicles are shown in Fig. 4.7a. Once again, the vehicles remain clear of all other vehicles and reach their respective destinations. Since the vehicles' scheduled times of arrival are separated by 5 seconds, the trajectories are predominately *time-separated*, with roughly two lanes for each pair of cities (one for going from city A to city B and another for from city B to city A). A high density of vehicles is achieved in the center since the 4 trajectories are intersecting in the center (Richmond-Oakland, Oakland-Richmond, Berkeley-San Francisco, San Francisco-Berkeley), but the STP algorithm ensures safety despite this high-density, as shown in the zoomed-in version of the center at an intermediate time when a large number of vehicles are passing through the central region (Fig. 4.7b).

We also simulated the system for the case in which $t_i^{\text{STA}} = 0 \ \forall i$. As is evident from Fig. 4.8, we get multiple lanes between each pair of cities in this case and trajectories become predominately *state-separated*, as we expect based on the discussion in Section 4.3.3.

The average computation time per vehicle is 4 minutes using BEACLS on a desktop computer with a Core i7 5820K processor and two GeForce GTX Titan X graphics processing units. The computation time is much longer than in the previous simulation in SF because of the larger space over which planning is done. Computations were done on a $209 \times 329 \times 15$ grid, which corresponds to a spatial resolution of 50 meters and an angular resolution of 24 degrees over a 10 kilometer $\times$ 15 kilometer area over the bay.

Once again all the computation is done offline and only a look-up table query is required

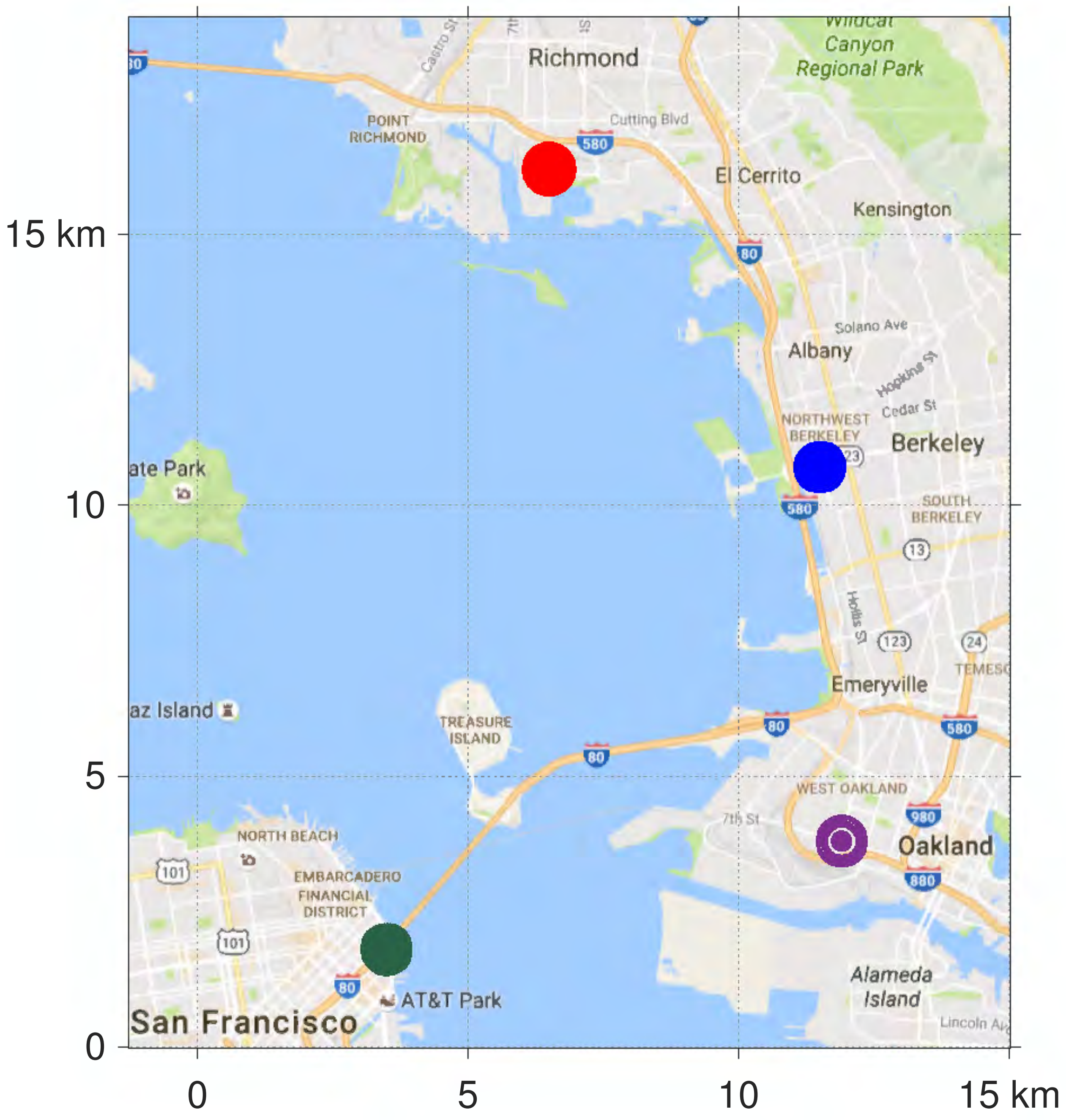


Figure 4.6: Multi-city simulation setup. A 300 km^2 area of San Francisco Bay Area is used as the state-space for vehicles. STP vehicles fly to and from the four cities indicated by the four disks. The simulations are performed under the strong winds condition with $d_r = 11$ m/s.

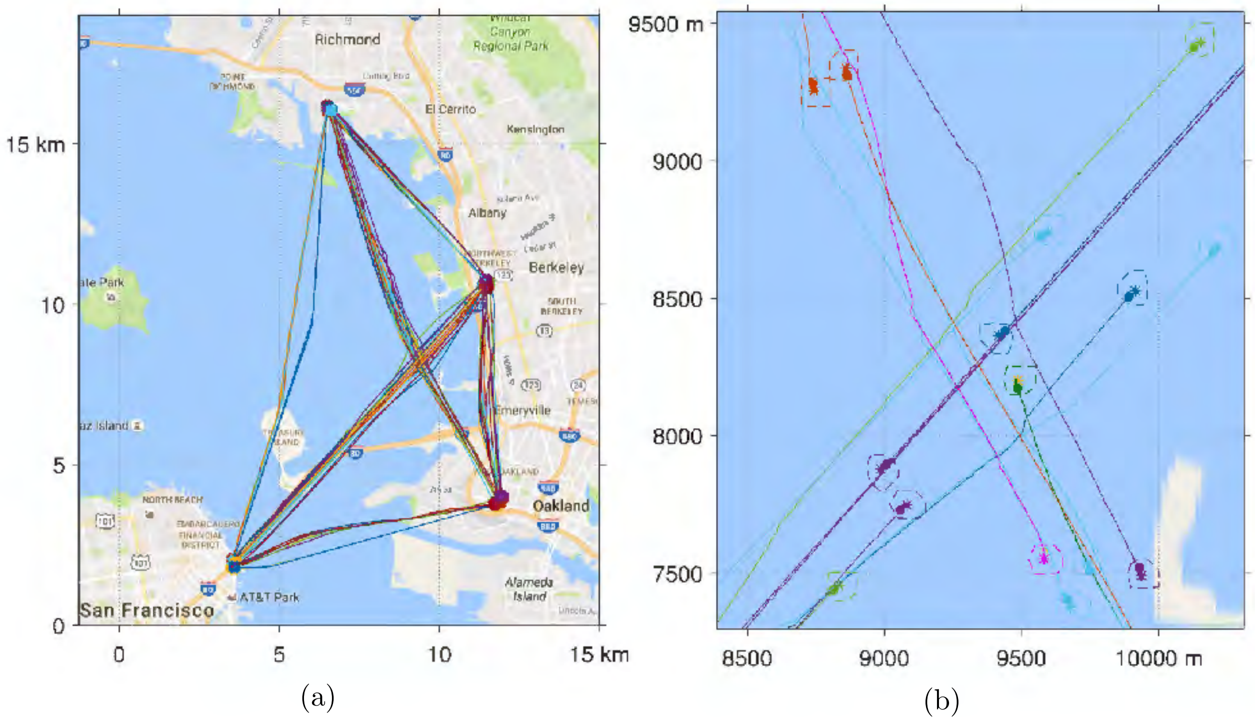


Figure 4.7: (a) Trajectories obtained from the STP algorithm for the multi-city simulation with $d_r = 11$ m/s, $t_i^{\text{STA}} = 5(i-1)$. (b) Zoomed-in version of the central area. A high density of vehicles is achieved at the center because of the intersection of several trajectories; however, the STP algorithm still ensures that vehicles do not enter each other's danger zones and reach their destinations.

in real-time, which can be performed very efficiently. Extrapolating the computation time comparison in Fig. 3.15, the MATLAB implementation would take prohibitively long – approximately 90 days for the entire simulation. This simulation illustrates the scalability and the potential of deploying the STP algorithm with BEACLS for provably safe trajectory planning for large multi-vehicle systems.

4.4 Sequential Trajectory Planning With An Adversarial Intruder

In Section 4.3, we discussed the basic STP algorithm that is robust to external disturbances. However, if a vehicle not in the set of STP vehicles enters the system, or even worse, if this vehicle is an adversarial intruder, the original plan can lead to vehicles entering into another vehicle's danger zone. If vehicles do not plan with an additional safety margin that takes a potential intruder into account, a vehicle trying to avoid the intruder may effectively become an intruder itself, leading to a domino effect. In this section, we propose a method that can

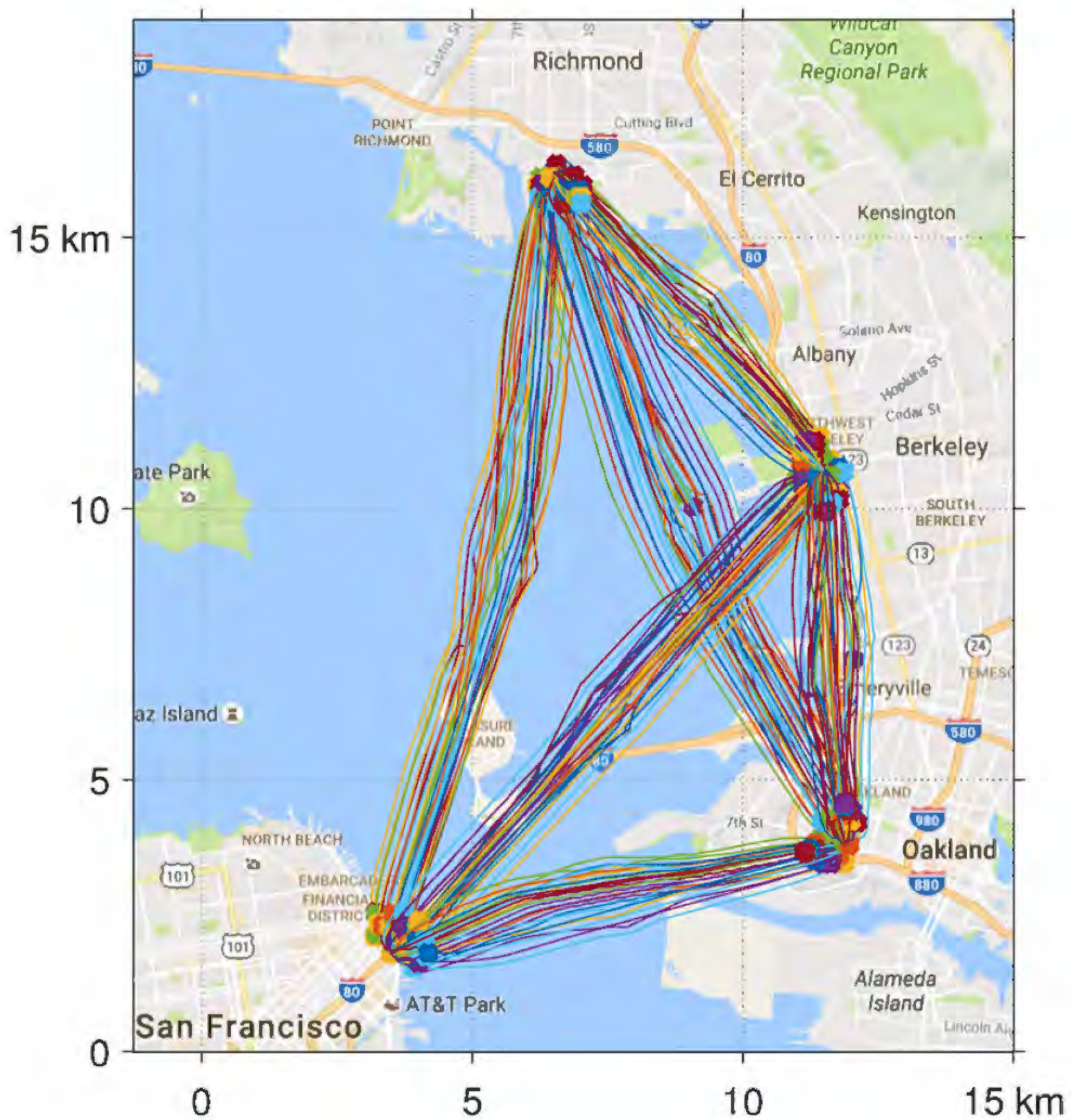


Figure 4.8: Vehicle trajectories for $d_r = 11$ m/s, $t_i^{\text{STA}} = 0$. Since different vehicles have same scheduled times of arrival, a multiple-lane behavior is observed between every pair of cities.

account for an intruder vehicle Q_I in the system while maintaining the STP structure.

An intruder vehicle may simply be a non-participating vehicle that could accidentally collide with other vehicles, or it could be one with malicious intent. This general definition of intruder allows us to develop algorithms that can also account for vehicles who are not communicating with the STP vehicles or do not know about the STP structure. In this section, our goal is to design a control policy for each vehicle that ensures separation with the intruder and other STP vehicles, and a successful transit to the destination.

4.4.1 Problem Setup and Solution Approach for Intruder Avoidance

In general, the effect of intruders on vehicles in structured flight can be unpredictable, since the intruders in principle could be adversarial in nature, and the number of intruders could be arbitrary. Therefore, to make our analysis tractable, we make the following two assumptions.

Assumption 3 *At most one intruder affects the STP vehicles at any given time. The intruder is removed after a duration of* t^{IAT}.

This assumption can be valid in situations where intruders are rare, and that some fail-safe or enforcement mechanism exists to force the intruder out of the planning space. Practically, over a large region of the unmanned airspace, this assumption implies that there would be one intruder vehicle per "planning region". Each planning region would perform STP independently from the others. The entire large region would be composed of several planning regions, and possibly several intruder vehicles. Note that we do not pose any restriction on the time at which intruder appears in the system; we only assume that once the intruder appears, it stays for a maximum duration of t^{IAT}.

Assumption 4 *The dynamics of the intruder are known and given by* $\dot{x}_I = f_I(x_I, u_I, d_I)$.

Assumption 4 is required for HJ reachability analysis. In situations where the dynamics of the intruder are not known exactly, a conservative model of the intruder may be used instead. We also denote the initial state of the intruder as x_I^0. Note that we only assume that the dynamics of the intruder are known, but its initial state x_I^0, control u_I and disturbance d_I it experiences are unknown.

Under the above assumptions, we present an intruder avoidance algorithm that ensures that only a *small and fixed* number of vehicles, $\bar{k}$, needs to replan their trajectories due to the intruder, regardless of the total number of vehicles, resulting in a constant replanning time. This is often an important considerations for real-world systems, since the replanning neeeds to be done during the runtime. Moreover, $\bar{k}$ is a design parameter, which can be chosen based on the resources available during run time.

Our algorithm consists of two phases: the planning phase and the replanning phase. In the planning phase, it is ensured that any two vehicles are sufficiently far enough from each other such that an intruder can be in the vicinity of at most $\bar{k}$ vehicles within the duration of

t^{IAT}. This division of the flight space guarantees that the intruder can affect the trajectory of at most $\bar{k}$ vehicles despite its best efforts, resulting in at most $\bar{k}$ replanning problems. In the replanning phase, we replan the trajectories of the affected vehicles by assigning the affected vehicles the lowest priority and using the STP algorithm presented in Section 4.3.

To design the flight space during the planning phase, we compute a *separation region* for each vehicle such that the vehicle needs to react to the intruder if and only if the intruder is inside this separation region. We then compute a *buffer region* between the separation regions of any two vehicles such that the intruder requires at least a duration of $t^{\text{BRD}} = \frac{t^{\text{IAT}}}{\bar{k}}$ to travel through this region. Thus, within the duration of t^{IAT}, the intruder can force at most $\bar{k}$ STP vehicles to deviate from their trajectories.

Remark 6 *For brevity, we present all our analyses and results in this section assuming that all STP vehicles have same dynamics and control constraints, and the intruder has the same state space as STP vehicles. However, the analysis to follow is more general and can easily be extended to the scenarios where the above assumptions do not hold. We refer the interested readers to the extended version of this section [36] for more details.*

4.4.2 Computation of Separation and Buffer Regions

The separation region, $\mathcal{S}_i(t)$, denotes the set of states of the intruder for which the vehicle Q_i is forced to apply an avoidance maneuver. $\mathcal{S}_i(t)$ is given by the set of states from which the joint states of Q_I and Q_i can enter the danger zone $\mathcal{Z}_{iI}$ despite the best efforts of Q_i to avoid Q_I.

Following [278], we define the relative state between Q_I and Q_i, which puts Q_i at the origin: $x_{Ii} = \phi(x_I, x_i)(x_I - x_i)$. For many dynamical systems, differentiating x_{Ii} with respect to time and choosing the appropriate $\phi(\cdot,\cdot)$ leads to the following relative dynamics, which is defined in the same dimensional space as the state space of each vehicle:

$$\dot{x}_{Ii} = f_r(x_{Ii}, u_i, u_I, d_i, d_I) \tag{4.25}$$

In relative state space, the set of potentially unsafe states is given by the backward reachable set $\mathcal{V}_i^{\text{A}}(t)$ with horizon $T = t^{\text{IAT}}$ and $t \in [0, t^{\text{IAT}}]$:

$$\begin{aligned} \mathcal{V}_i^{\text{A}}(t) =& \{x_{Ii} : \exists d_i(\cdot) \in \Gamma_i, \exists d_I(\cdot) \in \Gamma_I, \forall u_i(\cdot) \in \mathbb{U}_i, \exists u_I(\cdot) \in \mathbb{U}_I, \\ & \exists s \in [t, t^{\text{IAT}}], \xi(s; x_{Ii}, t, u_i(\cdot), u_I(\cdot), d_i(\cdot), d_I(\cdot)) \in \mathcal{L}_i^{\text{A}}\}, \\ \mathcal{L}_i^{\text{A}} =& \{x_{Ii} : \|p_{Ii}\|_2 \leq R_c\}. \end{aligned} \tag{4.26}$$

Here, $\mathcal{L}_i^{\text{A}}$ represent the collision states between Q_i and Q_I, and $\xi(\cdot)$ represent the trajectory corresponding to the dynamics in (4.25). For brevity purposes, we have dropped the time arguments from Γ and $\mathbb{U}$, and used the subscript to refer to the vehicle index. $\mathcal{V}_i^{\text{A}}$ can be computed by solving the following HJI VI

$$\min\{D_t V(t,x) + H(t,x,V(t,x)),\ l_i'(x) - V(t,x)\} = 0\ \forall x, t \qquad V(0,x) = l_i'(x), \tag{4.27}$$

where $\mathcal{L}_i^{\text{A}}$ is given by the subzero level set of l_i'. The Hamiltonian to compute the value function is given by:

$$H(t, x, V(t, x)) = \sup_{u_i \in \mathcal{U}_i} \inf_{\substack{u_I \in \mathcal{U}_I, \\ d_I \in \mathcal{D}_I, \\ d_i \in \mathcal{D}_i}} \langle D_x V(t, x), f_r(x, u_i, u_I, d_i, d_I) \rangle. \tag{4.28}$$

The interpretation of $\mathcal{V}_i^{\text{A}}(t)$ is that if Q_i starts inside this set, i.e., $x_{Ii}(t) \in \mathcal{V}_i^{\text{A}}(t)$, then the intruder can force Q_i to enter the danger zone $\mathcal{Z}_{iI}$ within a duration of $(t^{\text{IAT}} - t)$, regardless of the control applied by the vehicle. If Q_i starts at the boundary of this set (denoted as $\partial\mathcal{V}_i^{\text{A}}(t)$), i.e., $x_{Ii}(t) \in \partial\mathcal{V}_i^{\text{A}}(t)$, it can *barely* avoid the intruder for a duration of $(t^{\text{IAT}} - t)$ using the optimal avoidance control, u_i^{A}, that maximizes the Hamiltonian

$$u_i^{\text{A}}(x) = \arg \sup_{u_i \in \mathcal{U}_i} \inf_{\substack{u_I \in \mathcal{U}_I, \\ d_I \in \mathcal{D}_I, \\ d_i \in \mathcal{D}_i}} \langle D_x V(t, x), f_r(x, u_i, u_I, d_i, d_I) \rangle. \tag{4.29}$$

Finally, if Q_i starts outside $\mathcal{V}_i^{\text{A}}$, then Q_i and Q_I cannot instantaneously enter the danger zone $\mathcal{Z}_{iI}$, irrespective of the control applied by them at time t. In fact, Q_i can safely apply *any* control as long as it is outside the boundary of this set, but will have to apply the avoidance control once it reaches the boundary. Thus, if the intruder starts outside the separation region, the vehicle is guaranteed to avoid the intruder for at least a duration of t^{IAT}. Given $\mathcal{V}_i^{\text{A}}$, the separation region is given by

$$\mathcal{S}_i(t) = \mathcal{M}_i(t) + \mathcal{V}_i^{\text{A}}(0, t^{\text{IAT}}), \tag{4.30}$$

where the "+" in (4.30) denotes the Minkowski sum. Here, $\mathcal{M}_i(t)$ represents all possible states of Q_i at time t.

Remark 7 *In practice, vehicles are often sampled data systems, for which control signals can only be sent at regular time intervals. Thus, the control law in* (4.29) *is often hard to realize exactly on a system. Several papers in the literature [224, 89] formally addressing this issue. However, for most practical system, we can simply apply the avoidance controller when vehicle Q_i is within some positive distance of $\partial\mathcal{V}_i^A(0)$.*

We now compute a buffer region between the separation regions of any two vehicles such that the intruder requires at least a duration of t^{BRD} to travel through this region. This ensures that it can deviate at most $\bar{k}$ STP vehicles from their trajectories within a duration of t^{IAT}. In relative state space, the buffer region is given by the BRS, $\mathcal{V}_i^{\text{B}}(0)$, corresponding to the target set $\mathcal{V}_i^{\text{A}}(t^{\text{BRD}})$:

$$\begin{aligned} \mathcal{V}_i^{\text{B}}(0) = \{ & x_{Ii} : \exists d_i(\cdot) \in \Gamma_i, \exists d_I(\cdot) \in \Gamma_I, \exists u_i(\cdot) \in \mathbb{U}_i, \exists u_I(\cdot) \in \mathbb{U}_I, \\ & \exists s \in [0, t^{\text{BRD}}], \xi(s; x_{Ii}, 0, u_i(\cdot), u_I(\cdot), d_i(\cdot), d_I(\cdot)) \in \mathcal{V}_i^{\text{A}}(t^{\text{BRD}}) \}. \end{aligned} \tag{4.31}$$

$\mathcal{V}_i^{\mathrm{B}}(0)$ can be computed by solving a HJI VI similar to (4.27) with the Hamiltonian

$$H(t,x,V(t,x)) = \inf_{\substack{u_i\in\mathcal{U}_i, u_I\in\mathcal{U}_I,\\ d_i\in\mathcal{D}_i, d_I\in\mathcal{D}_I}} \langle D_x V(t,x), f_r(x,u_i,u_I,d_i,d_I)\rangle. \tag{4.32}$$

Intuitively, $\mathcal{V}_i^{\mathrm{B}}(0)$ represents the set of all relative states x_{Ii} from which it is possible to reach the boundary of $\mathcal{V}_i^{\mathrm{A}}(t^{\mathrm{BRD}})$ within a duration of t^{BRD}. Thus, as long as the initial relative state is outside $\mathcal{V}_i^{\mathrm{B}}(0)$, Q_i does not need to deviate from its path to avoid the intruder for at least a duration of t^{BRD}. Given $\mathcal{V}_i^{\mathrm{B}}(0)$, the buffer region $\mathcal{B}_{ij}(t)$ at time t between vehicle Q_i and a higher-priority vehicle Q_j is given by

$$\mathcal{B}_{ij}(t) = \partial\mathcal{S}_j(t) + \left(-\mathcal{V}_i^{\mathrm{B}}(0)\right). \tag{4.33}$$

Intuitively, if the intruder is affecting Q_j at time t, its state must be within the set $\mathcal{S}_j(t)$. Thus, Q_i should be at least $\mathcal{V}_i^{\mathrm{B}}(0)$ farther from the boundary of $\mathcal{S}_j(t)$. $\mathcal{B}_{ij}(t)$ formally captures this intuition.

4.4.3 Trajectory Planning for Intruder Avoidance

In addition to maintaining the buffer between Q_i and a higher priority vehicle, Q_j, we need to make sure that any two STP vehicles accidentally do not come too close to each other while applying the avoidance maneuver. Consequently, it is sufficient to ensure that their relative state remains outside the BRS, $\mathcal{V}_{ij}^{\mathrm{C}}(0)$, representing the set of all relative states x_{ij} from which the vehicles Q_i and Q_j can enter the set $\mathcal{Z}_{ij}$ within a time horizon of t^{IAT}. Thus, the lower priority vehicle should avoid the set

$$\mathcal{O}_i^j(t) = \mathcal{M}_j(t) + \mathcal{V}_{ij}^{\mathrm{C}}(0). \tag{4.34}$$

Finally, we compute the set of states from which Q_i can collide with any static obstacle, $\mathcal{O}_i^{\mathrm{static}}$. This set is given by the BRS $\mathcal{V}_i^{\mathrm{S}}(t)$, representing the set of all states of Q_i at time t that can lead to a collision with a static obstacle for some time $\tau \in [t, t+t^{\mathrm{IAT}}]$ for some control strategy of Q_i. For the detailed definitions of $\mathcal{V}_{ij}^{\mathrm{C}}$ and $\mathcal{V}_i^{\mathrm{S}}$, we refer the interested readers to the equations (50) and (32) respectively in [36].

Thus, the overall set of states that Q_i needs to avoid is:

$$\mathcal{G}_i(t) = \mathcal{V}_i^{\mathrm{S}}(t) \bigcup \cup_{j=1}^{i-1}\mathcal{O}_i^j(t) \bigcup \cup_{j=1}^{i-1}\mathcal{B}_{ij}(t). \tag{4.35}$$

To account for robustness to disturbances, we augment the total obstacle set by Ω_i:

$$\tilde{\mathcal{G}}_i(t) = \mathcal{G}_i(t) + \Omega_i. \tag{4.36}$$

Finally, we modify $\mathcal{L}_i$ to be $\tilde{\mathcal{L}}_i := \{x_i : \Omega_i + x_i \subseteq \mathcal{L}_i\}$, and compute a BRS $\mathcal{V}_i^{\mathrm{PP}}(t)$ for trajectory planning that contains the initial state of Q_i and avoids $\tilde{\mathcal{G}}_i(t)$:

$$\begin{aligned}\mathcal{V}_i^{\mathrm{PP}}(t) = \{x : \exists u_i(\cdot) \in \mathbb{U}_i^p, \forall s \in [t, t_i^{\mathrm{STA}}], \xi(s; x, t, u_i(\cdot)) \notin \tilde{\mathcal{G}}_i(s),\\ \exists s \in [t, t_i^{\mathrm{STA}}], \xi(s; x, t, u_i(\cdot)) \in \tilde{\mathcal{L}}_i\},\end{aligned} \tag{4.37}$$

Here, $\xi(\cdot)$ represent the system trajectory corresponding to the disturbance-free dynamics in (4.3), except the control authority is changed to $\mathcal{U}_i^p$. The BRT $\mathcal{V}_i^{\text{PP}}(t)$ can be computed in an analogous fashion to $\mathcal{V}_i^{\text{basic}}(t)$ using the HJI VI in (4.7). $\mathcal{V}_i^{\text{PP}}(\cdot)$ ensures goal satisfaction for Q_i in the absence of intruder and disturbance. The corresponding goal satisfaction controller is given by:

$$u_i^{\text{rtt}}(t,x) = \arg \inf_{u_i \in \mathcal{U}_i^p} \langle D_x V(t,x), f_i(x,u_i)\rangle. \tag{4.38}$$

The nominal trajectory $x_{r,i}(\cdot)$ can thus be obtained by using vehicle dynamics (4.3), with the optimal control given by (4.38). From the resulting nominal trajectory $x_{r,i}(\cdot)$, the overall control policy to reach $\mathcal{L}_i$ in the absence of intruder is given by:

$$u_i^{\text{PP}}(t,x_i) = \arg \sup_{u_i \in \mathcal{U}_i} \inf_{u_{r,i} \in \mathcal{U}_i^p, d_i \in \mathcal{D}_i} \langle D_x V(-\infty, e_i), f_{e_i}(e_i, u_i, u_{r,i}, d_j)\rangle. \tag{4.39}$$

where e_i is the error state (with respect to the nominal trajectory) as defined in (4.11).

When intruder is not present in the system, Q_i applies the control u_i^{PP} to safely reach its target. Once intruder appears in the system, Q_i applies the avoidance control u_i^{A} and hence might deviate from its nominal trajectory. The overall control policy for avoiding the intruder and collision with other vehicles is thus given by:

$$u_i^*(t,x_i,x_I) = \begin{cases} u_i^{\text{PP}}(t,x_i) & t < \hat{t}_i \\ u_i^{\text{A}}(t,x_{Ii}) & \text{otherwise} \end{cases} \tag{4.40}$$

where $\hat{t}_i$ denote the time at which Q_i is first forced to apply an avoidance maneuver. Mathematically, $\hat{t}_i = \min\{t : x_{Ii}(t) \in \mathcal{V}_i^{\text{A}}(t)\}$.

If Q_i starts within $\mathcal{V}_i^{\text{PP}}$ and uses the control u_i^*, it is guaranteed to avoid collision with the intruder and other STP vehicles, regardless of the control strategy of Q_I. Finally, since we use separation and buffer regions as obstacles during the trajectory planning of Q_i, it is guaranteed that at most $\bar{k}$ vehicles are forced to deviate from their path due to the intruder.

Note that after an intruder forces an STP vehicle to apply the avoidance controller, the STP vehicle will continue to apply this controller until the intruder leaves the system, at which time replanning will be done.

The planning phase for the intruder avoidance is summarized in Algorithm 4.

4.4.4 Replanning after Intruder Avoidance

After the intruder disappears, we have to replan the trajectories of the vehicles that were affected by Q_I. Let $\mathcal{N}^{\text{RP}}$ denote the set of all vehicles for whom replanning is required. $\mathcal{N}^{\text{RP}}$ can be obtained by checking if a vehicle Q_i applied any avoidance control during $[\underline{t}, \underline{t} + t^{\text{IAT}}]$, i.e.,

$$\mathcal{N}^{\text{RP}} = \{Q_i : \exists t \in [\underline{t}, \underline{t} + t^{\text{IAT}}], x_{Ii}(t) \in \mathcal{V}_i^{\text{A}}(t)\}, \tag{4.41}$$

where $\underline{t}$ denote the time at which the intruder was first detected in the system. Recall that due to the presence of separation and buffer regions, at most $\bar{k}$ vehicles can be affected by

Algorithm 4: The intruder avoidance algorithm: Planning-phase (performed offline)

```
input  : Set of vehicles $Q_i$ in the descending priority order, their dynamics (4.1)
         and initial states $x_i^0$;
         Vehicle destinations $\mathcal{L}_i$ and static obstacles $\mathcal{O}_i^{\text{static}}$;
         Intruder dynamics $f_I$ and the maximum avoidance time $t^{\text{IAT}}$ ;
         Maximum number of vehicles allowed to re-plan their trajectories $\bar{k}$.
output: The nominal controller $u^{\text{PP}}$ and the avoidance controller $u^{\text{A}}$ for all vehicles.
1  for $i = 1 : N$ do
2      Avoidance control $u_i^{\mathbf{A}}$ for $Q_i$
3      compute the avoid region $\mathcal{V}_i^{\text{A}}(t)$ using (4.26);
4      compute the avoidance controller $u_i^{\text{A}}$ using (4.29);
5      if $i \neq 1$ then
6          for $j = 1 : i - 1$ do
7              Computation of separation region for $Q_i$
8              given the base obstacles $\mathcal{M}_j(\cdot)$ and the avoid region $\mathcal{V}_j^{\text{A}}$, compute the
                 separation region in (4.30);
9              Computation of buffer region for $Q_i$
10             given the separation region, compute the buffer regions $\mathcal{B}_{ij}(\cdot)$ in (4.33);
11             Computation of obstacles for $Q_i$
12             given the base obstacles $\mathcal{M}_j(\cdot)$, compute the obstacle $\mathcal{O}_i^j(\cdot)$ in (4.34)
13     compute the effective obstacle $\tilde{\mathcal{G}}_i(t)$ for $Q_i$ in (4.36);
14     Trajectory planning for $Q_i$
15     given the total obstacle set $\tilde{\mathcal{G}}_i(t)$, compute the BRS $\mathcal{V}_i^{\text{PP}}(t)$ defined in (4.37);
16     The nominal controller of $Q_i$
17     compute the nominal controller $u_i^{\text{PP}}(\cdot)$ given by (4.39);
18     Base obstacle induced by $Q_i$
19     given the nominal controller $u_i^{\text{PP}}(\cdot)$ and the BRS $\mathcal{V}_i^{\text{PP}}(t)$, compute the base
         obstacles $\mathcal{M}_i(\cdot)$ using (4.18).
```

Q_I, i.e., $|\mathcal{N}^{\text{RP}}| \leq \bar{k}$. It is important to note that the intruder may re-introduce the coupling among the vehicles in the set $\mathcal{N}^{\text{RP}}$. However, once the intruder disappears from the system, we assign these vehicles the lowest priority and again use STP to decouple their planning process. In fact, one of the motivations for designing an algorithm for a fixed $\bar{k}$ is to ensure that this decoupling and replanning can be performed efficiently during the run-time based on the available computation resources.

We also note that there could be potential delays in the scheduled arrival of these vehicles depending on their state after the intruder disappears from the system; however, our algorithm ensures that at most $\bar{k}$ vehicles will be delayed.

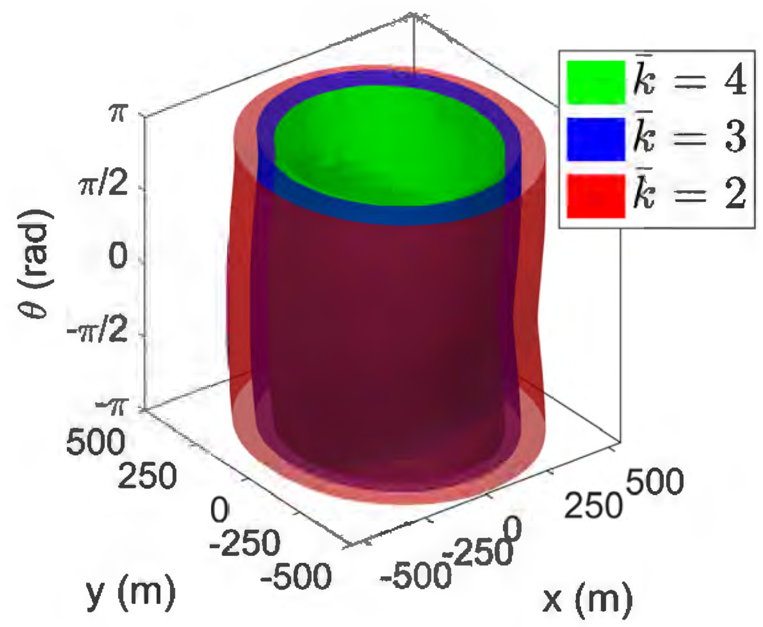


Figure 4.9: Buffer regions for different $\bar{k}$ (best visualized with colors). As $\bar{k}$ decreases, a larger buffer is required between vehicles to ensure that the intruder spends more time traveling through this buffer region so that it forces fewer vehicles to apply an avoidance maneuver.

Remark 8 *In this work, we assume worst-case scenarios in terms of the behavior of the intruder, the effect of disturbances, and the planned trajectories of each STP vehicle. Consequently, we are able to guarantee safety and goal satisfaction of all vehicles in all possible scenarios given the bounds on intruder dynamics and disturbances. To achieve denser operation of STP vehicles, known information about the intruder, disturbances, and specifies of STP vehicle trajectories may be incorporated; however, we defer such considerations to future work.*

4.4.5 City Environment Simulations in the Presence of an Intruder

We now illustrate the proposed intruder avoidance algorithm using the fifty-vehicle example introduced in Section 4.3.3.

Simulation setup. In addition to the setup in Section 4.3.3, the vehicles now also need to account for the possibility of the presence of an intruder for a maximum duration of t^{IAT} =10 s. The intruder dynamics are given by (4.24).

Results. We present the simulation results for $\bar{k} = 3$. The resultant buffer region is shown in Blue in Figure 4.9. For the comparison purposes, we also compute the buffer regions for

$\bar{k} = 2$ and $\bar{k} = 4$. As shown in Figure 4.9, a bigger buffer is required between vehicles when $\bar{k}$ is smaller. Intuitively, when $\bar{k}$ is smaller, a larger buffer is required to ensure that the intruder spends more time "traveling" through this buffer region so that it can affect fewer vehicles within the same duration of $t^{\text{IAT}} = 10\,\text{s}$.

These buffer region computations along with the induced obstacle computations are performed sequentially for each vehicle to obtain $\mathcal{G}(\cdot)$ in (4.35). This overall obstacle set is then used during their trajectory planning and the control policy $u^{\text{PP}}(\cdot)$ is computed, as defined in (4.39). Finally, the corresponding nominal trajectories are obtained by executing control policy $u^{\text{PP}}(\cdot)$. The time for planning for each vehicle is approximately 15 minutes on a MATLAB implementation on a desktop computer with a Core i7 5820K processor. With BEACLS using two GeForce GTX Titan X graphics processing units, this computation time is reduced to approximately 9 seconds per vehicle.

The nominal trajectories and the overall obstacles for different vehicles are shown in Figure 4.10. The numbers in the figure represent the vehicle numbers. The nominal trajectories (solid lines) are well separated from each other to ensure collision avoidance even during a worst-case intruder "attack". At any given time, the vehicle density is low to ensure that the intruder cannot force more than three vehicles to apply an avoidance maneuver. This is also evident from large obstacles induced by vehicles for the lower priority vehicles (dashed circles). This lower density of vehicles is the price that we pay for ensuring that the replanning can be done efficiently in real-time.

In Figure 4.11, we plot the distance between an STP vehicle and the intruder when the vehicle applies the control policies $u^{\text{PP}}(\cdot)$ (Red line) and u^{A} (Blue line) in the presence of the intruder. Black dashed line represents the collision radius $r = 100\,\text{m}$ between the vehicle and the intruder. If the vehicle continues to apply the control policy $u^{\text{PP}}(\cdot)$ in the presence of an intruder, the intruder enters in its danger zone. Thus, it is forced to apply the avoidance control, which can cause a deviation from the nominal trajectory, but will successfully avoid the intruder, as indicated by the Blue curve.

The relative buffer region between vehicles is computed under the assumption that both the STP vehicle and the intruder are trying to collide with each other; this is to ensure that the intruder will need at least a duration of t^{BRD} to reach the boundary of the avoid region of the next vehicle, irrespective of the control applied by the vehicle. However, a vehicle will be applying the control policy $u^{\text{PP}}(\cdot)$ unless the intruder forces it to apply an avoidance maneuver, which may not necessarily correspond to the policy that the vehicle will use to *deliberately* collide with the intruder. Therefore, it may not be able to affect $\bar{k}$ vehicles even with its best strategy to affect maximum vehicles. In this simulation, the intruder is able to force only two vehicles to apply an avoidance maneuver. The set of vehicles that will need to replan their trajectories is given by $\mathcal{N}^{\text{RP}} = \{Q_1, Q_2\}$. After the intruder disappears, Q_1 and Q_2 replan their trajectories. The replanning process using BEACLS takes approximately 18 seconds (9 seconds per vehicle).

However, 18 seconds is likely still too slow for several practical applications. Since reachability computations are highly parallelizable, with computations on each grid point being independent of others, replanning should be possible to do within a fraction of a second with

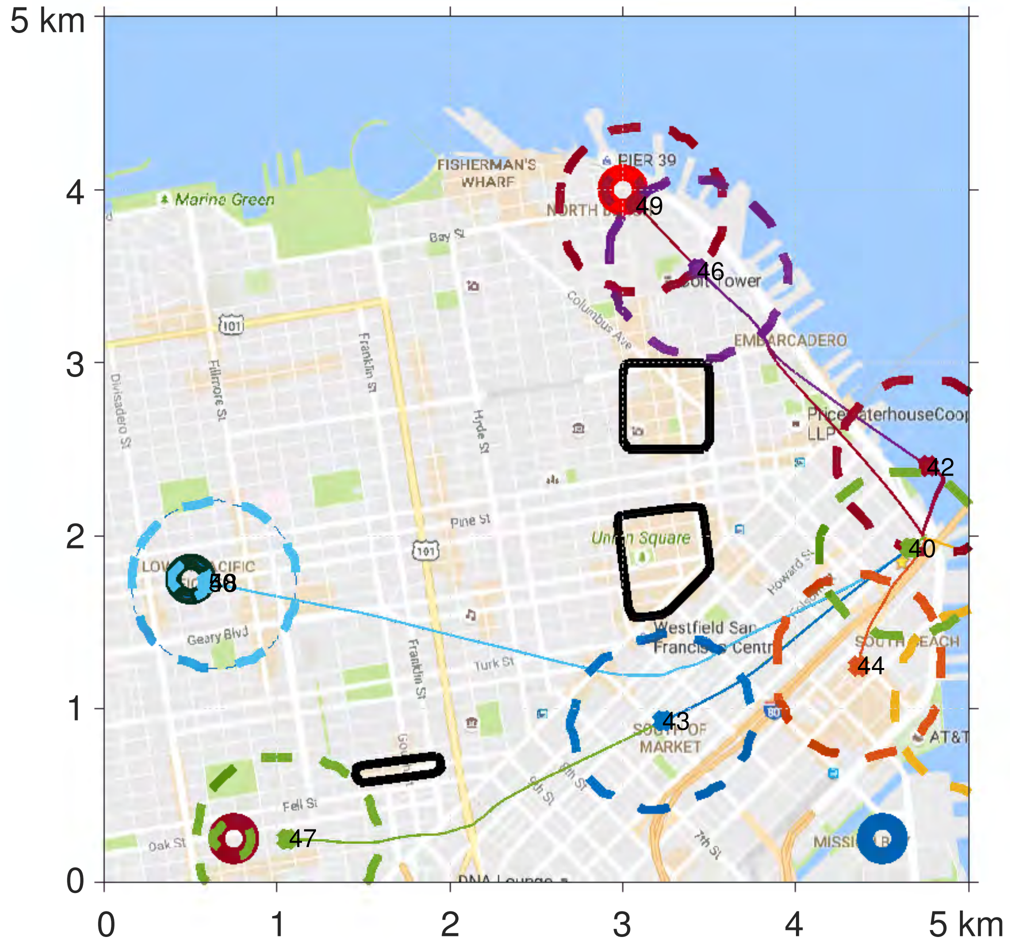


Figure 4.10: Nominal trajectories and induced obstacles by different vehicles. The nominal trajectories (solid lines) are well separated from each other to ensure that the intruder cannot force more than 3 vehicles to apply an avoidance maneuver.

more computational resources. For example, an off-board server could utilize many GPUs for the computations, and transmit the plans back to the vehicles in real time. However, the real-world scalability of the computation speed with the number of GPUs still needs to be tested; memory bandwidth could be a potential issue when a large number of GPUs is used. Alternatively, a smaller $\bar{k}$ can be used during the planning phase to make sure that the replanning needs to be done for fewer vehicles.

Discussion. The simulations illustrate the effectiveness of reachability in ensuring that the STP vehicles safely reach their respective destinations even in the presence of an intruder. However, they also highlight some of the conservatism in the worst-case reachability analysis. For example, in the proposed algorithm, we assume the worst-case disturbances and intruder behavior while computing the buffer region and the induced obstacles, which results in a

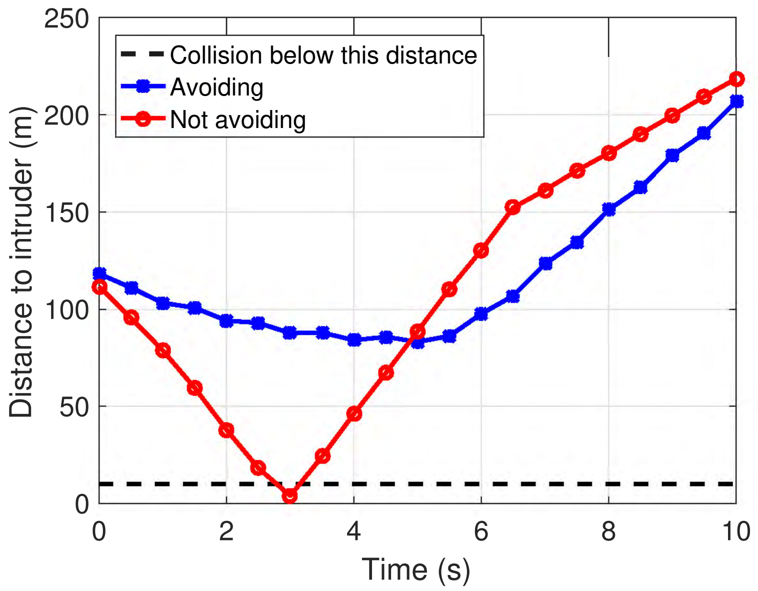


Figure 4.11: The trajectory of a STP vehicle when it applies the nominal controller vs when it applies the avoidance control. The vehicle is forced to apply the avoidance maneuver in the presence of an intruder, which can cause vehicle's deviation from its nominal trajectory.

large separation between vehicles and hence a lower vehicle density overall, as evident from Figure 4.10. Similarly, while computing the buffer region, we assumed that a vehicle is *deliberately* trying to collide with the intruder so we once again consider the worst-case scenario, even though the vehicle will only be applying the nominal control strategy $u^{\text{PP}}(\cdot)$, which is usually not be same as the worst-case control strategy. This worst-case analysis is essential to guarantee safety regardless of the actions of STP vehicles, the intruder, and disturbances, given no other information about the intruder's intentions and no model of disturbances except for the bounds. However, the conservatism of our results illustrates the need and the utility of acquiring more information about the intruder and disturbances, and of incorporating knowledge of the nominal strategy $u^{\text{PP}}(\cdot)$ in future work. One particularly promising direction might be to use online reachability methods, such as [150, 29], to update the reachable sets during the run-time as the new information about the intruder dynamics or strategy is acquired.

4.5 Chapter Summary

Provably safe multi-vehicle trajectory planning in an important problem that needs to be addressed to ensure that vehicles can fly in close proximity of each other. In this chapter, we propose a Sequential Trajectory Planning (STP) algorithm for provably safe, cooperative multi-vehicle trajectory planning in the presence of external disturbances. The proposed method assigns a strict priority ordering to vehicles to offer a tractable and practical approach to the multi-vehicle path planning problem that scales linearly with the number of vehicles. Under the proposed method, a portion of "space- time" is reserved for vehicles in the airspace in descending priority order to allow for dense vehicle configurations. We also demonstrate how different types of space-time trajectories emerge naturally out of the STP algorithm for different disturbance conditions and other problem parameters.

We also propose an algorithm to account for an adversarial intruder in sequential trajectory planning. The proposed method ensures that only a fixed number of vehicles need to replan their trajectories once the intruder disappears, irrespective of the total number of vehicles. Thus, the replanning process is feasible in real-time. Finally, we demonstrate how we can combine the linear scaling of the STP framework with BEACLS, that can leverage the computation power of GPUs, for efficient and provably safe large-scale multi-vehicle trajectory planning problems in urban and multi-city environments.

Part II

Going Beyond Known Dynamics Models and Environments: Learning-Based Control for *Unknown* Models and Environments

In Part 1 of this thesis, we discussed how we can design safe controllers for single and multi-agent systems using tools from robust optimal control. However, to perform this safety analysis, we often need to have a very good understanding of the environment and conditions in which these systems will operate. For example, for multi-vehicle trajectory planning problem discussed in Chapter 4, we need to know a dynamics model for each vehicle, the knowledge of all the static and dynamic obstacles in the environment, and even a dynamics model of the adversarial intruder. However, having this information beforehand is simply not possible for future autonomous systems. For example, a future autonomous car will need to drive around construction sites and blocked lanes in the city, in crowded city downtowns around humans, and in different types of weather such as sunny, rainy, and snowy. On the one hand, it is important to know these conditions for the safety analysis; on the other hand, it is not tractable to have that knowledge until the car *actually* goes out and drive on the road. As humans, we can drive reliably in the aforementioned uncertain conditions by leveraging our prior driving experience. So a natural question to ask is "can we use prior data to also operate autonomous systems in uncertain and unknown environments?"

In this part of the thesis, we will discuss how we can use data and tools from machine learning to go past the assumptions typically required for the control and safety analysis of the system. We will first discuss the scenarios where the dynamics of the system itself are not available (Chapter 5 and 6); we will then discuss the scenarios where a dynamics model is available, but the system is operating in *a priori* unknown environment (Chapter 7).

Chapter 5

Learning for Unknown Dynamics Models: Indirect Learning-Based Control

This chapter is based on the paper "Learning Quadrotor Dynamics Using Neural Network for Flight Control" [30] written in collaboration with Anayo K. Akametalu, Frank Jiang, Forrest Laine, and Claire Tomlin.

First principles-based dynamics models have been successfully used for decades within the control community for designing and analyzing controllers for autonomous systems. These models form the basis for the control of our chemical plants, thermostats, all the way to highly unstable system such as aircrafts. In previous chapters, we also saw how we can also use these models for the safety analysis of autonomous systems. However, as autonomous systems will operate in unstructured environments, they will inevitably experience additional external effects that are hard to model using first principles. For example, a quadrotor experiences blade flapping at high speeds; similarly, its aerodynamics change near ground (also referred to as *ground effects*). These inaccuracies in the model appear to the controller as external disturbances; thus, they can significantly affect the quadrotor performance, particularly when the system is operated near its limits.

One can take a robust control approach to account for these model inaccuracies – a bound on the inaccuracies is estimated and then the controller is designed to perform well despite the worst-case inaccuracy. However, a robust controller spend the controller energy to *counter* the inaccuracies, rather than *leveraging* these dynamic effects to its advantage. This can significantly limit controller's capabilities in countering the actual disturbances in the environment and/or satisfactorily completing the desired control task, as a part of the control authority is now being spent on countering the model inaccuracies.

To circumvent these issues, one can use data-driven methods to explicitly model these inaccuracies, *directly* from the data collected on the actual system. This identified model can then be used to design a controller for the system, in the hope that a more accurate dynamics model will also lead to better control performance. This two-step approach to controller design is somewhat *indirect* compared to a *direct* approach wherein a controller is directly designed to improve the control performance, rather than identifying the model inaccuracies first. The main advantage of indirect methods is that learning a model tends to be more sample efficient than directly optimizing the controller; moreover, the learned model is task-agnostic and thus can generalize the control performance to new tasks. Motivated by this, this chapter focuses on indirect learning-based control methods; we will discuss direct learning-based control methods in Chapter 6.

Within indirect learning-based control methods, one potential approach can be to model the inaccuracies using deep neural networks (DNN). DNNs are known to be universal function approximators; their structure allows them to model highly nonlinear functions and unobserved states directly from the observed data, which might in general be hard to model [192]. In this work, we investigate "*can we use DNNs to model the dynamics model inaccuracies, and more importantly, can we use the identified model for controller design?*"

Remark 9 *A notational shift that is worth noting starting from this chapter is that we will primarily pose the control problem in discrete-time. Discrete-time notation is particularly suitable for data-driven control methods because of the inherently discrete nature of the most datasets collected on the real systems. Furthermore, the discrete-time notation is more popular in reinforcement learning (RL) literature, which allows us to draw parallels between RL and control methods. Nevertheless, most of the algorithms that we will discuss can be straightforwardly extended to the continuous time, and we will comment on this extension wherever relevant.*

5.1 Related Work

System Identification (SysID). System identification (SysID) [199], the mathematical modeling of a system's dynamics using data, is one of the most basic and important components of control. Roughly speaking, the field of system identification uses statistical methods to fit a given set of basis functions (also called *features*) to approximate the system dynamics For example, the features could be monomials and the SysID process identifies the coefficient of these monomials to apprximate the overall system dynamics by a polynomial in state space. In other cases, a linear basis is used, resulting in a linear approximation of non-linear dynamics. SysID also includes the optimal design of experiments for efficiently generating informative data for fitting these features.

In other cases, the functional form of the dynamics is known (for example through first principles) and SysID is used to identify unknown parameters in the first principle models from the measured data. For example, these parameters could be mass and inertia for a quadcopter, or friction coefficients for a ground robot, that are unknown beforehand.

However, when an autonomous system is operating in unstructured environments, it might be challenging to even come up with the functional form or features for the unmodeled dynamics, making it challenging to apply traditional SysID approaches. In contrast, one of the advantages of the modern learning approaches is that they can *automatically* learn the appropriate features from the data.

Model-Based Reinforcement Learning (RL). In model-based RL, a system dynamics model is learned directly using the data collected on the system. Typically, the model is optimized so as to maximize the log likelihood of the observed data. For forward dynamics model, the model is optimized to minimize the prediction error of the next state given the current state and input. Model-based RL is a relatively younger field, which has evolved (and continue to evolve) significantly since this work was conducted in 2015. Here, we attempt to provide a sense of the current state of the field rather than a historical snapshot.

One popular regression tool used within model-based RL community to learn dynamics models is Gaussian Processes (GP) [94, 235, 233, 269, 242]. GPs are particularly suitable for learning dynamics models because GP is a probabilistic regression method and thus it also provides an uncertainty measure for the underlying function. In other words, GPs provide an estimate of where the learned dynamics model can be trusted and where it cannot be. This uncertainty measure can be used to avoid overoptimistic behaviors during the planning process. However, GP inference time scales cubically with the number of training samples [253], making it challenging to use for high-dimensional systems, where we often need a large number of data samples to learn a good model.

To overcome the challenges associated with GP-based dynamics models, DNN-based dynamics models have been explored for learning dynamics models for manipulators [193, 121], helicopters [251], microrobots [231], and even for learning dynamics models via high-dimensional observations such as images [301, 11]. However, DNN-based dynamics models typically struggle with long horizon planning because, unlike GPs, DNNs do not provide an estimate of the accuracy of the learned dynamics model, leading it to confidently predict inaccurate dynamics. To overcome this challenge, Bayesian Neural Networks (BNNs) [124] as well as several approximations of BNNs that provide an estimate of uncertainty in the NN output have been considered [82]; however, it remains a challenge to reliably estimate uncertainty in the outputs of DNNs.

Model-Free Reinforcement Learning. A natural way to optimize the control performance is through model-free RL methods or direct adaptive control methods, wherein controller parameters are directly optimized to improve the control performance without necessarily maintaining an intermediate dynamics model. For example, in [67], the authors directly learn the parameters of a linear feedback controller using BO to obtain walkimng gait for a bipedal microrobot. However, a typical controller might be non-linear and can contain hundreds of parameters; it is not feasible to optimize such high-dimensional controllers using BO [276]. Other model-free approaches such as Deep-Q learning [297], TRPO [271], and PPO [272] are very effective at learning complex, high-dimensional policies, but the convergence might require millions of trials and often lack the robustness of the model-based controllers [254]. Finally, unlike model-based approaches, learned controllers are task specific, and often, learning needs to be performed from scratch for a new task.

Indirect Adaptive Control. Another approach to deal with model inaccuracies during the controller design is through adaptive control [24, 264]. In adaptive control, a plant model or controller parameters are varied in real-time based on the observed data to improve the overall control performance. Unlike traditional SysID approaches which are typically performed offline, adaptive control mechanisms are performed in real-time, as the system operates in the environment, allowing it to adapt to changing conditions.

Adaptive control methods can be broadly classified as direct and indirect methods. Direct methods are ones wherein the controller parameters are directly adapted to improve the control performance. In contrast, indirect methods are those in which plant parameters are adapted, and then the adapted plant model is used to obtain a controller or to calculate required controller parameters. Hybrid methods rely on both estimation of plant parameters and direct modification of the control law.

At this point, it is important to comment on the similarities and differences between adaptive control, system identification, and reinforcement learning (RL). Direct adaptive control methods are closely related to model-free reinforcement learning algorithms, in that both aim to adjust the controller parameters directly to improve the control performance. The main distinction comes from the metrics and properties that are considered during this adaptation process. When designing adaptive control systems, special consideration is given to convergence, stability, and robustness issues. In particular, adaptive control emphasizes fast convergence *without* failure, whereas RL algorithms are permitted to fail during the process of learning. Thus, adaptive control mechanisms are more widely used when the stringent performance constraints are required to be met during the learning process, e.g., as needed in safety-critical systems like airplanes. However, this emphasis on convergence without failure in adaptive control often comes in the form of more assumptions and restrictions on the underlying system and controller, compared to very few assumptions made in model-free RL [267]. Another distinction between the two comes from the optimization objective: model-free RL can have rather general optimization objectives – not just, e.g., minimal tracking error. A final distinction comes from the communities in which these methods were developed and the applications that motivated their development – adaptive control methods have been primarily developed in control community, with primary motivation being improved control performance for aircrafts as its mass changes online (due to reduction in fuel); whereas, RL was primarily developed in artificial intelligence and computer science communities to understand the psychology of animal learning through trial and error [284].

Similarly, system identification and model-based reinforcement learning are closely related concepts as well, wherein a system dynamics model is identified using data from the actual system. Once again, the main distinction comes from the properties that are desirable in the identified model – SysID provides stronger claims about the identified model and the estimation uncertainty, but often it comes in the form of more assumptions on the underlying model class. One such example is the assumption of known feature basis (e.g., monomials or linear bases) which the true system model belongs from. In contrast, model-based RL aims to use recent advances in machine learning and computation to *automatically* learn the required model features. When the system dynamics model is additionally adapted or learned in an online fashion, it can be thought of as an indirect adaptive control method (sometimes also referred to as online system identification or online model-based RL). For example, a particular class of indirect adaptive control methods called model identification adaptive controllers (MIACs) can also be thought of as an online SysID approach, wherein SysID is performed while the system is running. However, other indirect adaptive control mechanisms exist as well that use a reference model for adaptation; therefore, indirect adaptive control can be thought of as a broader umbrella which online SysID or online model-based RL is a part of. Nevertheless, similar to SysID, indirect adaptive control methods also make strong assumptions about the underlying system in order to provide performance guarantees, making them challenging to apply directly in unstructured environments.

Contributions and Chapter Organization

In this work, we demonstrate how DNNs can be used to learn the dynamics inaccuracies in a system model. We focus on three practically important questions and investigate (i) whether a highly nonlinear dynamics model given by a DNN can be effectively used to design a controller for the system, (ii) whether it is general enough to be used to design a controller for the tasks that the network was *not* trained on, and (iii) what some important consideration are when using DNN-based dynamics models for a system.

For this purpose, we collect state-input data of a nano-quadrotor Crazyflie 2.0 by flying it on the trajectories that consist of translational or rotational motion, but not both. We next train a feed-forward Rectified-Linear Unit (ReLU) NN to learn the state-space dynamics of Crazyflie. To test the generalization capabilities of the trained NN, we use the learned NN model to control the quadrotor on a trajectory that consists of a simultaneous translational and rotational motion, requiring NN to infer highly nonlinear couplings between the rotational and translational dynamics.

5.2 Problem Formulation

Consider a general, time-invariant, non-linear dynamical system

$$\begin{aligned} x_{t+1} &= f(x_t, u_t), && (5.1)\\ &= f_{nom}(x_t, u_t) + f_{unknown}(x_t, u_t), && (5.2) \end{aligned}$$

where the dynamics f are not completely known. In particular, a nominal dynamics model of f_{nom} the system is known beforehand, which for example could be a first principle model. However, there are inaccuracies in the dynamics model or unmodeled dynamics effects $f_{unknown}$ which are unknown beforehand. Furthermore, functional form or bounds on $f_{unknown}$ are not known either.

The goal is to solve an optimal control problem for dynamics in (5.1), i.e., to design a controller such that it minimizes the following cost function:

$$J_t(x_0, \mathbf{u}) = \sum_{k=t}^{T} L(x_k, u_k, k) + M(x_T), \tag{5.3}$$

where $\mathbf{u} \equiv [u_t, u_{t+1}, \ldots, u_T]$ represents the sequence of control over the time horizon $[t, T]$. There are two primary methods to design such a controller: one is to directly find the optimal control sequence $\mathbf{u}$; the other approach is to explicitly identify the model inaccuracies $f_{unknown}$ first and then use the identified model for the controller design. In this work, we focus on the latter approach, which we refer to as *indirect learning-based control* approach. The main advantage of identifying the unknown dynamics first is that the identified model is task agnostic – it can be used to design a controller for a variety of optimal control problems, and not just for a specific cost function.

5.3 Identification of Unmodeled Dynamics Using Deep Neural Networks

In this section, we will discuss how modeling the dynamics inaccuracies $f_{unknown}$ can be posed as a regression problem. We will then discuss how this regression problem can be solved using deep neural networks.

5.3.1 Identifying model inaccuracies using parametric regression

One potential approach to identify $f_{unknown}$ is to parameterize it using a function approximator, and then identifying the parameters that best fits the observed data. In particular, let $\hat{f}_{unknown}(x, u; \theta)$ denote a parameterized approximation of $f_{unknown}(x, u)$, parameterized by vector θ. The system identification task then becomes to find, given input and state data, parameters θ that minimize the prediction error in $f_{unknown}$. Note that for a physics-based model, θ generally captures the physical properties of the system (for example, mass, moment of inertia, etc. for a quadrotor); for a DNN-based model, the parameters can be thought of as degrees of freedom a DNN has to learn different nonlinear function.

Suppose our dataset consists of N state-input trajectories, collected on the real system, with T number of data points per trajectory. The optimal parameters θ^* can be obtained by solving the following optimization problem:

$$\theta^* = \min_{\theta} \sum_{i=1}^{N} \sum_{t=1}^{T} l\left(\tilde{f}_{unknown}(x_t^i, u_t^i) - \hat{f}_{unknown}(x_t^i, u_t^i; \theta)\right), \tag{5.4}$$

where $\tilde{f}_{unknown}(x_t^i, u_t^i)$ represent the observed values of $f_{unknown}$ at state x_t^i and control u_t^i.

In particular, given a state-input trajectory, $\tilde{f}_{unknown}$ can be computed using Equation (5.1):

$$\tilde{f}_{unknown}(x_t^i, u_t^i) = x_{t+1}^i - f_{nom}(x_t^i, u_t^i) \tag{5.5}$$

l in (5.4) represent the prediction error function that we want to minimize. A popular choice of prediction error function is mean squared prediction error (MSE) over a training set of collected data, solving

$$\theta^* = \min_{\theta} \sum_{i=1}^{N} \sum_{t=1}^{T} \|\tilde{f}_{unknown}(x_t^i, u_t^i) - \hat{f}_{unknown}(x_t^i, u_t^i; \theta)\|^2, \tag{5.6}$$

Depending on the form of $\hat{f}_{unknown}$, (5.6) results in a linear or a nonlinear least squares problem. For rich function approximators, such as deep neural networks, (5.6) often results into a non-linear least square problem. For the DNN-based dynamics model, we can find the optimal parameters using supervised learning.

5.3.2 Identifying model inaccuracies using DNNs

In this work, we use a deep neural network (DNN) to represent $\hat{f}_{unknown}$. One of the motivations behind using a DNN to parameterize $\hat{f}_{unknown}$ is their ability to represent complex, high-dimensional non-linear functions.

The input to the neural network is current state x_t and control input u_t, and the output of the neural network is an estimate of the unknown dynamics, $\hat{f}_{unknown}(x_t, u_t)$. In particular, we can construct the following dataset:

$$\mathcal{D} = \{\Big((x_1^1, u_1^1), \tilde{f}_{unknown}(x_1^1, u_1^1)\Big), \Big((x_2^1, u_2^1), \tilde{f}_{unknown}(x_2^1, u_2^1)\Big), \ldots, \Big((x_T^N, u_T^N), \tilde{f}_{unknown}(x_T^N, u_T^N)\Big)\}$$

Given dataset $\mathcal{D}$, the parameters of the neural network θ can be learned using supervised learning with loss function given by the MSE in (5.6) (see Sec. 2.5.1 for more details).

Once the training process is complete, the obtained parameters can be used to evaluate $\hat{f}_{unknown}$ at new state and control pairs, including the ones that were not observed during the training time. The learned model can also be used along with f_{nom} to design a controller for the system, which is our main focus in this chapter.

5.4 Experiments: Learning Quadrotor Dynamics Using DNN

We will now use the framework in Sec. 5.3 to learn the unmodeled dynamics of a quadrotor, Crazyflie 2.0 (see Fig. 5.1). The Crazyflie 2.0 is an open source nano quadrotor platform

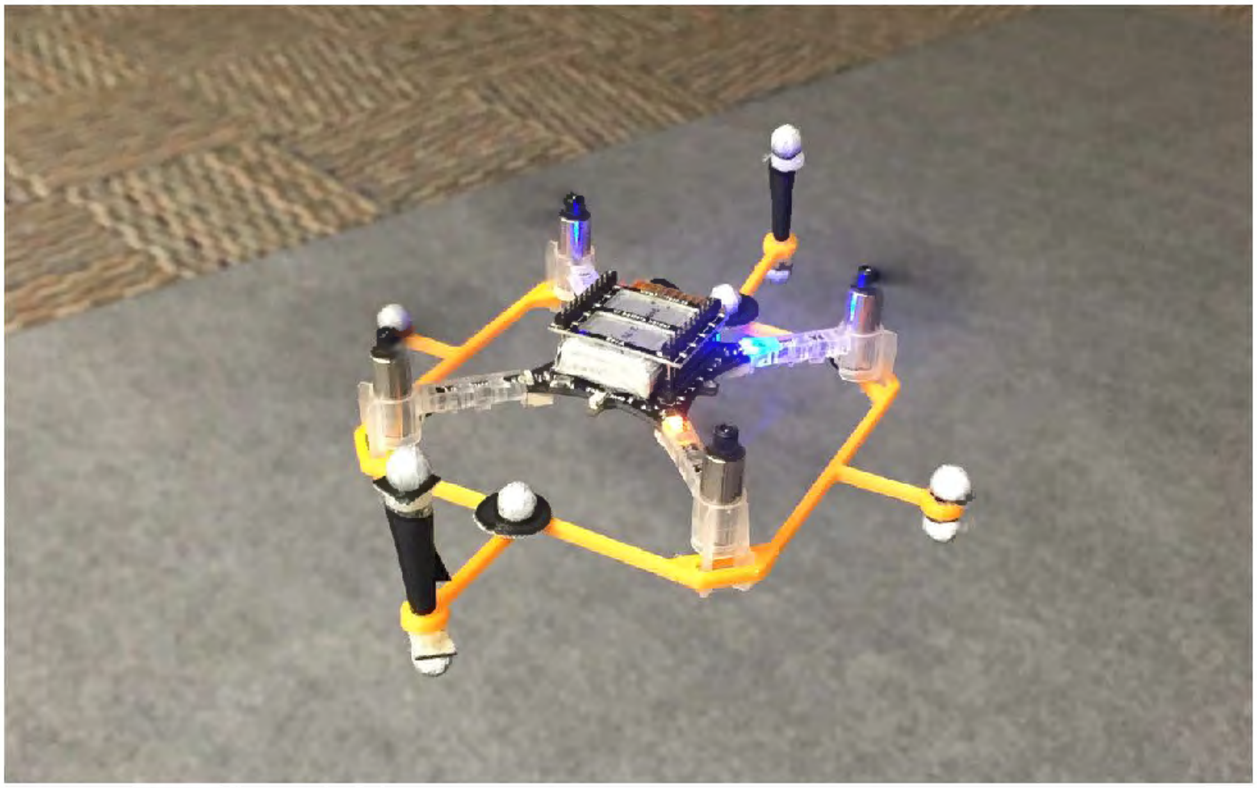


Figure 5.1: A picture of Crazyflie 2.0 quadrotor flying during one of our experiments.

developed by Bitcraze. The nano quadrotor nominally weighs 27g and has a motor to motor length of 65mm. Its small size, low cost, and robustness make it an ideal platform for testing new control paradigms. Recently it has been used to exemplify aggressive flight in cluttered environments and for human robot interaction research [188, 157]. Crazyflie is equipped with an on-board inertial measurement unit (IMU) that provides orientation and angular velocity measurements at 250 Hz. We use Crazyflie to collect the training data for the NN model, as well as to test the resulting controller.

5.4.1 Crazyflie Dynamics and Onboard Controller

The quadrotor system is modeled as a rigid body with a twelve dimensional state vector $x := \begin{bmatrix} p & v & \zeta & \omega \end{bmatrix}$, which includes the position $p = (p_x, p_y, p_z)$ in a North-East-Down inertial reference frame I, linear velocities $v = (v_x, v_y, v_z)$ in I, attitude (orientation) represented by Euler angles $\zeta = (\phi, \theta, \psi)$, and angular velocities $\omega = (\omega_x, \omega_y, \omega_z)$ expressed in the body-fixed coordinate frame B of the quadrotor[1]. The Euler angles parameterize the coordinate transformation from I to B with the standard *yaw-pitch-roll* convention, i.e. a rotation by

[1]Note that in this section we are overloading the notation θ to represent both the pitch angle of the quadrotor, as well as parameters of the DNN model. We clarify this distinction wherever unclear.

ψ about the z-axis in the inertial frame, followed by a rotation of θ about the y-axis of the body-fixed frame, and finally another rotation of ϕ about the x-axis in the new body-fixed frame. This is written compactly as

$$ {}^{B}_{I}R(\phi, \theta, \psi) = R_x(\phi)R_y(\theta)R_z(\psi), \tag{5.7}$$

where R_x,R_y, and R_z are basic 3×3 rotation matrices about their respective axes.

The system is controlled via four inputs $u := \begin{bmatrix} u_1 & u_2 & u_3 & u_4 \end{bmatrix}$, where u_1 is the thrust along the z-axis in B, and u_2, u_3 and u_4 are rolling, pitching, and yawing moments respectively, all in B[2]. The system evolves according to dynamics:

$$x_{t+1} = \begin{bmatrix} p_{t+1} \\ v_{t+1} \\ \zeta_{t+1} \\ \omega_{t+1} \end{bmatrix} = f(x_t, u_t; \theta) = f_{nom}(x_t, u_t) + f_{unknown}(x_t, u_t; \theta) \tag{5.8}$$

$$= \begin{bmatrix} p_t + \Delta T \cdot v_t \\ 0 \\ \zeta_t + \Delta T \cdot \hat{R}\omega_t \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ f_v(x_t, u_t; \theta_1) \\ 0 \\ f_\omega(x_t, u_t; \theta_2) \end{bmatrix}, \tag{5.9}$$

where the system model is parameterized by $\theta := (\theta_1, \theta_2)$, which will eventually represent the parameters of a DNN. ΔT represent a small discretization time step. The known nominal component of dynamics here simply represent the definition of linear and angular velocities – they are given by the derivatives of linear and angular displacements respectively. However, note that $\dot{\zeta} \neq \omega$ in general. $\dot{\zeta}$, or Euler rates as they are called, can be obtained by rotating the angular velocities to the inertial frame [45, 155]. The rotation matrix is given by:

$$\hat{R} = \begin{bmatrix} 1 & \sin\phi \tan\theta & \cos\phi \tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \frac{\sin\phi}{\cos\theta} & \frac{\cos\phi}{\cos\theta} \end{bmatrix}. \tag{5.10}$$

The unknown components in (5.8) are f_v and f_ω, the linear (or translational) and angular (or rotational) acceleration that the quadrotor undergoes, which we aim to approximate with a NN as a function of state, control, and model parameters. The system identification task for the quadrotor is thus to determine the NN parameters θ_1 (resp. θ_2), given observed values of f_v (resp. f_ω), x, and u.

Onboard PD Controller. To run any state-feedback control law on Crazyflie, we need the full state x. However, in practice, this information is obtained from different sensors which might run at different frequencies and hence the control update rate is limited by the

[2]These inputs are generated by varying the angular speeds of the four propellers, which map linearly to the inputs.

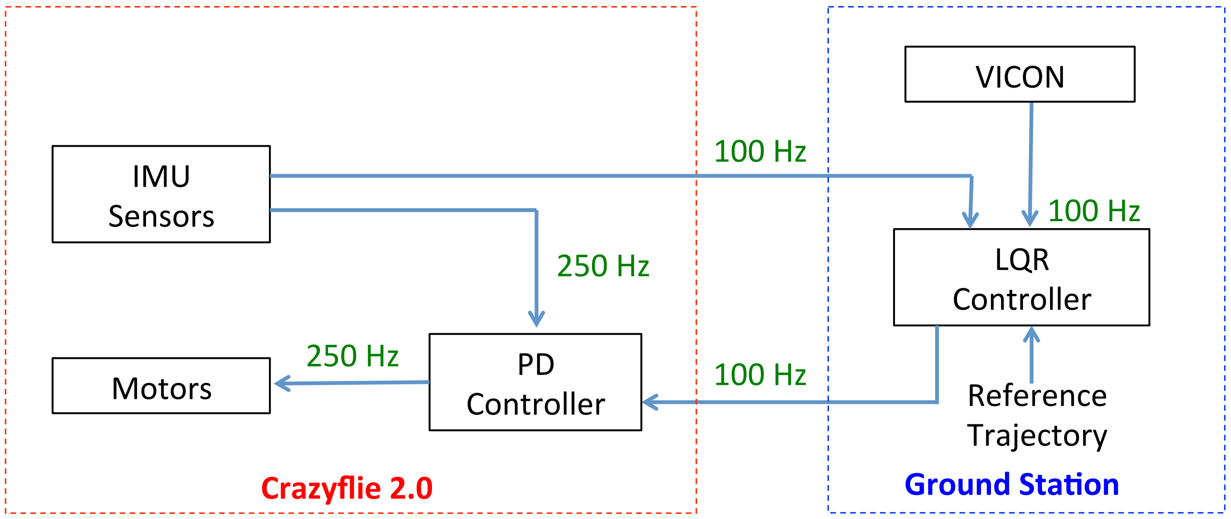


Figure 5.2: Control block diagram used to stabilize Crazyflie during experiments. At the ground station, LQR is running at 100Hz. On-board the Crazyflie, PD controller is running at 250Hz. Together they are able to stabilize the Crazyflie.

frequency of the slowest sensor. This frequency, however, might not be enough to effectively control the system, and a low-level controller is thus required in practice to control the system between the two updates of the state feedback loop. In this section, we provide more details about the low-level PD controller onboard Crazyflie to control the system between the feedback updates.

We use the on-board PD controller developed by [188], which takes into account only the angular position and angular velocities that are available at a higher frequency of 250 Hz from the onboard IMU sensor:

$$\begin{bmatrix} u_2 \\ u_3 \\ u_4 \end{bmatrix} = K_p \begin{bmatrix} \phi - \phi_{des} \\ \theta - \theta_{des} \\ \psi - \psi_{des} \end{bmatrix} + K_d \begin{bmatrix} \omega_x - \omega_{x,des} \\ \omega_y - \omega_{y,des} \\ \omega_z - \omega_{z,des} \end{bmatrix}, \tag{5.11}$$

where K_p and K_d are 3×3 matrices, $(\phi_{des}, \theta_{des}, \psi_{des})$ is the desired attitude, and $(\omega_{x,des}, \omega_{y,des}, \omega_{z,des})$ is the desired angular rate. This PD control law provides stabilization around a desired trajectory. Note that we also did the same augmentation on our system in (5.8) so that the new inputs to the system are now $\hat{u} := (u_1, \phi_{des}, \theta_{des}, \psi_{des}, \omega_{x,des}, \omega_{y,des}, \omega_{z,des})$, where mapping between inputs is given by (5.11). The control commands as well as the state feedback law are thus computed for the augmented system. The onboard PD controller is shown in left in our overall controller block diagram (Figure 5.2).

5.4.2 Data Collection

To collect data for training, we flew Crazyflie autonomously on a variety of trajectories, for example sinusoids in XY, XZ and YZ planes (but no yaw), and fixed position yaw-rotations, as well as manually on unstructured flights. For these flights, we recorded the state (x) and input ($\hat{u}$) data. For the autonomous flights during the training phase, we assume that a linear near-hover model of the quadrotor is available during the training time. An approximate *linear model* that accurately represents the quadrotor dynamics for small perturbations from the hover state is presented in [59, 45]. We use this linear model to design a linear feedback controller (LQR) to fly quadrotor on various trajectories. However, in general, the entire data can also be collected manually with experts flying the system, when such a linear model is not available.

A picture of Crazyflie flying during one of our experiments is shown in Figure 5.1. One of the autonomous flight videos during the data collection procedure can be found at YouTube[3]. For communication between the ground station and Crazyflie, we use the Robot Operating System (ROS) framework [252, 154]. In total there are 2400 seconds of flight time recorded, which correspond to $240,000$ $\Big((x,\hat{u}), \tilde{f}_{unknown}(x,\hat{u})\Big)$ data samples. Note that since we augment the system with a PD controller, we collect $\hat{u}$ during the flights, as opposed to u. We flew the Crazyflie in a VICON environment, a motion capture system, which allowed us to capture the position and velocity of the Crazyflie at 100Hz. We retrofit the Crazyflie with reflective markers to allow for accurate position and velocity estimation. Furthermore, Crazyflie is equipped with an on-board inertial measurement unit (IMU) that provides orientation and angular velocity measurements at 250 Hz. VICON and IMU together thus provide the 12 dimensional state of the system

5.4.3 Training Details

Neural Network Architecture. We use two three layer ReLU DNNs, one for each unknown dynamics components f_v and f_ω. Each NN consists of an input layer, one hidden layer, and an output layer. We next train the two NNs (denoted as NN1 and NN2 here on) using the collected data to learn the linear and angular acceleration components f_v and f_ω such that the MSE in (5.6) is minimized.

The input to NN1 is $(v, \omega, sin(\zeta), cos(\zeta), u_1)$ and is $(v, \omega, sin(\zeta), cos(\zeta), u_2, u_3, u_4)$ for NN2. Note that we do not include u_2, u_3, u_4 in the input to NN1 because our experiments indicate that including them as inputs result in over-fitting. Moreover, the physics of a quadrotor hints that the translational acceleration should not depend on these inputs [45]. The same argument holds for u_1 and NN2.

Note that we only feed the current state and input in the network, and not any information on the past states and inputs. Although providing the past state-input information will allow the NN to learn a more complex (and potentially more accurate) system dynamics

[3] Video link: https://www.youtube.com/watch?v=QREeZvHg0lQ

model, it will also make it harder to design a controller for the resultant dynamics. So a simple input structure is chosen to make sure that the NN can be effectively employed to design a feedback controller for the downstream task.

Data Preprocessing. Before training the NNs, we follow a few data pre-processing steps:

- Since we collect $\hat{u}$ (input to the PD-augmented system), we first derive u (input to the quadrotor system in (5.8)) from $\hat{u}$ using (5.11), and use it as the input to the NNs along with the state information. This is to make sure that the learned system dynamics are independent of the control scheme.
- Instead of providing orientation angles as the input to the NNs, we provide sines and cosines of the angles. This is to make sure that NNs can take into account the periodic nature of angles.
- We do not provide position as the input to any of the neural networks, as the translational and rotational accelerations should be position independent. We notice that including position in the NN inputs leads to overfitting.
- For each NN, we scale the observed outputs (for example, x, y, z components of translational acceleration) such that each of them has zero mean and unity standard deviation. This is to make sure that the NNs give equal weightage to MSE in the three components.

Training Hyperparameters. 60% of the total collected data was used for training, 25% was used for validation purposes and for tuning hyper parameters, and the rest was used for testing purposes. All weight and bias parameters were initially sampled from normal Gaussian distribution. For training the networks, we use the Neural Network Toolbox of MATLAB. We use the Resilient backpropagation learning algorithm. The learning rate, momentum constant, regularization factor and the number of hidden units were set at 0.01, 0.95, 0.1 and 100 respectively, but later tuned using the validation data. Overall, the learning algorithm makes about 100 passes through the data, and optimize the weights and biases to minimize the loss function. Once the training is complete, the optimal weights and biases are obtained, which can be substituted in (5.8) to get the full dynamics model.

5.4.4 Prediction Performance of the DNN Model

We now illustrate the performance of the learned models at predicting the next state given the current state and input. The (normalized) MSE numbers obtained for the training and the test data after learning f_v are 0.134 and 0.135 respectively, and that for f_ω are 0.341 and 0.344. Since the MSE numbers are very close for training and testing, it indicates that the NNs do an accurate prediction on the unseen data as well, meaning that our NNs are not overfitting on the training data. In Figure 5.3, we show the observed values and the predicted outputs of the trained NNs for roll and y accelerations. As evident from the

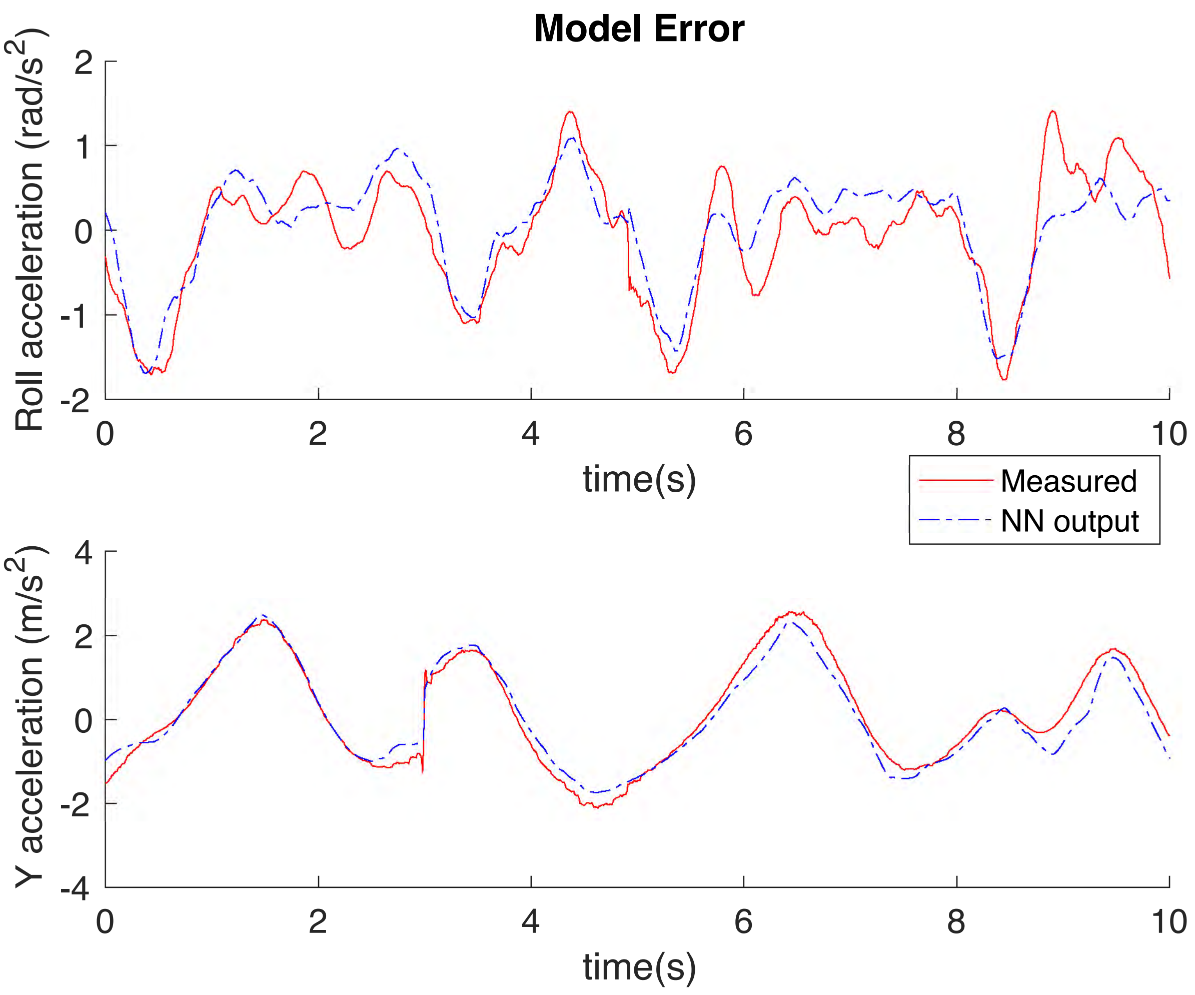


Figure 5.3: Observed and predicted values for the roll and y accelerations. The NNs are able to learn the acceleration models fairly accurately even with just the current state and input, indicating that the past states and inputs may not be required to learn the dynamics, and hence are avoided in this work to keep the control design simple.

figure, the NNs have been successfully able to learn the dynamics to a good accuracy. This indicates that a simple three-layer feed-forward NN structure used in this paper is sufficient to learn quadrotor dynamics to a good accuracy. Moreover, only the current state and input information is sufficient to learn the dynamics models, and hence past state and input information, which will potentially make the model as well as control design more complex, has not been used as an input to the NNs.

5.4.5 Controller Design Using Learned Dynamics

Once NN1 and NN2 are trained, the full quadrotor model is available through (5.8). In this section, our goal is to use this model to track a sinusoid-yaw trajectory, where quadrotor is undergoing a sinusoidal motion in the XY plane while yawing at the same time. Since the desired trajectory consists of a simultaneous translational and rotational movement, the learned NN models must capture the nonlinear couplings between these two movements to accurately track the trajectory.

Using the full model, we first compute a dynamically feasible reference that is as close as possible to the desired sinusoid-yaw trajectory (shown in Figure 5.4), using the sequential convex programming method. The reference trajectory is then tracked using a near-hover LQR controller along with a yaw rotation. Our experiment video can be found at YouTube[4].

Reference Trajectory Computation. Given a horizon N_H and a desired trajectory over that horizon $\mathbf{x}^d_{N_H} := \{x^d_0, x^d_1, \dots, x^d_{N_H}\}$, our goal is to find a control signal $\mathbf{u}_{N_H} := \{u_0, u_1, \dots, u_{N_H}\}$ that will achieve the desired trajectory when applied to the quadrotor model.

In most cases the desired trajectory may not be dynamically feasible, so no such control signal exists. Instead, we look for a dynamically feasible trajectory that is "as close as possible" to the desired trajectory. We thus want to solve the following optimization problem:

$$\begin{aligned} &\underset{\mathbf{x}_{N_H}, \mathbf{u}_{N_H}}{\text{argmin}} && \sum_{n=0}^{N_H} \|x_n - x^d_n\|_2 \\ &\text{s. t.} && x_{n+1} = f_{nom}(x_n, u_n) + f_{unknown}(x_n, u_n; \theta),\ n = 0, \dots, N_H - 1 \end{aligned} \tag{5.12}$$

In other words, we want to find the trajectory that minimizes the Euclidean distance to the desired trajectory, and the control that achieves such a trajectory. Since the NN output is nonlinear in general, $f_{unknown}$ is nonlinear; therefore, the above optimization problem is a non-convex problem. In this work, we use the sequential convex optimization (SCP) procedure proposed in [270] to solve this non-convex optimization problem. SCP solves a non-convex problem by repeatedly constructing a convex subproblem – an approximation to the problem around the current iterate $\mathbf{x}_{N_H}$. A local convex approximation of the non-convex constraints is added along with a penalty co-efficient in the objective function. This subproblem can be efficiently solved using convex solvers and used to generate a step $\Delta\mathbf{x}_{N_H}$ that makes progress on the original problem. The penalty co-efficient is then adjusted during the optimization to ensure that the constraint violation is driven to zero. For more details on the optimization procedure, we refer the interested readers to [270].

LQR Tracking Controller. Let us define the solution to (5.12) as $\mathbf{x}^*_{N_H}$ and $\mathbf{u}^*_{N_H}$. In practice, applying the control signal $\mathbf{u}^*_{N_H}$ could yield a trajectory that significantly differs from $\mathbf{x}^*_{N_H}$ due to (any remaining) mismatch between the model used in the optimization and

[4] Video link: https://www.youtube.com/watch?v=AeIfZbkjWPA

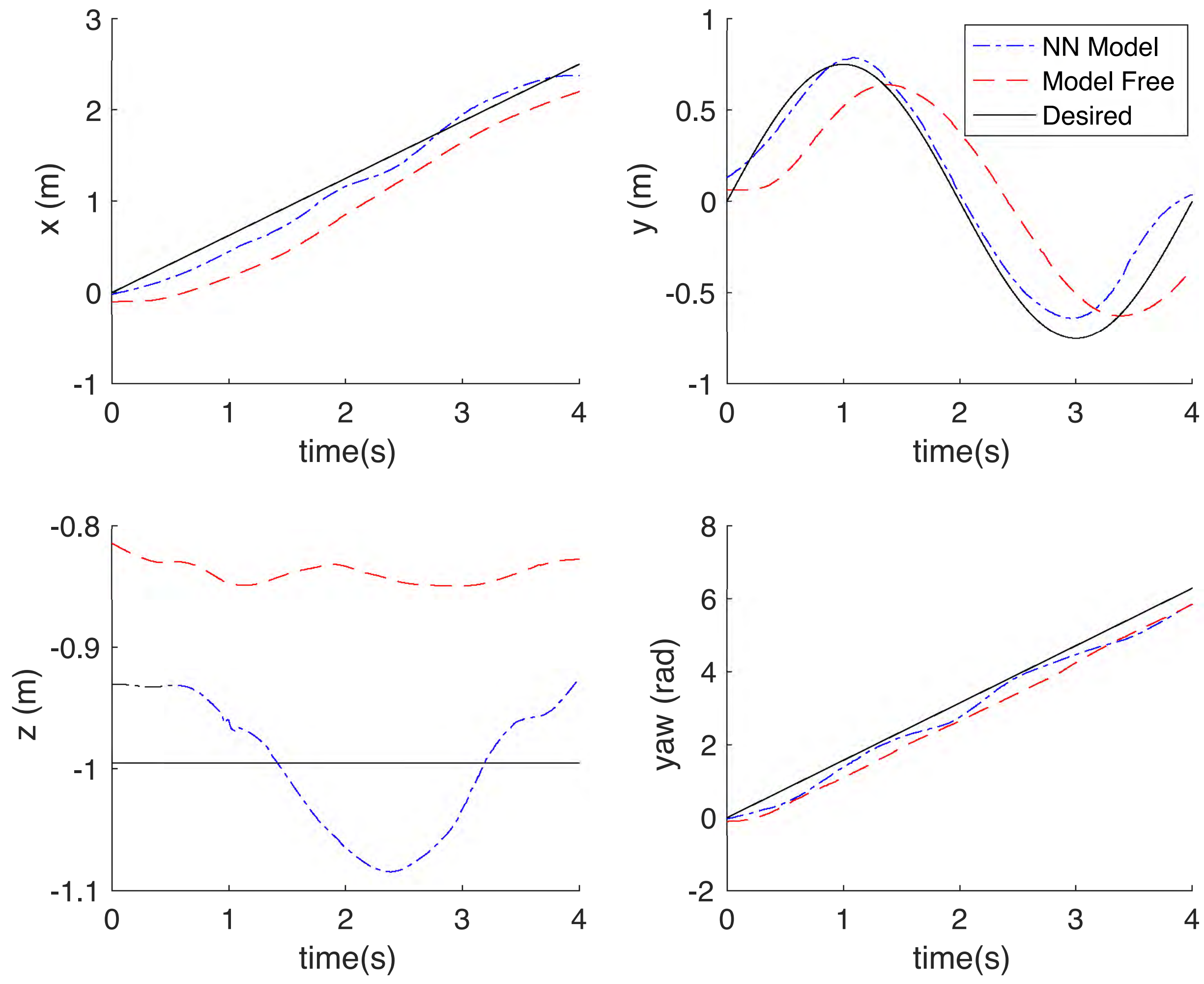


Figure 5.4: The reference, NN model and model-free trajectories obtained during the experiments. The NN model track the desired trajectory closely even though it involves both translational and rotational motion at the same time, which the NNs were not explicitly trained on, indicating the generalization capabilities of deep neural networks.

actual dynamics of the quadrotor and unmodeled disturbances. This can be mitigated via feedback.

To stabilize the quadrotors on a (feasible) reference trajectory, we use a LQR feedback controller designed for the near hover model of quadrotor, along with a reference rotation before applying the feedback to correct for a non-zero yaw. The feed-forward control command is given by the reference control trajectory, providing the closed loop controller

$$u_n = u_n^* + K(x_n - x_n^*), \tag{5.13}$$

where u_n^* and x_n^* are the reference control (feed-forward control) and reference state respectively. K is the feedback gain obtained using LQR.

Since the quadrotor hover dynamics are derived around zero yaw, they no longer accurately represent the dynamics of the quadrotor for non-zero yaw states. Consequently, the near hover model is no longer valid and hence state feedback law given in (5.13) will no longer be able to stabilize the system. One possible solution to mitigate this issue is to rotate the inertial frame at every time step to another inertial frame in which yaw is zero. In particular, let the state error at timestep n is given by $\bar{x}_n := (x_n - x_n^*)$. Also, let the yaw at time n be ψ_n. The rotation of the inertial frame is thus equivalent to rotating the error in the position and velocity vectors by a rotation matrix as follows:

$$\begin{bmatrix} \bar{p_x}^R \\ \bar{p_y}^R \end{bmatrix} = R_n \begin{bmatrix} \bar{p_x} \\ \bar{p_y} \end{bmatrix}, \quad \begin{bmatrix} \bar{v_x}^R \\ \bar{v_y}^R \end{bmatrix} = R_n \begin{bmatrix} \bar{v_x} \\ \bar{v_x} \end{bmatrix}, \tag{5.14}$$

where $\bar{p_x}$ and $\bar{p_y}$ is the error in x and y position respectively. $\bar{v_x}$ and $\bar{v_y}$ are similarly defined. The rotation matrix R is given by

$$R_n = \begin{bmatrix} cos(\psi_n) & sin(\psi_n) \\ -sin(\psi_n) & cos(\psi_n) \end{bmatrix}. \tag{5.15}$$

The corresponding rotated error vector is $\bar{x}_n^R := (\bar{p}_n^R, \bar{v}_n^R, \bar{\zeta}_n, \bar{\omega}_n)$. Our overall state feedback control is thus given by:

$$u_n = u_n^* + K\bar{x}_n^R. \tag{5.16}$$

Finally, we can use the feedback control in (5.16) along with the onboard PD controller to stabilize the Crazyflie around the reference trajectory.

Note that the state feedback control law in (5.16) is good at error correction only when an accurate open-loop state x_n^* and control u_n^* are provided. Since the open-loop control depends heavily on the system model, the tracking with (5.16) can only be as good as the system dynamics model itself. In particular, for sinusoid-yaw reference trajectories, the NN should be able to learn the couplings between translational and rotational motion for a good tracking.

Performance of the Controller Designed Using the Learned Dynamics. The desired trajectory and the obtained trajectory using the NN model along with the tracking controller in (5.16) are shown in Figure 5.4. As evident from the figure, the NN model trajectory is able to track the desired trajectory closely. This illustrates that:

- the trained NNs are able to generalize the dynamics beyond the training data. In particular, the NN models capture the nonlinear couplings between translational and rotational accelerations, and can be used to track the trajectories they were not trained on.
- even simple NN architectures, such as one used in this paper, have good generalization capabilities and can be used to control a quadrotor on complex trajectories.

From the results thus far it is not clear how much of the control performance is due to the feedforward signal derived from the NN model, since the LQR control may be correcting for model inconsistency. Therefore, we opted to fly Crazyflie using an LQR controller designed on the near-hover model of the quadrotor that was used during the training phase to fly Crazyflie autonomously. We label the results corresponding to this experiment as the 'model-free' trajectory. In Figure 5.5, we show the (absolute) tracking error for the NN model and

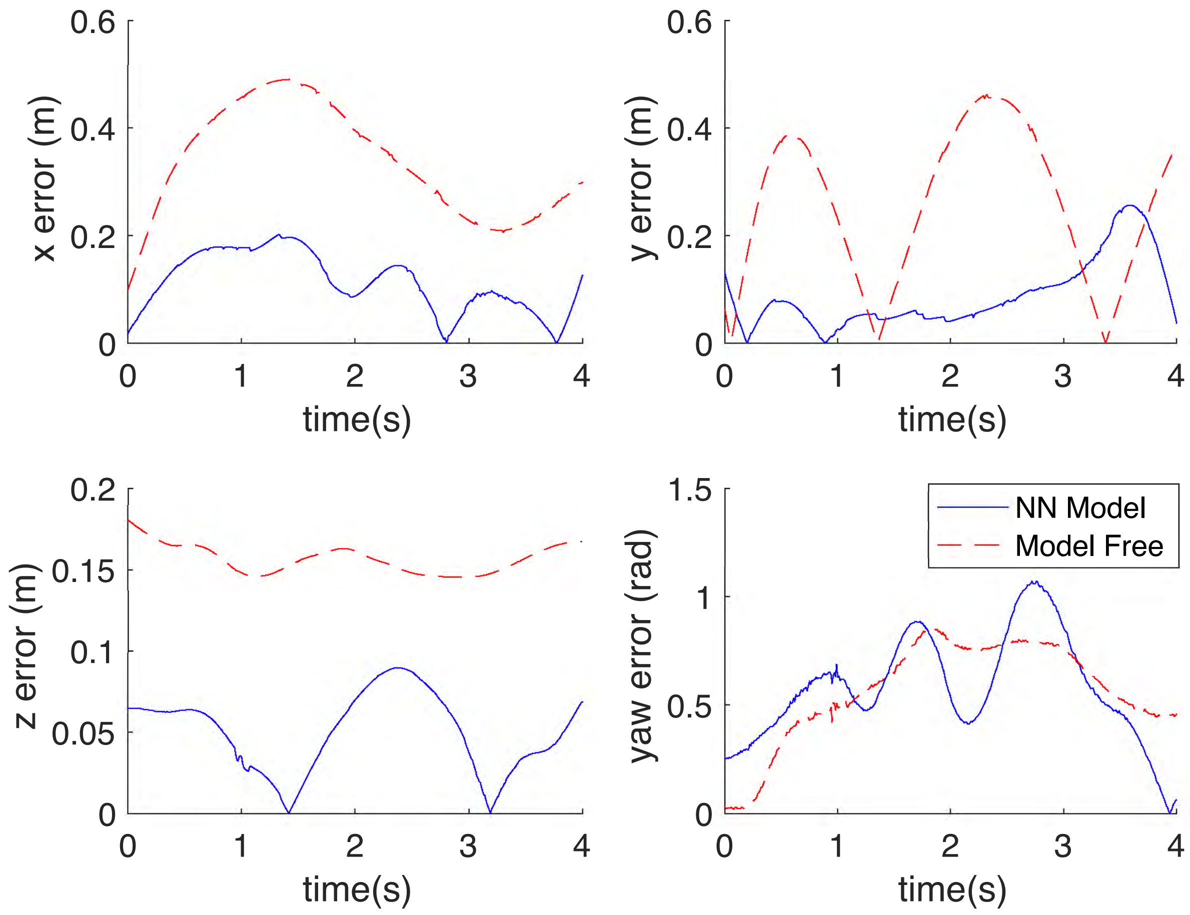


Figure 5.5: (Absolute) Tracking error for model-free and NN model trajectories. Model-free trajectory has a significantly higher tracking error compared to the NN model, especially in the translational motion, indicating that nonlinear coupling between translational and rotational motions should be taken into account while designing a controller, which in this work is captured by training a NN model that accurately represents the system dynamics.

model-free trajectories. The NN model has a significantly lower tracking error compared to the model-free trajectory, indicating that the feedforward control derived from the NN model results in better tracking of the desired trajectory.

5.5 Discussion

Even though DNNs are able to capture the unmodeled complex, nonlinear inaccuracies in dynamics, they also pose several practical challenges:

- We found it challenging to use traditional optimal control/MPC methods for DNN-based dynamics models. This is primarily for two reasons. First, the DNN-based models are inherently complex, especially when there are multiple hidden layers, making it challenging to solve optimization problems involving DNN-based dynamics constraints. Second, and perhaps more importantly, the DNN-based dynamics models do not reliably predict the gradients of the dynamics function $f_{unknown}$. This makes it challenging to use any optimal control method that relies on these gradients, for example for linearization of dynamics in iLQR or SQP.

- DNN-based dynamics models typically struggle with long horizon planning because, unlike GPs, DNNs do not provide an estimate of the accuracy of the learned dynamics model, leading it to confidently make inaccurate predictions. This inaccuracy in one-step prediction compounds over the planning horizon, leading to biased controllers. To overcome these challenges, Bayesian Neural Networks (BNNs) [124] as well as several approximations of BNNs that provide an estimate of uncertainty in the NN output have been considered [82]. An alternative is to learn a multi-step dynamics model instead of a single-step dynamics model, for example using Recurrent Neural Networks (RNNs).

- We noticed that using a nominal model and known information during the learning process were crucial for avoiding overfitting and ensuring data-efficient learning. For example, during the learning of the quadrotor model, we do not provide position as the input to any of the neural networks, as the translational and rotational accelerations should be position independent. Indeed, using position as an input to the DNN results in a significant overfitting, which otherwise might require significant more data to mitigate. Similarly, we do not learn the dynamics of position states (that are known very well, thanks to calculus), which further improve the sample complexity of the learning process.

5.6 Chapter Summary

As autonomous systems will operate in unstructured environments, they will inevitably experience additional external effects that are hard to model using first principles. In this chapter, we discuss how we can use deep neural networks to learn these dynamics inaccuracies directly using the data collected on the system. Our experiments indicate that even simple DNNs such as feed-forward networks can also have good generalization capabilities and can learn the dynamics to a good accuracy. More importantly, we demonstrate that the learned dynamics can be used effectively to control the system. We also discuss several practical challenges associated with DNN-based dynamics models.

Chapter 6

Learning for Unknown Dynamics Models: Direct Learning-Based Control

This chapter is based on the papers "Goal-Driven Dynamics Learning via Bayesian Optimization" [32], "Closed-Loop Model Selection for Kernel-Based Models Using Bayesian Optimization" [47], and "MBMF: Model-Based Priors for Model-Free Reinforcement Learning" [31] written in collaboration with Thomas Beckers, Roberto Calandra, Ted Xiao, Kurtland Chua, Sergey Levine, Sandra Hirche, and Claire Tomlin.

Given the system dynamics, optimal control schemes such as LQR, MPC, and feedback linearization can efficiently design a controller that maximizes a performance criterion. However, depending on the system complexity, it can be quite challenging to model its true dynamics using the first principles. In Chapter 5, we discussed how we can use DNNs to model the dynamic inaccuracies, and subsequently, leverage the learned model for controller design. However, for a *given* task, a globally accurate dynamics model is not always required to design a controller. Often, partial knowledge of the dynamics is sufficient, e.g., for trajectory tracking purposes a local linearization of a non-linear system is often sufficient.

In this chapter, we argue that for complex systems, if we are interested in designing a controller for a *specific* task, it might be preferable to adapt the controller design process to learn a system dynamics model that is sufficient to achieve a desired performance on that task, rather than learning a globally accurate dynamics model. This approach to the controller design is somewhere between indirect and direct learning-based control methods. It is not indirect because we are not explicitly optimizing the model for minimizing the prediction error. It is also not direct because we are not directly optimizing the controller. However, it is closer to a direct approach than an indirect approach because similar to the task-specific nature of the learned controller in direct approaches, the learned model is task-specific and optimized to maximize the resulting controller performance for the task at hand.

This direct approach to controller design has two advantages over indirect approaches, which first learn an accurate dynamics model and subsequently use it for controller design. First, if the actual dynamics do not belong to the class of models that is being considered during the learning process, the learned model might actually deteriorate the control performance rather than improving it. This is because the learning process will try to balance-off the prediction error of the model in different parts of the state-space so as to achieve an overall low prediction error. However, a "good" model (good in terms of prediction error) in the parts of the state space that concern the task at hand and "bad" in others might lead to an overall better control performance on the task, compared to an average model everywhere. In other words, for limited capacity models, it might be desirable to focus the model representation capabilities on the relevant parts of the state space. Second, even when the model class is rich, it may not be easy to design a controller for the learned model. For example, we discussed a few challenges associated with designing a controller on a highly nonlinear, DNN-based dynamics model in Chapter 5. Thus, using simple models that are only locally accurate but enable efficient controller design might be more useful in certain scenarios, especially when planning needs to be performed in real-time.

6.1 Related Work

System Identification (SysID), Model-Based RL, and Indirect Adaptive Control. Traditional system identification approaches are divided into two stages: 1) creating a dynamics model by minimizing some prediction error (e.g., using least squares) 2) using this dynamics model to generate an appropriate controller. Similar approaches have also been studied under the banner of model-based RL. In these approaches, modeling the dynamics can be considered an offline process as there is no information flow between the two design stages.

SysID and model-based RL methods can also be online, and have been studied extensively both in control and RL commuities, under the banner of indirect adaptive control and online (or on-policy) model-based RL approaches respectively[1]. In online methods, the dynamics model is instead iteratively updated using the new data collected by evaluating the controller (see [98, 94, 235, 233, 269, 242] for representative references from the RL community and [24, 142, 84, 229, 264] for references from the adaptive control community). Both for the online and the offline cases, creating a dynamics model based only on minimizing the prediction error can introduce sufficient inaccuracies that might lead to suboptimal control performance [162, 98, 26, 7, 129, 152]. In particular, if the model is inaccurate (e.g., due to intrinsic limitations of the models or the compounding of the inaccuracies over trajectory propagation) the expected cost might be wrong, even if the cost has been measured for the considered policy, leading to a sub-optimal control performance. This issue is often referred to as *model bias* [93]. Using machine learning techniques, such as

[1]For more details on these approaches and connections between them, we refer the interested readers to Sec. 5.1 in Chapter 5.

Gaussian processes, does not alleviate this issue [233]. Instead, authors in [162] proposed to optimize the dynamics model directly with respect to the controller performance, but since the dynamics model is optimized offline, the resultant model is not necessarily optimal for the actual system.

To mitigate the issues of the poor control performance with indirect methods, previous studies have also considered tuning the cost function (or performance criterion) in an online fashion. For example, in [208, 294, 258], the authors tuned the penalty matrices in an LQR problem for performance improvement using Bayesian Optimization (BO). Although interesting results emerge from these studies, tuning penalty matrices may not achieve the desired performance when a sufficiently accurate system dynamics model is not available.

Model-Free RL and Direct Adaptive Control. A natural way to optimize the control performance for a specific task is through model-free RL methods or direct adaptive control methods, wherein controller parameters are directly optimized to improve the control performance.

In [67], the authors directly learn the parameters of a linear feedback controller using BO. However, a typical controller might be non-linear and can contain hundreds of parameters; it is not feasible to optimize such high-dimensional controllers using BO [276]. Other model-free approaches such as Deep-Q learning [297], TRPO [271], and PPO [272] are very effective at learning complex, high-dimensional policies, but the convergence might require millions of trials and often lack the robustness of the model-based controllers [254].

Another model-free RL approach that has shown a lot of promise in data-efficient learning of a good controller is apprenticeship learning. In apprenticeship learning, a system is provided with expert trajectories that successfully complete the task, and a controller is learned to mimic the expert. [6] presents an apprenticeship learning approach to successfully perform advanced aerobatics on a helicopter under autonomous control. One limitation of the apprenticeship learning approach is that it requires expert trajectories, which may not be available.

A task-specific approach that is widely popular in adaptive control community is Iterative Learning Control (ILC), particularly because of its impressive performance within a handful of trials. ILC is a method of improving the tracking control using data for systems that work in a repetitive mode [309], [148]. Repetition allows the system to improve tracking accuracy from repetition to repetition, in effect learning the required input needed to track the reference exactly. The learning process uses information from previous repetitions to improve the control signal ultimately enabling a suitable control action can be found iteratively. However, ILC often requires *a priori* known, good nominal tracking controller, which can be challenging to obtain.

Combining Model-Based and Model-Free RL. Several prior works have also sought to combine model-based (MB) and model-free (MF) reinforcement learning, typically with the aim of accelerating MF learning while minimizing the effects of model bias on the final policy. These approaches range from using models for generating synthetic experience for MF approaches to reduce their data requirements [283, 143, 147] to produce a good initialization for the MF component [111, 230, 70, 304]. However, we still lack principled approaches that combine MB and MF approaches into a single RL method that does not inherit the inaccuracies of a model, yet is able to use the model structure across the entire policy space.

Contributions and Chapter Organization

In this chapter, we propose Dynamics Optimization via Bayesian Optimization (aDOBO), a Bayesian Optimization (BO) based active learning framework to learn the dynamics model that achieves the best performance for a given task based on the performance observed in experiments on the physical system (Section 6.3). This active learning framework takes into account all past experiments and suggests the next experiment in order to learn the most about the relationship between the performance criterion and the model parameters. Specifically, we use BO to optimize the dynamics model with respect to the desired task, where the dynamics model is updated after every experiment so as to maximize the performance on the physical system. This procedure corresponds to optimizing a *linear* dynamics model with the purpose of maximizing the performance of the final controller. Hence, unlike traditional system identification approaches, it does not necessarily correspond to finding the most accurate dynamics model, but rather the model yielding the best controller performance when provided to the optimal control method used. Moreover, since we learn a linear dynamics model, the planning can be done very efficiently. Since aDOBO does not aim at directly learning the controller, it is agnostic to the dimensionality of the controller. It can leverage the low-dimensional structure of the dynamics to optimize the high-dimensional controllers. Therefore, it bypass the typical data-inefficiency issues associated with direct learning-based control methods. To overcome the limited capacity of linear models, we also demonstrate how aDOBO can be extended to a richer class of models, such as Gaussian Processes (Section 6.4). To the best of our knowledge, aDOBO the first method that optimizes a dynamics model to maximize the control performance on the actual system.

Finally, we propose an algorithm that combines the complimentary advantages of indirect and direct learning-based control approaches, such as data-efficiency of indirect approaches and high performance of direct approaches, which overcoming their respective limitations, such as model bias of indirect approaches and data-inefficiency of direct approaches (Section 6.5).

6.2 Problem Formulation

We recall the problem setup from Chapter 5. Consider a general, time-invariant, non-linear dynamical system

$$\begin{aligned} x_{t+1} &= f(x_t, u_t), && (6.1)\\ &= f_{nom}(x_t, u_t) + f_{unknown}(x_t, u_t), && (6.2)\end{aligned}$$

where f_{nom} is the known nominal component of the dynamics, and $f_{unknown}$ is the unknown component. When no information about system dynamics is available beforehand, $f_{nom} = 0$.

Given an initial state x_0, the goal is to solve an optimal control problem for dynamics in (5.1), i.e., to design a controller such that it minimizes the following cost function:

$$\begin{aligned} J_t(x_t, \mathbf{u}_t) =& \sum_{k=t}^{T-1} L(x_k, u_k, k) + M(x_T), \\ \text{subject to } \; x_{k+1} =& f(x_k, u_k), \end{aligned} \tag{6.3}$$

where $\mathbf{u}_t \equiv [u_t, u_{t+1}, \dots, u_{T-1}]$ represent the sequence of control over the time horizon $[0, T]$. One of the key challenges in designing such a controller is the modeling of the unknown system dynamics in (6.1).

In this work, we model the unknown dynamics component in (6.1) as a linear time-invariant (LTI) system with system matrices (A_θ, B_θ). The system matrices are parameterized by $\theta \in \mathcal{M} \subseteq \mathbb{R}^d$, which is to be varied during the learning procedure.

For a given θ and the current system state x_k, let $\pi_k(x_k, \theta)$ denote the optimal control sequence for the system with *linear* unknown component (A_θ, B_θ) for the horizon $\{k, k+1, \dots, T\}$, i.e.,

$$\begin{aligned} \pi_k(x_k, \theta) := \bar{\mathbf{u}}_k =& \arg\min_{\mathbf{u}_k} J_k(x_k, \mathbf{u}_k)\,, \\ \text{subject to } \; x_{j+1} =& f_{nom}(x_j, u_j) + A_\theta x_j + B_\theta u_j. \end{aligned} \tag{6.4}$$

The key difference between (6.3) and (6.4) is that the controller is designed for the parameterized linear model as opposed to the true system. As θ is varied, different matrix pairs (A_θ, B_θ) are obtained, which result in different controllers $\pi(\cdot, \theta)$. Our aim is to find, among all linear models, the linear model $(A_{\theta^*}, B_{\theta^*})$ whose controller $\pi(\cdot, \theta^*)$ minimizes J_0 (ideally achieves J_0^*) for the *actual system*, i.e.,

$$\begin{aligned} \theta^* =& \arg\min_{\theta \in \mathcal{M}} J_0(x_0, \mathbf{u}_0)\,, \\ \text{subject to } \; x_{k+1} =& f(x_k, u_k)\,, \quad u_k = \pi_k^1(x_k, \theta), \end{aligned} \tag{6.5}$$

where $\pi_k^1(x_k, \theta)$ denotes the 1st control in the sequence $\pi_k(x_k, \theta)$. To make the dependence on θ explicit, we refer to J_0 in (6.5) as $J(\theta)$ here on. Note that $(A_{\theta^*}, B_{\theta^*})$ in (6.5) may not correspond to an actual linearization of the system, but simply to the linear model that gives the best performance on the actual system when its optimal controller is applied in a *closed-loop* fashion on the actual physical plant.

We choose LTI modeling to reduce the number of parameters used to represent the system, and make the dynamics learning process data efficient. Linear modeling also allows to efficiently design the controller in (6.4) for general cost functions (e.g., using MPC for any convex cost J). In general, the effectiveness of linear modeling depends on both the system and the control objective. If f is linear, a linear model is trivially sufficient for any control objective. If f is non-linear, a linear model may not be sufficient for all control tasks;

however, for regulation and trajectory tracking tasks, a linear model is often adequate, as will illustrate later in this chapter. A linear parameterization is also used in adaptive control for similar reasons [264]. Nevertheless, we will demonstrate how the proposed framework can easily be extended to other dynamics models, such as Gaussian Process-based dynamics models in Sec. 6.4.

Since f is unknown, the shape of the cost function, $J(\theta)$, in (6.5) is unknown. The cost is thus evaluated empirically in each experiment, which is often expensive as it involves conducting an experiment. Thus, the goal is to solve the optimization problem in (6.5) with as few evaluations as possible. In this work, we do so via Bayesian Optimization (BO). We refer the reader to Section 2.6 for a quick overview of Gaussian Processes and BO.

6.3 Goal-Driven Dynamics Learning (aDOBO)

This section presents the technical details of aDOBO, a novel framework for Dynamics Optimization via Bayesian Optimization (BO) for maximizing the resultant controller performance. In this work, $\theta \in \mathbb{R}^{n_x(n_x+n_u)}$, i.e., each dimension in θ corresponds to an entry of the A_θ or B_θ matrices. This parameterization is chosen for simplicity, but other parameterizations can easily be used.

Since the function $J(\theta)$ in (6.5) is unknown a priori, we use nonparametric Gaussian Process (GP) model to approximate it over its domain $\mathcal{M}$ (see Sec. 2.6.1 for more details on GP). This GP model is used by BO to optimize (A_θ, B_θ). Our approach is illustrated in Figure 6.1 and summarized in Algorithm 5.

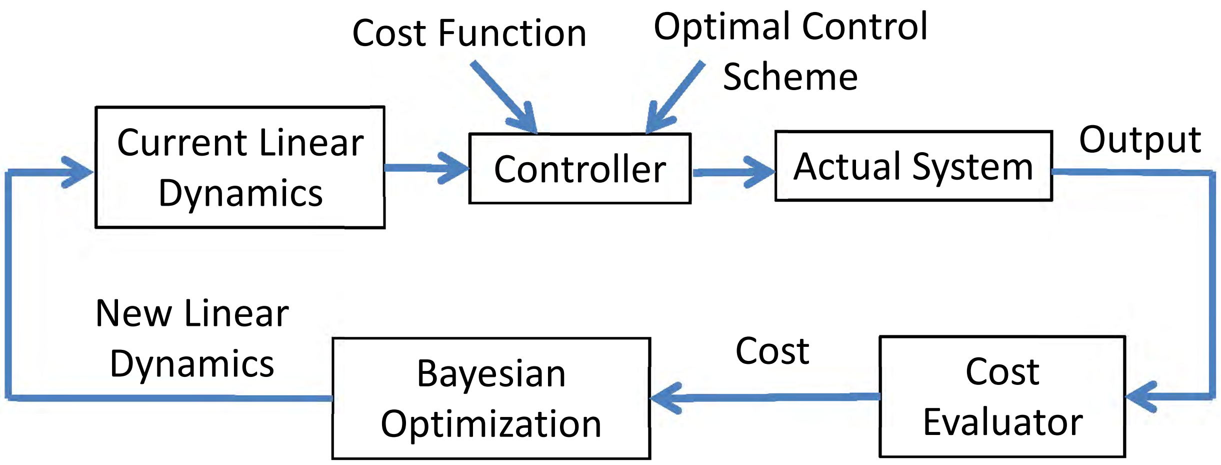


Figure 6.1: aDOBO: A Bayesian optimization-based active learning framework for optimizing the dynamics model for a given cost function, directly based on the observed cost values.

Given an initial state of the system x_0 and the current model of the unknown dynamics component $(A_{\theta'}, B_{\theta'})$, we design an optimal control sequence $\pi_0(x_0, \theta')$ that minimizes the

Algorithm 5: aDOBO algorithm for learning a task (or goal) driven dynamics model.

```
1  $\mathcal{D} \longleftarrow$ if available: $\{\theta, J\theta\}$
2  Prior $\longleftarrow$ if available: Prior of the GP hyperparameters
3  Initialize GP with $\mathcal{D}$
4  while optimize do
5      Find $\theta^* = \arg\min_\theta \alpha(\theta)$;   $\theta' \longleftarrow \theta^*$
6      $\mathbf{x}_0 = \{\}, \mathbf{u}_0 = \{\}$
7      for $i = 0 : T - 1$ do
8          Given $x_i$ and $(A_{\theta'}, B_{\theta'})$, compute $\pi_i(x_i, \theta')$
9          Apply $\pi_i^1(x_i, \theta')$ on the real system and measure $x_{i+1}$
10         $\mathbf{x}_0 \longleftarrow (\mathbf{x}_0, x_{i+1})$
11         $\mathbf{u}_0 \longleftarrow (\mathbf{u}_0, \pi_i^1(x_i, \theta'))$
12     Evaluate $J(\theta') := J_0(\mathbf{x}_0, \mathbf{u}_0)$ using (6.3)
13     Update GP and $\mathcal{D}$ with $\{\theta', J(\theta')\}$
```

cost function $J_0(x_0, \mathbf{u}_0)$, i.e., we solve the optimal control problem in (6.4). The first control of this control sequence is applied on the *actual system* and the next state x_1 is measured. We then similarly compute $\pi_1(x_1, \theta')$ starting at x_1, apply the first control in the obtained control sequence, measure x_2, and so on until we get x_T. Once the state and control trajectories $\mathbf{x}_0$ and $\mathbf{u}_0$ are obtained, we compute the true performance of $\mathbf{u}_0$ on the actual system by analytically computing $J_0(x_0, \mathbf{u}_0)$ using (6.3). We denote this cost by $J(\theta')$ for simplicity. We next update the GP based on the collected data sample $\{\theta', J(\theta')\}$. Finally, we compute θ^* that minimizes the corresponding acquisition function $\alpha(\theta)$ and repeat the process for $(A_{\theta^*}, B_{\theta^*})$.

Intuitively, aDOBO directly learns the shape of the cost function $J(\theta)$ as a function of linearizations (A_θ, B_θ). Instead of learning the global shape of this function through random queries, it analyzes the performance of all the past evaluations and by optimizing the acquisition function, generates the next query that provides the maximum information about the minima of the cost function. This direct *minima-seeking* behavior based on the *actual observed performance* ensures that our approach is data-efficient. Thus, in the space of all linearizations, we efficiently and directly search for the linearization whose corresponding controller minimizes J_0 on the actual system.

Note that the GP in our algorithm can be initialized with dynamics models whose controllers are known to perform well on the actual system. This generally leads to a faster convergence. For example, when a good linearization of the unknown dynamics components is available, it can be used to initialize $\mathcal{D}$. When no information is known about the unknown component a priori, the initial models are queried randomly. Finally, note that aDOBO can also be used when the real system is stochastic. In this case, aDOBO will minimize the expected cost.

6.3.1 Numerical Simulations

In this section, we present some simulation results on the performance of the proposed method for controller design.

6.3.1.1 Dubins Car System

For the first simulation, we consider a three dimensional non-linear Dubins car whose dynamics are given as

$$\begin{aligned} x_{k+1} &= x_k + \Delta T v_k \cos\theta_k \,, \\ y_{k+1} &= y_k + \Delta T v_k \sin\theta_k \,, \\ \theta_{k+1} &= \theta_k + \Delta T \omega_k \,, \end{aligned} \tag{6.6}$$

where (x_k, y_k, θ_k) is the state of system, $p_k = (x_k, y_k)$ is the position, θ_k is the heading, v_k is the speed, and ω_k is the turn rate at time k. The input (control) to the system is $u_k := (v_k, \omega_k)$. ΔT represents the sampling time and is chosen to be 0.1s in our simulations. Our goal is to design a controller that steers the system to the equilibrium point $x^* = 0, u^* = 0$ starting from the state $x_0 := (1.5, 1, \pi/2)$. In particular, we want to minimize the cost function

$$J_0(x_0, \mathbf{u}_0) = \sum_{k=0}^{T-1} \left(x_k^T Q x_k + u_k^T R u_k \right) + x_T^T Q_f x_T \,. \tag{6.7}$$

We choose $T = 30$. Q, Q_f and R are all chosen as identity matrices of appropriate sizes. We also assume that the dynamics model in 6.6 are not known beforehand and $f_{nom} = 0$; hence, we cannot directly design a controller to steer the system to the desired equilibrium. Instead, we use aDOBO to find a linearization of dynamics in (6.6) that minimizes the cost function in (6.7), directly from the experimental data.

In particular, we represent the system in (6.6) by a parameterized linear system $x_{k+1} = A_\theta x_k + B_\theta u_k$, design a controller for this system. The resulting controller is applied in closed-loop on the actual system and performance is recorded. Based on the observed performance, BO suggests a new linearization and the process is repeated. Since the cost function is quadratic in this case, the optimal control problem for a particular θ is an LQR problem, and can be solved efficiently.

For BO, we use the MATLAB library *BayesOpt* [210]. Since there are 3 states and 2 inputs, we learn 15 parameters in total, one corresponding to each entry of the A_θ and B_θ matrices. The bounds on the parameters are chosen randomly as $\mathcal{M} = [-2, 2]^{15}$. As acquisition function, we use EI (see eq. (2.96)). Since no information is assumed to be known about the system, the GP was initialized with a random θ. We also warp the cost function J using the *log* function before passing it to BO. Warping makes the cost function smoother while maintaining its monotonic properties, which makes the sampling process in BO more efficient and leads to a faster convergence.

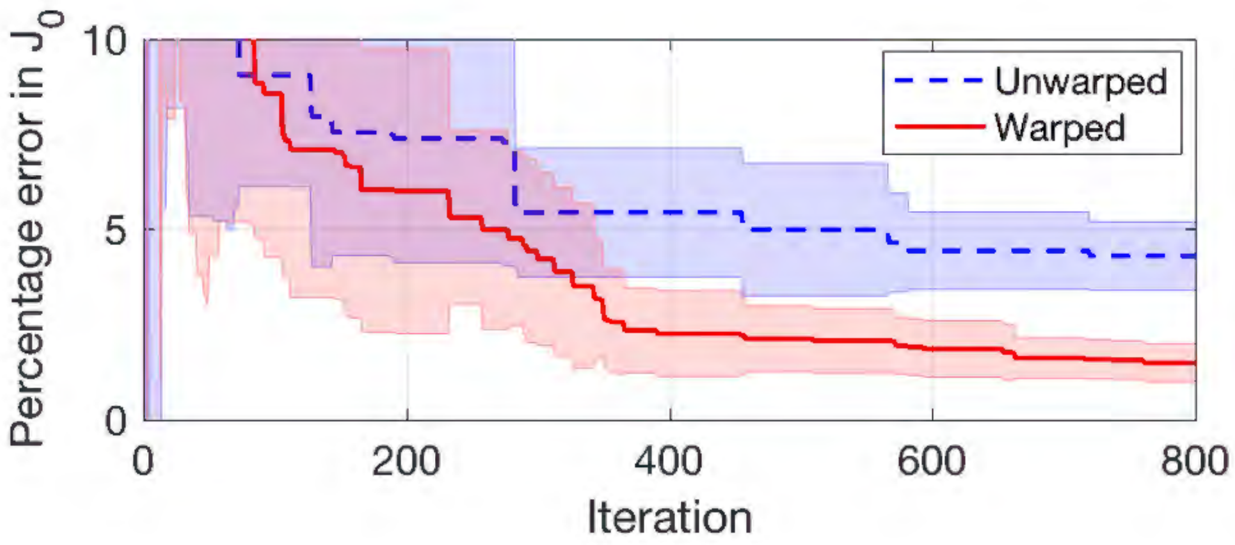


Figure 6.2: Dubins car: mean and standard deviation of η during the learning process (over 10 trials). aDOBO reaches within the 10% of the optimal cost in just 100 iterations, starting from a random dynamics model. Using a log warping on the cost function further accelerates the learning.

For comparison, we compute the true optimal controller that minimizes (6.7) subject to the dynamics in (6.6) using the non-linear solver *fmincon* in MATLAB to get the minimum achievable cost J_0^* across all controllers. We use the percentage error between the true optimal cost J_0^* and the cost achieved by aDOBO as our comparison metric in this work

$$\eta_n = 100 \times \frac{(J_0^* - J(\theta_n))}{J_0^*}, \tag{6.8}$$

where $J(\theta_n)$ is the best cost achieved by aDOBO by iteration n. In Fig. 6.2, we plot η_n for Dubins car. As learning progresses, aDOBO gathers more and more information about the minimum of J_0 and reaches within 10% of J_0^* in just 100 iterations, demonstrating its effectiveness in designing a controller for an unknown system just from the experimental data. Fig. 6.2 also highlights the effect of warping in BO. A well warped function converges faster to the optimal performance.

We also compared the control and state trajectories obtained from the learned controller with the optimal control and state trajectories. As shown in Fig. 6.3, the learned system matrices not only achieve the optimal cost, but also follow the optimal state and control trajectories very closely. Even though the trajectories are very close to each other for the true system and its learned linearization, this linearization may not correspond to any *actual* linearization of the system. The next simulation illustrates this key property of aDOBO more clearly.

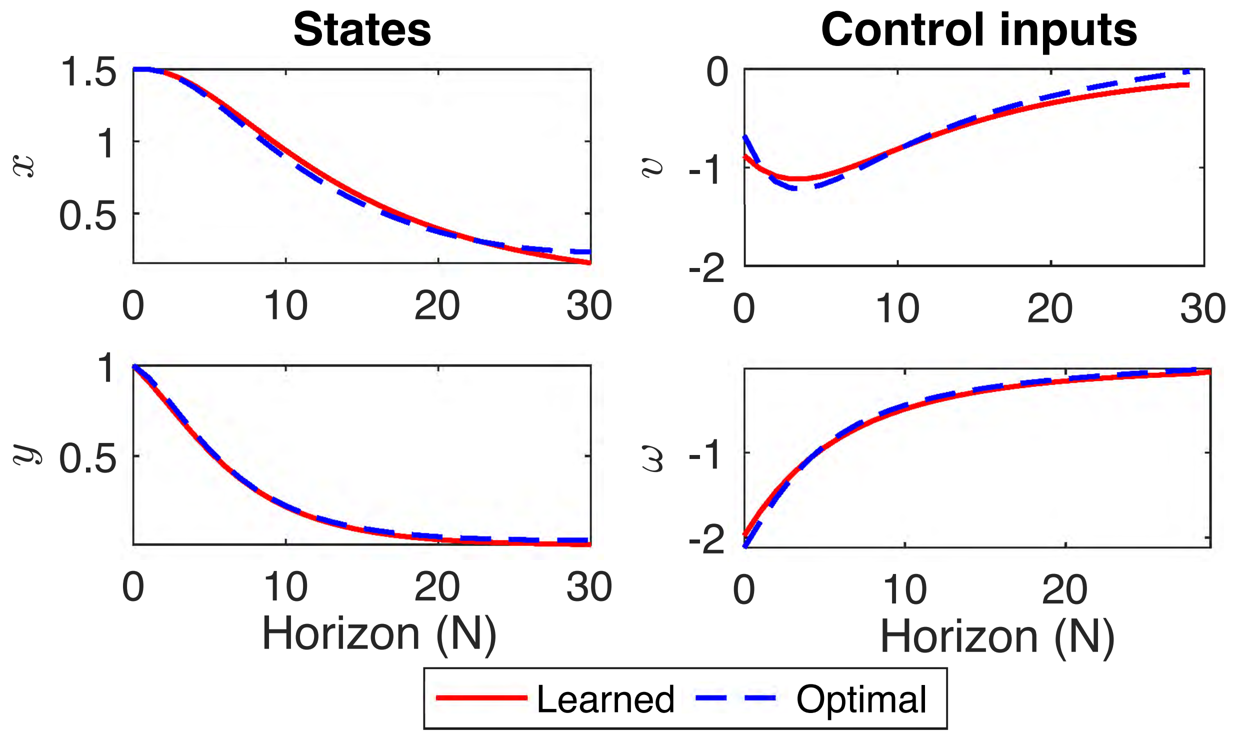


Figure 6.3: Dubins car: state and control trajectories for the learned and the true system. The two trajectories are very similar, indicating that the learned dynamics model represents the system behavior accurately around the desired state.

6.3.1.2 A Simple 1D Linear System

For this simulation, we consider a simple 1D linear system

$$x_{k+1} = x_k + u_k \,, \tag{6.9}$$

where x_k and u_k are the state and the input of the system at time k. Although the dynamics model is very simple, it illustrates some key insights about the proposed method. Our goal is to design a controller that minimizes (6.7) starting from the state $x_0 = 1$. We choose $T = 30$ and $R = Q = Q_f = 1$. We assume that the dynamics are unknown and use aDOBO to learn the dynamics, where $\theta := (\theta_1, \theta_2) \in \mathbb{R}^2$ are the dynamics parameters to be learned.

The learning process converges in 45 iterations to the true optimal performance ($J_0^* = 1.61$), which is computed using LQR on the real system. The converged parameters are $\theta_1 = 1.69$ and $\theta_2 = 2.45$, which are vastly different from the true parameters $\theta_1 = 1$ and $\theta_2 = 1$, even though the actual system is a linear system. To understand this, we plot the cost obtained on the true system J_0 as a function of linearization parameters (θ_1, θ_2) in Fig. 6.4. Since the performances of the two sets of parameters are very close to each other, a direct performance based learning process (e.g., aDOBO) cannot distinguish between them and both sets are equally optimal for it. More generally, a wide range of parameters lead to

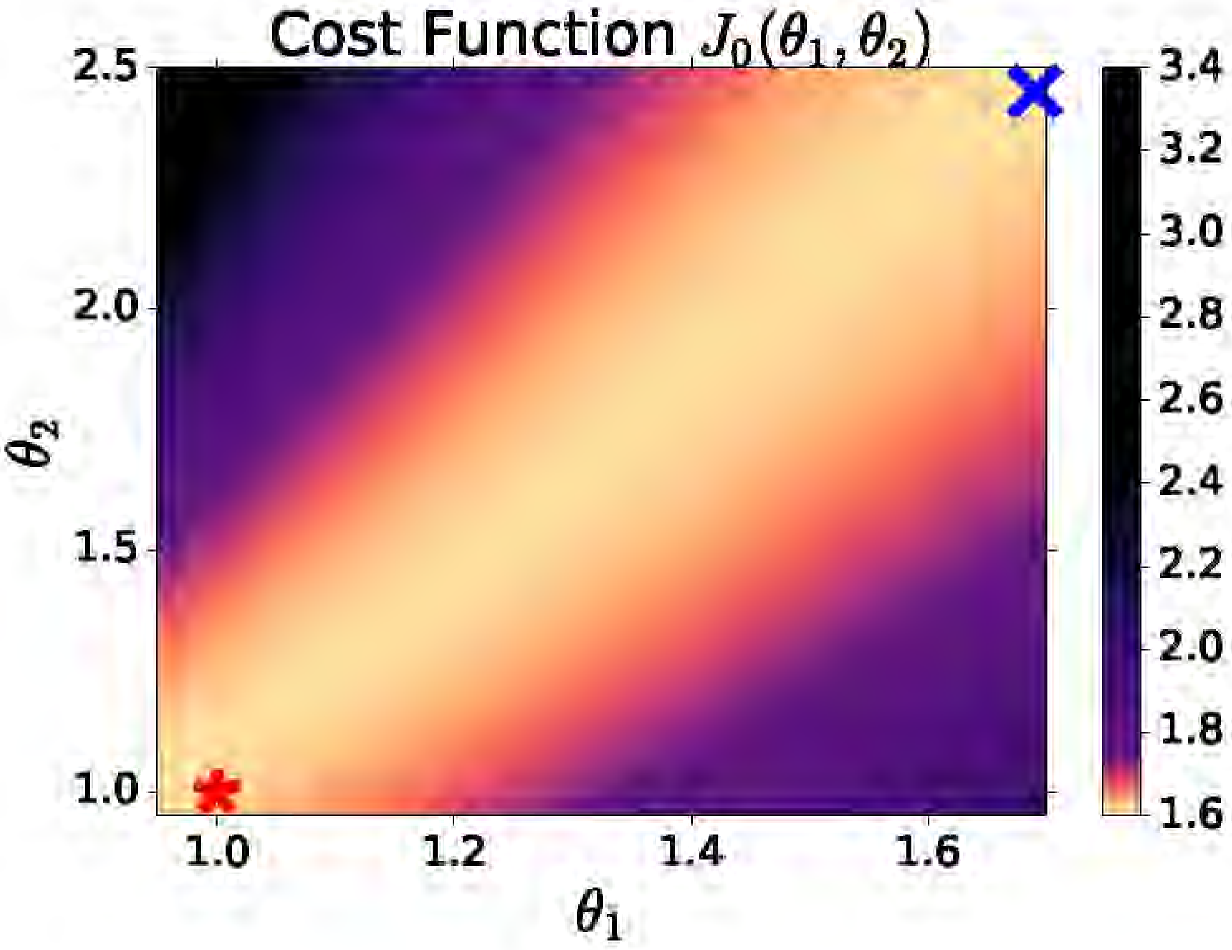


Figure 6.4: Cost of the actual system in (6.9) as a function of the linearization parameters (θ_1, θ_2). The parameters obtained by aDOBO (the pink X) yield to performance very close to the true system parameters (the green $*$). Note that aDOBO does not necessarily converge to the true parameters.

similar performance on the actual system. Hence, we can expect the proposed approach to recover the optimal controller and the actual state trajectories, but not necessarily the true dynamics or its true linearization. This simulation also suggests that the true dynamics of the system may not even be required as far as the control performance is concerned.

6.3.1.3 Cart-pole System

We next apply aDOBO to a cart-pole system in Fig. 6.5

$$\begin{aligned}(M+m)\ddot{x} - ml\ddot{\psi}\cos\psi + ml\dot{\psi}^2\sin\psi &= F\,,\\ l\ddot{\psi} - g\sin\psi &= \ddot{x}\cos\psi\,,\end{aligned} \tag{6.10}$$

where x denotes the position of the cart with mass M, ψ denotes the pendulum angle, and F is a force that serves as the control input. The massless pendulum is of length l with a mass m attached at its end. Define the system state as $(x, \dot{x}, \psi, \dot{\psi})$ and the input as $u := F$. Starting from the initial state $(0, 0, \frac{\pi}{6}, 0)$, the goal is to keep the pendulum straight up, while keeping the state within given lower and upper bounds. In particular, we want to minimize the cost

$$\begin{aligned} J_0(x_0, \mathbf{u}_0) = &\sum_{k=0}^{T-1} \left(x_k^T Q x_k + u_k^T R u_k\right) + x_T^T Q_f x_T \\ &+ \lambda \sum_{i=0}^{T} \max(0, \underline{x} - x_i, x_i - \overline{x}), \end{aligned} \tag{6.11}$$

where λ penalizes the deviation of state x_i below $\underline{x}$ and above $\overline{x}$. We assume that the dynamics are unknown and use aDOBO to optimize the dynamics.

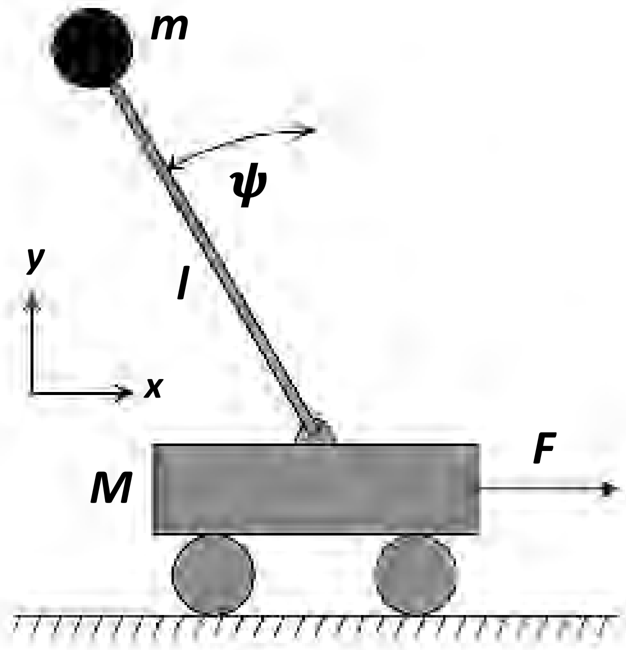


Figure 6.5: Cart-pole System

For simulation, we discretize the dynamics at a frequency of 10Hz. We choose $N = 30$, $M = 1.5$Kg, $m = 0.175$Kg, $\lambda = 100$ and $l = 0.28$m. The $Q = Q_f = \text{diag}([0.1, 1, 100, 1])$ and $R = 0.1$ matrices are chosen to penalize the angular deviation significantly. We use $\underline{x} = [-2, -\infty, -0.1, -\infty]$ and $\overline{x} = [2, \infty, \infty, \infty]$, i.e., we are interested in controlling the pendulum while keeping the cart position within $[-2, 2]$, and limiting the pendulum overshoot to 0.1. The optimal control problem for a particular linearization is a convex MPC problem and solved using YALMIP [201]. The true J_0^* is computed using *fmincon*.

As shown in Fig. 6.6, aDOBO reaches within 20% of the optimal performance in 250 iterations and continue to make progress towards finding the optimal controller. This sim-

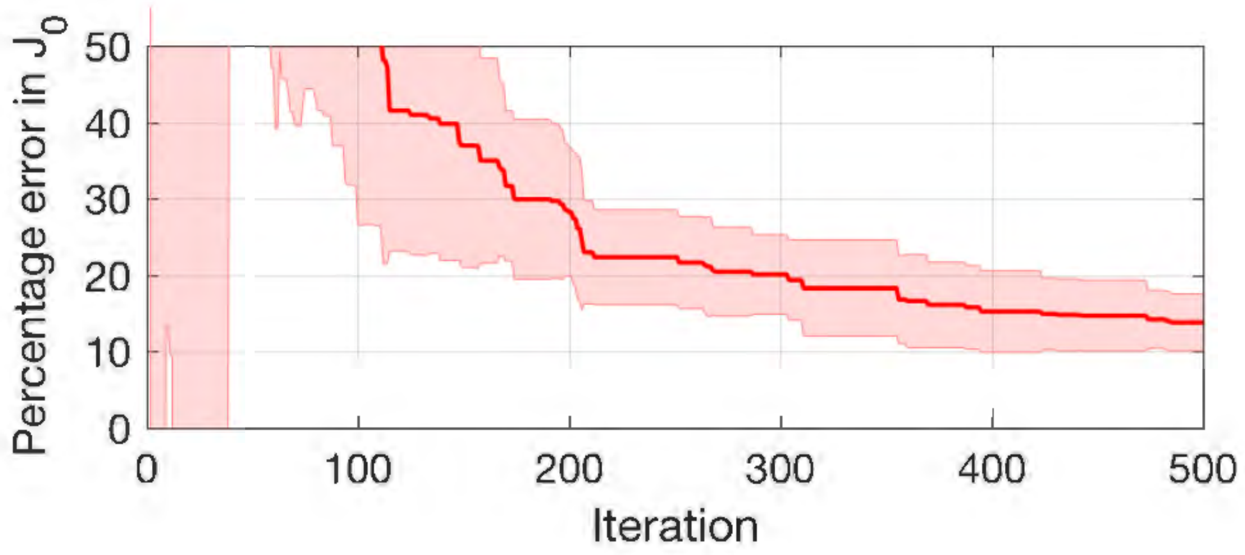


Figure 6.6: Cart-pole system: mean and standard deviation of η during the learning process. The learned controller reaches within 20% of the optimal cost in 250 iterations, demonstrating the applicability of aDOBO to highly non-linear systems.

ulation demonstrates that the proposed method (a) is also applicable to highly non-linear systems even though the system dynamics are approximated as a linear system, (b) can handle general convex cost functions that are not necessarily quadratic, and (c) different optimal control schemes can be used within the proposed framework. Since an MPC controller can in general be non-linear, this implies that the proposed method can also design complex non-linear controllers with an LTI parametrization.

6.3.2 Comparisons With Other Online Learning Methods

In this section, we compare our approach with some other online learning schemes for controller design, and discuss their relative advantages and disadvantages.

6.3.2.1 Indirect Learning-Based Control vs aDOBO

In this work, we aim to directly find the best linearization based on the observed performance. Another approach is to learn a linearization of the system based on the observed state and input trajectory during the experiments. The underlying hypothesis is that as more and more data is collected, a better linearization is obtained, eventually leading to an improved control performance. This approach is in-line with the traditional system identification approaches and indirect learning-based control approach discussed in Chapter 5, except that the identification is performed online. Thus, this approach can also be thought of as an indirect adaptive control algorithm.

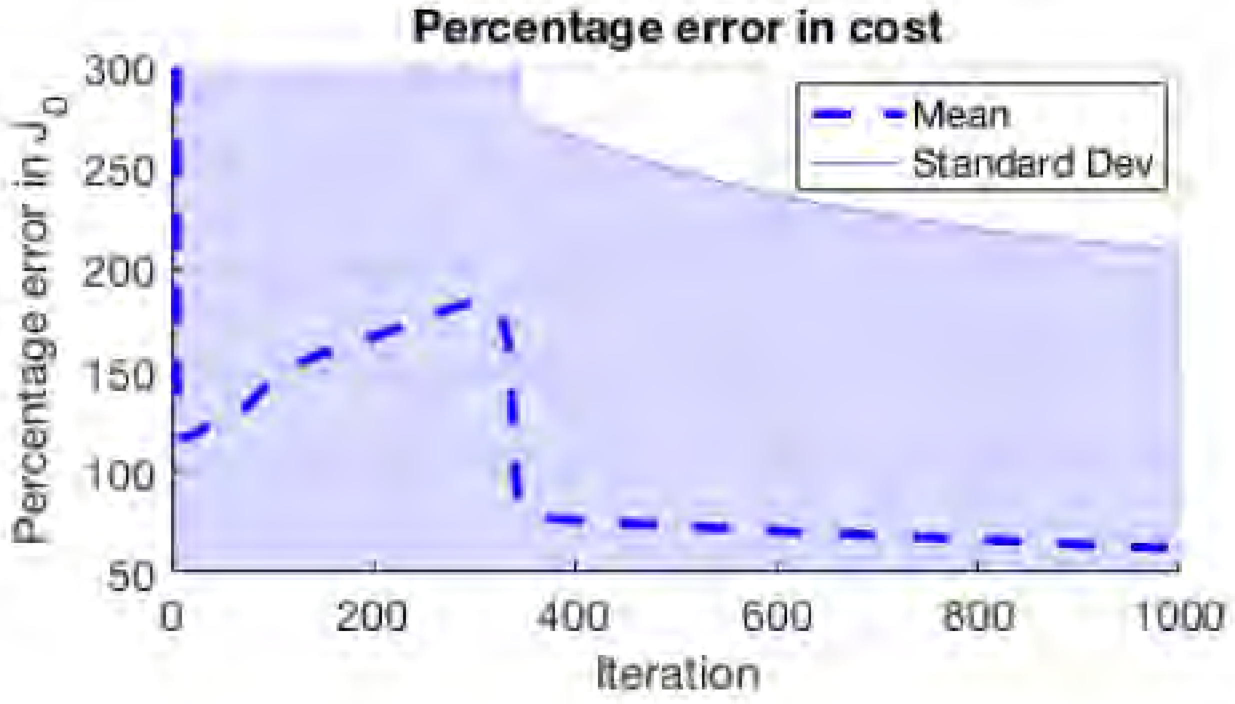


Figure 6.7: Performance obtained via learning (A, B) through state and input trajectories. When the underlying system is non-linear, this approach can result in a poor performance.

We record the state and input trajectories in each experiment. Let $(\mathbf{x}_0^j, \mathbf{u}_0^j)$ denotes the state and input trajectories for experiment j. We also let $\mathcal{D}_i = \cup_{j=1}^{i}(\mathbf{x}_0^j, \mathbf{u}_0^j)$. After experiment i, we fit an LTI model of the form $x_{k+1} = A_i x_k + B_i u_k$ using least squares on data in $\mathcal{D}_i$ and then use this model to obtain a new controller for experiment $i+1$. We apply the approach on the linear system in (6.13) and the non-linear system in (6.6) with the cost function in (6.7).

For the linear system, the approach converges to the true system dynamics in 5 iterations. However, this approach performs rather poorly on the non-linear system, as shown in Figure 6.7 and Table 6.1. When the underlying system is non-linear, all state and input trajectories may not contribute to the performance improvement. A good linearization should be obtained from the state and input trajectories in the region of interest, which depends on the task. For example, if we want to regulate the system to the equilibrium $(0, 0)$, a linearization of the system around $(0, 0)$ should be obtained. Thus, it is desirable to use the system trajectories that are close to this equilibrium point. However, a naive prediction error based approach has no means to select these "good" trajectories from the pool of trajectories and hence can lead to a poor performance. In contrast, aDOBO does not suffer from these limitations, as it explicitly utilizes a performance based optimization.

Iteration	aDOBO	Learning via LS
200	**6 ± 3.7%**	166.7 ± 411%
400	**2.2 ± 1.1%**	75.9 ± 189%
600	**1.8 ± 0.7%**	70.7 ± 166%

Table 6.1: Dubins car: mean and standard deviation of η obtained via learning (A, B) through least squares (LS), and through aDOBO.

6.3.2.2 Direct Adaptive Control vs aDOBO

Another potential approach to obtain the optimal controller for the task at hand is to directly parameterize and optimize the controller, such as the feedback matrix $K \in \mathbb{R}^{n_x n_u}$ for linear controllers [67]. For linear controllers, this optimization problem can be written as

$$\begin{aligned} \theta^* &= \arg\min_{\theta \in \mathcal{M}} J_0(x_0, \mathbf{u}_0)\,, \\ \text{sub. to} \;\; x_{k+1} &= f(x_k, u_k), \quad u_k = K_\theta x_k\,. \end{aligned} \tag{6.12}$$

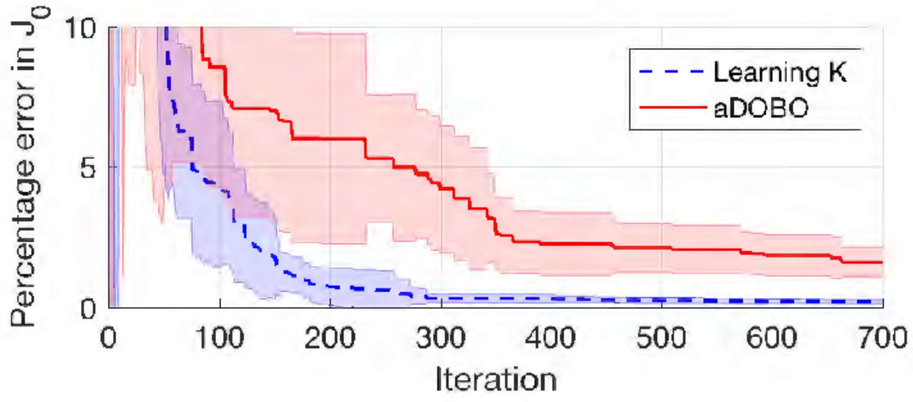
The advantage of this approach is that only $n_x n_u$ parameters are learned compared to $n_x(n_x + n_u)$ parameters in aDOBO, which is also evident from Fig. 6.8a, wherein the learning process for K converges much faster than that for aDOBO. However, a linear controller might not be sufficient for general cost functions, and non-linear controllers are required to achieve a desired performance. As shown in Sec. 6.3.1.3, aDOBO is not limited to linear controllers; hence, it outperforms the K learning method in such scenarios. Consider, for example, the linear system

$$x_{k+1} = x_k + y_k, \quad y_{k+1} = y_k + u_k\,, \tag{6.13}$$

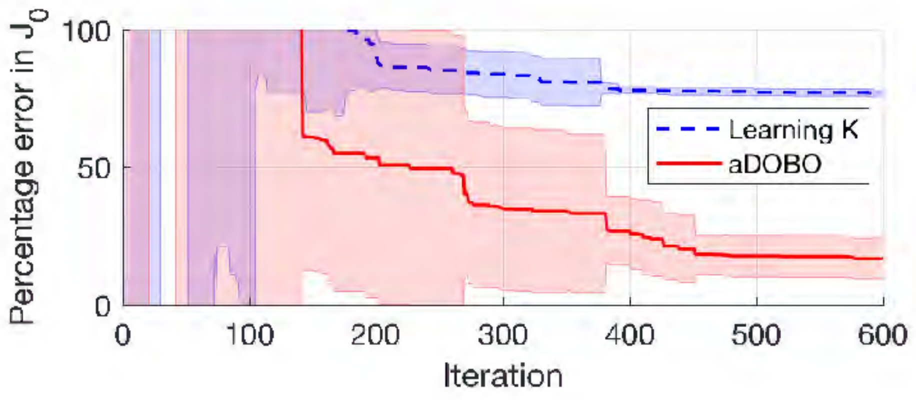
and the cost function in Eq. (6.11) with state $x_k = (x_k, y_k)$, $T = 30$, $\underline{x} = [0.5, -0.4]$ and $\overline{x} = [\infty, \infty]$. Q, Q_f and R are all identity matrices of appropriate sizes, and $\lambda = 100$.

As evident from Fig. 6.8b, directly learning a feedback matrix performs poorly with an error as high as 80% from the optimal cost. Since the cost is not quadratic, the optimal controller is not necessarily linear; however, since the controller in (6.12) is restricted to a linear space, it performs rather poorly in this case. In contrast, aDOBO continues to improve performance and reaches within 20% of the optimal cost within few iterations, because we implicitly parameterize a much richer controller space via learning A and B. In this example, we capture non-linear controllers by using a linear dynamics model with a convex MPC solver. Since the underlying system is linear, the true optimal controller is also in our search space. Our algorithm makes sure that we make a steady progress towards finding that controller. However, aDOBO is not restricted to learning a linear controller K.

One can also directly learn the actual control sequence to be applied to the system (which also captures the optimal controller) or use a rich parameterization for controller, such as a DNN. This approach may not be data-efficient compared to aDOBO as the parameter space can be very large depending on the problem cost function and horizon, and will require a large number of experiments. We demonstrate the performance of the direct adaptive control



(a) Dubins car



(b) System of Eq. (6.13)

Figure 6.8: Mean and standard deviation of η obtained via directly learning the feedback controller K [67] and aDOBO for different cost functions. (a) Comparison for the quadratic cost function of Eq. (6.7). Directly learning K converges to the optimal performance faster because fewer parameters are to be learned. (b) Comparison for the non-quadratic cost function of Eq. (6.11). Since the optimal controller for the actual system is not necessarily linear in this case, directly learning K leads to a poor performance.

approach that directly learns the optimal control sequence in Table 6.2. The performance error is more than 250% even after 600 iterations, rendering the method impractical for real systems.

6.3.2.3 Tuning Penalty Matrices vs aDOBO

In this section, we consider the case in which the cost function J_0 is quadratic (see Eq. (6.7)). Suppose that the nominal linearization of the system around $x^* = 0$ and $u^* = 0$ is known and given by (A^*, B^*). To design a controller for the actual system in such a case, it is a common practice to use an LQR controller for the linearized dynamics. However, the

Iteration	aDOBO	Learning Control Sequence
200	**53 ± 50%**	605 ± 420%
400	**27 ± 12%**	357 ± 159%
600	**17 ± 7%**	263 ± 150%

Table 6.2: System in (6.13): mean and standard deviation of η for aDOBO, and for directly learning the control sequence. Since the space of control sequence is huge, the error is substantial even after 600 iterations.

resultant controller may be sub-optimal for the actual non-linear system. To overcome this problem, authors in [208, 294] propose to optimize the controller by tuning penalty matrices Q and R in (6.7). In particular, we solve

$$\begin{aligned} \theta^* &= \arg\min_{\theta \in \mathcal{M}} J_0(x_0, \mathbf{u}_0^)\,, \\ \text{sub. to } \; x_{k+1} &= f(x_k, u_k), \quad u_k = K(\theta)x_k\,, \\ K(\theta) &= LQR(A^*, B^*, W_Q(\theta), W_R(\theta), Q_f)\,, \end{aligned} \tag{6.14}$$

where $K(\theta)$ denotes the LQR feedback matrix obtained for the system matrices (A^*, B^*) with W_Q and W_R as state and input penalty matrices, and can be computed analytically. For further details of LQR method, we refer interested readers to [54]. The difference between optimization problems (6.5) and (6.14) is that now we parameterize penalty matrices W_Q and W_R instead of system dynamics, and optimize them to achieve the optimal performance on the actual non-linear system directly using the experimental data. The optimization problem in (6.14) is solved using BO in a similar fashion as we solve (6.5) [208]. The parameter θ, in this case, can be initialized by the actual penalty matrices Q and R, instead of a random query, which generally leads to a much faster convergence. An alternative approach is to use aDOBO, starting from the nominal linear model. Actual penalty matrices Q and R are used for aDOBO.

When (A^*, B^*) are known to a good accuracy, (Q, R) tuning method is expected to converge quickly to the optimal performance compared to aDOBO as it needs to learn fewer parameters, i.e., $(n_x + n_u)$ (assuming diagonal penalty matrices) compared to $n_x(n_x + n_u)$ parameters for aDOBO. However, when there is error in (A^*, B^*) (or more generally if nominal dynamics are unavailable), the performance of the (Q, R) tuning method can degrade significantly as it relies on an accurate linearization of the system dynamics, rendering the method impractical for control design purposes. To compare the two methods we use the Dubins car model in Eq. (6.6). The rest of the simulation parameters are same as Section 6.3.1.1. We compute the linearization of Dubins car around $x^* = 0$ and $u^* = 0$ using (6.6) and add random matrices (A_r, B_r) to them to generate $A' = (1-\alpha)A^* + \alpha A_r$ and $B' = (1-\alpha)B^* + \alpha B_r$. We then initialize both methods with (A', B') for different αs. As shown in Fig. 6.9, the (Q, R) tuning method outperforms aDOBO, when there is no noise in (A^*, B^*). But as α increases, its performance deteriorates significantly. In contrast, aDOBO

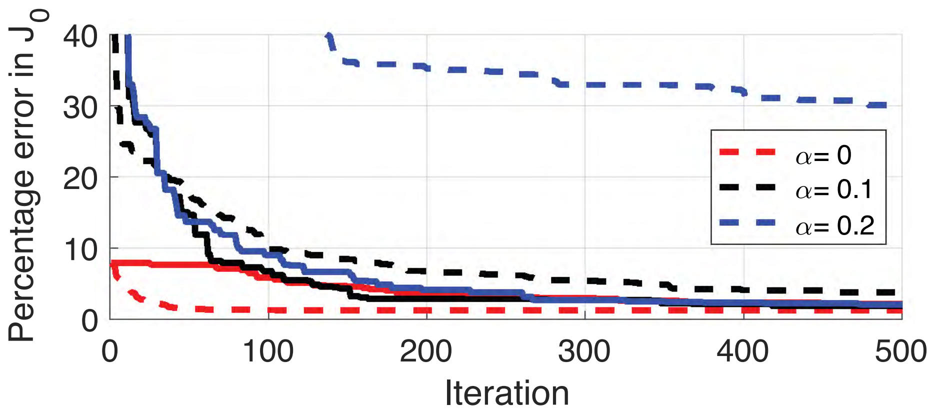


Figure 6.9: Dubins car: Comparison between tuning the penalty matrices (Q, R) [208] (dashed curves), and aDOBO (solid curves) for different noise levels in (A^*, B^*), the nominal linear dynamics around the desired goal state. When nominal dynamics are accurate, the (Q, R) tuning method outperforms aDOBO because fewer parameters are to be learned. Its performance, however, drops significantly as noise increases, rendering the method impractical for the scenarios where a good nominal dynamics model is not known to a good accuracy.

is fairly indifferent to the noise level, as it does not assume any accurate prior knowledge of system dynamics. The only information assumed to be known is penalty matrices (Q, R), which are generally designed by the user and hence are known a priori. Another limitation of this tuning process is that it is unclear which cost parameters should be tuned when the cost function is not quadratic, whereas aDOBO can be used for more general cost functions as shown in Sec. 6.3.1.3.

A summary of the advantages and limitations of different methods is provided in Table 6.3.

6.3.3 Hardware Experiments: Quadrotor Position Control

We now present the results of our experiments on Crazyflie 2.0, which is an open source nano quadrotor platform developed by Bitcraze (see Fig. 5.1). Its small size, low cost, and robustness make it an ideal platform for testing new control paradigms.

For small yaw, the quadrotor system is modeled as a rigid body with a ten dimensional state vector $x := [p, v, \zeta, \omega]$, which includes the position $p = (p_x, p_y, p_z)$ in an inertial frame I, linear velocities $v = (v_x, v_y, v_z)$ in I, attitude (orientation) represented by Euler angles ζ, and angular velocities ω. The system is controlled via three inputs $u := [u_1, u_2, u_3]$, where u_1 is the thrust along the z-axis, and u_2 and u_3 are rolling, pitching moments respectively. The full non-linear dynamics of a quadrotor are derived in [3] using first principles, and its physical parameters are computed in [188].

Method	Advantages	Limitations
(Q, R) learning [208]	Only $(n_x + n_u)$ parameters are to be learned so learning will be faster.	Performance can degrade significantly if nominal dynamics are not known to a good accuracy.
K learning [67]	Only $n_x n_u$ parameters are to be learned so learning will be faster.	Approach may not perform well for non-quadratic cost functions.
(A, B) learning via least squares	Can lead to a faster convergence when the underlying system is linear.	Approach is not suitable for non-linear system.
aDOBO	Does not require any prior knowledge of system dynamics. Applicable to general cost functions.	Number of parameters to be learned is higher, i.e., $(n_x^2 + n_x n_u)$.

Table 6.3: Relative advantages and limitations of different methods for task-specific controller design.

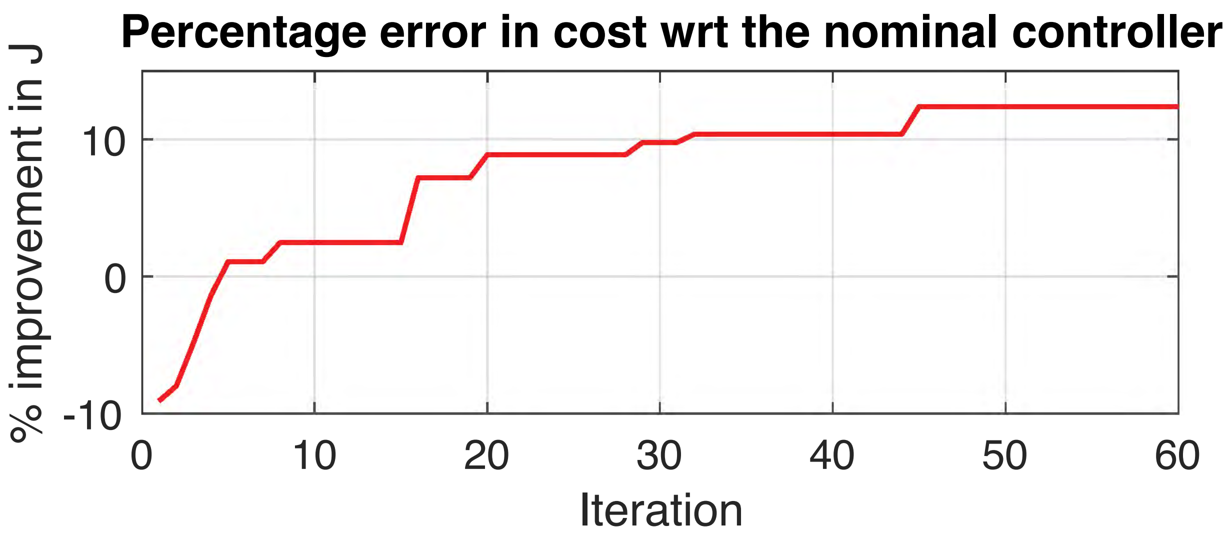


Figure 6.10: Crazyflie: percentage error between the learned and the nominal controller. The nominal controller is obtained by using the full 12D non-linear dynamics model of the quadrotor. As learning progresses, aDOBO outperforms the nominal controller by 12% on the actual system, indicating the capability of aDOBO to overcome modeling errors.

Our goal in this experiment is to track a desired position p^* starting from the initial position $p_0 = [0, 0, 1]$. Formally, we minimize

$$J_0(x_0, \mathbf{u}_0) = \sum_{k=0}^{T-1} \left(\bar{x}_k^T Q \bar{x} + u_k^T R u_k\right) + \bar{x}_T^T Q_f \bar{x}_T \,, \tag{6.15}$$

where $\bar{x} := \left[p - p^*, v, \zeta, \omega\right]$. Given the dynamics in [3], the desired optimal control problem can be solved using LQR; however, the resultant controller may still not be optimal for the actual system because (a) true underlying system is non-linear and (b) the actual system may not follow the dynamics in [3] exactly due to several unmodeled effects, as we discussed in Chapter 5. Hence, we use the linearized model obtained from the first principle model as the nominal dynamics model, and model the unknown dynamics of v_x and v_y as

$$\begin{bmatrix} f_{v_x} \\ f_{v_y} \end{bmatrix} = A_\theta \begin{bmatrix} \phi \\ \psi \end{bmatrix} + B_\theta u_1 \, , \tag{6.16}$$

where A and B are parameterized through θ. Our goal is to learn the parameter θ^* that minimizes the cost in (6.15) for the *actual* Crazyflie using aDOBO.

We use $T = 400$; the penalty matrix Q is chosen to penalize the position deviation. In our experiments, Crazyflie was flown in presence of a VICON motion capture system, which along with on-board sensors provides the full state information at 100Hz. The optimal control problem for a particular linearization in (6.16) is solved using LQR. For comparison, we compute the *nominal* optimal controller using the linear nominal dynamics obtained from [3].

Figure 6.10 shows the performance of the controller from aDOBO compared with the nominal controller during the learning process. The nominal controller outperforms the learned controller initially, but within a few iterations, aDOBO performs better than the controller derived from the known dynamics model of Crazyflie. This is because aDOBO optimizes controller based on the performance of the *actual* system and hence can account for unmodeled effects. In 45 iterations, the learned controller outperforms the nominal controller by 12%, demonstrating the performance potential of aDOBO on real systems. Figure 6.11 shows the error trajectories (from p^*) of p_x, p_y and p_z for the learned controller and the nominal controller. Although the p_x and p_y tracking is similar for the both controllers, the actual system is able to track the desired z position much better with the learned controller, resulting in its lower cost compared to the nominal controller.

6.4 Going Beyond Linear Models: Learning Task-Driven Gaussian Process (GP) Models

Even though we have seen how aDOBO enables controlling nonlinear systems through simple linear models, the linear models are still limited in their capacity to learn complex, nonlinear dynamics effects, which might be required to taken into account to satisfactorily complete the task at hand. In this section, we will discuss how we can extend aDOBO to Gaussian Process (GP)-based dynamics models.

Unlike linear models, GPs are non-parametric models so we can't directly optimize any parameters. However, using non-parametric, kernel-based approaches for modeling a system (e.g., a GP) requires the selection of an appropriate kernel function and a set of hyperparameters for that function. Typically, these selections are data-based, e.g. through minimizing

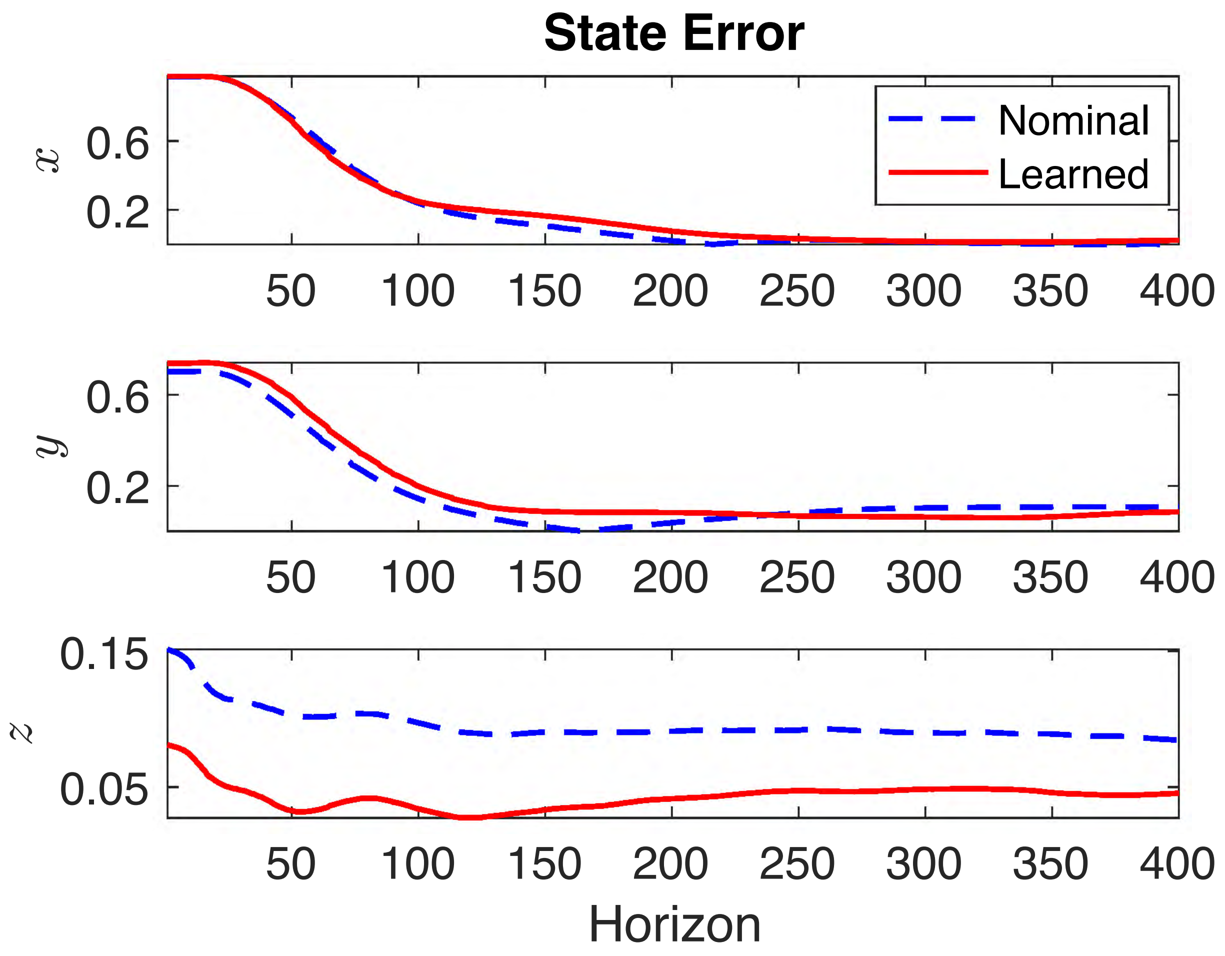


Figure 6.11: Crazyflie: tracking error for the learned and nominal controllers. The final learned controller does a better tracking of p_z state, which results in an overall better performance.

a loss function that is often a trade-off between the prediction error and the complexity of the model. However, as discussed earlier, a low prediction error might not necessarily lead to a better control performance. Instead, we will use aDOBO to optimize the kernel and its hyperparameters directly with respect to the performance of the closed-loop rather than the prediction error.

6.4.1 Problem Setup

Our goal is to optimize the kernel and its hyperparameters of a GP model with respect to the cost functional $J_0(x_0, \mathbf{u}_0)$. Thus, in contrast to the classical kernel selection problem, where the kernel is selected to minimize the state prediction error, our goal here is not to get the most accurate model but the one that achieves the best closed-loop behavior.

We define a set $\mathcal{K} = \{\mathfrak{K}^1, \dots, \mathfrak{K}^{n_\mathfrak{K}}\}$ of $n_\mathfrak{K} \in N$ kernel candidates $\mathfrak{K}^j$ that we want to choose the kernel from for our GP model. BO will be used to select the kernel with the best closed-loop performance in this set.

Remark 10 *The selection of possible kernels can be based on prior knowledge about the system, e.g. smoothness with the Matérn kernel or number of equlibria using a polynomial kernel, see [48] and [56] for general properties, respectively.*

In addition, each kernel depends on a set of hyperparameters. Since the number of hyperparameters could be different for each kernel, we define a set of sets $\mathcal{P} = \{\Phi^1, \dots, \Phi^{n_\mathfrak{K}}\}$ such that $\Phi^j \subset \mathbb{R}^{n_{\Phi^j}}$ is a closed set. Here, n_{Φ^j} represents the number of hyperparameters for the kernel $\mathfrak{K}^j$. Moreover, we assume that Φ^j is a valid *hyperparameter set.*

Definition 2 *The set Φ is called a valid hyperparameter set for a kernel function $\mathfrak{K}$ if and only if the set Φ is a domain for the hyperparameters of $\mathfrak{K}$.*

For example, for the squared exponential kernel, the lengthscale hyperparameters are always positive. The hyperparameter set Φ takes into account this non-negativity constraint to define a valid domain for the hyperparameters.

Given $\mathcal{K}$ and $\mathcal{P}$ our goal is to solve the following optimization problem:

$$\begin{aligned} \mathfrak{K}^*, \varphi^* =& \arg \min_{\mathfrak{K} \in \mathcal{K}, \varphi \in \Phi} J_0(x_0, \mathbf{u}_0)\,, \\ \text{subject to } \; & x_{k+1} = f(x_k, u_k)\,, \quad u_k = \pi_k^1(x_k, \mathfrak{K}, \varphi), \end{aligned} \tag{6.17}$$

where the controller $\pi_k^1(x_k, \mathfrak{K}, \varphi)$ is obtained through solving the optimal control problem on the GP model:

$$\begin{aligned} \pi_k(x_k, \mathfrak{K}, \varphi) := \bar{\mathbf{u}}_k =& \arg \min_{\mathbf{u}_k} J_k(x_k, \mathbf{u}_k)\,, \\ \text{subject to } \; x_{j+1} =& f_{nom}(x_j, u_j) + \mathcal{M}(x_j, u_j, \mathfrak{K}, \varphi). \end{aligned} \tag{6.18}$$

Since the GP regression is a probabilistic regression, the learned dynamics $\mathcal{M}$ is stochastic in general; however, it is a common practice to use just the mean of the GP model during the controller design process. In this work, $\mathcal{M}$ refers to the mean of the GP model, unless stated otherwise.

6.4.2 Closed-Loop Kernel and Hyperparameter Selection

We now describe the proposed overall procedure for the kernel selection to optimize the closed-loop behavior. We start with an initial kernel $\mathfrak{K}$ with hyperparameters φ for the GP model, and obtain the control law for the system as per (6.18). This control law is then applied to the actual system, and the cost function $J_0(x_0, \mathbf{u}_0)$ is evaluated after performing the control task. Depending on the obtained cost value, BO suggests a new kernel and corresponding hyperparameters for the GP model in order to minimize the cost function on the actual system. With this model, the control task is repeated and, based on the cost evaluation, BO suggests the next kernel and hyperparameters. This procedure is continued until a maximum number of task evaluations is reached or the user rates the closed-loop performance as sufficient enough. We now describe the above three steps, i.e. initialization, evaluation and optimization, in detail.

6.4.2.1 Initialization

For the first evaluation of the closed-loop, the kernel-based model $\mathcal{M}$ is created with an initial kernel $\mathfrak{K}^j$ of the set $\mathcal{K}$ and hyperparameters $\varphi^j \in \Phi^j$ with $j \in \{1, \dots, n_{\mathfrak{K}}\}$. One potential way to select the initial kernel and hyperparameters is to set them equal to the kernel and hyperparameters of a prediction model that is optimized with respect to a loss function, e.g., using cross-validation or maximization of the likelihood function [56].

6.4.2.2 Bayesian Optimization

In the next step, we use BO to minimize the cost function with respect to the kernel and its hyperparameters. This problem involves both continuous (i.e., the hyperparameters) and discrete variables (i.e., the kernel) in the optimization task whereas classical BO assumes continuous variables only. To overcome this restriction, a modified version of BO is used where the kernel function is transformed in a way such that integer-valued inputs are properly included [128].

Based on previous evaluations of the cost function, BO updates the prior and minimizes the acquisition function to provide the next kernel and the hyperparameters to evaluate on the real system.

6.4.2.3 Task Evaluation

For the i-th task evaluation, BO determines an index value $j \in \{1, \dots, n_{\mathfrak{K}}\}$ and a $\varphi^j \in \Phi^j$. The control law (6.18) for the kernel-based model $\mathcal{M}$, with the determined kernel $\mathfrak{K}^j$ and hyperparameters φ^j, is applied to the actual system. The corresponding input and state sequences $\mathbf{x}_0$ and $\mathbf{u}_0$, respectively, are recorded. Afterwards, the cost function $J_0(x_0, \mathbf{u}_0)$ is evaluated. The procedure is repeated until a maximum number of task evaluations has been reached or a sufficient performance level has been achieved.

Remark 11 *We focus here on a single, fixed initial state x_0. However, multiple (close by) initial states can be considered by using the expected cost across all initial states.*

6.4.3 Hardware Experiments: End Effector Tracking of a Robotic Arm in Viscous Liquid

In this section, we present an example with a 3-DOF robot that demonstrates the applicability of the proposed approach to hardware testbeds. BO is used with the expected-improvement-plus as acquisition function because of its satisfactory performance in practical applications, see [64].

6.4.3.1 Setup

For the experimental evaluation, we use the 3-DoF SCARA robot CARBO as pictured in 6.12. The links between the joints have a length of 0.3 m and a spoon is attached at the end effector of the robot. The goal is to follow a given trajectory as precisely as possible in a viscous liquid (jelly in this case), without using high feedback gains, which might result in several practical disadvantages, see [234]. Therefore, a precise model of the system's dynamics is necessary. Since the modeling of the nonlinear fluid dynamics with a parametric model would be very time consuming, we use a computed torque control method based on a GP model which allows high performance tracking control. Underlying, a low level PD-controller enforces the generated torque by regulating the voltage based on a measurement of the current. The controller is implemented in MATLAB/Simulink on a Linux real-time system with a sample rate of 1 ms. For the implementation of the GP model, we use the GPML toolbox [2]. The desired trajectory follows a circular stirring movement through the fluid with a frequency of 0.5 hz.

Modeling: The state of the robot consists of the position and velocity of all joints $[q, \dot{q}]$ and the corresponding torque for the i-th joint, τ_i. Here, we use a first principle model of the robotic arm as the nominal model. The parameters for the nominal model are physically measured; however, they do not take into account the change in dynamics due to the presence of the viscous liquid. To compensate for these dynamics inaccuracies, we learn a GP model.

We learn two different GP models: an open-loop model that minimizes the prediction error (negative log likelihood of the data) based on collected state-input pairs, and a closed-loop model that optimizes the closed-loop performance using aDOBO. The data for the nominal model is collected around the desired trajectory using a high gain controller. The placement of the training points heavily influences the control performance of the open-loop model; however, the proposed approach focuses on improving the performance based on existing data rather than designing the optimal experiment to collect the data for the open-loop model. Since the GP produces one-dimensional outputs only, 3 GPs are used in total for the modeling of the robot's dynamics. Each GP $i = 1, \dots, 3$ uses a squared exponential kernel

$$\mathfrak{K}(x, x') = \varphi_i^2 \exp\left(\frac{-\|x - x'\|^2}{\varphi_{i+3}^2}\right), \varphi_i \in \mathbb{R} \setminus \{0\}, \tag{6.19}$$

with the lengthscales $[\varphi_1, \dots, \varphi_6]$ and the signal noise $\sigma_n \in \mathbb{R}^3$ as the kernel hyperparameters, see [253]. Thus, a total number of 9 hyperparameters must be optimized. In these experiments, the kernel is fixed to reduce the optimization space and thus, the number of

[2]http://www.gaussianprocess.org/gpml/code

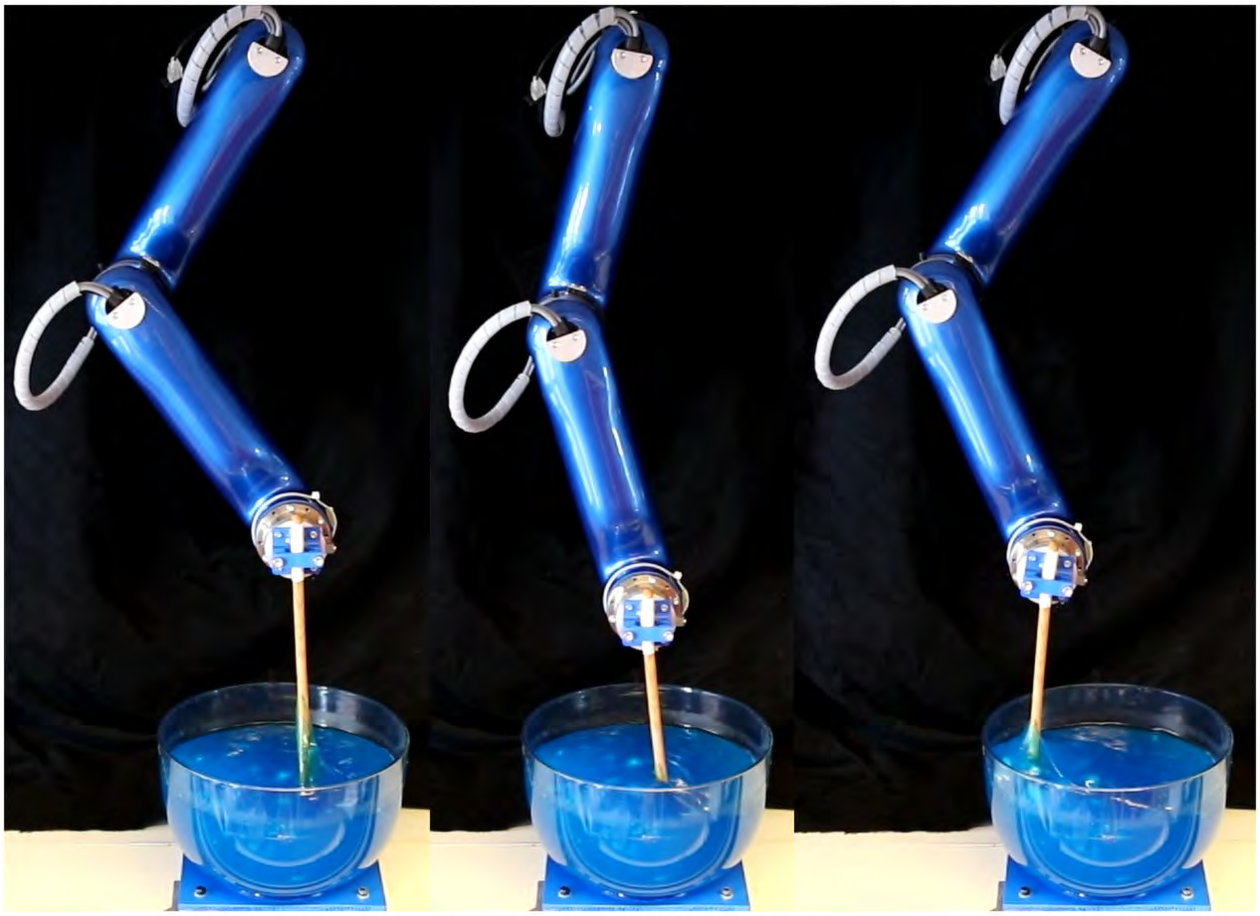


Figure 6.12: Stirring viscous liquid with the 3-dof SCARA robot CARBO using aDOBO.

task evaluations.

Control law: The control input, i.e. the torque $\tau(\dot{q}, q)$ for all joints, is generated based on the nominal model, the mean prediction of the GP model as feed-forward component, and a low gain PD-feedback part

$$\tau_d = \hat{H}\ddot{q}_d + \hat{C}\dot{q}_d + \hat{g} + \mathcal{M}(\dot{q}, q) - K_d\dot{e} - K_p e. \tag{6.20}$$

Here, the desired trajectory is given by $q_d, \dot{q}_d$, and $\ddot{q}_d$, with the error $e = (q_d - q)$ and $\dot{e} = (\dot{q}_d - \dot{q})$. The feedback matrices are given by $K_p = \mathrm{diag}([60, 40, 10])$ and $K_d = \mathrm{diag}([1, 1, 0.4])$. For the discretization of the control input, a zero-order method is used. For more details on the discretization of control, we refer the interested readers to [49].

Table 6.4: Comparison between open-loop and closed-loop optimization for learning dynamics inaccuracies.

Value	Data-based	Closed-loop
σ_n	$[0.10, 3\times10^{-3}, 6\times10^{-4}]$	$[0.20, 4\times10^{-3}, 3\times10^{-4}]$
$\varphi_{1,2,3}$	$[3.49, 1.42, 2.87]$	$[2.61, 1.68, 5.70]$
$\varphi_{4,5,6}$	$[1.21, 0.25, 0.27]$	$[0.80, 0.27, 0.29]$
Log. likelihood	$[89, -121, -176]$	$[115, -113, -136]$
Cost (Tracking error)	1.49	**1.05**

6.4.3.2 Evaluation

The evaluation of the performance of the closed-loop is based on the cost function

$$C = \frac{1}{2000}\sum_{k=0}^{2000} e_k^T e_k \tag{6.21}$$

with a discretization timestep of $\Delta T = 1\,\text{ms}$. Therefore, the cost function is a measure for the tracking accuracy of the stirring movement. We consider as kernel candidate the squared exponential kernel, such that only the hyperparameters $\sigma_n, \sigma_f, \varphi$ are optimized. For the open-loop model, the hyperparameters are optimized based on a gradient method to minimize the log likelihood function. In contrast, aDOBO is used to minimize the tracking error in the closed-loop model. The initial values of the hyperparameters for aDOBO are set to the values obtained my minimizing teh log-likelihood function. The lower and the upper bounds for the hyperparameters are defined as the 0.5 and 2 times of the initial values respectively. Table 6.4 shows the comparison between the open-loop and the closed-loop models. Even though the open-loop model leads to a lower prediction error (higher negative log likelihood), it leads to a higher tracking error.

The evolution of the minimum cost over the trials for the closed-loop model, where each trial is a single stirring movement, is shown in Fig. 6.13. After 100 trials, the tracking error is decreased by 30% through the optimization of the GP model using aDOBO. Even if the resulting hyperparameters are sub-optimal with respect to the likelihood function, the performance of the closed-loop is significantly improved.

The comparison of the joint position error for the open and closed-loop models is shown in Fig. 6.14. The open-loop model leads to a larger tracking error. We note that the performance of the open-loop model can be improved further by collecting more training data; however consequently, the inference time of the GP model also increases significantly. In contrast, the proposed method does not increase the computational burden of the Gaussian process prediction which is often critical in real-time applications.

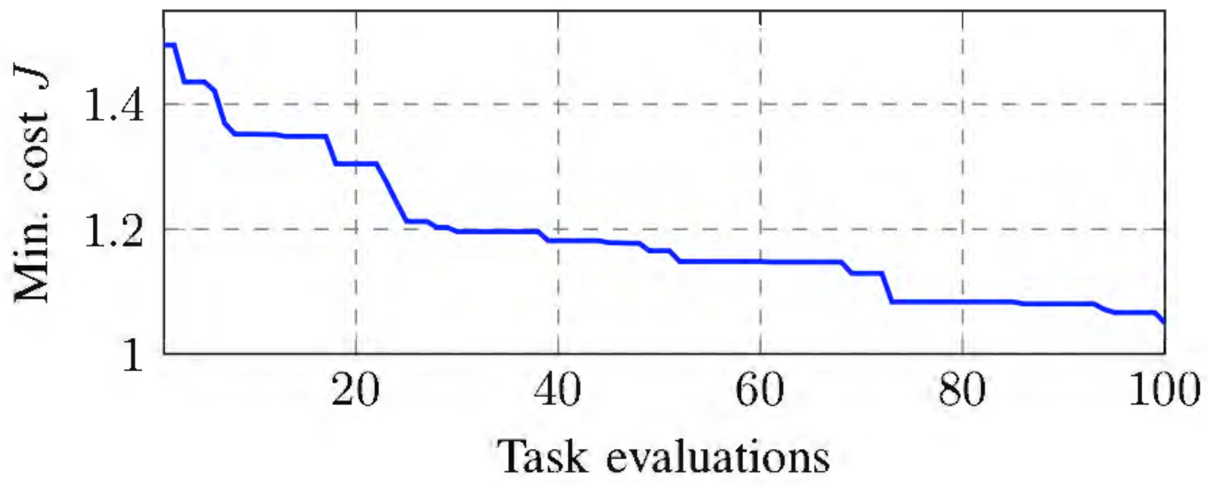


Figure 6.13: Minimum of the cost function over the number of trials for the closed-loop model. The closed-loop model is optimized using aDOBO.

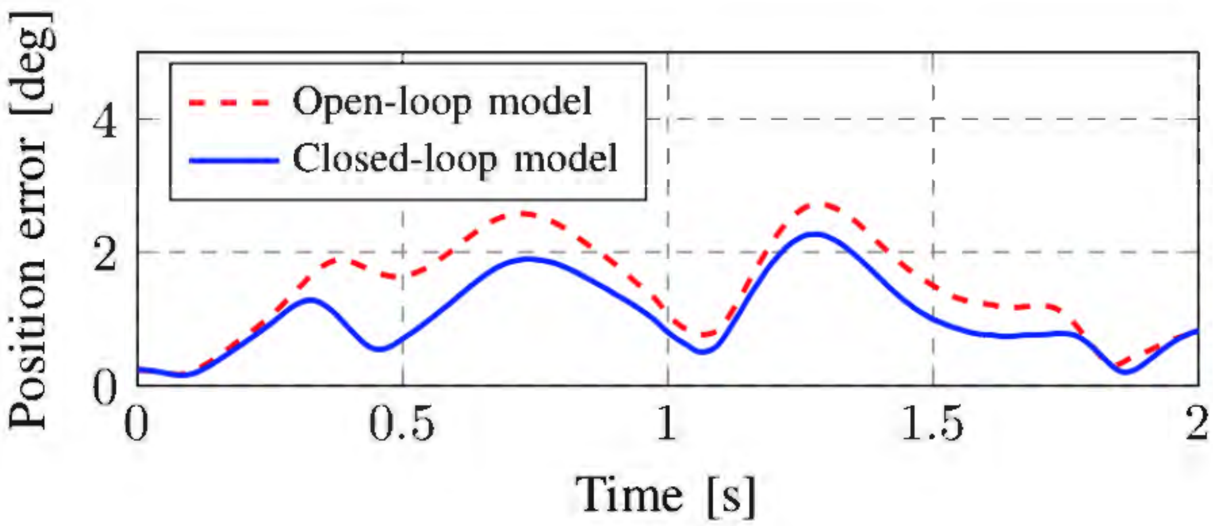


Figure 6.14: Comparison of the root square position error of all joints for the open-loop and closed-loop models.

6.5 Combining Indirect and Direct Learning-Based Control

In previous sections, we discussed how a direct learning-based control approach like aDOBO can lead to a better task-specific controller than an indirect approach. However, such a direct approach has some disadvantages too – the designed controller is task-specific and may not be robust to any changes in the task; whereas, a model learned through indirect approach is task-agnostic and can be used to design controllers for a variety of tasks.

Moreover, as the task gets more complex (or when the controller requires a significantly higher number of parameters), the direct learning-based approaches can be highly data inefficient. This dichotomy between direct and indirect approaches can be best understood through the lens of the amount of data available for learning. Whenever an experiment is conducted on the system, we obtain a state-input trajectories and an overall reward/cost for the trajectory. Since a state-input trajectory typically consists of several state-action pairs, which is what typically used in indirect approaches to learn a dynamics model, an experiment typically provides significantly more number of data points for an indirect approach than for a direct approach. Thus, from a data standpoint, indirect approaches tend to be more data efficient. The same argument also highlights the respective limitations of the two approaches: indirect approaches typically completely ignore the reward data and only use the state-input trajectories for learning, leading to a disconnect between the accuracy of the learned model and its control performance; on the other hand, direct approaches completely ignore the state-input trajectories and only use the reward data for learning, leading to a task-specific and data-inefficient learning. Thus, a natural question to ask is "can we combine the respective advantages of indirect and direct learning-based approaches?" This question is the focus of our discussion in this section.

In this section, we propose a probabilistic framework, Model-Based priors for Model-Free learning (MBMF), that integrates indirect (typically model-based) and direct (typically model-free) approaches. This bridging is achieved by using the cost estimated by the model learned by the indirect approach as the prior for the policy optimization via a direct approach. In particular, we learn a dynamics model from scratch which is used to compute the trajectory distribution corresponding to a given policy, which in turn can be used to estimate the cost of the policy. This estimate is used by a Bayesian Optimization-based model-free (MF) policy search to guide the policy exploration. In essence, this probabilistic framework allows us to model and combine the uncertainty in the cost estimates of the two methods, using both the state-input trajectories as well as the observation of the actual cost/reward during the learning process. The advantage of doing so is to exploit the structure and generality of the dynamics model throughout the entire state-action space. At the same time, the evidence provided by the observation of the actual cost can be integrated into the estimation of the posterior.

6.5.1 Model-Based Priors for Model-Free Learning (MBMF)

We now present our novel approach to incorporating a model-based (MB) prior in a direct, model-free (MF) method, which we term *MBMF*. As with most MB approaches, our algorithm starts with training a forward dynamics model $f_{unknown}$ from single-step state transition data $\mathcal{D}_1 := \{(x_k, u_k), x_{k+1}\}$. This model can be linear or non-linear and can be learned in a variety of ways, e.g., using linear regression, GP regression, etc. Once the dynamics model is trained, for any given policy parameterization $\pi(\cdot, \theta)$, we can predict the corresponding trajectory distribution by iteratively computing the distribution of states $x_k := \tilde{f}(x_{k-1}, u_{k-1}) \equiv f_{nom}(x_{k-1}, u_{k-1}) + f_{unknown}(x_{k-1}, u_{k-1})$ for $k = 1 \dots T$. Given the trajectory distribution, we compute the predicted distribution of the cost as a function of the policy parameters using (6.3). We denote the expected value of this predicted cost function as $\tilde{J}(\theta)$

At the same time, similarly to BO, we train a GP-based performance function, $\hat{J}(\theta)$, that predicts $J(\theta)$ given the measured tuples of $\mathcal{D}_2 = \{\theta, J(\theta)\}$. Here, $J(\theta)$ denotes the *observed* cost corresponding to the policy $\pi(\cdot, \theta)$ for the given horizon, as defined in (6.3). However, unlike plain BO, we employ the prediction of the cost distribution from the dynamics model as the prior mean of the performance function[3]. This modified performance function is then used to optimize the acquisition function α to compute the next policy parameters θ' to evaluate on the real system. The policy $\pi(\cdot, \theta')$ is then rolled out on the actual system. The observed state-input trajectories and the realized cost data is next added to $\mathcal{D}_1$ and $\mathcal{D}_2$ respectively, and the entire process is repeated again. A summary of our algorithm is provided in Algorithm 6.

Intuitively, the learned dynamics model has the capability to estimate the cost corresponding to a particular policy; however, it suffers from any unmodeled inaccuracies (also referred to as the *model bias*) which translates into a bias in the estimated cost. The BO performance function, on the other hand, can predict the true cost of a policy in the regime where it has observed the training samples, as it was trained directly on the observed performances. However, it can have a huge uncertainty in the cost estimates in the unobserved regime. Incorporating the model-based cost estimates as the prior allows it to leverage the structure of the dynamics model to guide its exploration in this unobserved regime. Thus, using the model-based prior in BO leads to a sample-efficient exploration of the policy space, and at the same time overcomes the biases in the model-based cost estimates, leading to the optimal performance on the *actual* system.

Note that we collect trajectory data at each iteration so, in theory, we can update the dynamics model, and hence the performance function prior, at each iteration. However, it might be desirable to update the prior every F iterations instead, as the dynamics model might change significantly between consecutive iterations, especially when the dataset $\mathcal{D}_1$ is small. We will demonstrate the effect of F on the learning progress in Section 6.5.2.

[3]A more correct, but computationally harder approach would be to treat the full cost distribution as a prior for the performance function.

Algorithm 6: MBMF Algorithm for combing indirect and direct learning-based control approaches.

```
1  init: Sample policy parameters $\theta \sim N(0, I)$
2  Apply sampled policies on the system and record resultant state-input trajectory
     and cost data
3  Initialize $\mathcal{D}_1 \leftarrow \{(x_k, u_k), x_{k+1}\}$;   $\mathcal{D}_2 \leftarrow \{\theta, J(\theta)\}$
4  Train dynamics model $f_{unknown} : x_k, u_k \to x_{k+1}$ using $\mathcal{D}_1$
5  Define $\tilde{J}(\cdot)$: Computed by evaluating the trajectory distribution corresponding to $\pi$
     using Monte-Carlo on $\tilde{f}$ and computing the expected cost in (6.3)
6  while optimize do
7      Train GP-based performance function $\hat{J} : \theta \to J(\theta)$ using $\mathcal{D}_2$ and $\tilde{J}(\theta)$ as the
         prior mean
8      Minimize the acquisition function $\alpha : \theta^{'} = \min_{\theta} \alpha(\hat{J}, \theta)$
9      Evaluate $\theta^{'}$ on the real system $f$
10     Collect trajectory data $(x_k, u_k, x_{k+1})$ and the observed cost $J(\theta^{'})$
11     Add $\{\theta^{'}, J(\theta^{'})\}$ to $\mathcal{D}_2$ and trajectory data to $\mathcal{D}_1$
12     Every $F$ iterations:
13         Update the dynamics model $f_{unknown}$ based on $\mathcal{D}_1$
14         Redefine $\tilde{J}(\cdot)$ based on the updated GP dynamics
```

It should also be noted that indirect learning-based algorithms like PILCO [94] can be thought of as a special case of our approach, where the performance function consists exclusively of the prior mean provided from a GP-based dynamics model, without any consideration of the evidences (i.e., the measured costs). In other words, PILCO does not take the dataset $\mathcal{D}_2$ into account. Leveraging $\mathcal{D}_2$ allows the BO to learn an accurate performance function by accounting for the differences between the "belief" cost based on the dynamics model and the actual cost observed on the system.

Remark 12 *It is important to note that we do not explicitly compute the function $\tilde{J}(\theta)$. The function is only computed for specific θ that are queried by the optimization algorithm during the optimization of the acquisition function (Line 8 of Algorithm 6).*

Remark 13 *The proposed approach is agnostic to the function approximator used to learn the dynamics model; thus, different dynamics models, e.g., linear models, neural networks, GPs, Bayesian neural networks, etc. can easily be used in the proposed framework.*

6.5.2 Numerical Simulations

In this section, we compare the performance of MBMF with a pure MB (or indirect) method, a pure MF (or direct) method, as well as a combination of the two where the model is used to "warm start" the MF method.

6.5.2.1 Experimental Setting

Task details. We apply the proposed approach as well as the baseline approaches on two different tasks. In the first task, a 2D point mass is moving in the presence of obstacles. The setup of the task is shown in Fig. 6.18. The agent has no information about the position and the type of the obstacles (the Grey cylinders). The goal is to reach the goal position (the Green circle) from the starting position (the Red circle). For the cost function, we penalize the squared distance from the goal position.

In the second task, an under-actuated three degree-of-freedom (DoF) robotic arm (only two of the three joints can be controlled) is trying to push an object from one position to another. The setup of the task is shown in Fig. 6.22. The Red box represents the object which needs to be moved to the goal position, denoted by the Green box. As before, the squared distance from the goal position of the object is used as the cost function.

These tasks pose challenging learning problem because they are under-observed, under-actuated, and have both contact and non-contact modes, which result in discontinuous dynamics.

Implementation details. For the GP regression, we use the GPy package [140]. We use the Dividing Rectangles (DIRECT) algorithm [123] for all policy searches in this paper. For simulating the tasks, we use OpenAI Gym [61] environments and the Mujoco [290] physics engine. In our experiments, we employ linear policies, but more complex policies can be easily incorporated as well.

Baselines details. For the MB method, we learn a dynamics model and use this dynamics model to perform policy search. In particular, we start with some initial random state-action, next-state pairs. A GP-based one-step forward dynamics model is learned using this data. Given the dynamics model and the cost function, we learn a linear policy using DIRECT. The resultant policy is then executed on the real system and the corresponding state and action trajectories, as well as the resultant cost are obtained. The observed trajectories are then added to the training set, and the entire process is repeated again. We denote this baseline as **MB** in our plots.

For the MF method, we use BO to directly find the optimal policy parameters, and denote it as **MF** in plots. In the final variant, we use the MB method above to optimize the policy for a given number of iterations, after which we switch to BO and continue the optimization. The cost observations obtained during the executions of MB method were used to initialize the BO. We denote it as **MB+MF** in the plots. We will simulate this baseline for different

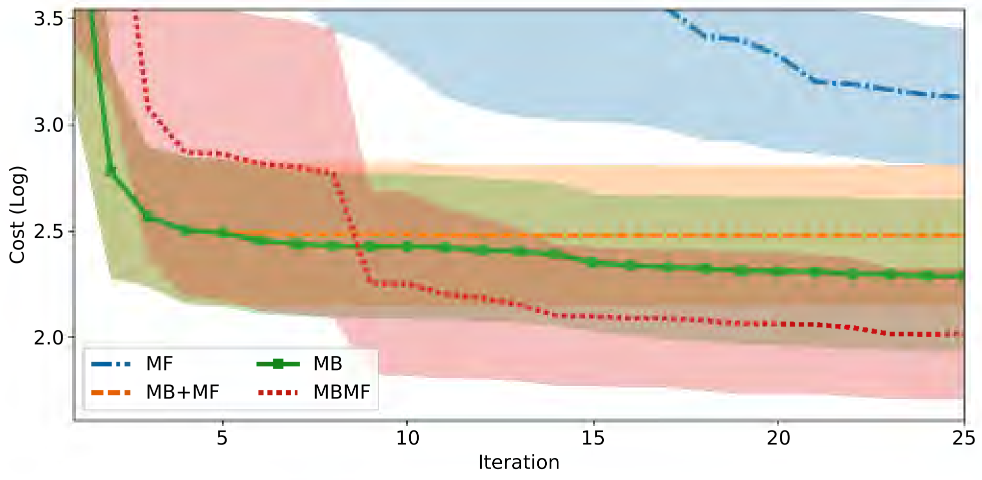


Figure 6.15: The mean (curves) and the standard deviation (shaded regions) of the cost obtained for different approaches for the 2D point mass system. Each iteration corresponds to one trial on the actual system. A pure MF approach is unable to perform well. A pure MB approach continues to improve, but is outperformed by the MBMF, indicating the utility of blending MB and MF approaches.

switching points, which corresponds to the number of iterations after which we switch from MB to MF approach. We denote this number by K in our plots. Finally, we denote our approach as **MBMF** and also simulate it for multiple prior update frequencies F.

6.5.2.2 2D Point Mass

The goal of this experiment is to demonstrate how leveraging the MB prior in the MF method can reduce the model-bias and yet maintains the data-efficiency. We use a GP-based dynamics model for this simulation, where we learn a separate GP for every dimension of the state. We use Monte-Carlo simulation to find the trajectory distribution which is highly parallelizable and known to be very effective for GPs [181]. Nevertheless, other schemes can be used to compute a good approximation of this distribution [94].

The optimal mean cost (curve) and the standard deviation (shaded area) obtained for different approaches (across thirty trials) as learning progresses are shown in Fig. 6.15, where each iteration corresponds to one execution on the real system. The MF approach (the dot-dashed Blue curve) improves as the learning progresses, but is still significantly outperformed by all other approaches, indicating the data-inefficiency of a pure MF approach. The pure MB approach (the Green curve) continues to improve as learning progresses; however, it is outperformed by MBMF very early on. Interestingly, in this case, using MB method to warm start the MF method (with K=5) doesn't improve the performance, as evident from the dotted Orange curve, indicating that using the MB component to initialize the MF component may not be sufficient for the policy improvement. In contrast, using model information as a prior for the MF method (with F=10) outperforms the other approaches and is able to learn a good policy roughly within 15 iterations, indicating the utility of systematically incorporating the model information during policy exploration. We note that MBMF also has a smaller variance compared to all the other baselines, indicating the consistency in its performance.

We also simulated MB+MF and MBMF for different K and F respectively. A naive switching from MB to MF fails to improve the policy even for different switching points, and thus are outperformed by the pure MB approach (Fig. 6.16). The frequency F at which the

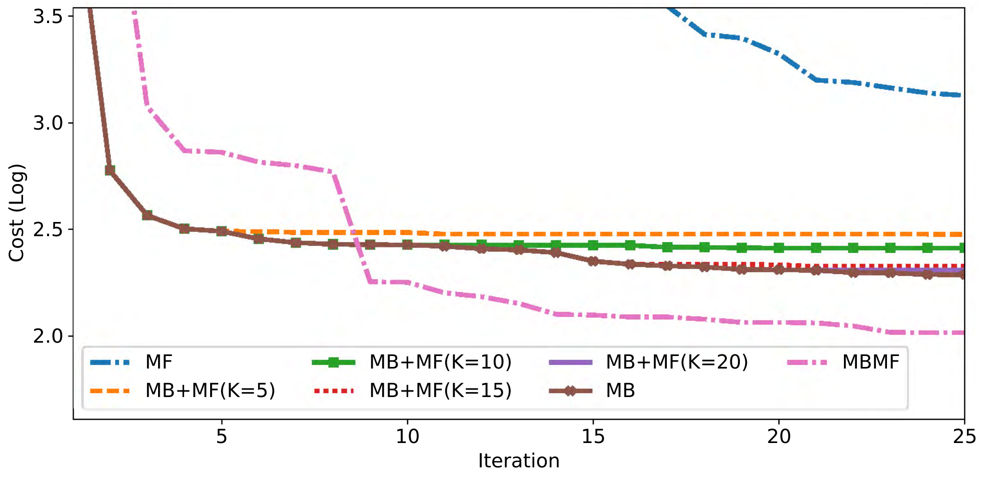


Figure 6.16: The mean cost obtained for different switching points K for the MB+MF approach for the 2D point mass system. Switching from MB to MF results in a flat learning curve in this case, indicating that a naive switching between the two may not be sufficient for the policy improvement.

prior is updated in the MBMF approach, however, affects the learning process (Fig. 6.17). We found that switching the model prior too frequently or too slowly both might lead to a suboptimal performance. Switching too often makes MBMF too sensitive to the changes in the dynamics model, which can change significantly especially early-on in the learning, and can "mis-guide" the policy exploration. On the other hand, switching too slowly may strip it of the full potential of the dynamics model. In this particular case, the optimal frequency turns out to be $F = 10$ (i.e., the MB prior is updated every 10 iterations). It might also be interesting to note that the MBMF approach is at least as good as the best baseline for all values of F. Nevertheless, systematically finding the optimal update frequency is an interesting future direction.

We also plot the trajectories obtained by executing the learned controller on the actual system for the MB, MF and MBMF approaches in Fig. 6.18. The initial and the goal positions are denoted by the Red and the Green circles respectively. For comparison purposes, the globally optimal trajectory (the dotted Red curve) was also computed, using the actual

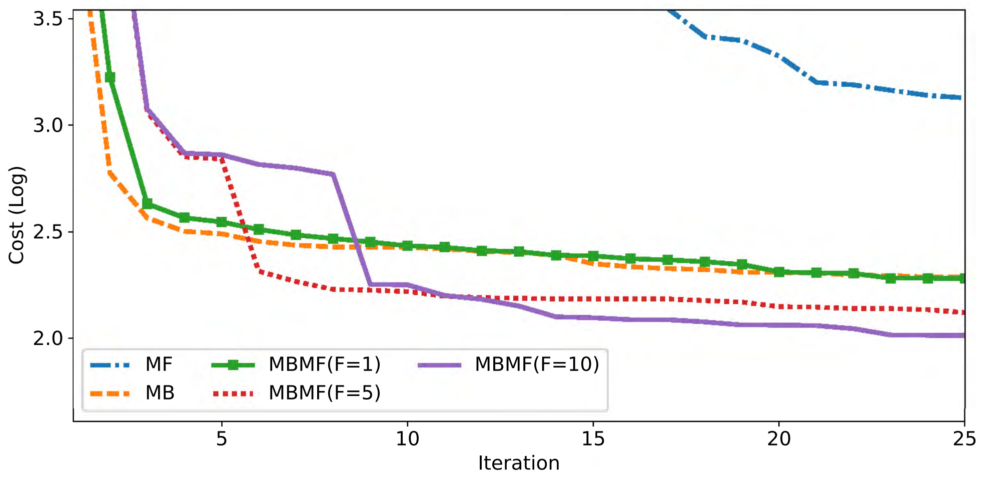
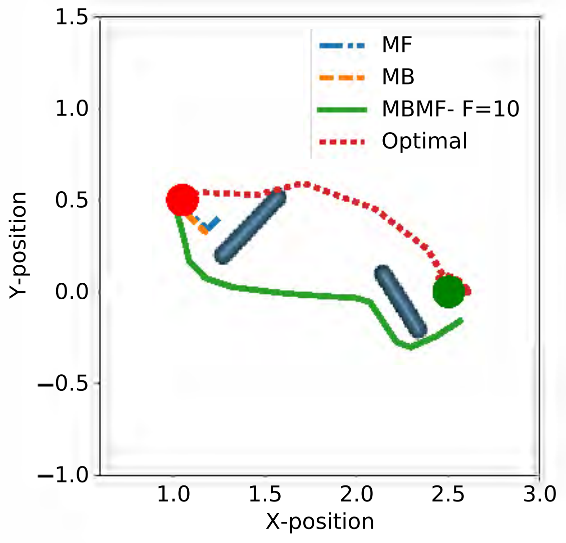


Figure 6.17: The mean cost obtained for different prior update frequencies F for the MBMF approach for the 2D point mass system. The learning efficiency of MBMF depends on the choice of the prior update frequency. Switching too often makes MBMF too sensitive to the changes in the dynamics model, which can "mis-guide" the policy exploration. On the other hand, if the prior update frequency is too small (F is large), then the MBMF lags behind the pure model-based approach, as it is not fully leveraging the dynamics model information. In this case, the optimal update frequency turns out to be $F = 10$; however, MBMF is at least as good as the best baseline for all update frequencies.

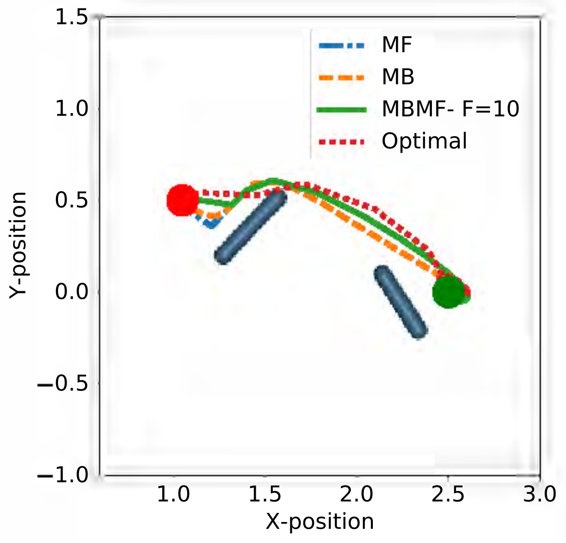
system dynamics obtained through MuJoCo; however, the dynamics are unknown to any of the learning method. We plot the trajectories for different trials, which correspond to different (but same across all methods) initial data. As evident from the figure, the MBMF approach is consistently able to reach the goal state, whereas the MB and MF approaches fail to achieve a consistent good performance. In particular, the optimal trajectory requires the system to overcome the obstacle next to the starting position. A pure MB approach is unable to consistently learn this behavior, potentially because it requires learning a discontinuous dynamics model. Consequently, it is often unable to reach the goal position and gets stuck in the obstacles (Figures 6.18a, 6.18d). Similarly, a pure MF approach is unable to learn to overcome the obstacles within 25 iterations. MBMF approach, however, can take evidence into account and is able to overcome this challenge to reach the goal state *consistently*, demonstrating its robustness to the training data, which is also evident from a lower variance in the performance of MBMF.

6.5.2.3 Three DoF Robotic Arm

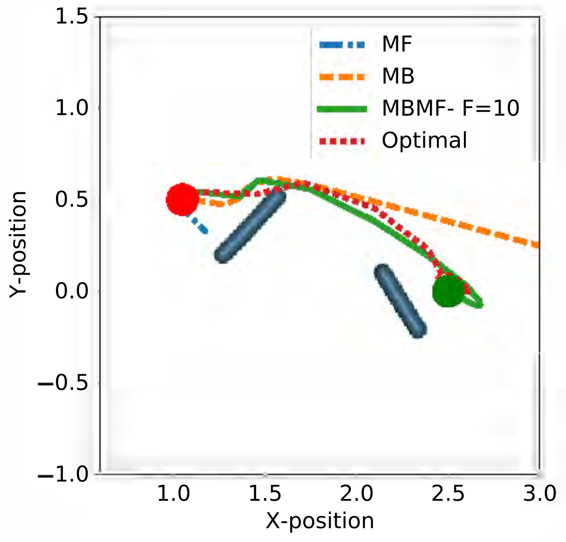
We again employ a GP-based dynamics model in this simulation. As evident from Fig. 6.19, MBMF(F=1) outperforms the other approaches and is continue to improve policy over iterations; however, due to the computational complexity of a GP-based dynamics model,



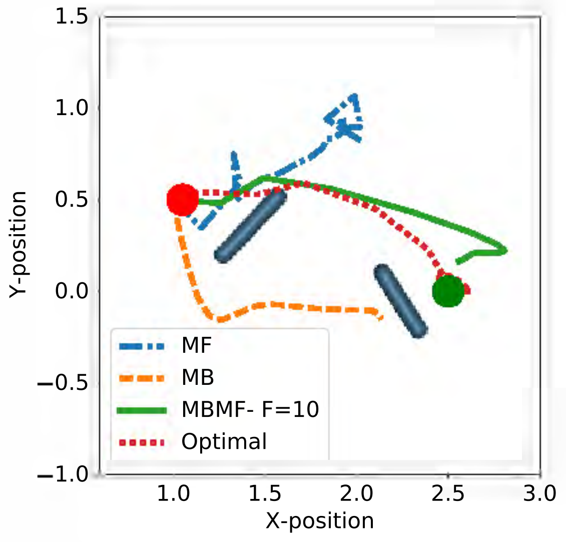
(a) Trial 1



(b) Trial 2



(c) Trial 3



(d) Trial 4

Figure 6.18: Trajectories obtained via executing the learned controller for the point mass system after 25 iterations. Each trial corresponds to different initial data, but was same across all approaches. The optimal trajectory requires the system to overcome the obstacles (the Grey cylinders) to reach from the initial position (the Red circle) to the goal position (the Green circle). MB and MF approaches have different behavior across different trials and they often get stuck in the obstacles. MBMF, on the other hand, is able to learn how to overcome the obstacles and consistently reaches the goal position.

we stop the learning process after 20 iterations. MB+MF approach(K=15) continues to improve after switching from MB to MF; however, it is still outperformed by the pure MB approach. We also note that MBMF has a significantly smaller variance compared to all other baselines, indicating that the MBMF approach is robust to the initial training data

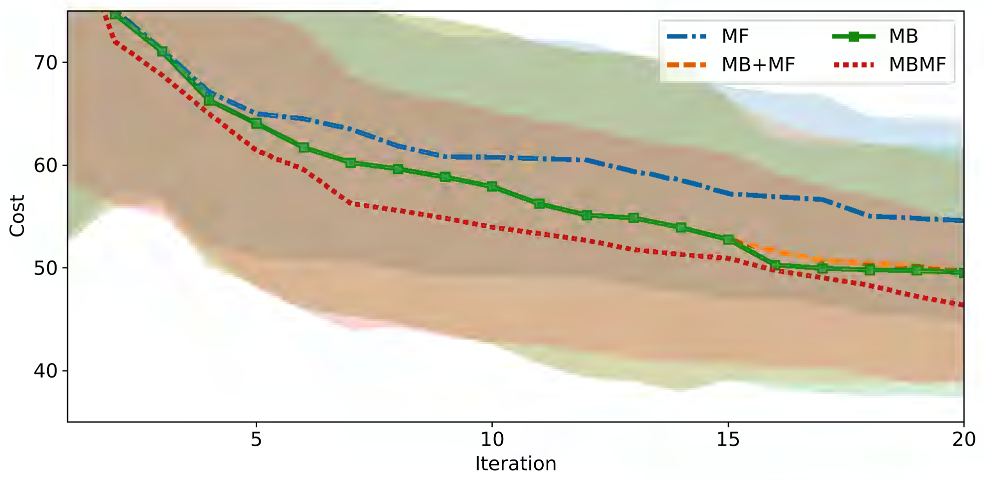


Figure 6.19: The mean (curves) and the standard deviation (shaded regions) of the cost obtained for different approaches for the three DoF robotic arm. MBMF leverages the advantages of both MB and MF approaches to design a better policy, indicating the data-efficiency of the MBMF approach, as well as its ability to overcome the model bias.

We also simulate the MB+MF and MBMF approaches for different Ks and Fs (see Fig. 6.20 and 6.21 respectively.) Interestingly, in this case, if the prior update frequency is too small (F is large), then the MBMF lags behind the pure MB approach, as it is not fully leveraging the dynamics model information. However, if the right update frequency is chosen, then MBMF can leverage the advantages of both MB and MF approaches and outperforms the two.

The corresponding trajectory comparison between MB and MBMF approaches in Figure 6.22 also highlight the efficacy of MBMF in leveraging the advantages of both the MB and MF components to quickly learn the optimal policy. A pure MB approach struggles with learning to move the object vertically in a straight line, potentially due to the complexity of the dynamics given the contact-rich nature of the task. The MBMF approach, on the other hand, has the capability to trade-off the observed costs and the predicted cost. As a result, it has been able to move the object closer to the goal position within a small number of iterations (20 in this case).

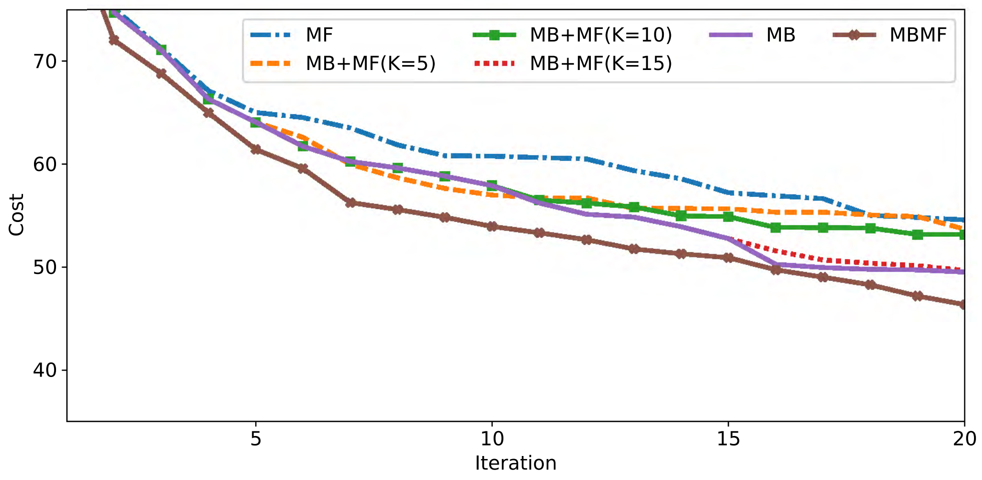


Figure 6.20: The mean cost obtained for different switching points K for the MB+MF approach for the robotic arm. Switching from MB to MF results in a slower learning compared to a pure MB approach.

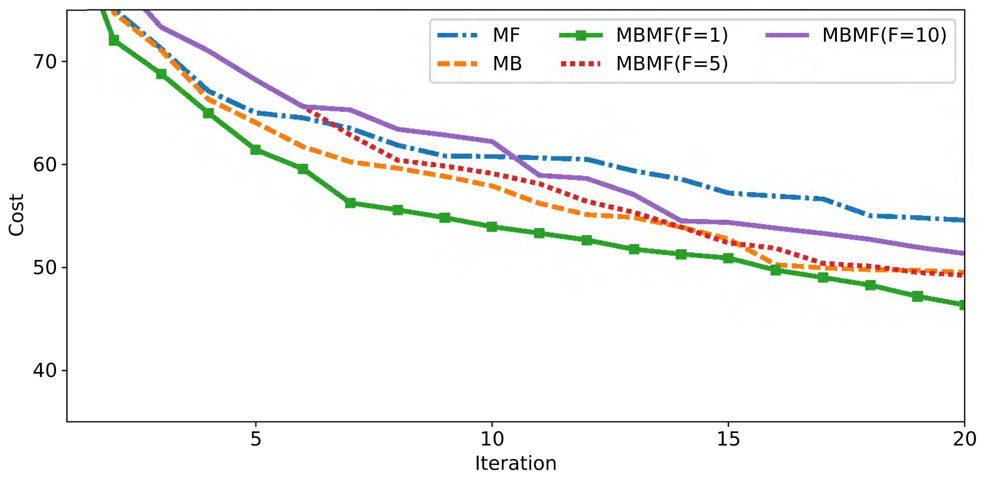
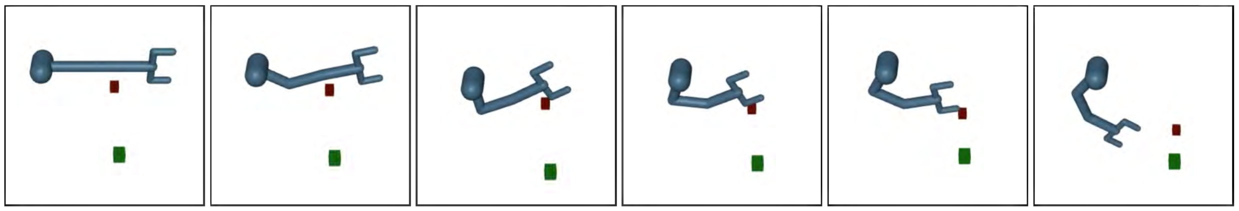
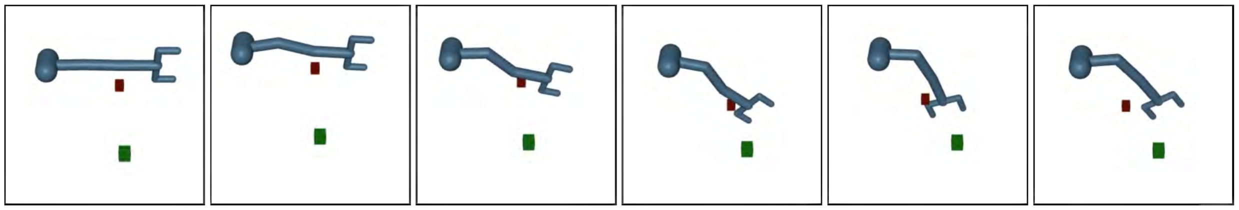


Figure 6.21: The mean cost obtained for different prior update frequencies F for the MBMF approach for the robotic arm. If the prior update frequency is too small (F is large), then the MBMF lags behind the pure MB approach in this case, as it is not fully leveraging the dynamics model information. In this case, the optimal update frequency turns out to be $F = 1$. Nevertheless, systematically finding the optimal prior update frequency is an important future direction.



(a) MBMF



(b) MB

Figure 6.22: (a) Trajectory obtained via executing the learned controller for the MBMF approach. The Red box represents the object, which needs to be moved to the Green box. MBMF is able to push the object fairly close to the goal position. (b) Trajectory obtained via executing the learned controller for the pure model-based approach. A pure MB approach struggles with accomplishing this task, with the final position of the object end up being very far from the goal position.

6.6 Chapter Summary

Real-world robots are becoming increasingly complex and commonly act in poorly understood environments where it is extremely challenging to model their dynamics from first principles. Even learning a model via indirect learning-based approaches can be prohibitive depending on the available data and the complexity of the underlying model. Therefore, it might be desirable to take a task-specific approach in such cases, wherein the focus is on explicitly learning the dynamics model which achieves the best control performance for the task at hand, rather than learning the true dynamics. In this chapter, we propose aDOBO, an active learning framework to learn a system dynamics model with the intent of maximizing the controller performance. Unlike traditional system identification approaches, the model is updated directly based on the performance observed in experiments on the physical system in an iterative manner until a desired performance is achieved. We demonstrate the efficacy of the proposed approach through simulations and experiments on a quadrotor testbed.

We then discuss how we can combine the complimentary advantages of indirect and direct learning-based approaches. To that end, we propose MBMF, that achieves this bridging by using the cost estimated by the model-based component as the prior for the model-free component. Our results show that the proposed approach can overcome the model bias and inaccuracies in the dynamics model, and yet retain the data-efficiency and generalization capabilities of model-based approaches.

Chapter 7

Learning for Unknown Environments: Perception-Based Control

This chapter is based on the papers "Combining optimal control and learning for visual navigation in novel environments" [38] and "Visual Navigation Among Humans with Optimal Control as a Supervisor" [291] written in collaboration with Varun Tolani, Saurabh Gupta, Aleksandra Faust, Jitendra Malik, and Claire Tomlin.

In Chapter 5 and 6, we discussed how we can use indirect and direct learning-based methods to capture the inaccuracies in the dynamics model of the system, and leverage learning to improve the control performance despite these inaccuracies. However, in many applications of interest, simple and well understood dynamics models are sufficient for control, and it is rather the vision and perception components that require learning. When a system operates in a priori unknown environment, such as an autonomous car operating around unknown construction sites in the city, perception enables the robot to interact with its environment and to decide how to act to complete the task they are supposed to. One such application that we will focus on in this chapter is that of autonomous navigation in *a priori* unseen environments based on onboard visual sensors.

Autonomous robot navigation, that is, how to convey a robot from one location to another, is one of the most fundamental and well-studied problems in robotics. Developing a fully autonomous robot that can navigate in *a priori* unknown environments is difficult due to challenges that span dynamics modeling, on-board perception, localization and mapping, trajectory generation, and optimal control.

One way to approach this problem is to generate a globally-consistent geometric map of the environment, and use it to compute a collision-free trajectory to the goal using optimal control and planning schemes. This approach is the basis of how a number of real physical systems are controlled, such as airplanes, autonomous cars, and industrial robots. However, the real-time generation of a globally consistent map tends to be computationally expensive, and can be challenging in texture-less environments or in the presence of transparent, shiny objects, or strong ambient lighting [14]. Moreover, explicit knowledge of the environment geometry or map may not even be required for navigation. When operating in dynamic environments, such as among humans, this approach additionally requires identification of the humans in the scene and prediction of their future motion. However, human recognition can be difficult because people come in different shapes and sizes, and might even be partially occluded. Human motion prediction, on the other hand, is challenging because the human's navigational goal (intent) is not known to the robot, and people have different temperaments and physical abilities which affect their motion (speed, paths etc.) [260, 308].

An alternative approach, that has gained a lot of traction recently in the learning community, employs end-to-end learning to side-step this explicit map estimation step. In end-to-end learning, navigation policies are learned to directly map on-board sensor readings to motor torques (or control commands in general) via imitation learning or reinforcement learning. The lack of any assumptions makes this research very attractive: policies can be learned without knowing anything about the underlying system, and policies can work in uninstrumented environments merely via partial views of the environment as obtained through inexpensive on-board sensors. However, such approaches tend to be extremely sample inefficient and highly specialized to the system they were trained on [254].

In this chapter, we will address the respective shortcomings of mapping-based approaches and learning-based approaches by coupling model-based control with learning-based perception. The use of learning in our pipeline will allow us to navigate in previously unseen buildings without any explicit map creation, thus bypassing the challenges associated with geometric maps. At the same time, the use of model-based control within the navigation pipeline addresses the data-inefficiency and lack of generalization issues associated with monolithic, end-to-end learning approaches. For navigation among humans, we will demonstrate that our modular navigation pipeline can navigate around humans in unknown indoor environments based only on monocular RGB images and does not require explicit state estimation and trajectory prediction of the human.

7.1 Related Work

An extensive body of research studies autonomous navigation. We cannot possibly hope to summarize all these works here, but we attempt to discuss the most closely related approaches.

Classical Robot Navigation. Classical robotics has made significant progress by factorizing the problem of robot navigation into sub-problems of mapping and localization [288, 122], path planning [190], and trajectory tracking. Mapping estimates the 3D structure of the world (using RGB / RGB-D images / LiDAR scans), which is used by a planner to compute paths to goal. However, such purely geometric intermediate representations do not capture navigational affordances (such as: to go far away, one should step into a hallway, etc.). Furthermore, mapping is challenging with just RGB observations, and often unreliable even with active depth sensors (such as in presence of shiny or thin objects, or in presence of strong ambient light) [14]. This motivates approaches that leverage object and region semantics during mapping and planning [60, 180]; however, such semantics are often hand-coded. Our work is similarly motivated, but instead of using geometry-based reasoning or hand-crafted heuristics, we aim to employ learning to capture such semantics without explicit map-building.

End-to-End (E2E) Learning for Navigation. There has been a recent interest in employing end-to-end learning for training policies for goal-driven navigation [311, 144, 173, 174]. The typical motivation here is to incorporate semantics and common-sense reasoning into navigation policies. While [311] learn policies that work well in training environments, the authors in [144] and [173] design policies that generalize to previously unseen environments. Most such works abstract out dynamics and work with a set of macro-actions (going forward x cm, turning $\theta°$). Such ignorance of dynamics results in jerky and inefficient policies that exhibit stop-and-go behavior on a real robot. Several works also use end-to-end learning for navigation using laser scans [80, 285, 308], for training and combining a local planner with a higher level roadmap for long range navigation [126, 112, 72, 22], or for visual servoing [261]. Numerous other works have tackled navigation in synthetic game environments [216, 266, 245], and largely ignore considerations of real-world deployment, such as dynamics and state estimation. Researchers have also employed learning to tackle locomotion problems [125, 165, 262, 166]. These works learn policies for collision-avoidance, i.e., how to move around in an environment without colliding. [165] uses motion primitives, while [125] and [262] use velocity control for locomotion. Other works such as [194] and [243] combine optimal control and end-to-end learning, by training neural network policies to mimic the optimal control-based control commands on the raw images. While all of these works implement policies for collision avoidance via lower level control, our work studies how policy learning itself should be modified to improve the data efficiency and generalization capabilities of E2E learning.

Combining Optimal Control and Learning for Navigation. A number of papers seek to combine the best of learning with optimal control for high-speed navigation [257, 102, 169, 164, 203]. For example, authors in [102, 103] learn a cost function from monocular images for aggressive race-track driving via Model Predictive Control (MPC). Instead, [168, 169] use learning to predict waypoints that are used with model-based control for drone racing. The focus of these works is on aggressive control in *training race-track environments*, with very few or no obstacles in the environment. This renders their approach for waypoint generation for learning (that does not reason about obstacles explicitly) ineffective in completely novel, cluttered real world testing environments. Authors in [228] predict waypoints from semantically segmented images and a user provided command for outdoor navigation, and use a PID controller for control. However, they do not explicitly handle obstacles and focuses primarily on lane keeping and making turns. Instead, rich, agile, and explicitly dynamically feasible and collision-free control behaviors are required for navigating in cluttered real-world indoor environments. Other works have used Riemannian motion policies [215] to handle dynamics constraints.

Visual Navigation Among Humans. The above methods primarily focus on navigation in static environments. For the problem of navigation in dynamic environments, such as among humans, the classical navigation pipeline is factorized into sub-problems of detection and tracking [62], human motion prediction [260], and planning [190]. However, reliable state estimation of the human might be challenging, especially when the robot is using narrow field-of-view sensors such as a monocular RGB camera. Moreover, human motion prediction itself can be quite challenging [117, 37] and is an active area of research [260].

Learning-based approaches have also been explored for navigation in dynamic environments. [248, 112, 80] use classical planning in static environments as the higher level planner, along with reinforcement learning for adaptive local planning and path tracking in dynamic environments, or train in photorealistic static environments [126] and evaluate in dynamic environments. This approach limits their ability to reason about the dynamic nature of human and planning optimal paths around it.

Other methods use RL to produce socially-compliant motion among humans [196]; however, the method requires human trajectories, and relies on detection and tracking algorithms to locate the humans. Yet other line of work uses depth sensors [247, 110, 71, 265, 107] to navigate in crowded spaces. These methods do not require high visual fidelity, but require expensive wide-field of view LiDAR sensors. However, it still remains a challenge to navigate around humans using *only* a monocular RGB image, without explicitly estimating human state or motion.

One of the challenges with learning-based approaches for navigation in dynamic environments is the lack of the availability of rich datasets. [211] proposes a dataset on multimodal social visual navigation, collected in real environments using real humans, manual annotation, and non-goal oriented navigation. Another benchmark on navigation [110] uses simulation for training, but is unsuitable for RGB-based visual navigation because humans in the scene have no visual texture and features, which are known to be important for closing the sim-to-real gap reliably [299]. However, we still lack the datasets that can simulate visually realistic humans and their motion, and where the data is collected in a goal-oriented navigation setting to enable rich human-robot interactions.

Contributions and Chapter Organization

In this chapter, we present a framework for autonomous, vision-based navigation in novel cluttered indoor environments based only on monocular RGB images received from an onboard camera. We take a factorized approach to navigation that uses *learning* to make high-level navigation decisions in unknown environments and leverages *optimal control* to produce smooth trajectories and a robust tracking controller. In particular, we train a Convolutional Neural Network (CNN) that incrementally uses the current RGB image observations to produce a sequence of intermediate states or *waypoints*. These waypoints are produced to guide the robot to the desired target location via a collision-free path in previously unknown environments, and are used as targets for a model-based optimal controller to generate smooth, dynamically feasible control sequences to be executed on the robot. We refer to our approach as *LB-WayPtNav* (Learning-Based WayPoint approach to Navigation).

LB-WayPtNav benefits from the advantages of classical control and learning-based approaches in a way that addresses their individual limitations. Learning can leverage statistical regularities to make predictions about the environment from partial views (RGB images) of the environment, allowing generalization to unknown environments. Leveraging underlying dynamics and feedback-based control leads to smooth, continuous, and efficient trajectories that are naturally robust to variations in physical properties and noise in actuation, allowing us to deploy our framework directly from simulation to real-world. Furthermore, learning now does not need to spend interaction samples to learn about the dynamics of the underlying system, and can exclusively focus on dealing with generalization to unknown environments.

To train the CNN, we propose a self-supervision method that uses Model Predictive Control (MPC) to generate rendered RGB images and corresponding optimal waypoints *entirely* using simulation environments. To deal with the challenges of obtaining the training data in the presence of humans, we additionally propose the Human Active Navigation Dataset (HumANav), consisting of photo-realistic renderings of humans moving in indoor environments. We then demonstrate the zero-shot, simulation-to-reality transfer of the learned navigation policies, without requiring any expensive demonstrations by an expert or causing any privacy and logistical challenges associated with human subjects.

To summarize our key contributions are:

- *an autonomous visual navigation method* that combines learning and optimal control to robustly maneuver the robot in novel, cluttered static and dynamic environments using only a single on-board RGB camera, and does not require explicit state estimation and trajectory prediction of the human;

- *self-supervised training scheme via MPC* for learning navigation policies without requiring any expensive demonstrations by an expert;

- through simulations and experiments on a mobile robot, we demonstrate that our approach is *better* and more *efficient* at reaching the goals, results in *smoother* trajectories, as compared to End-to-End learning, and more *reliable* than geometric mapping-based approaches;

- we demonstrate that our approach leads to a *zero-shot transfer* of learned policies from simulation to reality; and

- *HumANav*, an active dataset to benchmark visual navigation algorithms around humans.

The rest of the chapter is organized as follows. In Sec. 7.2, we formally define the problem of autonomous visual navigation in unknown environments. In Sec. 7.3, we discuss how we can combine learning and control to deal with this problem in static environments. In Sec. 7.4, we extend our approach to dynamic, human-centric environments.

7.2 Problem Formulation

In this work, we study the problem of autonomous navigation of a ground vehicle in previously unknown indoor environments. In this section, we focus only on static, unknown environments; dynamic environments will be discussed in Sec. 7.4.

We assume that the robot odometry is perfect (i.e., the exact vehicle state is available). Dealing with imperfect odometry is a problem in its own right, and we defer it to future work. We model our ground vehicle as a three-dimensional non-linear Dubins car system with dynamics:

$$\dot{x} = v\cos\phi, \quad \dot{y} = v\sin\phi, \quad \dot{\phi} = \omega\,, \tag{7.1}$$

where $z_t := (x_t, y_t, \phi_t)$ is the state (or pose) of the vehicle, $p_t = (x_t, y_t)$ is the position, ϕ_t is the heading, v_t is the speed, and ω_t is the turn rate at time t. The input (control) to the system is $u_t := (v_t, \omega_t)$. The linear and angular speeds v_t and ω_t are bounded within $[0, \bar{v}]$ and $[-\bar{\omega}, \bar{\omega}]$ respectively. We use a discretized version of the dynamics in Eqn. (7.1) for all planning purposes.

The robot is equipped with a monocular RGB camera mounted at a fixed height, oriented at a fixed pitch and pointing forwards. The goal of this paper is to learn control policies for goal-oriented navigation tasks: the robot needs to go to a target position, $p^* = (x^*, y^*)$, specified in the robot's coordinate frame (e.g., 11m forward, 5m left), without colliding with any obstacles. These tasks are to be performed in novel environments whose map or topology is not available to the robot. In particular, at a given time step t, the robot with state x_t receives as input an RGB image of the environment, I_t, and the target position $p_t^* = (x_t^*, y_t^*)$ expressed in the current coordinate frame of the robot. The objective is to obtain a control policy that uses these inputs to guide the robot to the target as quickly as possible.

7.3 Learning-Based Perception with Model-Based Control for Visual Navigation

We use a learning-based waypoint approach to navigation (LB-WayPtNav) for visual navigation in unseen, cluttered, static environments. The LB-WayPtNav framework is demonstrated in Figure 7.1 and summarized in Algorithm 9. In this section, we will provide more details on the architecture of LB-WayPtNav, as well as how we can train the learning components in this architcture.

Algorithm 7: Learning-Based WayPoint approach to Navigation (LB-WayPtNav)

```
1 Require $p^* := (x^*, y^*)$                                  // Goal location
2 for $t = 0$ to $T$ do
3     $z_t := (x_t, y_t, \phi_t)$;   $u_t := (v_t, \omega_t)$          // Measured robot pose, and
      linear/angular speed
4     Every $H$ steps do                                    // Replan every $H$ steps
5       $p_t^* := (x_t^*, y_t^*)$         // Goal location in the robot's coordinate frame
6       $\hat{w}_t = \psi(I_t, u_t, p_t^*)$                          // Predict next waypoint
7       $\{z^*, u^*\}_{t:t+H_p} = FitSpline(\hat{w}_t, u_t)$           // Plan spline-based smooth
        trajectory
8       $\{k, K\}_{t:t+H_p} = LQR(z^*_{t:t+H_p}, u^*_{t:t+H_p})$             // Tracking controller
9     $u_{t+1} = K_t(z_t - z_t^*) + k_t$                                // Apply control
```

7.3.1 Learning-Based WayPoint approach to Navigation (LB-WayPtNav)

LB-WayPtNav makes use of two submodules: perception and planning. The learning-based perception module produces a series of *waypoints* that guide the robot to the goal via a collision-free path. These waypoints are used by the model-based planning module to generate a smooth and dynamically feasible trajectory that is executed on the physical system using feedback control. We now describe each of these modules in more detail.

7.3.1.1 Perception Module

The goal of the perception module is to analyze the image and provide a high-level plan for the planning and control module. We implement the perception module using a CNN that takes as input a 224×224 pixel RGB image, I_t, captured from the onboard camera, the target position, p_t^*, specified in the vehicle's current coordinate frame, and vehicle's current linear and angular speed, u_t, and outputs the desired next state or a waypoint $\hat{w}_t := (\hat{x}_t, \hat{y}_t, \hat{\theta}_t) = \psi(I_t, u_t, p_t^*)$ (Line 6 in Algorithm 9).

Intuitively, the network can use the presence of surfaces and furniture objects like floors, tables, and chairs in the scene, alongside the learned priors about their shapes to generate an estimate of the next waypoint, without explicitly building an accurate map of the environment. This allows a more guided and efficient exploration in novel environments based on the robot's prior experience with similar scenes and objects. The CNN is trained using an automatically generated expert policy (Sec. 7.3.2).

7.3.1.2 Planning and Control Module

Given a waypoint $\hat{w}_t$, and the current linear and angular speed u_t, the planning module uses the system dynamics in Eqn. (7.1) to design a smooth trajectory, satisfying the dynamics and

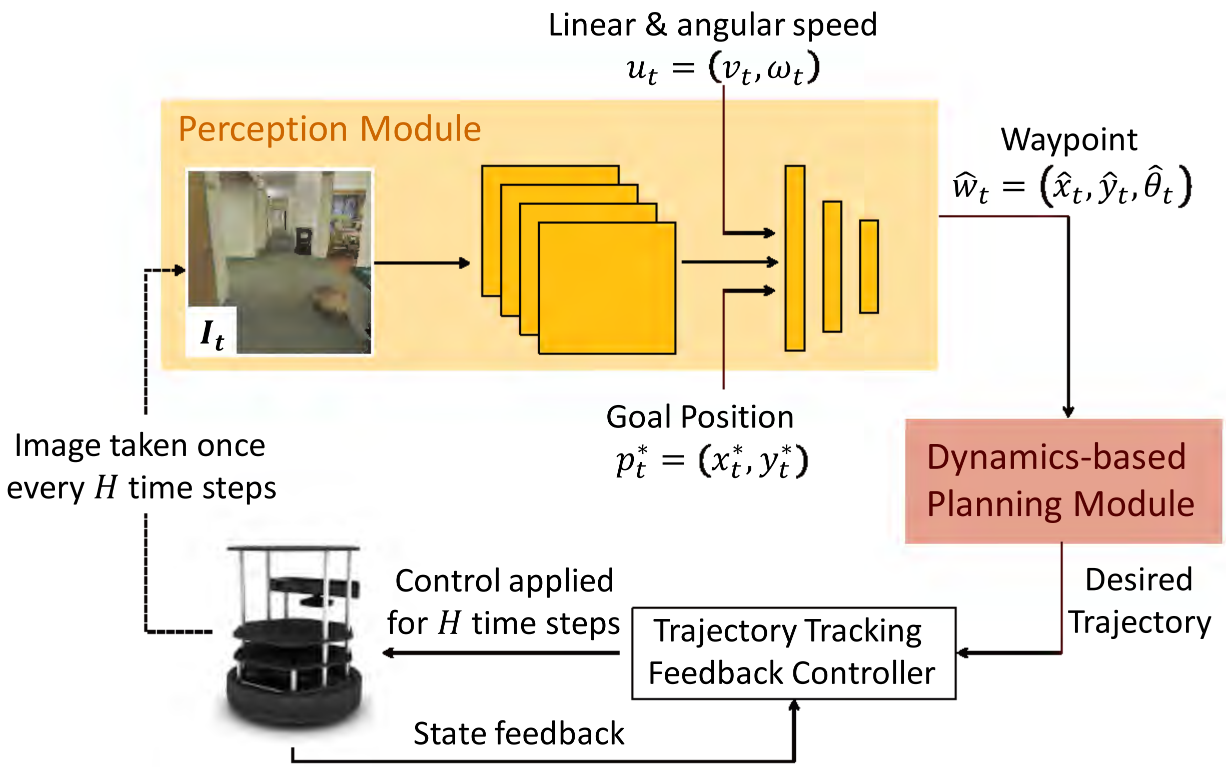

Linear & angular speed
$u_t = (v_t, \omega_t)$
Perception Module
I_t
Waypoint
$\hat{w}_t = (\hat{x}_t, \hat{y}_t, \hat{\theta}_t)$
Goal Position
$p_t^* = (x_t^*, y_t^*)$
Image taken once every H time steps
Dynamics-based Planning Module
Desired Trajectory
Control applied for H time steps
Trajectory Tracking Feedback Controller
State feedback

Figure 7.1: **Overview:** We consider the problem of navigation from a start position to a goal position. Our approach (LB-WayPtNav) consists of a learning-based perception module and a dynamics model-based planning module. The perception module predicts a waypoint based on the current first-person RGB image observation. This waypoint is used by the model-based planning module to design a controller that smoothly regulates the system to this waypoint. This process is repeated for the next image until the robot reaches the goal.

control constraints, from the current vehicle state to the waypoint. In this work, we represent the x and y trajectories using third-order splines, whose parameters can be obtained using $\hat{w}_t$ and u_t [300]. This corresponds to solving a set of linear equations, and thus, planning can be done efficiently onboard. Since the heading of the vehicle can be determined from the x and y trajectories, a spline-based planner ultimately provides the desired state and control trajectories, $\{z^*, u^*\}_{t:t+H_p} = FitSpline(\hat{w}_t, u_t)$, that the robot can follow for the time horizon $[t, t + H_p]$ to reach the waypoint $\hat{w}_t$ (Line 7).

Since the splines are third-order, the generated speed and acceleration trajectories are smooth. This is an important consideration for real robots, since jerky trajectories might lead to compounding sensor errors, poor tracking, or hardware damage [139]. While we use splines in this work for computational efficiency, other model-based planning schemes can also be used for trajectory planning.

To track the generated trajectory $\{z^*, u^*\}$, we design a LQR-based **linear feedback**

controller [54], $\{k,\ K\}_{t:t+H_p} = LQR(z^*_{t:t+H_p},\ u^*_{t:t+H_p})$ (Line 8). Here k and K represent the feed-forward and feedback terms respectively. The LQR controller is obtained using the dynamics in Eqn. (7.1), linearized around the trajectory $\{z^*, u^*\}$. LQR is a widely used feedback controller in robotic systems to make planning robust to external disturbances and mismatches between the dynamics model and the actual system [296]. This feedback controller allows us to deploy the proposed framework directly from simulation to a real robot (provided the real-world and simulation environments are visually similar), even though the model in Eqn. (7.1) may not capture the true physics of the robot.

The control commands generated by the LQR controller are executed on the system over a time horizon of H seconds (Line 8), and then a new image is obtained. Consequently, a new waypoint and plan is generated. This entire process is repeated until the robot reaches the goal position.

7.3.2 Data Generation Procedure and Training Details

We train the perception module entirely in simulation with self-supervision, using automatically generated RGB images and optimal waypoints as a source of supervision. The waypoints are generated so as to avoid the static obstacles and make progress towards the goal. To generate these waypoints, we assume that the location of all obstacles is known during training time. This is possible since we train the CNN in simulation. Under this assumption, we propose an MPC-based expert policy to generate optimal robot trajectories and waypoints. The proposed method does not require any human labeling and can be used to generate supervision for a variety of ground and aerial vehicles. Given first-person images and relative goal coordinates as input, the perception module is trained to predict these optimal waypoints. However, we note that the obstacle locations are not known during the test time and the robot relies only on an RGB image and other on-board sensors.

7.3.2.1 MPC-Based Expert Policy

To generate supervision for training the perception network, we use a Model Predictive Control (MPC) scheme to find a sequence of dynamically feasible waypoints and the corresponding spline trajectories that navigate the robot from the starting state to the goal state. This can be done during the training phase because a map of the environment is available during the training time. However, no such privileged information is used during the test time.

To generate the expert trajectory, we first sample a start position z_0 and a goal position p^* for the robot. Without loss of generality, we set the start state to be at origin. Given the start and the goal positions, at time t, we solve an MPC problem to determine the optimal trajectory of the robot over the time horizon $[t, t + H_p]$, denoted as $\{z^*, u^*\}_{t:t+H_p}$.

For the MPC problem, we define a cost function that trades-off the distance from the target position and the nearest obstacle, and optimize for the waypoint such that the resultant spline trajectory to the waypoint minimizes this cost function. More specifically, given the map of the environment, we compute the signed distance function to the obstacles, $d^{obs}(x, y)$, at any given position (x, y). Given the goal position, p^*, of the vehicle, we compute the minimum collision-free distance to the goal, $d^{goal}(x, y)$ (also known as the FMM distance to the goal). The overall cost function for the MPC problem is then given by:

$$J(\mathbf{z}, \mathbf{u}) = \sum_{i=0}^{T} J_i(z_i, u_i), \tag{7.2}$$

$$J_i(z_i, u_i) := \left(\max\{0, \lambda_1 - d^{obs}(x_i, y_i)\}\right)^3 + \lambda_2 \left(d^{goal}(x, y)\right)^2, \tag{7.3}$$

where $\mathbf{z} := (z_0, z_1, \ldots, z_T)$ is the state trajectory, $\mathbf{u}$ is the corresponding control trajectory, T is the maximum time horizon, and λ_2 trades-off the distance from the goal position and the obstacles. We only penalize for the obstacle cost when the corresponding robot trajectory is within a distance of λ_1 to an obstacle. Moreover, the obstacle cost is penalized more heavily compared to the goal distance (a cubic penalty vs a quadratic penalty). This is done to ensure that the vehicle trajectory does not go too close to the obstacles. We empirically found that it is significantly harder to learn to accurately predict the waypoints when the vehicle trajectory goes too close to the obstacles, as the prediction error margin for the neutral network is much smaller in such a case, leading to a much higher collision rate.

Given the cost function in (7.2), the overall MPC problem is given as

$$\min_{\mathbf{z},\mathbf{u}} J(\mathbf{z}, \mathbf{u}) \tag{7.4}$$

$$\text{subject to} \quad x_{i+1} = x_i + \Delta T v_i \cos\phi_i, \quad y_{i+1} = y_i + v_i \sin\phi_i, \quad \phi_{i+1} = \phi_i + \Delta T \omega_i\,, \tag{7.5}$$

$$v_i \in [0, \bar{v}], \quad \omega_i \in [-\bar{\omega}, \bar{\omega}]\,, \tag{7.6}$$

$$z_0 = (0, 0, 0), \tag{7.7}$$

where ΔT is the discretization step for the dynamics in (7.1), and the initial state and speed are assumed to be zero without loss of generality.

Starting from $i = 0$, we solve the MPC problem in (7.4) in a receding horizon fashion. In particular, at any timestep $i = t$, we find a waypoint such that the corresponding spline trajectory respects the dynamics and the control constraints in (7.5) and (7.6) (and hence is a dynamically feasible trajectory), and minimizes the cost in (7.4) over the time horizon $[t, t + H_p]$. Thus, we solve the following optimization problem:

$$\min_{\hat{w}_t} \sum_{i=t}^{t+H_p} J_i(z_i, u_i) \tag{7.8}$$

$$\text{subject to} \quad \{z, u\}_{t:t+H_p} = FitSpline(\hat{w}_t, u_t), \tag{7.9}$$

$$z_{t+H_p} = \hat{w}_t, \tag{7.10}$$

$$z_t, u_t \text{ - Given} \tag{7.11}$$

where $\hat{w}_t := (\hat{x}_t, \hat{y}_t, \hat{\theta}_t)$ is the waypoint, and $\{z, u\}_{t:t+H_1}$ are the corresponding state and control spline trajectories that satisfy the dynamics and the control constraints in (7.5) and

(7.6), and respect the boundary conditions on the trajectories imposed by z_t, u_t and $\hat{w}_t$. Such spline trajectories can be computed for a variety of aerial and ground vehicles (see [177, 213] for more details on the aerial vehicles and [300] for the ground vehicles).

Note that as per the above MPC problem, the feasible waypoints must be reachable from the current state and speed of the vehicle while respecting the vehicle's control bounds. For example, a pure rotation waypoint (i.e., $\hat{w}_t = (0, 0, \hat{\theta})$) is not feasible for the system if it has a non-zero linear speed at time t, and thus will not be considered as a candidate solution of (7.8). These dynamics considerations are often ignored in the learning-based methods in literature which work with a set of macro-actions. This often results in an undesirable, jerky, and stop-and-go behavior on a real robot. In this work, we use third-order polynomial splines, whose coefficients are computed using the values of z_t, u_t and $\hat{w}_t$ (see [300] for more details). Use of third-order splines make sure that the velocity and acceleration profiles of the vehicle are smooth, and respect the control bounds.

Given the low dimension of the waypoint (three in our case), we use a sampling-based approach to compute the optimal waypoint in (7.8). We sample the waypoints within the ground-projected field-of-view of the vehicle (ground projected assuming no obstacles). Note, even though we use a sampling-based approach to obtain the optimal waypoint, other gradient-based optimization schemes can also be used, especially when the state space of the vehicle is high-dimensional.

Let the optimal waypoint corresponding to the optimization problem in (7.8) be $\hat{w}_t^*$, and the corresponding optimal trajectories be $\{z^*, u^*\}_{t:t+H_p}$. The image obtained at state z_t^*, I_t, the relative goal position p_t^*, the speed of the robot u_t^*, and the optimal waypoint $\hat{w}_t^*$ thus constitutes one data point for the training.

We next apply the control sequence $u^*_{t:t+H_p}$ for the time horizon $[t, t+H_p]$ to obtain the state z^*_{t+H}, and repeat the entire procedure in (7.8) starting from time $t+H$. We continue this process until the robot reaches the goal position. In our work, we use $\lambda_1 = 0.3m, \Delta T = 0.05s, H_p = 6s$, $H = 1.5s$, and $\lambda_2 = 0.00064$. The parameter tuning was done manually in our work.

The procedure outlined in this section allows us to compute large training datasets using optimal control without requiring any explicit human labeling. Moreover, the generated waypoints are guaranteed to satisfy the dynamics and control constraints. Finally, the cost function in (7.2) allows us to explicitly ensure that the robot trajectory to the waypoint is collision-free, which is crucial in cluttered indoor environments. It is also worthwhile to note that this procedure can be applied to a variety of ground vehicles and aerial vehicles that are differentially flat. For differentially flat systems, the path planning can be done with respect to a much lower-dimensional state space (or waypoint) using spline trajectories [177, 300, 213], which makes the path planning tractable in real-time.

Remark 14 *Intuitively the waypoint in our framework summarizes the local obstacle information in the environment, and hence, its representation choice is a very crucial design choice. For example, the choice of $\hat{\theta}$ in $\hat{w}$ affects the shape of the trajectory the vehicle takes to the waypoint (see Figure 7.2). Typically, $\hat{\theta}$ is chosen as the line of sight angle between* *the robot's current position and the desired position $(\hat{x}, \hat{y})$ [169, 228]. However, $\hat{\theta}$ is an extra degree-of-freedom that can be chosen appropriately to generate rich, agile and collision-free trajectories in cluttered environments.*

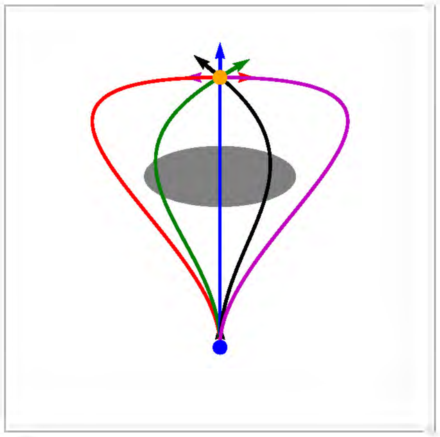


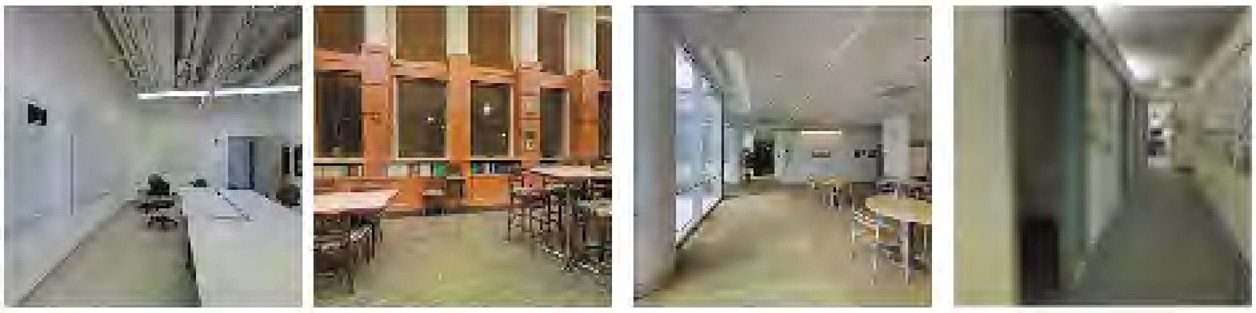
Figure 7.2: We visualize the spline trajectories produced by our planner for different waypoint angles $(\hat{\theta})$, starting from the same initial state, initial speed and to the same waypoint position. The gray region denotes an obstacle. Due to the dynamics constraints, $\hat{\theta}$ significantly affects the shape of the vehicle trajectory, and hence needs to be chosen appropriately to obtain a collision-free trajectory. In particular, the line of sight trajectory to the waypoint (the Blue trajectory) leads to a collision in this case, whereas if $\hat{\theta}$ is chosen appropriately, a smooth, agile trajectory that goes around the obstacle can be obtained.

7.3.2.2 Training Details

As discussed earlier, we train the perception module in LB-WayPtNav entirely in simulation. We use scans of real world buildings (from the Stanford large-scale 3D Indoor Spaces dataset [23]) and the MPC-based expert policy (see Sec. 7.3.2.1). Scans from 2 buildings were used to generate the training data. We illustrate some representative scenes from the test and the training buildings from the S3DIS dataset in Figure 7.3a. Note that the navigation tasks were not limited to these scenes and are spread across the entire building.

Implementation Details. We used a pre-trained ResNet-50 [146], pre-trained for ImageNet Classification, as the CNN backbone for the perception module in LB-WayPtNav, and finetuned it for waypoint prediction with MSE loss using the Adam optimizer. We remove the top layer of the ResNet, and use a downsampling convolution layer, followed by 5 fully connected layers with 128 neurons each to regress to the optimal waypoint. The image features obtained at the last convolution layer are concatenated with the egocentric target position and the current linear and angular speed before passing them to the fully connected layers (see Figure 7.1). During training, the ResNet layers are finetuned along with the fully connected layers to learn the features that are relevant to the navigation tasks.

We train the CNN on 125K data points generated by our expert policy (Section 7.3.2.1). All our models are trained with a single GPU worker using TensorFlow [2]. We use MSE loss on the waypoint prediction for training the CNN in our perception module. We use Adam optimizer to optimize the loss function with a batch size of 64. We train the CNN for 35K


(a) Training

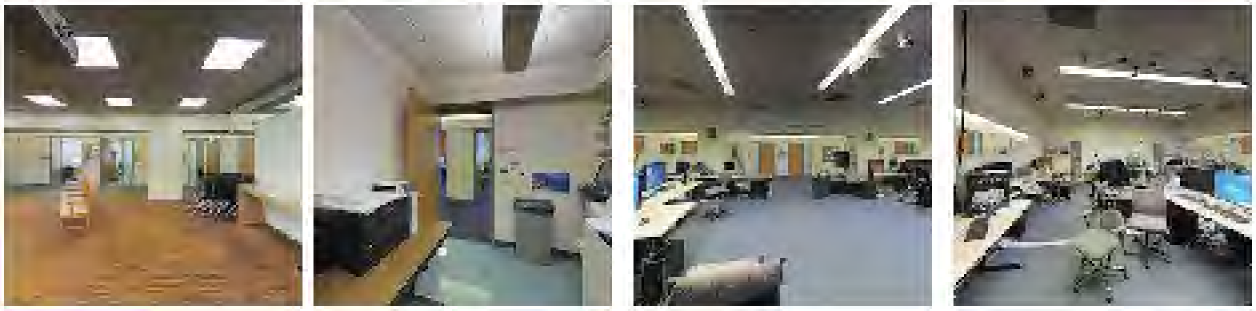

(b) Test

Figure 7.3: Some representative images of the buildings from which the training and the test data was collected. Even though the test environments are also office buildings, their layouts and appearances are different than the training buildings. However, our framework is still able to generalize to the domain shift.

iterations with a constant learning rate of 10^{-4} and use a weight decay of 10^{-6} to regularize the network.

We use standard techniques used to avoid overfitting including dropout following each fully connected layer except the last (with dropout probability 20%) [280]. During training, we also apply a variety of random distortions to images, such as adding blur, removing some pixels, adding some superpixels, changing image saturation, sharpness, contrast and brightness [277]. We also apply perspective distortions to images, such as varying the field-of-view and tilt (pitch) of the camera. For adding these distortions, we use the *imgaug* python library.

Some examples of sample distortions are shown in Figure 7.4. Adding these distortions significantly improves the generalization capability of our framework to unseen environments. We will discuss the quantitative benefit of adding these distortions in Section 7.3.3.

7.3.3 Simulation Results

LB-WayPtNav is aimed at combining classical optimal control with learning for interpreting images. In this section, we present experiments in simulation, and compare to representative

(a) Undistorted image

(b) Adding superpixels

(c) Gaussian blur

(d) Adding motion blur

(e) Image sharpening

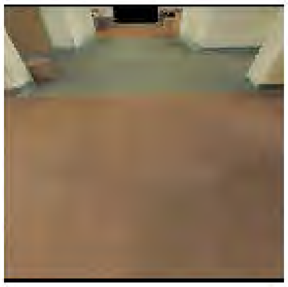
(f) Gaussian noise

(g) Brightness change

(h) Dropping pixels

(i) Changing saturation

(j) Changing contrast


(k) Changing FoV

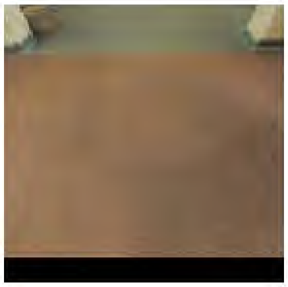

(l) Camera tilt change

Figure 7.4: Examples of several image distortions that have been randomly applied during the training phase. The actual undistorted image is shown in (a). Adding these distortions significantly improves the generalization capability of our framework to unseen environments.

methods that only use E2E learning (by ignoring all knowledge about the known system), and that only use geometric mapping and path planning (and no learning).

Simulation Setup: Our simulation experiments are conducted in environments derived from the Stanford large-scale 3D Indoor Spaces dataset [23]. Scans from 2 buildings were used to generate training data to train LB-WayPtNav. 185 test episodes (start, goal position pairs) in a 3rd *held-out* building were used for testing the different methods. Test episodes are sampled to include scenarios such as: going around obstacles, going out of the room, going from one hallway to another. Though training and test environments consists

Table 7.1: **Quantitative Comparisons in Simulation:** Various metrics for different approaches across the test navigation tasks: success rate (higher is better), average time to reach goal, jerk and acceleration along the robot trajectory (lower is better) for successful episodes. LB-WayPtNav conveys the robot to the goal location more often, faster, and produces considerably less jerky trajectories than E2E learning approach. Since LB-WayPtNav only uses the current RGB image, whereas the geometric mapping and planning approach integrates information from perfect depth images, it outperforms LB-WayPtNav in simulation. However, performance is comparable when the mapping based approach only uses the current image (like LB-WayPtNav, but still depth vs. RGB).

Agent	**Input**	**Success (%)**	**Time taken (s)**	**Acceleration** (m/s^2)	**Jerk** (m/s^3)
Expert	Full map	100	10.78 ±2.64	0.11 ±0.03	0.36 ±0.14
LB-WayPtNav (our)	RGB	80.65	11.52 ±3.00	0.10 ±0.04	0.39 ±0.16
End To End	RGB	58.06	19.16 ±10.45	0.23 ±0.02	8.07 ±0.94
Mapping (memoryless)	Depth	86.56	10.96 ±2.74	0.11 ±0.03	0.36 ±0.14
Mapping	Depth + Spatial Memory	97.85	10.95 ±2.75	0.11 ±0.03	0.36 ±0.14

of indoor offices and labs, their layouts and visual appearances are quite different (see Fig. 7.3 for some images).

Comparisons: We compare to two alternative approaches. *E2E Learning:* This approach is trained to directly output the velocity commands corresponding to the optimal trajectories produced by the spline-based planner. This represents a purely learning-based approach that does not explicitly use any system knowledge at test time. For E2E learning, we use $u^*_{t:t+H_p}$ instead of $\hat{w}^*_t$ for training the CNN; moreover, we use the same trajectories used to generate supervision for LB-WayPtNav. *Geometric Mapping and Planning:* This approach represents a learning-free, purely geometric approach. As inferring precise geometry from RGB images is challenging, we provide ideal depth images as input to this approach. These depth images are used to incrementally build up an occupancy map of the environment, that is used with the same spline-based planner, that was used to generate the expert supervision (see Sec. 7.3.2.1), to output the velocity controls. Results reported here are with control horizon, $H = 1.5s$. We also tried $H = 0.5, 1.0$, but the results and trends were the same.

Metrics: We make comparisons via the following metrics: success rate (if the robot reaches within $0.3m$ of the goal position without any collisions), the average time to reach the goal (for the successful episodes), and the average acceleration and jerk along the robot trajectory. The latter metrics measure execution smoothness and efficiency with respect to time and power.

7.3.3.1 Results

Comparison with the End-to-End learning approach. Table 7.1 presents quantitative comparisons. We note that LB-WayPtNav conveys the robot to the goal location more often (22% higher success rate), much faster (40% less time to reach the goal), and with less power consumption (50% less acceleration). Figure 7.5(left) shows top-view visualization of trajectories executed by the two methods. Top-views maps are being only used for visualization, both methods operate purely based on first-person RGB image inputs. As LB-WayPtNav uses a model-based planner to compute exact controls, it only has to learn "where to go" next, as opposed to the E2E method that also needs to learn "how to go" there. Consequently, LB-WayPtNav is able to successfully navigate through narrow hallways, and make tight turns around obstacles and corners, while E2E method struggles. This is further substantiated by the velocity control profiles in Figure 7.5(right). Even though the E2E method was trained to predict smooth control profiles (as generated by the expert policy), the control profiles at test time are still discontinuous and jerky. We also experimented with adding a smoothing loss while training the E2E policy; though it helped reduce the average jerk, there was also a significant decline in the success rate. This indicates that learning both an accurate and a smooth control profile can be a hard learning problem. In contrast, as LB-WayPtNav uses model-based control for computing control commands, it achieves average acceleration and jerk that is as low as that of an expert.[1] This distinction has a significant implication for actual robots since the consumed power is directly proportional to the average acceleration. Hence, for the same battery capacity, LB-WayPtNav will drive the robot twice as far as compared to E2E learning.

Comparison with Geometric Mapping and Planning Approach. We note that an online geometric mapping and planning approach, when used with ideal depth image observations, achieves near-expert performance. This is not surprising as perfect depth and egomotion satisfies the exact assumptions made by such an approach. Since LB-WayPtNav is a reactive planning framework, we also compare to a memory-less version that uses a map derived from *only* the current depth image. Even though the performance of memory-less planner is comparable to LB-WayPtNav, it still outperforms slightly due to the perfect depth estimation in simulation. However, since real-world depth sensors are neither perfect nor have an unlimited range, we see a noticeable drop in the performance of mapping-based planners in real-world experiments as discussed in real world experiments in Section 7.3.4.

Visualization of Learned Navigation Affordances. We conducted some analysis to understand what cues LB-WayPtNav leverages in order to solve navigation tasks. Figure 7.6 shows two related navigation tasks where the robot is initialized in the same state, but is tasked to either go inside a close by room (Case A), or to a far away room that is further down the hallway (Case B). LB-WayPtNav correctly picks appropriate waypoints, and is

[1]For a fair comparison, we report these metrics only over the test tasks that all approaches succeed at.

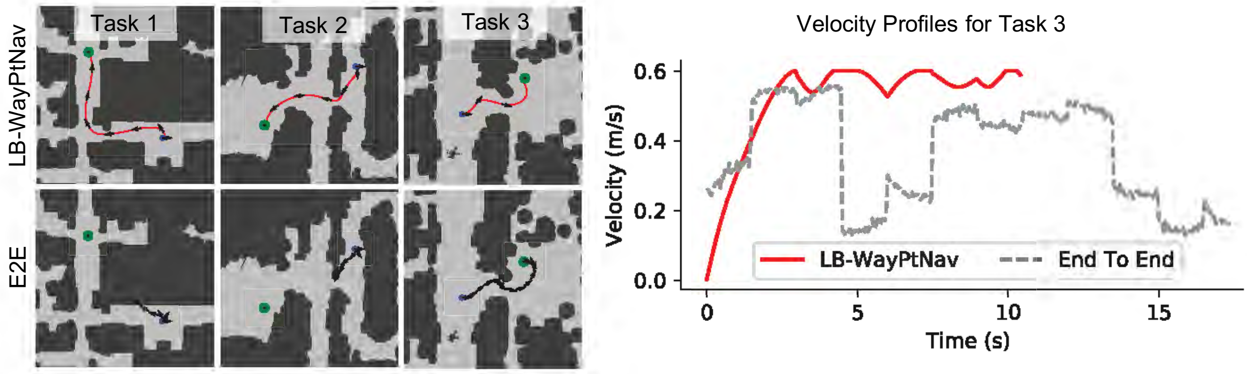


Figure 7.5: **Trajectory Visualization:** We visualize the trajectories produced by the model-based planning approach (top row) and the end-to-end (E2E) learning approach (bottom row) for sample test tasks. The E2E learning approach struggles to navigate around the tight corners or narrow hallways, whereas LB-WayPtNav is able to produce a smooth, collision-free trajectory to reach the target position. Even though both approaches are able to reach the target position for task 3, LB-WayPtNav takes only 10s to reach the target whereas the E2E learning approach takes about 17s. Moreover, the control profile produced by the E2E learning approach is significantly more jerky than LB-WayPtNav, which is often concerning for real robots as they are power inefficient, can lead to significant errors in sensors and cause hardware damage.

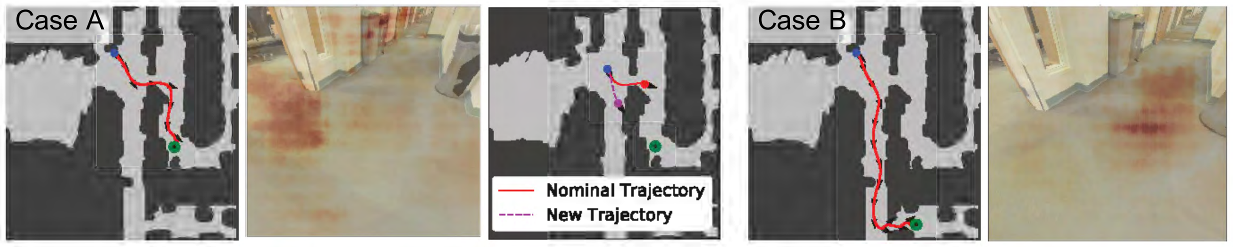


Figure 7.6: LB-WayPtNav is able to learn the appropriate navigation cues, such as entering the room through the doorway for a goal inside the room, continuing down the hallway for a farther goal. Such cues enable the robot to navigate efficiently in novel environments.

able to reason that a goal in a far away room is better achieved by going down the hallway as opposed to entering the room currently in front of the robot.

We also conduct an occlusion sensitivity analysis [307], where we measure the change in the predicted waypoint as a rectangular patch is zeroed out at different locations in the input image. We overlay the magnitude of this change in prediction on the input image in Red. LB-WayPtNav focuses on the walls, doorways, hallways and obstacles such as trash-cans as it predicts the next waypoint, and what the network attends to depends on where the robot is trying to go. Furthermore, for Case A, we also show the changed waypoint (in pink) as we zero out the wall pixels. This corresponds to a shorter path to the goal in the absence of the wall. More such examples can be found in [38].

Effect of distortions. Adding perspective and image distortions during training significantly improves the generalization of the trained network to unseen environments. For example, for LB-WayPtNav, adding distortions increases the success rate from 47.94% to 80.65%. Adding perspective distortions particularly improves the generalization to the real-world systems, for which the camera tilt will inevitably change as the robot is moving through the environment.

Failure Modes. Even though LB-WayPtNav is able to perform navigation tasks in novel environments, it can only do local reasoning (there is no memory in the network) and gets stuck in some situations. The most prominent failure modes are: a) when the robot is too close to an obstacle, and b) situations that require 'backtracking' from an earlier planned path.

7.3.4 Hardware Experiments

We next test LB-WayPtNav on a TurtleBot 2 hardware testbed. The TurtleBot 2 is a low-cost, open source differential drive robot, which we equip with an Orbbec Astra RGB-D camera. However, for LB-WayPtNav and E2E learning experiments, we only used the RGB image. For geometric mapping and planning-based schemes, we additionally use the depth image. A snapshot of our testbed is shown in Figure 7.7.

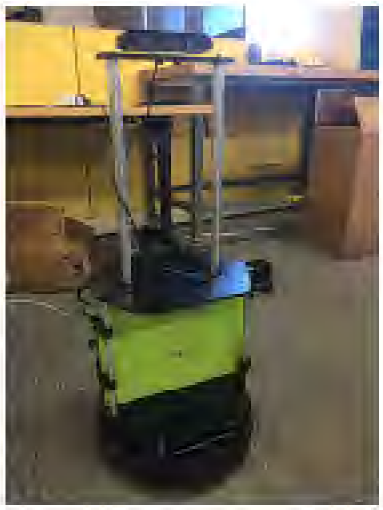


Figure 7.7: Our Turtlebot 2 hardware platform uses a Yujin Kobuki base, Gigabyte Aero Laptop, and Orbbec Astra camera.

We use the network trained in simulation, as described in Section 7.3.3, and deploy it directly on the TurtleBot without any additional training or finetuning. We tested the robot in two different buildings, neither of which is in the training dataset (in fact, not even in the S3DIS dataset). We show some representative images of our experiment environments in Figure 7.8.

The bulk of computation, including the deep network and the planning, runs on an onboard computer (Nvidia GTX 1060). The camera is attached to the onboard computer through a USB and supplies the RGB images. Given an image and the relative goal position, the onboard computer predicts the next waypoint for the robot, plans a spline trajectory to that waypoint, as well as computes the low-level control commands and the corresponding feedback controller. The desired speed and angular speed commands are sent to the Kobuki base, which then converts them to PWM signals to execute on the robot. For state measurement, we use the on-board odometry sensors on the TurtleBot.

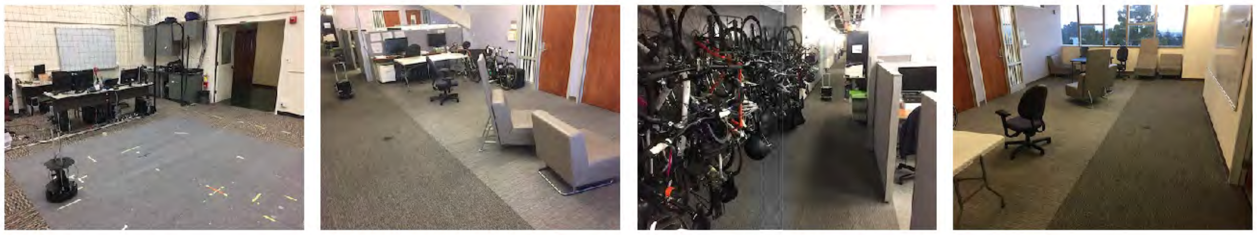


Figure 7.8: Some representative images of the buildings in which the experiments were conducted. None of these buildings were used for training/testing purposes in simulation.

Test environments for the experiments are described in Table 7.2. We repeat each experiment for our method and the three baselines: E2E learning, mapping-based planner, and a memoryless mapping-based planner, for 5 trials each. Results across all 20 trials are summarized in Table 7.3, where we report success rate, time to reach the goal, acceleration and jerk. Our experiment videos can be found on the project website[2].
Comparison to E2E learning are consistent with our conclusions from simulation experiments. LB-WayPtNav results in more reliable, faster, and smoother robot trajectories.

Comparison to Geometric Mapping and Planning. Geometric mapping and planning is implemented using the RTAB-Map package [185]. RTAB-Map uses RGB-D images as captured by an on-board RGB-D camera to output an occupancy map that is used with the spline-based planner to output motor commands. As our approach only uses the current image, we also report performance of a memory-less variant of this baseline where occupancy information is derived only from the current observation. While LB-WayPtNav is able to solve 95% of the trials, this memory-less baseline completely fails. It tends to convey the robot too close to obstacles, and fails to recover. In comparison, the map building scheme performs better, with a 40% success rate. This is still a lot lower than performance of our method (95%), and near perfect performance of this scheme in simulation. We investigated the reason for this poor performance, and found that this is largely due to imperfections in depth measurements in the real world. For example, the depth sensor fails to pick-up shiny bike-frames and helmets, black bike-tires and monitor screens, and thin chair legs and power strips on the floor. In Figure 7.9, we illustrate some examples of inaccurate depth estimations that we encounter during our experiments. These systematically missing obstacles cause the robot to collide in experiment 1 and 2. Map quality also substantially deteriorates in the presence of strong ambient light, such as when the sunlight comes in through the window in experiment 4. These are known fundamental issues with depth sensors, that limit the performance of classical navigation stacks that crucially rely on them.

[2] Project Website: https://smlbansal.github.io/LB-WayPtNav/

Table 7.2: Experiment setups, with top-views (obtained offline only for visualization), and sample images. Robot starts at the blue dot, and has to arrive at the green dot. Path taken by LB-WayPtNav is shown in red.

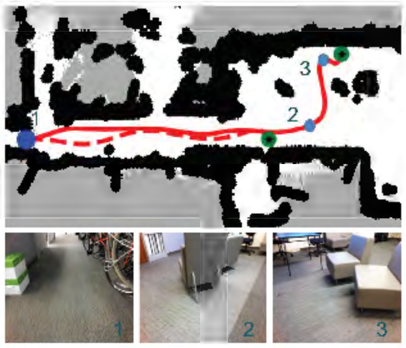
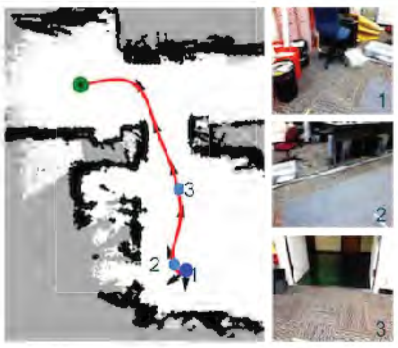
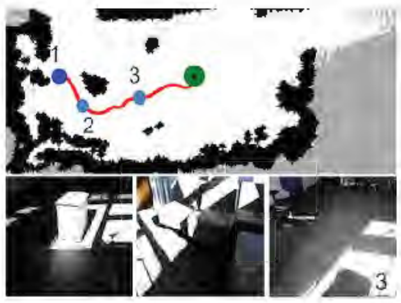
Experiment 1 and 2	**Experiment 3**	**Experiment 4**
		
Navigation through cluttered environments: This tests if the robot can skillfully pass through clutter in the real world: a narrow hallway with bikes on a bike-rack on one side, and an open space with chairs and sofas.	**Leveraging navigation affordances:** This tests use of semantic cue for effective navigation. Robot starts inside a room facing a wall. Robot needs to realize it must exit the room through the doorway in order to reach the target location.	**Robustness to lighting conditions:** Experiment area is similar to that used for experiment 1, but experiment is performed during the day when sunlight comes from the windows. Robot needs to avoid objects to get to the goal.

Table 7.3: **Quantitative Comparisons for Hardware Experiments:** We deploy LB-WayPtNav and baselines on a TurtleBot 2 hardware testbed for four navigation tasks for 5 trials per task. We report the success rate (higher is better), average time to reach goal, jerk and acceleration along the robot trajectory (lower is better).

Agent	**Input**	**Success** (%)	**Time taken** (s)	**Acceleration** (m/s^2)	**Jerk** (m/s^3)
LB-WayPtNav (our)	RGB	95	22.93 ±2.38	0.09 ±0.01	3.01 ±0.38
End To End	RGB	50	33.88 ±3.01	0.19 ±0.01	6.12 ±0.18
Mapping (memoryless)	RGB-D	0	N/A	N/A	N/A
Mapping	RGB-D + Spatial Memory	40	22.13 ±0.54	0.11 ±0.01	3.44 ±0.21

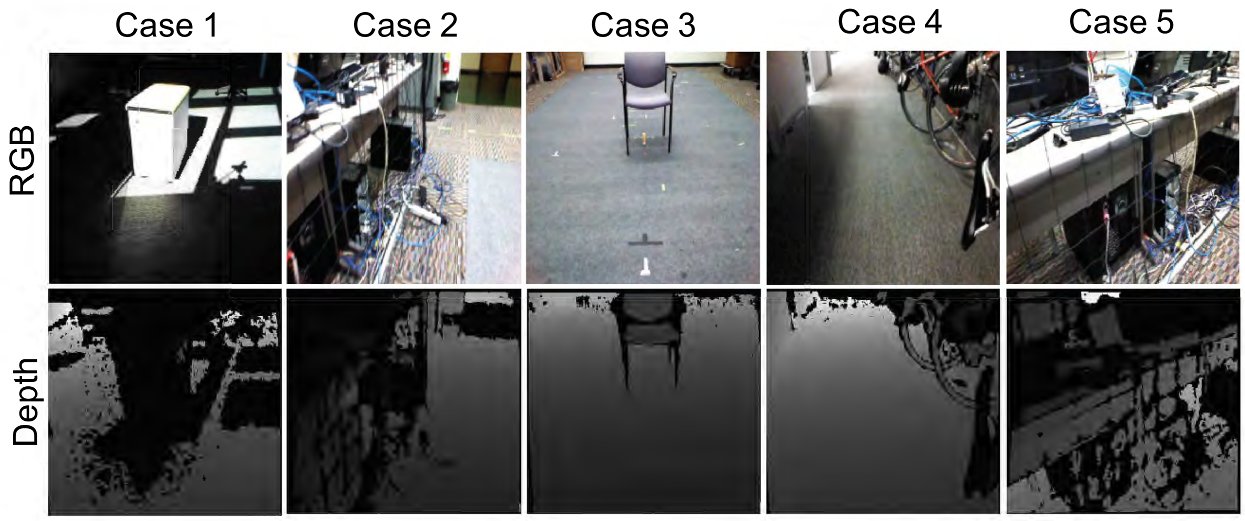


Figure 7.9: We visualize the RGB images captured by the robot and corresponding depth estimation. The black pixels in the depth images correspond to the regions where the depth estimator fails to accurately estimate the depth. The depth estimation is inaccurate when the robot encounters shiny, thin, or transparent objects, or in the presence of strong ambient lighting, such as sunlight. This results in a significant decline in the performance of a mapping-based approach.

Performance of LB-WayPtNav: In contrast, our proposed learning-based scheme that leverages robot's prior experience with similar objects, operates much better without need for extra instrumentation in the form of depth sensors, and without building explicit maps, for the short-horizon tasks that we considered. LB-WayPtNav is able to precisely control the robot through narrow hallways with obstacles (as in experiment 1 and 2) while maintaining a smooth trajectory at all times. This is particularly striking, as the dynamics model used in simulation is only a crude approximation of the physics of a real robot (it does not include any mass and inertia effects, for example). The LQR feedback controller compensates for these approximations, and enables the robot to closely track the desired trajectory (to an accuracy of $4cm$ in experiment 1). LB-WayPtNav also successfully leverages navigation cues (in experiment 3 when it exits the room through a doorway), even when such a behavior was never hard-coded. Furthermore, thanks to the aggressive data augmentation, LB-WayPtNav is able to perform well even under extreme lighting conditions as in Experiment 4. This demonstrates zero-shot, sim-to-real transfer capabilities of LB-WayPtNav.

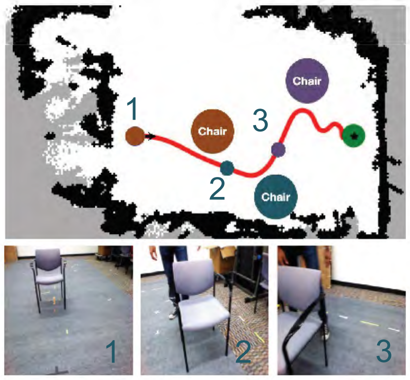


Figure 7.10: LB-WayPtNav can adapt to dynamic environments.

Furthermore, LB-WayPtNav is agile and reactive. It can adapt to changes in the environment. In an additional experiment (shown in Figure 7.10), we change the environment as the robot executes its policy. The robot's goal is to go straight $6m$. It must go around the brown chair. As the policy is executed, we move the chair to repeatedly block the robot's path (we show the new chair locations in blue and purple, and mark the corresponding positions of the robot at which the chair was moved by same colors). We decrease the control horizon to $0.5s$ for this experiment to allow for a faster visual feedback. The robot successfully reacts to the moving obstacle and reaches the target without colliding.

7.4 Visual Navigation in Dynamic, Human-Centric Environments

In Section 7.3, we discussed how we can combine a learning-based perception module and a dynamics model-based planning and control module to navigate in unknown static environments. We now discuss how we can extend the proposed framework for navigation among humans whose trajectories are unknown to the robot. We call this framework LB-WayPtNav-DH, where DH stands for Dynamic Human. Similar to LB-WayPtNav, we will train the perception module in LB-WayPtNav-DH to predict a waypoint that avoids the static obstacles and make progress towards the goal, but it now additionally anticipates the future human motion and react to it. As before, this training will be done entirely in simulation, and the learned policies will be transferred directly to real robots.

7.4.1 Data Generation Procedure for LB-WayPtNav-DH

Generating supervision for training the CNN in dynamic environments in challenging as (a) it requires simulation of visually realistic humans and their motion, and (b) the robot motion affects the future scenes so the dataset needs to be *active* (or on-policy) to enable rich human-robot interactions. To address the above challenges, we introduce the HumANav dataset that allows for photorealistic renderings of indoor environment scenes with humans in them. In this section, we provide more details about HumANav and how we can use this dataset along with the MPC-based expert policy in Sec. 7.3.2.1 to generate the training data for the CNN.

7.4.1.1 The Human Active Navigation Dataset (HumANav)

The HumANav Dataset, shown in Figure 7.11, is an active dataset incorporating human meshes from the SURREAL dataset [299] and indoor office building meshes from the SD3DIS building dataset [23]. The key component of HumANav is a rendering engine that automatically fuses these meshes in order to allow a user to load a human, specified by gender, texture (clothing, skin color, facial features), and body shape, into a desired building, at a specified position, orientation, speed, and angular speed. Additionally, the user can manipulate the human pose and render photo-realistic visuals (RGB, disparity, surface normals, etc.) of the

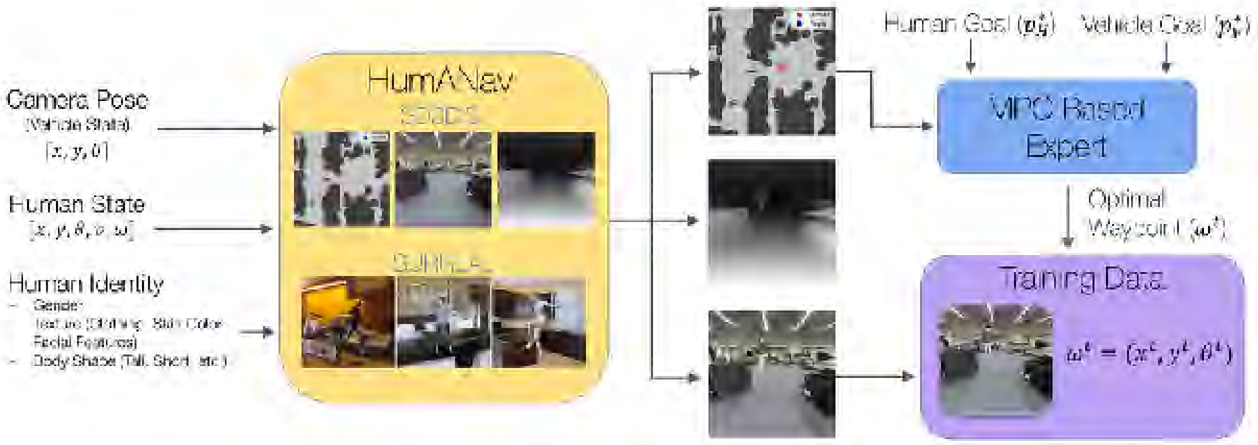


Figure 7.11: We consider the problem of autonomous visual navigation in *a priori* unknown, indoor environments *with* humans. Our approach, LB-WayPtNav-DH, consists of a learning-based perception module and a model-based planning and control module. To learn navigational behavior around humans, we create the HumANav dataset which allows for photorealistic renderings in simulated buildings environments *with* humans. We use an MPC-based expert along with HumANav to train LB-WayPtNav-DH *entirely* in simulation. At test time, LB-WayPtNav-DH navigates efficiently in never-before-seen buildings based only on monocular RGB images *and* demonstrates zero-shot, sim-to-real transfer to novel, real buildings around real humans (right).

human and building from arbitrary viewpoints using a standard perspective projection camera. Crucially, HumANav renders images with visual cues relevant for path planning (i.e. if the human is walking quickly the humans legs will be far apart in the image), ensuring that visual cues for downstream planning are present in imagery. Note that even though we use the SD3DIS dataset in HumANav, our rendering engine is independent of the meshes used and textured meshes from any office buildings can be used.

Once we generate the optimal human and robot trajectories using our MPC-based expert, we use HumANav to render the RGB images along those trajectories. The rendered RGB images along with the optimal waypoints can then be used to train the CNN in our perception module with supervised learning.

7.4.1.2 MPC-Based Expert Policy

We modify the expert policy discussed in Sec. 7.3.2.1 to additionally take into account the future motion of humans. In particular, we use MPC both to generate realistic human trajectories, as well as optimal waypoints for the robot. We now describe the human trajectory and optimal waypoint generation process.

To generate realistic human trajectories, we model the human as a goal-driven agent with state z^H and dynamics given by (7.1). We additionally make the simplifying assumption that the human follows a piecewise constant velocity trajectory. This assumption is often used in human-robot interaction literature to obtain a reasonable approximation of human trajectories [260].

The human and robot are both modeled as cylindrical agents. To generate the training data, we first sample the start positions (p_V^0, p_H^0) and the goal positions (p_V^*, p_H^*) for the robot and the human respectively, as well as a unique identity for the human (body shape, texture, gender). We then use receding horizon MPC to plan paths for both the robot and human for a horizon of H_p. In particular, at time t, the human solves for the optimal trajectory $\mathbf{z}_H^*$ that minimizes the following cost function

$$J^H(\mathbf{z}_H, \mathbf{u}_H) = \sum_{i=t}^{t+H_p} J_i^H(z_i^H, u_i^H) \tag{7.12}$$

$$J_i^H(z_i^H, u_i^H) := \left(\max\{0, \lambda_1^H - d^{obs}(z_i^H)\}\right)^3 + \lambda_2^H \left(d_H^{goal}(z_i^H)\right)^2 + \lambda_3^H \|u_i^H\|^2 \tag{7.13}$$

As before, $d_H^{goal}(z_i^H)$ represents the minimum collision-free distance between z_i^H and the human goal position p_H^* (also known as the FMM distance). d^{obs} represents the signed distance to the nearest static obstacle. The control penalty is added to encourage variety in the human trajectories. The coefficients $\lambda_1^H, \lambda_2^H, \lambda_3^H$ are chosen to weight the different costs with respect to each other.

Given the optimal human trajectory for time horizon $[t, t+H_p]$, $\mathbf{z}_H^*$, the robot optimizes for the waypoint, $\hat{w}_t$, such that the corresponding trajectory to that waypoint minimizes the following cost function:

$$J^V(\mathbf{z}_V, \mathbf{u}_V) = \sum_{i=t}^{t+H_p} J_i^V(x_i, u_i) \tag{7.14}$$

$$\begin{aligned} J_i^V(z_i^V, u_i^V) := \left(\max\{0, \lambda_1^V - d^{obs}(z_i^V)\}\right)^3 + \lambda_2^V \left(d_V^{goal}(z_i^V)\right)^2 + \\ \lambda_3^V \left(\max\{0, \lambda_4^V - d^{human}(z_i^V, z_i^H)\}\right)^3 \end{aligned} \tag{7.15}$$

Similar to the human's cost function, d_V^{goal} represents the collision-free distance to robot's goal, p_V^*. d^{obs} represents the signed distance to the nearest obstacle, and d^{human} represents the signed distance to the human at time i. The coefficients $\lambda_1^V, \lambda_2^V, \lambda_3^V$ are chosen to weight the different costs with respect to each other.

Both the robot and human plan paths in a receding horizon fashion, repeatedly planning (for a horizon of H_p) and executing trajectories (for a horizon of H) until the robot reaches its goal position. We then render the image seen at each of the robot's intermediate states (using HumANav) and save the corresponding pair $[(I_t, p_t^*, u_t^V), \hat{w}_t]$ for training.

Data Sampling Heuristics: We found that training on data with rich interaction between the robot and both static obstacles and humans was crucial to success in test scenarios, especially on our hardware setup; this includes episodes where the robot must navigate around chairs, through doorways, behind a slowly moving human, cut across a human's path, etc. To this end, we designed several heuristics to stimulate such interaction. *First,* we choose the human's initial state, p_H^0, such that it lies approximately along the robot's optimal path to its goal position in the absence of the human. *Second,* we penalize for proximity to the human *only* when the human is visible in the robot's current RGB image, i.e., λ_3^V is nonzero only when the human is visible in the robot's current RGB image. This facilitates downstream learning as it ensures the human is visible when information about the human is used for planning. We will present quantitative results in the importance of these sampling heuristics in Section 7.4.2.

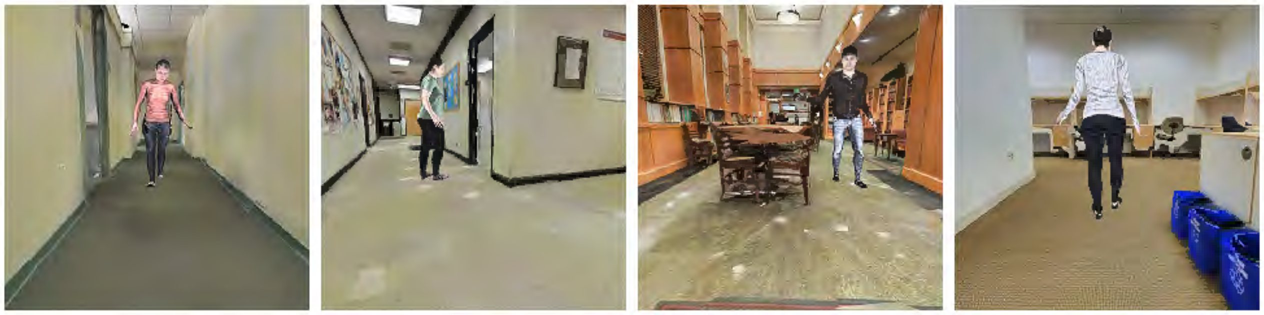
7.4.2 Simulation Results

We now present simulation results to investigate the following two key questions: (1) Can LB-WayPtNav-DH effectively plan goal-driven trajectories in novel environments while reasoning about the dynamic nature of humans? (2) What are the merits of combining model-based control with learning for perceptual understanding of humans, compared to fully learning-based methods and purely geometry-based, learning-free methods?

Simulation Setup: Our simulation experiments are conducted using the HumANav dataset described in Section 7.4.1.1. Scans from 3 buildings and 4800 human identities are used to generate training data. 150 test episodes (start, goal position pairs) in a 4th *held out* building and *held out* human identities (texture, body shape, etc.) are used to evaluate all methods. Some representative RGB images from our training and testing environments are shown in Figure 7.12. Even though both the training and the test environments are indoor office spaces, their layout and appearance differ significantly, but LB-WayPtNav-DH adapts well to the new environments. Train and test scenarios are sampled to stimulate rich interaction between the human and the robot as described in Section 7.4.1.2.

Implementation Details: The implementation details for LB-WayPtNav-DH are same as that of LB-WayPtNav, except that LB-WayPtNav-DH is trained on HumANav using the modified MPC-based expert policy. We refer the interested readers to Sec. 7.3.2.2 for the implementation details.

Comparisons: We compare LB-WayPtNav-DH with four baselines. *LB-WayPtNav*: the CNN is trained on the SD3DIS dataset with no humans (Sec. 7.3). *Mapping-SH (Static Human)*: the known robot's camera parameters are used to project its current depth image to an occupancy grid (treating the human as any other static obstacle), which is then used for model-based planning. *End-to-End (E2E) learning*: CNN trained on the same data as LB-WayPtNav-DH, but instead of a waypoint directly regress to control commands corresponding to the optimal robot trajectory. *Mapping-WC (Worst Case Human)*: same as Mapping-SH, but if the human is visible in the current frame, Mapping-WC plans a


(a) Sample training environments

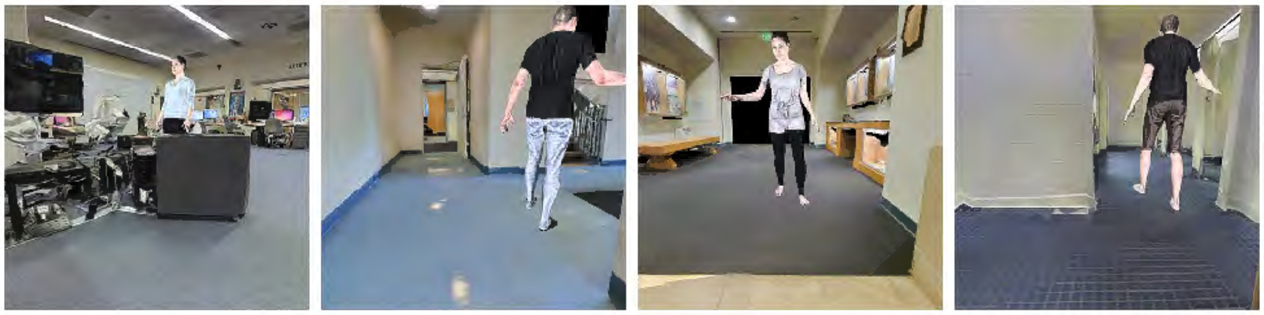

(b) Sample test environments

Figure 7.12: Representative images from training and testing scenarios using the HumANav dataset. The buildings used at training and test time are visually dissimilar and have substantially different layouts. We also keep a held-out set of human identities for our test scenarios. LB-WayPtNav-DH is able to generalize well to novel environments with never-before-seen humans at test time.

path around all possible future human behaviors assuming that the human's current state, $[x_t^H, y_t^H, \phi_t^H]$, is perfectly known and that the human moves at any speed in $[0, \bar{v}^H]$ for the entire planning horizon. We use a control horizon of $H = 0.5s$ for fast replanning around humans.

Metrics: As before, we compare the success rate across all methods, as well as episode specific metrics computed over the subset of goals where all methods succeed, such as the average time to reach the goal, average robot jerk, and acceleration (Acc) along the successful trajectories (lower is better).

7.4.2.1 Results

Comparison with LB-WayPtNav: LB-WayPtNav-DH reaches the goal on average 13% more than LB-WayPtNav (Table 7.4). As expected, LB-WayPtNav tends to fail in scenarios where anticipating future human motion plays a pivotal role in planning a collision-free path. LB-WayPtNav takes a greedy approach in such scenarios, treating the human like any other

Table 7.4: Performance of LB-WayPtNav-DH (ours) and the baselines in simulation. Best results shown in bold.

Agent	**Input**	**Success** (%)	**Time Taken** (s)	**Acc** (m/s^2)	**Jerk** (m/s^3)
Expert	Full map	100			
Learning Based Methods					
LB-WayPtNav-DH	RGB	**80.52**	12.27 ±1.99	.09 ±.02	.612 ±.14
LB-WayPtNav	RGB	67.53	13.97 ±2.68	.10 ±.02	.71 ±.13
E2E	RGB	52.60	14.95 ±5.27	.15 ±.02	3.95 ±1.04
Mapping Methods (memoryless)					
Mapping-SH	Depth	76.63	11.85 ±1.60	.11 ±.03	.76 ±.26
Mapping-WC	Depth + Human State	**94.81**	11.83 ±2.09	.11 ±.03	.72 ±.21

static obstacle, ultimately leading to a collision with the human. In Fig. 7.13 we analyze one such representative test scenario.

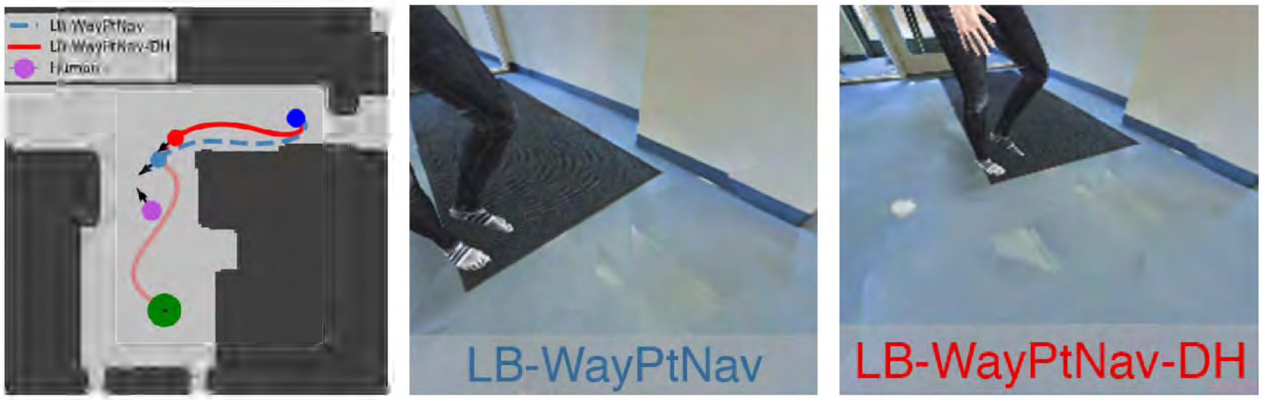


Figure 7.13: (left) The robot starts at the dark blue circle. Its goal it to move to the green goal region without colliding with static obstacles (dark gray) or humans (magenta). LB-WayPtNav follows the light-blue, dashed trajectory until the light blue dot, planning a path to the right of the human (in its direction of motion), leading to collision. LB-WayPtNav-DH follows the red trajectory until the red circle, planning a trajectory (transparent red) to the left the of the human which accounts for the its future motion, and ultimately leads to success. (middle & right) Corresponding RGB images seen by the robot.

Comparison with End-to-End learning: Our findings (Table 7.4) are consistent with results observed for static environments (see Table 7.1) – the use of model-based control in the navigation pipeline significantly improves the success rate of the agent as well as the overall trajectory efficiency and smoothness (see the *Jerk* column in Table 7.4). We note that E2E learning particularly fails in the scenarios where a precise control of the system is required, such as in narrow hallways or openings, since even a small error in control command prediction can lead to a collision in such scenarios.

Comparison with Mapping-SH: Similar to LB-WayPtNav, Mapping-SH fails to account for the dynamic nature of the human. Indeed, in episodes where inferring the dynamic nature of the human is crucial to success, Mapping-SH achieves a 0% success rate. LB-WayPtNav-DH, however, succeeds on 58.33% of these episodes, indicating that it can reason about the dynamic nature of the human.

It is important to note that Mapping-SH has access to the ground-truth depth (and consequently occupancy) and hence can avoid static obstacles perfectly. In contrast, LB-WayPtNav-DH *learns* to avoid collision with both static obstacles and dynamic humans based on a RGB image, and as a result, its failure modes include collision with static obstacles as well. Despite these collisions with static humans (which Mapping-SH does not suffer from), LB-WayPtNav-DH still overall outperforms Mapping-SH slightly. However, since real-world depth sensors are neither perfect nor have an unlimited range, we see a noticeable drop in the performance of Mapping-SH in real-world experiments as discussed in Sec. 7.4.3. On the other hand, LB-WayPtNav-DH is trained to be robust to sensor noise and exhibits similar error profiles on real and synthetic imagery (see Sec. 7.4.3).

On the goals where both methods succeed, Mapping-SH is approximately 9% faster than LB-WayPtNav-DH, since it has access to perfect scene geometry and can plan a path which barely avoids the human. LB-WayPtNav-DH, on the other hand, is trained to take conservative trajectories which avoid the human's potential future behavior.

Comparison with Mapping-WC: Mapping-WC unsurprisingly achieves near perfect (95%) success as it assumes perfect depth and human state estimation. Mapping-WC fails (5%) due to the receding horizon nature of its MPC planner, which might lead the robot to a future state from which it cannot avoid collision.

Interestingly, we found that in many cases, Mapping-WC reaches the goal faster than LB-WayPtNav-DH (Table 7.4) by exploiting precise geometry of the scene and human, taking an aggressive trajectory which barely avoids collision with the human (see Fig. 7.14 for an example). However, as expected, in other cases Mapping-WC takes overly conservative paths, planning a path that avoids *all* possible human trajectories regardless of their likelihood. In contrast, LB-WayPtNav-DH is trained to reason about the human's likely trajectory and thus plans more efficient paths and reaches the goal on average 6% faster than Mapping-WC.

Mapping-WC performance is also affected by noise in human state estimation. To quantify this, we add zero-centered, uniformly random noise to $[x_t^H, y_t^H, \phi_t^H]$ in Mapping-WC. As a result, the success rate of Mapping-WC drops by 7%, indicating the challenges associated with this approach, especially when the human state needs to be inferred from a monocular RGB image.

Learned Navigational Cues and Importance of Photorealism: We designed HumANav such that relevant visual cues for navigation are rendered in imagery; i.e. a human's legs will be spread apart if they are moving quickly and will stay closed if they are not moving. LB-WayPtNav-DH is able to incorporate these visual cues to anticipate future human

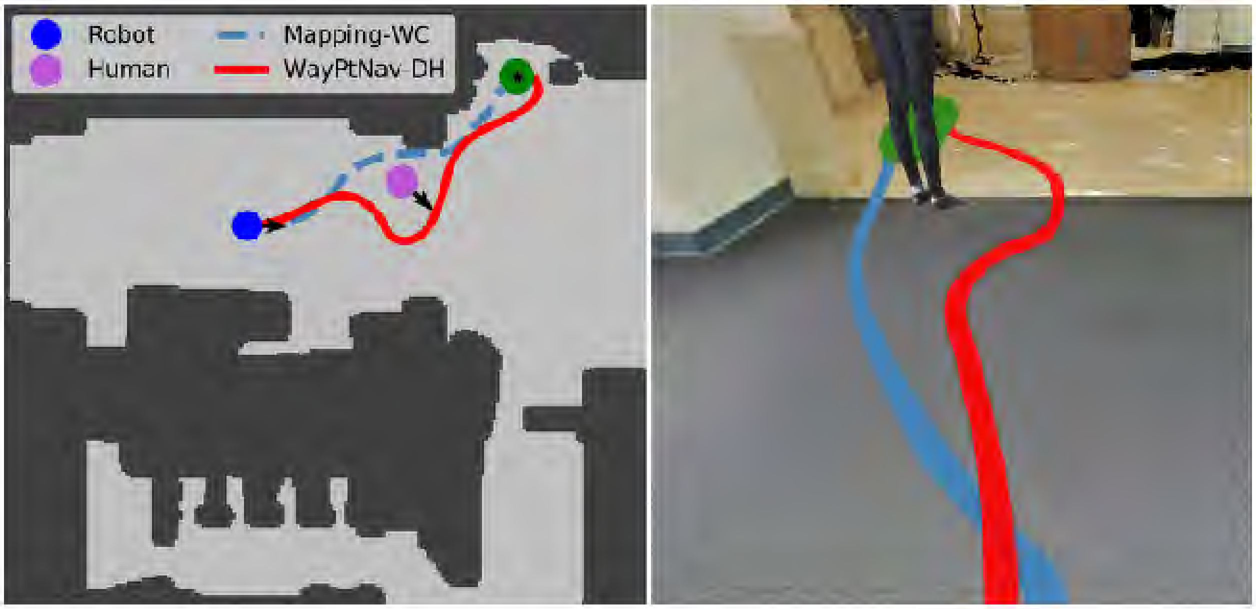


Figure 7.14: Topview of the trajectories taken by Mapping-WC and LB-WayPtNav-DH from the same state and the corresponding RGB image with the trajectories superimposed. Mapping-WC reaches the goal faster than LB-WayPtNav-DH as it has access to precise geometry of the scene and the human state and thus plans a path between the human and the wall which narrowly avoids collision. LB-WayPtNav-DH, on the other hand, takes a more cautious path as it does not have access to the human state.

motion and accordingly plan the robot's trajectory (Fig. 7.15).

To understand the importance of photorealistic images, we also trained LB-WayPtNav-DH on images of humans that are colored in gray (see Fig. 7.16). Consequently, we see a drop of 6% in the success rate, indicating that training LB-WayPtNav-DH with photorealistic textures (clothing, skin color, hair color, etc.) generalizes better to novel humans.

Effect of Sampling Heuristics: To understand the importance of our data sampling procedure, we train an additional baseline *LB-WayPtNav-DH-FOV*. In this baseline, the CNN is trained to predict waypoints which always avoid the human, regardless of whether the human is visible in the robot's current image or not. To generate optimal waypoints for training the CNN, the robot cost function always penalizes the proximity with a human even when the human in not within the field-of-view (FOV) at the current time. The results are presented in Table 7.5. LB-WayPtNav-DH reaches the goal on average 12% more than LB-WayPtNav-DH-FOV and on average 5% faster than LB-WayPtNav-DH-FOV. This indicates that restricting our expert to choose waypoints only considering information within its current field of view, as described in 7.4.1.2, facilitates downstream learning and ultimately the performance for LB-WayPtNav-DH. Intuitively, since the perception module is reactive, it has limited capabilities to reason about the human motion when the human is

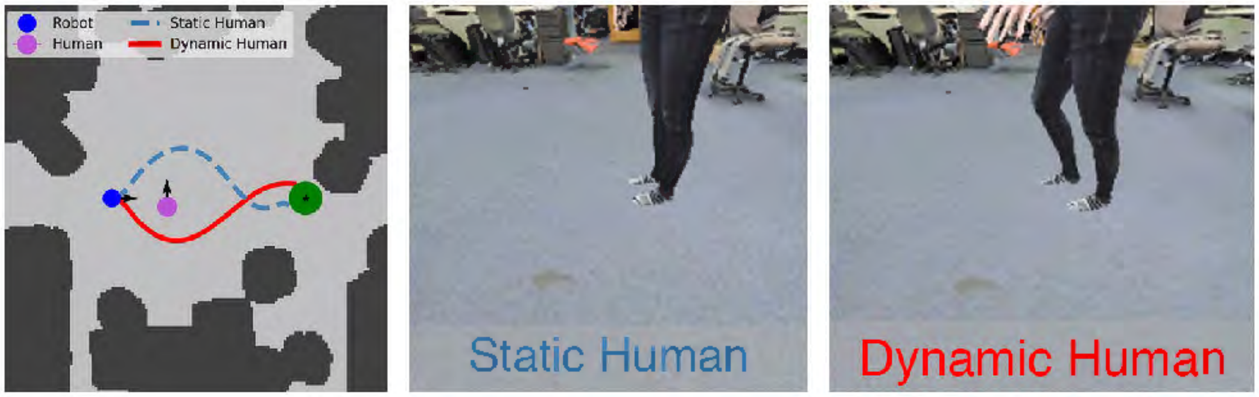


Figure 7.15: Topview of the trajectories taken by LB-WayPtNav-DH from the same state with a static human (light blue, dashed line) and a dynamic human (red, solid line), and the corresponding RGB images. HumANav enables LB-WayPtNav-DH to leverage cues, such as spread of humans legs and direction of human toes, to infer that the left RGB image likely corresponds to a static human and the right one to a moving human.

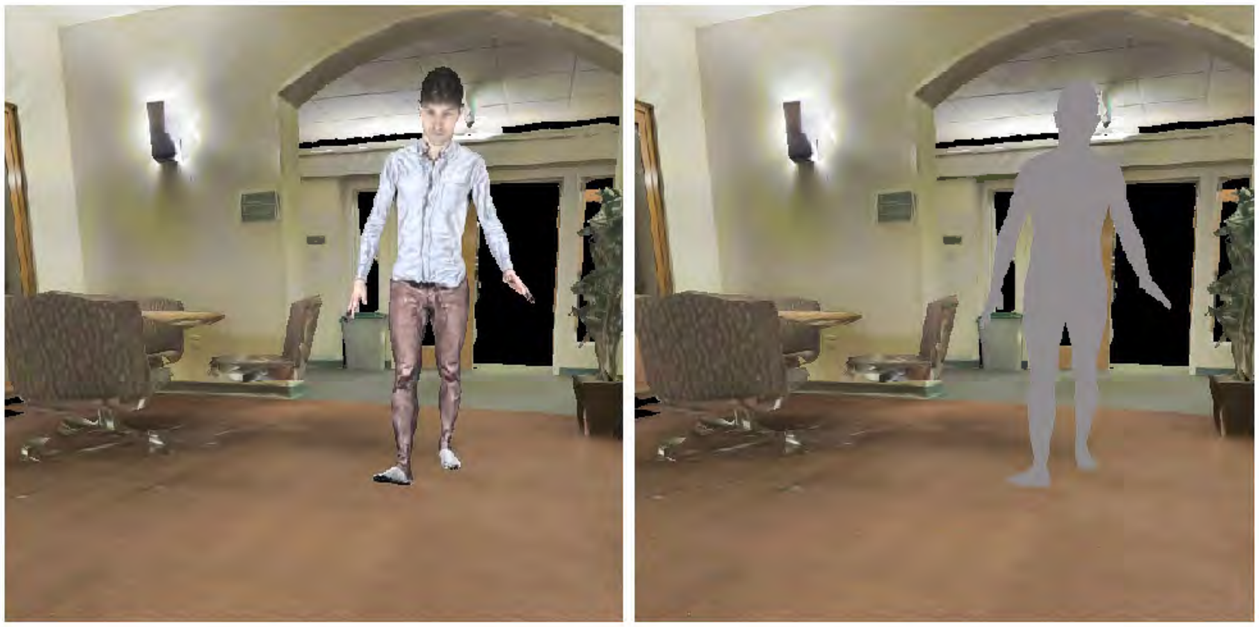


Figure 7.16: LB-WayPtNav-DH trained on images from HumANav with realistic textures (clothing, hair, skin color, facial features) (left) leads to a better generalization than training on human figures with gray textures (right).

not in robot's FOV. Thus, reasoning about the human motion when the human is not within the FOV can overconstrain the learning problem. In future, we will explore adding memory to the CNN (such as using LSTM or RCNN) that can overcome some of these challenges.

Table 7.5: Comparison between LB-WayPtNav-DH (ours) and LB-WayPtNav-DH-FOV methods on 150 test episodes. Average time taken, jerk, and acceleration numbers are reported on the scenarios where both methods succeed.

Agent	**Input**	**Success (%)**	**Time Taken (s)**	**Acc (m/s^2)**	**Jerk (m/s^3)**
LB-WayPtNav-DH	RGB	80.52	12.94 $\pm$3.06	.09 $\pm$.02	.64 $\pm$.13
LB-WayPtNav-DH-FOV	RGB	68.18	13.57 $\pm$3.52	.09 $\pm$.02	.66 $\pm$.13

Navigation Around Multiple Humans: LB-WayPtNav-DH is trained on environments with single human; however, we find that it can generalize to settings with multiple humans (Fig. 7.17). LB-WayPtNav-DH is able to successfully navigate around multiple humans walking side by side or separately in a narrow hallway. We hypothesize that LB-WayPtNav-DH succeeds in these scenarios as it reduces the multi-human avoidance problem to a single human avoidance problem (i.e. by treating both humans as a single large "meta-human" in the first scenario and by solving two smaller, single-human avoidance problems in the second scenario). The third scenario, on the other hand, is specifically designed to test whether LB-WayPtNav-DH can reason about multiple, distinct future human trajectories at once. LB-WayPtNav-DH, struggles to accurately infer both humans' future motion, and thus collides. In fact, the same scenario, when run without the second human, leads to LB-WayPtNav-DH successfully reaching the goal.

Failure Modes: LB-WayPtNav-DH successfully navigates around dynamic and static obstacles in novel environments, however it is primarily limited in its ability to recognize and predict the *long-term* motion of humans. These issues are tightly coupled with the robot's reactive nature (uses *only* the current RGB image) and limited field of view (forward facing camera) as humans may approach the robot from outside or on the fringe of its field of view.

7.4.3 Hardware Experiments

We directly deploy the LB-WayPtNav-DH framework, trained in simulation, onto a Turtlebot 2 hardware platform without any finetuning or additional training. Our algorithm is tested in two never-before-seen buildings. Representative images of our experiment environments are shown in Figure 7.18. Importantly, we note that our robot has only been trained on synthetic humans from the SURREAL dataset [299], constrained to piecewise constant velocity trajectories. Humans in our experiments, however, do not have such dynamical constraints. For robot state measurement, we use the Turtlebot's encoder based odometry.

Our experiment setups are shown in Fig. 7.19. The static clutter in these environments is limited as the experiments are designed to evaluate whether the robot has learned to reason about the dynamic nature of humans. In experiment 1, the human walks parallel to the robot but in the opposite direction; however, the human suddenly takes a turn towards the robot, requiring it to anticipate the human behavior to avoid a collision. In experiment 2, the

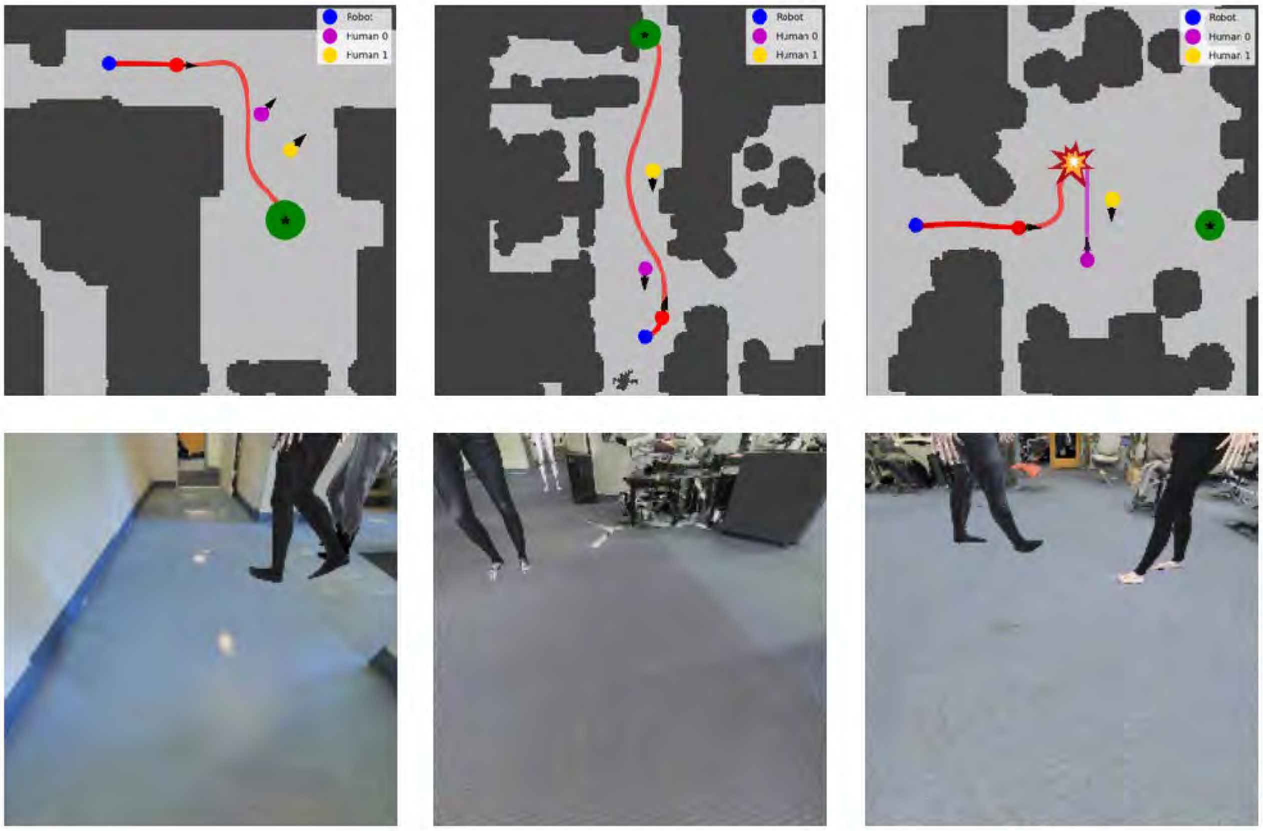


Figure 7.17: Navigation around multiple humans. LB-WayPtNav-DH successfully turns a corner while avoiding two humans walking side by side (left), navigates a long hallway with multiple humans walking down the hallway (middle). LB-WayPtNav-DH attempts to traverse a room, crossing the path of two different humans that are moving in opposing directions (right). LB-WayPtNav-DH is unable to reason about the future trajectory of both humans simultaneously which ultimately leads to a collision.

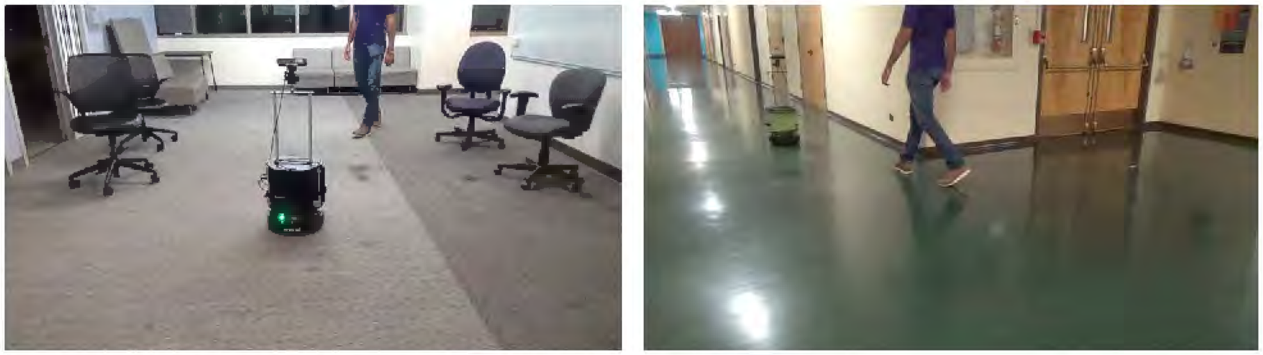


Figure 7.18: Some representative images of the experiment scenarios. Neither of these buildings were used for training/testing purposes in simulation.

robot and the human move in opposite directions, but cross each other near a tight corner, requiring the robot to take a cautious trajectory around the human to avoid a collision.

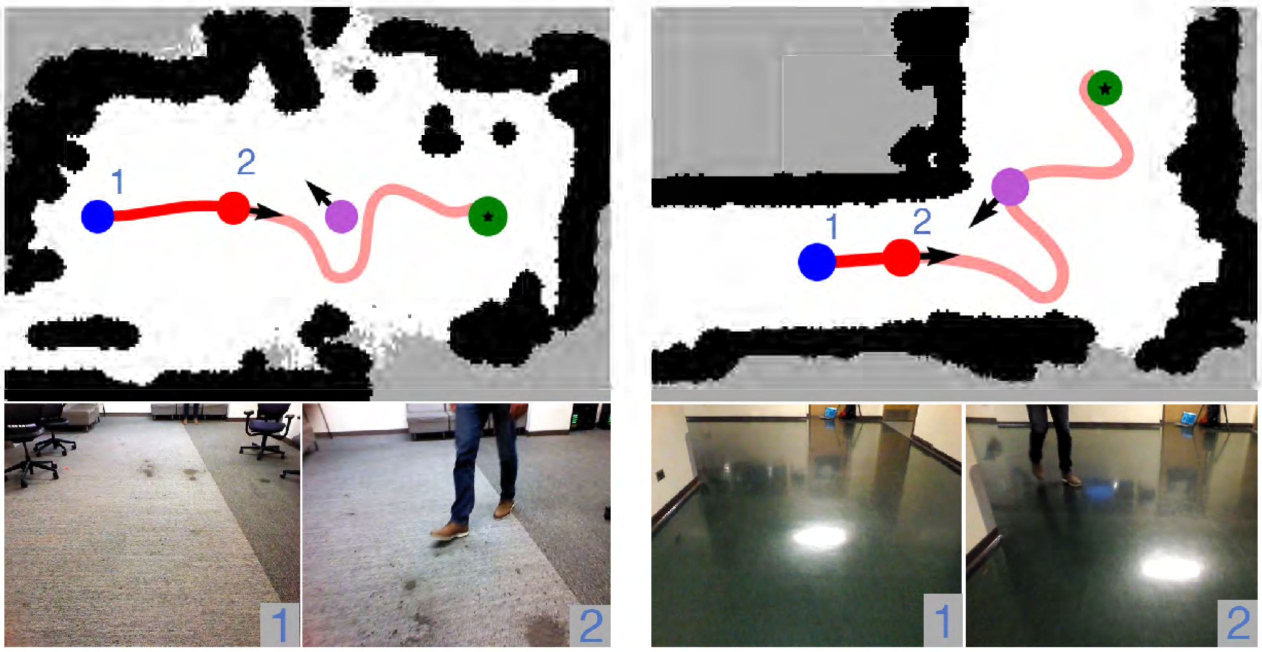


Figure 7.19: Two examples from the experiments. Top: executed trajectory. Purple dot is the human with an arrow depicting its direction. Bottom: RGB images from the robot. The robot selects the trajectories in the opposite direction from the human to avoid a collision, even if is means diverging from the optimal path to the goal.

We compare the performance of LB-WayPtNav-DH, LB-WayPtNav, and Mapping-SH on our hardware platform across two experimental settings for five trials each (Table 7.6). We do not compare to End-To-End or Mapping-WC on our hardware setup as the simulation performance of End-To-End is already very low and Mapping-WC requires access to the ground truth state information of the human, which was not reliable using our narrow field-of-view monocular RGB camera. Our experiment videos can be found on the project website[3].

Comparison With LB-WayPtNav: LB-WayPtNav succeeds in only 3 trials out of 10 (Table 7.6). In both experiments LB-WayPtNav attempts to avoid the human, treating it as a static obstacle, however the human advances towards the robot before it can correct course. This is unsurprising as this method is trained purely on static obstacles and these experiments are specifically designed to test the agent's understanding of the dynamic nature of humans. In cases where treating the human as a static obstacle does succeed however, the robot is approximately 20% more efficient than LB-WayPtNav-DH as it does not take

[3] Project Website: https://smlbansal.github.io/LB-WayPtNav-DH/

longer, cautious paths which allow for more human avoidant behavior.

Table 7.6: Experimental results, averaged over 10 trials (5 trials per experiment). LB-WayPtNav and Mapping-SH lack the understanding of the dynamic nature of the human, ultimately leading to a collision with the human.

Agent	**Success (%)**	**Time taken (s)**	**Acceleration (m/s^2)**	**Jerk (m/s^3)**
LB-WayPtNav-DH	**100.00**	19.15 ±1.25	0.06 ±0.01	2.62 ±0.26
LB-WayPtNav	30.00	15.56 ±0.08	0.06 ±0.01	**2.47** ±0.11
Mapping-SH	20.00	**11.38** ±0.23	0.22 ±0.05	10.45 ±0.10

Comparison With Mapping-SH: To implement Mapping-SH on the Turtlebot, we project the robot's current depth image onto an occupancy grid on the ground plane using the known camera intrinsic and extrinsic parameters. Similar to LB-WayPtNav, the performance of Mapping-SH deteriorates even further in our hardware experiment, succeeding in only 2 trials. Performance of Mapping-SH is further impacted by its exact reliance on the geometry of the scene, which can lead to failure when the depth sensor gives erroneous readings.

In cases where Mapping-SH does succeed, it reaches the goal approximately 40% faster than LB-WayPtNav-DH. This is expected as Mapping-SH is designed to exploit the exact geometry of the scene, barely avoiding obstacles on its way to the goal. Given the reactive nature of Mapping-SH and lack of understanding of the dynamic nature of the human, when Mapping-SH does succeed it does so by executing a last-resort, aggressive turn or stop to avoid imminent collision with the human. This behavior is reflected in the exceptionally high jerk in trajectories (10.45 m/s^3).

Performance of LB-WayPtNav-DH: LB-WayPtNav-DH succeeds in all 10 trials by exhibiting behavior which takes into account the dynamic nature of the human agent. These results demonstrate the capabilities of a learning algorithm trained *entirely* in simulation on the HumANav dataset to generalize to navigational problems in real buildings with real people.

In experiment 1, LB-WayPtNav-DH navigates around the human by moving contrary to its direction of motion, which allows it to reliably avoid collision. LB-WayPtNav and Mapping-SH, however, treat the human as a static obstacle and attempt to avoid it by moving in its direction of motion. Similarly, in experiment 2, while navigating through hallways, LB-WayPtNav-DH takes a larger radius turn around a corner to leave space for the human moving in the other direction. LB-WayPtNav and Mapping-SH exhibit greedy behavior, trying to navigate around the tight corner in hope for a shorter path to the goal, but ultimately failing more often as they cannot react quickly enough to maneuver around the human.

Similar to our simulation results, when all three methods succeed (2 out of 10 trials), LB-WayPtNav-DH is noticeably less efficient than Mapping-SH and LB-WayPtNav because the trajectories it takes around the human are more cautious, and thus less-efficient as it has been trained to avoid the human's current position *and* short-term future trajectory.

7.5 Chapter Summary

Visual navigation in *a priori* unknown environments is an important problem in robotics. In this chapter, we proposed LB-WayPtNav, an autonomous visual navigation framework that combines learning-based perception with model-based control for goal-driven navigation in novel indoor environments. LB-WayPtNav is better and more reliable at reaching unseen goals compared to an End-to-End learning or a geometric mapping-based approach. Use of a model-based feedback controller allows LB-WayPtNav to successfully generalize from simulation to physical robots.

We next extend the capabilities of LB-WayPtNav to dynamic environments. The new framework, LB-WayPtNav-DH, can autonomously navigate in *a priori* unknown indoor environments with humans. To train the perception module in LB-WayPtNav-DH, we also create a photorealistic dataset, HumANav, that can render rich indoor environment scenes with humans. The dataset consists of synthetic humans and can be fully autonomously generated, avoiding privacy and logistic difficulties present when working with real human subjects. We demonstrate that LB-WayPtNav-DH trained on HumANav can successfully learn to navigate around humans and transfer the learned policies from simulation to reality.

Part III

Safety for Learning-Enabled Control Systems

In Part 2 of this thesis, we discussed how we can use machine learning to control autonomous systems when their dynamics model or the environment is unknown. In this part, we will focus on how we can provide safety guarantees for the system when learning is involved in the control loop. In Chapter 8, we will focus on how to provide safety guarantees using the learned dynamics models. In Chapter 9, we will focus on providing safety guarantees when the system is operating in an unknown environment and using learning-based perception for control.

Chapter 8

Safety Analysis Using Learning-Based Dynamics Models

This chapter is based on the paper "A New Simulation Metric to Determine Safe Environments and Controllers for Systems with Unknown Dynamics" [131] written in collaboration with Shromona Ghosh, Sanjit Seshia, Alberto Sangiovanni-Vincentelli, and Claire Tomlin.

In Chapter 5 and 6, we discussed how we can use indirect and direct learning-based methods to capture the inaccuracies in the dynamics model of an autonomous system, and leverage learning to improve the control performance despite these inaccuracies. Since many of these autonomous systems are safety-critical, it is important to design provably-safe controllers while determining environments in which safety can be guaranteed. In Part-1 of this dissertation we discussed how dynamics models can be used for the safety analysis of a system. However, unlike traditional dynamical systems, one of the many verification challenges for ML-based systems [273] is that learned abstractions (i.e. a model, used interchangeably here on) cannot be directly used for verification, since it is not clear *a priori* how representative the abstraction is of the actual system. Hence, to use the abstraction to provide guarantees for the system, we need to first quantify the differences between it and the system.

In this chapter, we focus on providing safety guarantees for the *actual* system based on a learned dynamics model for reach-avoid objectives. Recall that in a reach-avoid problem, the goal is to design a controller to reach a target set of states (referred to as reach set) while avoiding unsafe states (avoid set) at all times. Reach-avoid problems are common for autonomous vehicles in the real world; for example, for the Crazyflie system discussed in Chapter 5, the reach set could be a desired goal position and the avoid set could be the set of the obstacles. In such a setting, it is important to determine the environments in which the drone can safely navigate, as well as the corresponding safe controllers.

The key idea behind our approach is to compute an estimate of the "distance" between the actual system and its learned abstraction. One such distance could be the maximal Euclidean distance between the system and the abstraction output trajectories over all finite horizon control sequences. The estimated distance is then used to *expand* the unsafe set (or

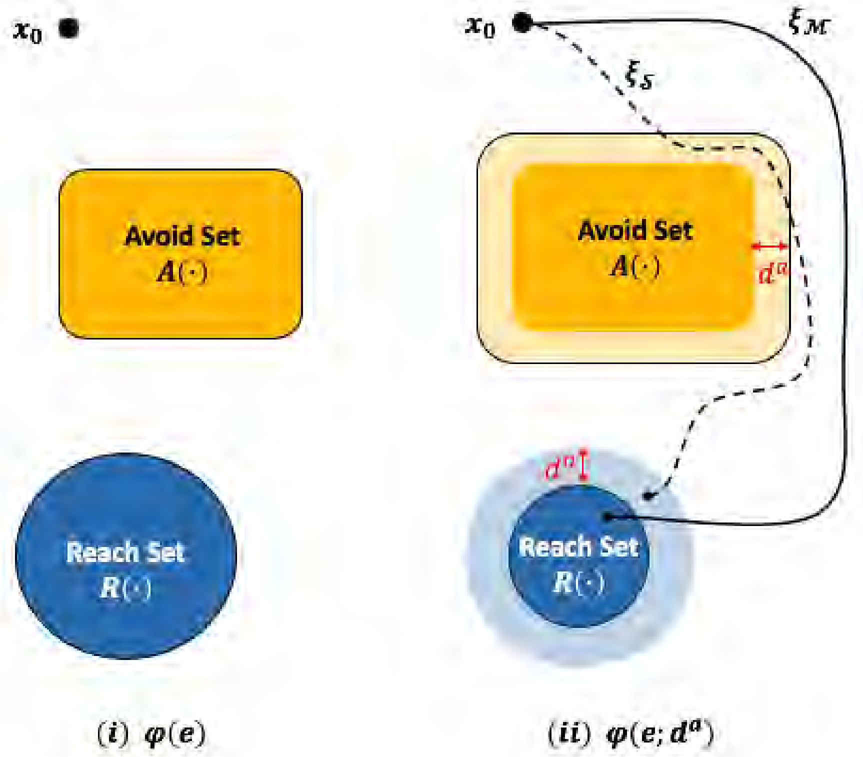


Figure 8.1: The avoid set is expanded and the reach set is contracted with the simulation metric d^a. If the abstraction trajectory ($\xi_{\mathcal{M}}$) stays clear of the expanded avoid set and reaches the contracted reach set, the system trajectory ($\xi_{\mathcal{S}}$) also stays clear of the original avoid set and reaches the original reach set.

avoid set) and *contract* the reach set, as shown in Figure 8.1. If we can synthesize a safe controller that ensures the abstraction trajectory avoids the expanded avoid set and reaches the contracted reach set, then the system trajectory is guaranteed to avoid and reach the original avoid set and reach set respectively. Consequently, the set of safe environments for the system can be obtained by finding the set of environments for which we can design a safe controller for the abstraction with the modified specification, which in turn can be computed using model-based safety analysis methods such as Hamilton-Jacobi reachability.

8.1 Related Work

Model Validation. The problem of computing mismatch between a system and its abstraction is studied extensively in the literature under the banner of model validation. The key idea of these approaches is to use model identification techniques that explicitly provide bounds on the mismatch either in time or frequency domain (see [130, 153, 200] and references therein). This bound is then used to design a provably stabilizing controller for the system. These approaches, however, have largely been limited to linear abstractions and systems, and the focus has been on designing asymptotically stabilizing controllers, as approach to reach-avoid specifications.

Simulation Metric. Another way to quantify the difference between a general non-linear system and its abstraction relies on the notion of a (approximate) *simulation metric* [19, 136, 28]. Such a metric measures the maximal distance between the system and the abstraction output trajectories over all finite horizon control sequences. Standard simulation metrics (referred to as SSM here on) have been used for a variety of purposes such as safety verification [137], abstraction design for discrete [189], nonlinear [249], switched [138] systems, piecewise deterministic and labelled Markov processes [96, 282], and stochastic hybrid systems [5, 127, 163, 63], model checking [28, 167], and model reduction [91, 244]. As shown in Figure 8.1, the SSM is used to expand the unsafe set (or avoid set) and contract the reach set. A safe controller for the abstraction for the modified reach-avoid problem is guaranteed to be safe for the actual system. This follows from the property that SSM captures the worst case distance between the trajectories of the system and the abstraction.

Even though powerful in its approach, SSM computes the maximal distance between the system and the abstraction trajectories across all possible controllers. This is unnecessary and might lead to a conservative bound on the quality of the abstraction for the purposes of controller synthesis. In particular, the larger the distance between the system and the abstraction, the larger the expansion (contraction) of the avoid (reach) set. In many cases, this results in unrealizability wherein there does not exist a safe controller for the abstraction for the modified specification. In this chapter, we propose a new distance metric to overcome these challenges.

Scenario Optimization. Another challenge with SSM is its computation when the dynamics of the system are not available. Several approaches have been proposed in the literature for computing SSM [4, 136, 163]; however, restrictive assumptions on the dynamics of the systems are often required to compute it. More recently, a randomized approach has been proposed to compute SSM [5, 127] for finite-horizon properties that relies on "scenario optimization", which was first introduced for solving robust convex programs via randomization [66] and then extended to semi-infinite chance-constrained optimization problems [68]. Scenario optimization is a sampling-based method to solve semi-infinite optimization problems, and has been used for system and control design [65, 69]. In this work, we will leverage scenario optimization to compute the distance metric between the system and its abstraction.

Contributions and Chapter Organization

In this paper, we propose SPEC, SPEcification-Centric simulation metric, a novel distance metric between the system and its abstraction. SPEC is also motivated by the key insight behind SSM, in that, if the reach-avoid specification is changed using SPEC in a similar fashion as that for SSM, it is guaranteed that a safe controller for the abstract model remains safe for the system. However, SPEC does not inherit the conservative nature of SSM, and can be used to design safe controllers for the system for a broader range of reach-avoid specifications. In fact, we show that, among all uniform distance bounds (i.e., a single distance bound is used to modify the specification in all environments), SPEC provides the largest set of environments such that a safe controller for the abstraction is also safe for the system. SPEC achieves this by computing the distance across

1. only those controllers that can be synthesized by a particular control scheme and that are safe for the abstraction (in the context of the original reach-avoid specification) — these are the only potential safe controllers for the system;

2. only those abstraction and system trajectories for which the system violates the reach-avoid specification, and

3. only between the abstraction trajectory and the reach and the avoid sets.

To compute SPEC, we propose a scenario optimization-based approach that has general applicability and is not restricted to a specific class of systems. Indeed, the only assumption is that the system is available as an oracle, with known state and control spaces, which we can simulate to determine the corresponding output trajectory. Given that the distance metric is obtained via randomization and, hence, is a random quantity, we provide probabilistic guarantees on the performance of SPEC. However, this confidence is a design parameter and can be chosen as close to 1 as desired (within a simulation budget).

8.2 Problem Formulation

Let $\mathcal{S}$ be an unknown, discrete-time, potentially non-linear, dynamical system with state space $\mathbb{R}^{n_x}$ and control space $\mathbb{R}^{n_u}$. Let $\mathcal{M}$ be a (learned) abstraction of $\mathcal{S}$ with the same state and control spaces as $\mathcal{S}$, i.e., a dynamics model of $\mathcal{S}$. We also assume that the bounds between the dynamics of $\mathcal{M}$ and $\mathcal{S}$ are not available beforehand (i.e., we cannot *a priori* quantify how different the two are). As before, $\xi_{\mathcal{S}}(t; x_0, \mathbf{u})$ denotes the trajectory of $\mathcal{S}$ at time t starting from the initial state x_0 at time 0 and applying the controller $\mathbf{u}$. $\xi_{\mathcal{M}}$ is similarly defined. For ease of notation, we drop $\mathbf{u}$ and x_0 from the trajectory arguments wherever convenient.

We define by $\mathcal{E} := \mathcal{X}_0 \times \mathcal{A} \times \mathcal{R}$ the set of all reach-avoid scenarios (also referred to as *environment scenarios* here on), for which we want to synthesize a controller for $\mathcal{S}$. A reach-avoid scenario $e \in \mathcal{E}$ is a three-tuple, $(x_0, A(\cdot), R(\cdot))$, where $x_0 \in \mathcal{X}_0 \subset \mathbb{R}^{n_x}$ is the initial state of $\mathcal{S}$. $A(\cdot) \in \mathcal{A}, A(\cdot) \subset \mathbb{R}^{n_x}$ and $R(\cdot) \in \mathcal{R}, R(\cdot) \subset \mathbb{R}^{n_x}$ are (potentially time varying) sequences of avoid and reach sets respectively. We leave $\mathcal{A}$ and $\mathcal{R}$ abstract except where necessary. If the sets are not time varying, we can replace $R(\cdot)$ (respectively $A(\cdot)$) by the stationary R (respectively A). Similarly, if there is no avoid or reach set at a particular time, we can represent $A(t) = \emptyset$ and $R(t) = \mathbb{R}^{n_x}$.

For each $e \in \mathcal{E}$, we define a reach-avoid specification, $\varphi(e)$

$$\varphi(e) := \{\xi(\cdot) : \ \forall t \in \mathcal{T} \ \xi(t) \notin A(t) \wedge \xi(t) \in R(t)\}, \tag{8.1}$$

where $\mathcal{T}$ denote the time-horizon $\{0, 1, \ldots, T\}$. We say $\xi(\cdot)$ satisfies the specification $\varphi(e)$, denoted $\xi(\cdot) \models \varphi(e)$, if $\xi(\cdot) \in \varphi(e)$. For mathematical brevity, we define the reach-avoid specification such that the output trajectory must remain within the reach set at all times. In cases where we are interested in *eventually* reaching a desired set of states, $R(t)$ can represent the backwards reachable tube corresponding to the desired set of states.

Finally, we define $\mathbb{U}_\Pi(e) \subseteq \mathbb{U}_0^T$ to be the space of all permissible controllers for e, and $\mathbb{U}_0^T$ to be the space of all finite horizon control sequences over $\mathcal{T}$. For example, if we restrict ourselves to linear feedback controllers, $\mathbb{U}_\Pi$ represents the set of all linear feedback controllers that are defined over the time horizon $\mathcal{T}$. We are now ready to formally state our problem.

Given the set of reach-avoid scenarios $\mathcal{E}$, the controller scheme $\mathbb{U}_\Pi$, and the abstraction $\mathcal{M}$, our goal is two-fold:

1. to find the environment scenarios for which it is possible to design a controller such that $\xi_{\mathcal{S}}(\cdot)$ satisfies the corresponding reach-avoid specification $\varphi(e)$,

2. to find a corresponding safe controller for each scenario in (1).

Mathematically, we are interested in computing the set $\mathcal{E}_{\mathcal{S}}$

$$\mathcal{E}_{\mathcal{S}} = \{e \in \mathcal{E} \ : \exists \mathbf{u} \in \mathbb{U}_\Pi(e) \ , \xi_{\mathcal{S}}(\cdot; x_0, \mathbf{u}) \models \varphi(e)\}, \tag{8.2}$$

and the corresponding set of safe controllers $\mathbb{U}_{\mathcal{S}}(e)$ for each $e \in \mathcal{E}_{\mathcal{S}}$

$$\mathbb{U}_{\mathcal{S}}(e) = \{\mathbf{u} \in \mathbb{U}_\Pi(e) \ : \xi_{\mathcal{S}}(\cdot; x_0, \mathbf{u}) \models \varphi(e)\}. \tag{8.3}$$

When an accurate dynamics model of $\mathcal{S}$ is known, several methods have been studied in literature to compute the sets $\mathcal{E}_{\mathcal{S}}$ and $\mathbb{U}_{\mathcal{S}}(e)$ for reach-avoid problems [293, 225, 292]. We discussed several such methods in Part-1 of this dissertation as well. In this work as well, we have a model of $\mathcal{S}$ (i.e., $\mathcal{M}$), except that it may not be accurate. For the analysis to follow, we make the following assumptions on $\mathcal{S}$ and $\mathcal{M}$:

Assumption 5 *$\mathcal{S}$ is available as an oracle that can be simulated, i.e., we can run an execution (or experiment) on $\mathcal{S}$ and obtain the corresponding system trajectory $\xi_{\mathcal{S}}(\cdot)$.*

Assumption 6 *For any $e \in \mathcal{E}$, we can determine if there exist a controller such that $\xi_{\mathcal{M}} \models \varphi(e)$ and can compute such a controller.*

Assumption 1 states that even though we do not know the dynamics of $\mathcal{S}$, we can run an execution of $\mathcal{S}$. Assumption 2 states that it is possible to verify whether $\mathcal{M}$ satisfies a given specification $\varphi(e)$ or not. Although it is not a straightforward problem, since the dynamics of $\mathcal{M}$ are known, several existing methods can be used for obtaining a safe controller for $\mathcal{M}$.

Under these assumptions, we show that we can convert a verification problem on $\mathcal{S}$ to a verification problem on $\mathcal{M}$. In particular, we compute a distance bound, SPEC, between $\mathcal{S}$ and $\mathcal{M}$ which along with $\mathcal{M}$ allows us to compute a conservative approximation of $\mathcal{E}_{\mathcal{S}}$ and $\mathbb{U}_{\mathcal{S}}(e)$.

Example 3 *We now introduce a very simple example that we will use to illustrate our approach, a 2 state linear system in which the system and the abstraction differ only in one parameter. Although simple, this example illustrates several facets of SPEC. Consider a system $\mathcal{S}$ whose dynamics are given as*

$$x(t+1) = \begin{bmatrix} x_1(t+1) \\ x_2(t+1) \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & 0.1 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} u(t). \tag{8.4}$$

We are interested in designing a controller for $\mathcal{S}$ to regulate it from the initial state $x(0) := x_0 = [0, 0]$ to a desired state $x^* = [x_1^*, 0]$ over a time-horizon of 20 steps, i.e, $T = 20$. In particular, we have

$$\mathcal{X}_0 = \{[0,0]\}, \quad \mathcal{A} = \emptyset, \quad \mathcal{R} = \bigcup_{-4 \le x_1^* \le 4} R(\cdot; x^*),$$

where

$$\begin{aligned} R(t; x^*) =& \mathbb{R}^2, t \in \{0, 1, \dots, T-1\}, \\ R(T; x^*) =& \{x : \|x - x^*\|_2 < \gamma\}. \end{aligned}$$

We use $\gamma = 0.5$ in our simulations. Thus, each $e \in \mathcal{E}$ consists of a final state x^* (equivalently, a reach set $R(T; x^*)$) to which we want the system to regulate, starting from the origin. Consequently, the system trajectory satisfies the reach-avoid specification in this case if $\xi_{\mathcal{S}}(T; x_0, \mathbf{u}) \in R(T; x^*)$.
For the purpose of this example, we assume that the system dynamics in (8.4) are unknown; only the dynamics of its abstraction $\mathcal{M}$ are known and given as

$$x(t+1) = \begin{bmatrix} x_1(t+1) \\ x_2(t+1) \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & 0.1 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + \begin{bmatrix} 1 \\ 0.1 \end{bmatrix} u(t). \tag{8.5}$$

In this example, we use the class of linear feedback controllers as $\mathbb{U}_\Pi(e)$, although other control schemes can very well be used. In particular, for any given environmental scenario e, the space of controllers $\mathbb{U}_\Pi(e)$ is given by

$$\mathbb{U}_\Pi(e) = \{LQR(q, x^*) : 0.1 \le q \le 100\},$$

where $LQR(q, x^*)$ is a Linear Quadratic Regulator (LQR) designed for the abstraction dynamics in (8.5) to regulate the abstraction trajectory to x^* [a], with the state penalty matrix $Q = qI$ and the control penalty coefficient $R = 1$. Here, $I \in \mathbb{R}^{2\times 2}$ is an identity matrix. Thus, for different values of q we get different controllers, which affect the various characteristics of the resultant trajectory, such as overshoot, undershoot, and final state. Our goal thus is to use the dynamics in (8.5) to find the set of final states to which $\mathcal{S}$ can be regulated and the corresponding regulator in $\mathbb{U}_\Pi(e)$.

[a] That is, we penalize the trajectory deviation to the desired state x^* in the LQR cost function.

8.3 SPEC: Specification Centric Simulation Metric

8.3.1 Computing Approximate Safe Sets Using $\mathcal{M}$ and Simulation Metric

Computing sets $\mathcal{E}_{\mathcal{S}}$ and $\mathbb{U}_{\mathcal{S}}$ exactly can be challenging since the dynamics of $\mathcal{S}$ are unknown *a priori*. Generally, we use the abstraction $\mathcal{M}$ as a replacement for $\mathcal{S}$ to synthesize and analyze safe controllers for $\mathcal{S}$. However, to provide guarantees on $\mathcal{S}$ using $\mathcal{M}$, we would need to quantify how different the two are.

We quantify this difference through a distance bound, d, between $\mathcal{S}$ and $\mathcal{M}$. d is used to modify the specification $\varphi(e)$ to a more stringent specification $\varphi(e;d)$ such that if $\xi_{\mathcal{M}}(\cdot) \models \varphi(e;d)$ then $\xi_{\mathcal{S}}(\cdot) \models \varphi(e)$. Thus, the set of safe controllers for $\mathcal{M}$ for $\varphi(e;d)$ can be used as an approximation for $\mathbb{U}_{\mathcal{S}}(e)$. In particular, if we define the sets $\mathbb{U}_{\varphi(e;d)}$ and $\mathcal{E}_{\varphi}(d)$ as

$$\begin{aligned} \mathbb{U}_{\varphi(e;d)} &:= \{\mathbf{u} \in \mathbb{U}_{\Pi}(e) : \xi_{\mathcal{M}}(\cdot; x_0, \mathbf{u}) \models \varphi(e;d)\} \\ \mathcal{E}_{\varphi}(d) &:= \{e \in \mathcal{E} : \mathbb{U}_{\varphi(e,d)} \neq \emptyset\}, \end{aligned} \tag{8.6}$$

then $\mathbb{U}_{\varphi(e;d)}$ and $\mathcal{E}_{\varphi}(d)$ can be used as an approximation of $\mathbb{U}_{\mathcal{S}}(e)$ and $\mathcal{E}_{\mathcal{S}}$ respectively. Consequently, a verification problem on $\mathcal{S}$ can be converted into a verification problem on $\mathcal{M}$ using the modified specification.

One such distance bound d is given by the simulation metric, SSM, between $\mathcal{M}$ and $\mathcal{S}$ defined as

$$d^a = \max_{e \in \mathcal{E}} \max_{\mathbf{u} \in \mathbb{U}_{\Pi}(e)} \|\xi_{\mathcal{S}}(\cdot; x_0, \mathbf{u}) - \xi_{\mathcal{M}}(\cdot; x_0, \mathbf{u})\|_{\infty} \tag{8.7}$$

Here, the ∞-norm is the maximum distance between the trajectories across all timesteps. Typically SSM is computed over the space of all finite horizon controls $\mathbb{U}$ instead of $\mathbb{U}_{\Pi}(e)$ [137]. Since we are interested in a given control scheme, we restrict this computation to $\mathbb{U}_{\Pi}(e)$. In general, d^a is difficult to compute, because it requires searching over (the potentially infinite) space of controllers and environments. An approximate technique to compute d^a was presented for systems whose dynamics were unknown with probabilistic guarantees in [5].

However, if d^a can be computed then it can be used to modify a specification $\varphi(e)$ to $\varphi(e;d^a)$ as follows: "expand" the avoid set $A(\cdot)$ to get the augmented avoid set $A(\cdot;d^a) = A(\cdot) \oplus d^a$, and "contract" the reach set $R(\cdot)$ to obtain a conservative reach set $R(\cdot;d^a) = R(\cdot) \ominus d^a$ (see Figure 8.1). Here, $\oplus$ ($\ominus$) is the Minkowski sum(difference)[1]. Consequently, $\varphi(e;d^a)$ is the set of trajectories which avoid $A(\cdot;d^a)$ and are always contained in $R(\cdot;d^a)$,

$$\varphi(e;d^a) := \{\xi(\cdot) : \xi(t) \notin A(t;d^a), \xi(t) \in R(t;d^a) \forall t \in \mathcal{T}\}. \tag{8.8}$$

Then it can be shown that any controller that satisfies the specification $\varphi(e;d^a)$ for $\mathcal{M}$ also ensures that $\mathcal{S}$ satisfies the specification $\varphi(e)$.

[1]The Minkowski sum of a set K and a scalar d is the set of all points that are the sum of any point in K and $B(d)$, where $B(d)$ is a disc of radius d around the origin.

Proposition 8 *For any $e \in \mathcal{E}$ and controller $\mathbf{u} \in \mathbb{U}_\Pi(e)$, we have $\xi_\mathcal{M}(\cdot; x_0, \mathbf{u}) \models \varphi(e; d^a)$ implies $\xi_\mathcal{S}(\cdot; x_0, \mathbf{u}) \models \varphi(e)$.*

Proof: *Let us consider for a given environment $e \in \mathcal{E}$ and control $\mathbf{u} \in \mathbb{U}_\Pi(e)$, $\xi_\mathcal{M}(\cdot; x_0, \mathbf{u}) \models \varphi(e; d^a)$. We would like to prove that $\xi_\mathcal{S}(\cdot; x_0, \mathbf{u}) \models \varphi(e)$. From* (8.7)*, we have*

$$\|\xi_\mathcal{S}(t) - \xi_\mathcal{M}(t)\| \leq d^a \; \forall t \in \mathcal{T}. \tag{8.9}$$

From the definition of specification in (8.8)*, we have $\xi_\mathcal{M}(\cdot) \models \varphi(e; d^a)$ if and only if $\xi_\mathcal{M}(\cdot) \in \varphi(e; d^a)$. Therefore, $\xi_\mathcal{M}(t) \notin A(t) \oplus d^a$ and $\xi_\mathcal{M}(t) \in R(t) \ominus d^a \; \forall t \in \mathcal{T}$. Since $\xi_\mathcal{M}(t) \notin A(t) \oplus d^a$,*

$$\|\xi_\mathcal{M}(t) - a\| > d^a \;, \forall t \in \mathcal{T} \;, \forall a \in A(t). \tag{8.10}$$

Combining (8.9) *and* (8.10) *implies that*

$$\|\xi_\mathcal{S}(t) - a\| > 0 \;, \forall t \in \mathcal{T} \;, \forall a \in A(t). \tag{8.11}$$

Equation (8.11) *implies that $\xi_\mathcal{S}(t) \notin A(t)$ for any $t \in \mathcal{T}$. Similarly, it can be shown that*

$$\|\xi_\mathcal{S}(t) - r\| > 0 \;, \forall t \in \mathcal{T} \;, \forall r \in R(t)^c,$$

where $R(t)^c$ denotes the complement of the set $R(t)$. Therefore, $\xi_\mathcal{S}(t) \in R(t) \; \forall t \in \mathcal{T}$. Since $\xi_\mathcal{S}(t) \notin A(t)$ and $\xi_\mathcal{S}(t) \in R(t)$ for all $t \in \mathcal{T}$, we have $\xi_\mathcal{S}(\cdot; x_0, \mathbf{u}) \models \varphi(e)$. ■

Proposition 8 implies that $\mathcal{E}_\varphi(d^a)$ and $\mathbb{U}_{\varphi(e;d^a)}$ can be used as approximations of $\mathcal{E}_\mathcal{S}$ and $\mathbb{U}_\mathcal{S}(e)$ respectively. However, the distance bound in (8.7) does not take into account the reach-avoid specification (environment) for which a controller needs to be synthesized. Thus, d^a can be quite conservative. As a result, the modified specification can be so stringent that the set of environments $\mathcal{E}_\varphi(d^a)$ for which we can synthesize a provably safe controller for the abstraction (and hence for the system) itself will be very small, resulting in a very conservative approximation of $\mathcal{E}_\mathcal{S}$.

8.3.2 Specification-Centric Simulation Metric (SPEC)

To overcome these limitations, we propose SPEC,

$$d^b = \max_{e \in \mathcal{E}} \max_{u \in \mathbb{U}_{\varphi(e)}} d(\xi_\mathcal{S}(\cdot), \xi_\mathcal{M}(\cdot)), \tag{8.12}$$

where

$$d(\xi_\mathcal{S}(\cdot), \xi_\mathcal{M}(\cdot)) = \min_{t \in \mathcal{T}} (\min\{h\left(\xi_\mathcal{M}(t; x_0, \mathbf{u}), A(t)\right), -h\left(\xi_\mathcal{M}(t; x_0, \mathbf{u}), R(t)\right)\}) \mathbb{1}_{(\xi_\mathcal{S}(\cdot) \not\models \varphi(e))} \tag{8.13}$$

Here $\mathbb{U}_{\varphi(e)} := \{\mathbf{u} \in \mathbb{U}_\Pi : \xi_{\mathcal{M}}(\cdot; x_0, \mathbf{u}) \models \varphi(e)\}$ is the set of all controls such that $\mathcal{M}$ satisfies the specification $\varphi(e)$. $\mathbb{1}_l$ represents the indicator function which is 1 if l is true and 0 otherwise, and $h(x, K)$ is the signed distance function defined as

$$h(x, K) := \begin{cases} \inf_{k \in K} \|x - k\|, & \text{if } x \notin K \\ -\inf_{k \in K^C} \|x - k\|, & \text{otherwise.} \end{cases}$$

If for any $e \in \mathcal{E}$, $\mathbb{U}_{\varphi(e)}$ is empty, we define the distance function $d(\xi_{\mathcal{S}}(\cdot), \xi_{\mathcal{M}}(\cdot))$ to be zero. Similarly, if there is no $A(\cdot)$ or $R(\cdot)$ at a particular t, the corresponding signed distance function is defined to be ∞. There are four major differences between (8.7) and (8.12):

1. To compute the d^b we only consider the feasible set of controllers that can be synthesized by the control policy, $\mathbb{U}_{\varphi(e)} \subseteq \mathbb{U}_\Pi(e)$, as all other controllers do not help us in synthesizing a safe controller for $\mathcal{S}$ (as they are not even safe for $\mathcal{M}$).

2. To compute the distance between $\mathcal{S}$ and $\mathcal{M}$, we only consider those trajectories where $\mathcal{S}$ violates the specification. This is because a non-zero distance between the trajectories of $\mathcal{S}$ and $\mathcal{M}$, where the $\xi_{\mathcal{S}} \models \varphi(e)$ does not give us any additional information in synthesizing a safe controller.

3. Within a falsifying $\xi_{\mathcal{S}}$, we compute the minimum distance of the abstraction trajectory from the avoid and reach sets rather than the system trajectory, as that is sufficient to obtain a margin to discard behaviors that are safe for the abstraction but unsafe for the system.

4. Finally, a minimum over time of this distance is sufficient to discard an unsafe trajectory, as the trajectory will be unsafe if it is unsafe at any t.

These considerations ensure that d^b is far less conservative compared to d^a and allows us to synthesize a safe controller for the system for a wider set of environments. We first prove that d^b can be used to compute an approximation of $\mathcal{E}_{\mathcal{S}}$.

Proposition 9 *If* $\mathbb{U}_{\varphi(e;d^b)} \subseteq \mathbb{U}_{\varphi(e)}$*, then* $\xi_{\mathcal{M}}(\cdot; x_0, \mathbf{u}) \models \varphi(e; d^b)$ *implies* $\xi_{\mathcal{S}}(\cdot; x_0, \mathbf{u}) \models \varphi(e)$ $\forall e \in \mathcal{E}, \mathbf{u} \in \mathbb{U}_\Pi(e)$.

Proof: *We prove the desired result by contradiction. Suppose there exists an environment* $e \in \mathcal{E}$ *and a controller* $\mathbf{u} \in \mathbb{U}_{\varphi(e,d^b)}$ *such that* $\xi_{\mathcal{M}}(\cdot; x_0, \mathbf{u}) \models \varphi(e; d^b)$ *but* $\xi_{\mathcal{S}}(\cdot; x_0, \mathbf{u}) \not\models \varphi(e)$.

Since $\xi_{\mathcal{M}}(\cdot; x_0, \mathbf{u}) \models \varphi(e; d^b)$*, we have that,*

$$\forall t \in \mathcal{T}\ \xi_{\mathcal{M}}(t; x_0, \mathbf{u}) \notin A(t; d^b) = A(t) \oplus d^b \tag{8.14}$$
$$\forall t \in \mathcal{T}\ \xi_{\mathcal{M}}(t; x_0, \mathbf{u}) \in R(t; d^b) = R(t) \ominus d^b \tag{8.15}$$

Since d^b *is the solution to* (8.12)*, and* $\mathbf{u} \in \mathbb{U}_{\varphi(e)}$ *(as* $\mathbb{U}_{\varphi(e;d^b)} \subseteq \mathbb{U}_{\varphi(e)}$*) is such that*

$\xi_{\mathcal{M}}(\cdot) \not\models \varphi(e)$, (8.12) and (8.13) imply that,

$$\min_{t\in\mathcal{T}} \left(\min\{h\left(\xi_{\mathcal{M}}(t), A(t)\right), -h\left(\xi_{\mathcal{M}}(t), R(t)\right)\}\right) \leq d^b. \tag{8.16}$$

Therefore, $\exists t' \in \mathcal{T}$ such that

$$\min\{h\left(\xi_{\mathcal{M}}(t'), A(t')\right), -h\left(\xi_{\mathcal{M}}(t'), R(t')\right)\} \leq d^b, \tag{8.17}$$

which implies that either

1. *$h\left(\xi_{\mathcal{M}}(t'), A(t')\right) \leq d^b$, or*
2. *$h\left(\xi_{\mathcal{M}}(t'), R(t')\right) \geq -d^b$*

If $h\left(\xi_{\mathcal{M}}(t'), A(t')\right) \leq d^b$, $\exists a \in A(t')$ such that

$$\|\xi_{\mathcal{M}}(t') - a\| \leq d^b.$$

Therefore, $\xi_{\mathcal{M}}(t') \in A(t') \oplus d^b$, which contradicts (8.14). Similarly, if $h\left(\xi_{\mathcal{M}}(t'), R(t')\right) \geq -d^b$, $\exists r \in R(t')^c$ such that

$$\|\xi_{\mathcal{M}}(t') - r\| \leq d^b,$$

which implies that $\xi_{\mathcal{M}}(t') \notin R(t') \ominus d^b$, which contradicts (8.15).

When $\mathbb{U}_{\varphi(e,d^b)} \not\subseteq \mathbb{U}_{\varphi(e)}$, for any controller $\mathbf{u} \in \mathbb{U}_{\varphi(e,d^b)} \setminus \mathbb{U}_{\varphi(e)}$ such that $\xi_{\mathcal{M}}(\cdot; x_0, \mathbf{u}) \models \varphi(e; d^b)$, we can no longer comment on the behavior of the corresponding system trajectory. This is because while computing d^b, these controllers were not taken into account. ■

Thus, if we define $\mathbb{U}_{\varphi(e;d^b)}$ and $\mathcal{E}_{\varphi}(d^b)$ as in (8.6) then they can be used as approximations of $\mathbb{U}_{\mathcal{S}}(e)$ and $\mathcal{E}_{\mathcal{S}}$ respectively. Note that Proposition 9 requires that the set of controllers that satisfy the modified specification, $\mathbb{U}_{\varphi(e;d^b)}$, is a subset of the set of the controllers that satisfy the actual specification, $\mathbb{U}_{\varphi(e)}$. When $\mathbb{U}_{\Pi}(e) = \mathbb{U}$, this condition is trivially satisfied as the modified specification is more stringent than the actual specification. Other control schemes, such as the set of linear feedback controllers and feasibility-based optimization schemes also satisfy this condition. In fact, in such cases, the proposed metric, d^b, quantifies the tightest (largest) approximation of $\mathcal{E}_{\mathcal{S}}$, i.e., $\nexists d < d^b$, such that $\mathcal{E}_{\varphi}(d) \subseteq \mathcal{E}_{\mathcal{S}}$.

Theorem 6 *Let $\mathbb{U}_{\Pi}$ be such that $\mathbb{U}_{\varphi(e;d_1)} \subseteq \mathbb{U}_{\varphi(e;d_2)}$ whenever $d_1 > d_2$. Let $d \in \mathbb{R}^+$ be any distance bound such that*

$$\forall e \in \mathcal{E}\ , \forall \mathbf{u} \in \mathbb{U}_{\Pi}(e)\ , \xi_{\mathcal{M}}(\cdot) \models \varphi(e; d) \rightarrow \xi_{\mathcal{S}}(\cdot) \models \varphi(e). \tag{8.18}$$

Then $\forall e \in \mathcal{E}, \mathbb{U}_{\varphi(e;d)} \subseteq \mathbb{U}_{\varphi(e;d^b)} \subseteq \mathbb{U}_{\mathcal{S}}(e)$. Moreover, $\mathcal{E}_{\varphi}(d) \subseteq \mathcal{E}_{\varphi}(d^b) \subseteq \mathcal{E}_{\mathcal{S}}$. Hence, $\mathcal{E}_{\varphi}(d^b)$ and $\mathbb{U}_{\varphi(e;d^b)}$ quantify the tightest (largest) approximations of $\mathcal{E}_{\mathcal{S}}$ and $\mathbb{U}_{\mathcal{S}}(e)$ respectively *among all uniform distance bounds d.*

Proof: *Consider any $d > d^b$. From the statement of Theorem 6, we have that $\mathbb{U}_{\varphi(e;d)} \subseteq \mathbb{U}_{\varphi(e;d^b)}$. Hence, $\mathcal{E}_\varphi(d) \subseteq \mathcal{E}_\varphi(d^b)$ follows from the definition of $\mathcal{E}_\varphi(d)$ in (8.6). $\mathbb{U}_{\varphi(e;d^b)} \subseteq \mathbb{U}_{\mathcal{S}}(e)$ and $\mathcal{E}_\varphi(d^b) \subseteq \mathcal{E}_{\mathcal{S}}$ is already ensured by Proposition 9, and hence Theorem 1 follows.*

We now prove that for all $0 < d < d^b$, $\exists\ e \in \mathcal{E}$ such that (8.18) does not hold, and hence the result of Theorem 1 trivially holds. We prove the result by contradiction. Suppose $0 < d < d^b$ be such that (8.18) holds. Let $(e^*, \mathbf{u}^*)$ be the environment, controller pair where $d(\xi_{\mathcal{M}}(\cdot; x_0^*, \mathbf{u}^*), \xi_{\mathcal{S}}(\cdot; x_0^*, \mathbf{u}^*)) = d^b$. Equation (8.12) and (8.13) thus imply that

$$\min_{t\in\mathcal{T}} \left(\min\{h\left(\xi_{\mathcal{M}}(t; x_0^*, \mathbf{u}^*), A^*(t)\right), -h\left(\xi_{\mathcal{M}}(t; x_0^*, \mathbf{u}^*), R^*(t)\right)\}\right) = d^b, \tag{8.19}$$

and $\xi_{\mathcal{S}}(\cdot; x_0^*, \mathbf{u}^*) \not\models \varphi(e^*)$. Equation (8.19) implies that

$$\forall t \in \mathcal{T}, h\left(\xi_{\mathcal{M}}(t; x_0^*, \mathbf{u}^*), A^*(t)\right) \geq d^b \tag{8.20}$$
$$\forall t \in \mathcal{T}, h\left(\xi_{\mathcal{M}}(t; x_0^*, \mathbf{u}^*), R^*(t)\right) \leq -d^b. \tag{8.21}$$

Equations (8.20) and (8.21) imply that

$$\forall t \in \mathcal{T},\ \xi_{\mathcal{M}}(t; x_0^*, \mathbf{u}^*) \notin A^*(t) \oplus d, \xi_{\mathcal{M}}(t; x_0^*, \mathbf{u}^*) \in R^*(t) \ominus d. \tag{8.22}$$

Consequently, we have $\xi_{\mathcal{M}}(\cdot; x_0^*, \mathbf{u}^*) \models \varphi(e^*; d)$. This contradicts (8.18) since $\xi_{\mathcal{S}}(\cdot; x_0^*, \mathbf{u}^*) \not\models \varphi(e^*)$. Therefore, for all $0 < d < d^b$, $\exists\ e \in \mathcal{E}$, $\mathbf{u} \in \mathbb{U}_{\Pi}(e)$, $\xi_{\mathcal{M}}(\cdot) \models \varphi(e; d) \not\to \xi_{\mathcal{S}}(\cdot) \models \varphi(e)$. ■

Theorem 6 states that d^b is the smallest among all (uniform) distance bounds between $\mathcal{M}$ and $\mathcal{S}$, such that a safe controller synthesized on $\mathcal{M}$ is also safe for $\mathcal{S}$. Even though this is a stricter condition than we need for defining $\mathcal{E}_{\mathcal{S}}$, where we care about the existence of at least one safe controller for $\mathcal{S}$, it allows us to use *any* safe controller for $\mathcal{M}$ as a safe controller for $\mathcal{S}$. Formally, $d^b \leq d$, for all $d \in \mathbb{R}^+$ such that $\forall e \in \mathcal{E}_\varphi(d)$, $\forall \mathbf{u} \in \mathbb{U}_{\varphi(e;d)}$, $\xi_{\mathcal{S}}(\cdot) \models \varphi(e)$.

Intuitively, to compute (8.12), we collect all $\xi_{\mathcal{M}}(\cdot), \xi_{\mathcal{S}}(\cdot)$ pairs (across all $e \in \mathcal{E}$ and $\mathbf{u} \in \mathbb{U}_{\varphi(e)}$) where $\xi_{\mathcal{M}}(\cdot) \models \varphi(e)$ and $\xi_{\mathcal{S}}(\cdot) \not\models \varphi(e)$. We then evaluate (8.13) for each pair and take the maximum to compute d^b. By expanding (contracting) every $A(\cdot) \in \mathcal{A}$ ($R(\cdot) \in \mathcal{R}$) uniformly by d^b, we ensure that none of the $\xi_{\mathcal{M}}(\cdot)$ collected above is feasible once the specification is modified, and hence, $\xi_{\mathcal{S}}(\cdot)$ will never falsify $\varphi(e)$. To ensure this, we prove that d^b is the minimum distance by which the avoid sets should be augmented (or the reach sets should be contracted). Thus, d^b can also be interpreted as the minimum d by which the specification should be modified to ensure that $\mathbb{U}_{\varphi(e;d)} \subseteq \mathbb{U}_{\mathcal{S}}(e)$ for all $e \in \mathcal{E}$.

Corollary 1 *Let $d \in [0, d^b]$ satisfies (8.18), then $\xi_{\mathcal{M}}(\cdot) \models \varphi(e; d)$ implies $\xi_{\mathcal{M}}(\cdot) \models$*

$\varphi(e; d^b)$.

Proof: *To prove the corollary, we first prove that if $d_1 > d_2$, then $\xi_{\mathcal{M}}(\cdot) \models \varphi(e; d_1)$ implies $\xi_{\mathcal{M}}(\cdot) \models \varphi(e; d_2)$,$\forall e \in \mathcal{E}$. Since $\xi_{\mathcal{M}}(\cdot) \models \varphi(e; d_1)$, we have*

$$\forall t \in \mathcal{T}, \xi_{\mathcal{M}}(t) \notin A(t) \oplus d_1, \xi_{\mathcal{M}}(t) \in R(t) \ominus d_1$$

Since $d_1 > d_2$, the above equation implies that

$$\forall t \in \mathcal{T}, \xi_{\mathcal{M}}(t) \notin A(t) \oplus d_2, \xi_{\mathcal{M}}(t) \in R(t) \ominus d_2.$$

Therefore, $\xi_{\mathcal{M}}(\cdot) \models \varphi(e; d_2)$. The corollary now follows from noting that for all $0 < d < d^b$, $\exists\ e \in \mathcal{E}$, $\mathbf{u} \in \mathbb{U}_{\Pi}(e)$,$\xi_{\mathcal{M}}(\cdot) \models \varphi(e; d) \not\rightarrow \xi_{\mathcal{S}}(\cdot) \models \varphi(e)$. ■

We conclude this section by discussing the relative advantages and limitations of SPEC and SSM, and a few remarks.

Comparing SPEC and SSM SSM is specification-independent (and hence environment-independent); and hence can be reused across different tasks and environments. This is ensured by computing the distance between trajectories across all input control sequences; however, the very same aspect can make SSM overly-conservative. Making SPEC specification-dependent trades in generalizability for a less conservative measure. Although environment-dependent, the set of safe environments obtained using SPEC is larger compared to SSM. This is an important trade-off to make for any distance metric–the utility of a distance metric could be somewhat limited if it is too conservative.

The computational complexities for computing SPEC and SSM are the same since they both can be computed using Algorithm 8. To compute SSM we sample from a domain of all finite horizon controls. To compute SPEC we additionally need to be able to define and sample from the set of environment scenarios, but we believe that some representation of the environment scenarios is important for practical applications.

Remark 15 *Note that the proposed framework can also be used in the scenarios where there is a deterministic controller for each environment. In such cases, $\mathbb{U}_{\Pi}(e)$ (and $\mathbb{U}_{\varphi(e)}$) is a singleton set for every environment e (see Section 8.4.2 for an example). However, from a control theory perspective, it might be useful to have a set of safe controllers that have different transient behaviors, that the system designer can choose from without recomputing the distance metric.*

Remark 16 *Note that SPEC does not strictly meet the requirements for a metric because of the indicator function within its definition. However, it is possible to remove the indicator function from the definition and constrain the space of feasible environments and controls instead; i.e, we can replace $u \in \mathbb{U}_{\varphi(e)}$ by $u \in \mathbb{U}_{\varphi(e)} \wedge \xi_{\mathcal{S}}(\cdot) \not\models \varphi(e)$ in (8.12). The two definitions are equivalent, and SPEC would no longer be zero for two distinct input arguments; thus, it satisfies the properties of a metric.*

8.3.3 Distance Metric Computation Using Scenario Optimization

Since a dynamics model of $\mathcal{S}$ is not available, the computation of the distance bound d^b is generally difficult. Interestingly, this computational issue can be resolved using a randomized approach, such as scenario optimization [65]. Scenario optimization has been used for a variety of purposes [69, 68], such as robust control, model reduction, as well as for the computation of SSM [5].

Computing d^b by scenario optimization is summarized in Algorithm 8. We start by (randomly) extracting N realizations of the environment e_i, $i = 1, 2, \ldots, N$ (Line 2). Each realization e_i consists of an initial state x_0^i, and a sequence of reach and avoid sets, $A^i(t)$ and $R^i(t)$. For each e_i, we extract a controller $\mathbf{u}_i \in \mathbb{U}_{\varphi(e_i)}$ (Line 5). If such a controller does not exist, we denote $\mathbf{u}_i$ to be a null controller $\mathbf{u}_\phi$. $\mathbf{u}_i$ (if not $= \mathbf{u}_\phi$) is then applied to both the system as well as the abstraction to obtain the corresponding trajectories $\xi_{\mathcal{S}}^i(\cdot; x_0^i, \mathbf{u}_i)$ and $\xi_{\mathcal{M}}^i(\cdot; x_0^i, \mathbf{u}_i)$ (Line 6). We next compute the distance between these two trajectories, d_i, using (8.13) (Line 7). If $\mathbf{u}_i = \mathbf{u}_\phi$, no satisfying controller exists for $\mathcal{M}$, and hence $d(\xi_{\mathcal{S}}(\cdot; x_0^i, \mathbf{u}_n), \xi_{\mathcal{M}}(\cdot; x_0^i, \mathbf{u}_n))$ is trivially 0. The maximum across all these distances, $\hat{d}_\epsilon$, is then used as an estimate for d^b (Line 10).

Although simple in its approach, scenario optimization provides provable approximation guarantees. In Algorithm 8, we have to sample both an $e \in \mathcal{E}$ and a corresponding controller $\mathbf{u} \in \mathbb{U}_{\varphi(e)}$. We define a joint sample space

$$\mathcal{D} = \{(e \times \mathbb{U}_{\varphi(e)}) : e \in \mathcal{E}, \mathbb{U}_{\varphi(e)} \neq \emptyset\} \cup \{(e, \mathbf{u}_\phi) : e \in \mathcal{E}, \mathbb{U}_{\varphi(e)} = \emptyset\} \tag{8.23}$$

$\mathcal{D}$ contains all feasible $(e, \mathbf{u})$ pairs for $\mathcal{M}$. We create a dummy sample $(e, \mathbf{u}_\phi)$ for all e where a satisfying controller does not exist. We next define a probability distribution on $\mathcal{D}$, $p(e, \mathbf{u}) = p(e) \cdot p(\mathbf{u} \mid e)$ where $p(e)$ is probability of sampling $e \in \mathcal{E}$ and $p(\mathbf{u} \mid e)$ is the probability of sampling $\mathbf{u} \in \mathbb{U}_{\varphi(e)}$ given e. This distribution is key to capture the sequential nature of sampling $\mathbf{u}$ only after sampling e. For $e \in \mathcal{E}$ where $\mathbb{U}_{\varphi(e)} = \emptyset$, $p(\mathbf{u}_\phi \mid e) = 1$ since $\mathcal{D}$ has only a single entry for e, i.e, $(e, \mathbf{u}_\phi)$. In Algorithm 8, in Line 2, we sample $e_i \sim p(e)$. In Line 5, we sample $\mathbf{u}_i \sim p(\mathbf{u} \mid e_i)$.

Proposition 10 *Let $\mathcal{D}$ be the joint sample space as defined in* (8.23)*, with the probability distribution $p_{\mathcal{D}} = p(e, \mathbf{u})$. Select a 'violation parameter' $\epsilon \in (0, 1)$ and a 'confidence parameter' $\beta \in (0, 1)$. Pick N such that*

$$N \geq \frac{2}{\epsilon}\left(\ln \frac{1}{\beta} + 1\right), \tag{8.24}$$

then, with probability at least $1 - \beta$, the solution $\hat{d}_\epsilon$ to Algorithm 8 satisfies the following conditions:

1. $\mathbb{P}((e, \mathbf{u}) \in \mathcal{D} : d(\xi_{\mathcal{S}}(\cdot; x_0, \mathbf{u}), \xi_{\mathcal{M}}(\cdot; x_0, \mathbf{u})) > \hat{d}_\epsilon) \leq \epsilon$

2. $\mathbb{P}\Big((e,\mathbf{u}) \in \mathcal{D} : \xi_{\mathcal{M}}(\cdot; x_0, \mathbf{u}) \models \varphi(e; \hat{d}_\epsilon) \rightarrow \xi_{\mathcal{S}}(\cdot; x_0, \mathbf{u}) \models \varphi(e)\Big) > 1 - \epsilon$ *provided* $\mathbb{U}_{\varphi(e,\hat{d}_\epsilon)} \subseteq \mathbb{U}_{\varphi(e)}$.

Proof: *Statement (1) of Proposition 10 follows directly from the guarantees provided by Scenario Optimization (Theorem 1 in [69]). To use the result in [69], we need to prove: (a) computing* d^b *can be converted into a standard Scenario Optimization problem and (b) Algorithm 8 samples i.i.d from* $\mathcal{D}$ *with probability* $p_{\mathcal{D}}$.

(8.12) *can be re-written as* $d^b = \max_{(e,\mathbf{u}) \in \mathcal{D}} d(\xi_{\mathcal{S}}(\cdot), \xi_{\mathcal{M}}(\cdot))$ *which can be formalized as the following optimization problem,*

$$\begin{aligned} &\min g \\ &\text{s.t. } \forall (e,\mathbf{u}) \in \mathcal{D}\ , d(\xi_{\mathcal{S}}(\cdot), \xi_{\mathcal{M}}(\cdot)) \leq g \end{aligned}$$

This is semi-infinite optimization problem where the constraints are convex (in fact, linear) in the optimization variable g *for any given* $(e,\mathbf{u})$. *Statement (1) now follows from Theorem 1 in [69] by replacing* $c = 1$, γ *by* g, Δ *by* $\mathcal{D}$, *and* f *by* $d(\xi_{\mathcal{S}}(\cdot), \xi_{\mathcal{M}}(\cdot)) - g$. *Theorem 1 in [69], however, requires that i.i.d samples are chosen from the distribution* $p_{\mathcal{D}}$. *This can be proved by noticing that, in Algorithm 8, we first sample* $e_i \sim p(e)$ *(in Line 2), and then sample* $\mathbf{u}_i \sim p(\mathbf{u} \,|\, e)$ *(in Line 4.) Hence, every* $(e_i, \mathbf{u}_i)$ *is sampled from* $p_{\mathcal{D}} = p(e) \cdot p(\mathbf{u} \mid e)$. *Since each* $i = 1, \ldots, N$ *is sampled randomly and independent of each other, the* $(e_i, \mathbf{u}_i)$ *pairs are indeed sampled i.i.d from* $p_{\mathcal{D}}$.

Algorithm 8 returns an estimate $\hat{d}_\epsilon$ *for* d^b. *We have already established that* $\hat{d}_\epsilon$ *satisfies the probabilistic guarantees provided by scenario optimization (Statement (1)). From Proposition 9, we have* $\forall (e,\mathbf{u}) \in \mathcal{D}$ *where* $d(\xi_{\mathcal{S}}(\cdot), \xi_{\mathcal{M}}(\cdot)) \leq \hat{d}_\epsilon$, $\xi_{\mathcal{M}}(\cdot; x_0, \mathbf{u}) \models \varphi(e; \hat{d}_\epsilon) \rightarrow \xi_{\mathcal{S}}(\cdot; x_0, \mathbf{u}) \models \varphi(e)$, *provided* $\mathbb{U}_{\varphi(e;\hat{d}_\epsilon)} \subseteq \mathbb{U}_{\varphi(e)}$. *Therefore,*

$$\mathbb{P}\Big((e,\mathbf{u}) \in \mathcal{D} : \xi_{\mathcal{M}}(\cdot; x_0, \mathbf{u}) \models \varphi(e; \hat{d}_\epsilon) \rightarrow \xi_{\mathcal{S}}(\cdot; x_0, \mathbf{u}) \models \varphi(e)\Big) > 1 - \epsilon.$$

■

Intuitively, Proposition 10 states that $\hat{d}_\epsilon$ is a high confidence estimate of d^b, if a large enough N is chosen. If we discard the confidence parameter β for a moment, this proposition states that the size of the violation set (the set of $(e,\mathbf{u}) \in \mathcal{D}$ where the corresponding distance is greater than $\hat{d}_\epsilon$) is smaller than or equal to the prescribed ϵ value. As ϵ tends to zero, $\hat{d}_\epsilon$ approaches the desired optimal solution d^b. In turn, the simulation effort grows unbounded since N is inversely proportional to ϵ.

As for the confidence parameter β, one should note that $\hat{d}_\epsilon$ is a random quantity that depends on the randomly extracted $(e,\mathbf{u})$ pairs. It may happen that the extracted samples are not representative enough, in which case the size of the violation set will be larger than ϵ. Parameter β controls the probability that this happens; and the final result holds with probability $1 - \beta$. Since N in (8.24) depends logarithmically on $1/\beta$; β can be pushed down to small values such as 10^{-16}, to make $1 - \beta$ so close to 1 to lose any practical importance.

Algorithm 8: Scenario optimization for estimating SPEC

```
1  set $\hat{d}_\epsilon = 0$
2  extract $N$ realizations of the environment $e_i, i = 1, 2, \dots, N$
3  **for** $i = 0 : N - 1$ **do**
4      **if** $\mathbb{U}_{\varphi(e_i)} \neq \emptyset$ **then**
5          extract a realization of a feasible controller $\mathbf{u}_i \in \mathbb{U}_{\varphi(e_i)}$
6          run the controller $\mathbf{u}_i$ on $\mathcal{S}$ and $\mathcal{M}$, and obtain $\xi^i_{\mathcal{S}}(\cdot)$ and $\xi^i_{\mathcal{M}}(\cdot)$
7          compute $d_i = d(\xi^i_{\mathcal{S}}(\cdot), \xi^i_{\mathcal{M}}(\cdot))$
8      **else**
9          $\mathbf{u}_i = \mathbf{u}_\phi$ and $d_i = 0$
10 set $\hat{d}_\epsilon = \max_{i \in \{1,2,\dots,N\}} d_i$
```

Finally, once we have a high confidence estimate of d^b, we can use it with Proposition 9 to provide guarantees on the safety of a controller for the system, provided that it is safe for the abstraction. (Statement (2) in Proposition 10)

Note that the controller $\mathbf{u}_i$ is extracted randomly from the set $\mathbb{U}_{\varphi(e_i)}$ (Line 5). Obtaining $\mathbb{U}_{\varphi(e_i)}$ and randomly sampling from it can be challenging in itself depending on the control scheme, Π, and the specification, $\varphi(e_i)$. However, one way to randomly extract $\mathbf{u}_i$ is using rejection sampling, i.e., we randomly sample controllers from the set $\mathbb{U}_\Pi$ until we find a controller that satisfies the specification for the model. Since the controller performance is evaluated only on the model during this process, it is often cheap and does not put the system at risk. Nevertheless, choosing a good control scheme makes this process more efficient, as the number of samples rejected before a feasible controller is found will be fewer (see Example 4 below for further discussion on this). Rejection sampling, however, poses a problem when $\mathbb{U}_{\varphi(e_i)} = \emptyset$ and there is no way of knowing that beforehand. In such cases, one can impose a limit on the number of rejected samples to make sure the algorithm terminates. This problem can also be overcome easily when there is a single safe controller for each environment, i.e., $\mathbb{U}_{\varphi(e_i)}$ is a singleton set (see Remark 1).

Remark 17 *Even though we have presented scenario optimization to estimate d^b, alternative derivative free optimization approaches such as Bayesian optimization, simulated annealing, evolutionary algorithms, and covariance matrix adaptation can be used as well. However, for a lot of these algorithms, it might be challenging to provide formal guarantees on the quality of the resultant estimate of the distance bound.*

Example 4 *We now apply the proposed algorithm to compute d^b for the setting described in Example 3. $\mathbb{U}_{\varphi(e)}$ in this case is given as*

$$\mathbb{U}_{\varphi(e)} =\{\mathbf{u} \in \mathbb{U}_{\Pi}(e) : \|\xi_{\mathcal{M}}(T; x_0, \mathbf{u}) - x^*\|_2 < \gamma\},$$

where $\mathbb{U}_{\Pi}(e)$ is the set of LQR controllers (see Example 3). To illustrate the importance of the choice of distance metric, we compute two different distance metrics between $\mathcal{S}$ and $\mathcal{M}$: d^a in (8.7) and d^b in (8.12). To compute d^b, we use Algorithm 8. To compute d^a, we modify Algorithm 8 to sample a random controller from $\mathbb{U}_{\Pi}(e)$ in Line 5 and compute d_i using (8.7) in Line 7.

According to the scenario approach with $\epsilon = 0.01$ and $\beta = 10^{-6}$, we extract $N = 2964$ different reach-avoid scenarios (i.e., N different final states to reach). For each $e_i, i \in \{1, 2, \ldots, 2964\}$, we obtain a feasible LQR controller $\mathbf{u}_i \in \mathbb{U}_{\varphi(e_i)}$ using rejection sampling. In particular, we randomly sample a penalty parameter q, solve the corresponding Riccati equation to obtain $LQR(q)$, and apply it on $\mathcal{M}$. If the corresponding $\xi_{\mathcal{M}}(\cdot)$ satisfies $\varphi(e_i)$, we use $\mathbf{u}_i$ as our feasible controller sample; otherwise, we sample a new q and repeat the procedure until a feasible controller is found. This procedure tends to be really fast and requires simulating only $\mathcal{M}$. A feasible controller was found within 3 samples of q for all e_i in this case. For d^a, we randomly sample a penalty parameter q and use $LQR(q)$ as the controller.

The obtained distance metrics are $d^a = 0.43, d^b = 0$. Since $d^a < \gamma$, it can be used to synthesize a safe controller for $\mathcal{S}$; however, we can synthesize controller only for those reach-avoid scenarios where $\mathcal{M}$ satisfies a much stringent specification: $\xi_{\mathcal{M}}$ must reach within a ball of radius 0.07 around the target state. Consequently, the set $\mathcal{E}_{\varphi}(d^a)$ is likely to be very small. In contrast, $d^b = 0$; thus, Proposition 10 ensures that any controller designed on $\mathcal{M}$ that satisfies $\varphi(e)$ is guaranteed to satisfy it for $\mathcal{S}$ as well. In particular, the dynamics of $\mathcal{S}$ and $\mathcal{M}$ are same for the state x_1, and state x_2 is uncontrollable for $\mathcal{S}$ and remain 0 at all times. Thus, any controller that reaches within a ball of radius γ around a desired state x_1^* for $\mathcal{M}$, if applied on $\mathcal{S}$, also ensures that the system state reaches within the same ball. Even though this relationship between $\mathcal{S}$ and $\mathcal{M}$ is unknown, d^b is able to capture it only through simulations of $\mathcal{S}$. This example also illustrates that d^b significantly reduces the conservativeness in SSM, and does not unnecessarily contract the set of safe environments.

8.4 Numerical Simulations

We now demonstrate how SPEC can be used to obtain the safe set of environments and controllers for an autonomous quadrotor and an autonomous car. In Section 8.4.1, we demonstrate how SPEC provides much larger safe sets compared to SSM. In Section 8.4.2, we demonstrate how SPEC not only captures the differences between the dynamics of $\mathcal{S}$ and $\mathcal{M}$, but also other aspects of the system, in particular the sensor error, that might affect the satisfiability of a specification.

8.4.1 Safe Altitude Control for Quadrotor

Our first example illustrates how the proposed distance metric behaves when the only difference between the system and the abstraction is the value of one parameter. However, unlike the running example, the system and the abstraction dynamics are non-linear. Moreover, we illustrate how SPEC can be used in the cases where all safe controllers for $\mathcal{M}$ may not be safe for $\mathcal{S}$.

We use the reach-avoid setting described in [114], where the authors are interested in controlling the altitude of a quadrotor in an indoor setting while ensuring that it does not go too close to the ceiling or the floor, which are obstacles in our experiments.

A dynamic model of quadrotor vertical flight can be written as:

$$\begin{aligned} z(t+1) =& z(t) + \Delta T v_z(t) \\ v_z(t+1) =& v_z(t) + \Delta T(ku(t) + g), \end{aligned} \tag{8.25}$$

where z is the vehicle's altitude, v_z is its vertical velocity and u is the commanded average thrust. The gravitational acceleration is $g = -9.8m/s^2$ and the discretization step ΔT is 0.01. The control input $u(t)$ is bounded to $[0, 1]$. We are interested in designing a controller for $\mathcal{S}$ that ensures safety over a horizon of 100 timesteps. In particular, we have $\mathcal{X}_0 = \{(z, v_z) : 0.5 \leq z \leq 2.5 \wedge -3 \leq v_z \leq 4\}$, $\mathcal{A} = \{A(\cdot)\}$, and $\mathcal{R} = \mathbb{R}^2$. The avoid set at any time t is given as $A(t) = \{(z, v_z) \in \mathbb{R}^2 : 0.5m \leq z(t) \leq 2.5m\}$. We again assume that the dynamics in (8.25) are unknown. Consider an abstraction of $\mathcal{S}$ with same dynamics as (8.25) except that the value of parameter k in the abstraction dynamics, $k_{\mathcal{M}}$, is different.

The space of controllers $\mathbb{U}_\Pi(e)$ is given by all possible control sequences over the time horizon (i.e., $\mathbb{U}_\Pi(e) = \mathbb{U}$.) For computing $\mathbb{U}_{\varphi(e)}$, we use the Level Set Toolbox [218] that gives us both the set of initial states from which there exist a controller that will keep the $\xi_{\mathcal{M}}(\cdot)$ outside the avoid set at all times (also called the reachable set), as well as the corresponding least restrictive controller. In particular, we can apply any control when the abstraction trajectory is inside the reachable set and the safety-preserving control (given by the toolbox) when the trajectory is near the boundary of the reachable set. For computation of the distance bounds, we sample a random controller sequence according to this safety-preserving control law. If any initial state lies outside the reachable set, then it is also guaranteed that $\mathbb{U}_{\varphi(e)} = \emptyset$ so we do not need to do any rejection sampling in this case.

When $k_{\mathcal{M}} < k$, $\mathcal{M}$ has strictly less control authority compared to $\mathcal{S}$. Thus, any controller that satisfies the specification for $\mathcal{M}$ will also satisfy the specification for $\mathcal{S}$, so $\mathcal{E}_\varphi(0)$ itself is an under approximation of $\mathcal{E}_{\mathcal{S}}$. SPEC is again able to capture this behavior. Indeed, we computed an estimate for the distance bound using Algorithm 8 and the obtained numbers

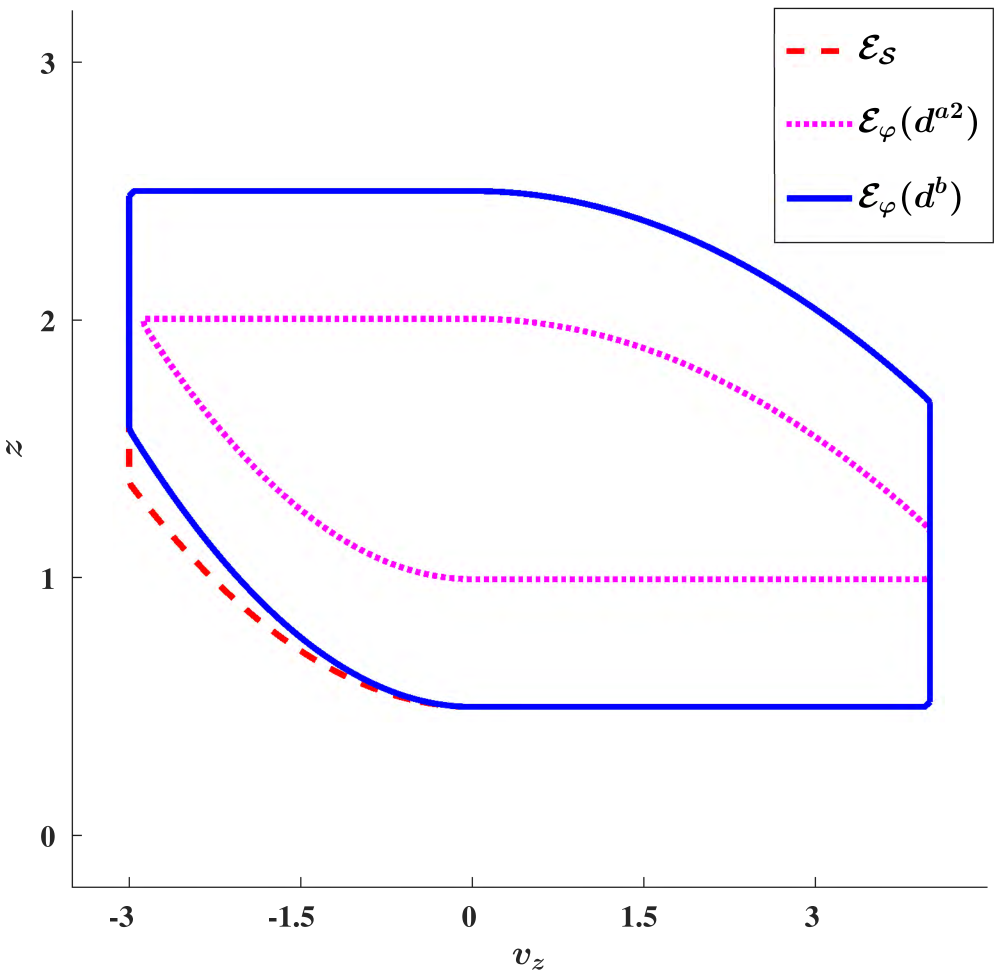


Figure 8.2: Different reachable sets when the quadrotor abstraction is conservative. The distance metric d^b only considers the distance between trajectories that violates the specification on the system and satisfies it on the abstraction, leading to a less conservative estimate of the distance, and a better approximation of $\mathcal{E}_{\mathcal{S}}$.

are $d^a = 0.30$ and $d^b = 0$. Note that not only is d^a conservative, it may not be particularly useful in synthesizing a safe controller for $\mathcal{S}$. d^a computed using Algorithm 8 ensures that a safe controller designed on $\mathcal{M}$ for $\varphi(e; d^a)$ is also safe for $\mathcal{S}$ with high probability, only when this controller is *randomly* selected from the set $\mathbb{U}_{\Pi}$. However, a random controller selected from $\mathbb{U}_{\Pi}$ is unlikely to satisfy $\varphi(e; d^a)$ for $\mathcal{M}$ itself, and thus nothing can be said about $\mathcal{S}$ either. Thus, it is hard to *actually* compute an approximation of $\mathcal{E}_{\mathcal{S}}$. In contrast, d^b samples a controller from the set $\mathbb{U}_{\varphi(e)}$ in Algorithm 8. Therefore, to synthesize a controller, we *randomly* select a controller from the set $\mathbb{U}_{\varphi(e;d^b)}$, which is guaranteed to be safe on both $\mathcal{M}$ and $\mathcal{S}$ with high probability. Therefore, it might be better to compare d^b to d^{a2}, which is defined similar to d^a, except the inner maximum in (8.7) is computed over $\mathbb{U}_{\varphi(e)}$ instead.

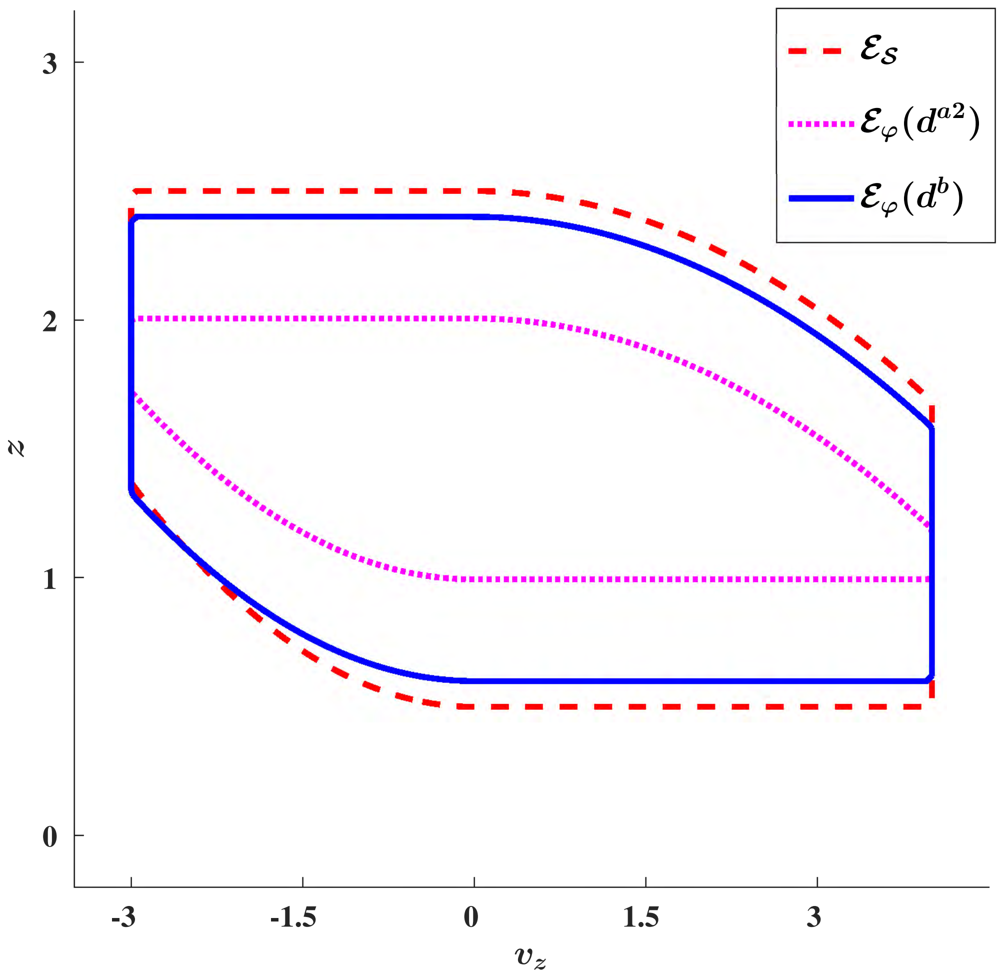


Figure 8.3: Different reachable sets when the quadrotor abstraction is overly optimistic. The distance metric d^b achieves a far less conservative under-approximation of $\mathcal{E}_{\mathcal{S}}$ compared to the other distance metrics.

d^{a2} in this case turns out to be 0.5.

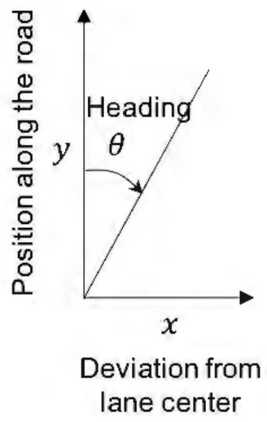
Note that if we could instead compute the distance metrics exactly, $d^{a2} \leq d^{a}$, since $\mathbb{U}_{\varphi(e)} \subset \mathbb{U}_{\Pi}$. However, random sampling based estimate of d^{a2} can be greater than that of d^a if the controllers corresponding to a large distance between the $\xi_{\mathcal{S}}(\cdot)$ and $\xi_{\mathcal{M}}(\cdot)$ are sparse in $\mathbb{U}_{\Pi}$ compared to that in $\mathbb{U}_{\varphi(e)}$.

For illustration purposes, we also compute the reachable set $\mathcal{E}_{\varphi}(d^b)$, by augmenting the avoid set by d^b and recomputing the reachable sets using the Level Set Toolbox. As shown in Figure 8.2, $\mathcal{E}_{\varphi}(0)$ (the area withing the blue contour) is indeed contained within $\mathcal{E}_{\mathcal{S}}$ (the area within the red contour). Here, $\mathcal{E}_{\mathcal{S}}$ has been computed using the system dynamics. Even though $\mathcal{E}_{\varphi}(d^{a2})$ (the area within the magenta contour) is also contained in $\mathcal{E}_{\mathcal{S}}$, it is significantly smaller in size compared to $\mathcal{E}_{\varphi}(d^b)$.

When $k_{\mathcal{M}} > k$, $\mathcal{S}$ has strictly less control authority compared to $\mathcal{M}$. Consequently, there might exist some environments for which it is possible to synthesize a safe controller for $\mathcal{M}$, but the same controller when deployed on $\mathcal{S}$ might lead to an unsafe behavior. We again compute the distance bounds using Algorithm 8 and the obtained numbers are $d^a = 0.30, d^{a2} = 0.49, d^b = 0.1$. The corresponding reachable sets are shown in Figure 8.3. Even though we start with an overly optimistic abstraction, both d^{a2} and d^b are able to compute an under approximation of $\mathcal{E}_{\mathcal{S}}$; however, the set estimated by d^{a2} is, once again, overly conservative.

8.4.2 Safe Lane Keeping for An Autonomous Car

We now show the application of the proposed metric for designing a safe lane keeping controller for an autonomous car. In this example, we use the **Webots** simulator [302]. The car model within the simulator is our $\mathcal{S}$. For the abstraction $\mathcal{M}$ we consider the bicycle model,



$$\begin{aligned}
\dot{x} &= v \cdot \sin\theta \\
\dot{y} &= v \cdot \cos\theta \\
\dot{v} &= a \\
\dot{\theta} &= \frac{v}{l} \tan\omega
\end{aligned} \tag{8.26}$$

where $[x, y, v, \theta]$ is the state, representing perpendicular deviation from the center of the lane, position along the road, speed, and heading respectively. The maximum speed is limited to $v_{max} = 10$ km/hr. We have two inputs, (1) a discrete acceleration control $a = \{-\bar{a}, 0, \bar{a}\}$; and (2) a continuous steering control $\omega \in [-\pi/4, \pi/4]$ rad/s. For our experiments, we use $H = 200$, which translates to about 6 seconds of simulated trajectory. The dynamics of $\mathcal{S}$ are typically much more complex than $\mathcal{M}$ and include the physical effects like friction and slip on the road.

In this case, $\mathcal{X}_0 = \{(x_0, \theta_0) : \|x\| \leq 0.2m \wedge \|\theta\| \leq \pi/4rad\}$; the initial y_0 and v_0 is set to *zero*. $R(t) = \{[x(t), y(t), v(t), \theta(t)] \in \mathbb{R}^4 : \|x(t)\| \leq 0.5m\} \forall t \in \mathcal{T}$. The reach set corresponds to keeping the car within the $0.5m$ of the center of the lane. For keeping the car in the lane, the car is equipped with two sensors, a camera (to capture the lane ahead) and compass (to measure the heading of the car). There is an on board perception module, which first captures the image of the road ahead; and processes it to detect the lane edges and provide an estimate of the deviation of the car from the center of the lane.

There is another car (referred to as the environment car hereon) driving in the front of $\mathcal{S}$, which might obstruct the lane and cause the perception module to incorrectly detect the

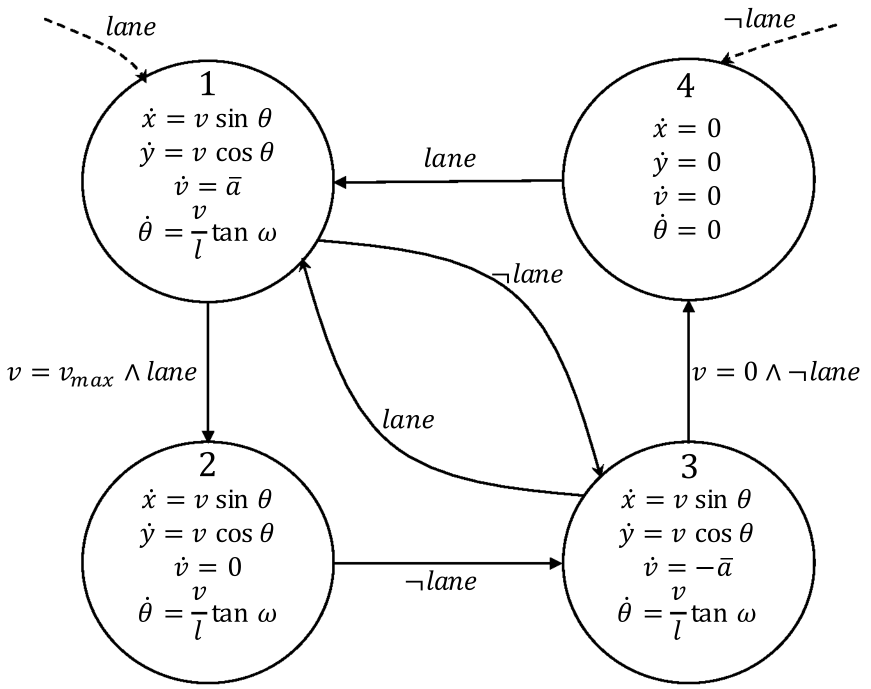
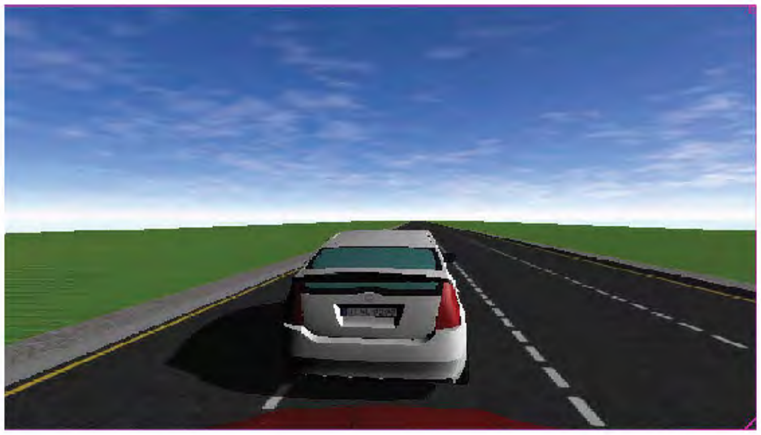


Figure 8.4: Hybrid controller for lane keeping. *lane* means a lane is detected by the perception system. The dashed line represents the transitions taken on initialization based on the value of *lane*. To closely follow the center of the lane, we synthesize a LQR controller in each mode.

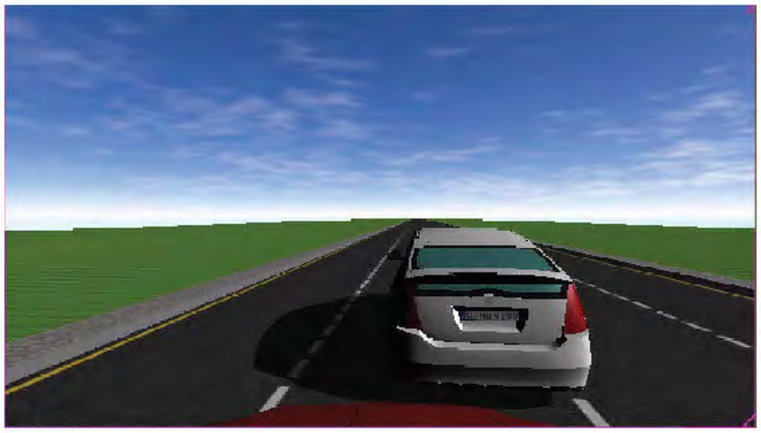
lane center. For each $e \in \mathcal{E}$, the set of possible initial states of the environment car is given by $\mathcal{P} = \{(x_e, y_e) : \|x_e - x_0\| \leq 2.0m \wedge 6.25m \leq y_e - y_0 \leq 8m\}$. We set the initial speed v_e and heading θ_e of the environment car to $v_{\max}$ and 0 respectively. We want to make sure that $\mathcal{S}$ remains within the lane despite all possible initial positions of the environment car. For this purpose, we compute the worst-case d^b across all $p \in \mathcal{P}$.

If the environment car or its shadow covers the lane edges (see Figure 8.5 for some possible scenarios), then the lane detection fails. Technically speaking, if such a scenario occurs, then $\mathcal{S}$ should slow down and come to stop until the image processing starts detecting the lane again. Consequently, our control scheme $\mathbb{U}_\Pi$, is a hybrid controller shown in Figure 8.4, where in each mode the controller is given by an LQR controller (with a fixed Q and R matrix) corresponding to the (linearized) dynamics in that mode. In this example, our controller is a deterministic controller since the Q and R matrices are fixed, and hence $|\mathbb{U}_\Pi| = 1$. In Figure 8.4, in mode (1), the lane is detected and $v(t) < v_{max}$. When the $v(t) = v_{max}$ we transition to mode (2) given the lane is still detected. When the lane is no longer detected, we transition to mode (3) if $v(t) > 0$, or mode (4) if $v(t) = 0$. In modes (3) and (4), the car slows down until the lane is detected again.

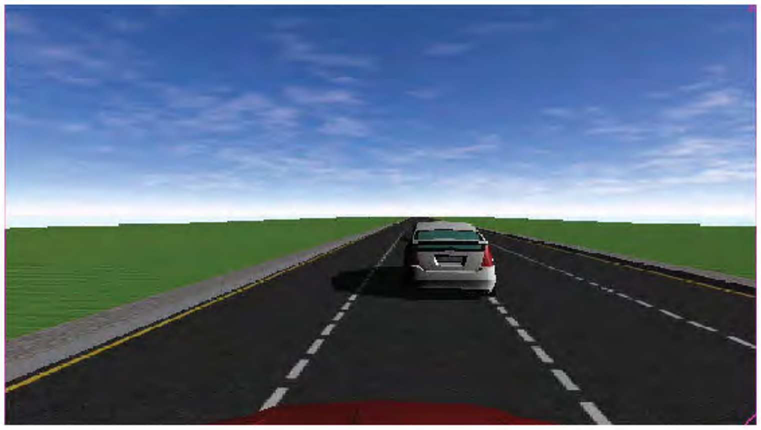
By setting $\epsilon = 0.01$ and $\beta = 1e - 6$ we get $N \geq 2964$. We used Algorithm 8, to sample N different initial states of the $\mathcal{S}$, $(x_0, \theta_0) \in \mathcal{X}_0$; and environment car in the simulator,



(a) Environment car covers left lane.



(b) Shadow of environment car covers left lane.



(c) Lane detected correctly.

Figure 8.5: The lane detection fails for (a) and (b) and $\mathcal{S}$ car tries to slow down. When lane is correctly detected (c), the LQR controller tries to follow the lane

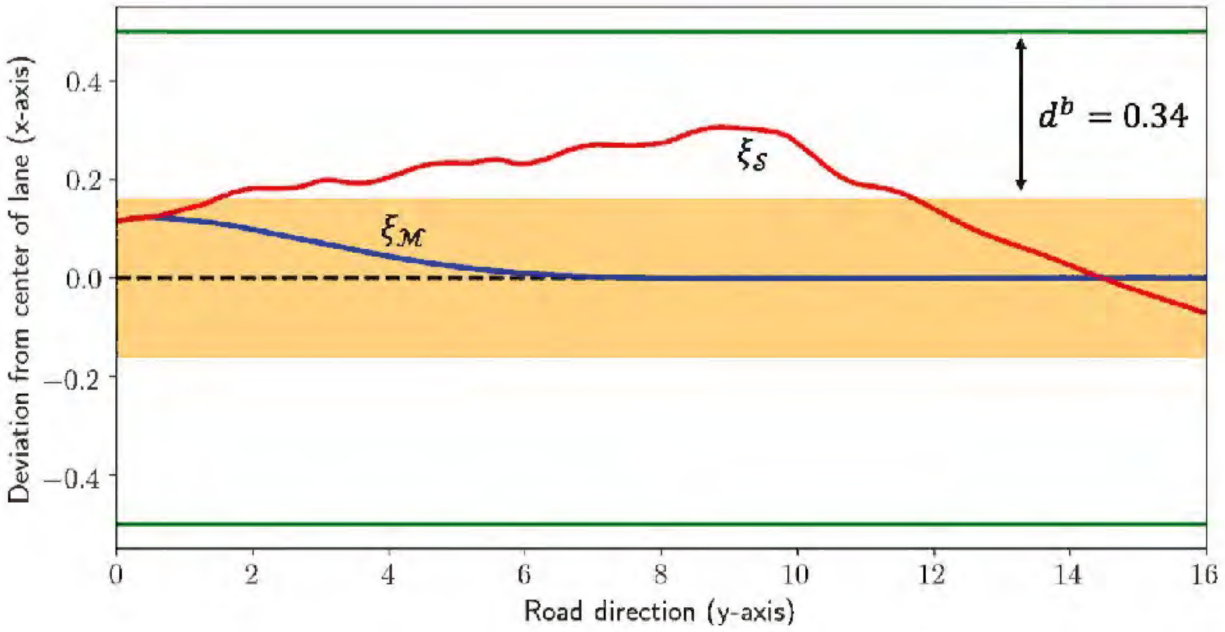


Figure 8.6: The green lines represent the boundaries of the original reach set. The yellow region is the contracted reach set for the model computed using $\hat{d}_\epsilon$. The model's trajectory shown in blue is entirely contained within the yellow region. Consequently, the system's trajectory (shown in dotted red) leaves the yellow region but is contained within the original reach set at all times.

$p \in \mathcal{P}$. Since the controller is deterministic, the set of feasible controllers is a singleton set, and hence we do not need to sample a feasible controller (Line 5 in Algorithm 8). Among these environment scenarios, the controller on $\mathcal{M}$ is also able to safely control $\mathcal{S}$ for 2519 scenarios. $\hat{d}_\epsilon$ is determined entirely by the remaining 445 controller, and computed to be 0.34m. We show the application of the the computed $\hat{d}_\epsilon$ for a sample environment scenario in Figure 8.6. The green lines represent the original reach set. The yellow shaded region represents the contracted reach set for the model computed using $\hat{d}_\epsilon$. The model's trajectory (shown in blue) is contained in the yellow region and hence satisfies the more constrained specification. As a result, even though the system's trajectory (shown in dotted red) leaves the yellow region, it is contained within the original reach set at all times.

We also analyze these 445 environmental scenarios that contribute to $\hat{d}_\epsilon$, and notice that the fault lies within the perception module. In Figure 8.7, we show one such scenario. In this case, $\theta_0 = -\pi/4$. Because of the left rotation of the car, the rightmost lane appears smaller and farther due to the perspective distortion. Furthermore, the presence of the environment car completely cover the rightmost lane in the image. The image processing module now detects the leftmost lane as the center lane and the center lane as the rightmost lane. Consequently, the module returns an inaccurate estimation of the center of the lane, causing $\mathcal{S}$ to go outside the center lane. This example illustrates that the samples in Algorithm 8 that

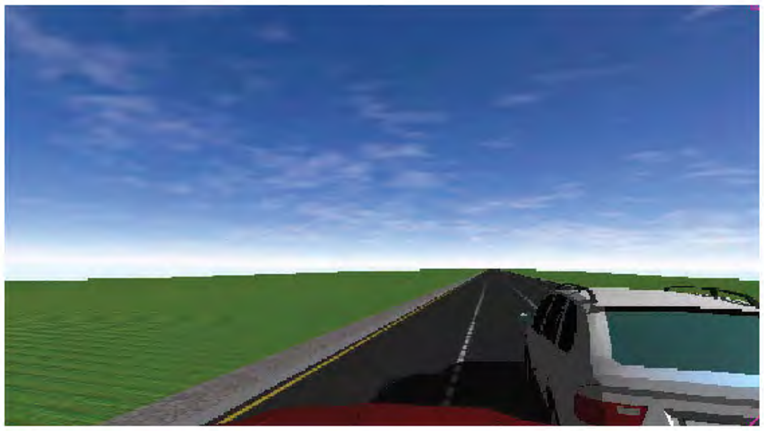


Figure 8.7: An example of the environment scenario that contributes to the distance between the model and the system. The environment samples used for computing SPEC can be used to identify the reasons behind the violation of the safety specification by the system.

contributed to $\hat{d}_\epsilon$ could also be used to analyze the reasons behind the violation of the safety specification by $\mathcal{S}$.

8.5 Chapter Summary

Determining safe environments and synthesizing safe controllers for autonomous systems is an important problem. Typically, we rely on an abstraction of the system to synthesize controllers in different environments. However, when a data-driven model is used to control the system, the relationship between the dynamics of the model and the *actual system* is not known; and hence it is difficult to provide safety guarantees for the system using this model. In this chapter, we propose a specification-centric simulation metric SPEC that can be used to determine the set of safe environments; and to synthesize a safe controller using such data-driven abstractions. We also present an algorithm to compute this metric using executions on the system without knowing its true dynamics. The proposed metric is less conservative and allows controller synthesis for reach-avoid specifications over a broader range of environments compared to the standard simulation metric. Case studies using simulators for quadrotors and autonomous cars illustrate the advantages of SPEC for determining safe environment sets and controllers using incorrect models.

Chapter 9

Safe Learning-Enabled Perception Components

This chapter is based on the paper "An Efficient Reachability-Based Framework for Provably Safe Autonomous Navigation in Unknown Environments" [29] and "Generating Robust Supervision for Learning-Based Visual Navigation Using Hamilton-Jacobi Reachability" [195] written in collaboration with Andrea Bajcsy, Anjian Li, Eli Bronstein, Georgios Giovanis, Varun Tolani, Mo Chen, and Claire Tomlin.

The previous chapter focused on the safety of autonomous system when learned models are used for controlling the system. In this chapter, we will consider the case when learning is used to deal with the unknown environment component instead. Indeed, autonomous vehicles operating in the real world often need to navigate through *a priori* unknown environments using on-board, limited-range sensors. Even though deep learning-based perception approaches can combine this on-board sensor information with robot's prior experience to navigate in completely new buildings, these perception modules inevitably make prediction errors when they encounter out-of-distribution images. Since most of these autonomous systems are safety-critical, it is important to ensure that they operate safely despite the error in the learning module.

Broadly speaking, there are two different ways to ensure safe navigation despite the errors in the perception module: first, one can formally verify the perception module, i.e., find all the input RGB images that may lead to a prediction error. This verification problem can be very challenging due to the high-dimensionality of RGB images and complexity of the modern CNN-based perception modules [273], especially when input images are not generated from a low-dimensional latent distribution. An alternative approach could be to monitor the output of the perception module to recognize a failure and provide a corrective safe action when necessary. This latter approach bypasses the need for an explicit verification of the perception module, while still maintaining system-level safety guarantees.

For both of the above two approaches, the system-level safety analysis is further complicated by the fact that the system is navigating in *a priori* unknown environment; thus, even the unsafe states (for example, all the obstacles in the environment) are not known beforehand. In this chapter, we will present a Hamilton-Jacobi reachability based approach to compute a monitor for the learning-based perception module, as well as a corrective safe action. To deal with the environment uncertainty, we will present an algorithm to update this monitor and safe action during run time, as new environment information is acquired.

9.1 Related Work

An extensive body of research deals with motion planning and safe exploration for robots in unknown environments, some of which focuses on safety guarantees despite modeling error and external disturbances. However, these approaches often do not deal with the limited sensing horizon of perception sensors, especially that of RGB cameras. They also often do not deal with deep learning-based perception modules in the navigation pipeline. Approaches that focus on system-level safety analysis with learning-based perception module in the loop have also been proposed. We cannot hope to summarize all these works here, but we attempt to discuss several of the most closely related approaches.

Verification of Systems with Learning-Based Perception Modules. In the past few years, a significant progress has been made towards verification of autonomous systems with complex, CNN-based perception modules in the feedback loop. A core principle behind several of these approaches is *system-level* specification, wherein a desired end-to-end behavior of the system is formally specified [273, 274]. Subsequently, the system containing the CNN is verified, rather than an explicit verification of the CNN. A key advantage of system-level specification is that it allows for a compositional approach to verification, wherein the inputs to the perception module that might lead to unsafe behavior for the *overall* system are determined (see e.g., [100, 132]). The same approach forms the basis of VerifAI [101], a formal verification toolkit for learning-enabled feedback loops. To perform the falsification of the perception module within VerifAI, a probabilistic programming language, Scenic [120], has been developed. Scenic acts as a low-dimensional representation of the scene (i.e., the RGB image); thus, falsification of the CNN can be performed in a tractable fashion. However, one limitation of this approach is that the falsification of CNN can be very challenging when an underlying low-dimensional representation of the input images is not available, which is often the case for indoor navigation datasets such as SD3DIS and Matterport building datasets [23]. In this work, we overcome this challenge by instead relying on the multi-modality of the perception sensor to construct a runtime monitor for the perception module.

Safe Motion Planning. Methods that ensure safety despite modeling error and disturbances are largely motivated by the trade-off between safety and efficiency during real-time planning. A popular approach is to perform *offline* computations that quantify disturbances and modeling error which can be used *online* to determine collision-free trajectories [206, 151, 279]. Alternatively, [10, 21] use control barrier functions to design provably stable controllers while satisfying given state-space constraints. However, these methods assume that a recursively feasible collision-free path can be obtained despite the unknown environment, which may not be possible in real-world environments. Several works address this problem for single-agent scenarios within a model predictive control framework [256, 259], as well as for multiple vehicles using sequential trajectory planning [268, 36]. However, these works assume *a priori* knowledge of all obstacles, whereas we are interested in a framework that guarantees safety in an *a priori* unknown environments for potentially high-order nonlinear dynamics.

Ensuring safety with respect to both modeling error and limited sensing horizons have been studied using sum-of-squares [178], linear temporal logic [186], reactive synthesis approaches [263], graph-based kinodynamic planner [172] among others. These works typically impose restrictions on sensors or planners to ensure safety with respect to the unknown environment. In contrast, the proposed framework is sensor and planner agnostic, provided that the sensor can accurately identify the obstacles within its sensing region.

Safe Exploration. The problem of finding feasible trajectories to a specified goal in an unknown environment has also been studied in the robotic exploration literature for simplified kinematic motion models using frontier-exploration methods [306] and D* [175]. Other works include sampling-based motion planners for drift-less dynamics [50] and dynamic exploration methods for vehicles with a finite stopping time [159]. Robotic exploration has been also studied within the context of fully and partially observable Markov decision processes [257, 227] and reinforcement learning [8, 165] to reduce collision probabilities; however, no theoretical safety guarantees are typically provided.

Safe exploration has also been studied in terms of Lyapunov stability [55, 81]. Even though stability is often desirable, it is insufficient to guarantee collision avoidance. In contrast, our formulation uses a stronger definition of safety, and is more in line with the run-time verification methods such as [118, 114, 95], which characterize safety using reachable sets.

Contributions and Chapter Organization

In this chapter, we propose a safety framework for autonomous vehicles operating in *a priori* unknown static environments. Our framework is based on Hamilton Jacobi (HJ) reachability analysis. In particular, we treat the unknown environment at any given time as an obstacle and use HJ reachability to compute the *backward reachable set (BRS)*, i.e. the set of states from which the vehicle can enter the unknown and potentially unsafe part of the environment, despite the best control effort. The complement of the BRS therefore represents the safe set for the vehicle and acts as a monitor for the overall feedback loop. With this computation, we also obtain the corresponding least restrictive safety controller, which does not interfere with the planner unless the safety of the vehicle is at risk. Use of HJ reachability analysis in our framework allows us to ensure safety for general nonlinear vehicles, sensors, and planners, including for learning and perception-based planners.

To overcome the challenge of run-time update of the monitor, we build on our work on the local update of the BRS in light of new environment information (see Sec. 3.3.2). We demonstrate our approach on different sensors and planners on a vehicle with nonlinear dynamics in the presence of external disturbances, as well as on a hardware testbed using a state-of-the-art, vision-based planner.

9.2 Problem Formulation

In this work, we study the problem of autonomous navigation in *a-priori unknown static* environments. Consider a stable, deterministic, nonlinear dynamical model of the vehicle

$$\dot{x} = f(x, u, d), \tag{9.1}$$

where $x \in \mathbb{R}^{n_x}$, u, and d represent the state, the control, and the disturbance experienced by the vehicle. Here, d can include the effect of both the external disturbances or dynamics mismatch. For convenience, we partition the state x into the position component $p \in \mathbb{R}^{n_p}$ and the non-position component $h \in \mathbb{R}^{n_x - n_p}$: $x = (p, h)$. We also assume that the vehicle state x can be accurately sensed at all times.

Let x_0 and x^* denote the start and the goal state of the vehicle. The vehicle aims to navigate from x_0 to x^* in an *a priori* unknown environment, $\mathcal{E}$, whose map or topology is not available to the robot. At any time t and state $x(t)$, the vehicle has a planner $\Pi(x(t), x^*, \mathcal{E})$, which outputs the control command $u(t)$ to be applied at time t. This, for example, could be a learning-based planner. The vehicle also has a sensor which at any given time exposes a region of the state space $\mathcal{S}_t \subset \mathbb{R}^{n_x}$, and provides a conservative estimate of the occupancy within $\mathcal{S}_t$. For example, if the vehicle has a camera sensor, $\mathcal{S}_t$ would be a triangular region (prismatic in 3D) representing the field-of-view of the camera. We assume perfect perception within this limited sensor range. Dealing with erroneous perception, sensor noise, and dynamic environments are problems in their own right, and we defer them to future work. Finally, we assume that there is a known initial obstacle-free region around x_0 given by $\mathcal{X}_{\text{init}} \subset \mathbb{R}^{n_x}$; e.g. this is the case when the vehicle is starting at rest and its initial state is collision-free.

Given x_0, x^*, $\mathcal{X}_{\text{init}}$, the planner Π, and the sensor $\mathcal{S}$, our goal in this chapter is to design a least restrictive control mechanism to navigate the vehicle to the goal state while remaining safe, which means avoiding obstacles at all times. Since the environment $\mathcal{E}$ is unknown, the safety needs to be ensured given the partial observations of the environment obtained through the sensor, which in general is challenging. We use the HJ reachability-based framework to ensure safety despite only partial knowledge of the environment.

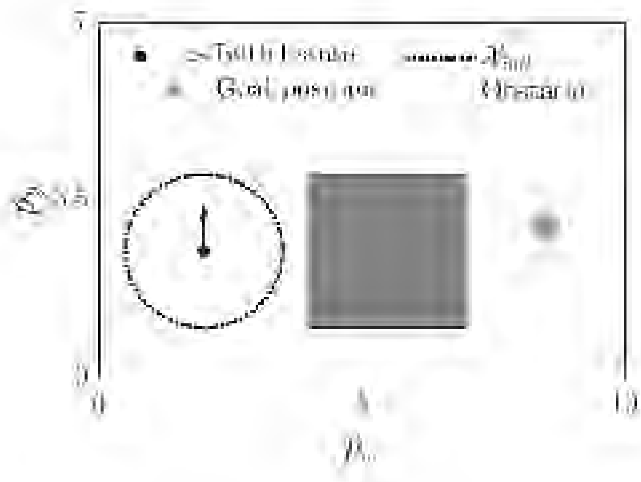
Example 5 *To illustrate our approach and provide intuition behind the proposed framework, we introduce a simple running example: a 3-dimensional Dubins' car system with disturbances added to the velocity. The dynamics of the system are given by:*

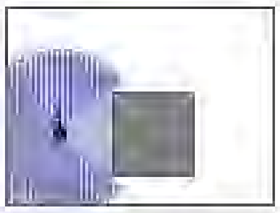
$$\begin{aligned} \dot{p}_x = v\cos\phi + d_x, \quad \dot{p}_y = v\sin\phi + d_y, \quad \dot{\phi} = \omega\,, \\ \underline{v} \le v \le \bar{v}, \quad |\omega| \le \bar{\omega}, \quad |d_x|, |d_y| \le d_r \end{aligned} \tag{9.2}$$

where $x := (p_x, p_y, \phi)$ is the state, $p = (p_x, p_y)$ is the position, ϕ is the heading, and $d = (d_x, d_y)$ is the disturbance experienced by the vehicle. The control of the vehicle is $u := (v, \omega)$, where v is the speed and ω is the turn rate. Both controls have a lower and upper bound, which for this example are chosen to be $\underline{v} = 0.1m/s$, $\bar{v} = 1m/s$, $\bar{\omega} = 1rad/s$. The disturbance bound is chosen as $d_r = 0.1m/s$.
The environment setup for is shown in Figure 9.1. The vehicle start and the goal state are given by $x_0 = [2, 2.5, \pi/2]$ (shown in black) and $x^* = [8.5, 3, -\pi/2]$ (the center of the green area). The goal is to reach within $0.3m$ of x^* (the light green area). However, there is an obstacle in the environment which is not known to the vehicle beforehand (shown in grey). At the beginning of the running example navigation task, we assume that there is no obstacle within $1.5m$ of x_0, and obtain the initial obstacle-free region $\mathcal{X}_{init} := \{x : \|p - p_0\| \le 1.5\}$ (the area inside the dashed black line).
To demonstrate the sensor-agnostic nature of our approach, we simulate the Dubins' car with two different sensors: a LiDAR and a camera. For a LiDAR, the sensing region $\mathcal{S}_t$ is given by a circle of radius R centered around the current position $p(t)$, where $R = 3m$ in this simulation (shown in Figure 9.1a). For a camera, the sensing region $\mathcal{S}_t$ is determined by a triangular region with solid angle F (also called the field-of-view) and apex at the current vehicle heading, and a maximum extent of R. We use $F = \pi/3$ and $R = 20m$ for our simulations (shown in Figure 9.1b). However, part of the regions of $\mathcal{S}_t$ can be occluded by the obstacles, as would be the case for any real-world sensors.
Additionally, for each sensor, we demonstrate our approach on two different planners Π: a sampling-based planner and a model-based planner. For the sampling-based planner, we use a Rapidly-Exploring Random Tree (RRT) [191], and for the model-based planner, we use a spline-based planner [300]. Our goal is to safely navigate to the goal position despite the unknown obstacles.

9.3 Reachability-Based Safety Monitor in Unknown Environments

We propose an HJ-reachability-based framework to ensure safety in an *a priori* unknown environment. Our framework treats the unsensed environment at any given time as an




(a) LiDAR sensing region

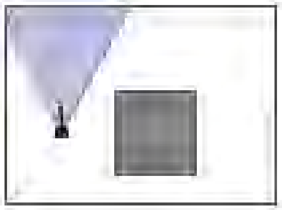

(b) Camera sensing region

Figure 9.1: The initial setup for the running example. The goal is to safely reach the goal (center of the green area) from the initial position (black marker) in the presence of an unknown obstacle (the grey square). We also show the initial sensing region for the LiDAR and camera sensors.

obstacle. The unsensed environment along with the sensed obstacles are used to compute a safe region for the vehicle using HJI reachability. This ensures that the vehicle never enters the unknown and potentially unsafe part of the environment, despite the worst case disturbance.

Let $\mathcal{F}_t$ denote the sensed obstacle-free region of the environment at any time t. Given the initial obstacle-free region $\mathcal{F}_0 = \mathcal{X}_{\text{init}}$, we compute the corresponding safe set $\mathcal{K}_0$ by solving the HJI VI, assuming everything outside $\mathcal{F}_0$ is an obstacle. The safe set is given by the viability kernel of $\mathcal{F}_0$. The HJI VI to compute the viability kernel (or safe set) is given as:

$$\min\{D_t V(\tau, x) + H(\tau, x, V(\tau, x)),\ l_0(x) - V(\tau, x)\} = 0\ \forall x, \tau \qquad V(\infty, x) = l_0(x), \quad (9.3)$$

where the terminal value function, $V(\infty, x) := l_0(x)$ is positive inside $\mathcal{F}_0$ and negative outside. For more details on the computation of viability kernel, we refer the interested readers to Sec. 2.3.2.3.

Starting from $l_0(x)$, the HJI VI is solved to obtain the converged value function $V_0(x) := \lim_{\tau \to 0} V(\tau, x)$. $V_0(x)$ is then used to compute the safe region $\mathcal{K}_0$ as follows:

$$\mathcal{K}_0 = V_0(x) > 0. \quad (9.4)$$

As long as the vehicle is inside $\mathcal{K}_0$, a controller exists to ensure that it does not collide with the known or unknown obstacles.

We next execute a controller on the system for the time horizon $t \in [0, H)$ as per the following control law:

$$u(t) = \begin{cases} \Pi(x(t), x^*, \mathcal{E}), & \text{if } x(t) \in \mathcal{K}_t \\ u^*(t, x(t)), & \text{otherwise} \end{cases} \tag{9.5}$$

where $u^*(t, x(t))$ is the optimal safe controller corresponding to $\mathcal{K}_t$ and is given by:

$$u^*(t, x) = \sup_{u \in \mathcal{U}} \inf_{d \in \mathcal{D}} \langle D_x V_0(x), f(x, u, d) \rangle. \tag{9.6}$$

Also, until the safe set is updated, we use the last computed safe set for finding the optimal safe controller, i.e., $\mathcal{K}_t = \mathcal{K}_0 \; \forall t \in [0, H)$. The control mechanism in (9.5) is least restrictive in the sense that it lets the planner execute the desirable control on the system, except when the system is at the risk of violating safety. Note that the control horizon H in our framework can be arbitrarily chosen by the system designer while still ensuring safety.

While the system is executing the control law in (9.5), it will obtain new sensor measurements $\mathcal{S}_t$ at each time t, which is used to obtain $\mathcal{F}_t$, the free space sensed at that time. If the sensor is completely occluded by an obstacle at any time, the corresponding free space is an empty set. Thus, the overall *known* free space at time t is given by:

$$\mathcal{M}_t = \bigcup_{s \in [0,t]} \mathcal{F}_\tau. \tag{9.7}$$

At the end of the control horizon, H, we compute another safe region $\mathcal{K}_H$ assuming everything outside $\mathcal{M}_T$ is an obstacle. This safe region is, once again, obtained by solving the HJI VI until convergence. We then execute a control law similar to in (9.5), except that the safety controller intervenes only when the system is at the boundary of $\mathcal{K}_H$. The entire procedure is repeated until the system reaches the goal state.

Since the safety controller does not allow the system trajectory to leave the known free space, the proposed framework is guaranteed to avoid collision at all times. However, the safe set can be rather conservative especially early on when most of the environment is still unexplored, which is a trade-off we make to ensure safety against all unexpected obstacles. If additional information about the obstacles in the environment is known, it can be incorporated and will only reduce the conservativeness of the safe set.

Note that the safe set does not necessarily need to be updated every H seconds. It can be updated faster, slower or at the same rate as the control horizon. Essentially one can use the most recent safe set in the control law in (9.5), and still ensure safety at all times. This is because the safe set at any time t_1 is only smaller than the safe set at time t_2 when $t_1 < t_2$. However, the safe set should be updated as quickly as possible to minimize interference with the planner.

9.3.1 Updating Safe Set Using Warm-Start Reachability

To overcome the computational challenges associated with updating the safe set in real time, we build upon our work on warm-start reachability and local update of reachable sets (see

Algorithm 9: Safe navigation using HJ reachability

```
1  $x_0, x^*$: Start and the goal states
2  $\mathcal{F}_0 \longleftarrow \mathcal{X}_{\text{init}}$: The initial obstacle-free region
3  $\mathcal{E}$: The unknown environment
4  $\Pi(\cdot, x^*, \mathcal{E})$: The planner for the vehicle
5  $H$: The control horizon
6  $\mathcal{K}_0$: The initial safe region obtained by solving the HJI VI in (9.3)
7  $\mathcal{K}_{\text{last}} \longleftarrow \mathcal{K}_0$; $\mathcal{V}_{\text{last}} \longleftarrow \mathcal{K}_0^c$; $t_{\text{last}} \longleftarrow 0$: The last computed safe set, BRS, and the
     corresponding time
8  while the vehicle is not at the goal do
9      Obtain the current sensor observation $\mathcal{S}_t$ and free space $\mathcal{F}_t$
10     Apply the least restrictive control $u(t)$ given by (9.5)
11     for every $H$ seconds do
12         Obtain the current map $\mathcal{M}_t$ of the environment using (9.7)
13         Warm-start the BRS computation using $\mathcal{V}_{\text{last}}$, $\mathcal{M}$, and (9.8)
14         Obtain the new safe region, $\mathcal{K}_t$, by solving the HJI VI with the warm-started
             value function
15         $\mathcal{K}_{\text{last}} \longleftarrow \mathcal{K}_t$; $\mathcal{V}_{\text{last}} \longleftarrow \mathcal{K}_t^c$; $t_{\text{last}} \longleftarrow t$
```

Section 3.3). In particular, given the last computed safe set at time t_{last}, the maps at t_{last}, and the current time t, we "warm-start" the value function for the BRS computation at time t as follows:

$$V_t(\infty, x) = \begin{cases} l_t(x), & \text{if } x \in \mathcal{M}_t \cap \mathcal{M}_{t_{\text{last}}}^c \\ V_{t_{\text{last}}}(x), & \text{otherwise} \end{cases} \tag{9.8}$$

where $l_t(x)$ as before is defined such that it is positive inside $\mathcal{M}_t$ and negative outside. Intuitively, instead of initializing the value function with $l_t(x)$ everywhere in the state space, (9.8) initializes it with the last computed value function for the states where no new information has been obtained since the last computation, and with $l_t(x)$ only at the states which were previously assumed to be occupied but are actually obstacle-free. This leads to a much faster computation of the BRS because the value function needs to be updated only for a much smaller number of states that are newly found out to be free. At all the other states, the value function is already almost accurate and only small refinements are required. Interestingly, this procedure also maintains the conservativeness of the safe region, which is sufficient to ensure collision avoidance at all times. We refer the interested readers to Theorem 4 in Section 3.3 for a formal proof.

Our overall approach with warm-starting to update the safe set is summarized in Algorithm 9. We start with the initial known free space $\mathcal{X}_{\text{init}}$ and compute the initial safe set $\mathcal{K}_0$ using the HJI VI (Line 6). The value function for this computation is initialized by the signed distance to $\mathcal{X}_{\text{init}}$. We also maintain the last computed BRS $\mathcal{V}_{\text{last}}$, the safe set $\mathcal{K}_{\text{last}}$, and

Algorithm 10: Local update of the BRS

```
1  $\mathcal{Q} \longleftarrow \mathcal{M}_t \cap \mathcal{M}^c_{t_{\text{last}}}$: Initialize list of states for which the value function should be
     updated
2  $\mathcal{Q} \longleftarrow \mathcal{Q} \cup \mathcal{N}(\mathcal{Q})$: Add neighboring states to $\mathcal{Q}$
3  Warm-start the value for states in $\mathcal{Q}$, $V_t(0, \mathcal{Q})$, using (9.8)
4  $V_{\text{old}} \longleftarrow V_t(0, \mathcal{Q})$: The last computed value function for states in $\mathcal{Q}$
5  while $\mathcal{Q}$ is not empty do
6      $V_{\text{updated}} \longleftarrow$ Update the value function $V_{\text{old}}$ for a time step $\Delta T$
7      $\Delta V = \|V_{\text{updated}} - V_{\text{old}}\|$: Change in the value function
8      $\mathcal{Q}_{\text{remove}} \longleftarrow \{x \in \mathcal{Q} : \Delta V = 0\}$: States with unchanged value
9      $\mathcal{Q} \longleftarrow \mathcal{Q} - \mathcal{Q}_{\text{remove}}$: Remove states with unchanged value
10     $\mathcal{Q} \longleftarrow \mathcal{Q} \cup \mathcal{N}(\mathcal{Q})$: Add neighboring states to $\mathcal{Q}$
11     $V_{\text{old}} \longleftarrow V_{\text{updated}}$
```

the corresponding time t_{last} (Line 7). At every state, the vehicle obtains the current sensor observation and extracts the sensed free space (Line 9 and 10). Next, a control command is applied to the vehicle (Line 11). If the vehicle is inside $\mathcal{K}_{\text{last}}$, the planner is used to obtain the control command; otherwise, the safety controller is applied. Every H seconds, the safe set and controller are updated based on the free space sensed by the vehicle so far using the HJI VI (Line 15). The value function for this computation is warm-started with V_{last} except at the states which are discovered to be obstacle free since t_{last} as described in (9.8) (Line 14). The whole procedure is repeated until the vehicle reaches its goal.

9.3.2 Updating Safe Set Using Local Update

One can further accelerate the update of safe set using local update along with warm-start reachability. Our safety framework is still same as what described in Algorithm 9—only the computational procedure for the safe set computation (Line 15 in Algorithm 9) is being modified to update the value function locally. We outline this procedure in Algorithm 10.

Algorithm 10 closely mimics Algorithm 1 in Section 3.3 by noting that $\mathcal{L} = \mathcal{M}_t$, $\mathcal{L}' = \mathcal{M}_{t_{\text{last}}}$, and $V^*_l(x) = V_{t_{\text{last}}}(x)$ in Equation (3.66). However, we repeat the Algorithm here for completeness. In Algorithm 10, we maintain a list of states $\mathcal{Q}$ at which the value function needs to be updated in light of the new environment observations. $\mathcal{Q}$ is initialized to be the set of states that are newly discovered to be free since t_{last}, i.e., $\mathcal{Q} = \mathcal{M}_t \cap \mathcal{M}^c_{t_{\text{last}}}$ (Line 1). Since the change in the value of the states in $\mathcal{Q}$ (compared to $V_{t_{\text{last}}}(x)$) would also cause a change in the value of the neighboring states, $\mathcal{N}(\mathcal{Q})$, we also add them to $\mathcal{Q}$ (Line 2). Thus, $\mathcal{Q} = \mathcal{Q} \cup \mathcal{N}(\mathcal{Q})$. Once the neighbors are added to $\mathcal{Q}$, the value for all the states in $\mathcal{Q}$ is initialized as per (9.8) (Line 3), and their value is updated using the HJI VI in (4.7) for some time step ΔT (Line 6). This computation is much faster than a classical HJI VI computation since it is typically performed for many fewer states. Next, we remove all those states from $\mathcal{Q}$ whose value function hasn't changed significantly over this ΔT (Line 8 and 9), as these states won't cause any change in the value function for any other state. The neighbors of the remaining states are next added to $\mathcal{Q}$ (Line 10) and the entire procedure is repeated until the value function is converged for all states. Note that Algorithm 10 still maintains the conservatism of the safe set since it is just a different computational procedure for computing the warm-started value function, which is still used within the safety framework in Algorithm 9.

9.4 Numerical Simulations

9.4.1 Running example revisited

We now return to our running example and demonstrate the proposed approach in simulation (described in Example 5). We implement our safety framework with three different methods to update the BRS: using the full standard HJI VI, the warm-start approach, and the local update approach. The corresponding system trajectories for different planners and sensors for all the three methods are shown in Figure 9.2. For all combinations of planners and sensors, the proposed framework is able to safely navigate the vehicle to its goal position despite the external disturbances and no *a priori* knowledge of the obstacle (none of the trajectories go through the obstacle). As the vehicle navigates through the environment, the planner makes optimistic decisions at several states that might lead to a collision; however, the safety controller intervenes to ensure safety. States where the safety controller is applied are marked in red. Note that the safety controller intervenes more frequently for the camera sensor as compared to the LiDAR. This is because the field-of-view (FoV) of a camera is typically much narrower than a LiDAR (which senses the obstacles in all directions within a range). Given this limited FoV, the safety controller needs to account for a much larger unexplored environment, which in turn leads to more cautious control.

We compare the computation time required for each of the three methods to compute the BRS for the camera and LiDAR sensors in Table 9.1. All computations were done on a MATLAB implementation on a desktop computer with a Core i7 5820K processor using the Level Set toolbox [219]. As expected, across all scenarios, warm-starting the value function for the BRS computation leads to a significant improvement in computation time compared to the full HJI VI; however, the computation time might still not be practical for most real-world applications. Only locally updating the value function in addition to warm-starting leads to a significant further improvement in the computation time, and the BRS is updated in approximately 1s on average for all sensors and planners. This improvement is impressive considering that the computation was done in MATLAB without any parallelization of BEACLS which is known to decrease the computation time by a factor of 100 (Section 3.4).

Theorem 4 indicates that the safe set obtained by warm-starting the value function is conservative compared to the one obtained by the full HJI VI. Therefore, we also compare the percentage volume of the states at which the safe set is conservative. This over-conservative

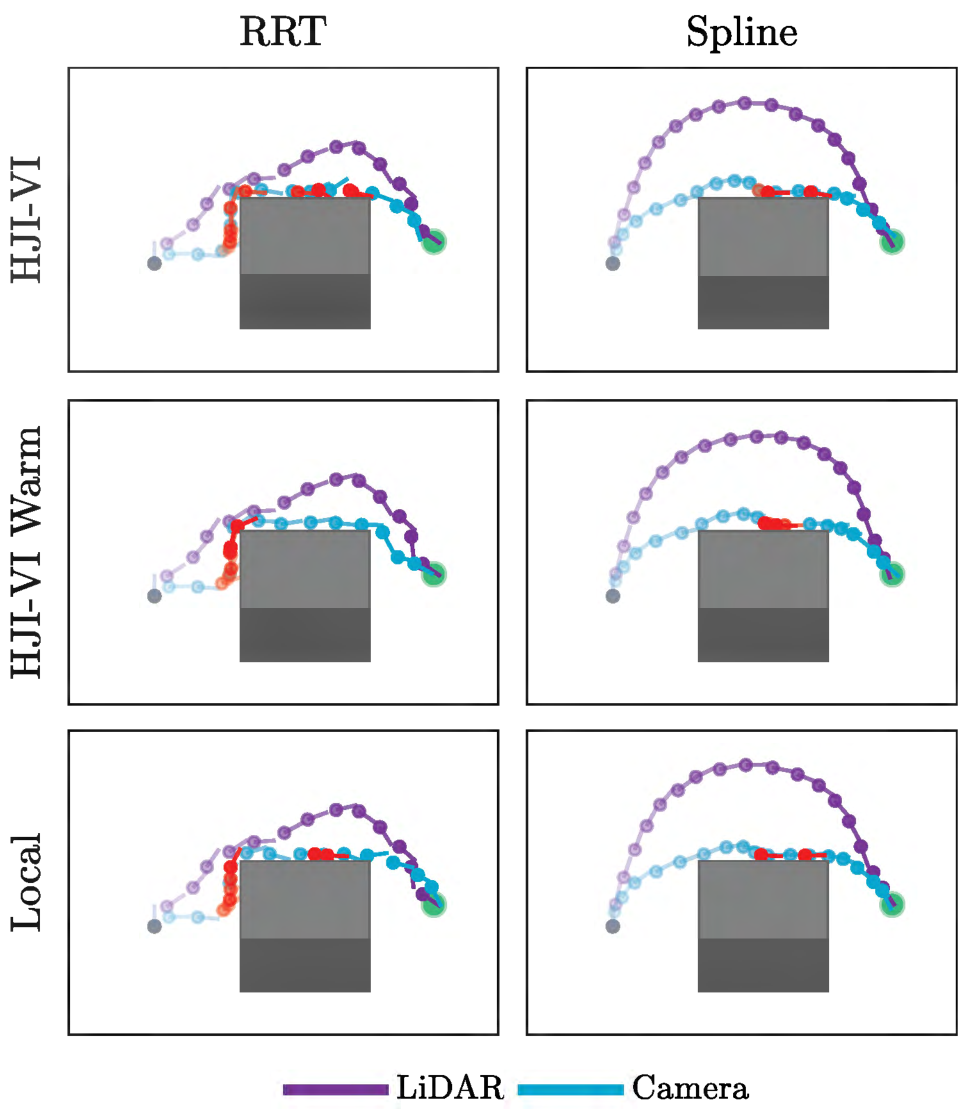


Figure 9.2: The vehicle trajectories for the problem setting in Figure 9.1 for both planners (RRT and Spline planners) and both sensors (LiDAR and Camera sensors) with the safety controller computed from each of the three candidate safety approaches. The proposed framework is able to safely navigate the vehicle to the goal in all cases. When the planner makes unsafe decisions, the safety controller intervenes (the states marked in red) to ensure safety.

volume is typically limited to 0.5% which indicates that the warm-starting approach is able to approximate the true value function quite well.

Finally, we take a closer look at how the safe control comes into play when the system is operating with a range-limited sensor. Figure 9.3a showcases a Dubins' car with a camera sensor and an RRT planner, where the current robot state is shown in black, the corre-

Simulated Camera Results				
Metric	Planner	HJI VI	Warm	Local
Average Compute Time (s)	RRT	45.688	26.290	0.596
	Spline	51.723	12.489	0.898
% Over-conservative States	RRT	0.0	1.112	0.517
	Spline	0.0	0.474	0.506

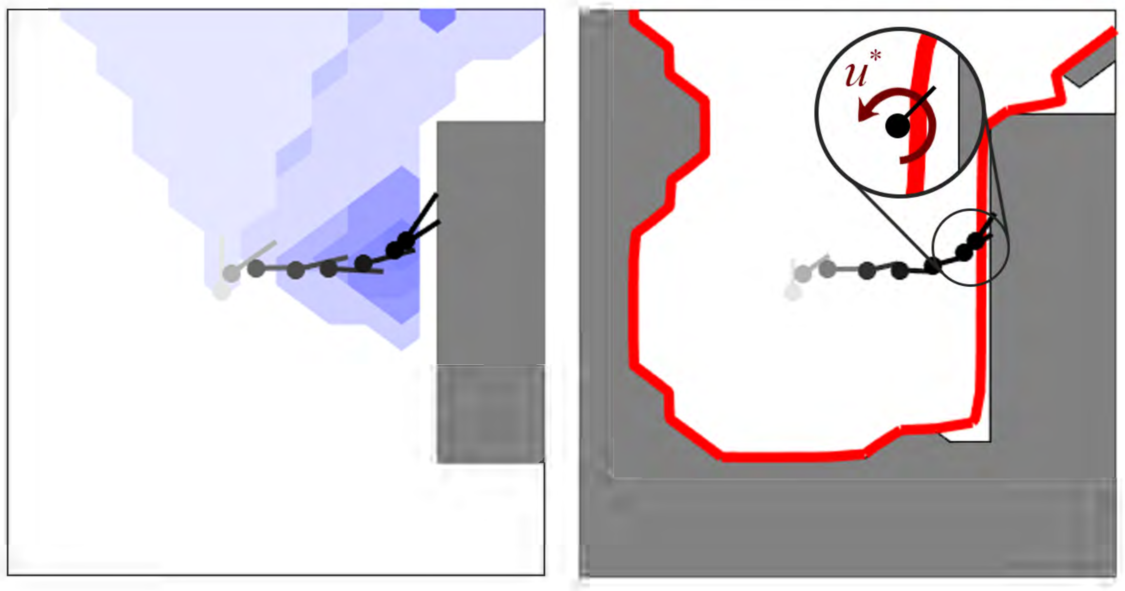
Simulated LiDAR Results				
Metric	Planner	HJI VI	Warm	Local
Average Compute Time (s)	RRT	21.145	6.075	1.108
	Spline	25.318	3.789	1.158
% Over-conservative States	RRT	0.0	0.032	0.290
	Spline	0.0	0.024	0.240

Table 9.1: Numerical comparison of average compute time and relative volume of over-conservative states for each planner and sensor across different BRS update methods. Local updates compute an almost exact BRS in ≈1 second, and are significantly faster than both a standard HJI VI and warm-start.

sponding sensed region is in dark blue, and the trajectory and corresponding sensed regions are shown in grey and light blue respectively. Since the camera's FoV is occluded by an obstacle at the current state, it cannot sense the environment past the obstacle. Figure 9.3b illustrates the corresponding current belief map of the environment which is the union of the free space sensed by the vehicle so far (shown in white). Since the current sensed region is contained within the sensed region at the previous state, no new environment information is obtained and hence the BRS is not updated. The slice of the safe set at the current vehicle heading is shown in Figure 9.3b (the area within the red boundary). Since the vehicle is at the boundary of the safe set, the safety controller intervenes and applies a control u^* that leads the system towards the interior of the safe set (the red arrow) to ensure collision avoidance.

9.4.2 Safety for a learning-based planner

Since the proposed safety framework is planner-agnostic, we can use it to ensure safe navigation even in the presence of a learning-based planner. In particular, we use the vision-based planner introduced in Chapter 7 for navigation in *a priori* unknown environment. The plan-



(a) (b)

Figure 9.3: (a) The sensed region by the vehicle at different states in time for the camera sensor. (b) The overall free space sensed by the vehicle and the corresponding safe set (interior of the red boundary). Since the vehicle is at the boundary of the safe set, the safety controller u^* is applied to steer the robot inside the safe set and ensure collision avoidance.

ner takes an RGB camera image and the goal position as input, and uses a Convolutional Neural Network-based perception module to produce a desired next state that moves the robot towards its goal while trying to avoid obstacles on its way. This desired next state is used by a model-based low-level planner to produce a smooth trajectory from the vehicle's current state to the desired state. Even though we demonstrated that the proposed planner can leverage robot's prior experience to navigate efficiently in novel indoor cluttered environments, it still leads to collisions in several real-world scenarios, like when the vehicle needs to go through narrow spaces or around new objects that it hasn't seen during training. We use the proposed safety framework to ensure both safe planning in such difficult, out-of-distribution navigation scenarios.

The task setup for our simulation is shown in Figure 9.4a. The robot needs to go through a very narrow hallway, followed by a door into the room to reach its goal (the green circle) starting from the initial state (black arrow). At the beginning, the robot has no knowledge about the obstacles (shown in dark grey). We simulate this scenario using the S3DIS dataset which contains mesh scans of several Stanford buildings. By rendering this mesh at any state, we can obtain the image observed by the camera (used by the planner) as well as the occupancy information within the robot's FoV (used for the safety computation). For the robot dynamics, we use the 4D Dubins' car model:

$$\dot{p}_x = v\cos\phi, \quad \dot{p}_y = v\sin\phi, \quad \dot{v} = a, \quad \dot{\phi} = \omega \tag{9.9}$$

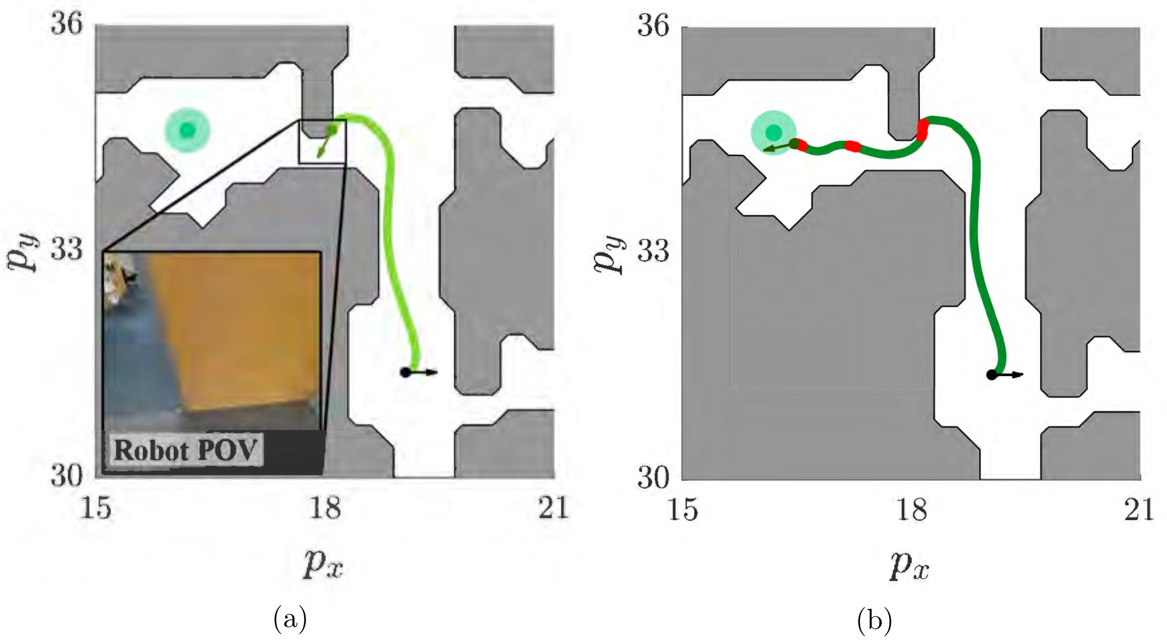


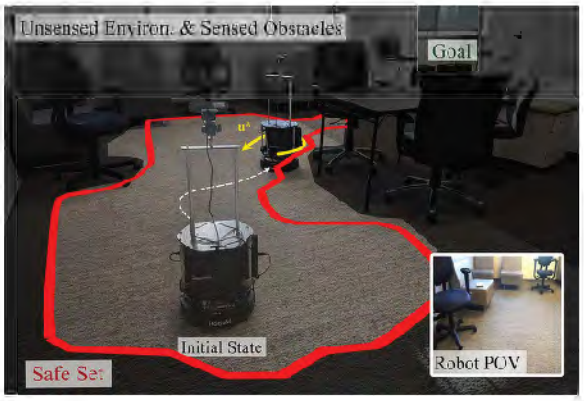
Figure 9.4: The proposed framework can be exploited to provide safety guarantees around vision-based planners that incorporate learning in the loop. The vision-based planner plans a path through the doorway. Without safety control (a) this results in collision, however with safety (b) the robot avoids collision and reaches the goal.

where $p = (p_x, p_y)$ is the position, ϕ is the heading, and v is the speed of the vehicle. The control is $u := (a, \omega)$, where $|a| \leq 0.4$ is the acceleration and $|\omega| \leq 1.1$ is the turn rate.

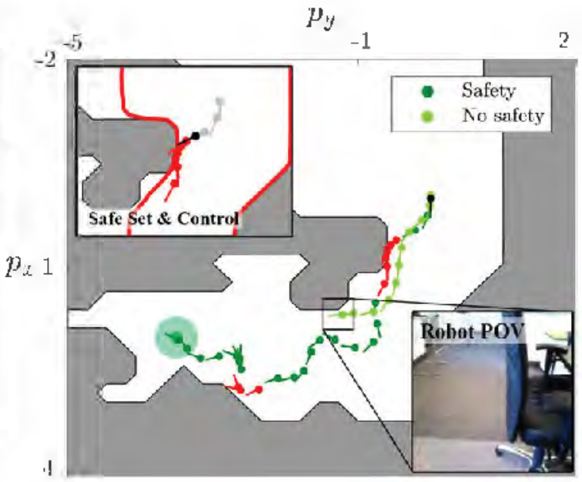
The trajectory taken by the learning-based planner in the absence of the safety module is shown in Figure 9.4a. Even though the vehicle is able to go through the narrow hallway, it collides with the door eventually. The trajectory taken by the vehicle when the planner is combined with the proposed safety framework is shown in Figure 9.4b. When the planner takes an unsafe action near the door, the safety controller intervenes (marked in red) and guides the robot to safely go through the doorway. We also illustrate the image observed by the robot near the doorway in Figure 9.4a. Even though most of the robot's vision is blocked by the door, the planner makes a rather optimistic decision of moving forward and leads to a collision. In contrast, the safety controller makes a conservative decision of rotating in place to explore the environment more before moving forward, and eventually goes through the doorway to reach the goal. The planner-agnostic nature of our framework allows us to provide safety guarantees around learning-based planners as well.

9.5 Hardware Experiments

We test the proposed approach in hardware using a TurtleBot 2 with a mounted stereo RGB camera. For the vehicle state measurement, we use the on-board odometry sensors on the



(a)



(b)

Figure 9.5: (Left) We show an application of our approach on a Turtlebot using a vision-based planner. When the robot is at risk of colliding, the safe controller (u^*) keep the system safe. (Right) We show the top-view of our experiment setting, and the corresponding system trajectories with and without the proposed safety framework. Without the safety framework, the robot collides into the chair. In contrast, our safety framework is able to safely navigate the robot to its goal by intervening when the vehicle is too close to the obstacles.

TurtleBot. For more details on our hardware testbed, please refer to Sec. 7.3.4. Videos of our approach and experiments are available on the project website[1].

In our experiment, the vehicle needs to navigate through an unknown cluttered indoor environment to reach its goal (shown in Figure 9.5a). For the BRS computation, we use the dynamics model in (9.9). We pre-map the environment using an open-source Simultaneous Localization and Mapping (SLAM) algorithm and the on-board stereo camera. This pre-mapping step is used to avoid the significant delay and inaccuracies in the real-time SLAM map update. However, the full map is not provided to the robot during deployment. Instead, for the safe set computation at any given time, the current FoV of the camera is projected on the SLAM map and only the information within the FoV is used to update the safe set. This emulates the limited sensor range of the Turtlebot's camera. Regardless, this alludes to one of the important and interesting future research directions of ensuring safety despite sensor noise.

For planning, we once again use the vision-based planner described in Chapter 7 that uses the current RGB image to determine a candidate desired next state. A top-view of our experiment setting is shown in Figure 9.5b. The vehicle starting position and heading are shown in black, the goal region is shown in green, and the obstacles (unknown to the vehicle beforehand) are shown in grey. We ran the experiment with and without the safety controller and show the corresponding trajectories in Figure 9.5b. Without the safety controller, the learning-based planner struggles with making sharp turns near the corner, and eventually

[1]Project website: https://smlbansal.github.io/website-safe-navigation/

collides into the obstacle (the chair, in this case). For context, we also show the RGB observation received by the planner near the corner. Even though the robot is very close to the chair, the planner makes the unsafe decision of continuing to move forward. However, when the learning-based planner is used within the proposed safety framework, the safety controller is able to account for this unsafe situation and safely steer the vehicle away from the obstacle. We show the corresponding safe set when the vehicle is at the obstacle boundary and the corresponding vehicle trajectory obtained using the safety controller. Afterwards, the planner takes over and steers the vehicle to the goal.

9.6 Chapter Summary

Even though vision-based planners might enable the autonomous systems to navigate in completely new environments, they inevitably make prediction error when encountered with out-of-distribution inputs. In this chapter, we propose an HJ reachability-based safety framework for navigation in unknown environments that is applicable to a wide variety of planners and sensors, including vision-based planners. The proposed framework acts as a monitor for the vision-based planner and provide a corrective safe action whenever necessary, despite the environment uncertainty. We demonstrate our approach to provide safety guarantees around the state-of-the-art vision-based planner developed in Chapter 7 both in simulation and on a hardware testbed.

Chapter 10

Towards a Synergistic Learning and Control Future

It is an exciting and important time in robotics and automation, as robots are slowly but steadily weaving their way from factory floors into our lives. However, ensuring a safe and seamless integration of autonomous systems in our society is a complex problem, requiring significant advances in several research domains. In this dissertation, we discussed how we can combine tools from the areas of control theory, machine learning and computer vision, with the purpose of enabling experience-based scalable, structured and safety-aware reasoning abilities in autonomous systems as they operate in unstructured and dynamic environments under real-time, limited-range perception constraints. Of course, there are still lots of unsolved problems before we can have intelligent and safe controllers for real-world autonomous systems!

Robust integration between perception and control. Learning-based perception components will inevitably make prediction errors. These errors can be caused by imprecise real-world perception sensors that result in incomplete, inaccurate, and intermittent sensor data, or simply when the learning module encounters an out-of-distribution input. From a control perspective, such prediction errors can be treated as sensor errors that affect the feedback controller; therefore, in theory, tools from robust control can be used to design feedback loops that are robust to such errors. However, a key challenge here is the reliable estimation of prediction error that can be used during the controller design process. And even though several sampling-based methods have been developed for uncertainty estimation in the output of deep networks, the uncertainty bounds themselves are not reliable enough for control.

Given the uncertainty, design of a robust controller is not trivial either. Existing methods in robust control typically assume deterministic, worst-case bounds on the uncertainty. For data-driven components, such bounds can be huge for out of distribution samples, rendering the robust control impractical. In such cases, a probabilistic approach can be promising that takes into account the distribution of uncertainty.

Representations for interfacing perception and control. One important question that needs to be addressed to develop frameworks that seamlessly combine perception and control is "what is the right representation to interface perception and control?" For example, the waypoint representation used in this work to interface perception and control for navigation may not be sufficient for grasping and manipulation tasks, such as opening doors or cutting onions. Even for the navigation tasks, it is unclear what a waypoint should consist of. Position of the robot, its orientation, its speed, or a combination of the above? Should this representation change with the autonomous system? For example, should the waypoint representation be different for a wheeled robot vs a walking robot? In general, the representation should be such that it should be easily *learnable* from the perception point of view, yet it should be *sufficient for control* purposes. It might be interesting to take inspiration from humans to attack this problem and understand the kind of representations humans use to solve different tasks flexibly. Another interesting direction could be to *learn* the correct representations between perception and control, as done in end-to-end learning.

Scalable safety analysis of data-driven systems. A constant theme throughout my work has been to develop scalable safety analysis methods for autonomous systems. Even though I believe it is important to develop methods that can leverage the problem structure to achieve scalable safety analysis wherever possible, for learning-enabled systems we will need to go beyond the problem structure and bridge model-based analysis with statistical approaches for safety. Statistical approaches are naturally suitable for learning components that are inherently data-driven and will inevitably fail outside their data distribution, whereas model-based approaches have been very successful for dynamical systems. At the expense of a small probability of failure, a rapprochement between the two has the potential to significantly alleviate the computational complexity of the safety analysis. Our preliminary results in Chapter 8 show that a sampling-based, data-driven approach can be combined with model-based analysis to provide strong probabilistic safety guarantees on the closed-loop system. It will be interesting to further explore these hybrid verification approaches for safety analysis of learning-enabled systems.

Much of the work in this dissertation focuses on how to learn while satisfying safety constraints but less on developing learning approaches for safety. The same power of modern compute and data that is fueling perception can also be leveraged to scale up verification and synthesis. This will, however, require designing "correct-by-construction" learning components to avoid circular reasoning. There are some promising initial results in this direction [34, 39] that are worth investigate further in future.

Closing the loop between design and analysis of learning-enabled systems. During runtime, learning systems might encounter out of distribution samples, which might lead to a high uncertainty in the output of the learning system or even the failure of the overall component. But this begs an important question "how can a learning system use these samples in order to improve itself over time?" For example, the realization that a narrow turn around the corner can lead to a collision with an unobserved obstacle can be used to teach an autonomous car to make a wider turn and first examine the intersection scenario in future. Currently, a popular mechanism to incite this improvement is to collect these out-of-distribution samples and augment them to the training dataset. This approach will at best correct the learning system behavior near the catastrophic failure; however, it does necessarily correct the learning system behavior at the states that led it to this catastrophic scenario in the first place. For instance, in the narrow turn around the corner example, training on out-of-distribution samples will predict a hard stop in future if the car encounters an obstacle after turning; however, it does not necessarily incentivize the car to take a wider turn. To actually enable improvement in the learning systems, we need to close the loop between the analysis and training of a learning system, where failures detected during analysis provide "feedback" for the kind of training data required for a learning system.

Safe online learning-enabled systems. In learning-enabled control feedback loops, if a learning system is being adapted online, its output can change over time. This, in turn, leads to the evolution of the overall control feedback loop. From a control standpoint, it is important to understand which evolutions of the learning system might result in the maximum change in the overall performance of the autonomous system, and more importantly, how to make sure that the overall system remains safe despite these changes in the learning system.

For example, consider a robot navigating in the presence of a pedestrian, whose intent is unknown to the robot. Moreover, suppose that the robot is using an online learning mechanism to infer the intent of the human based on received observations. Based on the inferred intent, the robot predicts the future motion of the pedestrian and plans a path around it. Thus, depending on how the inference of the learning system changes over time, the plan of the robot itself might change significantly. To ensure safety in this scenario, it might be important to figure out all possible changes in the learning system based on likely future observations, and to safeguard against these changes. Our preliminary results indicate that this problem can be formulated as a reachability problem [37]. Possible future human observations are treated as 'input' to a dynamical system, and how the human intent parameters (i.e., learning parameters) change based on the data are the 'dynamics' of the system. Preliminary simulations and experiments indicate that this introspective nature of the human motion predictor with respect to the unobserved data enables safe planning around humans while being resilient to the changes in the data-driven predictive model. In future, it would be interesting to explore this direction further, particularly on how these techniques can be used for active gathering of informative data samples to reduce uncertainty in the learning system.

Bibliography

[1] 3D Robotics. *Solo Specs: Just the facts*. 2015. URL: `https://news.3dr.com/solo-specs-just-the-facts-14480cb55722%5C#.w7057q926`.

[2] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. "Tensorflow: a system for large-scale machine learning." In: *Symposium on operating systems design and implementation*. 2016.

[3] N. Abas, A. Legowo, and R. Akmeliawati. "Parameter identification of an autonomous quadrotor". In: *International Conference On Mechatronics*. 2011.

[4] A. Abate. "A contractivity approach for probabilistic bisimulations of diffusion processes". In: *Conference on Decision and Control*. 2009.

[5] A. Abate and M. Prandini. "Approximate abstractions of stochastic systems: A randomized method". In: *Conference on Decision and Control and European Control Conference*. 2011.

[6] P. Abbeel, A. Coates, and A. Ng. "Autonomous helicopter aerobatics through apprenticeship learning". In: *The International Journal of Robotics Research* (2010).

[7] P. Abbeel, M. Quigley, and A. Ng. "Using inaccurate models in reinforcement learning". In: *International conference on Machine learning*. 2006.

[8] J. Achiam, D. Held, A. Tamar, and P. Abbeel. "Constrained policy optimization". In: *International Conference on Machine Learning*. 2017.

[9] D. Adalsteinsson and J. A. Sethian. "A fast level set method for propagating interfaces". In: *Journal of computational physics* 118.2 (1995), pp. 269–277.

[10] A. Agrawal and K. Sreenath. "Discrete Control Barrier Functions for Safety-Critical Control of Discrete Systems with Application to Bipedal Robot Navigation." In: *Robotics: Science and Systems*. 2017.

[11] P. Agrawal, A. Nair, P. Abbeel, J. Malik, and S. Levine. "Learning to poke by poking: Experiential learning of intuitive physics". In: *Advances in Neural Information Processing Systems*. 2016.

[12] A. Ahmadzadeh, N. Motee, A. Jadbabaie, and G. Pappas. "Multi-vehicle path planning in dynamically changing environments". In: *International Conference on Robotics and Automation*. 2009.

[13] A. K. Akametalu, S. Ghosh, J. F. Fisac, and C. J. Tomlin. “A Minimum Discounted Reward Hamilton-Jacobi Formulation for Computing Reachable Sets”. In: *IEEE Transactions on Automatic Control* (2018).

[14] F. Alhwarin, A. Ferrein, and I. Scholl. “IR stereo kinect: improving depth images by combining structured light with IR stereo”. In: *Pacific Rim International Conference on Artificial Intelligence*. 2014.

[15] M. Althoff. “An Introduction to CORA 2015”. In: *Proc. ARCH@ CPSWeek*. 2015.

[16] M. Althoff. “Formal and compositional analysis of power systems using reachable sets”. In: *IEEE Transactions on Power Systems* 29.5 (2014), pp. 2270–2280.

[17] M. Althoff and J. Dolan. “Set-based computation of vehicle behaviors for the online verification of autonomous vehicles”. In: *Conference on Intelligent Transportation Systems*. 2011.

[18] M. Althoff, O. Stursberg, and M. Buss. “Computing reachable sets of hybrid systems using a combination of zonotopes and polytopes”. In: *Nonlinear analysis: hybrid systems* 4.2 (2010), pp. 233–249.

[19] R. Alur, T. A. Henzinger, G. Lafferriere, and G. J. Pappas. “Discrete abstractions of hybrid systems”. In: *Proceedings of the IEEE*. 2000.

[20] Amazon.com, Inc. *Amazon Prime Air*. 2016. URL: `http://www.amazon.com/b?node=8037720011`.

[21] A. D. Ames, X. Xu, J. W. Grizzle, and P. Tabuada. “Control barrier function based quadratic programs for safety critical systems”. In: *IEEE Transactions on Automatic Control* 62.8 (2017), pp. 3861–3876.

[22] A. Amini, G. Rosman, S. Karaman, and D. Rus. “Variational End-to-End Navigation and Localization”. In: *arXiv preprint arXiv:1811.10119* (2018).

[23] I. Armeni, O. Sener, A. R. Zamir, H. Jiang, I. Brilakis, M. Fischer, and S. Savarese. “3D Semantic Parsing of Large-Scale Indoor Spaces”. In: *Conference on Computer Vision and Pattern Recognition*. 2016.

[24] K. Åström and B. Wittenmark. *Adaptive control*. Courier Corporation, 2013.

[25] A. Aswani, H. Gonzalez, S. Sastry, and C. Tomlin. “Provably safe and robust learning-based model predictive control”. In: *Automatica* 49.5 (2013), pp. 1216–1226.

[26] C. Atkeson. “Nonparametric model-based reinforcement learning”. In: *Advances in neural information processing systems*. 1998.

[27] AUVSI News. *UAS Aid in South Carolina Tornado Investigation*. 2016. URL: `http://www.auvsi.org/blogs/auvsi-news/2016/01/29/tornado`.

[28] C. Baier, J. Katoen, and K. Larsen. *Principles of Model Checking*. Cambridge, MA: MIT Press, 2008.

[29] A. Bajcsy, S. Bansal, E. Bronstein, V. Tolani, and C. Tomlin. "An Efficient Reachability-Based Framework for Provably Safe Autonomous Navigation in Unknown Environments". In: *Conference on Decision and Control*. 2019.

[30] S. Bansal, A. Akametalu, F. Jiang F.and Laine, and C. Tomlin. "Learning quadrotor dynamics using neural network for flight control". In: *Conference on Decision and Control*. 2016.

[31] S. Bansal, R. Calandra, S. Levine, and C. Tomlin. "MBMF: Model-Based Priors for Model-Free Reinforcement Learning". In: *arXiv preprint arXiv:1709.03153* (2017).

[32] S. Bansal, R. Calandra, T. Xiao, S. Levine, and C. Tomlin. "Goal-Driven Dynamics Learning via Bayesian Optimization". In: *Conference on Decision and Control*. 2017.

[33] S. Bansal, M. Chen, J. Fisac, and C. Tomlin. "Safe Sequential Path Planning of Multi-Vehicle Systems Under Presence of Disturbances and Imperfect Information". In: *American Control Conference*. 2017.

[34] S. Bansal, M. Chen, S. Herbert, and C. Tomlin. "Hamilton-Jacobi Reachability: A Brief Overview and Recent Advances". In: *Conference on Decision and Control*. 2017.

[35] S. Bansal, M. Chen, K. T., and C. Tomlin. "Provably Safe and Scalable Multi-Vehicle Trajectory Planning". In: *IEEE Transactions on Control Systems Technology* (2020).

[36] S. Bansal, M. Chen, and C. Tomlin. "Safe and Resilient Multi-vehicle Trajectory Planning Under Adversarial Intruder". In: *arXiv preprint arXiv:1711.02540* (2017).

[37] S. Bansal, A. Bajcsy, E. Ratner, A. D. Dragan, and C. J. Tomlin. "A Hamilton-Jacobi reachability-based framework for predicting and analyzing human motion for safe planning". In: *International Conference on Robotics and Automation* (2020).

[38] S. Bansal, V. Tolani, S. Gupta, J. Malik, and C. Tomlin. "Combining optimal control and learning for visual navigation in novel environments". In: *Conference on Robot Learning* (2019).

[39] S. Bansal and C. Tomlin. "DeepReach: A Deep Learning Approach to High-Dimensional Reachability". In: *arXiv preprint arXiv:2011.02082* (2020).

[40] E. Barron. "Differential games with maximum cost". In: *Nonlinear analysis: Theory, methods & applications* 14.11 (1990), pp. 971–989.

[41] A. Barry, A. Majumdar, and R. Tedrake. "Safety verification of reactive controllers for UAV flight in cluttered environments using barrier certificates". In: *International Conference on Robotics and Automation*. 2012.

[42] T. Başar and G. J. Olsder. *Dynamic noncooperative game theory*. SIAM, 1998.

[43] A. Bayen, I. Mitchell, M. Osihi, and C. Tomlin. "Aircraft Autolander Safety Analysis Through Optimal Control-Based Reach Set Computation". In: *Journal of Guidance, Control, and Dynamics* 30.1 (2007), pp. 68–77.

[44] BBC Technology. *Google plans drone delivery service for 2017.* 2016. URL: `http://www.bbc.com/news/technology-34704868`.

[45] R. Beard. "Quadrotor dynamics and control". In: *Brigham Young University* 19.3 (2008), pp. 46–56.

[46] R. Beard and T. McLain. "Multiple UAV cooperative search under collision avoidance and limited range communication constraints". In: *Conference on Decision and Control.* 2003.

[47] T. Beckers, S. Bansal, C. Tomlin, and S. Hirche. "Closed-loop model selection for kernel-based models using Bayesian optimization". In: *Conference on Decision and Control.* 2019.

[48] T. Beckers and S. Hirche. "Equilibrium distributions and stability analysis of Gaussian Process State Space Models". In: *Conference on Decision and Control.* 2016.

[49] T. Beckers, D. Kulić, and S. Hirche. "Stable Gaussian Process based Tracking Control of Euler-Lagrange Systems". In: *Automatica* 103 (2019), pp. 390–397.

[50] K. E. Bekris and L. E. Kavraki. "Greedy but safe replanning under kinodynamic constraints". In: *International Conference on Robotics and Automation.* 2007.

[51] J. Bellingham, M. Tillerson, M. Alighanbari, and J. How. "Cooperative path planning for multiple UAVs in dynamic and uncertain environments". In: *Conference on Decision and Control.* 2002.

[52] R. Bellman. "On the theory of dynamic programming". In: *Proceedings of the National Academy of Sciences of the United States of America* 38.8 (1952), p. 716.

[53] C. Belta, B. Yordanov, and E. Gol. *Formal Methods for Discrete-Time Dynamical Systems.* Vol. 89. Studies in Systems, Decision and Control. Springer International Publishing, 2017.

[54] D. Bender and A. Laub. "The linear-quadratic optimal regulator for descriptor systems: discrete-time case". In: *Automatica* (1987).

[55] F. Berkenkamp, M. Turchetta, A. Schoellig, and A. Krause. "Safe model-based reinforcement learning with stability guarantees". In: *Conference on Neural Information Processing Systems* (2017).

[56] C. Bishop. *Pattern recognition and machine learning.* Vol. 4. Springer New York, 2006.

[57] O. Bokanowski and H. Zidani. "Minimal Time Problems With Moving Targets and Obstacles". In: *IFAC Proceedings Volumes* 44.1 (2011), pp. 2589–2593.

[58] M. Bolton, E. Bass, and R. Siminiceanu. "Using Formal Verification to Evaluate Human-Automation Interaction: A Review". In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 43.3 (2013), pp. 488–503.

[59] P. Bouffard. “On-board Model Predictive Control of a Quadrotor Helicopter: Design, Implementation, and Experiments”. MA thesis. University of California, Berkeley, 2012.

[60] S. L. Bowman, N. Atanasov, K. Daniilidis, and G. J. Pappas. “Probabilistic data association for semantic slam”. In: *International Conference on Robotics and Automation*. 2017.

[61] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. *OpenAI Gym*. 2016. eprint: `arXiv:1606.01540`.

[62] A. Brunetti, D. Buongiorno, G. F. Trotta, and V. Bevilacqua. “Computer vision and deep learning techniques for pedestrian detection and tracking: A survey”. In: *Neurocomputing* (2018).

[63] M. L. Bujorianu, J. Lygeros, and M. C. Bujorianu. “Bisimulation for general stochastic hybrid systems”. In: *International Workshop on Hybrid Systems: Computation and Control*. 2005.

[64] A. Bull. “Convergence rates of efficient global optimization algorithms”. In: *Journal of Machine Learning Research* 12.Oct (2011), pp. 2879–2904.

[65] G. C. Calafiore and M. C. Campi. “The scenario approach to robust control design”. In: *IEEE Transactions on Automatic Control*. 2006.

[66] G. C. Calafiore and M. C. Campi. “Uncertain convex programs: randomized solutions and confidence levels”. In: *Mathematical Programming*. 2005.

[67] R. Calandra, A. Seyfarth, J. Peters, and M. Deisenroth. “Bayesian Optimization for Learning Gaits under Uncertainty”. In: *Annals of Mathematics and Artificial Intelligence* 76.1 (2015), pp. 5–23.

[68] M. C. Campi and S. Garatti. “A sampling-and-discarding approach to chance-constrained optimization: feasibility and optimality”. In: *Journal of Optimization Theory and Applications*. 2011.

[69] M. C. Campi, S. Garatti, and M. Prandini. “The scenario approach for systems and control design”. In: *Annual Reviews in Control*. 2009.

[70] Y. Chebotar, K. Hausman, M. Zhang, G. Sukhatme, S. Schaal, and S. Levine. “Combining Model-Based and Model-Free Updates for Trajectory-Centric Reinforcement Learning”. In: *arXiv preprint arXiv:1703.03078* (2017).

[71] C. Chen, Y. Liu, S. Kreiss, and A. Alahi. “Crowd-robot interaction: Crowd-aware robot navigation with attention-based deep reinforcement learning”. In: *International Conference on Robotics and Automation*. 2019.

[72] K. Chen, J. P. de Vicente, G. Sepulveda, F. Xia, A. Soto, M. Vazquez, and S. Savarese. “A behavioral approach to visual navigation with graph localization networks”. In: *Robotics: Science and Systems*. 2019.

[73] M. Chen, S. Bansal, J. Fisac, and C. Tomlin. "Robust Sequential Path Planning Under Disturbances and Adversarial Intruder". In: *IEEE Transactions on Control Systems Technology* (2017).

[74] M. Chen, S. Herbert, and C. Tomlin. "Exact and efficient Hamilton-Jacobi-based guaranteed safety analysis via system decomposition". In: *International Conference on Robotics and Automation* (2017).

[75] M. Chen, S. Herbert, M. Vashishtha, S. Bansal, and C. Tomlin. "Decomposition of reachable sets and tubes for a class of nonlinear systems". In: *IEEE Transactions on Automatic Control* 63.11 (2018), pp. 3675–3688.

[76] M. Chen, Q. Hu, J. Fisac, K. Akametalu, C. Mackin, and C. Tomlin. "Reachability-Based Safety and Goal Satisfaction of Unmanned Aerial Platoons on Air Highways". In: *Journal of Guidance, Control, and Dynamics* (2017), pp. 1–14.

[77] M. Chen, Q. Hu, C. Mackin, J. Fisac, and C. Tomlin. "Safe platooning of unmanned aerial vehicles via reachability". In: *Conference on Decision and Control*. 2015.

[78] M. Chen, J. Shih, and C. Tomlin. "Multi-Vehicle Collision Avoidance via Hamilton-Jacobi Reachability and Mixed Integer Programming". In: *Conference on Decision and Control*. 2016.

[79] X. Chen, E. Ábrahám, and S. Sankaranarayanan. "Flow*: An Analyzer for Non-linear Hybrid Systems". In: *International Conference on Computer Aided Verification*. 2013.

[80] H.-T. L. Chiang, A. Faust, M. Fiser, and A. Francis. "Learning navigation behaviors end-to-end with AutoRL". In: *IEEE Robotics and Automation Letters* 4.2 (2019), pp. 2007–2014.

[81] Y. Chow, O. Nachum, E. Duenez-Guzman, and M. Ghavamzadeh. "A lyapunov-based approach to safe reinforcement learning". In: *Conference on Neural Information Processing Systems*. 2018.

[82] K. Chua, R. Calandra R.and McAllister, and S. Levine. "Deep reinforcement learning in a handful of trials using probabilistic dynamics models". In: *Advances in Neural Information Processing Systems*. 2018.

[83] Y. Chuang, Y. Huang, M. D'Orsogna, and A. Bertozzi. "Multi-vehicle flocking: Scalability of cooperative control algorithms using pairwise potentials". In: *International Conference on Robotics and Automation*. 2007.

[84] D. Clarke, P. Kanjilal, and C. Mohtadi. "A generalized LQG approach to self-tuning control part i. aspects of design". In: *International Journal of Control* 41.6 (1985), pp. 1509–1523.

[85] E. Coddington and N. Levinson. *Theory of ordinary differential equations*. Tata McGraw-Hill Education, 1955.

[86] S. Coogan and M. Arcak. "Efficient finite abstraction of mixed monotone systems". In: *International Conference on Hybrid Systems: Computation and Control*. 2015.

[87] S. Coogan, M. Arcak, and C. Belta. "Formal Methods for Control of Traffic Flow: Automated Control Synthesis from Finite-State Transition Models". In: *IEEE Control Systems* 37.2 (2017), pp. 109–128.

[88] M. Crandall and P. Lions. "Viscosity solutions of Hamilton-Jacobi equations". In: *Transactions of the American mathematical society* 277.1 (1983), pp. 1–42.

[89] C. Dabadie, S. Kaynama, and C. Tomlin. "A practical reachability-based collision avoidance algorithm for sampled-data systems: Application to ground robots". In: *Conference on Intelligent Robots and Systems*. 2014.

[90] J. Darbon and S. Osher. "Algorithms for overcoming the curse of dimensionality for certain Hamilton–Jacobi equations arising in control theory and elsewhere". In: *Research in the Mathematical Sciences* 3.1 (2016), p. 19.

[91] T. Dean and R. Givan. "Model minimization in Markov decision processes". In: *AAAI/IAAI*. 1997.

[92] W. DeBusk. "Unmanned Aerial Vehicle Systems for Disaster Relief: Tornado Alley". In: *Infotech@ Aerospace Conferences*. 2010.

[93] M. Deisenroth and C. Rasmussen. "PILCO: A model-based and data-efficient approach to policy search". In: *International Conference on Machine Learning*. 2011.

[94] M. Deisenroth, D. Fox, and C. Rasmussen. "Gaussian Processes for Data-Efficient Learning in Robotics and Control". In: *Transactions on Pattern Analysis and Machine Intelligence* (2015).

[95] A. Desai, S. Ghosh, S. A. Seshia, N. Shankar, and A. Tiwari. "SOTER: A Runtime Assurance Framework for Programming Safe Robotics Systems". In: *International Conference on Dependable Systems and Networks*. 2019.

[96] J. Desharnais, A. Edalat, and P. Panangaden. "Bisimulation for labelled Markov processes". In: *Information and Computation*. 2002.

[97] J. Ding, J. Sprinkle, S. Sastry, and C. Tomlin. "Reachability calculations for automated aerial refueling". In: *Conference on Decision and Control*. 2008.

[98] P. Donti, B. Amos, and J. Z. Kolter. "Task-based End-to-end Model Learning". In: *arXiv preprint arXiv:1703.04529* (2017).

[99] T. Dreossi, T. Dang, and C. Piazza. "Parallelotope Bundles for Polynomial Reachability". In: *Conference on Hybrid Systems: Computation and Control*. 2016.

[100] T. Dreossi, A. Donzé, and S. A. Seshia. "Compositional falsification of cyber-physical systems with machine learning components". In: *Journal of Automated Reasoning* 63.4 (2019), pp. 1031–1053.

[101] T. Dreossi, D. J. Fremont, S. Ghosh, E. Kim, H. Ravanbakhsh, M. Vazquez-Chanlatte, and S. A. Seshia. "Verifai: A toolkit for the formal design and analysis of artificial intelligence-based systems". In: *International Conference on Computer Aided Verification*. 2019.

[102] P. Drews, G. Williams, B. Goldfain, E. A. Theodorou, and J. M. Rehg. “Aggressive Deep Driving: Combining Convolutional Neural Networks and Model Predictive Control”. In: *Conference on Robot Learning*. 2017.

[103] P. Drews, G. Williams, B. Goldfain, E. A. Theodorou, and J. M. Rehg. “Vision-Based High-Speed Driving With a Deep Dynamic Observer”. In: *IEEE Robotics and Automation Letters* (2019).

[104] P. Duggirala, C. Fan, M. Potok, B. Qi, S. Mitra, M. Viswanathan, S. Bak, S. Bogomolov, T. Johnson, L. Nguyen, et al. “Tutorial: Software tools for hybrid systems verification, transformation, and synthesis: C2E2, HyST, and TuLiP”. In: *Conference on Control Applications*. 2016.

[105] P. Duggirala, S. Mitra, M. Viswanathan, and M. Potok. “C2E2: A Verification Tool for Stateflow Models”. In: *Conference on Tools and Algorithms for the Construction and Analysis of Systems*. 2015.

[106] L. Evans. *Partial differential equations*. American Mathematical Society, 2010.

[107] M. Everett, Y. F. Chen, and J. P. How. “Motion planning among dynamic, decision-making agents with deep reinforcement learning”. In: *Conference on Intelligent Robots and Systems*. 2018.

[108] G. Fainekos, A. Girard, H. Kress-Gazit, and G. Pappas. “Temporal logic motion planning for dynamic robots”. In: *Automatica* 45.2 (2009), pp. 343–352.

[109] C. Fan, B. Qi, S. Mitra, M. Viswanathan, and P. Duggirala. “Automatic Reachability Analysis for Nonlinear Hybrid Models with C2E2”. In: *International Conference on Computer Aided Verification*. 2016.

[110] T. Fan, X. Cheng, J. Pan, P. Long, W. Liu, R. Yang, and D. Manocha. “Getting Robots Unfrozen and Unlost in Dense Pedestrian Crowds”. In: *IEEE Robotics and Automation Letters* (2019).

[111] F. Farshidian, M. Neunert, and J. Buchli. “Learning of closed-loop motion control”. In: *Conference on Intelligent Robots and Systems*. 2014.

[112] A. Faust, O. Ramirez, M. Fiser, K. Oslund, A. Francis, J. Davidson, and L. Tapia. “PRM-RL: Long-range Robotic Navigation Tasks by Combining Reinforcement Learning and Sampling-based Planning”. In: *International Conference on Robotics and Automation*. 2018.

[113] P. Fiorini and Z. Shiller. “Motion Planning in Dynamic Environments Using Velocity Obstacles”. In: *International Journal of Robotics Research* 17.7 (July 1998), pp. 760–772.

[114] J. Fisac, A. Akametalu, M. Zeilinger, S. Kaynama, J. Gillula, and C. J. Tomlin. “A general safety framework for learning-based control in uncertain robotic systems”. In: *IEEE Transactions on Automatic Control* (2018).

[115] J. Fisac, M. Chen, C. Tomlin, and S. Sastry. “Reach-avoid problems with time-varying dynamics, targets and constraints”. In: *Conference on Hybrid Systems: Computation and Control*. 2015.

[116] J. Fisac, N. Lugovoy, V. Rubies-Royo, S. Ghosh, and C. Tomlin. “Bridging Hamilton-Jacobi Safety Analysis and Reinforcement Learning”. In: *International Conference on Robotics and Automation* (2019).

[117] J. F. Fisac, A. Bajcsy, S. L. Herbert, D. Fridovich-Keil, S. Wang, C. J. Tomlin, and A. D. Dragan. “Probabilistically safe robot planning with confidence-based human predictions”. In: *Robotics: Science and Systems* (2018).

[118] T. Fraichard and H. Asama. “Inevitable collision states-A step towards safer robots?” In: *Advanced Robotics* 18.10 (2004), pp. 1001–1024.

[119] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler. “SpaceEx: Scalable Verification of Hybrid Systems”. In: *International Conference on Computer Aided Verification*. 2011.

[120] D. J. Fremont, T. Dreossi, S. Ghosh, X. Yue, A. L. Sangiovanni-Vincentelli, and S. A. Seshia. “Scenic: a language for scenario specification and scene generation”. In: *Conference on Programming Language Design and Implementation*. 2019.

[121] J. Fu, S. Levine, and P. Abbeel. “One-shot learning of manipulation skills with online dynamics adaptation and neural network priors”. In: *arXiv preprint arXiv:1509.06841* (2015).

[122] J. Fuentes-Pacheco, J. Ruiz-Ascencio, and J. M. Rendón-Mancha. “Visual simultaneous localization and mapping: a survey”. In: *Artificial Intelligence Review* (2015).

[123] J. Gablonsky. “Modifications of the DIRECT Algorithm.” In: (2001).

[124] Y. Gal, R. McAllister, and C. Rasmussen. “Improving PILCO with Bayesian neural network dynamics models”. In: *Data-Efficient Machine Learning workshop, ICML*. 2016.

[125] D. Gandhi, L. Pinto, and A. Gupta. “Learning to fly by crashing”. In: *Conference on Intelligent Robots and Systems*. 2017.

[126] W. Gao, D. Hsu, W. S. Lee, S. Shen, and K. Subramanian. “Intention-net: Integrating planning and deep learning for goal-directed autonomous navigation”. In: *Conference on Robot Learning*. 2017.

[127] S. Garatti and M. Prandini. “A simulation-based approach to the approximation of stochastic hybrid systems”. In: *Analysis and design of hybrid systems*. 2012.

[128] E. Garrido-Merchán and D. Hernández-Lobato. “Dealing with Integer-valued Variables in Bayesian Optimization with Gaussian Processes”. In: *arXiv preprint arXiv:1706.03673* (2017).

[129] M. Gevers. “Identification for control: From the early achievements to the revival of experiment design”. In: *European journal of control* 11.4-5 (2005).

[130] M. Gevers, X. Bombois, B. Codrons, G. Scorletti, and B. D. O. Anderson. "Model Validation for Control and Controller Validation in a Prediction Error Identification framework-Part I: Theory". In: *Automatica*. 2003.

[131] S. Ghosh, S. Bansal, A. Sangiovanni-Vincentelli, S. A. Seshia, and C. Tomlin. "A new simulation metric to determine safe environments and controllers for systems with unknown dynamics". In: *International Conference on Hybrid Systems: Computation and Control*. 2019.

[132] S. Ghosh, H. Ravanbakhsh, and S. A. Seshia. *Counterexample-Guided Synthesis of Perception Models and Control*. 2019. arXiv: `1911.01523 [eess.SY]`.

[133] A. Giese, D. Latypov, and N. Amato. "Reciprocally-Rotating Velocity Obstacles". In: *International Conference on Robotics and Automation*. 2014.

[134] J. Gillula, G. Hoffmann, H. Huang, M. Vitus, and C. Tomlin. "Applications of hybrid reachability analysis to robotic aerial vehicles". In: *The International Journal of Robotics Research* 30.3 (2011), pp. 335–354.

[135] A. Girard. "Reachability of uncertain linear systems using zonotopes". In: *International Workshop on Hybrid Systems: Computation and Control*. 2005.

[136] A. Girard and G. J. Pappas. "Approximate bisimulation relations for constrained linear systems". In: *Automatica*. 2007.

[137] A. Girard and G. J. Pappas. "Approximate Bisimulation: A bridge between computer science and control theory". In: *European Journal of Control*. 2011.

[138] A. Girard, G. Pola, and P. Tabuada. "Approximately bisimilar symbolic models for incrementally stable switched systems". In: *IEEE Transactions on Automatic Control*. 2010.

[139] D. González, J. Pérez, V. Milanés, and F. Nashashibi. "A Review of Motion Planning Techniques for Automated Vehicles." In: *IEEE Transactions on Intelligent Transportation Systems* (2016).

[140] GPy. *GPy: A Gaussian process framework in python*. `http://github.com/SheffieldML/GPy`. since 2012.

[141] M. Greenstreet and I. Mitchell. "Integrating Projections". In: *Workshop on Hybrid Systems: Computation and Control*. 1998.

[142] M. Grimble. "Implicit and explicit LQG self-tuning controllers". In: *Automatica* 20.5 (1984), pp. 661–669.

[143] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine. "Continuous deep Q-learning with model-based acceleration". In: *arXiv preprint arXiv:1603.00748* (2016).

[144] S. Gupta, V. Tolani, J. Davidson, S. Levine, R. Sukthankar, and J. Malik. "Cognitive Mapping and Planning for Visual Navigation". In: *arXiv preprint arXiv:1702.03920* (2017).

[145] M. Hafner and D. Del Vecchio. “Computation of safety control for uncertain piecewise continuous systems on a partial order”. In: *Conference on Decision and Control held jointly with Chinese Control Conference*. 2009.

[146] K. He, X. Zhang, S. Ren, and J. Sun. “Deep residual learning for image recognition”. In: *Conference on Computer Vision and Pattern Recognition*. 2016.

[147] N. Heess, G. Wayne, D. Silver, T. Lillicrap, T. Erez, and Y. Tassa. “Learning continuous control policies by stochastic value gradients”. In: *Advances in Neural Information Processing Systems*. 2015.

[148] M. Hehn and R. D’Andrea. “An iterative learning scheme for high performance, periodic quadrocopter trajectories”. In: *European Control Conference*. 2013.

[149] D. Henrion and M. Korda. “Convex computation of the region of attraction of polynomial control systems”. In: *IEEE Transactions on Automatic Control* 59.2 (2014), pp. 297–312.

[150] S. Herbert, S. Bansal, S. Ghosh, and C. Tomlin. “Reachability-Based Safety Guarantees using Efficient Initializations”. In: *Conference on Decision and Control*. 2019.

[151] S. Herbert, M. Chen, S. Han, S. Bansal, J. Fisac, and C. Tomlin. “FaSTrack: a Modular Framework for Fast and Guaranteed Safe Motion Planning”. In: *Conference on Decision and Control*. 2017.

[152] H. Hjalmarsson, M. Gevers, and F. De Bruyne. “For model-based control design, closed-loop identification gives better performance”. In: *Automatica* 32.12 (1996).

[153] H. Hjalmarsson and L. Ljung. “Estimating model variance in the case of undermodeling”. In: *IEEE Transactions on Automatic Control*. 1992.

[154] W. Hoenig, C. Milanes, L. Scaria, T. Phan, M. Bolas, and N. Ayanian. “Mixed Reality for Robotics”. In: *IEEE/RSJ Intl Conf. Intelligent Robots and Systems*. Sept. 2015, pp. 5382–5387.

[155] G. Hoffmann, H. Huang, S. Waslander, and C. Tomlin. “Quadrotor helicopter flight dynamics and control: Theory and experiment”. In: *AIAA Guidance, Navigation, and Control Conference*. 2007.

[156] G. Hoffmann and C. Tomlin. “Decentralized cooperative collision avoidance for acceleration constrained vehicles”. In: *Conference on Decision and Control*. 2008.

[157] W. Honig, C. Milanes, L. Scaria, T. Phan, M. Bolas, and N. Ayanian. “Mixed reality for robotics”. In: *Conference on Intelligent Robots and Systems*. 2015.

[158] H. Huang, J. Ding, W. Zhang, and C. Tomlin. “A differential game approach to planning in adversarial scenarios: A case study on capture-the-flag”. In: *International Conference on Robotics and Automation*. 2011.

[159] L. Janson, T. Hu, and M. Pavone. “Safe Motion Planning in Unknown Environments: Optimality Benchmarks and Tractable Policies”. In: *arXiv preprint* (2018).

[160] S. Jha, V. Raman, D. Sadigh, and S. Seshia. “Safe Autonomy Under Perception Uncertainty Using Chance-Constrained Temporal Logic”. In: *Journal of Automated Reasoning* (2017).

[161] Joint Planning and Development Office. *Unmanned Aircraft Systems (UAS) Comprehensive Plan*. Tech. rep. Federal Aviation Administration, 2014.

[162] J. Joseph, A. Geramifard, J. Roberts, J. How, and N. Roy. “Reinforcement learning with misspecified model classes”. In: *International Conference on Robotics and Automation*. 2013.

[163] A. A. Julius and G. J. Pappas. “Approximations of stochastic hybrid systems”. In: *IEEE Transactions on Automatic Control*. 2009.

[164] S. Jung, S. Hwang, H. Shin, and D. H. Shim. “Perception, guidance, and navigation for indoor autonomous drone racing using deep learning”. In: *IEEE Robotics and Automation Letters* (2018).

[165] G. Kahn, A. Villaflor, V. Pong, P. Abbeel, and S. Levine. “Uncertainty-aware reinforcement learning for collision avoidance”. In: *arXiv preprint arXiv:1702.01182* (2017).

[166] K. Kang, S. Belkhale, G. Kahn, P. Abbeel, and S. Levine. “Generalization through Simulation: Integrating Simulated and Real Data into Deep Reinforcement Learning for Vision-Based Autonomous Flight”. In: *arXiv preprint arXiv:1902.03701* (2019).

[167] J. P. Katoen, T. Kemna, I. Zapreev, and D. N. Jansen. “Bisimulation minimisation mostly speeds up probabilistic model checking”. In: *International Conference on tools and algorithms for the construction and analysis of systems*. 2007.

[168] E. Kaufmann, M. Gehrig, P. Foehn, R. Ranftl, A. Dosovitskiy, V. Koltun, and D. Scaramuzza. “Beauty and the Beast: Optimal Methods Meet Learning for Drone Racing”. In: *International Conference on Robotics and Automation*. 2019.

[169] E. Kaufmann, A. Loquercio, R. Ranftl, A. Dosovitskiy, V. Koltun, and D. Scaramuzza. “Deep drone racing: Learning agile flight in dynamic environments”. In: *Conference on Robot Learning*. 2018.

[170] S. Kaynama and M. Oishi. “A Modified Riccati Transformation for Decentralized Computation of the Viability Kernel Under LTI Dynamics”. In: *IEEE Transactions on Automatic Control* 58.11 (2013), pp. 2878–2892.

[171] S. Kaynama and M. Oishi. “Schur-based decomposition for reachability analysis of linear time-invariant systems”. In: *Conference on Decision and Control held jointly with Chinese Control Conference*. 2009.

[172] D. Keil, J. Fisac, and C. J. Tomlin. “Safe and Complete Real-Time Planning and Exploration in Unknown Environments”. In: *arXiv preprint* (2018).

[173] A. Khan, C. Zhang, N. Atanasov, K. Karydis, V. Kumar, and D. D. Lee. “Memory Augmented Control Networks”. In: *International Conference on Learning Representations*. 2018.

[174] D. K. Kim and T. Chen. “Deep neural network for real-time autonomous indoor navigation”. In: *arXiv preprint arXiv:1511.04668* (2015).

[175] S. Koenig and M. Likhachev. “Fast replanning for navigation in unknown terrain”. In: *IEEE Transactions on Robotics* 21.3 (2005), pp. 354–363.

[176] S. Kong, S. Gao, W. Chen, and E. Clarke. “dReach: δ-Reachability Analysis for Hybrid Systems”. In: *Conference on Tools and Algorithms for the Construction and Analysis of Systems*. 2015.

[177] T. J. Koo and S. Sastry. “Differential flatness based full authority helicopter control design”. In: *Conference on Decision and Control*. 1999.

[178] S. Kousik, S. Vaskov, F. Bu, M. J. Roberson, and R. Vasudevan. “Bridging the Gap Between Safety and Real-Time Performance in Receding-Horizon Trajectory Design for Mobile Robots”. In: *arXiv preprint* (2018).

[179] A. Krizhevsky, I. Sutskever, and G. Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Advances in neural information processing systems*. 2012.

[180] B. Kuipers and Y.-T. Byun. “A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations”. In: *Robotics and autonomous systems* (1991).

[181] A. Kupcsik, M. Deisenroth, J. Peters, A. Loh, P. Vadakkepat, and G. Neumann. “Model-based contextual policy search for data-efficient generalization of robot skills”. In: *Artificial Intelligence* (2014).

[182] A. Kurzhanski and P. Varaiya. “Ellipsoidal techniques for reachability analysis: internal approximation”. In: *Systems & control letters* 41.3 (2000), pp. 201–211. DOI: `10.1016/S0167-6911(00)00059-1`.

[183] A. Kurzhanski and P. Varaiya. “On Ellipsoidal Techniques for Reachability Analysis. Part II: Internal Approximations Box-valued Constraints”. In: *Optimization Methods and Software* 17.2 (2002), pp. 207–237.

[184] H. Kushner. “A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise”. In: *Journal of Basic Engineering* 86 (1964), p. 97.

[185] M. Labbé and F. Michaud. “RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation”. In: *Journal of Field Robotics* (2019).

[186] M. Lahijanian, M. R. Maly, D. Fried, L. E. Kavraki, H. Kress-Gazit, and M. Vardi. “Iterative Temporal Planning in Uncertain Environments with Partial Satisfaction Guarantees”. In: *IEEE Transactions on Robotics* 32 (2016), pp. 583–599.

[187] E. Lalish, K. Morgansen, and T. Tsukamaki. “Decentralized reactive collision avoidance for multiple unicycle-type vehicles”. In: *American Control Conference*. 2008.

[188] B. Landry. “Planning and control for quadrotor flight through cluttered environments”. MA thesis. Massachusetts Institute of Technology, 2015.

[189] K. G. Larsen and A. Skou. “Bisimulation through probabilistic testing”. In: *Information and computation*. 1991.

[190] S. LaValle. *Planning algorithms*. Cambridge university press, 2006.

[191] S. M. LaValle. “Rapidly-exploring random trees: A new tool for path planning”. In: (1998).

[192] Y. LeCun, Y. Bengio, and G. Hinton. “Deep learning”. In: *Nature* 521.7553 (2015), pp. 436–444.

[193] I. Lenz, R. Knepper, and A. Saxena. “DeepMPC: Learning deep latent features for model predictive control.” In: *Robotics: Science and Systems*. 2015.

[194] S. Levine, C. Finn, T. Darrell, and P. Abbeel. “End-to-end training of deep visuomotor policies”. In: *Journal of Machine Learning Research* (2016).

[195] A. Li, S. Bansal, G. Giovanis, V. Tolani, C. Tomlin, and M. Chen. “Generating Robust Supervision for Learning-Based Visual Navigation Using Hamilton-Jacobi Reachability”. In: *Proceedings of Machine Learning Research* 1 (2020), p. 11.

[196] M. Li, R. Jiang, S. S. Ge, and T. H. Lee. “Role playing learning for socially concomitant mobile robot navigation”. In: *CAAI Transactions on Intelligence Technology* (2018).

[197] F. Lian and R. Murray. “Real-time trajectory generation for the cooperative path planning of multi-vehicle systems”. In: *Conference on Decision and Control*. 2002.

[198] Y. Lin and S. Saripalli. “Collision avoidance for UAVs using reachable sets”. In: *Conference on Unmanned Aircraft Systems*. 2015.

[199] L. Ljung. “System identification”. In: *Wiley encyclopedia of electrical and electronics engineering* (1999), pp. 1–19.

[200] L. Ljung. *System identification: theory for the user*. 1987.

[201] J. Lofberg. “YALMIP: A toolbox for modeling and optimization in MATLAB”. In: *International Conference on Robotics and Automation*. 2004.

[202] J. Löfberg. *Minimax approaches to robust model predictive control*. Vol. 812. Linköping University Electronic Press, 2003.

[203] A. Loquercio, A. I. Maqueda, C. R. del-Blanco, and D. Scaramuzza. “Dronet: Learning to fly by driving”. In: *IEEE Robotics and Automation Letters* 3.2 (2018), pp. 1088–1095.

[204] J. Maidens, S. Kaynama, I. Mitchell, M. Oishi, and G. Dumont. “Lagrangian methods for approximating the viability kernel in high-dimensional systems”. In: *Automatica* 49.7 (2013), pp. 2017–2029.

[205] A. Majumdar, A. Ahmadi, and R. Tedrake. “Control design along trajectories with sums of squares programming”. In: *International Conference on Robotics and Automation.* 2013.

[206] A. Majumdar and R. Tedrake. “Funnel libraries for real-time robust feedback motion planning”. In: *International Journal of Robotics Research* 36.8 (2017), pp. 947–982.

[207] A. Majumdar, R. Vasudevan, M. Tobenkin, and R. Tedrake. “Convex optimization of nonlinear feedback controllers via occupation measures”. In: *International Journal of Robotics Research.* Vol. 33. 9. 2014, pp. 1209–1230.

[208] A. Marco, P. Hennig, J. Bohg, S. Schaal, and S. Trimpe. “Automatic LQR tuning based on Gaussian process global optimization”. In: *International Conference on Robotics and Automation.* 2016.

[209] K. Margellos and J. Lygeros. “Hamilton-Jacobi Formulation for Reach-Avoid Differential Games”. In: *IEEE Transactions on Automatic Control* 56.8 (2011), pp. 1849–1861.

[210] R. Martinez-Cantin. “BayesOpt: a Bayesian optimization library for nonlinear optimization, experimental design and bandits.” In: *Journal of Machine Learning Research* 15.1 (2014), pp. 3735–3739.

[211] R. Martın-Martın, H. Rezatofighi, A. Shenoi, M. Patel, J. Gwak, N. Dass, A. Federman, P. Goebel, and S. Savarese. “JRDB: A Dataset and Benchmark for Visual Perception for Navigation in Human Environments”. In: *arXiv preprint arXiv:1910.11792* (2019).

[212] M. Massink and N. De Francesco. “Modelling free flight with collision avoidance”. In: *Conference on Engineering of Complex Computer Systems.* 2001.

[213] D. Mellinger and V. Kumar. “Minimum snap trajectory generation and control for quadrotors”. In: *International Conference on Robotics and Automation.* 2011.

[214] D. Mellinger, A. Kushleyev, and V. Kumar. “Mixed-integer quadratic program trajectory generation for heterogeneous quadrotor teams”. In: *International Conference on Robotics and Automation.* 2012.

[215] X. Meng, N. Ratliff, Y. Xiang, and D. Fox. “Neural Autonomous Navigation with Riemannian Motion Policy”. In: *arXiv preprint arXiv:1904.01762* (2019).

[216] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, et al. “Learning to navigate in complex environments”. In: *International Conference on Learning Representations.* 2017.

[217] I. Mitchell. “Scalable calculation of reach sets and tubes for nonlinear systems with terminal integrators: a mixed implicit explicit formulation”. In: *International conference on Hybrid systems: computation and control*. 2011.

[218] I. M. Mitchell. “The flexible, extensible and efficient toolbox of level set methods”. In: *Journal of Scientific Computing*. 2008.

[219] I. Mitchell. “A toolbox of level set methods”. In: *Department of Computer Science, University of British Columbia, Vancouver, BC, Canada, http://www. cs. ubc. ca/˜ mitchell/ToolboxLS/toolboxLS.pdf, Tech. Rep. TR-2004-09* (2004).

[220] I. Mitchell. “Application of Level Set Methods to Control and Reachability Problems in Continuous and Hybrid Systems”. PhD thesis. Stanford University, 2002.

[221] I. Mitchell. “Comparing forward and backward reachability as tools for safety analysis”. In: *International Workshop on Hybrid Systems: Computation and Control*. 2007.

[222] I. Mitchell, A. Bayen, and C. Tomlin. “A time-dependent Hamilton-Jacobi formulation of reachable sets for continuous dynamic games”. In: *IEEE Transactions on Automatic Control* 50.7 (2005), pp. 947–957.

[223] I. Mitchell and C. Tomlin. “Overapproximating Reachable Sets by Hamilton-Jacobi Projections”. In: *Journal of Scientific Computing* 19.1-3 (2003), pp. 323–346.

[224] I. M. Mitchell, M. Chen, and M. Oishi. “Ensuring safety of nonlinear sampled data systems through reachability”. In: *IFAC Proceedings Volumes* 45.9 (2012), pp. 108–114. DOI: `10.3182/20120606-3-NL-3011.00104`.

[225] S. Mitsch, K. Ghorbal, and A. Platzer. “On Provably Safe Obstacle Avoidance for Autonomous Robotic Ground Vehicles”. In: *Robotics: Science and Systems*. 2013.

[226] J. Močkus. “On Bayesian methods for seeking the extremum”. In: *Optimization Techniques IFIP Technical Conference*. 1975.

[227] T. Moldovan and P. Abbeel. “Safe exploration in Markov decision processes”. In: *arXiv preprint* (2012).

[228] M. Müller, A. Dosovitskiy, B. Ghanem, and V. Koltun. “Driving policy transfer via modularity and abstraction”. In: *arXiv preprint arXiv:1804.09364* (2018).

[229] R. Murray-Smith and D. Sbarbaro. “Nonlinear adaptive control using nonparametric Gaussian process prior models”. In: *IFAC Proceedings Volumes* 35.1 (2002), pp. 325–330.

[230] A. Nagabandi, G. Kahn, R. Fearing, and S. Levine. “Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning”. In: *arXiv preprint arXiv:1708.02596* (2017).

[231] A. Nagabandi, G. Yang, T. Asmar, G. Kahn, S. Levine, and R. Fearing. “Neural Network Dynamics Models for Control of Under-actuated Legged Millirobots”. In: *arXiv preprint arXiv:1711.05253* (2017).

[232] New Atlas. *Amazon Prime Air.* URL: `http://newatlas.com/amazon-new-delivery-drones-us-faa-approval/36957/`.

[233] D. Nguyen-Tuong and J. Peters. "Model learning for robot control: a survey". In: *Cognitive Processing* 12.4 (2011), pp. 319–340.

[234] D. Nguyen-Tuong, M. Seeger, and J. Peters. "Computed torque control with non-parametric regression models". In: *American Control Conference.* 2008.

[235] D. Nguyen-Tuong, M. Seeger, and J. Peters. "Model learning with local Gaussian process regression". In: *Advanced Robotics* 23.15 (2009), pp. 2015–2034.

[236] P. Nilsson and N. Ozay. "Synthesis of separable controlled invariant sets for modular local control design". In: *American Control Conference.* 2016.

[237] R. Olfati-Saber and R. Murray. "Distributed cooperative control of multiple vehicle formations using structural potential functions". In: *IFAC Proceedings Volumes* 35.1 (2002), pp. 495–500.

[238] M. Osborne, R. Garnett, and S. Roberts. "Gaussian processes for global optimization". In: *Learning and Intelligent Optimization (LION3).* 2009, pp. 1–15.

[239] S. Osher and R. Fedkiw. *Level Set Methods and Dynamic Implicit Surfaces.* Springer-Verlag, 2006.

[240] S. Osher, R. Fedkiw, and K. Piechor. "Level set methods and dynamic implicit surfaces". In: *Applied Mechanics Reviews* 57.3 (2004), B15–B15.

[241] L. Pallottino, V. Scordio, A. Bicchi, and E. Frazzoli. "Decentralized Cooperative Policy for Conflict Resolution in Multivehicle Systems". In: *IEEE Transactions on Robotics* 23.6 (2007), pp. 1170–1183.

[242] Y. Pan and E. Theodorou. "Probabilistic differential dynamic programming". In: *Advances in Neural Information Processing Systems.* 2014.

[243] Y. Pan, C.-A. Cheng, K. Saigol, K. Lee, X. Yan, E. Theodorou, and B. Boots. "Agile Off-Road Autonomous Driving Using End-to-End Deep Imitation Learning". In: *Robotics: Science and Systems.* 2018.

[244] A. V. Papadopoulos and M. Prandini. "Model reduction of switched affine systems". In: *Automatica.* 2016.

[245] E. Parisotto and R. Salakhutdinov. "Neural Map: Structured memory for deep reinforcement learning". In: *International Conference on Learning Representations.* 2018.

[246] P. Parrilo. "Structured semidefinite programs and semialgebraic geometry methods in robustness and optimization". Ph.D. Dissertation. California Institute of Technology, 2000. URL: `http://resolver.caltech.edu/CaltechETD:etd-05062004-055516`.

[247] M. Pfeiffer, M. Schaeuble, J. Nieto, R. Siegwart, and C. Cadena. "From perception to decision: A data-driven approach to end-to-end motion planning for autonomous ground robots". In: *International Conference on Robotics and Automation.* 2017.

[248] A. Pokle, R. Martın-Martın, P. Goebel, V. Chow, H. M. Ewald, J. Yang, Z. Wang, A. Sadeghian, D. Sadigh, S. Savarese, and M. Vázquez. "Deep Local Trajectory Replanning and Control for Robot Navigation". In: *International Conference on Robotics and Automation.* 2019.

[249] G. Pola, A. Girard, and P. Tabuada. "Approximately bisimilar symbolic models for nonlinear control systems". In: *Automatica.* 2008.

[250] T. Prevot, J. Rios, P. Kopardekar, J. Robinson III, M. Johnson, and J. Jung. "UAS Traffic Management (UTM) Concept of Operations to Safely Enable Low Altitude Flight Operations". In: *AIAA Aviation Technology, Integration, and Operations Conference.* 2016.

[251] A. Punjani and P. Abbeel. "Deep learning helicopter dynamics models". In: *Robotics and Automation (ICRA), 2015 IEEE International Conference on.* IEEE. 2015, pp. 3223–3230.

[252] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Ng. "ROS: an open-source Robot Operating System". In: *ICRA workshop on open source software.* 2009.

[253] C. Rasmussen and C. Williams. *Gaussian Processes for Machine Learning.* The MIT Press, 2006.

[254] B. Recht. "A tour of reinforcement learning: The view from continuous control". In: *Annual Review of Control, Robotics, and Autonomous Systems* (2018).

[255] M. Reynolds. "Continuous Temporal Models". In: *Australian Joint Conference on Artificial Intelligence.* 2001.

[256] A. Richards and J. How. "Model predictive control of vehicle maneuvers with guaranteed completion time and robust feasibility". In: *American Control Conference* (2003).

[257] C. Richter, W. Vega-Brown, and N. Roy. "Bayesian learning for safe high-speed navigation in unknown environments". In: *Robotics Research.* Springer, 2018, pp. 325–341.

[258] J. Roberts, I. Manchester, and R. Tedrake. "Feedback controller parameterizations for reinforcement learning". In: *Symposium on Adaptive Dynamic Programming And Reinforcement Learning (ADPRL).* 2011, pp. 310–317.

[259] U. Rosolia and F. Borrelli. "Learning model predictive control for iterative tasks. a data-driven control framework". In: *IEEE Transactions on Automatic Control* 63.7 (2018), pp. 1883–1896.

[260] A. Rudenko, L. Palmieri, M. Herman, K. M. Kitani, D. M. Gavrila, and K. O. Arras. "Human Motion Trajectory Prediction: A Survey". In: *arXiv preprint arXiv:1905.06113* (2019).

[261] F. Sadeghi. “DIViS: Domain invariant visual servoing for collision-free goal reaching”. In: *Robotics: Science and Systems* (2019).

[262] F. Sadeghi and S. Levine. “$(\text{CAD})^2$RL: Real Single-Image Flight without a Single Real Image”. In: *Robotics: Science and Systems*. 2017.

[263] S. Sarid, B. X., and H. Kress-gazit. “Guaranteeing high-level behaviors while exploring partially known maps”. In: *Robotics: Science and Systems*. 2012.

[264] S. Sastry and M. Bodson. *Adaptive control: stability, convergence and robustness*. Courier Corporation, 2011.

[265] A. J. Sathyamoorthy, J. Liang, U. Patel, T. Guan, R. Chandra, and D. Manocha. “DenseCAvoid: Real-time navigation in dense crowds using anticipatory behaviors”. In: *arXiv* (2020).

[266] N. Savinov, A. Dosovitskiy, and V. Koltun. “Semi-parametric Topological Memory for Navigation”. In: *International Conference on Learning Representations*. 2018.

[267] S. Schaal. “Learning robot control”. In: *The handbook of brain theory and neural networks* 2 (2002), pp. 983–987.

[268] T. Schouwenaars and E. Feron. “Decentralized Cooperative Trajectory Planning of Multiple Aircraft with Hard Safety Guarantees”. In: *AIAA Guidance, Navigation and Control Conference*. 2004.

[269] J. Schreiter, P. Englert, D. Nguyen-Tuong, and M. Toussaint. “Sparse Gaussian Process Regression for Compliant, Real-Time Robot Control”. In: *International Conference on Robotics and Automation*. 2015.

[270] J. Schulman, J. Ho, A. Lee, I. Awwal, H. Bradlow, and P. Abbeel. “Finding Locally Optimal, Collision-Free Trajectories with Sequential Convex Optimization.” In: *Robotics: Science and Systems*. 2013.

[271] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. “Trust region policy optimization”. In: *International conference on machine learning*. 2015.

[272] J. Schulman, P. Wolski F.and Dhariwal, A. Radford, and O. Klimov. “Proximal policy optimization algorithms”. In: *arXiv preprint arXiv:1707.06347* (2017).

[273] S. A. Seshia, D. Sadigh, and S. S. Sastry. “Towards Verified Artificial Intelligence”. In: *arXiv:1606.08514*. 2016.

[274] S. A. Seshia, A. Desai, T. Dreossi, D. J. Fremont, S. Ghosh, E. Kim, S. Shivakumar, M. Vazquez-Chanlatte, and X. Yue. “Formal specification for deep neural networks”. In: *International Symposium on Automated Technology for Verification and Analysis*. 2018.

[275] J. Sethian. “A fast marching level set method for monotonically advancing fronts”. In: *National Academy of Sciences* 93.4 (1996), pp. 1591–1595.

[276] B. Shahriari, K. Swersky, Z. Wang, R. Adams, and N. de Freitas. "Taking the human out of the loop: A review of Bayesian optimization". In: *Proceedings of the IEEE* 104.1 (2016), pp. 148–175.

[277] P. Y. Simard, D. Steinkraus, J. C. Platt, et al. "Best practices for convolutional neural networks applied to visual document analysis." In: *ICDAR*. 2003.

[278] S. Singh, M. Chen, S. Herbert, C. Tomlin, and M. Pavone. "Robust Tracking with Model Mismatch for Fast and Safe Planning: An SOS Optimization Approach". In: *International Workshop on the Algorithmic Foundations of Robotics*. 2020.

[279] S. Singh, A. Majumdar, J. Slotine, and M. Pavone. "Robust online motion planning via contraction theory and convex optimization". In: *International Conference on Robotics and Automation*. 2017.

[280] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. "Dropout: A simple way to prevent neural networks from overfitting". In: *Journal of Machine Learning Research* (2014).

[281] D. Stipanović, P. Hokayem, M. Spong, and D. Šiljak. "Cooperative Avoidance Control for Multiagent Systems". In: *Journal of Dynamic Systems, Measurement, and Control* 129.5 (2007), p. 699.

[282] S. Strubbe and A. Van Der Schaft. "Bisimulation for communicating piecewise deterministic Markov processes (CPDPs)". In: *International Workshop on Hybrid Systems: Computation and Control*. 2005.

[283] R. Sutton. "Dyna, an integrated architecture for learning, planning, and reacting". In: *ACM SIGART Bulletin* 2.4 (1991), pp. 160–163.

[284] R. Sutton and A. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[285] L. Tai, G. Paolo, and M. Liu. "Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation". In: *Conference on Intelligent Robots and Systems*. 2017.

[286] helperOC Team. *helperOC Library*. https://github.com/HJReachability/helperOC. 2019.

[287] R. Tedrake, I. Manchester, M. Tobenkin, and J. Roberts. "LQR-trees: Feedback Motion Planning via Sums-of-Squares Verification". In: *International Journal of Robotics Research* 29.8 (2010), pp. 1038–1052.

[288] S. Thrun, W. Burgard, and D. Fox. *Probabilistic robotics*. MIT press, 2005.

[289] B. Tice. "Unmanned Aerial Vehicles: The Force Multiplier of the 1990s". In: *Airpower Journal* (1991).

[290] E. Todorov, T. Erez, and Y. Tassa. "MuJoCo: A physics engine for model-based control". In: *Conference on Intelligent Robots and Systems*. 2012.

[291] V. Tolani, S. Bansal, A. Faust, and C. Tomlin. "Visual Navigation Among Humans with Optimal Control as a Supervisor". In: *arXiv preprint arXiv:2003.09354* (2020).

[292] C. J. Tomlin, J. Lygeros, and S. Sastry. "A game theoretic approach to controller design for hybrid systems". In: *Proceedings of the IEEE*. 2000.

[293] C. J. Tomlin, G. J. Pappas, and S. Sastry. "Conflict resolution for air traffic management: A study in multiagent hybrid systems". In: *IEEE Transactions on automatic control*. 1998.

[294] S. Trimpe, A. Millane, S. Doessegger, and R. D'Andrea. "A self-tuning LQR approach demonstrated on an inverted pendulum". In: *IFAC Proceedings Volumes* 47.3 (2014), pp. 11281–11287.

[295] J. Van den Berg, L. Ming, and D. Manocha. "Reciprocal Velocity Obstacles for real-time multi-agent navigation". In: *International Conference on Robotics and Automation*. 2008.

[296] J. Van Den Berg, P. Abbeel, and K. Goldberg. "LQG-MP: Optimized path planning for robots with motion uncertainty and imperfect state information". In: *The International Journal of Robotics Research* (2011).

[297] H. Van Hasselt, A. Guez, and D. Silver. "Deep reinforcement learning with double q-learning". In: *AAAI conference on artificial intelligence*. 2016.

[298] P. Varaiya. "On the existence of solutions to a differential game". In: *SIAM Journal on Control* 5.1 (1967), pp. 153–162.

[299] G. Varol, J. Romero, X. Martin, N. Mahmood, M. J. Black, I. Laptev, and C. Schmid. "Learning from Synthetic Humans". In: *Conference on Computer Vision and Pattern Recognition*. 2017.

[300] R. Walambe, N. Agarwal, S. Kale, and V. Joshi. "Optimal trajectory generation for car-type mobile robot using spline interpolation". In: *IFAC-PapersOnLine* 49.1 (2016), pp. 601–606.

[301] M. Watter, J. Springenberg, J. Boedecker, and M. Riedmiller. "Embed to control: A locally linear latent dynamics model for control from raw images". In: *Advances in neural information processing systems*. 2015.

[302] Webots. *http://www.cyberbotics.com*. Ed. by C. Ltd. Commercial Mobile Robot Simulation Software. 1998.

[303] Wikipedia. *Beaufort scale*. URL: `https://en.wikipedia.org/wiki/Beaufort-scale`.

[304] A. Wilson, A. Fern, and P. Tadepalli. "Using trajectory data to improve bayesian optimization for reinforcement learning". In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 253–282.

[305] A. Wu and J. How. "Guaranteed infinite horizon avoidance of unpredictable, dynamically constrained obstacles". In: *Autonomous Robots* 32.3 (2012), pp. 227–242.

[306] L. Yoder and S. Scherer. “Autonomous exploration for infrastructure modeling with a micro aerial vehicle”. In: *Field and service robotics*. 2016.

[307] M. Zeiler, R. Fergus. “Visualizing and understanding convolutional networks”. In: *European Conference on Computer Vision*. 2014.

[308] W. Zeng, W. Luo, S. Suo, A. Sadat, B. Yang, S. Casas, and R. Urtasun. “End-to-end Interpretable Neural Motion Planner”. In: *Conference on Computer Vision and Pattern Recognition*. 2019.

[309] M. Zhaowei, H. Tianjiang, S. Lincheng, K. Weiwei, Z. Boxin, and Y. Kaidi. “An iterative learning controller for quadrotor UAV path following at a constant altitude”. In: *Chinese Control Conference*. 2015.

[310] D. Zhou, Z. Wang, S. Bandyopadhyay, and M. Schwager. “Fast, On-line Collision Avoidance for Dynamic Vehicles Using Buffered Voronoi Cells”. In: *IEEE Robotics and Automation Letters* 2 (2017), pp. 1047–1054.

[311] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi. “Target-driven Visual Navigation in Indoor Scenes using Deep Reinforcement Learning”. In: *International Conference on Robotics and Automation*. 2017.