Safe Real-World Autonomy in Uncertain and Unstructured Environments

Sylvia Herbert

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2020-147
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2020/EECS-2020-147.html

August 13, 2020

Acknowledgement

To James Herbert and Lynn Brandsma. For years of joy, support, love, and ``character-building."

Safe Real-World Autonomy in Uncertain and Unstructured Environments

by

Sylvia Herbert

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Engineering – Electrical Engineering and Computer Sciences

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Claire Tomlin, Chair
Professor Shankar Sastry
Professor Thomas Griffiths

Summer 2020

Safe Real-World Autonomy in Uncertain and Unstructured Environments

Abstract

Safe Real-World Autonomy in Uncertain and Unstructured Environments

by

Sylvia Herbert

Doctor of Philosophy in Engineering – Electrical Engineering and Computer Sciences

University of California, Berkeley

Professor Claire Tomlin, Chair

We are captivated by the promise of autonomous systems in our everyday life. However, ensuring that these systems act safely is an immense challenge: introducing complex systems into real-world uncertain environments while guaranteeing safety at all times is impossible in applications like self-driving vehicles, collaborative factory robots, and assistive robots. These systems will inevitably need to make real-time decisions with limited computational resources, and incomplete knowledge of the environment and other agents. This dissertation is an effort towards achieving trustworthy real-world autonomy by enabling autonomous systems to:

- make theoretical safety guarantees efficiently based on known information, and
- bridge the safety gap between this theory and the real world by reasoning about uncertainty in the environment and other agents.

Towards this goal this dissertation covers various methods for scalable safety and real-time decision-making that draw from control theory, cognitive science, and learning, and are backed by both rigorous theory and physical testing on robotic platforms. We begin with an overview of reachability analysis and its applications for optimal control with safety guarantees. We then tackle the curse of dimensionality associated with reachability computation by decomposing systems or updating previous solutions in a warm-starting fashion. Next we explore planning in a simplified, low-dimensional space online with precomputed safety guarantees and tracking controls offline. This is extended to meta-planning, in which the online algorithm switches between faster/conservative modes and slower/accurate modes. Then we apply all of these tools towards navigating in uncertain environments among hard-to-predict agents such as human pedestrians. We use Bayesian machine learning methods to reason about human intention and navigate in a probabilistically safe manner. Finally, the dissertation ends with information about code bases and reachability tutorial examples.

To James Herbert and Lynn Brandsma. For years of joy, support, love, and "character-building."

Contents

List of Figures

List of Tables

Acknowledgments

Graduate school is a strange experience. I have stretched my brain to absorb concepts and ideas that I could have never imagined before, and with every bit of new information gained I get more appreciation for how vast the collection of human knowledge spans–and the impossibility of fully grasping even a small slice of it. And I have learned that for every research paper written, a dozen (or more) new questions appear. At first this experience feels overwhelming–how can you feel you've made progress when every step forward reveals more that is unknown? Which direction is the most important? The most feasible? The most exciting?

Fortunately, I have been surrounded by an amazing team of fellow explorers who have lit the way for me and help me navigate through countless seemingly dead ends. This has transformed the graduate school experience to one of exciting exploration and many lively discussions around a whiteboard (or a beer). Because of these colleagues I will always look back and treasure my time in grad school, painful brain-stretching and all.

I would first like to thank my advisor Claire Tomlin. When I entered her lab coming from a different field, I knew no programming languages, no probability, and hardly any systems theory. Every week I would find myself in doubt, wondering if I would ever get to the point of being useful. But after my weekly meeting with her, I would leave feeling supported, energized, and reassured that my questions and explorations were worthwhile. She has supported me through many personal emergencies and helped me navigate the politics of academia. I can not imagine having a better boss, and I hope I can emulate her as I start as a faculty member.

My graduate career has additionally been shaped by the other members of my thesis committee. Shankar Sastry has guided many of my research directions by helping me keep sight of the big picture, and has debugged multiple ideas and proofs. Tom Griffiths welcomed my nascent interest in cognitive science and took me on a guided tour of the history and current state-of-the-art work on computational models of cognition, inspiring much of my most-cited work.

I would also like to thank my labmates. This diverse group of students have taught me just about everything I know technically and have challenged me to learn how to communicate ideas to people of different backgrounds, culture, and expertise. By learning how to think and speak from different perspectives my research has evolved in ways that would simply not have happened at all if I was working alone. I owe my professional success to these people, as well as my personal day-to-day happiness. I would like in particular to thank my frequent collaborators. Mo Chen was my graduate mentor starting out, and his guidance groomed me as a researcher. Somil Bansal and Andrea Bacjsy have also been crucial to my technical knowledge through many long discussions in front of a whiteboard. David Fridovich-Keil and Jaime Fisac always made projects entertaining with silly antics and fun diversions into theory and algorithms. I am so grateful for the opportunity to work with these people, along with all the others in my lab and in the controls community at Berkeley.

Next, I would like to thank my family. My parents were the first to push me towards exploring STEM, and along with my grandmother have enthusiastically supported me throughout my education. Growing up with many very different brothers has taught me how to work well on diverse teams and to not take myself too seriously.

I met my husband at the start of graduate school–we were the only two students to have both signed up for Linear Systems Theory and Introduction to Optical Engineering. We were dating before the end of semester, and have been together ever since. We have helped each other through countless research and personal struggles. He is always asking questions and exploring ideas about the world that I never thought to question, broadening my mind and leading to many enthusiastic debates. One of our first purchases together was a whiteboard, which has seen a lot of use over breakfasts and dinners. His support and his humor have been a rock for me through this period of our lives.

Finally, I would like to thank our dog Abby Normal. She is the best de-stresser I have ever experienced, and has forced me to maintain a healthy lifestyle during my years in grad school. She has this effect on everyone–when she visits the department she is swarmed by other graduate students who fight over taking turns to watch over her.

As is evident by this section, I have led a very lucky life and have been particularly fortunate during my time in graduate school. I am overwhelmed with gratitude for those listed here and everyone else who has supported me in this venture. Thank you.

Chapter 1

Introduction

My research has been motivated by a fairly straightforward idea: we want autonomous systems to function safely in real-world environments. Historically, this meant that we wanted to guarantee the safety of every possible trajectory of a system through its environment. But the systems that we are interested in have become more and more complex (e.g. factory robots, cars, assistive robots). Moreover, these systems are operating in less structured environments, often in the presence of other agents (e.g. humans). As a result, many companies rely on heuristic measures of safety, sometimes with disastrous and even fatal results. Finding a way to advance rigorous safety analysis to be able to deal with these challenges is crucial for safe real-world autonomy.

1.1 Safety Analysis for Autonomous Systems

However, verification of 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.

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 [12, 24]. 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 [146, 64] allows specification of properties such as time in terms of real numbers, and chance-constrained temporal logic [93] 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 [47] and human-

automation interaction [28].

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 [74], Flow* [42], CORA [6], C2E2 [60, 65], and dReach [104] have been developed for this purpose; the authors in [61] 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 [139, 164]. This is done, for example, by constructing barrier certificates [20] or funnels [121, 122] with Lyapunov properties.

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 [19, 126, 27, 73].

This dissertation largely focuses on a technique called Hamilton-Jacobi (HJ) reachability analysis. Hamilton-Jacobi (HJ) reachability analysis is a verification method for guaranteeing performance and safety properties of systems, overcoming some of the above challenges. HJ reachability can be distinguished from other methods because it is applicable to general nonlinear systems, easily handles control and disturbance variables, and is able to represent sets of arbitrary shapes. However, this flexibility comes with the cost of computational complexity.

The focus of this dissertation is to tackle challenges to safe real-world autonomy, including computational complexity, uncertainty and changing information during online operation, and multi-agent environments such as those involving humans. We will introduce techniques that work towards overcoming these challenges in order to provide safety guarantees and assurances for autonomous systems operating in a world of disturbances, uncertainties, and other agents.

1.2 Challenges in Safety for Real-World Autonomy[1]

Computational Efficiency of Safety Verification

In reachability analysis, one computes the reach-avoid set, defined as the set of states from which the system can be driven to a target set while satisfying time-varying state constraints at all times. A major practical appeal of this approach stems from the availability of modern numerical tools, which can compute various definitions of reachable sets [158, 138, 129, 42]. For example, these numerical tools have been successfully used to solve a variety of differential games, path planning problems, and optimal control problems. Concrete prac-

[1]This is by no means a complete list of challenges, but includes the major challenges addressed in this dissertation.

tical applications include aircraft auto-landing [21], automated aerial refueling [57], model predictive control (MPC) of quadrotors [29, 10], multiplayer reach-avoid games [91], large-scale multiple-vehicle path planning [40, 41], and real-time safe motion planning [88, 107]. However, HJ reachability becomes computationally intractable as the state space dimension increases.

Traditionally, reachable set computations 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." This issue can be mitigated somewhat for linear systems by using convex optimization applied to the Hopf-Lax formula to allow for real-time computation of the HJ PDE solution at any desired state and time instant [50, 44]. Other backward reachability methods make other trade-offs. For example, [74, 112, 120] present scalable methods for affine systems that rely on polytopic or ellipsoidal representation of sets, while the methods presented in [121, 58, 86] are well-suited to systems with polynomial dynamics. However, general nonlinear systems remain a challenge.

Adapting to New Information and Changing Assumptions Online

A result of the computational complexity of these verification techniques is that systems struggle to update safety analyses in the face of new information. Instead, assumptions about the system dynamics, disturbances, and the environment must be made ahead of time in order to precompute the verification. In the real world, some assumptions will inevitably be wrong, invalidating the safety precomputation. Understanding how to efficiently adapt these analyses online will be important to real-world safety.

Multi-Agent and Human-Centered Systems

Operating in an environment with multiple agents is particularly relevant and challenging for real-world applications. Other agents may be hard to predict and plan around. Even under the assumption that the agents are actively working together, reasoning directly in the joint state space of multiple agents quickly becomes computationally intractable for realistic systems. Clever inference, prediction, and decomposed planning techniques will be required to make autonomous systems operate intelligently among humans and other systems.

1.3 Thesis Overview and Summary of Contributions

First, in Chapter 2 we will review the background information that serves as the foundation of the research in this thesis. This material draws largely from the HJ reachability tutorial [17].

Part I: Theory for Computational Gains in Reachability Analysis. The first part of the thesis provides theoretical advances for more scalable computing of HJ reachability

analyses for nonlinear systems with exact or conservative guarantees that preserve safety. Chapter 3 begins with the decoupling as disturbance technique that forcibly splits high-dimensional systems (around 10 dimensions, compared the standard 3 or 4) into smaller (and more tractable) subsystems while maintaining conservative safety guarantees [35]. Chapter 4 follows this with the self-contained subsystems work that allows certain formulations of coupled dynamics to decompose in a manner that allows for exact computation in reachability analysis [38, 34]. Finally, Chapter 5 introduces recent work on warm-start reachability analysis, which allows an autonomous system to update its safety analysis based on new information and assumptions with exact or conservative guarantees [87]. By updating the analysis from a previous solution the computation time can be drastically decreased for high-dimensional systems. All of these techniques have enabled us to (a) apply reachability analysis to more realistic higher-dimensional dynamic models and (b) make use of offline computation of safety analyses (even if these analyses become quickly outdated by new information) by updating those offline computations using warm-start reachability.

Part II: Fast and Safe Tracking. While scalability work in Part I for theoretical guarantees is an important step towards safe real-world autonomy, these analyses depend on pre-defined assumptions about the system model and environment that may be wrong in practice. Therefore, we must equip autonomous systems with the ability to build upon theoretical safety guarantees by reasoning about uncertainty in the real world. This is where humans excel compared to current systems. We can reason efficiently in unstructured environments with limited computational resources. Researchers believe that humans plan using simplified physical models, with an internal understanding of how these simple plans map back to control policies and safety constraints on our true high-dimensional model. Chapter 6 introduces the framework FaSTrack: Fast and Safe Tracking [88, 43] that mimics human cognition by allowing for real-time online planning using a simplified planning model, and "robustifying" this model with a precomputed feedback controller for the true system to track the simple model. This is extended in Chapter 7 with a hybrid model based on the idea that humans adaptively switch between a fast (and error-prone) mode and a slow (but accurate) mode. This meta-planning hybrid model adaptively switches to the best planning mode for the local sensed environment [76]. Together, this work has been used on multiple platforms within and beyond UC Berkeley. Its success can be attributed to the simplicity of the online algorithm, modularity with respect to "robustifying" different motion planning techniques, and ability to separate the safety analysis from the online planning.

Part III: Navigation in Multi-Agent Environments. The next stage in producing safe autonomy for the real world is tackling the challenge of dynamic and uncertain environments. Trying to predict the movement of other agents like humans is both computationally challenging and inevitably imperfect. The psychology of human prediction suggests that we use simple noisily-rational models of other humans to make fast predictions of their future movement. We have high confidence in these predictions when humans match our simplified models (for example, when walking in straight lines along the sidewalk), and lose confidence when a human is acting unexpectedly. Chapter 8 introduces a confidence-aware prediction framework that allows us to employ simple models of human motion while reasoning about

the mismatch between these models and the observed human behavior [72, 75]. The framework employs a probabilistic Boltzmann model of human behavior, where the distribution's variance is automatically adjusted based on measurements. When the observed human behavior is well-explained by the model, our framework provides a tight confident distribution over future human motion. When our model does not explain human behavior well, the probability distribution over the next human states increases in uncertainty. Combined with FaSTrack, we can provide probabilistic safety assurances for real-time planning and control in environments with human pedestrians. Chapter 9 extends this work to multi-agent scenarios with multiple humans and multiple cooperative robots [13].

Part IV: Codebases and Conclusions. The dissertation concludes with an overview of the codebases used throughout the dissertation, followed by some concluding thought and future directions.

Additional Contributions

There are two additional papers from my PhD research that are not covered in this dissertation. Both focus on computing HJ reachability analysis without grid-based techniques. The first [150] trains a neural-network based classifier on the optimal policy for a system using approximate dynamic programming, resulting in conservative guarantees. The second [161] uses sum-of-squares optimization to compute higher-dimensional reachability analyses. Both papers use FaSTrack as a motivating application.

Chapter 2

Background

This chapter is based in part on the paper "Hamilton-Jacobi Reachability: A Brief Overview and Recent Advances" [17], written in collaboration with Somil Bansal, Mo Chen, and Claire J. Tomlin.

2.1 Dynamic Systems

The work in this dissertation is focused on understanding and controlling systems that operate in time. The concept of system here is quite general, and can be applied to physical systems (e.g. robotics, vehicles, humans) or non-physical systems (e.g. financial markets, decision processes). As a simple motivating running example, consider the handcar in Fig. 2.1. We are interested in analyzing how a system evolves over time, and what inputs can be applied to steer the system towards desired outcomes and away from unsafe outcomes. Appropriately, we call these inputs *control inputs*. In the running example, a person can directly control the acceleration of the handcar.

We may also have uncertainty. This could be due to a variety of reasons, such as a mismatch between our mathematical model and the true system, actions of other players that may affect the system, or external forces that act upon the system. To account for this uncertainty, we introduce a *disturbance input* to the model. This is a term that we have no control over, and can potentially change instantaneously and in unpredicted ways. In the motivating example, we will assume that there is wind in the environment that can affect the velocity of the handcar. We will assume that this wind velocity can change instantaneously, but it will be bounded by a maximum possible wind speed.

To analyze the evolution of the system over time, one must keep track of the *state* of the system. This is a vector that comprises all parameters that are necessary to keep track of the evolution of the model. In the running example, we will consider the state of the system as consisting of the horizontal position and velocity of the handcar. Note that this is a fairly simplistic model–we are not considering the lever used to operate the handcar, or the dynamics of the human controlling the handcar. A more complicated model would

require a richer state space.

Note also that we define the handcar system independently of the environment. Obstacles and goals in the environment will be handled separately from our model of the handcar. This is compared to the field of artificial intelligence, where the system and environment are grouped together (for example, consider the game pong, where the AI field considers the paddle and the surrounding environment as a single system termed "the environment"). Separating the system from the environment allows us to easily generalize our analysis of the system to different environments.

Mathematically, let $x \in \mathbb{R}^n$ be the system state, which evolves according to the ordinary differential equation (ODE)

$$\begin{aligned} \dot{x}(\tau) &= f(x(\tau), u(\tau), d(\tau)), \\ \tau \in [t, 0], u(\tau) &\in \mathcal{U}, d(\tau) \in \mathcal{D}, \end{aligned} \tag{2.1}$$

where $u(\tau)$ and $d(\tau)$ denote the control and the disturbance respectively. We assume that these inputs are drawn from compact sets ($\mathcal{U} \subset \mathbb{R}^{n_u}$ and $\mathcal{D} \subset \mathbb{R}^{n_d}$). This assumption typically holds, as the magnitude of control that we can assert on the system is typically bounded. Assuming that the disturbance is bounded is often true in practice, but can become an issue if these bounds are too big (and therefore allow the disturbance to have a huge effect on the system) or too small (and are therefore invalid compared to the true disturbance).

The system dynamics, or flow field, $f : \mathbb{R}^n \times \mathcal{U} \times \mathcal{D} \to \mathbb{R}^n$ is assumed to be uniformly continuous, and Lipschitz continuous in x uniformly[1] in $u(\cdot)$ and $d(\cdot)$. Therefore, given $u(\cdot) \in \mathbb{U}$ and $d(\cdot) \in \mathbb{D}$, there exists a unique trajectory solving (2.1) [46]. We will denote solutions, or trajectories of (2.1) starting from state x at time t under control $u(\cdot), d(\cdot)$ as

$$\zeta(\tau; x, t, u(\cdot), d(\cdot)) : [t, T] \to \mathbb{R}^n. \tag{2.2}$$

This notation can be read as the state achieved at time τ parameterized by initial state x, initial time t, and input functions $u(\cdot)$ and $d(\cdot)$ applied over $[t, \tau]$. Because we tend to solve reachability problems backwards in time, we use the notation that forward trajectories end at final time $\tau = T$, and start at an initial negative time t.

The output of the trajectory will always be a state. The point of trajectory notation is to aid in keeping track of the progression of states along a trajectory over time. This is useful in many of the proofs. The trajectory satisfies (2.1) with an initial condition almost everywhere:

$$\begin{aligned} \frac{d}{d\tau}\zeta(\tau; x, t, u(\cdot), d(\cdot)) &= f\Big(\zeta\Big(\tau; x, t, u(\cdot), d(\cdot)\Big), u(\tau), d(\tau)\Big) \\ \zeta(t; x, t, u(\cdot), d(\cdot)) &= x \end{aligned} \tag{2.3}$$

Here we have replaced the state x in (2.1) with the trajectory notation.

[1]For the remainder of dissertation, I will omit the notation (τ) from variables such as x when referring to function values.

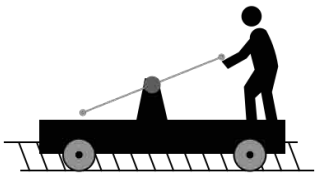


Figure 2.1: Motivating system example: a handcar. Dynamics assumed to be a double integrator, as seen in the running example

A Brief Note on Time, State, and Trajectory Notation

Unless otherwise noted, the time horizon we consider is $\tau \in [t, T]$. Often, we think of the final time $T = 0$, with a negative initial time $t \leq 0$. This may seem confusing now, but makes for cleaner notation when updating trajectories backwards in time, as we will do throughout this dissertation.

State, control, and disturbance are all functions of time. Sometimes it is necessary to reason explicitly about the time associated with a state (or control, or disturbance), and so we explicitly write $x(t)$ or x_t (following the same pattern for control and disturbance). This can make for very long equations, so whenever the time is obvious, we omit the time argument.

When referring to an input function over time, we typically use $u(\cdot)$ and $d(\cdot)$. When we reason about a state trajectory over time, we should use the trajectory notation $\zeta(\tau; x, t, u(\cdot), d(\cdot))$, referring to a trajectory that starts at state x, time t, and applies control $u(\cdot)$ and disturbance $d(\cdot)$, ending at time τ. This trajectory will output the state at the final time, i.e. $\zeta(\tau; x, t, u(\cdot), d(\cdot)) = x(\tau)$. For compactness we typically refer to these trajectories as $\zeta_{x,t}^{u,d}(\tau)$. However, for brevity's sake, when the trajectory is obvious we often simply use $x(\cdot)$ to refer to a general trajectory, and $x(\tau)$ to refer to the trajectory at time τ.

Example 2.1. *For illustration we use the handcar as a running example. We assume it operates as a double integrator. Recall that our analysis can handle general nonlinear systems, but sometimes a simple linear system is helpful for illustration. Its system dynamics are:*

$$\dot{x} = \begin{bmatrix} \dot{p} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} v + d \\ u \end{bmatrix}, \tag{2.4}$$

with states position p and velocity v, where $u \in [-1, 1]$ is acceleration. By default the disturbance is $d = 0$.

2.2 Optimal Control

The notion of optimal control is fairly intuitive: if I have some control over a system, what inputs should I apply that will cause the system to reach some goal (or in general achieve some objective) in an optimal way? We would ideally like our HVAC units, transit systems, and robots to all function optimally. This raises the question of what optimality means.

The Optimal Control Problem

In the literature of optimal control, optimality means minimizing some cost function:

$$J(x, t, u(\cdot)) = \int_t^T c(x(\tau), u(\tau))d\tau + l(x(T)) \tag{2.5}$$

where $c(x(t), u(t))$ is a running cost that the system occurs at every instant (for example, amount of fuel being used), and $l(x(T))$ is a terminal cost (for example, distance to goal). Determining the "correct" or desired cost function is non-trivial and in general is defined by the user. In this thesis we will largely consider costs that reward entering goal sets and avoiding unsafe sets.

In an optimal control problem, the goal is to use the control signal optimally in order to minimize the total cost. This leads to the value function, written as:

$$V(x(t), t) = \inf_{u(\cdot)} \quad J(x, t, u(\cdot)) \tag{2.6}$$

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

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

Thus, the "value" of a particular initial state and time depends on the cost of the system from that state and time given optimal control input. The optimization is constrained by the fact that the state evolution of the system must obey physics (the system dynamics), and the control signal must stay within its bounds.

Example 2.2. *Example of Optimal Control Problem for 2D Cart*
Recall the 2D cart from Example 2.1, and consider the problem of minimizing distance to the origin in position space after a time interval $[t, T]$. This is a terminal cost and can be written as $l(x) = |p|$. The associated value function at the initial time will be:

$$V(x(t),t) = \inf_{u(\cdot)} \quad |p| \tag{2.9}$$

$$\textit{subject to} \quad \begin{bmatrix}\dot{p}\\ \dot{v}\end{bmatrix} = \begin{bmatrix}v+d\\ u\end{bmatrix}, \quad \forall \tau \in [t,T] \tag{2.10}$$

$$u(\tau) \in \mathcal{U} \tag{2.11}$$

Optimal control problems have been studied extensively over the past few decades. There are two major approaches to solving such problems: calculus of variations, and dynamic programming. *Calculus of variations* brings the constraints of an optimal control problem into the objective and considers how the value of the system changes under local deformations of the trajectory. Unfortunately, this technique does not provide a global solution to the optimal control problem.

Dynamic programming is currently the most commonly used technique for solving optimal control and reinforcement learning problems. The core idea of dynamic programming is to reason recursively about the optimal control of a system backwards in time. This recursive update equation is called the Bellman equation (or Bellman backup) in discrete time, and the Hamilton-Jacobi-Bellman (HJB) partial differential equation (PDE) in continuous time. In either case, this equation provides a necessary and sufficient condition that the value function must satisfy, resulting in a global solution when applied recursively. This dissertation will focus on using the HJB PDE and its variants.

Dynamic Programming and the Hamilton-Jacobi-Bellman PDE

The core idea of dynamic programming is simple but may not be obvious at first glance. Bellman's principle of optimality [23] states: "An optimal policy has the property that whatever the initial state and initial decisions are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decisions."

This means that if we are given the value of the system at the final time, we can recursively build up the value at each previous time step. This notion can be nonintuitive at first glance; [25] is a fantastic introduction to the concept.
Let's begin by plugging the (2.5) into (2.6):

$$V(x(t),t) = \inf_{u(\cdot)} \int_t^T c(x(\tau),u(\tau))d\tau + l(x(T)). \tag{2.12}$$

At the final time the value is equal to the terminal cost, $V(x(T),T) = l(x(T))$. Given that we know the value at the final time, it would be nice to have a recursive equation to build up to the value at some initial time t. Therefore, we would like to be able to write the value at any given time $V(x(t),t)$ as a function of some future value $V(x(t+\delta),t+\delta)$. We will work towards this by first breaking the value function down into time intervals $[t,t+\delta]$ and $[t+\delta,T]$:

$$V(x(t),t) = \inf_{u(\cdot)} \left\{ \int_t^{t+\delta} c(x(\tau),u(\tau))d\tau + \int_{t+\delta}^{T} c(x(\tau),u(\tau))d\tau + l(x(T)) \right\}, \tag{2.13}$$

adding an additional infimum inside the first in order to get the second half of the equation in the form of $V(x(t+\delta),t+\delta)$,

$$V(x(t),t) = \inf_{u(\cdot)\in\mathbb{U}_t^T} \left\{ \int_t^{t+\delta} c(x(\tau),u(\tau))d\tau + \inf_{u(\cdot)\in\mathbb{U}_{t+\delta}^T} \left[\int_{t+\delta}^{T} c(x(\tau),u(\tau))d\tau + l(x(T)) \right] \right\}. \tag{2.14}$$

allows us to replace the math in blue with in $V(x(t+\delta),t+\delta)$,

$$V(x(t),t) = \inf_{u(\cdot)} \left[\int_t^{t+\delta} c(x(\tau),u(\tau))d\tau + V(x(t+\delta),t+\delta) \right]. \tag{2.15}$$

That is, the value $V(x(t),t)$ is the value from $V(x(t+\delta),t+\delta)$ plus the cost we incur when optimizing over the horizon $[t,t+\delta]$. This is Bellman's principle of optimality (the dynamic programming principle). If δ is small, we can write this as:

$$V(x(t),t) = \inf_{u\in\mathcal{U}} \left[c(x(t),u(t))\delta + V(x(t+\delta),t+\delta) \right]. \tag{2.16}$$

We will use Taylor's expansion to expand $V(x(t+\delta),t+\delta)$:

$$V(x(t+\delta),t+\delta) \approx V(x(t),t) + D_tV(x(t),t)\delta + D_xV(x(t),t)\cdot\frac{dx}{dt}\delta \tag{2.17}$$

We can plug this into the previous equation to get:

$$V(x(t),t) = \inf_{u\in\mathcal{U}} \left[c(x(t),u(t))\delta + V(x(t),t) + D_tV(x(t),t)\delta + D_xV(x(t),t)\cdot\frac{dx}{dt}\delta \right], \tag{2.18}$$

We can move the terms that do not depend on control outside of the optimization:

$$V(x(t),t) = V(x(t),t) + D_tV(x(t),t)\delta + \inf_{u\in\mathcal{U}} \left[c(x(t),u(t))\delta + D_xV(x(t),t)\cdot\frac{dx}{dt}\delta \right], \tag{2.19}$$

Canceling redundant terms, dividing by δ (noting that $\delta > 0$), and plugging in $\frac{dx}{dt} = f(x,u)$, we have:

$$\begin{aligned} 0 = D_tV(x(t),t) + \inf_{u\in\mathcal{U}} \left[c(x(t),u(t)) + D_xV(x(t),t)\cdot f(x,u) \right], \\ V(x,T) = l(x). \end{aligned} \tag{2.20}$$

This is the differential statement of the dynamic programming principle, otherwise called the Hamilton-Jacobi-Bellman (HJB) equation. Using this, we can propagate the value function

backwards from the terminal condition $V(x(T), T) = l(x(T))$. The resulting optimal control at any particular state and time will be

$$u^*(x,t) = \arg\inf_{u \in \mathcal{U}} \left[c(x,u) + D_x V(x,t) \cdot f(x,u) \right]. \tag{2.21}$$

An example of solving an optimal control problem using the HJB equation is in Section 10.1.

Variations to the Optimal Control Problem

There are many different variations of this optimal control problem:

- Lagrange problems (running cost, no terminal cost)
- Mayer problems (terminal cost, no running cost)
- Bolza problems (both running cost and terminal cost)
- Discrete-time problems (Instead of a PDE this formulation requires the discrete Bellman equation, a.k.a. "Bellman backup")
- Robust optimal problems / differential games (includes a disturbance and/or second player. More on this in Section 2.3
- Changing control bounds over time, space

Viscosity Solutions

Note that in the derivation of the HJB equation we assumed that the value function is continuous and differentiable. This is a strong assumption that may not be true even in very simple scenarios. Fortunately, this assumption was relaxed by mathematicians Crandall and Lions in the 1980's with the introduction of a viscosity solution.

A bounded, uniformly continuous value function V is a viscosity solution to the HJB if it satisfies the HJB partial differential equation at all points that are differentiable. At non-differentiable points, V must satisfy a relaxed condition that involves bounding the value function above and below with continuously differentiable functions. For more details on viscosity solutions, please refer to Jaime Fisac's dissertation [66].

2.3 Optimal Control for Zero-Sum Differential Games

When we add disturbance to our system we can treat it as equivalent to adding a second player to our optimal control problem. In the original problem, the controller is the only "player," and their objective is to minimize the cost of the system. Now we've introduced

a second player that controls disturbance, and assume that their objective is to do the opposite–to maximize the cost of the system. This leads us to solving the robust control problem, which asks how well the controller can do even if the disturbance to the system is acting in a worst-case, adversarial fashion. The cost of our system now includes the disturbance term:

$$J(x,t,u(\cdot),d(\cdot)) = \int_t^T c(x(\tau),u(\tau),d(\tau))d\tau + l(x(T)) \tag{2.22}$$

Player Strategies

Before defining our value function, 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. For robust control problems, we generally assume that the Player 2 uses a strategy that is a map $d : \mathbb{U}(t) \to \mathbb{D}(t)$ that restricts the input signal for Player 2 as a function of what Player 1 chooses. We assume that Player 2 uses only non-anticipative strategies Γ [126], defined as follows:

$$\begin{aligned} d \in \Gamma_t^T := \{\mathcal{N} : \mathbb{U}(t) \to \mathbb{D}(t) : u(\tau) = \hat{u}(\tau) \text{ a. e. } \tau \in [t,T] \\ \Rightarrow \mathcal{N}[u](\tau) = \mathcal{N}[\hat{u}](\tau) \text{ a. e. } \tau \in [t,T]\} \end{aligned} \tag{2.23}$$

That is, Player 2 cannot respond differently to two Player 1 controls until they become different. Therefore, Player 2 cannot anticipate the future control actions of Player 1 ahead of time. Yet, in this setting, Player 2 has the advantage of factoring in Player 1's choice of input at every instant t and adapting its own accordingly (think of this as Player 2 going second in each instantaneous "round" of deciding an action). Thus, Player 2 has an *instantaneous informational advantage*. We do this because we care about worst-case scenarios, and therefore we want to make sure that we analyze the case where the disturbance (or adversary) is given the slight advantage.

One particular class of problems in which the notion of non-anticipative strategies is applicable is robust control problems, in which one wants to obtain the robust control (Player 1) with respect to the worst-case disturbance (Player 2), which can then be modeled as an adversary with the instantaneous informational advantage (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).

Formulation

Our value function for the robust control case can be written as: [2].

$$\begin{aligned} V(x,t) = \inf_{u(\cdot)} \sup_{d(\cdot)\in\Gamma_t^T} \quad & J(x,t,u(\cdot),d(\cdot)) \\ \text{subject to} \quad & \dot{x}(t) = f(x(\tau),u(\tau),d(\tau)), \quad \forall \tau \in [t,T] \\ & u(\tau)\in\mathcal{U}, \qquad d(\tau)\in\mathcal{D}. \end{aligned} \tag{2.24}$$

The notation here gets a bit tricky. At every time instant, the disturbance plays second, and thus is listed second. However, since the disturbance function over time is dependent on the control signal over time, the value function is often written with the disturbance function first so that it can set its strategy with respect to the control signal. All of these notation issues go away when we get to the actual differential form of the dynamic programming principle (below), and so I will leave the value function notation in its more intuitive and consistent formulation (i.e. disturbance listed second).

We can follow the same dynamic programming proof to arrive at what we now call the Hamilton-Jacobi-Isaacs equation:[3]

$$\begin{aligned} 0 = D_t V(x(t),t) + \inf_{u\in\mathcal{U}} \sup_{d\in\mathcal{D}} \left[c(x(t),u(t),d(t)) + D_x V(x(t),t)\cdot f(x,u,d) \right], \\ V(x,T) = l(x). \end{aligned} \tag{2.25}$$

The optimal control and optimal disturbance can be given by:

$$\begin{aligned} u^*(x,t) &= \arg\inf_{u\in\mathcal{U}} \sup_{d\in\mathcal{D}} \left[c(x,u,d) + D_x V(x,t)\cdot f(x,u,d) \right], \\ d^*(x,t) &= \inf_{u\in\mathcal{U}} \arg\sup_{d\in\mathcal{D}} \left[c(x,u,d) + D_x V(x,t)\cdot f(x,u,d) \right]. \end{aligned} \tag{2.26}$$

2.4 Reachability Introduction

Consider a system operating in an environment under the presence of disturbances. As a simple example, we can imagine a old-timey handcar along a track, the kind you see in

[2] There is some dispute over how to best represent the notation for the robust value function. I have shown it with the disturbance playing second, which matches the actual structure of the game in (2.25) for an instantaneous time step (where we allow disturbance to play second, as explained in player strategies). However, when we write the equation like this in terms of input signals over time, this formulation abuses notation somewhat, as the disturbance strategy Γ depends on the control signal $u(\cdot)$, and therefore $u(\cdot)$ would need to be declared before $d(\cdot)$ in the order of operations. Because of this, many papers switch the order of the inf and sup. This disagreement over representation does not affect the actual equation used in computation, namely (2.25).

[3] we use Isaacs instead of Bellman here because Isaac worked on dynamic programming for game theory, and thus we use his name when referring to more than one player.

cartoons as in Fig. 2.1. A person can pump the lever to accelerate the car left or right along the track. Their ability to do this will depend on some external forces, like for example the wind–if a strong wind is blowing against the person, it will be harder for them to move against it. We will treat external forces like these as disturbances to the system.

In reachability problems there is a target set $\mathcal{L}$ that is meaningful to the agent: it can be either a set of goal states (for example, a train station), or a set of unsafe states (for example, a boulder that has fallen on the tracks). Here we will give intuitive introductions to the different types of reachability problems. A more mathematically thorough description will follow in Sec. 2.5.

Typically, HJI reachability seeks to find the set of initial states for which the system, acting optimally and under worst-case disturbances, will enter the target set $\mathcal{L}$ either at a particular time (backward reachable set, or BRS as shown in Fig. 2.2a) or within a time horizon (backward reachable tube, or BRT as shown Fig. 2.2b).

Optimal behavior of the system depends on the nature of the target set and can be formulated as a differential game: for a goal set, the Player 1 (the human in this example) will seek to minimize distance to the goal whereas Player 2 (which is in this case adversarial wind) will maximize distance to the goal. For an unsafe set, the control will maximize and the disturbance will minimize. Both cases (and various combinations of cases) can be solved using HJI reachability analysis.

In our example, HJI reachability could tell us the initial states from which (under worst-case wind conditions) we are doomed to hit the boulder (unsafe set) even if we try to decelerate as much as possible, as in as in Fig. 2.2c. Alternatively, HJI reachability can also compute the initial states from which our handcar will reach the train station (goal set) even under the worst-case wind conditions (where the wind is actively blowing against us), as in Fig. 2.2d.

Another distinction between types of reachability problems is the time horizon. We already discussed the differences between a set (where we want to know if we will enter the target set at exactly time t) vs. tube (where we can to know if we will ever enter the target set within the time horizon $[t, T]$). For the tube case, we can look at finite time horizons $[t, T]$ as in Fig. 2.2e, or infinite time horizons $[t, \infty]$ as in Fig. 2.2f.

The last type of problem we will discuss is the forward reachable tube set and forward reachable tube. Here the target set is actually the initial state, and we flip the problem to ask where are all the final states that can be reached from this initial state at either a specific time (FRS) or within a time horizon (FRT). An FRT is visualized in Fig. 2.2h. Note that we call this forward reachability because we analyze the system forward in time instead of backwards.

Note that there could be cases with both goals and obstacles. This leads us to reach-avoid problems, wherein we solve for the set of initial states from which you are guaranteed to reach the goal while avoiding the obstacles in the process.

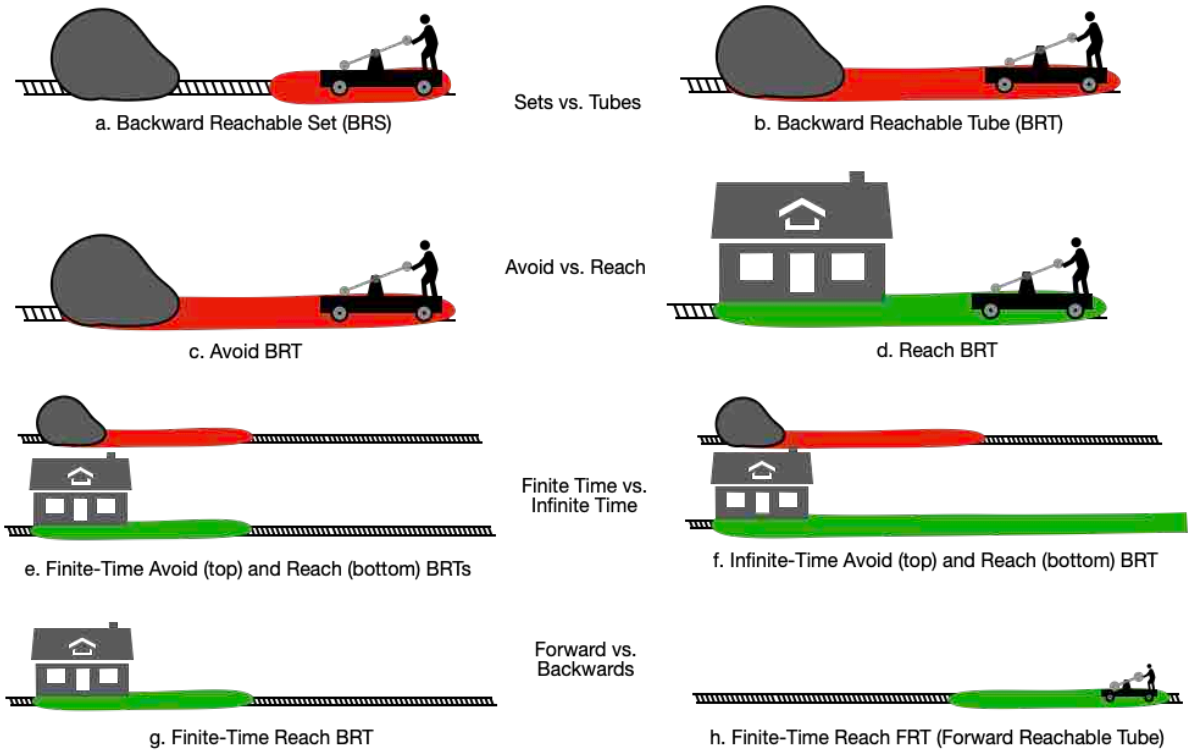


Figure 2.2: Types of reachability problems. **Sets vs. Tubes:** (a) Backward reachable set (BRS) indicating states from which the system will hit the boulder at exactly the end of the time horizon. (b) Backward reachable tube (BRT) indicating states from which the system will hit the boulder at any point in the time horizon. **Avoid vs. Reach:** (c) Avoid BRT, same as (b). (d) Reach BRT, indicating states from which the system is guaranteed to reach a goal at any point within the time horizon. **Finite vs. Infinite Time:** (e) BRTs with a finite time horizon. (f) BRTs with an infinite-time horizon. Note that avoid BRTS tend to converge, and reach BRTs tend to continue indefinitely. **Backward vs. Forward:** (g) a BRT, as shown in all sub-figures up until this point. Here the reachability problem propagates backwards in time from goal or unsafe set until the time horizon. (h) a Forward Reachable Tube (FRT), where the reachability problem propagates forward in time from the initial condition. This instead computes the set of states that the system can reach within the time horizon.

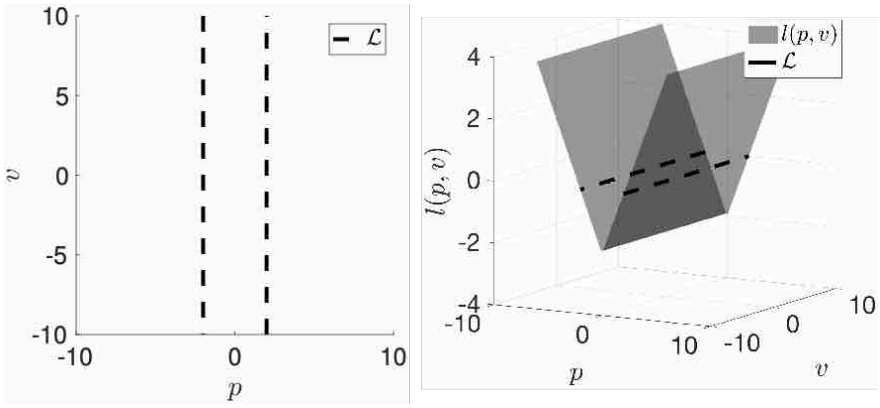


Figure 2.3: Visualization of the boundaries of a target set $\mathcal{L}$ (can be either a goal set or an unsafe set, depending on formulation) and corresponding target function $l(p, v)$ for the running example. Here the states p, v are position and velocity. In this case the target set defines a set centered at the origin in position space p and across all velocity space v.

2.5 Defining the Reachability Problem

This section formally defines the types of reachability problems visualized in Fig. 2.2, and constructs the value functions that correspond to each type of reachability problem. These value functions, once solved for, provide the solutions to the reachability problems.

Avoid Problems (Safety)

Avoid Backward Reachable Set

In an avoid problem, the target set $\mathcal{L}$ represents the unsafe states. Let us first assume that we want to solve a BRS, defined as:

$$\mathcal{V}_{\text{BRS}}(t) = \{x : \exists d(\cdot) \in \Gamma_t^T, \forall u(\cdot) \in \mathbb{U}_t^T, \zeta_{x,t}^{u,d}(T) \in \mathcal{L}\}, \tag{2.27}$$

where $\Gamma(\cdot)$ in (2.27) denotes the feasible set of strategies for Player 2. This is the set of all states x for which there exists a strategy for Player 2 for all inputs of Player 1 such that the trajectory starting at time t and state x will enter the target set $\mathcal{L}$ at *exactly* the end of the time horizon (at time T).

The differential game that must be solved in order to compute the BRS is a "game of kind" rather than a "game of degree", i.e., games in which the outcome is determined by *whether or not* the state of the system reaches a given configuration under specified constraints at any time within the duration of the game. 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.

To solve for this BRS, we first define a target function $l(x)$ whose sub-zero level set is the target set, i.e. $\mathcal{L} = \{x : l(x) \leq 0\}$. This can be considered as a reward function that

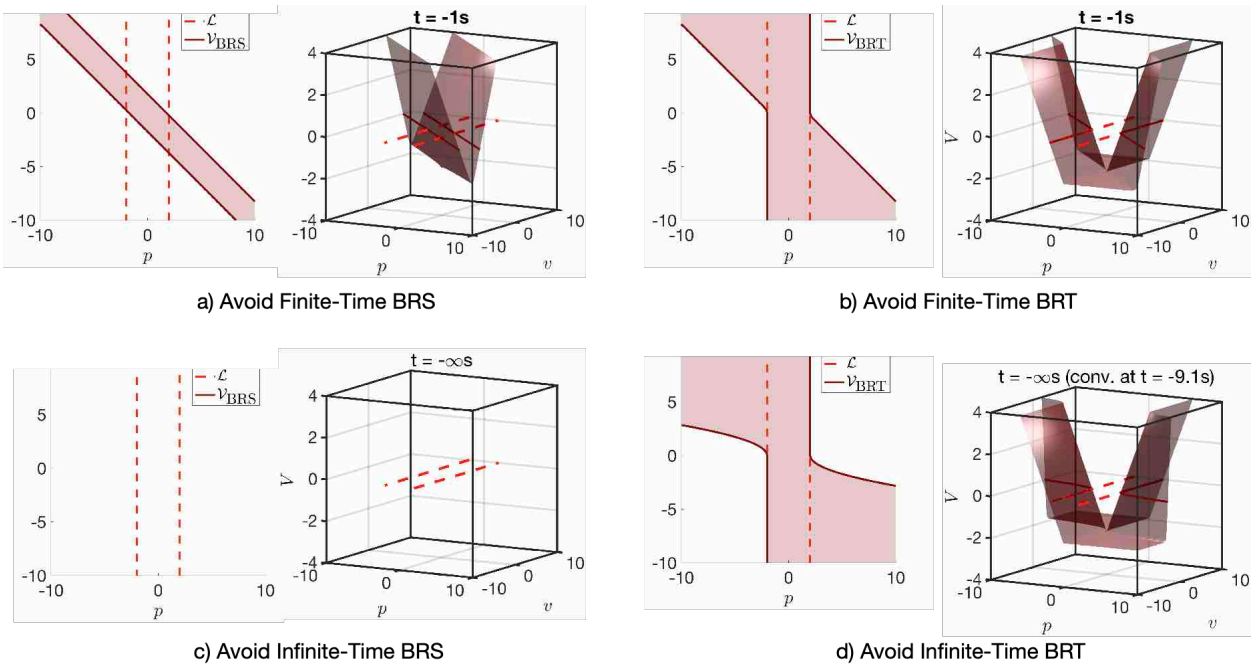


Figure 2.4: Avoid problem visualizations for the double integrator example. In all figures, the red dotted lines indicate the bounds of the target set $\mathcal{L}$ (which is here the unsafe set), defined as an obstacle centered at the origin in position space and across all velocities. Note that to reduce visual clutter the target function $l(x)$ is not visualized here. The dark red function represents the value function, and the corresponding BRS/BRT is shown as the subzero level set of that function (also in dark red). (a) Avoid finite-time BRS with a time horizon of $t = -1s$. (b) Avoid finite-time BRT with the same time horizon. (c) Avoid infinite-time BRS. Note that the avoid set vanishes over time. This matches our intuition–given enough time, the system can either decelerate enough to not enter the obstacle at the end of the time horizon *or* accelerate through the obstacle before the end of the time horizon. (d) Avoid infinite-time BRT. This captures all trajectories that ever enter the obstacle within infinite time, even if they are able to then exit the obstacle after. Note that it has converged at $t = -9.1s$. Trajectories from all states outside this BRT will be able to decelerate enough to not enter the unsafe set. The videos associated with these computations can be seen at `https://youtu.be/f_I1QmDuxpk`

must be (a) lipschitz continuous, and (b) positive outside of the unsafe set and negative inside. Typically $l(x)$ is defined as a signed distance function that measures distance to $\mathcal{L}$, as visualized in Fig. 2.3.

To compute the BRS we seek to capture the reward of the trajectories of the system at the end of the time horizon, i.e.

$$J_{\text{BRS}}(x, t, u(\cdot), d(\cdot)) = l(\zeta_{x,t}^{u,d}(T)), \ t \leq T. \tag{2.28}$$

The corresponding value function will optimize for the optimal control signal that maximizes the reward (and drives the system away from the unsafe target set) and the worst-case disturbance signal that minimizes the reward:

$$V_{\text{BRS}}(x,t) = \sup_{u(\cdot)} \inf_{d(\cdot)} \Big\{ J_{\text{BRS}}\Big(x,t,u(\cdot),d(\cdot)\Big)\Big\}. \tag{2.29}$$

This value function can be computed using the HJI PDE (2.25). Level sets of the value function correspond to level sets of the target (reward) function. If a state has a negative value, optimal trajectories starting from that state achieve negative reward at the end of the trajectory Therefore, once the value function is computed, the BRS is the subzero level set of this function:

$$\mathcal{V}_{\text{BRS}}(t) = x : \{V_{\text{BRS}}(x,t) \leq 0\} \tag{2.30}$$

Avoid Backward Reachable Tube

If instead we wish to solve a BRT, then we seek to find all trajectories that will enter $\mathcal{L}$ *at any point* in the time horizon, and therefore become unsafe. This is often the desired computation in avoid problems, as one cares about whether the system *ever* entered the unsafe set. This is as compared to the BRS setting, where trajectories that enter and then leave the unsafe set before the end of the time horizon are considered safe. The BRT is defined as:

$$\mathcal{V}_{\text{BRT}}(t) = \{x : \exists d \in \Gamma_t^T, \forall u(\cdot) \in \mathbb{U}_t^T, \exists \tau \in [t,T] \;\; \zeta_{x,t}^{u,d}(\tau) \in \mathcal{L}\}. \tag{2.31}$$

The BRT is computed by finding the *minimum reward* (and therefore minimum distance to $\mathcal{L}$) over time, leading to the cost function:

$$J_{\text{BRT}}(x,t,u(\cdot),d(\cdot)) = \min_{\tau \in [t,T]} l(\zeta_{x,t}^{u,d}(\tau)), \;\; t \leq T. \tag{2.32}$$

The corresponding value function will once again optimize for the optimal control signal that maximizes the reward (and drives the system away from the unsafe target set) and the worst-case disturbance signal that minimizes the reward:

$$V_{\text{BRT}}(x,t) = \sup_{u(\cdot)} \inf_{d(\cdot)} \Big\{ J_{\text{BRT}}\Big(x,t,u(\cdot),d(\cdot)\Big)\Big\}. \tag{2.33}$$

Computing this value function for the BRT case is more complicated than simply applying the HJI PDE due to the minimization over time. In section 2.6, we will derive the new update equation that solves for this value function. For now, know that intuitively this involves updating the value function using the HJI PDE, and then minimizing the new value function with the original target function $l(x)$ to capture the minimum reward over that time instant. This process repeats until the value function for the initial time has been reached.

The BRT is the subzero level set of the value function:

$$\mathcal{V}_{\text{BRT}}(t) = x : \{V_{\text{BRT}}(x,t) \leq 0\} \tag{2.34}$$

In the BRS case a negative value meant that optimal trajectories starting from that state achieve negative reward at the end of the trajectory (i.e. the system is in the obstacle at the end of the trajectory). Conversely, with a BRT a negative value means that optimal trajectories starting from that state achieve negative reward at *some point during the time horizon* (i.e. the system has entered the obstacle *at some point* during $[t, T]$). Typically we care about BRTs more when doing avoid problems, since we care if the system *ever* entered the obstacle, not if the system happens to be in the obstacle at exactly the end of the time horizon.

Reach Problem (Goal-Satisfaction)

A reach problem is the opposite of the avoid problem. Here, the target set $\mathcal{L}$ represents the goal set, and is defined the same way: $\mathcal{L} = \{x : l(x) \leq 0\}$. One can think of the target function as a cost function instead of a reward function. In this case we seek to identify the states for which Player 1 can enter the goal set (i.e. achieve negative cost) despite the worst-case efforts of Player 2. Therefore, the BRS and BRT are defined as:

$$\mathcal{V}_{\text{BRS}}(t) = \{x : \exists u(\cdot) \in \mathbb{U}_t^T, \forall d \in \Gamma_t^T, \zeta_{x,t}^{u,d}(T) \in \mathcal{L}\}, \tag{2.35}$$

$$\mathcal{V}_{\text{BRT}}(t) = \{x : \exists u(\cdot) \in \mathbb{U}_t^T, \forall d \in \Gamma_t^T, \exists \tau \in [t,T] \zeta_{x,t}^{u,d}(T) \in \mathcal{L}\}. \tag{2.36}$$

The cost function J_{BRS} and J_{BRT} remain the same as in (2.28),(2.32):

$$J_{\text{BRS}}(x,t,u(\cdot),d(\cdot)) = l(\zeta_{x,t}^{u,d}(T)),\ t \leq T. \tag{2.37}$$

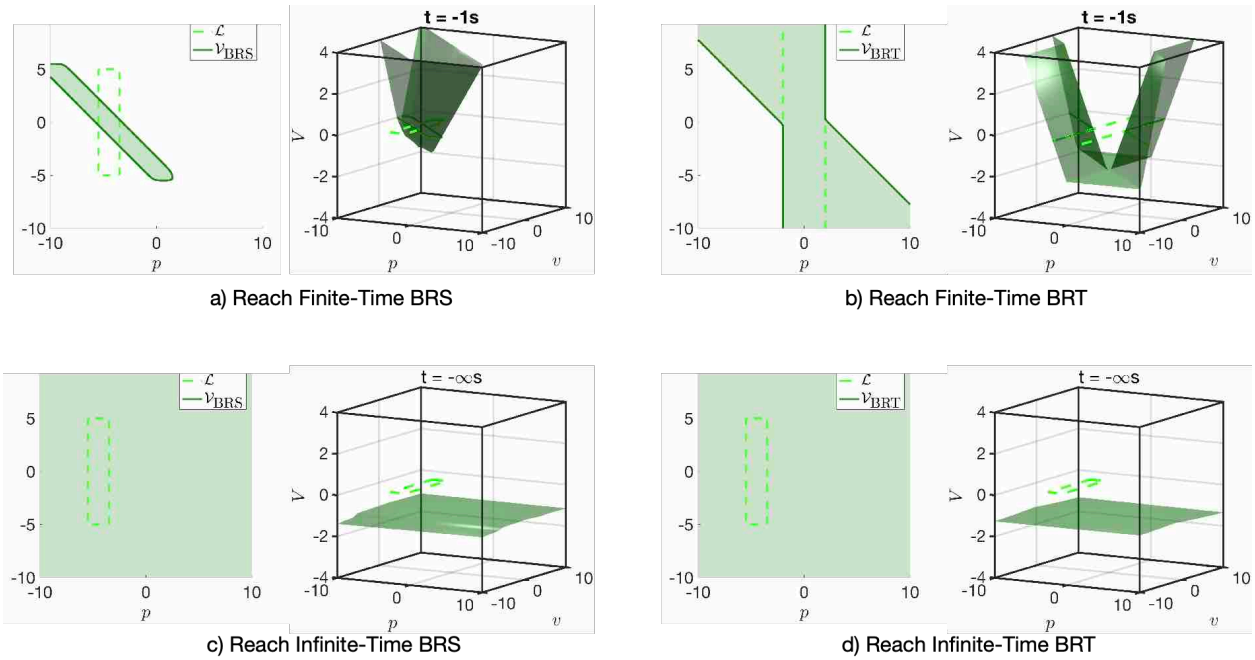
$$J_{\text{BRT}}(x,t,u(\cdot),d(\cdot)) = \min_{\tau \in [t,T]} l(\zeta_{x,t}^{u,d}(\tau)),\ t \leq T. \tag{2.38}$$

The value function now flips the direction of optimization for both players (because now Player 1 seeks to minimize the cost and Player 2 seeks to do the opposite):

$$V_{\text{BRS}}(x,t) = \inf_{u(\cdot)} \sup_{d(\cdot)} \Big\{ J_{\text{BRS}}\Big(x,t,u(\cdot),d(\cdot)\Big) \Big\}, \tag{2.39}$$

$$V_{\text{BRT}}(x,t) = \inf_{u(\cdot)} \sup_{d(\cdot)} \Big\{ J_{\text{BRT}}\Big(x,t,u(\cdot),d(\cdot)\Big) \Big\}. \tag{2.40}$$

Once again, level sets of the value function correspond to level sets of the target function, as visualized in Fig. 2.5. If a state has a negative value, optimal trajectories starting from that state achieve negative cost at either the end of the trajectory (for BRS) or at some point



a) Reach Finite-Time BRS

b) Reach Finite-Time BRT

c) Reach Infinite-Time BRS

d) Reach Infinite-Time BRT

Figure 2.5: Reach problem visualizations for the double integrator example. In all figures, the green dotted lines indicate the bounds of the target set $\mathcal{L}$ (which is here the goal set), defined as a goal that is left of the origin and must be achieved at a specific range of velocities ± 5. Note that to reduce visual clutter the target function $l(x)$ is not visualized here. The dark green function represents the value function, and the corresponding BRS/BRT is shown as the subzero level set of that function (also in dark green). (a) Reach finite-time BRS with a time horizon of $t = -1s$. (b) Reach finite-time BRT with the same time horizon. (c) Reach infinite-time BRS. Note that the reach set eventually encompasses the entire state space. This matches our intuition–given enough time, the system can eventually reach the goal. (d) Reach infinite-time BRT, which in this case is identical to the BRS. The videos associated with these computations can be seen at `https://youtu.be/f_I1QmDuxpk`

in the trajectory (for BRT), meaning they have entered the target set $\mathcal{L}$ sometime within the time horizon. Therefore, the sub-zero level set of the value function comprises the BRS or the BRT, depending on the formulation:

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

$$\mathcal{V}_{\text{BRT}}(t) = x : \{V_{\text{BRT}}(x,t) \leq 0\} \tag{2.42}$$

Reach-Avoid Problem

What happens when there are both goal and unsafe states? This brings us to a reach-avoid problem. Here we will define the goal set(s) using the same target set formulation: $\mathcal{L} = \{x : l(x) \leq 0\}$. We will then define a second function to represent the unsafe set(s):

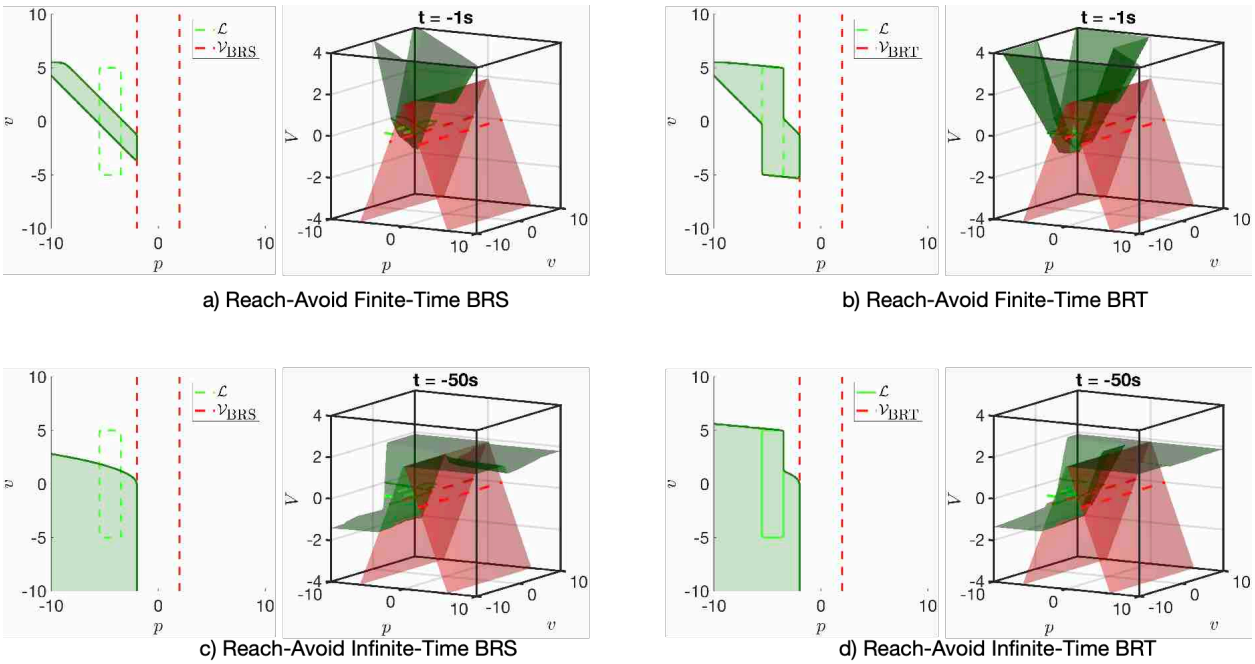


Figure 2.6: Reach-Avoid problem visualizations for the double integrator example. In all figures, the green dotted lines indicate the bounds of the target set $\mathcal{L}$ (which is here the goal set), and the red dotted lines are the bounds of the obstacle set $\mathcal{G}$. The target function $l(x)$ is not visualized here, but the obstacle function $g(x)$ is shown in red. The dark green function represents the value function, and the corresponding BRS/BRT is shown as the subzero level set of that function (also in dark green). (a) Reach-Avoid finite-time BRS with a time horizon of $t = -1s$. (b) Reach-Avoid finite-time BRT with the same time horizon. (c) Reach-Avoid infinite-time BRS. (d) Reach-Avoid infinite-time BRT. The videos associated with these computations can be seen at `https://youtu.be/f_I1QmDuxpk`

$\mathcal{G} = \{x : g(x) \geq 0\}$[4]. Note that this function is positive inside the unsafe set and negative outside.

The BRS in this case are the set of states such that the system is able to enter a goal set at exactly the end of the time horizon without entering any unsafe sets along the way. The BRT are the set of states that enter a goal set at some point in the time horizon without entering any unsafe sets along the way:

$$\begin{aligned}\mathcal{V}_{\text{BRS}}(t) = \{x(t) : \exists u \in \mathbb{U}_t^T, \forall d \in \Gamma_t^T, \\ \zeta_{x,t}^{u,d}(T) \in \mathcal{L} \wedge \forall \tau \in [t, T] \zeta_{x,t}^{u,d}(\tau) \in \mathcal{G}^{\complement}\}.\end{aligned} \tag{2.43}$$

$$\begin{aligned}\mathcal{V}_{\text{BRT}}(t) = \{x(t) : \exists u \in \mathbb{U}_t^T, \forall d \in \Gamma_t^T, \\ \exists \tau \in [t, T] \zeta_{x,t}^{u,d}(\tau) \in \mathcal{L} \wedge \forall s \in [t, \tau] \zeta_{x,t}^{u,d}(s) \in \mathcal{G}^{\complement}\}.\end{aligned} \tag{2.44}$$

[4]I understand the frustration of using a function called $g(x)$ to represent constraints rather than goals. To maintain consistency with [73] I am keeping the notation the same.

The cost functionals for the reach-avoid case are:

$$J_{\text{BRS}}(x,t,u(\cdot),d(\cdot)) = \max\Big\{l(\zeta_{x,t}^{u,d}(T),T), \max_{\tau\in[t,T]} g(\zeta_{x,t}^{u,d}(\tau),\tau)\Big\}. \tag{2.45}$$

$$J_{\text{BRT}}(x,t,u(\cdot),d(\cdot)) = \min_{\tau\in[t,T]} \max\Big\{l(\zeta_{x,t}^{u,d}(\tau),\tau), \max_{s\in[t,\tau]} g(\zeta_{x,t}^{u,d}(s),s)\Big\}. \tag{2.46}$$

The maximizations ensure that a positive cost of $g(x)$ overrides a negative cost of $l(x)$, i.e. trajectories that are able to reach the goal only by entering unsafe states are still counted as unsafe. The corresponding value functions are the same as in the reach case:

$$V_{\text{BRS}}(x,t) = \inf_{u(\cdot)} \sup_{d(\cdot)} \Big\{ J_{\text{BRS}}\Big(x,t,u(\cdot),d(\cdot)\Big)\Big\}, \tag{2.47}$$

$$V_{\text{BRT}}(x,t) = \inf_{u(\cdot)} \sup_{d(\cdot)} \Big\{ J_{\text{BRT}}\Big(x,t,u(\cdot),d(\cdot)\Big)\Big\}. \tag{2.48}$$

Solving for these value functions is non-trivial, and a derivation is shown in Section 2.6. The BRS and BRT are once again the subzero level set:

$$\mathcal{V}_{\text{BRS}}(t) = x : \{V_{\text{BRS}}(x,t) \le 0\} \tag{2.49}$$

$$\mathcal{V}_{\text{BRT}}(t) = x : \{V_{\text{BRT}}(x,t) \le 0\} \tag{2.50}$$

These value functions and sets for the reach-avoid case are visualized in Fig. 2.6.

Time-Varying Goals and Unsafe Sets

One of the advantages of the HJ formulation used in this dissertation over others (such as in [126]) is that this formulation can handle time-varying goal sets and unsafe sets, as visualized in Fig. 2.7. For such a case, the target and obstacle functions are also a function of t, i.e. $l(x,t)$ and $g(x,t)$. Otherwise the equations remain the same. For the proof and more information, see [73]. Another advantage of this formulation is its amenability to warm-start reachability, as shown in Chapter 5.

Forward Reachability

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 either within a time horizon $[T,t], t \ge T$ for an FRT, or at exactly the end of the time horizon for an FRS. These are formally defined as:

$$\begin{aligned}\mathcal{V}_{\text{FRS}}(t) =& \{y : \exists u(\cdot) \in \mathbb{U}_t^T, \forall d \in \Gamma_t^T, x \in \mathcal{L}, \\ & \zeta_{x,T}^{u,d}(t) = y\}, t > T,\end{aligned} \tag{2.51}$$

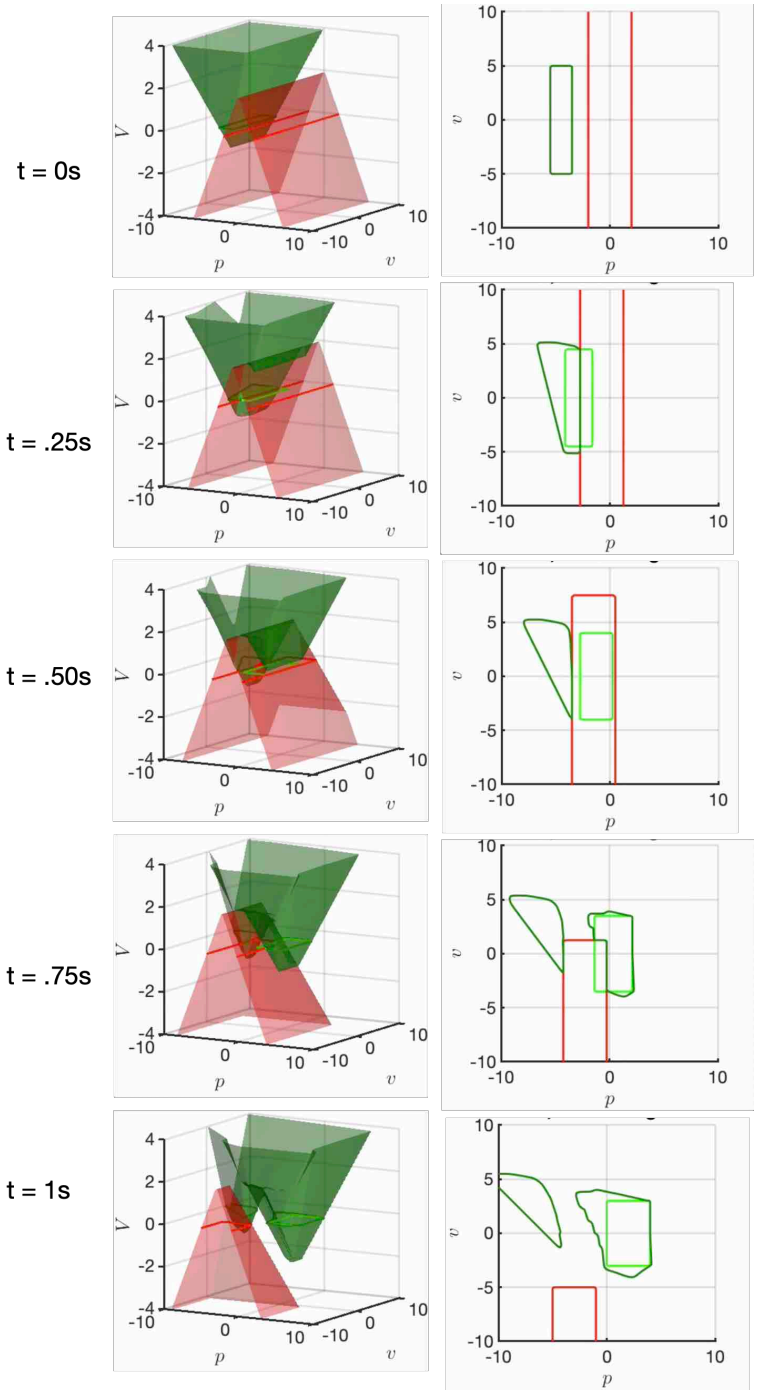


Figure 2.7: Time-varying goals and obstacles. Here, the dark green set and function represent a time-varying reach-avoid finite-time BRT.

$$\begin{aligned}\mathcal{V}_{\mathrm{FRT}}(t) =&\{y : \exists u(\cdot)_t^T \in \mathbb{U}, \forall d \in \Gamma_t^T, x \in \mathcal{L}, \\ &\exists \tau \in [T, t] \quad \zeta_{x,T}^{u,d[u]}(\tau) = y\}, t > T.\end{aligned} \tag{2.52}$$

Here, $\mathcal{L}$ represents the set of initial states of the system. $\mathcal{V}_{\mathrm{FRS}}(t)$ is the set of all states that system can reach at exactly t, and $\mathcal{V}_{\mathrm{FRT}}(t)$ is the set of all states that system can reach within the time horizon ending at t. Equations (2.51) and (2.52) are assuming the "reach" case where Player 1 is trying to reach as many states as possible and Player 2 is doing the opposite. This can be flipped easily as above for the case where Player 1 attempts to stay in the initial set and Player 2 works to drive the system away from the initial set:

$$\begin{aligned}\mathcal{V}_{\mathrm{FRS}}(t) =&\{y : \exists d \in \Gamma(t), \forall u(\cdot) \in \mathbb{U}, x \in \mathcal{L}, \\ &\zeta_{x,T}^{u,d[u]}(t) = y\}, t > 0,\end{aligned} \tag{2.53}$$

$$\begin{aligned}\mathcal{V}_{\mathrm{FRT}}(t) =&\{y : \exists d \in \Gamma(t), \forall u(\cdot) \in \mathbb{U}, x \in \mathcal{L}, \\ &\exists \tau \in [T, t] \quad \zeta_{x,T}^{u,d[u]}(\tau) = y\}, t > 0.\end{aligned} \tag{2.54}$$

The FRS can be computed in a similar fashion as the BRS. The only difference is that an initial value HJB PDE needs to be solved instead of a final value PDE, which can always be converted into an equivalent final value PDE by change of variables [63]. More details on the computation of FRS and some of their concrete applications can be found in [39, 127].

A Note about Reachability with Running Cost

In optimal control problems, an objective can have both a running cost and a terminal cost. However, having a mixed objective like this means that the optimization may value a trajectory with a very low running cost that violates the terminal conditions. In reachability problems, this terminal condition should never be violated, otherwise the subzero level set of the value function is no longer guaranteed to reach (avoid) the target set. Therefore, reachability problems are solved without a running cost. In the reinforcement learning this issue is typically handled by putting a large weight on the terminal cost in the objective, but in practice this still does not lead to guarantees.

2.6 Computing the Reachability Problem

We have derived the cost for each reachability problem and its corresponding value function. We will now explain how to use dynamic programming to obtain the correct update equation in each case that will solve for the value function. A handy summary of these equations are in Table 2.1

Table 2.1: Set Notation, Value Functions, and Update Equations for Reachability Problems

Reach BRS	BRS (2.35):	$\mathcal{V}_{\text{BRS}}(t) = \{x : \exists u(\cdot) \in \mathbb{U}_t^T, \forall d \in \Gamma_t^T, \zeta_{x,t}^{u,d}(T) \in \mathcal{L}\}$
	Value Function (2.39):	$V_{\text{BRS}}(x,t) = \inf_{u(\cdot)} \sup_{d(\cdot)} l(\zeta_{x,t}^{u,d}(T))$
	HJI PDE (2.55):	$0 = D_t V + \inf_u \sup_d D_x V \cdot f(x,u,d)$
Avoid BRS	BRS (2.27):	$\mathcal{V}_{\text{BRS}}(t) = \{x : \exists d(\cdot) \in \Gamma_t^T, \forall u(\cdot) \in \mathbb{U}_t^T, \zeta_{x,t}^{u,d}(T) \in \mathcal{L}\}$
	Value Function (2.29):	$V_{\text{BRS}}(x,t) = \sup_{u(\cdot)} \inf_{d(\cdot)} l(\zeta_{x,t}^{u,d}(T))$
	HJI PDE (2.55):	$0 = D_t V + \sup_u \inf_d D_x V \cdot f(x,u,d)$
Reach BRT	BRT (2.36):	$\mathcal{V}_{\text{BRT}}(t) = \{x : \exists u(\cdot) \in \mathbb{U}_t^T, \forall d \in \Gamma_t^T,$ $\exists \tau \in [t,T] \zeta_{x,t}^{u,d}(T) \in \mathcal{L}\}$
	Value Function (2.40):	$V_{\text{BRS}}(x,t) = \inf_{u(\cdot)} \sup_{d(\cdot)} \min_{\tau \in [t,T]} l(\zeta_{x,t}^{u,d}(\tau))$
	Variational Inequality (2.65):	$0 = \min\Big\{l(x,t) - V(x,t),$ $D_t V + \inf_u \sup_d D_x V \cdot f(x,u)\Big\}$
Avoid BRT	BRT (2.31):	$\mathcal{V}_{\text{BRT}}(t) = \{x : \exists d \in \Gamma_t^T, \forall u(\cdot) \in \mathbb{U}_t^T,$ $\exists \tau \in [t,T] \ \ \zeta_{x,t}^{u,d}(\tau) \in \mathcal{L}\}$
	Value Function (2.33):	$V_{\text{BRS}}(x,t) = \sup_{u(\cdot)} \inf_{d(\cdot)} \min_{\tau \in [t,T]} l(\zeta_{x,t}^{u,d}(\tau))$
	Variational Inequality (2.65):	$0 = \min\Big\{l(x,t) - V(x,t),$ $D_t V + \sup_u \inf_d D_x V \cdot f(x,u)\Big\}$
Reach-Avoid BRS	BRS (2.43):	$\mathcal{V}_{\text{BRS}}(t) = \{x(t) : \exists u \in \mathbb{U}_t^T, \forall d \in \Gamma_t^T,$ $\zeta_{x,t}^{u,d}(T) \in \mathcal{L} \wedge \forall \tau \in [t,T] \zeta_{x,t}^{u,d}(\tau) \in \mathcal{G}^{\complement}\}$
	Value Function (2.47):	$V_{\text{BRS}}(x,t) = \inf_{u(\cdot)} \sup_{d(\cdot)} \max\Big\{$ $l(\zeta_{x,t}^{u,d}(T), T), \max_{\tau \in [t,T]} g(\zeta_{x,t}^{u,d}(\tau), \tau)\Big\}$
	Modified Double-Obstacle Isaacs (2.83):	$0 = \max\Big\{D_t V + \inf_u \sup_d D_x V \cdot f(x,u),$ $g(x,t) - V(x,t)\Big\}$
Reach-Avoid BRT	BRT (2.44):	$\mathcal{V}_{\text{BRT}}(t) = \{x(t) : \exists u \in \mathbb{U}_t^T, \forall d \in \Gamma_t^T,$ $\exists \tau \in [t,T] \zeta_{x,t}^{u,d}(\tau) \in \mathcal{L} \wedge \forall s \in [t,\tau] \zeta_{x,t}^{u,d}(s) \in \mathcal{G}^{\complement}\}$
	Value Function (2.48):	$V_{\text{BRT}}(x,t) = \inf_{u(\cdot)} \sup_{d(\cdot)} \min_{\tau \in [t,T]} \max\Big\{$ $l(\zeta_{x,t}^{u,d}(\tau), \tau), \max_{s \in [t,\tau]} g(\zeta_{x,t}^{u,d}(s), s)\Big\}$
	Variational Inequality (Double-Obstacle Isaacs Equation) (2.80):	$0 = \max\Big\{ \min\Big\{l(x,t) - V(x,t), D_t V$ $+ \inf_u \sup_d D_x V \cdot f(x,u)\Big\}, g(x,t) - V(x,t)\Big\}$

Solving the Value Function for a Reach (or Avoid) BRS

To solve a Reach BRS problem we solve the value function (2.39) using the same process described in the optimal control section, resulting in a modified version of the HJI PDE (2.25) with no running cost:

$$\boxed{\begin{aligned} 0 = D_t V(x(t),t) + \inf_{u \in \mathcal{U}} \sup_{d[u] \in \Gamma} \left[D_x V(x(t),t) \cdot f(x,u,d) \right], \\ V(x,T) = l(x). \end{aligned}} \tag{2.55}$$

For the Avoid BRS value function (2.29), we simply switch the optimization so that the control is maximizing and the disturbance is minimizing. Examples of using the HJI PDE to solve for a reach BRS (and avoid BRS) are in 10.1.

Solving the Value Function for a Reach (or Avoid) BRT

For visual ease we will assume no disturbance in our derivation, but the derivation follows in the same way when disturbance is included. Starting with the value function for a Reach BRT (2.40):

$$V(x(t),t) = \inf_{u(\cdot) \in \mathbb{U}_t^T} \min_{\tau \in [t,T]} l(x(\tau),\tau), \tag{2.56}$$

to get a recursive equation we can break this into $[t, t+\delta]$ and $[t+\delta, T]$:

$$V(x(t),t) = \inf_{u(\cdot) \in \mathbb{U}_t^T} \min \left\{ \min_{\tau \in [t,t+\delta]} l(x(\tau),\tau), \min_{\tau \in [t+\delta,T]} l(x(\tau),\tau) \right\}. \tag{2.57}$$

Adding an extra infimum allows us to get to the formula for $V(x(t+\delta), t+\delta)$:

$$V(x(t),t) = \inf_{u(\cdot) \in \mathbb{U}_t^T} \min \left\{ \min_{\tau \in [t,t+\delta]} l(x(\tau),\tau), \inf_{u(\cdot) \in \mathbb{U}_{t+\delta}^T} \min_{\tau \in [t+\delta,T]} l(x(\tau),\tau) \right\}, \tag{2.58}$$

$$V(x(t),t) = \inf_{u(\cdot) \in \mathbb{U}_t^T} \min \left\{ \min_{\tau \in [t,t+\delta]} l(x(\tau),\tau), V(x(t+\delta),t+\delta) \right\}. \tag{2.59}$$

For a small δ, we can approximate this as

$$V(x(t),t) = \inf_{u} \min \left\{ l(x(t),t), V(x(t+\delta),t+\delta) \right\}. \tag{2.60}$$

Next, plug in the Taylor expansion for $V(x(t+\delta),t+\delta)$ like we did in Section 2.2:

$$V(x(t),t) = \inf_{u} \min \left\{ l(x(t),t), V(x(t),t) + D_t V(x(t),t)\delta + D_x V(x(t),t) \cdot f(x,u)\delta \right\}. \tag{2.61}$$

Apply the infimum over the control to both sides of the minimization

$$V(x(t),t) = \min\left\{\inf_u l(x(t),t), \inf_u \left[V(x(t),t) + D_t V(x(t),t)\delta + D_x V(x(t),t)\cdot f(x,u)\delta\right]\right\}, \tag{2.62}$$

and pull out any terms that don't rely on control

$$V(x(t),t) = \min\left\{l(x(t),t), V(x(t),t) + D_t V(x(t),t)\delta + \inf_{u(\cdot)} D_x V(x(t),t)\cdot f(x,u)\delta\right\}. \tag{2.63}$$

Subtracting $V(x(t),t)$ on both sides:

$$0 = \min\left\{l(x(t),t) - V(x(t),t), \left[D_t V(x(t),t) + \inf_{u(\cdot)} D_x V(x(t),t)\cdot f(x,u)\right]\delta\right\}. \tag{2.64}$$

Note that this results in two possible cases:

- The left hand side of the minimization will be active, forcing $l(x(t),t) = V(x(t),t)$, and the right hand side will be greater than or equal to zero, or
- The right hand side of the minimization will be active, resulting in the HJI PDE, and the left hand side will be greater than or equal to zero.

Given that the δ is always positive, scaling the RHS by a positive number doesn't affect the this minimization comparison. Therefore we can remove the delta, resulting in the Hamilton Jacobi Isaacs Variational Inequality (HJI VI):

$$\boxed{0 = \min\left\{l(x(t),t) - V(x(t),t), D_t V(x(t),t) + \inf_{u(\cdot)} D_x V(x(t),t)\cdot f(x,u)\right\}.} \tag{2.65}$$

This is called a variational inequality because the HJI PDE must evolve in a way that respects the inequality constraint that the value function cannot be greater than $l(x(t),t)$.

A more convenient form of the HJI VI is in the form of dynamic programming. We add $V(x(t),t)$ back to both sides and note the partial Taylor's expansion $V(x(t),t) + D_t V(x(t),t)\delta = V(x(t),t+\delta)$,

$$V(x(t),t) = \min\left\{l(x(t),t), V(x(t),t) + D_t V(x(t),t)\delta + \inf_{u(\cdot)} D_x V(x(t),t)\cdot f(x,u)\delta\right\} \tag{2.66}$$

$$\boxed{V(x(t),t) = \min\left\{l(x(t),t), V(x(t),t+\delta) + \inf_{u(\cdot)} D_x V(x(t),t)\cdot f(x,u)\delta\right\}} \tag{2.67}$$

This modified form of the HJI VI in (2.67) is used to recursively update the value function at each instant in time to obtain the reach BRT. The avoid BRT value function (2.33) can be solved similarly with a supremum over control. Examples of using the HJI VI to solve for a reach BRT (and avoid BRT) are in 10.1.

Solving the Value Function for Reach-Avoid Problems

With two optimal control derivations under our belt we're ready to solve for the update equation for a reach-avoid BRT. We're going to revert back to using trajectory notation for this derivation due to the complicated time intervals. Recall that $\zeta_{x,t}^{u,d}(t) = x(t)$ because it refers to a trajectory that begins at state x and time t and ends at the same time t. Starting with the value function,

$$V(x(t),t) = \inf_{u(\cdot)} \min_{\tau\in[t,T]} \max \left\{ l(\zeta_{x,t}^{u,d}(\tau),\tau), \max_{s\in[t,\tau]} g(\zeta_{x,t}^{u,d}(s),s) \right\}. \tag{2.68}$$

We will break this into time intervals $[t, t+\delta]$ and $[t+\delta, T]$:

$$\begin{aligned} V(x(t),t) = \inf_{u(\cdot)} \min \Bigg\{ & \min_{\tau\in[t,t+\delta]} \max \left[l(\zeta_{x,t}^{u,d}(\tau),\tau), \max_{s\in[t,\tau]} g(\zeta_{x,t}^{u,d}(s),s) \right], \\ & \min_{\tau\in[t+\delta,T]} \max \left[l(\zeta_{x,t}^{u,d}(\tau),\tau), \max_{s\in[t,\tau]} g(\zeta_{x,t}^{u,d}(s),s) \right] \Bigg\}. \end{aligned} \tag{2.69}$$

To get this in terms of $V(x(t+\delta), t+\delta)$ we need the maximum over the second g to be over $t+\delta, \tau$, so we split the equation into:

$$\begin{aligned} V(x(t),t) = \inf_{u(\cdot)} \min \Bigg\{ & \min_{\tau\in[t,t+\delta]} \max \left[l(\zeta_{x,t}^{u,d}(\tau),\tau), \max_{s\in[t,\tau]} g(\zeta_{x,t}^{u,d}(s),s) \right], \\ & \min_{\tau\in[t+\delta,T]} \max \left[l(\zeta_{x,t}^{u,d}(\tau),\tau), \max_{s\in[t,t+\delta]} g(\zeta_{x,t}^{u,d}(s),s), \max_{s\in[t+\delta,\tau]} g(\zeta_{x,t+\delta}^{u,d}(s),s) \right] \Bigg\}. \end{aligned} \tag{2.70}$$

We then rearrange the maximization terms to approach the formula for $V(x(t+\delta), t+\delta)$:

$$\begin{aligned} V(x(t),t) = \inf_{u(\cdot)} \min \Bigg\{ & \min_{\tau\in[t,t+\delta]} \max \left[l(\zeta_{x,t}^{u,d}(\tau),\tau), \max_{s\in[t,\tau]} g(\zeta_{x,t}^{u,d}(s),s) \right], \\ \max \Bigg\{ & \min_{\tau\in[t+\delta,T]} \max \left[l(\zeta_{x,t}^{u,d}(\tau),\tau), \max_{s\in[t+\delta,\tau]} g(\zeta_{x,t+\delta}^{u,d}(s),s) \right], \max_{s\in[t,t+\delta]} g(\zeta_{x,t}^{u,d}(s),s) \Bigg\} \Bigg\}. \end{aligned} \tag{2.71}$$

Splitting up the infimum over u brings us finally to the formula for $V(x(t+\delta), t+\delta)$:

$$\begin{aligned} V(x(t),t) = & \inf_{u(\cdot)\in\mathbb{U}_t^{t+\delta}} \min \Bigg\{ \min_{\tau\in[t,t+\delta]} \max \left[l(\zeta_{x,t}^{u,d}(\tau),\tau), \max_{s\in[t,\tau]} g(\zeta_{x,t}^{u,d}(s),s) \right], \\ & \max \Bigg\{ \inf_{u(\cdot)\in\mathbb{U}_{t+\delta}^{T}} \Bigg\{ \min_{\tau\in[t+\delta,T]} \max \left[l(\zeta_{x,t}^{u,d}(\tau),\tau), \max_{s\in[t+\delta,\tau]} g(\zeta_{x,t+\delta}^{u,d}(s),s) \right] \Bigg\}, \max_{s\in[t,t+\delta]} g(\zeta_{x,t}^{u,d}(s),s) \Bigg\} \Bigg\}, \end{aligned} \tag{2.72}$$

we can now replace the math in blue with the value function,

$$V(x(t),t) = \inf_{u(\cdot)\in\mathbb{U}_t^{t+\delta}} \min\Big\{ \min_{\tau\in[t,t+\delta]} \max\left[l(\zeta_{x,t}^{u,d}(\tau),\tau), \max_{s\in[t,\tau]} g(\zeta_{x,t}^{u,d}(s),s)\right], \\ \max\Big\{ V(x(t+\delta),t+\delta), \max_{s\in[t,t+\delta]} g(\zeta_{x,t}^{u,d}(s),s)\Big\}\Big\}. \tag{2.73}$$

Now that we finally have a recursive equation for dynamic programming, we can once again plug in the Taylor's expansion,

$$V(x(t),t) = \inf_{u(\cdot)\in\mathbb{U}_t^{t+\delta}} \min\Big\{ \min_{\tau\in[t,t+\delta]} \max\left[l(\zeta_{x,t}^{u,d}(\tau),\tau), \max_{s\in[t,\tau]} g(\zeta_{x,t}^{u,d}(s),s)\right], \\ \max\Big\{ V(x(t),t) + D_tV\delta + D_xV\cdot f(x,u)\delta, \max_{s\in[t,t+\delta]} g(\zeta_{x,t}^{u,d}(s),s)\Big\}\Big\}, \tag{2.74}$$

and then assume that δ is small:

$$V(x(t),t) = \inf_{u} \min\Big\{ \max\left[l(\zeta_{x,t}^{u,d}(t),t), g(\zeta_{x,t}^{u,d}(t),t)\right], \\ \max\Big\{ V(x(t),t) + D_tV\delta + D_xV\cdot f(x,u)\delta, g(\zeta_{x,t}^{u,d}(t),t)\Big\}\Big\}. \tag{2.75}$$

Note that $\zeta_{x,t}^{u,d}(t)$ is a trajectory that starts at state x and time t and stops at the same time t. Therefore, it has not moved and remains at the same state $\zeta_{x,t}^{u,d}(t) = x$, so we can rewrite the equation as

$$V(x(t),t) = \inf_{u} \min\Big\{ \max\left[l(x,t), g(x,t)\right], \\ \max\Big\{ V(x(t),t) + D_tV\delta + D_xV\cdot f(x,u)\delta, g(x(t),t)\Big\}\Big\}. \tag{2.76}$$

We will now pull everything outside of the infimum over control except for expressions that depend on control:

$$V(x(t),t) = \min\Big\{ \max\left[l(x,t), g(x,t)\right], \\ \max\Big\{ V(x(t),t) + D_tV\delta + \inf_{u}\{D_xV\cdot f(x,u)\}\delta, g(x(t),t)\Big\}\Big\}. \tag{2.77}$$

Noting the identity that $\min[\max(a,b),\max(b,c)] = \max[\min(a,c),b]$,

$$V(x(t),t) = \max\Big[\\ \min\Big\{ l(x,t), V(x(t),t) + D_tV\delta + \inf_{u}\{D_xV\cdot f(x,u)\}\delta\Big\}, g(x(t),t)\Big]. \tag{2.78}$$

Subtract $V(x(t),t)$ from both sides of the equation:

$$0 = \max\Big[\min\Big\{l(x,t) - V(x(t),t), D_tV(x(t),t)\delta + \inf_u\{D_xV(x(t),t)\cdot f(x,u)\}\delta\Big\}, \\ g(x(t),t) - V(x(t),t)\Big], \tag{2.79}$$

and remove the positive δ as before, resulting in the Double-Obstacle Isaacs version of the variational inequality:

$$\boxed{0 = \max\Big[\min\Big\{l(x,t) - V(x(t),t), D_tV + \inf_u\{D_xV\cdot f(x,u)\}\Big\}, \\ g(x(t),t) - V(x(t),t)\Big].} \tag{2.80}$$

A more convenient form of the Double-Obstacle Isaacs equation is in the form of dynamic programming. We add $V(x,t)$ back to both sides and note the partial Taylor's expansion $V(x(t),t) + D_tV\delta = V(x(t),t+\delta)$,

$$V(x(t),t) = \max\Big[\min\Big\{l(x,t), V(x(t),t) + D_tV(x(t),t)\delta \\ + \inf_u\{D_xV(x(t),t)\cdot f(x,u)\delta\}\Big\}, g(x(t),t)\Big], \tag{2.81}$$

$$\boxed{V(x(t),t) = \max\Big[\min\Big\{l(x,t), V(x(t),t+\delta) + \inf_u\{D_xV(x(t),t)\cdot f(x,u)\delta\}\Big\}, \\ g(x(t),t)\Big].} \tag{2.82}$$

This modified form of the Double-Obstacle Isaacs in (2.82) is used to recursively update the value function at each instant in time to obtain the reach-avoid BRT. Intuitively, one solves for the value function using the HJI PDE, then minimizes with $l(x)$ to obtain a minimization over time, followed by a maximization with $g(x)$ to obtain a maximum with the obstacle cost.

To solve a reach-avoid BRS, we simply remove the minimization with $l(x)$:

$$\boxed{0 = \max\Big[D_tV(x(t),t) + \inf_u\{D_xV(x(t),t)\cdot f(x,u)\}, g(x(t),t) - V(x(t),t)\Big],} \tag{2.83}$$

$$\boxed{V(x(t),t) = \max\Big[\ V(x(t),t+\delta) + \inf_u\{D_xV(x(t),t)\cdot f(x,u)\delta\}, g(x(t),t)\Big].} \tag{2.84}$$

Examples of using the HJI PDE to solve for a reach-avoid BRT and BRS are in 10.1.

2.7 Chapter Summary

This chapter covered the background knowledge of dynamic systems, optimal control, and reachability analysis. The following chapters will now dive into the work on safe real-world autonomy in uncertain and unstructured environments.

Part I

Theory for Computational Gains in Reachability Analysis

Chapter 3

Decoupling as Disturbance

This chapter is based on the paper "Fast Reachable Set Approximations via State Decoupling Disturbances" [35], written in collaboration with Mo Chen and Claire J. Tomlin.

Optimal control problems and differential games are important and powerful theoretical tools for analyzing a wide variety of systems, particularly in safety-critical scenarios. They have been extensively studied in the past several decades [169, 62, 19, 165, 126, 26, 73], and have been successfully applied to practical problems such as pairwise collision avoidance [126], aircraft in-flight refueling [57], vehicle platooning [41], and many others [21, 91]. With the recent growing interest in using safety-critical autonomous systems such as autonomous cars and unmanned aerial vehicles for civil purposes [140, 159, 116, 141], the importance and necessity of having flexible tools that can provide safety guarantees have substantially increased.

Intuitively, in an optimal control problem, one seeks to find the cheapest way a system described by an ordinary differential equation (ODE) model can perform a certain task. In a differential game, a system is controlled by two adversarial agents competing to respectively minimize and maximize a joint cost function. Hamilton-Jacobi (HJ) reachability is a common and effective way to analyze both optimal control problems and differential games because of the guarantees that it provides and its flexibility with respect to the system dynamics.

In a reachability problem, one is given some system dynamics described by an ODE, and a target set which describes the set of final conditions under consideration. Depending on the application, the target set can represent either a set of desired or undesired states. The goal in reachability analysis is to compute the backward reachable set (BRS). When the target set is a set of desired states, the BRS represents the set of states from which the system can be guaranteed to be driven to the target set, despite the worst case disturbance. In contrast, when the target set is a set of undesired states, the BRS represents the set of states from which the system may be driven into the target set under some disturbance, despite its best control efforts to remain outside. Because of the theoretical guarantees that reachability analysis provides, it is ideal for analyzing the newest problems involving

autonomous systems.

Despite these advantages, there are serious drawbacks to using HJ Reachability on large systems. In order to compute the BRS, an HJ partial differential equation (PDE) must be solved on a grid representing a discretization of the state space. This means that the complexity of computing a BRS grows exponentially with the number of system states, making the standard HJ reachability formulation intractable for systems higher than approximately five dimensions. To address this difficulty, a number of approximation techniques have been developed, such as those involving projections, approximate dynamic programming, and occupation measure theory [131, 124, 115].

In this chapter, we propose a general method to remove coupling in systems by treating coupling variables as disturbances. This uncoupling of dynamics transforms the system into a form that is suitable for analysis using methods such as [128] and [37], which exploit system structure. This method can also be combined with previous work such as [131, 124, 115]. to reduce computation complexity even further. We show that our approach results in BRSs that are conservative in the desired direction, and demonstrate the performance of our method when combined with the decoupled formulation in [37].

3.1 Problem Formulation

Consider a differential game between two players described by the time-invariant system with state $x \in \mathbb{R}^n$ evolving according to the ODE

$$\begin{aligned} &\dot{x} = f(x, u, d), \quad t \in [t, T] \\ &u \in \mathcal{U}, d \in \mathcal{D} \end{aligned} \tag{3.1}$$

where u is the control of Player 1, and d is the control of Player 2. Often, u and d are regarded as the control and disturbance, respectively, of a system described by (3.1). As in [169, 62, 126], we allow Player 2 to only draw from nonanticipative strategies Γ_t^T, defined in Section 2.3.

Suppose that the state can be written as $x = (x_1, x_2)$ such that the control u and disturbance d can be written as $u = (u_{x_1}, u_{x_2}), d = (d_{x_1}, d_{x_2})$, and such a decomposition of the control leads to the following form of system dynamics:

$$\begin{aligned} &\dot{x_1} = g(x_1, x_2, u_{x_1}, d_{x_1}) \quad \dot{x_2} = h(x_1, x_2, u_{x_2}, d_{x_2}) \\ &u_{x_1} \in \mathcal{U}_{x_1}, u_{x_2} \in \mathcal{U}_{x_2}, \quad d_{x_1} \in \mathcal{D}_{x_1}, d_{x_2} \in \mathcal{D}_{x_2}, \quad \tau \in [t, T] \end{aligned} \tag{3.2}$$

where $x_1 \in \mathbb{R}^{n_{x_1}}, x_2 \in \mathbb{R}^{n_{x_2}}, n_{x_1} + n_{x_2} = n$, and g, h are components of the system dynamics that involve $(u_{x_1}, d_{x_1}), (u_{x_2}, d_{x_2})$, respectively. Note that this assumption on u and d is a mild one, and is satisfied by any system in which each of the control components u_{x_1}, u_{x_2} and disturbance components d_{x_1}, d_{x_2} have independent control sets $\mathcal{U}_{x_1}, \mathcal{U}_{x_2}$ and disturbance sets $\mathcal{D}_{x_1}, \mathcal{D}_{x_2}$, respectively; note that we can also write $\mathcal{U} = \mathcal{U}_{x_1} \times \mathcal{U}_{x_2}$, and $\mathcal{D} = \mathcal{D}_{x_1} \times \mathcal{D}_{x_2}$. This decomposition is very common in real world systems, where control input bounds such as maximum acceleration and maximum turn rate are independent of each other.

For the system in the form of (3.2), we would like to compute the BRS of time horizon T, denoted $\mathcal{V}_{\text{BRS}}(T)$:

$$\mathcal{V}_{\text{BRS}}(t) = \{x : \exists u(\cdot) \in \mathbb{U}_t^T, \forall d \in \Gamma_t^T, \zeta_{x,t}^{u,d}(T) \in \mathcal{L}\}, \tag{3.3}$$

Standard HJ formulations exist for computing the BRS in the full dimensionality n [19, 126, 26, 73]. In addition, special HJ formulations can be used to substantially reduce computation complexity for systems with special properties such as having terminal integrators or having fully decoupled dynamics [128, 37]. The goal of this chapter will be to demonstrate how to take advantage of previous work on BRS computation for systems of particular forms, even when the actual system dynamics do not exactly satisfy necessary assumptions. For concreteness, in this chapter we will focus on removing coupling to put systems into a fully decoupled form that satisfies the assumptions in [37].

Our proposed approach computes an approximation of the BRS in dimension $\max(n_{x_1}, n_{x_2})$ instead of in dimension n, dramatically reducing computation complexity. This is done by removing coupling in the dynamics by treating certain variables as disturbances. The computed approximation is conservative in the desired direction, meaning any state in the approximate BRS is also in the true BRS.

3.2 Background

Given the system in (3.1), the BRS $\mathcal{V}_{\text{BRS}}(t)$ can be computed using the following terminal value HJI PDE:

$$\begin{aligned} D_t V(x,t) + \min_{u \in \mathcal{U}} \max_{d \in \mathcal{D}} D_x V(x,t) \cdot f(x,u,d) &= 0 \\ V(x,T) &= l(x) \end{aligned} \tag{3.4}$$

In (3.4), the target set $\mathcal{L}$ is represented as the sub-zero level set of the function $l(x)$: $\mathcal{L} = \{x \in \mathbb{R}^n : l(x) \leq 0\}$. The reach BRS $\mathcal{V}_{\text{BRS}}(t)$ is the subzero level set of the value function: $\mathcal{V}_{\text{BRS}}(t) = \{x \in \mathbb{R}^n : V(x,t) \leq 0\}$ [126, 48]. The control u tries to grow the BRS as much as possible, and the disturbance d tries to do the opposite. Similar definitions of the BRS, for example one in which the control tries to inhibit its growth, can be computed by adjusting the minimum and maximum. The value function can be computed using (3.4).

In the case that the system is fully decoupled in the form of (3.5), [37] provides a method to exactly compute $V(x,t)$. The computation is done in each of the decoupled components, substantially reducing computation complexity as long as the system is decoupled and as long as the target $\mathcal{L}$ can be written as the intersection $\cap_i \mathcal{L}_i$.

$$\begin{aligned} \dot{x_i} &= f_i(x_i, u_i, d_i), \quad \tau \in [t, T] \\ u_i &\in \mathcal{U}_i, d_i \in \mathcal{D}_i, \quad i = 1, \ldots, N \end{aligned} \tag{3.5}$$

The reason for the dimension reduction is intuitive, and can be easily verified by checking that the following statements are equivalent:

1. $\exists u_i \in \mathbb{U}_i, \forall d_i \in \mathbb{D}_i, x_i(\cdot)$ satisfies (3.5), $x_i(0) \in \mathcal{L}_i, i = 1, 2$
2. $\exists (u_1, u_2) \in \mathbb{U}_1 \times \mathbb{U}_2, \forall (d_1, d_2) \in \mathbb{D}_1 \times \mathbb{D}_2$, $x_i(\cdot)$ satisfies (3.5), $(x_1(0), x_2(0)) \in \mathcal{L}_1 \cap \mathcal{L}_2$
3. $\exists u \in \mathbb{U}, \forall d \in \mathbb{D}, x(\cdot)$ satisfies (3.1), $x(0) \in \mathcal{L}$

Coupling as Disturbance

The decoupled formulation summarized in Section 3.2 enables the exact computation of n dimensional BRSs in the space of the n_i dimensional decoupled components. However, this substantial computation benefit can only be gained if the system dynamics satisfy (3.5).

In the case where the dynamics are in the form of (3.2), the decoupled formulation cannot be directly used. However, if one treats x_2 as a disturbance in the function g, and x_1 as a disturbance in the function h, the system would become decoupled. Mathematically, (3.2) becomes the following:

$$\begin{array}{lll} \dot{x_1} = \hat{g}(x_1, x_{2d}, u_{x_1}, d_{x_1}) & \dot{x_2} = \hat{h}(x_2, x_{1d}, u_{x_2}, d_{x_2}) & \\ x_1 \in \mathcal{X}_1, x_{2d} \in \mathcal{X}_2 & x_{1d} \in \mathcal{X}_1, x_2 \in \mathcal{X}_2 & \tau \in [t, T] \\ u_{x_1} \in \mathcal{U}_{x_1}, d_{x_1} \in \mathcal{D}_{x_1} & u_{x_2} \in \mathcal{U}_{x_2}, u_{x_2} \in \mathcal{D}_{x_2} & \end{array} \tag{3.6}$$

where $\mathcal{X}_1 \times \mathcal{X}_2$ represents the full-dimensional domain over which computation is done. By treating the coupled variables as a disturbance, we have uncoupled the original system dynamics (3.2), and produced approximate dynamics (3.6) that are decoupled, allowing us to do the computation in the space of each decoupled component.

Compared to the original system dynamics given in (3.2), the uncoupled dynamics given in (3.6) experiences a larger disturbance, since the x_2 dependence of the function g and the x_1 dependence on the function h are treated as disturbances. The approximate BRS computed using the dynamics (3.6) is an under-approximation of the true BRS. We formalize this in the proposition below.

Proposition 1. *Let $\mathcal{V}^{x_1}_{BRS}(t), \mathcal{V}^{x_2}_{BRS}(t)$ be the BRSs of the subsystem* (3.6) *from the target sets $\mathcal{L}_{x_1}, \mathcal{L}_{x_2}$, and let $\mathcal{V}_{BRS}(t)$ be the BRS of the system* (3.2) *from the target set $\mathcal{L}$. Then*[1]*, $\mathcal{L} = \mathcal{L}_{x_1} \cap \mathcal{L}_{x_2} \Rightarrow \mathcal{V}^{x_1}_{BRS}(t) \cap \mathcal{V}^{x_2}_{BRS}(t) \subseteq \mathcal{V}_{BRS}(t)$.*

Proof. It suffices to show that given any state $({x_1}^0, {x_2}^0) = (x_1(t), x_2(t))$ such that ${x_1}^0, {x_2}^0$ are in the BRSs $\mathcal{V}^{x_1}_{\text{BRS}}(t), \mathcal{V}^{x_2}_{\text{BRS}}(t)$ for the system in (3.6), respectively, then $({x_1}^0, {x_2}^0)$ is in the BRS $\mathcal{V}_{\text{BRS}}(t)$ for the system in (3.2).

For convenience, we will use $x_1(\cdot) \in \mathbb{X}_1$ to denote $x_1(\tau) \in \mathcal{X}_1 \ \forall \tau \in [t, T]$, with $x_2(\cdot) \in \mathbb{X}_2$ having the analogous meaning. Applying the definition of BRS in (3.3) to the subsystems in (3.6), at the state $x = (x_1, x_2)$ we have

[1]Strictly speaking, $\mathcal{L}_{x_1}, \mathcal{L}_{x_2}, \mathcal{V}^{x_1}_{\text{BRS}}(t), \mathcal{V}^{x_2}_{\text{BRS}}(t)$ would need to be "back projected" into the higher dimensional space before their intersections can be taken, but we will use the abuse of notation for convenience.

1. $\exists u_{x_1} \in \mathbb{U}_{x_1}, \forall d_{x_1} \in \mathbb{D}_{x_1}, \forall x_{2d} \in \mathbb{X}_2, x_1(\cdot)$ satisfies (3.6), $x_1(T) \in \mathcal{L}_{x_1}$

2. $\exists u_{x_2} \in \mathbb{U}_{x_2}, \forall d_{x_2} \in \mathbb{D}_{x_2}, \forall x_{1d} \in \mathbb{X}_1, x_2(\cdot)$ satisfies (3.6), $x_2(T) \in \mathcal{L}_{x_2}$

The above two conditions together imply

$$\begin{aligned} &\exists (u_{x_1}, u_{x_2}) \in \mathbb{U}_{x_1} \times \mathbb{U}_{x_2}, \forall (d_{x_1}, d_{x_2}) \in \mathbb{D}_{x_1} \times \mathbb{D}_{x_2}, \\ &\forall (x_{1d}, x_{2d}) \in \mathbb{X}_1 \times \mathbb{X}_2, (x_1(\cdot), x_2(\cdot)) \text{ satisfies (3.6)}, \\ &(x_1(T), x_2(T)) \in \mathcal{L} \end{aligned} \tag{3.7}$$

In particular, since $x_1(\cdot) \in \mathbb{X}_1, x_2(\cdot) \in \mathbb{X}_2$, the above is true also when $x_{1d} = x_1(\cdot), x_{2d} = x_2(\cdot)$, so

$$\begin{aligned} &\exists (u_{x_1}, u_{x_2}) \in \mathbb{U}_{x_1} \times \mathbb{U}_{x_2}, \forall (d_{x_1}, d_{x_2}) \in \mathbb{D}_{x_1} \times \mathbb{D}_{x_2}, \\ &(x_{1d}, x_{2d}) = (x_1(\cdot), x_2(\cdot)), (x_1(\cdot), x_2(\cdot)) \text{ satisfies (3.6)}, \\ &(x_1(T), x_2(T)) \in \mathcal{L} \end{aligned} \tag{3.8}$$

But if $x_d = x(\cdot), x_{2d} = x_2(\cdot)$, then (3.6) becomes (3.2), thus

$$\begin{aligned} &\exists (u_{x_1}, u_{x_2}) \in \mathbb{U}_{x_1} \times \mathbb{U}_{x_2}, \forall (d_{x_1}, d_{x_2}) \in \mathbb{D}_{x_1} \times \mathbb{D}_{x_2}, \\ &(x_1(\cdot), x_2(\cdot)) \text{ satisfies (3.2)}, (x_1(T), x_2(T)) \in \mathcal{L}. \end{aligned} \tag{3.9}$$

□

By treating the coupled states as disturbance, the computation complexity reduces from $O(k^n T)$ for the full formulation to $O(k^{\max\{n_{x_1}, n_{x_2}\}} T)$ for the decoupled approximate system (3.6).

3.3 Disturbance Splitting

By treating coupling variables y and x_1 as disturbances in g and h, respectively, we introduce conservatism in the BRS computation. This conservatism is always in the desired direction. In situations where $\mathcal{X}_1$ and $\mathcal{X}_2$ are large, the degree of conservatism can be reduced by splitting the disturbance x_{1d} and y_d into multiple sections, as long as the target set $\mathcal{L}$ does not depend on the state variables being split. For example, $x_{1d} \in \mathcal{X}_1$ can be split as follows:

$$\begin{aligned} &x_{1_d}^i \in \mathcal{X}_{1i}, i = 1, 2, \ldots, M \\ &\text{where } \bigcup_{i=1}^{M} \mathcal{X}_{1i} = \mathcal{X}_1 \end{aligned} \tag{3.10}$$

This disturbance splitting results in the following family of approximate system dynamics

$$\begin{array}{ll} \dot{x_1} = g(x_1, x_{2d}, u_{x_1}, d_{x_1}) & \dot{x_2} = h(x_2, x_{1_d}^i, u_{x_2}, d_{x_2}) \\ u_{x_1} \in \mathcal{U}_{x_1}, d_{x_1} \in \mathcal{D}_{x_1} & u_{x_2} \in \mathcal{U}_{x_2}, d_{x_2} \in \mathcal{D}_{x_2} \\ x_{2d} \in \mathcal{X}_2 & x_{1_d}^i \in {\mathcal{X}_1}^i \\ \tau \in [t, T] & i = 1, 2, \ldots, M_{x_1} \end{array} \tag{3.11}$$

from which a BRS can be computed in ${\mathcal{X}_1}^i \times \mathcal{X}_2$. Since ${\mathcal{X}_1}^i \subseteq \mathcal{X}_1$, the uncoupling disturbance is reduced whenever the disturbance x_{1d} is split. In addition, the uncoupling disturbance x_{2d} can also be split into M_{x_2}, for a total of $M = M_{x_1} M_{x_2}$ total "pieces" of uncoupling disturbances. However, a smaller disturbance bound also restricts the allowable trajectories of each approximate system, so overall it is difficult to determine *a priori* the optimal way to split the uncoupling disturbances. The trade-off between the size of disturbance bound and degree of restriction placed on trajectories can be seen in Fig. 3.2.

3.4 Examples of Decoupling System Dynamics

Our proposed method applies to any system of the form (3.2), as we will demonstrate with the example in Section 3.5. Systems with light coupling between groups of state variables are particularly suitable for the application or our proposed method. Below are other example systems for which treating the coupling variables y in g or x in h as disturbances would lead to decoupling.

Linear systems with large Jordan blocks, for example,

$$\dot{x} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} x + Bu \tag{3.12}$$

If the third dimension of x is treated as a disturbance in the equation for the second dimension, we could decompose the system into the first two and last two states.

Lateral Quadrotor Dynamics near Hover [29]:

$$\dot{z} = \begin{bmatrix} z_2 \\ g \tan z_3 \\ z_4 \\ -d_0 z_3 - z_1 z_4 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ n_0 \end{bmatrix} u \tag{3.13}$$

where z_1 is the longitudinal position, z_2 is the longitudinal velocity, z_3 is the pitch angle, z_4 is the pitch rate, and u is the desired pitch angle. These dynamics are valid for small pitch angles. The system would become decoupled into the $x_1 = (z_1, z_2)$ and $x_2 = (z_3, z_4)$ components if z_3 is treated as a disturbance in $\dot{z}_2$. In fact, the full ten-dimensional (10D)

quadrotor dynamics given in [29] can be decomposed into five decoupled components of 2D systems, allowing an approximation of the 10D BRS to be computed in 2D space.

It is worth noting that after decoupling the 4D system in (3.13) into two 2D systems, each decoupled component is in the form $\dot{x_2} = g(x_2, u), \dot{x_1} = b(x_2)$. This is exactly the form of the dynamics in [128], allowing the 4D BRS to be exactly computed in 1D! In general, removing coupling may bring the system dynamics into a form suitable for analysis using methods that require specific assumptions on the dynamics, potentially greatly reducing computation complexity.

3.5 Numerical Results

We demonstrate our proposed method using a 4D model of an aircraft flying at constant altitude, given by

$$\begin{aligned} \dot{p}_{x_1} &= v\cos\psi & \dot{p}_{x_2} &= v\sin\psi \\ \dot{\psi} &= \omega & \dot{v} &= a \\ \underline{\omega} \le \omega &\le \bar{\omega} & \underline{a} \le a &\le \bar{a} \end{aligned} \tag{3.14}$$

where (p_{x_1}, p_{x_2}) represent the plane's position in the x_1 and x_2 directions, ψ represents the plane's heading, and v represents the plane's longitudinal velocity. The plane's controls are $u = [\omega, a]$, where ω is the limited turn rate and a is the limited longitudinal acceleration. For our example, the computation bounds are

$$\begin{aligned} p_x &\in [-40, 40] \text{ m} & p_y &\in [-40, 40] \text{ m} \\ \psi &\in [-\pi, \pi] \text{ rad} & v &\in [6, 12] \text{ m/s} \end{aligned}$$

Using the decoupled approximation technique, we create the following decoupled approximation of the system with (p_{x_1}, ψ) and (p_{x_2}, v) as the decoupled components:

$$\begin{aligned} \dot{p}_{x_1} &= d_v \sin(\psi) & \dot{p}_{x_2} &= v\sin(d_\psi) \\ \dot{\psi} &= \omega & \dot{v} &= a \\ \underline{\omega} \le \omega &\le \bar{\omega} & \underline{a} \le a &\le \bar{a} \\ \underline{d_v} \le d_v &\le \bar{d_v} & \underline{d_\psi} \le d_\psi &\le \bar{d_\psi} \end{aligned} \tag{3.15}$$

We define the target set as a square of length 4 m centered at $(p_{x_1}, p_{x_2}) = (0, 0)$, described by $\mathcal{L} = \{(p_{x_1}, p_{x_2}, \psi, v) : |p_{x_1}|, |p_{x_2}| \le 2\}$. This can be interpreted as a positional goal centered at the origin that can be achieved for all angles and velocities within the computation grid bounds. From the target set, we define $l(x)$ such that $l(x) \le 0 \Leftrightarrow x \in \mathcal{L}$. To analyze our newly decoupled system we must likewise decouple the target set by letting $\mathcal{L}_i, i = 1, 2$ be

$$\begin{aligned} \mathcal{L}_1 &= \{(p_{x_1}, \psi) : |p_{x_1}| \le 2\} \\ \mathcal{L}_2 &= \{(p_{x_2}, v) : |p_{x_2}| \le 2\} \end{aligned} \tag{3.16}$$

These target sets have corresponding implicit surface functions $l_i(x_i), i = 1, 2$, which then form the 4D target set represented by $l(x) = \max_i l_i(x_i), i = 1, 2$.

We set $\mathcal{L}$ as the target set in our reachability problem and computed the BRS $\mathcal{V}_{\text{BRS}}(t)$ from $\mathcal{L}$ using both the direct 4D computation as well as the proposed decoupled approximation method.

Backward Reachable Sets

The BRS describes in this case the set of initial conditions from which the system is guaranteed to reach the target set within a given time period $T - t$ despite worst possible disturbances. To analyze the BRS we vary the degree of conservatism using the disturbance splitting method described in Section 3.3. After applying splitting, we arrive at the following piece-wise system:

$$\begin{array}{ll} \dot{p}_{x_1} = d_v^i \sin(\psi) & \dot{p}_{x_2} = v \sin(d_\psi^j) \\ \dot{\psi} = \omega & \dot{v} = a \\ \underline{\omega} \leq \omega \leq \bar{\omega} & \underline{a} \leq a \leq \bar{a} \\ \underline{d}_v^i \leq d_v^i \leq \bar{d}_v^i & \underline{d}_\psi^j \leq d_\psi^j \leq \bar{d}_\psi^j \\ i = 1, 2, \ldots, M_v & j = 1, 2, \ldots, M_\psi \end{array} \tag{3.17}$$

We analyzed the decoupled approximation formulation with M_v and M_ψ ranging from 1 to 32. To visually depict the computed 4D BRS, we plot 3D slices of the BRS in Fig. 3.1. In these slices the green sets are the BRS computed using the full formulation, and the gray sets are the decoupled approximations. With the definition of BRS given in (3.3), all the decoupled approximations are constructed to be under-approximations.

The top row of plots shows the 3D projections through all values of v. The bottom row of plots shows the 3D projections through all values of ψ. Moving from left to right, each column of plots shows the decoupled approximations with an increasing number of split sections M_ψ and M_v.

Reconstruction Performance

To compare the degree of conservatism of the decoupled approximations, we determined the total 4D volume of the BRS computed using both methods. We then took the ratio of the decoupled approximation volume to the full formulation BRS volume. Since under-approximations are computed, a higher volume ratio indicates a lower degree of conservatism.

Fig. 3.2 shows this volume ratio as a function of M_ψ and M_v. For example, the purple curve represents the volume ratio for $M_v = 1$ across various values of M_ψ, and on the other extreme, the yellow curve represents the volume ratio for $M_v = 32$ across various values of M_ψ. The highest number of sections computed was with $M_\psi = M_v = 32$.

Initially the decoupled approximations become less conservative as M_v and M_ψ increase. This is because splitting the disturbance range has the effect of mitigating the strength

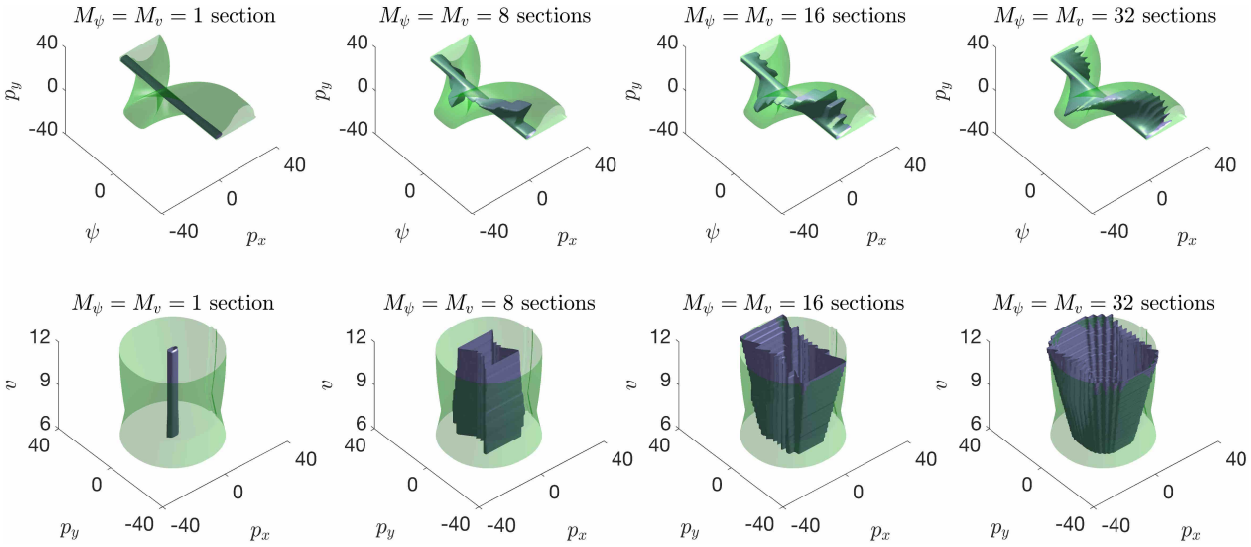


Figure 3.1: 3D slices of the BRSs across a range of M_v and M_ψ. The full formulation sets are in green, and the decoupled approximation sets are in gray. The top row shows 3D projections through all values of v. The top left plot shows this projection for $M_\psi = M_v = 1$ with M_ψ, M_v increasing as we move right in the list of plots, up to $M_\psi = M_v = 32$ at top right. The bottom figures show the same sections for 3D projections through ψ.

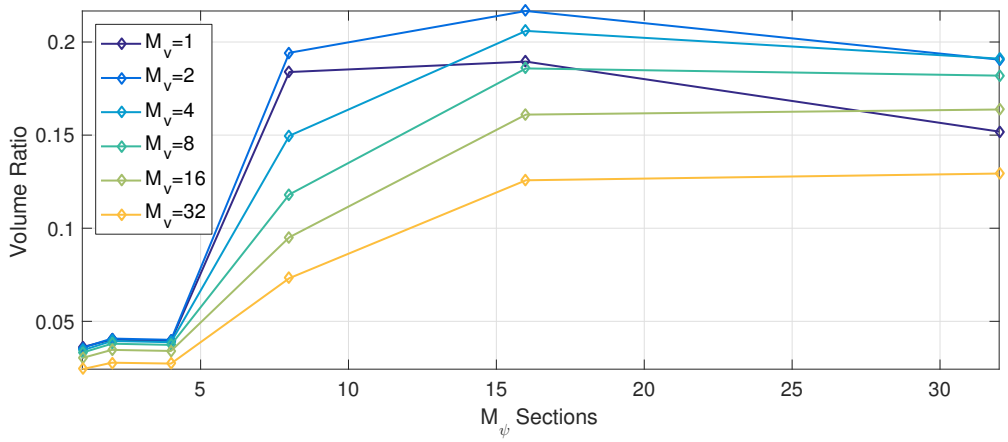


Figure 3.2: The vertical axis represents the ratio of the reconstructed BRS volume over the full formulation volume. The graph shows how this ratio changes as a function of the number of disturbance sections. The highest volume ratio (and therefore least conservative BRS) was for $M_\psi = 16, M_v = 2$.

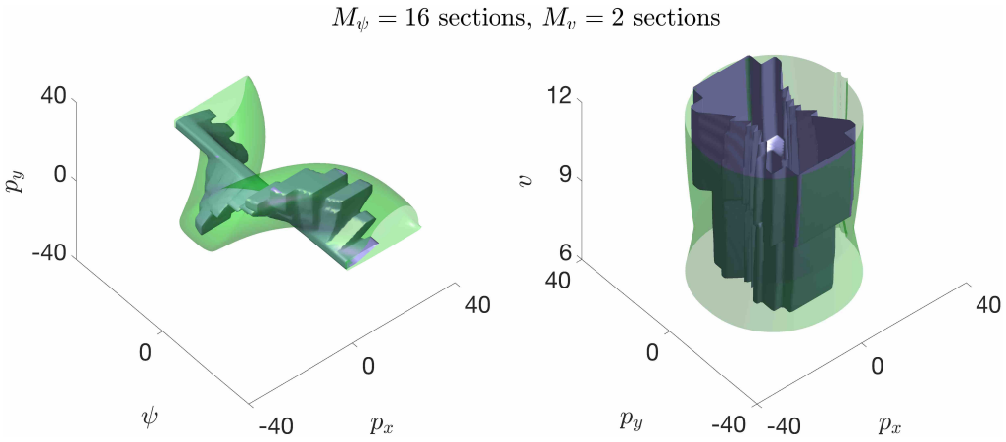


Figure 3.3: 3D slices of the BRS for $M_\psi = 16, M_v = 2$. This decoupled approximation provides the largest and least conservative under-approximation.

of the disturbances. However, splitting the disturbance range also restricts the allowable trajectories of the system and can eventually introduce more conservatism. For example, if the velocity disturbance range is $6 \leq d_v \leq 12$, the trajectories must stay within this velocity range for the duration of T. If this range is split, the set of disturbances has a smaller range, but likewise the trajectories for each subsection must remain within the smaller split velocity range for the time period. Therefore, there is an optimal point past which splitting does not help decrease conservatism.

In this system the least conservative approximation was for $M_\psi = 16, M_v = 2$. The volume ratio for this approximation was 0.217, meaning that the decoupled approximation had a volume that is 21.7% of the volume of the BRS computed using the full formulation. The 3D projections of the set computed by $M_\psi = 16, M_v = 2$ can be seen in Fig. 3.3.

Computation Time Performance

In Fig. 3.4 we compare the computation time of the two methods as a function of the number of grid points in each dimension. Computations were done on a desktop computer with a Core i7-5820K CPU and 128 GB of random-access memory (RAM). The full formulation (yellow curve) quickly becomes intractable as grid points are added; 100 grid points in each dimension requires 12.7 hours and 97 GB of RAM.

The decoupled approximation is orders of magnitude faster than the full formulation, and therefore can be done with many more grid points in each dimension. The decoupled approximation used was for $M_\psi = 16, M_v = 2$ sections, as this provided the most accurate BRS as determined in Section 3.5. The runtimes would be even faster using $M_\psi = M_v = 1$ sections. We plot both the runtimes for reachability computation with 4D-reconstruction (red curve) as well as the runtimes for reachability computation alone (blue curve). Compared to the full formulation, at 100 grid points in each dimension the decoupled approximation

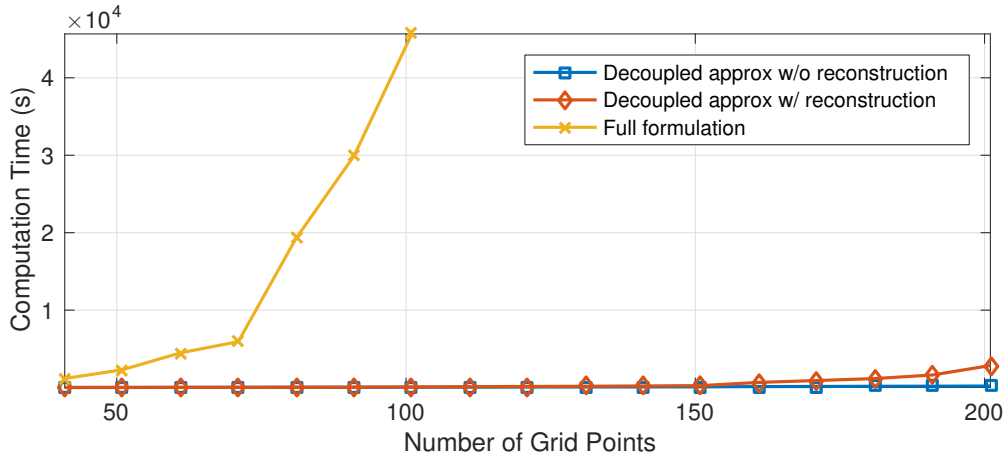


Figure 3.4: Computation time as a function of the number of grid points in each dimension. The full formulation (yellow curve) is orders of magnitude slower than the decoupled approximation. The decoupled approximation with reconstruction (red curve) takes a bit more time (and significantly more memory) than without reconstruction (blue curve).

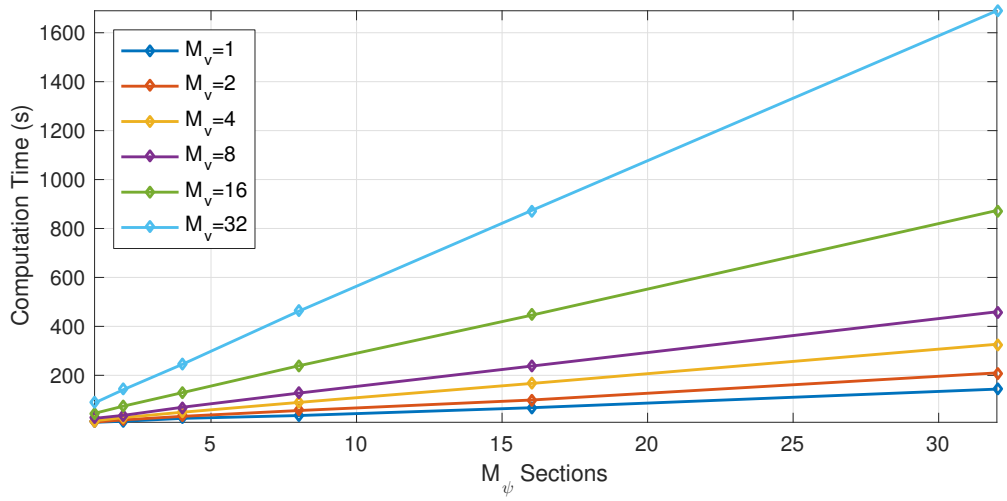


Figure 3.5: Computation time to compute decoupled approximation sets without reconstruction as a function of M_v and M_ψ. As the number of sections increases, the computation time increases approximately linearly.

takes 50 seconds to run and 36 seconds to reconstruct, with 625 MB of RAM to run and 6.75 GB of RAM to reconstruct. At 200 grid points the decoupled approximation takes 3.37 minutes to run and 44.1 minutes to reconstruct, with 1 GB of RAM to run and 120 GB of RAM to reconstruct.

In general we recommend computing the value function in only a region near a state of interest, bypassing the time and RAM required to reconstruct the function over the entire grid. Without full reconstruction of the value function, we are able to obtain results faster and for higher numbers of grid points before running out of memory, improving the accuracy of the computation.

In Fig. 3.5 we compare the computation time of the 2D computations in the decoupled approximation as a function M_v and M_ψ. Each line of the graph represents the computation time for a fixed number of M_v across various values of M_ψ. As the number of sections increases, the computation time required increases approximately linearly, as expected.

3.6 Chapter Summary

Hamilton-Jacobi reachability analysis can provide safety and performance guarantees for many practical systems, but the curse of dimensionality limits its application to systems with less than approximately five state variables. By treating state variables as disturbances, key state dependencies can be eliminated, reducing the system dynamics to a simpler form and allowing reachable sets to be calculated conservatively using available efficient methods in the literature.

Chapter 4

Self-Contained Subsystems

This chapter is based on the papers "Exact and Efficient Hamilton-Jacobi-based Guaranteed Safety Analysis via System Decomposition" [34] and "A General System Decomposition Method for Computing Reachable Sets and Tubes" [38], written in collaboration with Mo Chen, Mahesh Vashishtha, Somil Bansal, and Claire J. Tomlin.

In this chapter, we present a system decomposition method for computing BRSs and BRTs of a class of nonlinear systems. Our method drastically reduces dimensionality without making any other trade offs. 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. 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 [31, 153]. Since BRTs are also of great interest in many situations, we prove conditions under which BRTs can also be decomposed.

The theory we present in this chapter is compatible with any methods that compute BRSs and BRTs, such as [42, 6, 74, 128, 37]. In addition, when different decomposition methods are combined together, even more dimensionality reduction can be achieved. This chapter will be presented as follows:

- In Sections 4.1 and 4.2 we introduce the basic concept of reachability, and all the definitions needed for our proposed decomposition technique.
- In Sections 4.3 and 4.4 we present our theoretical results related to decomposing BRSs for systems involving a control variable, but *not* involving a disturbance variable.
- In Section 4.5 we show how BRTs can be decomposed.
- In Section 4.6 we demonstrate our decomposition method on high-dimensional systems.

- In Section 4.7 we discuss how the presence of disturbances affects the above theoretical results.
- We will also present numerical results obtained through the Hamilton-Jacobi (HJ) reachability formulation in [126] throughout the chapter to validate our theory.

4.1 Background

There are various formulations for computing the BRS and BRT when the system dimensionality is low. In this section, we give the basic mathematical problem setup to provide a foundation on which we build the new proposed theory.

System Dynamics

Let $z \in \mathbb{R}^n$ be the system state, which evolves according to the ordinary differential equation (ODE)

$$\frac{dz(\tau)}{d\tau} = \dot{z}(\tau) = f(z(\tau), u(\tau)), \tau \in [t, T], u(\tau) \in \mathcal{U} \tag{4.1}$$

In general, the theory we present is applicable when some states are periodic dimensions (such as angles), but for simplicity we will consider $\mathbb{R}^n$. The control is denoted by $u(\tau)$, with the control function $u(\cdot) \in \mathbb{U}$ being drawn from the set of measurable functions.

The control signal $u \in \mathcal{U} \subset \mathbb{R}^{n_u}$ is compact, $T = 0$, and $t < T$. The system dynamics, or flow field, $f : \mathbb{R}^n \times \mathcal{U} \to \mathbb{R}^n$ is assumed to be uniformly continuous, bounded, and Lipschitz continuous in[1] z for fixed u. Therefore, given $u(\cdot) \in \mathbb{U}$, there exists a unique trajectory solving (4.1) [46]. We will denote solutions, or trajectories of (4.1) starting from state z at time t under control $u(\cdot)$ as $\zeta(\tau; z, t, u(\cdot)) : [t, T] \to \mathbb{R}^n$. ζ satisfies (4.1) with an initial condition almost everywhere:

$$\begin{aligned} \frac{d}{d\tau}\zeta(\tau; z, t, u(\cdot)) &= f(\zeta(\tau; z, t, u(\cdot)), u(\tau)) \\ \zeta(t; z, t, u(\cdot)) &= z \end{aligned} \tag{4.2}$$

Since the dynamics (4.1) are time-invariant, the time variables in trajectories can also be shifted by some constant[2] s:

$$\zeta(\tau; z, t, u(\cdot)) = \zeta(\tau + s; z, t + s, u(\cdot)), \forall z \in \mathbb{R}^n \tag{4.3}$$

[1]For the remainder of the chapter, we will omit the notation "(τ)" from variables such as z and u when referring to function values.

[2]In this case, it is implicit that the control function $u(\cdot)$ is also time-shifted by the same amount s.

Backward Reachable Sets and Tubes

We consider two different definitions of the BRS and two different definitions of the BRT.

Intuitively, a BRS represents the set of states $z \in \mathbb{R}^n$ from which the system can be driven into some set $\mathcal{L} \subseteq \mathbb{R}^n$ *at the end* of a time horizon of duration $|t - T|$. We call $\mathcal{L}$ the "target set". First we define the "Maximal BRS"; in this case the system seeks to enter $\mathcal{L}$ using *some* control function. We can think of $\mathcal{L}$ as a set of goal states. The Maximal BRS represents the set of states from which the system is guaranteed to reach $\mathcal{L}$. The second definition is for the "Minimal BRS"; in this case the BRS is the set of states that will lead to $\mathcal{L}$ for *all possible* controls. Here we often consider $\mathcal{L}$ to be an unsafe set such as an obstacle. The Minimal BRS represents the set of states that leads to violation of safety requirements. Formally, the two definitions of BRSs are below [3]:

Definition 1. ***Maximal BRS***.

$$\mathcal{R}(t) = \{z : \exists u(\cdot) \in \mathbb{U}, \zeta(T; z, t, u(\cdot)) \in \mathcal{L}\}$$

Definition 2. ***Minimal BRS***.

$$\mathcal{A}(t) = \{z : \forall u(\cdot) \in \mathbb{U}, \zeta(T; z, t, u(\cdot)) \in \mathcal{L}\}$$

While BRSs indicate whether a system can be driven into $\mathcal{L}$ at the end of a time horizon, BRTs indicate whether a system can be driven into $\mathcal{L}$ *at some time* during the time horizon of duration $|t|$. Figure 4.1 demonstrates the difference. BRTs are very important notions especially in safety-critical applications, in which we are interested in determining the "Minimal BRT": the set of states that could lead to danger at some time within a specified time horizon. Formally, the two definitions of BRTs are as follows:

Definition 3. ***Maximal BRT***.

$$\bar{\mathcal{R}}(t) = \{z : \exists u(\cdot) \in \mathbb{U}, \exists \tau \in [t, T], \zeta(\tau; z, t, u(\cdot)) \in \mathcal{L}\}$$

Definition 4. ***Minimal BRT***.

$$\bar{\mathcal{A}}(t) = \{z : \forall u(\cdot) \in \mathbb{U}, \exists \tau \in [t, T], \zeta(\tau; z, t, u(\cdot)) \in \mathcal{L}\}$$

The terms "maximal" and "minimal" refer to the role of the optimal control [127]. In the maximal (or minimal) case, the control causes the BRS or BRT to contain as many (or few) states as possible – to have maximal (or minimal) size.

While BRSs and BRTs indicate sets of states of interest, from a practical implementation perspective controller synthesis based on the reachable sets is extremely important. Much

[3] Sometimes in the literature, the argument of $\mathcal{R}$, $\mathcal{A}$, $\bar{\mathcal{R}}$, or $\bar{\mathcal{A}}$ is some non-negative number $\tau = -t$; however, for simplicity we will use the non-positive number t to refer to the time horizon of the BRS and BRT.

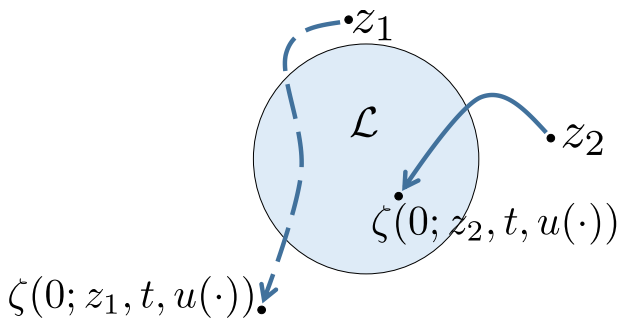


Figure 4.1: The difference between a BRS and a BRT. The dashed trajectory starts at some state z_1 and passes through $\mathcal{L}$ during the period $[t, T]$ where $T = 0$, but exits $\mathcal{L}$ by the end of the time period. Therefore the z_1 is in the BRT, but not in the BRS. The solid trajectory starting from z_2 is in $\mathcal{L}$ at the end of the time period. Therefore, z_2 is in both the BRS and the BRT.

of the prior work on reachable set computations also include controller synthesis, which is usually done by casting the reachability problem as an optimal control or optimization problem, often with a functional representation of BRSs and BRTs. The controller is given as decision variables in the optimization [169, 19, 126, 27, 121, 176]. We will not delve into the details of controller synthesis, since the theory we present in this chapter is agnostic to these details.

4.2 Problem Formulation

In this chapter, we seek to obtain the BRSs and BRTs in Definitions 1 and 4 via computations in lower-dimensional subspaces under the assumption that the system (4.1) can be decomposed into self-contained subsystems (SCS) (4.5). 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 through other variables such as heading.

Subsystem Dynamics

Let the state $z \in \mathbb{R}^n$ be partitioned as $z = (z_1, z_2, z_c)$, with $z_1 \in \mathbb{R}^{n_1}, z_2 \in \mathbb{R}^{n_2}, z_c \in \mathbb{R}^{n_c}, n_1, n_2 > 0, n_c \geq 0$, $n_1 + n_2 + n_c = n$. Note that n_c could be zero. We call z_1, z_2, z_c "state partitions" of the system. Intuitively, z_1 and z_2 are states belonging to subsystems 1 and 2, respectively, and z_c states belonging to both subsystems.

Under the above notation, the system dynamics (4.1) become

$$\begin{aligned}\dot{z}_1 &= f_1(z_1, z_2, z_c, u)\\ \dot{z}_2 &= f_2(z_1, z_2, z_c, u)\\ \dot{z}_c &= f_c(z_1, z_2, z_c, u)\end{aligned} \tag{4.4}$$

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

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

Definition 5. ***Self-contained subsystem****. Consider the following special case of* (4.4)*:*

$$\begin{aligned}\dot{z}_1 &= f_1(z_1, z_c, u)\\ \dot{z}_2 &= f_2(z_2, z_c, u)\\ \dot{z}_c &= f_c(z_c, u)\end{aligned} \tag{4.5}$$

We call each of the subsystems with states defined as $x_i = (z_i, z_c)$ a "self-contained subsystem" (SCS), or just "subsystem" for short. Intuitively (4.5) means that the evolution of each subsystem depends only on the subsystem states: $\dot{x}_i$ depends only on $x_i = (z_i, z_c)$. Explicitly, the dynamics of the two subsystems are as follows:

$\dot{z}_1 = f_1(z_1, z_c, u)$	$\dot{z}_2 = f_2(z_2, z_c, u)$
$\dot{z}_c = f_c(z_c, u)$	$\dot{z}_c = f_c(z_c, u)$
(Subsystem 1)	(Subsystem 2)

Note that the two subsystems are coupled through the common state partition z_c and control u.

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

$$\begin{bmatrix}\dot{p}_x\\ \dot{p}_y\\ \dot{\theta}\end{bmatrix} = \begin{bmatrix}v\cos\theta\\ v\sin\theta\\ \omega\end{bmatrix}, \qquad \omega \in \mathcal{U} \tag{4.6}$$

with state $z = (p_x, p_y, \theta)$ representing the x position, y position, and heading, and control $u = \omega$ representing the turn rate. The state partitions are simply the system states: $z_1 = p_x, z_2 = p_y, z_c = \theta$. The dynamics of the subsystems are

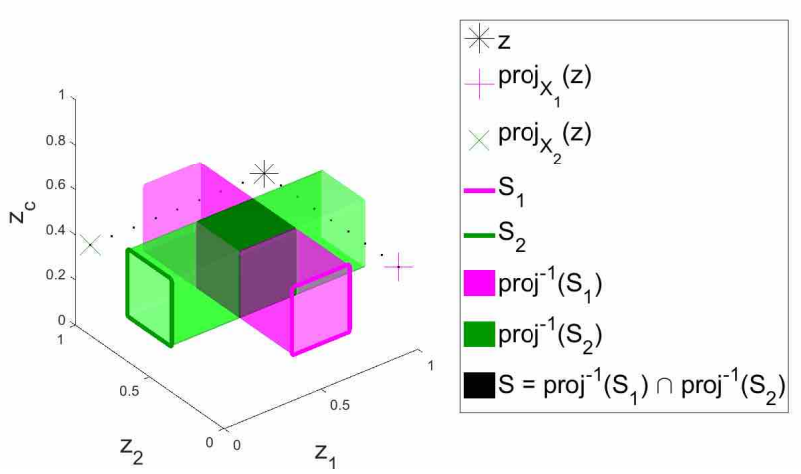


Figure 4.2: This figure shows the back-projection of sets in the z_1-z_c plane S_1 and the z_2-z_c plane (S_2) to the 3D space to form the intersection shown as the black cube (S). The figure also shows projection of a point z onto the lower-dimensional subspaces in the z_1-z_c and z_2-z_c planes.

$$\begin{aligned} \dot{x_1} &= \begin{bmatrix} \dot{z_1} \\ \dot{z_c} \end{bmatrix} = \begin{bmatrix} \dot{p_x} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ \omega \end{bmatrix} \\ \dot{x_2} &= \begin{bmatrix} \dot{z_2} \\ \dot{z_c} \end{bmatrix} = \begin{bmatrix} \dot{p_y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\sin\theta \\ \omega \end{bmatrix} \\ u &= \omega \end{aligned} \tag{4.7}$$

where the overlapping state is θ, and the shared control is u itself. For more examples of systems decomposed into SCSs, see Section 4.6.

Although there may be common or overlapping states in x_1 and x_2, the evolution of each subsystem does not depend on the other explicitly. In fact, if we for example entirely ignore the subsystem x_2, the evolution of the subsystem x_1 is well-defined and can be considered a full system on its own; hence, each subsystem is self-contained.

Projection Operators

For the projection operators, it will be helpful to refer to Fig. 4.2. Define the projection of a state $z = (z_1, z_2, z_c)$ onto a subsystem state space $\mathbb{R}^{n_i+n_c}$ as

$$\text{proj}_i(z) = x_i = (z_i, z_c) \tag{4.8}$$

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

$$\text{proj}^{-1}(x_i) = \{z \in \mathcal{Z} : (z_i, z_c) = x_i\} \tag{4.9}$$

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 subsystems set to full dimensional sets. In this case, we overload the back-projection operator:

$$\text{proj}^{-1}(\mathcal{S}_i) = \{z \in \mathcal{Z} : \exists x_i \in \mathcal{S}_i, (z_i, z_c) = x_i\} \tag{4.10}$$

Subsystem Trajectories

Since each subsystem in (4.5) is self-contained, we can denote the subsystem trajectories $\xi_i(\tau; x_i, t, u(\cdot))$. When needed, we will write the subsystem trajectories more explicitly in terms of the state partitions as $\xi_i(\tau; z_i, z_c, t, u(\cdot))$. The subsystem trajectories satisfy the subsystem dynamics and initial condition:

$$\begin{aligned} \frac{d}{ds}\xi_i(\tau; x_i, t, u(\cdot)) &= g_i(\xi_i(\tau; x_i, t, u(s)) \\ \xi_i(t; x_i, t, u(\cdot)) &= x_i \end{aligned} \tag{4.11}$$

where $g_i(x_i, u) = (f_i(z_i, z_c, u), f_c(z_c, u))$, and the full system trajectory and subsystem trajectories are simply related to each other via the projection operator:

$$\text{proj}_i\big(\zeta(\tau; z, t, u(\cdot))\big) = \xi_i(\tau; x_i, t, u(\cdot)) \tag{4.12}$$

where $x_i = \text{proj}_i(z)$.

Goals of This Chapter

We assume that the full system target set $\mathcal{L}$ can be written in terms of the subsystem target sets $\mathcal{L}_1 \subseteq \mathcal{X}_1, \mathcal{L}_2 \subseteq \mathcal{X}_2$ in one of the following ways:

$$\mathcal{L} = \text{proj}^{-1}(\mathcal{L}_1) \cap \text{proj}^{-1}(\mathcal{L}_2) \tag{4.13}$$

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

$$\mathcal{L} = \text{proj}^{-1}(\mathcal{L}_1) \cup \text{proj}^{-1}(\mathcal{L}_2) \tag{4.14}$$

where the full target set is the union of the back-projections of subsystem target sets. Fig. 4.2 helps provide intuition for these concepts: applying (4.13) to $\mathcal{S}_1$ and $\mathcal{S}_2$ results in the black cube. Applying (4.14) would result in the cross-shaped set encompassing both $\text{proj}^{-1}(\mathcal{S}_1)$ and $\text{proj}^{-1}(\mathcal{S}_2)$.

In practice, this is not a strong assumption, since $\mathcal{L}_1$ and $\mathcal{L}_2$ share the common variables z_c. Relatively complex shapes, for example those in Fig. 4.4 and 4.6, can be represented by an intersection or union of back-projections of lower-dimensional sets that share common

variables. In addition, such an assumption is reasonable since the full system target set should at least be representable in some way in the lower-dimensional spaces.

Next, we define the subsystem BRSs $\mathcal{R}_i, \mathcal{A}_i$ the same way as in Definitions 1 and 2, but with the subsystems in (4.5) and subsystem target sets $\mathcal{L}_i, i = 1, 2$, respectively:

$$\begin{aligned} \mathcal{R}_i(t) &= \{x_i : \exists u(\cdot), \xi_i(T; x_i, t, u(\cdot)) \in \mathcal{L}_i\} \\ \mathcal{A}_i(t) &= \{x_i : \forall u(\cdot), \xi_i(T; x_i, t, u(\cdot)) \in \mathcal{L}_i\} \end{aligned} \tag{4.15}$$

Subsystem BRTs are defined analogously:

$$\begin{aligned} \bar{\mathcal{R}}_i(t) &= \{x_i : \exists u(\cdot), \exists \tau \in [t, T], \xi_i(\tau; x_i, t, u(\cdot)) \in \mathcal{L}_i\} \\ \bar{\mathcal{A}}_i(t) &= \{x_i : \forall u(\cdot), \exists \tau \in [t, T], \xi_i(\tau; x_i, t, u(\cdot)) \in \mathcal{L}_i\} \end{aligned} \tag{4.16}$$

Given a system in the form of (4.5) with target set that can be represented by (4.13) or (4.14), our goals are as follows.

- **Decomposition of BRSs.** First, we would like to compute full-dimensional BRSs by performing computations in lower-dimensional subspaces. Specifically, we would like to first compute the subsystem BRSs $\mathcal{R}_i(t)$ or $\mathcal{A}_i(t)$, and then reconstruct the full system BRS $\mathcal{R}(t)$ or $\mathcal{A}(t)$. This process greatly reduces computation burden by decomposing the full system into two lower-dimensional subsystems. Formally, we would like to investigate the situations in which each of the following four cases is true:

$$\begin{aligned} &(4.13) \Rightarrow \mathcal{R}(t) = \text{proj}^{-1}(\mathcal{R}_1(t)) \cap \text{proj}^{-1}(\mathcal{R}_2(t)) \\ &(4.13) \Rightarrow \mathcal{A}(t) = \text{proj}^{-1}(\mathcal{A}_1(t)) \cap \text{proj}^{-1}(\mathcal{A}_2(t)) \\ &(4.14) \Rightarrow \mathcal{R}(t) = \text{proj}^{-1}(\mathcal{R}_1(t)) \cup \text{proj}^{-1}(\mathcal{R}_2(t)) \\ &(4.14) \Rightarrow \mathcal{A}(t) = \text{proj}^{-1}(\mathcal{A}_1(t)) \cup \text{proj}^{-1}(\mathcal{A}_2(t)) \end{aligned} \tag{4.17}$$

  Results related to BRSs are outlined for SCSs in Theorems 1 and 2. In the case that the subsystem controls are independent, Propositions 2 and 3 state stronger results.

- **Decomposition of BRTs**. BRTs are useful since they provide guarantees over a time horizon as opposed to at a particular time. However, often BRTs cannot be decomposed the same way as BRSs. Therefore, our second goal is to propose how BRTs can be decomposed. These results are stated in Propositions 4 and 5, and Theorem 3.

- **Treatment of disturbances**. Finally, we investigate how the above theoretical results change in the presence of disturbances. In Section 4.7, we will show that slightly conservative BRSs and BRTs can still be obtained using our decomposition technique.

Tables 4.1, 4.2, and 4.3 summarize our theoretical results and where details of each result can be found.

Table 4.1: Backward Reachable Set Decomposition

Section	**4.3**		**4.4**		**4.7**		**4.7**	
Shared Controls	**Yes**		**No**		**Yes**		**No**	
Shared Disturbance	**No**		**No**		**Yes**		**Yes**	
Target	**Intersection**	**Union**	**Intersection**	**Union**	**Intersection**	**Union**	**Intersection**	**Union**
Recover Max. BRS?	No	Yes, exact	Yes, exact	Yes, exact	No	Yes, consrv	Yes, consrv	Yes, consrv
Recover Min. BRS?	Yes, exact	No	Yes, exact	Yes, exact	Yes, consrv	No	Yes, consrv	Yes, consrv
Locations & Equation(s)	Thm 2, (4.19)	Thm 1, (4.18)	Prop 2, (4.34) Thm 2, (4.19)	Thm 1, (4.18) Prop 3, (4.35)	Cor 4, (4.73)	Cor 3, (4.71)	Cor 5, (4.74)	Cor 6, (4.76)

Summary of possible decompositions of the BRS, 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 z_c in (4.5); in the case that there are no shared states this becomes a straightforward decoupled system.

Table 4.2: BRT Results for Reconstruction from Tubes

Section	**4.5**				**4.7**			
Shared Controls	**Yes**		**No**		Yes		**No**	
Shared Disturbance	**No**		**No**		Yes		**Yes**	
Target	**Intersection**	**Union**	**Intersection**	**Union**	**Intersection**	**Union**	**Intersection**	Union
Recover Max. BRT?	No	Yes, exact	No	Yes, exact	No	Yes, conserv	No	Yes, conserv
Recover Min. BRT?	No	No	No	Yes, exact	No	No	No	Yes, conserv
Equation(s)	N/A	Prop 4, (4.40)	N/A	Prop 4, (4.40) Prop 4, (4.41)	N/A	Cor 7, (4.77)	N/A	Cor 7, (4.77) Cor 8, (4.78)

Summary of possible decompositions of 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 z_c in (4.5); in the case that there are no shared states this becomes a straightforward decoupled system.

Table 4.3: BRT Results for Reconstruction from Sets

Section	**4.5**	**4.7**
Disturbance	**No**	**Yes**
Recover Max. BRT?	Yes, exact	Yes, conserv
Recover Min. BRT?	Yes, exact*	Yes, exact*
Equation(s)	Prop 5, (4.42) Thm 3, (4.44)	Cor 9, (4.79) Thm 3, (4.44)

Summary of computation of the BRT via union of BRSs, whether they are exact or conservative. Exact means that no additional approximation errors are introduced. These results can be combined with those of Table 4.1 to, for the negative cases in Table 4.2, efficiently compute BRTs by first computing BRSs via decomposition and then taking the union of BRSs to obtain the BRT. "*" indicates that the solution here can be found only if the minimal BRSs are non-empty for the entire time period.

4.3 Self-Contained Subsystems

Suppose the full system (4.1) can be decomposed into SCSs in (4.5), then the full-dimensional BRS can be reconstructed, without incurring additional approximation errors, from lower-dimensional BRSs in situations stated in Theorem 1 and 2.

Remark 1. *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 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 in* (4.1) *can be decomposed into the form of* (4.5)*, then*

$$\begin{aligned} &\mathcal{L} = proj^{-1}(\mathcal{L}_1) \cup proj^{-1}(\mathcal{L}_2) \\ &\Rightarrow \mathcal{R}(t) = proj^{-1}(\mathcal{R}_1(t)) \cup proj^{-1}(\mathcal{R}_2(t)) \end{aligned} \tag{4.18}$$

Remark 2. *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 a target set $\mathcal{L}$ in the form of an intersection of subsystem target sets, 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 in* (4.1) *can be decomposed into the form of* (4.5)*, then*

$$\begin{aligned} &\mathcal{L} = \overset{\leftarrow}{proj}^{-1}(\mathcal{L}_1) \cap \overset{\leftarrow}{proj}^{-1}(\mathcal{L}_2) \\ &\Rightarrow \mathcal{A}(t) = \overset{\leftarrow}{proj}^{-1}(\mathcal{A}_1(t)) \cap \overset{\leftarrow}{proj}^{-1}(\mathcal{A}_2(t)) \end{aligned} \tag{4.19}$$

These two theorems imply that the optimal closed-loop state feedback controller is decomposable: at any given (full) state, the optimal control depends only on either of the subsystem states, but not the full state. Therefore, BRSs for a system using a pre-specified closed-loop controller can be decomposed if the control depends only on subsystem states at any given (full) state. It is also interesting to note that BRSs for a system using a pre-specified open-loop controller can be decomposed, since augmenting the state space with $\dot{t} = 1$ would maintain the structure of self-contained subsystems. The additional computational burden from the state augmentation can be eliminated through a time-varying formulation of reachability, for example, in reference [73].

To prove the theorems, we need some intermediate results.

Lemma 1. *Let* $\bar{z} \in \mathcal{Z}, \bar{x}_i = proj_i(\bar{z}), \mathcal{S}_i \subseteq \mathcal{X}_i$. *Then,*

$$\bar{x}_i \in \mathcal{S}_i \Leftrightarrow \bar{z} \in \overset{\leftarrow}{proj}^{-1}(\mathcal{S}_i) \tag{4.20}$$

Proof. Forward direction: Suppose $\bar{x}_i \in \mathcal{S}_i$, then trivially $\exists x_i \in \mathcal{S}_i, \text{proj}_i(\bar{z}) = x_i$. By the definition of back-projection in (4.10), we have $\bar{z} \in \text{proj}^{-1}(\mathcal{S}_i)$.

Backward direction: Suppose $\bar{z} \in \text{proj}^{-1}(\mathcal{S}_i)$, then by (4.10) we have $\exists x_i \in \mathcal{S}_i, \text{proj}_i(\bar{z}) = x_i$. Denote such an x_i to be $\hat{x}_i$, and suppose $\bar{x}_i \notin \mathcal{S}_i$. Then, we have $\hat{x}_i \neq \bar{x}_i$, a contradiction, since $\bar{x}_i = \text{proj}_i(\bar{z}) = \hat{x}_i$. □

Corollary 1. *If* $\mathcal{S} = \overset{\leftarrow}{proj}^{-1}(\mathcal{S}_1) \cup \overset{\leftarrow}{proj}^{-1}(\mathcal{S}_2)$, *then*

$$\bar{z} \in \mathcal{S} \Leftrightarrow \bar{x}_1 \in \mathcal{S}_1 \vee \bar{x}_2 \in \mathcal{S}_2, \text{ where } \bar{x}_i = proj_i(\bar{z})$$

Corollary 2. *If* $\mathcal{S} = \overset{\leftarrow}{proj}^{-1}(\mathcal{S}_1) \cap \overset{\leftarrow}{proj}^{-1}(\mathcal{S}_2)$, *then*

$$\bar{z} \in \mathcal{S} \Leftrightarrow \bar{x}_1 \in \mathcal{S}_1 \wedge \bar{x}_2 \in \mathcal{S}_2, \text{ where } \bar{x}_i = proj_i(\bar{z})$$

Proof of Theorem 1

Proof. We will prove the following equivalent statement:

$$\bar{z} \in \mathcal{R}(t) \Leftrightarrow \bar{z} \in \text{proj}^{-1}(\mathcal{R}_1(t)) \cup \text{proj}^{-1}(\mathcal{R}_2(t)) \tag{4.21}$$

Consider the relationship between the full system trajectory and subsystem trajectory in (4.12). Define $\bar{x}_i = \text{proj}_i(\bar{z})$ and $\xi_i(T; \bar{x}_i, t, u(\cdot)) = \text{proj}_i(\zeta(T; \bar{z}, t, u(\cdot)))$.

Backward direction: By Corollary 1, (4.21) is equivalent to

$$\bar{x}_1 \in \mathcal{R}_1(t) \vee \bar{x}_2 \in \mathcal{R}_2(t) \tag{4.22}$$

WLOG, assume $\bar{x}_1 \in \mathcal{R}_1(t)$. By the subsystem BRS definition in (4.15), this is equivalent to

$$\exists u(\cdot), \xi_1(T; \bar{x}_1, t, u(\cdot)) \in \mathcal{L}_1 \tag{4.23}$$

By Lemma 1, we equivalently have $\bar{z} \in \text{proj}^{-1}(\mathcal{R}_1(t))$. This proves the backward direction.

Forward direction: We begin with $\bar{z} \in \mathcal{R}(t)$, which by Definition 1 is equivalent to $\exists u(\cdot), \zeta(T; \bar{z}, t, u(\cdot)) \in \mathcal{L}$. By Corollary 1, we then have

$$\exists u(\cdot), \xi_1(T; \bar{x}_1, t, u(\cdot)) \in \mathcal{L}_1 \vee \xi_2(T; \bar{x}_2, t, u(\cdot)) \in \mathcal{L}_2 \tag{4.24}$$

Finally, distributing "$\exists u(\cdot)$" gives (4.22). □

Proof of Theorem 2

Proof. We will prove the following equivalent statement:

$$\bar{z} \notin \mathcal{A}(t) \Leftrightarrow \bar{z} \notin \text{proj}^{-1}(\mathcal{A}_1(t)) \cap \text{proj}^{-1}(\mathcal{A}_2(t)) \tag{4.25}$$

The above statement is equivalent to

$$\bar{z} \in \mathcal{A}^c(t) \Leftrightarrow \bar{z} \in \left[\text{proj}^{-1}(\mathcal{A}_1(t))\right]^c \cup \left[\text{proj}^{-1}(\mathcal{A}_2(t))\right]^c \tag{4.26}$$

By the Definition 2 (minimal BRS), we have that $\bar{z} \in \mathcal{A}^c(t)$ is equivalent to $\exists u(\cdot) \in \mathbb{U}, \zeta(T; \bar{z}, t, u(\cdot)) \in \mathcal{L}^c$. Also,

$$\bar{z} \in \left[\text{proj}^{-1}(\mathcal{A}_1(t))\right]^c \cup \left[\text{proj}^{-1}(\mathcal{A}_2(t))\right]^c$$

is equivalent to $\bar{x}_1 \in \mathcal{A}_1^c(t) \vee \bar{x}_2 \in \mathcal{A}_2^c(t)$.

From here, we can proceed in the same fashion as the proof of Theorem 1, with "$\mathcal{R}(t)$" replaced with "$\mathcal{A}^c(t)$", "$\text{proj}^{-1}(\mathcal{R}_i(t))$" replaced with "$\left[\text{proj}^{-1}(\mathcal{A}_i(t))\right]^c$", and "$\mathcal{L}$", "$\mathcal{L}_i$" replaced with "$\mathcal{L}^c$", "$\mathcal{L}_i^c$", respectively. □

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 and beyond the scope of this chapter. A summary of the results from this section can be seen in Table 4.1 under Section 4.3.

Numerical Example: The Dubins Car

The Dubins Car is a well-known system whose dynamics are given by (4.6). 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

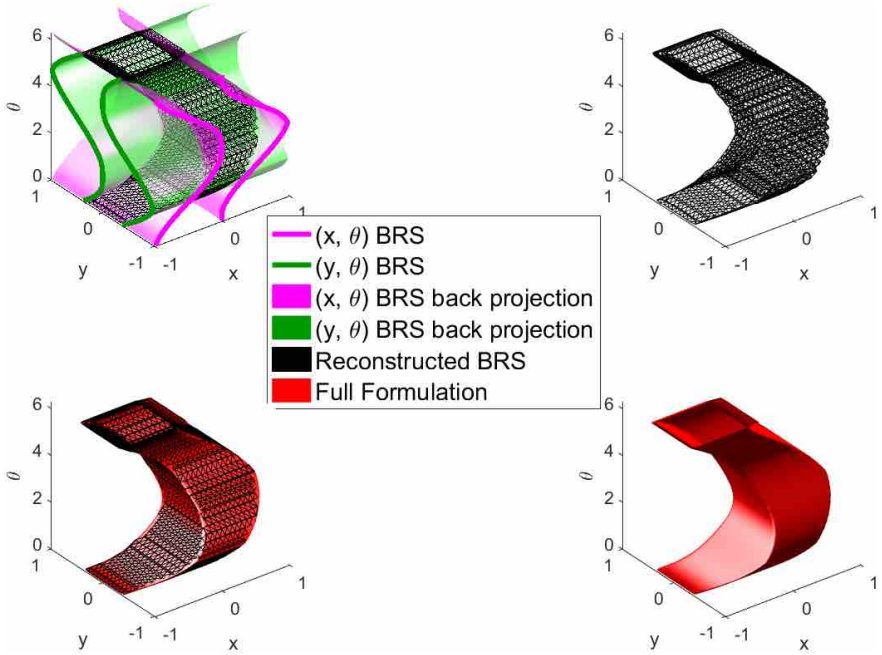


Figure 4.3: Comparison of the Dubins Car BRS $\mathcal{A}(t = -0.5)$ computed using the full formulation and via decomposition. Left top: BRSs in the lower-dimensional subspaces and how they are combined to form the full-dimensional BRS. Top right: BRS computed via decomposition. Bottom left: BRSs computed using both methods, superimposed, showing that they are indistinguishable. Bottom right: BRS computed using the full formulation.

can be decomposed according to (4.7). 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:

$$\mathcal{L} = \{(p_x, p_y, \theta) : |p_x|, |p_y| \leq 0.5\} \tag{4.27}$$

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 $|t|$. From $\mathcal{L}$, we computed the BRS $\mathcal{A}(t)$ at $t = -0.5$. The resulting full formulation BRS is shown in Fig. 4.3 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

$$\begin{aligned} \mathcal{L}_1 = \{(p_x, \theta) : |p_x| \leq 0.5\}, \mathcal{L}_2 = \{(p_y, \theta) : |p_y| \leq 0.5\} \\ \mathcal{L} = \text{proj}^{-1}(\mathcal{L}_1) \cap \text{proj}^{-1}(\mathcal{L}_2) \end{aligned} \tag{4.28}$$

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: $\mathcal{A}(t) = \text{proj}^{-1}(\mathcal{A}_1(t)) \cap \text{proj}^{-1}(\mathcal{A}_2(t))$. The subsystem BRSs and their back-projections are shown in magenta and

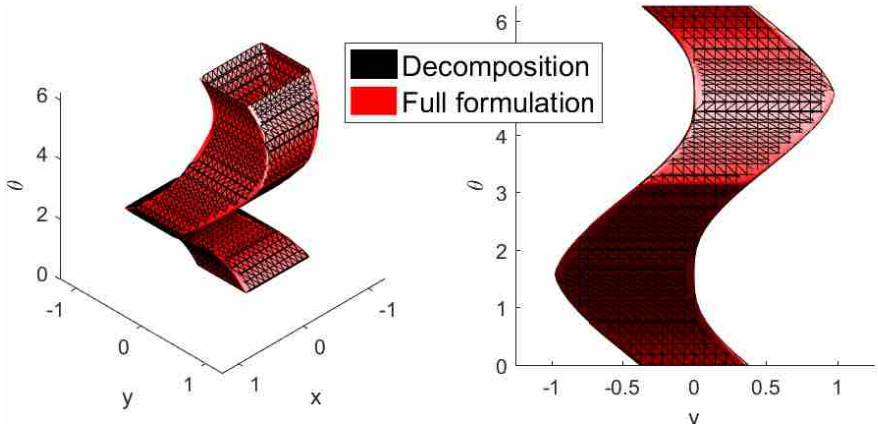


Figure 4.4: The Dubins Car BRS $\mathcal{A}(t = -0.5)$ computed using the full formulation and via decomposition, other view angles.

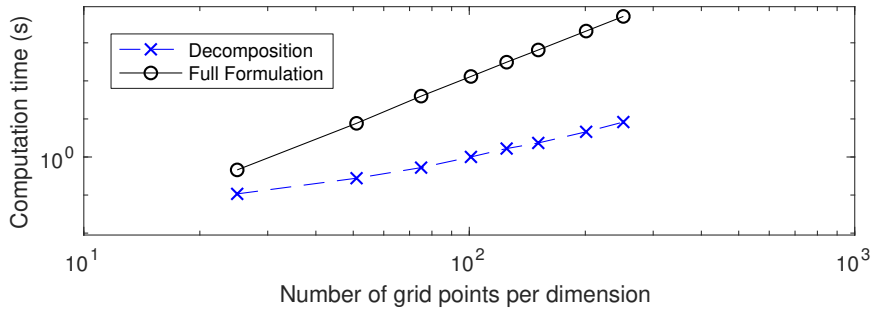


Figure 4.5: Computation times of the two methods in log scale for the Dubins Car. The time of the direct computation in 3D increases rapidly with the number of grid points per dimension. In contrast, computation times in 2D with decomposition are negligible in comparison.

green in the top left subplot of Fig. 4.3. The reconstructed BRS is shown in the top left, top right, and bottom left subplots of Fig. 4.3 (black mesh).

In the bottom left subplot of Fig. 4.3, 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. 4.4. 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 HJ 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. 4.5. 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

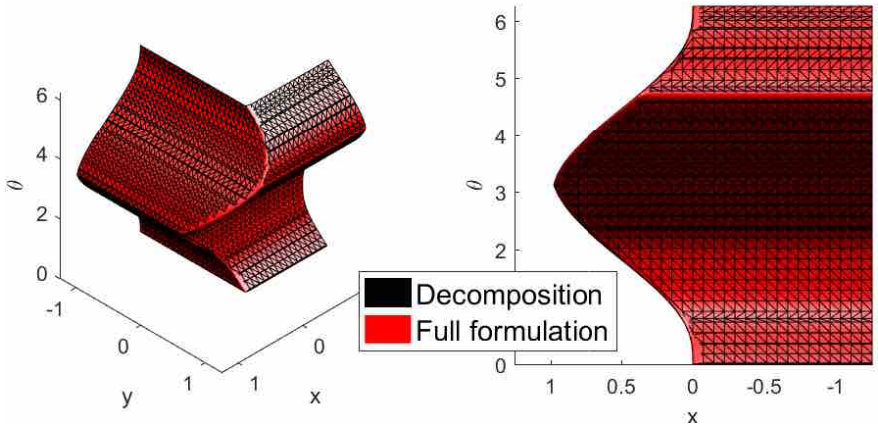


Figure 4.6: Comparison of the $\mathcal{R}(t)$ computed using our decomposition method and the full formulation. The computation results are indistinguishable. Note that the surface shows the boundary of the set; the set itself is on the "near" side of the left subplot, and the left side of the right subplot.

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.

Fig. 4.6 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 $\mathcal{L} = \text{proj}^{-1}(\mathcal{L}_1) \cup \text{proj}^{-1}(\mathcal{L}_1)$ where $\mathcal{L}_1 = \{(p_x, \theta) : p_x \leq 0.5\}, \mathcal{L}_2 = \{(p_y, \theta) : p_y \leq 0.5\}$.

4.4 SCSs with Decoupled Control

In this section, we consider a special case of (4.5) in which the subsystem controls are independent. The results from Section 4.3 still hold, and *in addition* we can state the results in Propositions 2 and 3. The special case of (4.5) is as follows:

$$\begin{aligned} \dot{z}_1 &= f_1(z_1, z_c, u_1) \\ \dot{z}_2 &= f_2(z_2, z_c, u_2) \\ \dot{z}_c &= f_c(z_c) \end{aligned} \tag{4.29}$$

where the subsystem controls are independent, so that we have $u = (u_1, u_2)$. Furthermore, we can define the trajectory of z_c as $\eta(\tau; z_c, t))$, which satisfies

$$\begin{aligned} \frac{d}{d\tau}\eta_c(\tau; z_c, t) &= f_c(\eta_c(\tau; z_c, t)) \\ \eta_c(t; z_c, t)) &= z_c \end{aligned} \tag{4.30}$$

Note that since the trajectory $\eta_c(\tau; z_c, t)$ does not depend on the control, we can treat $\eta_c(\tau; z_c, t)$ as a constant when given z_c and τ. Therefore, given z_c, the other *state partitions* also become self-contained, with dynamics

$$\dot{z}_i = f_i(z_i, z_c, u_i) = f_i(z_i, u_i; \eta_c(s; z_c, t)) \tag{4.31}$$

and with trajectories $\eta_i(\tau; z_i, z_c, t, u_i(\cdot))$ satisfying

$$\begin{aligned} &\frac{d}{d\tau}\eta_i(\tau; z_i, z_c, t, u_i(\cdot)) \\ &\quad = f_i(\eta_i(\tau; z_i, z_c, t, u_i(\cdot)), u_i(\tau); \eta_c(\tau; z_c, t)) \\ &\quad \eta_i(t; z_i, z_c, t, u_i(\cdot)) = z_i \end{aligned} \tag{4.32}$$

Therefore, the *subsystem trajectories* can be written as

$$\xi_i(\tau; x_i, t, u_i(\cdot)) = \big(\eta_i(\tau; z_i, z_c, t, u_i(\cdot)), \eta_c(\tau; z_c, t)\big) \tag{4.33}$$

Proposition 2. *Suppose that the full system in* (4.1) *can be decomposed into the form of* (4.29)*. Then,*

$$\begin{aligned} &\mathcal{L} = proj^{-1}(\mathcal{L}_1) \cap proj^{-1}(\mathcal{L}_2) \\ &\Rightarrow \mathcal{R}(t) = proj^{-1}(\mathcal{R}_1(t)) \cap proj^{-1}(\mathcal{R}_2(t)) \end{aligned} \tag{4.34}$$

Proposition 3. *Suppose that the full system in* (4.1) *can be decomposed into the form of* (4.29)*. Then,*

$$\begin{aligned} &\mathcal{L} = proj^{-1}(\mathcal{L}_1) \cup proj^{-1}(\mathcal{L}_2) \\ &\Rightarrow \mathcal{A}(t) = proj^{-1}(\mathcal{A}_1(t)) \cup proj^{-1}(\mathcal{A}_2(t)) \end{aligned} \tag{4.35}$$

Remark 3. *Systems with fully decoupled subsystems in the form of* $x_1 = z_1, x_2 = z_2$ *are a special case of* (4.29)*.*

Proof of Proposition 2

Proof. We will prove the following equivalent statement:

$$\bar{x} \in \mathcal{R}(t) \Leftrightarrow \bar{x} \in \text{proj}^{-1}(\mathcal{R}_1(t)) \cap \text{proj}^{-1}(\mathcal{R}_2(t)) \tag{4.36}$$

By Definition 1 (maximal BRS), we have

$$\bar{x} \in \mathcal{R}(t) \Leftrightarrow \exists u(\cdot), \zeta(T; \bar{x}, t, u(\cdot)) \in \mathcal{L} \tag{4.37}$$

Consider the relationship between the full system trajectory and subsystem trajectory in (4.12). Define

$$\begin{aligned}\bar{x}_i &= (\bar{z}_i, \bar{z}_c) = \text{proj}_i(\bar{x}), \text{ and} \\ \xi_i(\tau; \bar{x}_i, t, u_i(\cdot)) &= \text{proj}_i(\zeta(T; \bar{x}, t, u(\cdot)))\end{aligned}$$

By (4.33) we can write

$$\big(\eta_i(\tau; \bar{z}_i, \bar{z}_c, t, u_i(\cdot)), \eta_c(\tau; \bar{z}_c, t)\big) = \text{proj}_i(\zeta(T; \bar{x}, t, u(\cdot)))$$

Since $\mathcal{L} = \text{proj}^{-1}(\mathcal{L}_1) \cap \text{proj}^{-1}(\mathcal{L}_2)$, (4.37) is equivalent to

$$\begin{aligned}&\exists\,(u_1(\cdot), u_2(\cdot))\,, \\ &\quad \big(\eta_1(\tau; \bar{z}_1, \bar{z}_c, t, u_1(\cdot)), \eta_c(\tau; \bar{z}_c, t)\big) \in \mathcal{L}_1 \wedge \\ &\quad \big(\eta_2(\tau; \bar{z}_2, \bar{z}_c, t, u_2(\cdot)), \eta_c(tdummy; \bar{z}_c, t)\big) \in \mathcal{L}_2 \\ &\qquad \text{(by Corollary 2)}\end{aligned} \tag{4.38}$$

$$\begin{aligned}\Leftrightarrow \quad &\exists u_1(\cdot), \big(\eta_1(\tau; \bar{z}_1, \bar{z}_c, t, u_1(\cdot)), \eta_c(\tau; \bar{z}_c, t)\big) \in \mathcal{L}_1 \wedge \\ &\exists u_2(\cdot), \big(\eta_2(\tau; \bar{z}_2, \bar{z}_c, t, u_2(\cdot)), \eta_c(\tau; \bar{z}_c, t)\big) \in \mathcal{L}_2 \\ &\text{(since subsystem controls do not share components)}\end{aligned} \tag{4.39}$$

$$\begin{aligned}\Leftrightarrow \quad &x_1 \in \mathcal{R}_1(t) \wedge x_2 \in \mathcal{R}_2(t) \\ &\text{(by definition of subsystem BRS in (4.15))}\end{aligned}$$

$$\begin{aligned}\Leftrightarrow \quad &\bar{x} \in \text{proj}^{-1}(\mathcal{R}_1(t)) \cap \text{proj}^{-1}(\mathcal{R}_2(t)) \\ &\text{(by Corollary 2)}\end{aligned}$$

□

Proof of Proposition 3

Proof. This proof follows the same arguments as 2, but with "$\mathcal{R}$", "$\cap$", "$\exists$" replaced with "$\mathcal{A}$", "$\cup$", "$\forall$", respectively. □

Remark 4. *When the subsystem controls are not independent, the control chosen by each subsystem may not agree with the other. This is the intuition behind why the results of Propositions 2 and 3 only hold true when the subsystem controls are independent. Note that the theorems hold despite the state coupling between the subsystems.*

A summary of the results from this section can be seen in Table 4.1 under Section 4.4.

4.5 Decomposition of Reachable Tubes

Sometimes, BRTs are desired; for example, in safety analysis, the computation of the BRT $\bar{\mathcal{A}}(t)$ in Definition 2 is quite important, since if the target set $\mathcal{L}$ represents an unsafe set of states, then $\bar{\mathcal{A}}(t)$ contains all states that would lead to some unsafe state at *some time* within a duration of length $|t|$.

We now first discuss a special case where the full system BRT can be directly reconstructed from subsystem BRTs in Section 4.5, and then present a general method in which a BRT can be obtained via the union of BRSs in Section 4.5.

Full System BRTs From Subsystem BRTs

Intuitively, it may seem like the results related to BRSs outlined in Sections 4.3 and 4.4 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 chapter only easily carry over for BRTs if $\mathcal{L} = \text{proj}^{-1}(\mathcal{L}_1) \cup \text{proj}^{-1}(\mathcal{L}_2)$. This is formally stated in the following Proposition:

Proposition 4. *Suppose* (4.14) *holds, that is,*

$$\mathcal{L} = \widetilde{proj}^{-1}(\mathcal{L}_1) \cup \widetilde{proj}^{-1}(\mathcal{L}_2)$$

Then, the full-dimensional BRT can be reconstructed from the lower-dimensional BRTs without incurring additional approximation errors. For systems with SCSs as in (4.5)*, we have*

$$\bar{\mathcal{R}}(t) = \widetilde{proj}^{-1}(\bar{\mathcal{R}}_1(t)) \cup \widetilde{proj}^{-1}(\bar{\mathcal{R}}_2(t)) \tag{4.40}$$

This is proven by starting the proof of Theorem 1 with Definition 3 for the BRT $\bar{\mathcal{R}}(t)$ instead of Definition 1 for the BRS $\mathcal{R}(t)$.

For systems with independent subsystem controls, we in addition have

$$\bar{\mathcal{A}}(t) = \widetilde{proj}^{-1}(\bar{\mathcal{A}}_1(t)) \cup \widetilde{proj}^{-1}(\bar{\mathcal{A}}_2(t)) \tag{4.41}$$

This can be proven by starting the proof of Proposition 3 from Definition 4 for the BRT $\bar{\mathcal{A}}(t)$ instead of Definition 2 for the BRS $\mathcal{A}(t)$. Note that (4.41) does not necessarily hold for systems whose subsystem controls are not independent.

BRTs From Union of BRSs

If $\mathcal{L} = \text{proj}^{-1}(\mathcal{L}_1) \cap \text{proj}^{-1}(\mathcal{L}_2)$, the BRT cannot be directly reconstructed from lower-dimensional BRTs because when computing BRTs, we lose information about the exact time that a trajectory enters a set. Instead, we provide an alternative method of obtaining

the BRT: First compute BRSs, and then take their union over time. We show that $\bar{\mathcal{R}}(t) = \bigcup_{\tau\in[t,T]} \mathcal{R}(\tau)$, and $\bar{\mathcal{A}}(t) = \bigcup_{\tau\in[t,T]} \mathcal{A}(\tau)$ when $\mathcal{A}(\tau) \neq \emptyset\ \forall \tau \in [t,T]$. These results related to the indirect reconstruction of BRTs are given in Proposition 5 and Theorem 3.

This method is independent of how the BRSs are obtained and whether subsystem controls are independent, since the union is taken over time of BRSs of the same dimensionality as the BRT. To take advantage of decomposition, one would compute BRSs via decomposition for the cases outlined in Sections 4.3 and 4.4, and then take the union over time of the BRSs to obtain the BRT.

Note that, as mentioned, this alternative method of computing the BRT is needed for the case where $\mathcal{L} = \text{proj}^{-1}(\mathcal{L}_1) \cap \text{proj}^{-1}(\mathcal{L}_2)$; however, it is also applicable for the cases presented in Section 4.5, where $\mathcal{L} = \text{proj}^{-1}(\mathcal{L}_1) \cup \text{proj}^{-1}(\mathcal{L}_2)$, as long as the BRSs can be computed via decomposition; thus there is overlap of applicability between the two BRT computation methods we present.

Proposition 5.

$$\bigcup_{\tau\in[t,T]} \mathcal{R}(\tau) = \bar{\mathcal{R}}(t) \tag{4.42}$$

Theorem 3.

$$\bigcup_{\tau\in[t,T]]} \mathcal{A}(\tau) \subseteq \bar{\mathcal{A}}(t) \tag{4.43}$$

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

$$\bigcup_{\tau\in[t,T]]} \mathcal{A}(\tau) = \bar{\mathcal{A}}(t). \tag{4.44}$$

Proposition 5 and the first part of Theorem 3 are known [127], but we present them in this chapter 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.

Remark 5. *The reason the theorems in Sections 4.3 and 4.4 trivially carry over when* $\mathcal{L} = proj^{-1}(\mathcal{L}_1) \cup proj^{-1}(\mathcal{L}_2)$ *is that in this case, any subsystem trajectory that reaches the corresponding subsystem target set implies that the full system trajectory reaches the full system target set.*

In contrast, in the case $\mathcal{L} = proj^{-1}(\mathcal{L}_1) \cap proj^{-1}(\mathcal{L}_2)$, *both subsystem trajectories must be in the corresponding subsystem target sets at the same time. Mathematically, recall the definitions of subsystem BRTs in* (4.16)*:*

$$\bar{\mathcal{A}}_i(t) = \{x_i : \forall u(\cdot), \exists \tau \in [t,T]], \xi_i(\tau; x_i, t, u(\cdot)) \in \mathcal{L}_i\}$$
$$\bar{\mathcal{R}}_i(t) = \{x_i : \exists u(\cdot), \exists \tau \in [t,T]], \xi_i(\tau; x_i, t, u(\cdot)) \in \mathcal{L}_i\}$$

The set of "τ*" during which each subsystem trajectory is in* $\mathcal{L}_i$ *may not overlap for the different subsystems. In this case, we can still first compute the BRSs in lower-dimensional subspaces, and then convert the BRSs to the BRT using Propositions 5 and Theorem 3.*

Proof of Proposition 5

Proof. We start with Definition 1 (maximal BRS):

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

If some state x is in the union $\bigcup_{\tau\in[t,T]]} \mathcal{R}(\tau)$, then there is some $\tau \in [t, T]]$ such that $x \in \mathcal{R}(\tau)$. Therefore, the union can be written as

$$\bigcup_{\tau\in[t,T]]} \mathcal{R}(\tau) = \{x : \exists \tau \in [t, T]], \exists u(\cdot), \zeta(T]; x, \tau, u(\cdot)) \in \mathcal{L}\} \tag{4.45}$$

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

$$\exists \tau \in [t, T]], \exists u(\cdot) \in \mathbb{U}, \zeta(T]; x, \tau, u(\cdot)) \in \mathcal{L} \tag{4.46}$$

Using (4.3), the time-invariance of the system, we can shift the trajectory time arguments by $t - \tau$ to get

$$\exists \tau \in [t, T]], \exists u(\cdot) \in \mathbb{U}, \zeta(t - \tau; x, t, u(\cdot)) \in \mathcal{L} \tag{4.47}$$

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

$$\exists \tau \in [t, T]], \exists u(\cdot) \in \mathbb{U}, \zeta(\tau; x, t, u(\cdot)) \in \mathcal{L} \tag{4.48}$$

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

$$\exists u(\cdot) \in \mathbb{U}, \exists \tau \in [t, T]], \zeta(\tau; x, t, u(\cdot)) \in \mathcal{L} \tag{4.49}$$

which is equivalent to $x \in \bar{\mathcal{R}}(t)$ by Definition 3 (maximal BRT). □

Proof of Theorem 3

Proof. We first establish $\bigcup_{\tau\in[t,T]]} \mathcal{A}(\tau) \subseteq \bar{\mathcal{A}}(t)$. Consider Definition 2 (minimal BRS):

$$\mathcal{A}(t) = \{x : \forall u(\cdot) \in \mathbb{U}, \zeta(T]; x, t, u(\cdot)) \in \mathcal{L}\}$$

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

$$\bigcup_{\tau\in[t,T]]} \mathcal{A}(\tau) = \{x : \exists \tau \in [t, T]], \forall u(\cdot), \zeta(T]; x, \tau, u(\cdot)) \in \mathcal{L}\} \tag{4.50}$$

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

$$\exists \tau \in [t, T]], \forall u(\cdot) \in \mathbb{U}, \zeta(T]; x, \tau, u(\cdot)) \in \mathcal{L} \tag{4.51}$$

Using (4.3), the time-invariance of the system, we can shift the trajectory time arguments by $t-\tau$ to get

$$\exists \tau \in [t,T]], \forall u(\cdot) \in \mathbb{U}, \zeta(t-\tau; x, t, u(\cdot)) \in \mathcal{L} \tag{4.52}$$

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

$$\exists \tau \in [t,T]], \forall u(\cdot) \in \mathbb{U}, \zeta(\tau; x, t, u(\cdot)) \in \mathcal{L} \tag{4.53}$$

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

$$\begin{aligned} &\forall u(\cdot) \in \mathbb{U}, \zeta(\bar{\tau}; x, t, u(\cdot)) \in \mathcal{L} \\ &\Rightarrow \forall u(\cdot) \in \mathbb{U}, \exists \tau \in [t,T]], \zeta(\tau; x, t, u(\cdot)) \in \mathcal{L} \end{aligned} \tag{4.54}$$

By Definition 4, we have $x \in \bar{\mathcal{A}}(t)$.

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

$$x \notin \bigcup_{\tau \in [t,T]]} \mathcal{A}(\tau) \Rightarrow x \notin \bar{\mathcal{A}}(t) \tag{4.55}$$

First, observe that by the definition of minimal BRS, we have that if any state $\bar{x} \in \mathcal{A}(t)$, then

$$\forall \tau \in [t,T]], \forall u(\cdot) \in \mathbb{U}, \zeta(\tau; \bar{x}, t, u(\cdot)) \in \mathcal{A}(\tau) \tag{4.56}$$

since otherwise, we would have for some $\bar{\tau} \in [t,T]]$,

$$\begin{aligned} &\exists u(\cdot) \in \mathbb{U}, \zeta(\bar{\tau}; \bar{x}, t, u(\cdot)) \notin \mathcal{A}(\bar{\tau}) \\ &\Rightarrow \exists u(\cdot) \in \mathbb{U}, \zeta(T]; \zeta(\bar{\tau}; \bar{x}, t, u(\cdot)), \bar{\tau}, u(\cdot)) \notin \mathcal{L} \\ &\Leftrightarrow \exists u(\cdot) \in \mathbb{U}, \zeta(T]; \bar{x}, t, u(\cdot)) \notin \mathcal{L} \end{aligned} \tag{4.57}$$

which contradicts $\bar{x} \in \mathcal{A}(t)$.

Given $x \notin \mathcal{A}(t)$, there exists some control $\bar{u}(\cdot)$ such that $\zeta(T]; x, t, \bar{u}(\cdot)) \notin \mathcal{L} = \mathcal{A}(T])$. Moreover, we must have $\forall \tau \in [t,T]], \zeta(\tau; x, t, \bar{u}(\cdot)) \notin \mathcal{L}$, since otherwise, we would have $\exists \hat{\tau}$ such that

$$\begin{aligned} &\zeta(\hat{\tau}; x, t, \bar{u}(\cdot)) \in \mathcal{L} = \mathcal{A}(T]) \\ &\Rightarrow x = \zeta(t; x, t, \bar{u}(\cdot)) \in \mathcal{A}(t - \hat{\tau}) \end{aligned} \tag{4.58}$$

which contradicts $x \notin \bigcup_{\tau \in (t,T])} \mathcal{A}(\tau)$.

Using time-invariance of the system dynamics, we have $\forall \tau \in [t,T]], \zeta(T]; x, t-\tau, \bar{u}(\cdot)) \notin \mathcal{L}$, which is equivalent to $\forall \tau \in [t,T]], \zeta(T]; x, \tau, \bar{u}(\cdot)) \notin \mathcal{L}$. Therefore, $\exists u(\cdot) \in \mathbb{U}, \forall \tau \in [t,T]], \zeta(T]; x, \tau, \bar{u}(\cdot)) \notin \mathcal{L} \Leftrightarrow x \notin \bar{\mathcal{A}}(t)$. $\square$

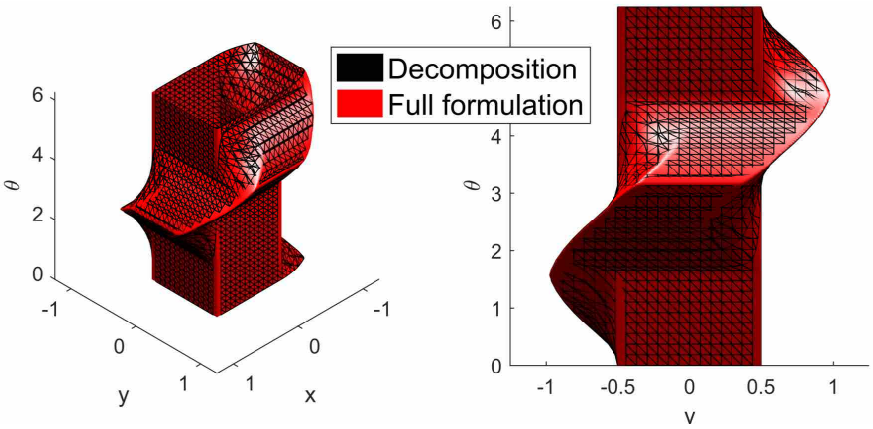


Figure 4.7: The BRT computed directly in 3D (red surface) and computed via decomposition in 2D (black mesh). Using our decomposition technique, we first compute the BRSs $\mathcal{A}(\tau), \tau \in [-0.5, 0]$, and then obtain the BRT by taking their union.

Remark 6. *The way to compute the union of BRSs over a continuous time interval depends on the reachability formulation. In the case of HJ reachability, which is used to obtain numerical examples in this chapter, time is discretized to satisfy the Courant-Friedrichs-Lewy (CFL) condition, ensuring the underlying numerical methods are stable. We save sets at each time interval, resulting in a series of look-up tables of dimension n, where n is the number of states; the number of such look-up tables is the number of discrete time steps. As in [126], sets are represented by the zero sublevel sets of functions, so taking the union of sets amounts to taking an element-wise minimum between a finite number of different look-up tables corresponding to each time interval.*

Remark 7. *When $\exists \tau \in [t, T]], \mathcal{A}(\tau) = \emptyset$, it is currently not known whether the union of the BRSs $\mathcal{A}(\tau)$ will be equal to the BRT $\bar{\mathcal{A}}(t)$ or a proper subset of the BRT $\bar{\mathcal{A}}(t)$. Both are possibilities. Finding a weaker condition under which the union of BRSs is equal to the BRT is an important future direction that we plan to investigate.*

Remark 8. *Note that Proposition 5 and Theorem 3 also hold for decoupled control.*

A summary of the results from this section can be seen in Table 4.2 under Section 4.5, and Table 4.3 under Section 4.5.

Numerical Results

We now revisit the Dubins Car, whose full system and subsystem dynamics are given in (4.6) and (4.7) respectively. Using the target set $\mathcal{L}$ given in (4.27) and writing $\mathcal{L}$ in the form of (4.28), we computed the BRT $\bar{\mathcal{A}}(t), t = -0.5$ by first computing $\mathcal{A}(\tau), \tau \in [-0.5, 0]$, and then taking their union.

Fig. 4.7 shows the BRT $\bar{\mathcal{A}}(t), t = -0.5$ computed directly in 3D and via decomposition. Since $\mathcal{A}(\tau) \neq \emptyset \ \forall \tau \in [-0.5, 0]$, reconstruction does not incur additional approximation errors.

4.6 High-Dimensional Numerical Results

In this section, we show numerical results for the 6D Acrobatic Quadrotor and the 10D Near-Hover Quadrotor, two systems whose exact BRSs and BRTs were intractable to compute with previous methods to the best of our knowledge.

The 6D Acrobatic Quadrotor

In [80], 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 $z = (p_x, v_x, p_y, v_y, \phi, \omega)$, and dynamics

$$\begin{bmatrix} \dot{p}_x \\ \dot{v}_x \\ \dot{p}_y \\ \dot{v}_y \\ \dot{\phi} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} v_x \\ -\frac{1}{m} C_D^v v_x - \frac{T_1}{m} \sin\phi - \frac{T_2}{m} \sin\phi \\ v_y \\ -\frac{1}{m} \left(mg + C_D^v v_y\right) + \frac{T_1}{m} \cos\phi + \frac{T_2}{m} \cos\phi \\ \omega \\ -\frac{1}{I_{yy}} C_D^\phi \omega - \frac{l}{I_{yy}} T_1 + \frac{l}{I_{yy}} T_2 \end{bmatrix} \tag{4.59}$$

where p_x, p_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.

We decompose the system into the following subsystems:

$$x_1 = (p_x, v_x, \phi, \omega) \tag{4.60}$$

$$x_2 = (p_y, v_y, \phi, \omega) \tag{4.61}$$

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 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 $l(z)$ such that $l(z) \leq 0 \Leftrightarrow z \in \mathcal{L}$. This target set is then decomposed as follows:

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

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

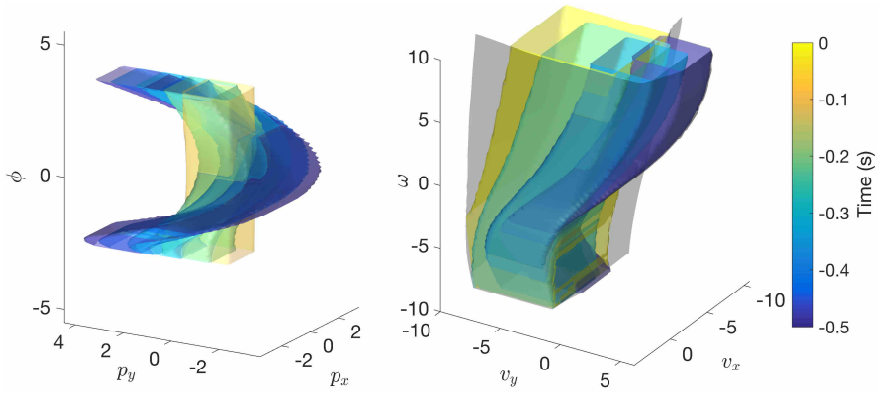


Figure 4.8: Left: 3D positional slices of the reconstructed 6D BRSs at $v_x = v_y = 1$, $\omega = 0$ at different points in time. The BRT cannot be seen in this image because it encompasses the entire union of BRSs. Right: 3D velocity slices of the reconstructed 6D BRSs at $p_x, p_y = 1.5$, $\phi = 1.5$ at different points in time. The BRT can be seen as the transparent gray surface that encompasses the sets.

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. 4.8 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. 4.8 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. In the case of HJ reachability, time is discretized, and sets are represented via the zero sublevel set of an implicit surface function [126]. Thus, the union in Theorem 3 for obtaining the BRT is computed by taking the element-wise minimum between the stored BRSs at each discretized time step in the time interval. The BRT can be seen in Fig. 4.8 as the gray surface.

The 10D Near-Hover Quadrotor

The 10D Near-Hover Quadrotor was used for experiments involving learning-based MPC [29]. Its dynamics are

$$\begin{bmatrix} \dot{p}_x \\ \dot{v}_x \\ \dot{\theta}_x \\ \dot{\omega}_x \\ \dot{p}_y \\ \dot{v}_y \\ \dot{\theta}_y \\ \dot{\omega}_y \\ \dot{p}_z \\ \dot{v}_z \end{bmatrix} = \begin{bmatrix} v_x + d_x \\ g \tan \theta_x \\ -d_1 \theta_x + \omega_x \\ -d_0 \theta_x + n_0 S_x \\ v_y + d_y \\ g \tan \theta_y \\ -d_1 \theta_y + \omega_y \\ -d_0 \theta_y + n_0 S_y \\ v_z + d_z \\ k_T T_z - g \end{bmatrix} \tag{4.62}$$

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:

$$x_1 = (p_x, v_x, \theta_x, \omega_x) \tag{4.63}$$
$$x_2 = (p_y, v_y, \theta_y, \omega_y) \tag{4.64}$$
$$x_3 = (p_z, v_z) \tag{4.65}$$

The target set is chosen to be

$$\begin{aligned} \mathcal{L} = \{(p_x, v_x, \theta_x, \omega_x, p_y, v_y, \theta_y, \omega_y, p_z, v_z) : \\ |p_x|, |p_y| \leq 1, |p_z| \leq 2.5\} \end{aligned} \tag{4.66}$$

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

$$\begin{aligned} \mathcal{L}_1 &= \{(p_x, v_x, \theta_x, \omega_x) : |p_x| \leq 1\} \\ \mathcal{L}_2 &= \{(p_y, v_y, \theta_y, \omega_y) : |p_y| \leq 1\} \\ \mathcal{L}_3 &= \{(p_z, v_z) : |p_z| \leq 2.5\} \end{aligned} \tag{4.67}$$

Since the subsystems do not have any common controls or disturbances, and $\mathcal{L} = \bigcap_{i=1}^{3} \text{proj}^{-1}(\mathcal{L}_{x_i})$, we can compute the full-dimensional $\mathcal{R}(t)$ and $\bar{\mathcal{R}}(t)$ by reconstructing lower-dimensional BRSs and BRTs. A discussion of disturbances can be found in Section 4.7.

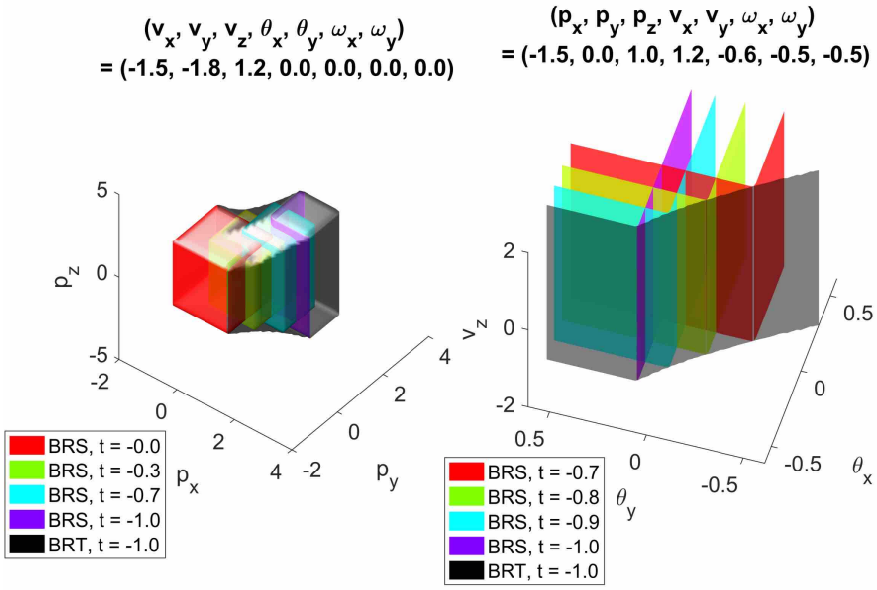


Figure 4.9: 3D slices of the 10D BRSs over time (colored surfaces) and BRT (black surface) for the Near-Hover Quadrotor. The slices are taken at the indicated 7D point.

From the target set, we computed the 10D BRS and BRT, $\mathcal{R}(s), \bar{\mathcal{R}}(s), s \in [-1, 0]$. In the left subplot of Fig. 4.9, we show a 3D slice of the BRS and 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 5.

The BRS and BRT in the position space makes intuitive sense. As expected, the BRSs $\mathcal{A}(s)$ move in the opposite direction of the velocity slice $(v_x, v_y, v_z) = (1.5, -1, -1)$ as s goes backward in time (decreases). In addition, the BRSs $\mathcal{A}(s)$ become smaller as s goes backward in time, which makes sense since the farther the quadrotor is away from the obstacle, the more time it has to apply a control to avoid the obstacle, and hence the set of unsafe states is smaller.

The right subplot of Fig. 4.9 shows the BRS and 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 BRT is not possible to obtain using previous methods, since a high-dimensional system model like (4.62) is needed for analyzing the angular behavior of the system.

4.7 Handling disturbances

Under the presence of disturbances, the full system dynamics changes from (4.1) to

$$\frac{dx}{d\tau} = \dot{x} = f(x, u, d), \tau \in [t, T], u \in \mathcal{U}, d \in \mathcal{D} \tag{4.68}$$

where $d \in \mathcal{D}$ represents the disturbance, with $d(\cdot) \in \mathbb{D}$ drawn from the set of measurable functions.

In addition, we assume that the disturbance function $d(\cdot)$ is drawn from the set of non-anticipative strategies [126], denoted $\Gamma(t)$. We denote the mapping from $u(\cdot)$ to $d(\cdot)$ as $d(\cdot)$ as in [126]. The subsystems in (4.5) are now written as

$$\begin{aligned} \dot{z}_1 &= f_1(z_1, z_c, u, d) \\ \dot{z}_2 &= f_2(z_2, z_c, u, d) \\ \dot{z}_c &= f_c(z_c, u, d) \end{aligned} \tag{4.69}$$

In general, subsystem disturbances may be coupled. Whether this is the case is very important, as some of the results involving disturbances become stronger when the subsystem disturbances are independent.

Trajectories of the system and subsystems are now denoted

$$\zeta(\tau; x, t, u(\cdot), d(\cdot)), \xi_i(\tau; x_i, t, u(\cdot), d(\cdot)),$$

and satisfy conditions analogous to (4.2) and (4.11) respectively. We also need to incorporate the disturbance into the BRS and BRT definitions:

$$\begin{aligned} \bar{\mathcal{A}}(t) = \{x : \exists d(\cdot), \forall u(\cdot), \exists \tau \in [t, T], \\ \zeta(\tau; x, t, u(\cdot), d(\cdot)) \in \mathcal{L}\} \\ \bar{\mathcal{R}}(t) = \{x : \forall d(\cdot), \exists u(\cdot), \exists \tau \in [t, T], \\ \zeta(\tau; x, t, d(\cdot), d(\cdot)) \in \mathcal{L}\} \\ \mathcal{A}(t) = \{x : \exists d(\cdot), \forall u(\cdot), \zeta(T; x, t, u(\cdot), d(\cdot)) \in \mathcal{L}\} \\ \mathcal{R}(t) = \{x : \forall d(\cdot), \exists u(\cdot), \zeta(T; x, t, u(\cdot), d(\cdot)) \in \mathcal{L}\} \end{aligned} \tag{4.70}$$

Subsystem BRSs $\mathcal{R}_i, \mathcal{A}_i, i = 1, 2$ are defined analogously.

Self-Contained Subsystems

Under the presence of disturbances, the results from Section 4.3 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.

Corollary 3. *Suppose that the full system in* (4.1) *can be decomposed into the form of* (4.69)*, then*

$$\begin{aligned} &\mathcal{L} = proj^{-1}(\mathcal{L}_1) \cup proj^{-1}(\mathcal{L}_2) \\ &\Rightarrow \mathcal{R}(t) \supseteq proj^{-1}(\mathcal{R}_1(t)) \cup proj^{-1}(\mathcal{R}_2(t)) \end{aligned} \tag{4.71}$$

To solve this, in the proof of Theorem 1, (4.24) *becomes*

$$\begin{aligned} \forall d(\cdot), \exists u(\cdot), \quad & \xi_1(T; \bar{x}_1, t, u(\cdot), d(\cdot)) \in \mathcal{L}_1 \vee \\ & \xi_2(T; \bar{x}_2, t, u(\cdot), d(\cdot)) \in \mathcal{L}_2 \end{aligned} \tag{4.72}$$

The expression "$\forall d(\cdot), \exists u(\cdot)$" can no longer be distributed, thus making 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.

Corollary 4. *Suppose that the full system in* (4.1) *can be decomposed into the form of* (4.69)*, then*

$$\begin{aligned} & \mathcal{L} = proj^{-1}(\mathcal{L}_1) \cap proj^{-1}(\mathcal{L}_2) \\ & \Rightarrow \mathcal{A}(t) \subseteq proj^{-1}(\mathcal{A}_1(t)) \cap proj^{-1}(\mathcal{A}_2(t)) \end{aligned} \tag{4.73}$$

The proof of Theorem 2 makes the same arguments 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.

If the subsystem disturbances have no shared components, then (4.72) becomes

$$\begin{aligned} \forall \left(\gamma_1[u](\cdot), \gamma_2[u](\cdot)\right), \exists u(\cdot), \quad & \xi_1(T; \bar{x}_1, t, u(\cdot), \gamma_1[u](\cdot)) \in \mathcal{L}_1 \vee \\ & \xi_2(T; \bar{x}_2, t, u(\cdot), \gamma_2[u](\cdot)) \in \mathcal{L}_2 \end{aligned}$$

where $d(\cdot)$ is written as $(\gamma_1[u](\cdot), \gamma_2[u](\cdot))$.

In this case, the expression "$\forall \left(\gamma_1[u](\cdot), \gamma_2[u](\cdot)\right), \exists u(\cdot)$" can be distributed. Therefore, in this case Theorems 1 and 2 still hold.

Subsystems with Decoupled Control

For systems with decoupled control, but coupled disturbance in the subsystems, results from Section 4.7 still hold since the system dynamics structure is a special case of that in Section 4.7. In addition, results from Section 4.4 hold with some modifications. Propositions 2 and 3 need to be modified, and again the reconstructed BRS is now an approximation conservative in the right direction.

Corollary 5. *Suppose that the full system in* (4.1) *can be decomposed into the form of* (4.29)*, with the addition of coupled disturbances. Then,*

$$\begin{aligned} & \mathcal{L} = proj^{-1}(\mathcal{L}_1) \cap proj^{-1}(\mathcal{L}_2) \\ & \Rightarrow \mathcal{R}(t) \supseteq proj^{-1}(\mathcal{R}_1(t)) \cap proj^{-1}(\mathcal{R}_2(t)) \end{aligned} \tag{4.74}$$

To prove this, we modify Proposition 2 by changing (4.38) *to*

$$\begin{aligned}&\forall d(\cdot), \exists\, (u_1(\cdot), u_2(\cdot)) \\ &\quad \big(\eta_1(\tau; \bar{z}_1, \bar{z}_c, t, u_1(\cdot), d(\cdot)), \eta_c(\tau; \bar{z}_c, t)\big) \in \mathcal{L}_1 \wedge \\ &\quad \big(\eta_2(\tau; \bar{z}_2, \bar{z}_c, t, u_2(\cdot), d(\cdot)), \eta_c(\tau; \bar{z}_c, t)\big) \in \mathcal{L}_2\end{aligned} \tag{4.75}$$

The expression "$\forall d(\cdot), \exists\, (u_1(\cdot), u_2(\cdot))$*" cannot be distributed to lead to a statement analogous to* (4.39)*. Hence, the forward direction of Proposition 2 does not hold, and conservativeness is introduced.*

By the same reasoning, the result of Proposition 3 changes to the following.

Corollary 6.

$$\begin{aligned}&\mathcal{L} = \mathit{proj}^{-1}(\mathcal{L}_1) \cup \mathit{proj}^{-1}(\mathcal{L}_2) \\ &\Rightarrow \mathcal{A}(t) \subseteq \mathit{proj}^{-1}(\mathcal{A}_1(t)) \cup \mathit{proj}^{-1}(\mathcal{A}_2(t))\end{aligned} \tag{4.76}$$

In both cases, conservative approximations of the BRS can still be obtained. A summary of the results from this section can be seen in Table 4.1 under Sections 4.7 & 4.7.

Decomposition of Reachable Tubes

Under disturbances, the results from Section 4.5 carry over with modifications. For reconstruction from other BRTs, the arguments in Proposition 4 do not change. However, in the case where there is coupling in the subsystem disturbances, the reconstructed BRTs become conservative approximations.

Corollary 7. *Suppose our system has coupled control and disturbance as in* (4.69)*, then*

$$\begin{aligned}&\mathcal{L} = \mathit{proj}^{-1}(\mathcal{L}_1) \cup \mathit{proj}^{-1}(\mathcal{L}_2) \\ &\Rightarrow \bar{\mathcal{R}}(t) \supseteq \mathit{proj}^{-1}(\bar{\mathcal{R}}_1(t)) \cup \mathit{proj}^{-1}(\bar{\mathcal{R}}_2(t))\end{aligned} \tag{4.77}$$

Corollary 8. *Suppose our system has independent subsystem controls, then*

$$\begin{aligned}&\mathcal{L} = \mathit{proj}^{-1}(\mathcal{L}_1) \cup \mathit{proj}^{-1}(\mathcal{L}_2) \\ &\Rightarrow \bar{\mathcal{A}}(t) \subseteq \mathit{proj}^{-1}(\bar{\mathcal{A}}_1(t)) \cup \mathit{proj}^{-1}(\bar{\mathcal{A}}_2(t))\end{aligned} \tag{4.78}$$

For Proposition 5, the union of the BRSs now becomes an under-approximation of the BRT in general:

Corollary 9. *Suppose our system has coupled control and disturbance as in* (4.69)*, then*

$$\begin{aligned}&\mathcal{L} = \mathit{proj}^{-1}(\mathcal{L}_1) \cap \mathit{proj}^{-1}(\mathcal{L}_2) \\ &\Rightarrow \bigcup_{\tau \in [t,T]} \mathcal{R}(\tau) \subseteq \bar{\mathcal{R}}(t)\end{aligned} \tag{4.79}$$

To show this, all arguments in the proof of Proposition 5 remain the same, except (4.45) *no longer implies* (4.49)*. Instead, the implication is unidirectional:*

$$\begin{aligned} &\exists \tau \in [t, T], \forall d(\cdot), \exists u(\cdot), \zeta(T; x, \tau, u(\cdot), d(\cdot)) \in \mathcal{L} \\ &\Rightarrow \forall d(\cdot), \exists u(\cdot), \exists \tau \in [t, T], \zeta(\tau; x, t, u(\cdot), d(\cdot)) \in \mathcal{L} \end{aligned} \tag{4.80}$$

This is due to the switching of the order of the expressions "$\exists \tau \in [t, T]$" and "$d(\cdot)$". 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.

In contrast to Proposition 5, all the arguments of Theorem 3 hold, since there is no change of order of expressions involving existential and universal quantifiers. A summary of the results from this section can be seen in Table 4.2 & 4.3 under Section 4.7.

Dubins Car with Disturbances

Under disturbances, the Dubins Car dynamics are given by

$$\begin{aligned} \begin{bmatrix} \dot{p}_x \\ \dot{p}_y \\ \dot{\theta} \end{bmatrix} &= \begin{bmatrix} v \cos\theta + d_x \\ v \sin\theta + d_y \\ \omega + d_\theta \end{bmatrix} \\ \omega \in \mathcal{U}, \quad &(d_x, d_y, d_\theta) \in \mathcal{D} \end{aligned} \tag{4.81}$$

with state $x = (p_x, p_y, \theta)$, control $u = \omega$, and disturbances $d = (d_x, d_y, d_\theta)$. The state partitions are $z_1 = p_x, z_2 = p_y, z_c = \theta$. The subsystems dynamics are as follows:

$$\begin{aligned} \dot{x_1} = \begin{bmatrix} \dot{z_1} \\ \dot{z_c} \end{bmatrix} = \begin{bmatrix} \dot{p_x} \\ \dot{\theta} \end{bmatrix} &= \begin{bmatrix} v \cos\theta + d_x \\ \omega + d_\theta \end{bmatrix} \\ \dot{x_2} = \begin{bmatrix} \dot{z_2} \\ \dot{z_c} \end{bmatrix} = \begin{bmatrix} \dot{p_y} \\ \dot{\theta} \end{bmatrix} &= \begin{bmatrix} v \sin\theta + d_y \\ \omega + d_\theta \end{bmatrix} \\ u_c &= \omega = u \\ d_1 = d_x, d_2 &= d_y, d_c = d_\theta \end{aligned} \tag{4.82}$$

where the overlapping state is $\theta = z_c$. 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 b_1 and b_2 have the shared component d_θ.

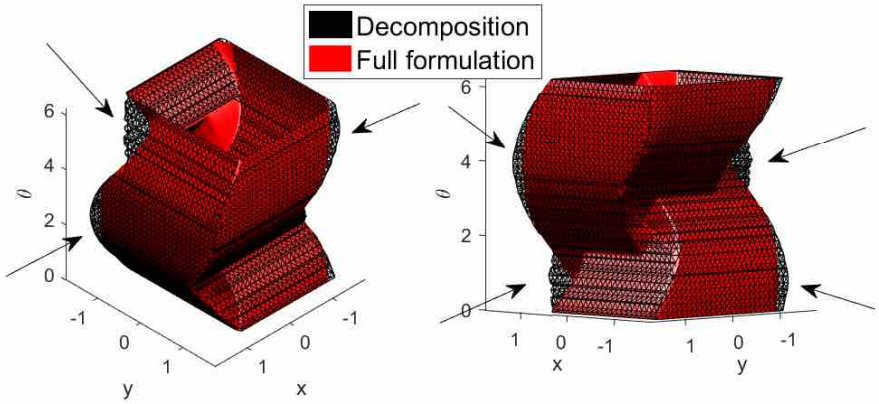


Figure 4.10: Minimal BRTs computed directly in 3D and via decomposition in 2D for the Dubins Car under coupled subsystem disturbances. The reconstructed BRT is an over-approximation of the true BRT. The over-approximated regions of the reconstruction are indicated by the arrows.

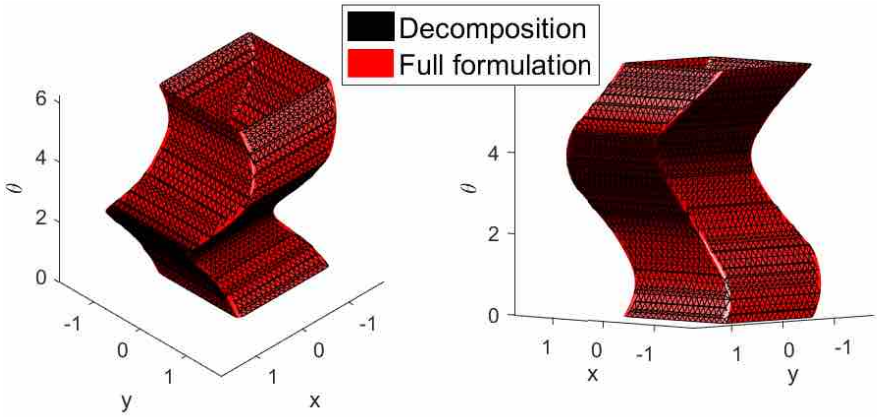


Figure 4.11: Minimal BRTs computed directly in 3D and via decomposition in 2D for the Dubins Car under independent subsystem disturbances. In this case, the BRT computed using decomposition matches the true BRT.

Fig. 4.10 compares the BRT $\bar{\mathcal{A}}(t), t = -0.5$ computed directly from the target set in (4.27), and using our decomposition technique from the subsystem target sets in (4.28), wherein the union is taken over a discrete number of BRSs over the time interval. For this computation, we chose $\bar{d}_x, \bar{d}_y = 1, \bar{d}_\theta = 5$.

Since there is coupling 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. 4.10.

Fig. 4.11 shows the same computation with $\bar{d}_\theta = 0$, so that subsystem disturbances are independent. In this case, one can see that the BRTs computed directly in 3D and via decomposition in 2D are the same.

4.8 Chapter Summary

In this chapter, we presented a general system decomposition method for efficiently computing BRSs and BRTs in several scenarios. By performing computations in lower-dimensional subspaces, computation burden is substantially reduced, allowing currently tractable computations to be orders of magnitude faster, and currently intractable computations to become tractable. Unlike related work on computation of BRSs and BRTs, our method can significantly reduce dimensionality without sacrificing any optimality.

Under disturbances, the reconstructed BRSs and BRTs sometimes become slightly conservative approximations which are still useful for providing performance and safety guarantees. To the best of our knowledge, such guarantees for high-dimensional systems are now possible for the first time. Our decomposition technique can also be used in combination with other dimensionality reduction or approximation techniques, further alleviating the curse of dimensionality.

We are currently extending our decomposition technique to other scenarios, including more representations of full-dimensional sets in lower-dimensional subspaces and more families of system dynamics. In addition, we look forward to combining our technique with other related techniques such as reinforcement learning and machine learning, automating the system decomposition process, and demonstrating our theory in hardware experiments.

Chapter 5

Warm-Start Reachability

This chapter is based on the paper "Reachability-Based Safety Guarantees using Efficient Initializations" [87], written in collaboration with Shromona Ghosh, Somil Bansal, and Claire J. Tomlin.

As humanity increasingly relies on autonomous systems, ensuring provable safety guarantees and controllers for these systems is vital. To achieve safety for nonlinear systems, tools such as Hamilton-Jacobi-Isaacs (HJI) reachability analysis can provide both a guarantee and a corresponding control input [17, 126]. Applications include collision avoidance [126, 39], safe tracking of online motion planners [88, 107], stormwater management [32], and administering anesthesia [99], and others [21, 91, 57]. 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. In these situations the safety analysis must be updated.

Unfortunately, performing HJI reachability analysis is computationally intensive for large systems and cannot be computed efficiently as new information is acquired. There are some methods for speeding up this computation using decomposition [38], and there are other efficient approaches that require simplified problem formulations and/or dynamics [74, 83, 111, 112, 120, 136, 100]. The methods in [6, 42, 58, 74, 122], can handle more complex dynamics, but may be less scalable or unable to represent complex sets. Efficient reachability analysis remains challenging for general system dynamics and problem setups.

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 [3, 70]. 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.

Until now there were no guarantees of convergence for warm-starting HJ reachability without using a discount factor. In this chapter we prove that warm-start reachability with no discount factor 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 [129]).

In addition to these proofs, we provide several common problem classes for which we can prove this exact convergence. We demonstrate these results on an illustrative example with a double integrator, and a more practical example of a realistic 10D quadcopter model that experiences changes in mass and disturbances and must update its safety guarantees accordingly. In these examples warm-start reachability is 1.6 times faster than standard reachability and 6.2 times faster than (untuned) discounted reachability formulation.

5.1 Problem Formulation

Consider an autonomous agent in an environment in the presence of external disturbance. This environment contains a target set $\mathcal{L}$ that is meaningful to the agent: it can be either a set of goal states, or a set of unsafe states. HJI reachability seeks to find the set of initial states for which the system, acting optimally and under worst-case disturbances, will end up in the target set $\mathcal{L}$ either at a particular time (backward reachable set, or BRS) or within a time horizon (backward reachable tube, or BRT). Optimal behavior of the system depends on the nature of the target set and can be formulated as a game: for a goal set, the control will seek to minimize distance to the goal whereas the worst-case disturbance will maximize distance to the goal. For an unsafe set, the control will maximize and the disturbance will minimize. Both cases (and various combinations of cases) can be solved using HJI reachability analysis.[1]

The theory in this chapter applies to BRTs with infinite-time horizons. Typically this scenario is more interesting in the avoid case (where the system seeks to avoid an unsafe set of states forever), and will therefore be the focus of this chapter. In this section we define the agent's dynamics and formally introduce HJI reachability analysis.

[1]Note that there are combination reach-avoid problems that seek to reach a goal set while avoiding unsafe sets. There are also problems that use *forward* reachable sets and tubes. More information can be found in [17].

Dynamic System Model

We assume that the autonomous system (i.e. agent) has initial state $x \in \mathbb{R}^n$ and initial time t, and evolves according to the ordinary differential equation (ODE):

$$\dot{x} = f(x, u, d), \quad u \in \mathcal{U}, d \in \mathcal{D}. \tag{5.1}$$

Here the system has a control u and disturbance d. We assume that these inputs are drawn from compact sets ($\mathcal{U}$, $\mathcal{D}$), and their signals over time ($u(\cdot)$, $d(\cdot)$) are drawn from the set of measurable functions $\mathbb{U} : [t, T] \to \mathcal{U}$ and $\mathbb{D} : [t, T] \to \mathcal{D}$.

We assume that the flow field $f : \mathbb{R}^n \times \mathcal{U} \times \mathcal{D} \to \mathbb{R}^n$ is uniformly continuous and Lipschitz continuous in x for fixed u and d. Under these assumption there exists a unique solution of these system dynamics for a given $u(\cdot), d(\cdot)$ [46], providing trajectories of the system: $\zeta(\tau; x, t, u(\cdot), d(\cdot))$. This notation can be read as 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 $[t, \tau]$. For compactness we will refer to trajectories using $\zeta_{x,t}^{u,d}(\tau)$. Because we tend to solve reachability problems backwards in time, we use the notation that forward trajectories end at final time $\tau = T$, and start at an initial negative time t.

Running example: *In this chapter we use a double integrator as a running example. Its system dynamics are:*

$$\dot{x} = \begin{bmatrix} \dot{p} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} v + d \\ ub \end{bmatrix}, \tag{5.2}$$

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.

Hamilton-Jacobi-Isaacs Reachability

Defining the Value Function

We define a target function $l(x)$ whose sub-zero level set is the target set $\mathcal{L}$ describing the unsafe states, i.e. $\mathcal{L} = \{x : l(x) \leq 0\}$. Typically $l(x)$ is defined as a signed distance function that measures distance to $\mathcal{L}$. This can be considered as a measure of reward that is positive outside of the unsafe set and negative inside.

This problem formulation seeks to find all trajectories that will enter $\mathcal{L}$ at any point in the time horizon, and therefore become unsafe. This is computed by finding the minimum reward (and therefore minimum distance to $\mathcal{L}$) over time:

$$J(x, t, u(\cdot), d(\cdot)) = \min_{\tau \in [t, 0]} l(\zeta_{x,t}^{u,d}(\tau)), \ t \leq 0. \tag{5.3}$$

More specifically, the goal is to capture this minimum reward for *optimal trajectories* of the system. To do this we optimize for the optimal control signal that maximizes the reward

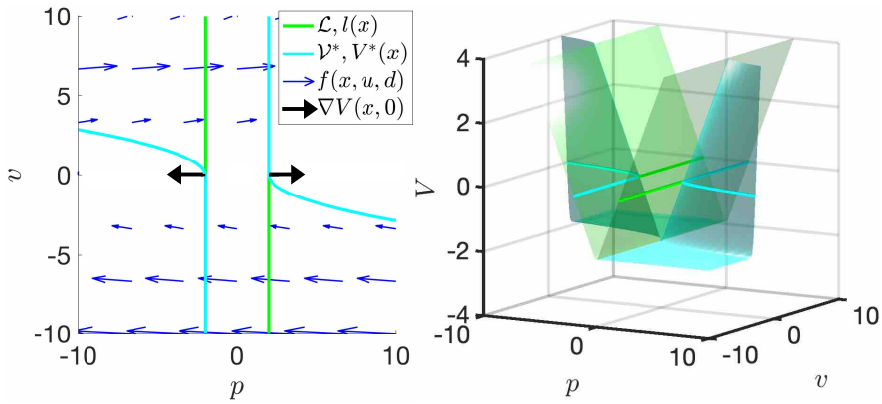


Figure 5.1: Visualization of the running example using a double integrator model. The target set $\mathcal{L}$ and corresponding function $l(x)$ are in green. We initialize $V(x,0) = l(x)$, and update the function using (5.5) by optimizing over the inner product between the spatial gradients (seen for $V(x,0)$ 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.

(and drives the system away from the unsafe target set) and the worst-case disturbance signal that minimizes the reward. This leads to the value function:[2]

$$V(x,t) = \inf_{d(\cdot)} \sup_{u(\cdot)} \Big\{ J\Big(x, t, u(\cdot), d(\cdot)\Big) \Big\}. \tag{5.4}$$

Level sets of the value function correspond to level sets of the target function. If a state has a negative value, optimal trajectories starting from that state achieve negative reward at some point in their trajectory, meaning they have entered the target set $\mathcal{L}$ sometime within the time horizon. Therefore, the sub-zero level set of the value function comprises the backward reachable tube (BRT), notated as $\mathcal{V}$: the set of states from which the system is guaranteed to enter the target set within the time horizon under optimal control and worst-case disturbance. For the infinite-time avoid BRT, if the limit exists, we define the converged value function as $V^*(x) = \lim_{t \to -\infty} V(x,t)$. The sub-zero level set of this converged value function is the infinite-time avoid backwards reachable tube: $\mathcal{V}^* = \{x : V^*(x) \leq 0\}$. Trajectories initialized from states in this set will eventually enter the unsafe target set despite the control's best effort. The complement of this set is therefore the safe set.

Running example: *In the running example the target set is* $\mathcal{L} = \{(p,v) : |p| \leq 2, |v| < \infty\}$. *This set and its corresponding target function* $l(x)$ *can be seen in Fig. 5.1 in green. 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).*

[2]Technically, $d(\cdot)$ in (5.4) is actually $\gamma[u(\cdot)](\cdot)$, where γ maps control inputs to disturbance inputs, $\gamma = \{d : \mathcal{U} \to \mathcal{D}\}$. As in [126], we restrict the disturbance to draw from nonanticipative strategies.

Solving for the Value Function

To solve this optimization problem for the value function, the state space is discretized and the value function is initialized to be equal to the target function, $V(x,0) = l(x)$. The function $V(\cdot,\cdot)$ satisfies the Hamilton-Jacobi-Isaacs variational inequality (HJI VI):

$$\min\Big\{D_t V(x,t) + H\Big(V(x,t), f(x,u,d)\Big), \\ l(x) - V(x,t)\Big\} = 0. \tag{5.5}$$

Here H is the Hamiltonian, which optimizes over the inner product between the spatial gradients of the value function and the flow field of the dynamics to compute the optimal control and disturbance inputs:

$$H\Big(V(x,t), f(x,u,d)\Big) = \\ \max_u \min_d \langle \nabla V(x,t), f(x,u,d)\rangle. \tag{5.6}$$

d

The term $l(x) - V(x,t)$ in (5.5) restricts the value function from becoming more positive than the target function, effectively enforcing that all trajectories that achieve negative reward at any time will continue to have negative reward for the rest of the time horizon. For more details on the derivation of this HJI VI and variations that include forward reachability and reach-avoid scenarios, please refer to [123, 73, 17, 126].

Running example: *For the running example the initial spatial gradients for $V(x,0) = l(x)$ can be seen as black arrows in Fig. 5.1. The Hamiltonian* (5.6) *will optimize over the inner product between these gradients and the flow field of the dynamics $f(x,u,d)$, seen as blue arrows.*

We solve the HJI VI (5.5) using dynamic programming:

$$V(x,t) = \max_{u(\cdot)} \min_{d(\cdot)} \min\Big\{ \inf_{\tau\in[t,t+dt)} l\Big(\zeta_{x,t}^{u,d}(\tau)\Big), \\ V\Big(\zeta_{x,t}^{u,d}(t+dt), t+dt\Big)\Big\}. \tag{5.7}$$

In the limit as the time step $dt \to 0$, this is equivalent to:

$$V(x,t) = \min\Big\{ l\Big(\zeta_{x,t}^{u,d}(t)\Big), \\ \Big[\int_t^{t+dt} H\Big(V(\zeta_{x,t}^{u,d}(\tau),\tau), f(\zeta_{x,t}^{u,d}(\tau),u,d)\Big) d\tau \\ + V\Big(\zeta_{x,t}^{u,d}(t+dt), t+dt\Big)\Big]\Big\}. \tag{5.8}$$

We use (5.7) to update the value of each discretized state backwards in time using the level set method toolbox and associated helperOC toolbox [129, 17]. This update occurs

until the initial time has been reached, or for the infinite-horizon cases considered in this chapter, until convergence. At convergence the infinite-horizon value function's sub-zero level set $\mathcal{V}^*$ corresponds to the set of states that should be avoided in order to remain safe for all time. Online, the system avoids these states by solving for the instantaneous optimal control at state x using the Hamiltonian and inifinite-horizon value function: $u^* = \text{argmax}_u \min_d \langle \nabla V^*(x), f(x,u,d) \rangle$.

Discounted Reachability

In [3] and [70], the authors introduced a discounting factor λ into the cost function (5.3) motivated by the sum of discounted rewards in reinforcement learning. The dynamic programming equation in (5.7) is thus changed to:

$$\begin{aligned} V(x,t) = \max_{u(\cdot)} \min_{d(\cdot)} \min \Big\{ & \inf_{\tau \in [t, t+dt)} l\Big(\zeta_{x,t}^{u,d}(\tau)\Big) \exp(\lambda \cdot \tau), \\ & \exp(\lambda \cdot dt) V\Big(\zeta_{x,t}^{u,d}(t+dt), t+dt\Big) \Big\}, \end{aligned} \tag{5.9}$$

where $t \leq 0$. The authors show that this is a contraction mapping and, hence, $V(x,0)$ can be initialized to any function (not just $l(x)$). The discounting allows $V_l(x,t)$ to *forget* any incorrect initializations over a longer time horizon. However, this formulation can still result in a slow convergence without careful tuning of λ, as we demonstrate in Section 5.4.

5.2 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 $V^*(\cdot,\cdot)$ requires recomputing the entire value function starting with the target function ($V(x,0) = l(x)$). Instead, we initialize with a previous computed (converged) value function.

We define this warm-starting function as $k(x)$, with subzero level set $\mathcal{K} = \{x : k(x) \leq 0\}$. To develop the theory, we revisit the cost function (5.3). We rewrite, $J_l(x,t,u(\cdot),d(\cdot)) = \min \Big\{ \inf_{\tau \in [t,0)} l(\zeta_{x,t}^{u,d}(\tau)), l(\zeta_{x,t}^{u,d}(0)) \Big\}$ and $V_l(x,t)$ is defined as in (5.4) by replacing J by J_l,

$$\begin{aligned} V_l(x,t) &= \max_{u(\cdot)} \min_{d(\cdot)} J_l(x,t,u(\cdot),d(\cdot)) \\ &= \max_{u(\cdot)} \min_{d(\cdot)} \min \Big\{ \inf_{\tau \in [t,0)} l(\zeta_{x,t}^{u,d}(\tau)), V_l(\zeta_{x,t}^{u,d}(0),0) \Big\}, \\ &= \max_{u(\cdot)} \min_{d(\cdot)} \min \Big\{ \inf_{\tau \in [t,0)} l(\zeta_{x,t}^{u,d}(\tau)), l(\zeta_{x,t}^{u,d}(0)) \Big\}, \end{aligned} \tag{5.10}$$

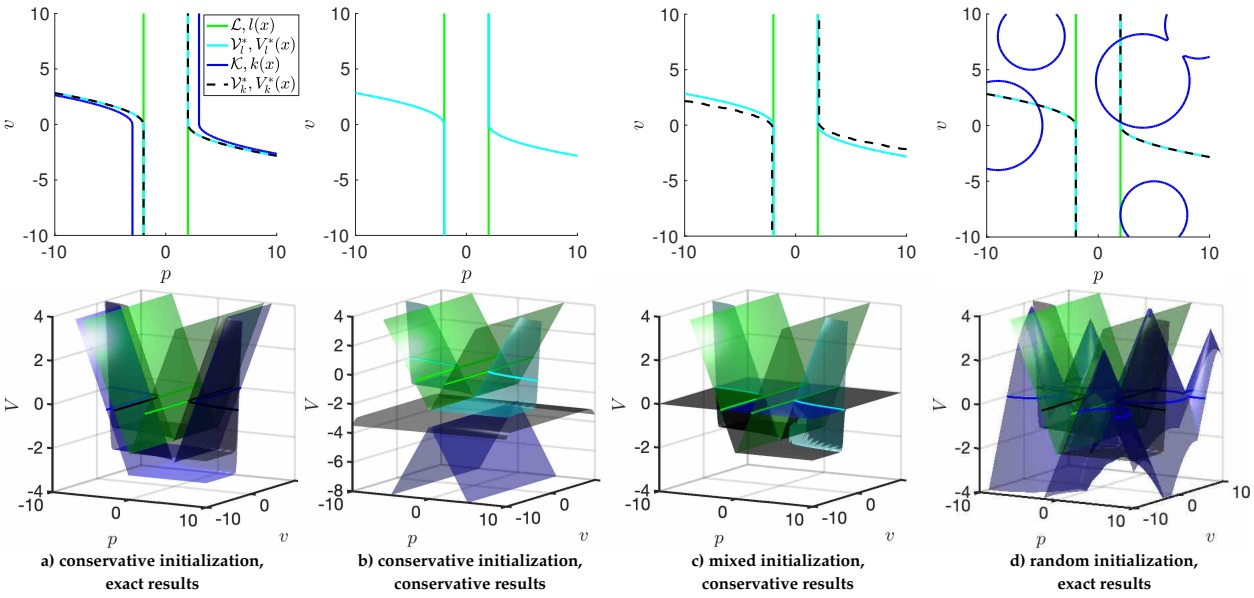


Figure 5.2: The top row shows the target sets and backward reachable tubes, which are the subzero level sets of the target and value functions (bottom row). For all examples shown, green is the target set and function, cyan is the true BRT and converged value function, blue is the warm-start initialization, 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.

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

$$\begin{aligned} 0 = \min\Big\{ & D_t V_l(x,t) + H_l\Big(V_l(x,t), f(x,u,d)\Big), \\ & l(x) - V_l(x,t)\Big\}, \\ H_l(V_l(x,t), f(x,u,d)) = & \\ & \max_u \min_d \langle \nabla V_l(x,t), f(x,u,d)\rangle, \\ V_l(x,0) = l(x,0). & \end{aligned} \tag{5.11}$$

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

$$J_k(x,t,u(\cdot),d(\cdot)) = \min\Big\{ \inf_{\tau\in[t,0)} l(\zeta_{x,t}^{u,d}(\tau)), k(\zeta_{x,t}^{u,d}(0))\Big\}. \tag{5.12}$$

V_k can be defined as in (5.4) with $J = J_k$, i.e

$$\begin{aligned} V_k(x,t) &= \max_{u(\cdot)} \min_{d(\cdot)} J_k(x,t,u(\cdot),d(\cdot)) \\ &= \max_{u(\cdot)} \min_{d(\cdot)} \min \Big\{ \inf_{\tau\in[t,0)} l(\zeta_{x,t}^{u,d}(\tau)), k(\zeta_{x,t}^{u,d}(0)) \Big\}. \end{aligned} \tag{5.13}$$

V_k is the solution to the HJI-VI defined similarly as in (5.11) with $V_k(x,0) = k(x)$. The converged value function is defined as $V_k^*(x) = \lim_{t\to-\infty} V_k(x,t)$.

In this section we prove that the converged value function $V_k^*(x)$ that is initialized as above $V_k(x,0) = k(x)$ will always be more negative than the value function $V_l^*(x)$ achieved by standard reachability (i.e. initialized as $V_l(x,0) = l(x)$). 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.

Conservative Warm-Start Reachability

If $[V_k(x,0) = k(x)] \leq V_l^*(x)$, a contraction mechanism is required to raise $V_k^*(x)$ towards the true solution $V_l^*(x)$. Recall the HJI VI from (5.5). 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 $V_k(x,0) = k(x)$, the result will be conservative: $\forall x, t < 0,\ V_k(x,t) \leq V_l(x,t)$.*

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

$k(x) < l(x)$ For $\forall x, t < 0$, let $V_l(x,t)$ be defined as (5.10) and $V_k(x,t)$ be defined as (5.13). At $t = 0$, we have $\Big[V_k(x,0) = k(x)\Big] < \Big[l(x) = V_l(x,0)\Big] \Rightarrow V_k(x,0) < V_l(x,0)$. For any $t < 0$:

$$\begin{aligned} V_k(x,t) &= \max_{u(\cdot)} \min_{d(\cdot)} \min \Big\{ \inf_{\tau\in[t,0)} l(\zeta_{x,t}^{u,d}(\tau)), k(\zeta_{x,t}^{u,d}(0)) \Big\}, \\ &\leq \max_{u(\cdot)} \min_{d(\cdot)} \min \Big\{ \inf_{\tau\in[t,0)} l(\zeta_{x,t}^{u,d}(\tau)), l(\zeta_{x,t}^{u,d}(0)) \Big\}, \\ &= V_l(x,t). \end{aligned} \tag{5.14}$$

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

$k(x) \geq l(x)$ When $t = 0$, $\Big[V_k(x,0) = k(x)\Big] \geq \Big[l(x) = V_l(x,0)\Big]$. For a time instance $t = 0^-$,

$$\begin{aligned} V_k(x,t) &= \max_{u(\cdot)} \min_{d(\cdot)} \min \Big\{ \inf_{\tau \in [t,0)} l(\zeta_{x,t}^{u,d}(\tau)), k(\zeta_{x,t}^{u,d}(0)) \Big\} \\ &= \min\{l(\zeta_{x,t}^{u,d}(0^-)), k(\zeta_{x,t}^{u,d}(0)\} \\ &= l(\zeta_{x,t}^{u,d}(0^-)) = V_l(x,t). \end{aligned} \tag{5.15}$$

We can re-write (5.10) and (5.13) by replacing 0 by 0^-. The rest follows from proof of case (a). Here 0^- implies an infinitesimally small change in time and we are effectively computing $V_k(x,0^-) = \min(k(x), l(x))$ and treating $V_k(x,0) = V_k(x,0^-)$. One could derive the same proof by considering $V_k(x,0) = \min(k(x), l(x))$. □

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.

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 $V_k(x,0) = k(x)$, such that $\forall x \; k(x) \geq V_l^*(x)$, then, $V_k^*(x) = V_l^*(x)$.*

The proof of Theorem 5 follows from Theorem 4 and Lemma 2, stated as:

Lemma 2. *If $\forall x \; k(x) \geq V_l^*(x)$, we have,*

$$V_k(x,t) \geq V_l^*(x) \quad \forall x, t < 0 \tag{5.16}$$

Proof. To prove Lemma 2, let us consider $k'(x) = V_l^*(x)$. For $t < 0$, using dynamic programming we have,

$$\begin{aligned} V_{k'}(x,t) &= \max_{u(\cdot)} \min_{d(\cdot)} \min \Big\{ \inf_{\tau \in [t,0)} l(\zeta_{x,t}^{u,d}(\tau)), V_{k'}(\zeta_{x,t}^{u,d}(0), 0) \Big\}, \\ &= \max_{u(\cdot)} \min_{d(\cdot)} \min \Big\{ \inf_{\tau \in [t,0)} l(\zeta_{x,t}^{u,d}(\tau)), k'(\zeta_{x,t}^{u,d}(0)) \Big\}. \end{aligned} \tag{5.17}$$

For any $t < 0$, we can follow the same logic as in (5.14). Hence, $\forall x, t$, we have $V_{k'}(x,t) \leq V_k(x,t)$. Moreover, since $V_{k'}(x,0) = V_l^*(x)$, we know that $V_{k'}(x,t) = V_l^*(x) \quad \forall t$ since $V_l^*(x)$ is the converged value function corresponding to $l(x)$. Hence, $V_k(x,t) \geq V_l^*(x) \quad \forall x, t < 0$. Since this holds for all time, it also holds for $t \to -\infty$: $V_k^*(x) \geq V_l^*(x)$. □

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

$$\begin{aligned} V_k^*(x) &\leq V_l^*(x) \quad (Theorem\ 4) \\ V_k^*(x) &\geq V_l^*(x) \quad (Lemma\ 2) \\ \Rightarrow V_k^*(x) &= V_l^*(x) \end{aligned} \tag{5.18}$$

□

5.3 Conservative Warm-Start Examples

Below we demonstrate several scenarios using the running example that result in conservative solutions. All experiments were run on a desktop computer with an Intel Core i7-5280K CPU @3.30GHz ×12 processor and 12.8GB of memory. For all examples the value function is considered converged when the maximum change of value in one time step ($dt = 0.01$) is less than 0.001.

Conservative Initialization with Exact Results

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. 5.2a 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 (5.5) contracting the value function until the solution has been reached.

Conservative Initialization with Conservative Results

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 5.2b. This is a fairly unrealistic initial estimate for the true value function, 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.

Mixed Initialization with Conservative Results

In Fig. 5.2c 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 $\mathcal{V}_l^*$.

Random Initialization with Exact Results

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. 5.2d 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.

5.4 Exact Warm-Start Examples

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 default running example. Each subsection introduces 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. $k \leq V_k(x,t) \leq V_l^*(x)\ \forall x, t$). We show in Table 1 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 [3]).

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. Refer to the Appendix for details. We demonstrate this in Fig. 5.3a, 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 show the reverse case for a decreasing target set in Fig. 5.3b.

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

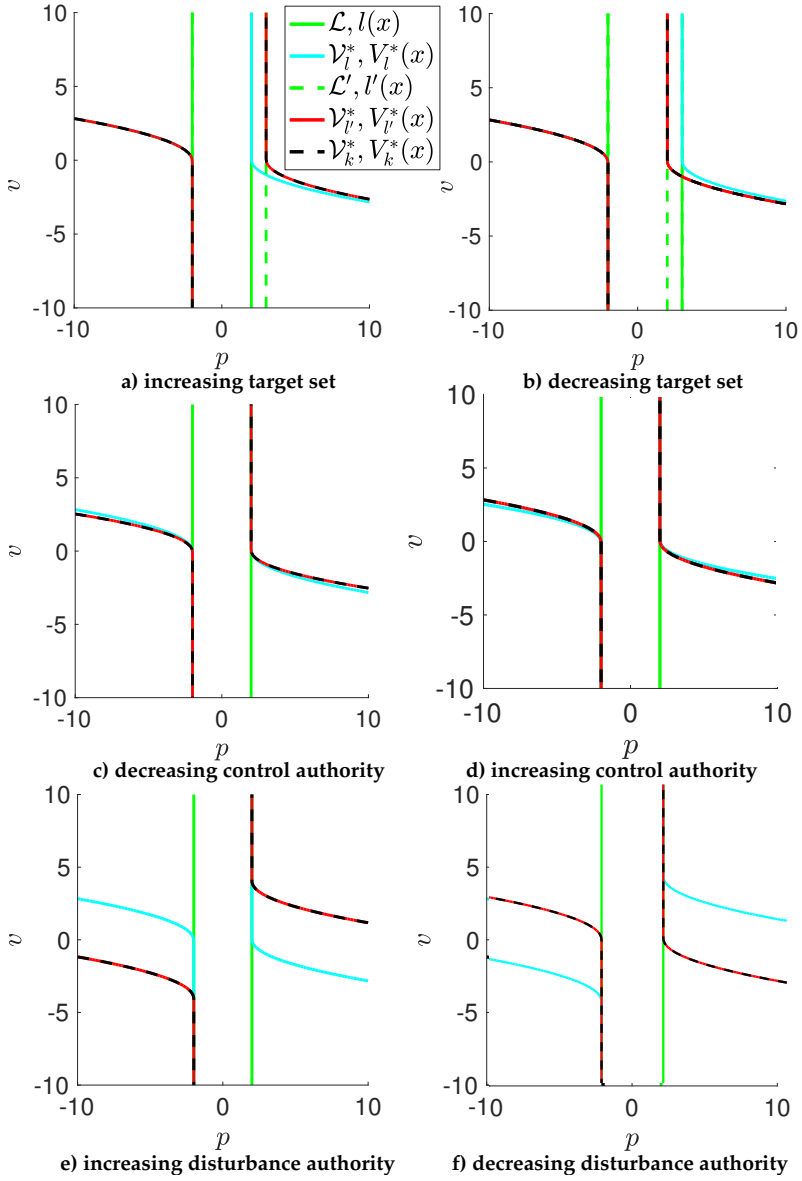


Figure 5.3: For all examples shown, the region between the green lines is the target set. Similarly cyan marks the boundary of the original BRT, red marks the BRT based on new conditions, and black is the boundary of the warm-start converged BRT. The left column shows cases in which the exact solution (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.

Table 1: Runtime Analysis for Reachability Methods

	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

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. Proof is in the Appendix.

To demonstrate this case of reduced control authority we vary the parameter b in the system model (5.2). When b decreases, the effective control authority decreases. In Fig. 5.3c 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 new red value function exactly, as shown in black. We similarly show the reverse case for an increasing control authority in Fig. 5.3d.

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 (5.2). In Fig. 5.3e 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. 5.3f.

Table 1: Runtime Analysis for Reachability Methods

	Standard	Warm-Start	Discounted
a) Increasing $\mathcal{L}$	6.4s 115 steps	6.0s 109 steps	12.5s 231 steps
b) Decreasing $\mathcal{U}$	6.5s 124 steps	6.0s 111 steps	20.3s 374 steps
c) Increasing $\mathcal{D}$	21.4s 311 steps	7.7s 112 steps	13.3s 195 steps
d) 10D quad increasing m, $\mathcal{D}$	3.7hr 86 steps	2.8hr 65 steps	>18 hr >400 steps
e) 10D quad decreasing m, $\mathcal{D}$			

5.5 high-dimensional example

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 from [29, 38]. When the quadcopter experiences changes to its constraints or dynamics (e.g. changes in mass or disturbances), it must update its safety guarantees appropriately.

The 10D near-hover quadcopter dynamics has states (p_x, p_y, p_z) denoting the position, (v_x, v_y, v_z) for velocity, (θ_x, θ_y) for pitch and roll, and (ω_x, ω_y) for pitch and roll rates. Its controls are (S_x, S_y), which respectively represent the desired pitch and roll angle, and T_z, which represents the vertical thrust. The disturbances are (d_x, d_y, d_z) which represents wind, and g is gravity. Its model is:

$$\begin{bmatrix} \dot{p}_x \\ \dot{v}_x \\ \dot{\theta}_x \\ \dot{\omega}_x \\ \dot{p}_y \\ \dot{v}_y \\ \dot{\theta}_y \\ \dot{\omega}_y \\ \dot{p}_z \\ \dot{v}_z \end{bmatrix} = \begin{bmatrix} v_x + d_x \\ g\tan\theta_x \\ -d_1\theta_x + \omega_x \\ -d_0\theta_x + n_0 S_x \\ v_y + d_y \\ g\tan\theta_y \\ -d_1\theta_y + \omega_y \\ -d_0\theta_y + n_0 S_y \\ v_z + d_z \\ (k_T/m)T_z - g \end{bmatrix}. \tag{5.19}$$

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 in [38], we can decompose this 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. Discounted reachability still hadn't converged after 400 steps (18+ hours), with max errors of 0.0034 in p_x, p_y and .325 in p_z.

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 Sec. 5.3, 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 and 82 for discounting (with errors of $8.6e-06$ in the p_x, p_y subsystems and .004 in the p_z subsystem). 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.

5.6 Chapter Summary

Warm-starting infinite-horizon HJI reachability computations with intelligent initializations is beneficial because it may lead to a sizable reduction in computation time by reducing the number of iterations required for convergence. In this chapter we proved that warm-starting will provide guaranteed conservative safety analyses and controllers. Moreover, when the initialization is under-conservative (i.e. $k(x) \geq V_l^*(x)$), we proved that the reachability analysis is guaranteed to converge to the true solution. We also showed several conditions for which exact convergence is achieved, and several cases that will either move closer to the correct safety guarantees or remain conservative with every iteration. In practice we frequently converge to the correct solution regardless of the conservativeness of the initialization.

We demonstrated these results through several examples, including a 10D quadcopter model experiencing changes in mass and disturbance bounds. We were able to accurately recover the updated value function representing the backwards reachable tube in fewer iterations than standard or discounted reachability. For our examples we find that warm-start reachability is 1.6 times faster than standard reachability and 6.2 times faster than (untuned) discounting. For high-dimensional systems this can save hours of computation time.

In many of the examples explored in this chapter finding a good initialization is obvious: with slight changes in assumptions, simply use the previously computed value function. However, in general finding good initialization is not always obvious and would be interesting future work.

This new formulation opens the door to many different methods for solving HJI reachability problems efficiently. One direction would be to numerically parameterize the value function (for example, by different masses), then warm-start online using an interpolated ini-

tialization based on updated problem information. Another exciting direction is to update the value function locally for local changes in the environment, or to use sparse or adaptive gridding of the state space for fast initializations. Finally, we could use the conclusions drawn from this chapter to inform a more tractable formulation of discounted reachability.

5.7 Appendix

Changing the Target Set

For a target function, $l'(x)$, we define $J_{l'}$ and $V_{l'}(x,t)$ as (5.3) and (5.10) by replacing $l(x)$ by $l'(x)$. We define $V_k(x,t)$ as the value function for $\mathcal{L}'$ when we warm-start from $V_l^*(x)$.

Proposition 1. *If $\mathcal{L} \subseteq \mathcal{L}'$ (and hence $l'(x) \leq l(x)$) and we warm-start the value function computation for $\mathcal{L}'$ with $k(x) = V_l^*(x)$, i.e., $V_k(x,0) = V_l^*(x)$, then*

$$\lim_{t \to -\infty} V_k(x,t) = V_{l'}^*(x) \tag{5.20}$$

Proof. To prove this, it suffices to prove that $\forall x\ V_{l'}^*(x) \leq V_l^*$. We have $\forall x, t < 0$,

$$\begin{aligned} V_{l'}(x,t) &= \max_{u(\cdot)} \min_{d(\cdot)} \inf_{\tau \in [t,0]} l'(\zeta_{x,t}^{u,d}(\tau)) \\ &\leq \max_{u(\cdot)} \min_{d(\cdot)} \inf_{\tau \in [t,0]} l(\zeta_{x,t}^{u,d}(\tau)) = V_l(x,t) \end{aligned} \tag{5.21}$$

As $t \to -\infty$, $V_{l'}^*(x) \leq V_l^*(x)$. From Theorem 5, if we warm-start, $V_{l'}(x,0) = k(x)$ where $k(x) \geq V_{l'}^*(x)$, then $V_k^*(x) = V_l^*$. Since, $k(x) = V_l^*(x) \geq V_{l'}^*(x)$, Theorem 5 holds. □

Remark 9. *By reversing the proof with conditions $\mathcal{L} \supseteq \mathcal{L}'$, then*
$k(x) \leq V_k(x,t) \leq V_l^*(x)\ \forall x, t.$

Changing the Control Authority

For a control domain, $\mathcal{U}'$, we define $V_{l'}(x,t)$ similar to (5.10) by replacing $u \in \mathcal{U}$ by $u \in \mathcal{U}'$. We define $V_k(x,t)$ as the value function for $\mathcal{U}'$ when we warm-start from $V_l^*(x)$.

Proposition 2. *If the effective control authority $\mathcal{U}'$ reduces, i.e., $\mathcal{U}' \subseteq \mathcal{U}$ and if $V_k(x,0) = k(x) = V_l^*(x)$, then*

$$\lim_{t \to -\infty} V_k(x,0) = V_{l'}^*(x) \tag{5.22}$$

Proof. To prove this, we need only prove that, $\forall x\ V_{l'}^*(x) \leq V_l^*(x)$. We have, $\forall x, t < 0$,

$$\begin{aligned} V_{l'}(x,t) &= \max_{u \in \mathcal{U}'} \min_{d \in \mathcal{D}} \min \left\{ \inf_{\tau \in [t,0)} l(\zeta_{x,t}^{u,d}(\tau)), l(\zeta_{x,t}^{u,d}(0)) \right\} \\ &\leq \max_{u \in \mathcal{U}} \min_{d \in \mathcal{D}} \min \left\{ \inf_{\tau \in [t,0)} l(\zeta_{x,t}^{u,d}(\tau)), l(\zeta_{x,t}^{u,d}(0)) \right\} \\ &= V_l(x,t) \end{aligned} \tag{5.23}$$

As $t \to -\infty$, $V_{l'}^*(x) \leq V_l^*(x)$. From Theorem 5, if we warm-start with $V_k(x,0) = k(x)$ with $k(x) \geq V_l^*(x)$, then $V_k^*(x) = V_{l'}^*(x)$. Hence, $k(x) = V_{(l}^*(x)$ satisfies Theorem 5. □

Remark 10. *By reversing the proof with conditions $\mathcal{U} \subseteq \mathcal{U}'$, then $k(x) \leq V_k(x,t) \leq V_l^*(x)$ $\forall x, t$.*

Part II

Fast and Safe Tracking

Chapter 6

FaSTrack: Fast and Safe Tracking

This chapter is based on a conference and a journal paper, both titled "FaSTrack: a Modular Framework for Fast and Guaranteed Safe Motion Planning" [88, 43], written in collaboration with Mo Chen, Haimin Hu, Ye Pu, Jaime Fisac, Somil Bansal, SooJean Han, and Claire J. Tomlin. Please note that for this chapter we will use the time notation $t \in [T, 0]$, $T \leq 0$ instead of $\tau \in [t, T]$.

In autonomous dynamical systems, safety and real-time planning are both crucial for many applications. This is particularly true when environments are *a priori* unknown, because replanning based on updated information about the environment is often necessary. However, achieving safe navigation in real time is difficult for many common dynamical systems due to the computational complexity of generating and formally verifying the safety of dynamically feasible trajectories. To achieve real-time planning, many algorithms use highly simplified model dynamics or kinematics to create a nominal trajectory that is then tracked by the system using a feedback controller such as a linear quadratic regulator (LQR). These nominal trajectories may not be dynamically feasible for the true autonomous system, resulting in a tracking error between the planned path and the executed trajectory. This concept is illustrated in Fig. 6.1, where the path was planned using a simplified planning model, but the real dynamical system cannot track this path exactly. Additionally, external disturbances (e.g. wind) can be difficult to account for using real-time planning algorithms, causing another source of tracking error. These tracking errors can lead to dangerous situations in which the planned path is safe, but the actual system trajectory enters unsafe regions. Therefore, real-time planning is achieved at the cost of guaranteeing safety. Common practice techniques augment obstacles by an ad hoc safety margin, which may alleviate the problem but is performed heuristically and therefore does not guarantee safety.

To attain fast planning speed while maintaining safety, we propose the modular framework FaSTrack: Fast and Safe Tracking. FaSTrack also allows planning algorithms to use a simplified model of the system in order to operate in real time using augmented obstacles. However, in FaSTrack, the obstacle augmentation bound is rigorously computed and comes

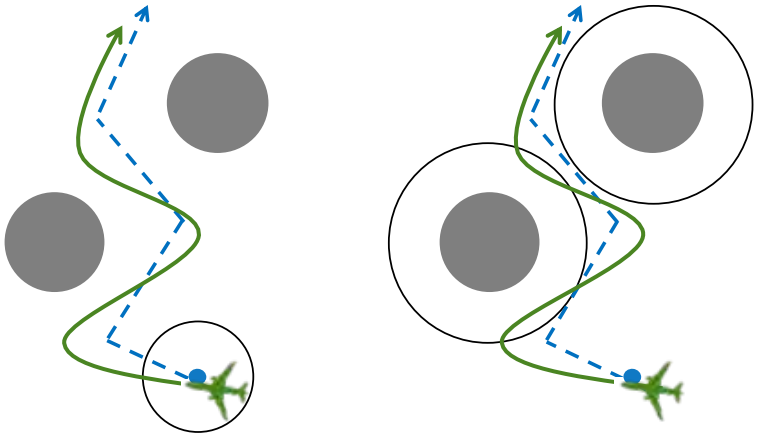


Figure 6.1: Left: A planning algorithm uses a fast but simple model (blue disk), to plan around obstacles (gray disks). The more complicated tracking model (green plane) tracks the path. By using FaSTrack the autonomous system is guaranteed to stay within some TEB (black circle). Right: Safety can be guaranteed by planning with respect to obstacles augmented by the TEB (large black circles).

with a corresponding optimal tracking controller. Together this *tracking error bound* (TEB) and controller guarantee safety for the autonomous system as it tracks the simplified plans (see Fig. 6.1, right). We compute this bound and controller by modeling the navigation task as a pursuit-evasion game between a sophisticated *tracking model* (pursuer) and the simplified *planning model* of the system (evader). The tracking model accounts for complex system dynamics as well as bounded external disturbances, while the simple planning model enables the use of real-time planning algorithms. Offline, the pursuit-evasion game between the two models can be analyzed using any suitable method [126, 161, 150] to produce a TEB. To provide a TEB for all states of the planning model through solving a single pursuit-evasion game and to reduce the problem dimensionality, TEB computations are performed using the relative dynamics between the two models.

This results in a *tracking error function* that maps the initial relative state between the two models to the TEB: the maximum possible relative distance that could occur over time. The TEB can be thought of as a "safety bubble" around the planning model of the system that the tracking model of the system is guaranteed to stay within.

The resulting TEB from this precomputation may converge to an invariant set (i.e. the planning model cannot move arbitrarily far away from the tracking model when both act optimally), or may result in a time-varying set. Intuitively, a time-varying TEB means that as time progresses the tracking error bound increases by a known amount.

Because the tracking error is bounded in the relative state space, we can precompute and store the *optimal tracking controller* that maps the real-time relative state to the optimal tracking control action for the tracking model to pursue the planning model. The offline computations are *independent* of the path planned in real time.

Online, the autonomous system senses local obstacles, which are then augmented by the TEB to ensure that no potentially unsafe paths can be computed. Next, any chosen path or trajectory planning algorithm uses the simplified planning model and the local environment to determine the next planning state. The autonomous system (represented by the tracking model) then finds the relative state between itself and the next desired state. If this relative state is nearing the TEB then it is plugged into the optimal tracking controller to find the instantaneous optimal tracking control of the tracking model required to stay within the error bound; otherwise, any tracking controller may be used. In this sense, FaSTrack provides a *least-restrictive* control law. This process is repeated for as long as the planning algorithm (rapidly-exploring random trees, model predictive control, etc.) is active.

FaSTrack is modular, and can be used with any method for computing the TEB together with any existing path or trajectory planning algorithms. Any feature of the planning algorithm, such as the ability to account for time-varying obstacles, is inherited when used in the FaSTrack framework. This enables motion planning that is real-time, guaranteed safe, and dynamically accurate. FaSTrack was first introduced in [88], and is generalized here in a number of important ways.

First, we adopt definitions in [161] to refine the notion of relative system to be much more general; in particular, the new definition of relative state allows the pairing of a large class of tracking and planning models.

Next, we introduce and prove the time-varying formulation of FaSTrack, which uses time-varying tracking controllers to provide a time-varying TEB (tvTEB). This has significant practical impact, since the ability to obtain a tvTEB means that the FaSTrack framework does not depend on the convergence of pursuit-evasion games, which is in general not guaranteed and computationally expensive to check.

Furthermore, for both the time-invariant and time-varying cases, we provide mathematical proofs. Lastly, we demonstrate the FaSTrack framework using three different real-time planning algorithms that have been "robustified" by precomputing the TEB and tracking controller to demonstrate our framework.

Precomputation of the TEB and tracking control law for each planning-tracking model pair is done by solving a Hamilton-Jacobi (HJ) partial differential equation (PDE) in this chapter; we encourage readers to refer to [161] for TEB computations using SOS optimization. The planning algorithms used in our numerical examples are the fast sweeping method (FSM) [163], rapidly-exploring random trees (RRT) [110, 98], and model predictive control (MPC) [142, 175].

In the three examples, we also consider different tracking and planning models, one of which utilizes the refined definition of relative system, and another involving a tvTEB. In the simulations, the system travels through a static environment with constraints defined, for example, by obstacles, while experiencing disturbances. The constraints are only fully known through online sensing (e.g. once obstacles are within the limited sensing region of the autonomous system). By combining the TEB with real-time planning algorithms, the system is able to safely plan and track a trajectory through the environment in real time.

6.1 Related Work

Motion planning is an active research area in the controls and robotics communities [90]. In this section we will discuss past work on path, kinematic, and dynamic planning. A major current challenge is to find an intersection of robust and real-time planning for general nonlinear systems. Sample-based planning methods like rapidly-exploring random trees (RRT) [110], probabilistic road maps (PRM) [98], fast marching tree (FMT) [92], fast sweeping method [163] and many others [148, 97, 102] can find collision-free paths through known or partially known environments. While extremely effective in a number of use cases, these algorithms are not designed to be robust to model uncertainty or disturbances, and may not even use a dynamic model of the system in the first place. Motion planning for kinematic systems can also be accomplished through online trajectory optimization using methods such as TrajOpt [156] and CHOMP [145]. These methods can work extremely well in many applications, but are generally challenging to implement in real time for nonlinear dynamic systems.

Model predictive control (MPC) has been a very successful method for dynamic trajectory optimization [142]. However, combining speed, safety, and complex dynamics is a difficult balance to achieve. Using MPC for robotic and aircraft systems typically requires model simplification to take advantage of linear programming or mixed integer linear programming [170, 174, 149]; robustness can also be achieved in linear systems [147, 53]. Nonlinear MPC is most often used on systems that evolve more slowly over time [55, 154], with active work to speed up computation [54, 134]. Adding robustness to nonlinear MPC is being explored through algorithms based on min-max formulations and tube MPCs that bound output trajectories around a nominal path (see [90] for references).

There are other methods of dynamic trajectory planning that manage to cleverly skirt the issue of solving for optimal trajectories online. One such class of methods involve motion primitives [81, 52]. Other methods include making use of safety funnels [122], or generating and choosing random trajectories at waypoints [95, 157]. The latter methods have been implemented successfully in many scenarios, but can be risky in their reliance on finding combinations of pre-computed or randomly-generated safe trajectories.

One notable real-time planning method that also involves robustness guarantees is given by [107], in which a forward reachable set for a high-fidelity model of the system is computed offline and then used to prune motion plans generated online using a low-fidelity model. The approach relies on an *assumed* model mismatch bound; therefore our work has potential to complement works such as [107] by providing the TEB as well as a corresponding feedback tracking controller.

Recent work has considered using offline Hamilton-Jacobi analysis to guarantee tracking error bounds, which can then be used for robust trajectory planning [18]. A class of closely-related techniques define safe tubes around nominal dynamic trajectories by constructing control-Lyapunov functions, which tend to be very difficult to compute [30]. In recent years, methods involving using contraction theory and numerous optimization techniques have enabled computation of conservative approximations of control-Lyapunov functions in the

context of robust trajectory tracking [139, 122, 160, 1].

Finally, some online control techniques can be applied to trajectory tracking with constraint satisfaction. For control-affine systems in which a control barrier function can be identified, it is possible to guarantee forward invariance of the desired set through a state-dependent affine constraint on the control, which can be incorporated into an online optimization problem, and solved in real time [7].

The work presented in this chapter differs from the robust planning methods above because FaSTrack is designed to be modular and easy to use in conjunction with any path or trajectory planner. Additionally, FaSTrack can handle bounded external disturbances (e.g. wind) and work with both known and unknown environments with obstacles.

6.2 Problem Formulation

FaSTrack is a modular framework to plan and track a trajectory (or path converted to a trajectory) online and in real time. Planning is done using a relatively simple model of the system, called the *planning model*. The planning model and algorithm should be chosen to allow real-time planning.

On the other hand, tracking is achieved by a *tracking model* that more accurately represents the autonomous system. Practically, the tracking model should be chosen to consider factors such as higher-order dynamics and disturbances. Under the FaSTrack framework, a tracking error bound (TEB) is computed to account for the mismatch between the planning and tracking models to realize the benefits of both using a simplified model and a higher-fidelity model. Although this chapter focuses on using the Hamilton-Jacobi method to compute time-invariant and time-varying TEBs, in general any method can be used under the FaSTrack framework. For example, the authors in [161] use SOS optimization to achieve better computational scalability.

The environment may contain static obstacles that are *a priori* unknown and can be observed by the system within a limited sensing range. In this section we define the tracking and planning models, as well as the goals of the chapter.

Tracking Model

The tracking model is a relatively accurate and typically higher-dimensional representation of the autonomous system dynamics. Let $s \in \mathcal{S} \subseteq \mathbb{R}^{n_s}$ represent the states of the tracking model. The evolution of the tracking model dynamics satisfies the following ordinary differential equation (ODE):

$$\begin{aligned} \frac{ds}{dt} = \dot{s} = f(s(t), u_s(t), d(t)), t \in [0, T], \\ s(t) \in \mathcal{S}, u_s(t) \in \mathcal{U}_s, d(t) \in \mathcal{D}, \end{aligned} \tag{6.1}$$

We assume that the tracking model dynamics $f : \mathcal{S} \times \mathcal{U}_s \times \mathcal{D} \to \mathcal{S}$ is Lipschitz continuous in the system state s for a fixed control and disturbance functions $u_s(\cdot), d(\cdot)$. At every time t, the control u_s is constrained by the compact set $\mathcal{U}_s \subseteq \mathbb{R}^{n_{u_s}}$, and the disturbance d by the compact set $\mathcal{D} \subseteq \mathbb{R}^{n_d}$. Furthermore, the control function $u_s(\cdot)$ and disturbance function $d(\cdot)$ are measurable functions of time:

$$u_s(\cdot) \in \mathbb{U}_s := \{\phi : [0, T] \to \mathcal{U}_s, \phi(\cdot) \text{ is measurable}\}, \tag{6.2}$$
$$d(\cdot) \in \mathbb{D} := \{\phi : [0, T] \to \mathcal{D}, \phi(\cdot) \text{ is measurable}\}. \tag{6.3}$$

where $\mathbb{U}_s$ and $\mathbb{D}$ represent the set of functions that respectively satisfy control and disturbance constraints at all times. Under these assumptions there exists a unique trajectory solving (6.1) for a given $u_s(\cdot) \in \mathbb{U}_s, d(\cdot) \in \mathbb{D}$ [46]. The trajectories of (6.1) that solve this ODE will be denoted as $\zeta_f(t; s, t_0, u_s(\cdot), d(\cdot))$, where $t_0, t \in [0, T]$ and $t_0 \leq t$. This trajectory notation represents the state of the system at time t, given that the trajectory is initiated at state s and time t_0 and the applied control and disturbance functions are $u_s(\cdot)$ and disturbance $d(\cdot)$ respectively. These trajectories will satisfy the initial condition and the ODE (6.1) almost everywhere:

$$\begin{aligned}
&\frac{d}{dt}\zeta_f(t; s_0, t_0, u_s(\cdot), d(\cdot)) = \\
&\qquad f(\zeta_f(t; s_0, t_0, u_s(\cdot), d(\cdot)), u_s(t), d(\cdot)), \\
&\zeta_f(t_0; s_0, t_0, u_s(\cdot), d(\cdot)) = s_0.
\end{aligned}$$

Let $\mathcal{G} \subset \mathcal{S}$ represent the set of goal states, and $\mathcal{C} \subset \mathcal{S}$ represent state constraints for all time. Often, $\mathcal{C}$ represents the complement of obstacles that the system must avoid.

Example 6.1. *We introduce a running example for illustration throughout the chapter. In this example a car will have to navigate through an environment with a priori unknown obstacles ($\mathcal{C}^\complement$) towards a goal ($\mathcal{G}$). The tracking model of the car is represented by the following five-dimensional dynamics:*

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{v} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} v\cos\theta + d_x \\ v\sin\theta + d_y \\ \omega \\ a + d_a \\ \alpha + d_\alpha \end{bmatrix}, \tag{6.4}$$

where (x, y, θ) represent the pose (position and heading) of the 5D car model, and (v, ω) are the speed and turn rate. The control of the 5D model consists of the linear and angular

acceleration, (a, α), and the disturbances are $(d_x, d_y, d_a, d_\alpha)$. The model parameters are chosen to be $a \in [-0.5, 0.5]$, $|\alpha| \leq 6$, $|d_x|, |d_y|, |d_\alpha| \leq 0.02$, $|d_a| \leq 0.2$.

Planning Model

The planning model is a simpler, lower-dimensional model of the system. Replanning is necessary for navigation in unknown environments, so the planning model is typically constructed by the user so that the desired planning algorithm can operate in real time.

Let p represent the state of the planning model, and let u_p be the control. We assume that the planning state $p \in \mathcal{P}$ are a subset of the tracking state $s \in \mathcal{S}$, so that $\mathcal{P}$ is a subspace within $\mathcal{S}$. This assumption is reasonable since a lower-fidelity model of a system typically involves a subset of the system's states. The dynamics of the planning model satisfy

$$\frac{dp}{dt} = \dot{p} = h(p, u_p), t \in [0, T], \quad p \in \mathcal{P}, u_p \in \mathcal{U}_p, \tag{6.5}$$

with the analogous assumptions on continuity and boundedness as those for (6.1).

Note that the planning model does not include a disturbance input. This is a key feature of FaSTrack: the treatment of disturbances is only necessary in the tracking model, which is modular with respect to any planning method. Therefore we can and will assume that the planning model (and the planning algorithm) do not consider disturbances. This allows the algorithm to operate efficiently without the need to consider robustness. If the planning algorithm does consider disturbances then the added robustness of FaSTrack may result in added conservativeness.

Let $\mathcal{G}_p \subset \mathcal{P}$ and $\mathcal{C}_p \subset \mathcal{P}$ denote the projection of $\mathcal{G}$ and $\mathcal{C}$ respectively onto the subspace $\mathcal{P}$. We will assume that $\mathcal{C}_p$ is *a priori* unknown, and must be sensed as the autonomous system moves around in the environment. Therefore, for convenience, we denote the currently known, or "sensed" constraints as $\mathcal{C}_{p,\text{sense}}(t)$. Note that $\mathcal{C}_{p,\text{sense}}(t)$ depends on time, since the system may gather more information about constraints in the environment over time. In addition, as described throughout the chapter, we will augment $\mathcal{C}_{p,\text{sense}}(t)$ according to the TEB between the tracking and planning models. We denote the augmented obstacles as $\mathcal{C}_{p,\text{aug}}(t)$.

Example 6.2. *For efficient planning use a simpler 3D model with the following dynamics:*

$$\dot{p} = \begin{bmatrix} \dot{\hat{x}} \\ \dot{\hat{y}} \\ \dot{\hat{\theta}} \end{bmatrix} = \begin{bmatrix} \hat{v}\cos\hat{\theta} \\ \hat{v}\sin\hat{\theta} \\ \hat{\omega} \end{bmatrix}, \tag{6.6}$$

where $(\hat{x}, \hat{y}, \hat{\theta})$ represent the pose (position and heading) of the 3D car model. Here the speed $\hat{v}$ is a constant, and the turn rate $\hat{\omega}$ is the control. The planning model must reach

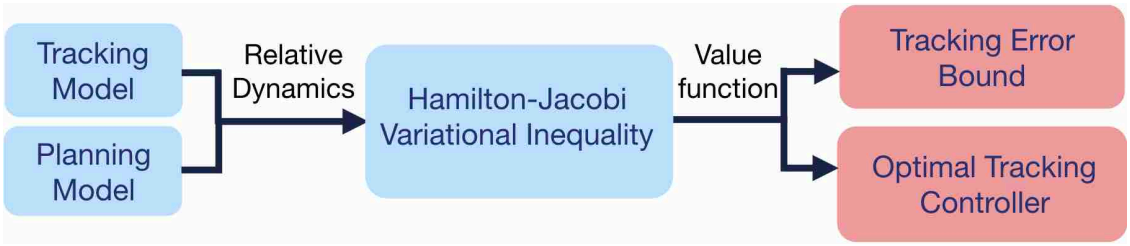


Figure 6.2: Offline framework. Output of the offline framework shown in red.

its goal $\mathcal{G}_p$ while avoiding obstacles represented by $\mathcal{C}_{p,aug}(t)$. The model parameters are chosen to be $\hat{v} = 0.1$, $|\hat{\omega}| \leq 1.5$.

Goals and Approach

Given system dynamics in (6.1), initial state s_0, goal states $\mathcal{G}$, and constraints $\mathcal{C}$ such that $\mathcal{C}_p$ is *a priori* unknown and determined in real time, we would like to steer the system to $\mathcal{G}$ with formally guaranteed satisfaction of $\mathcal{C}$, despite any disturbances the system may experience.

To achieve this goal, FaSTrack decouples the formal safety guarantee from the planning algorithm. Instead of having the system, represented by the tracking model, directly plan trajectories towards $\mathcal{G}$, the system (represented by the tracking model) "chases" the planning model of the system, which may use any planning algorithm to obtain trajectories in real time. The autonomous system is guaranteed to stay within the TEB relative to the planning model. Therefore, we set $\mathcal{G}_{p,\text{contr}}$ to be the projection of $\mathcal{G}$ onto the subspace $\mathcal{P}$ and contracted by one TEB. When the planning algorithm reaches $\mathcal{G}_{p,\text{contr}}$, we know that the autonomous system will be within $\mathcal{G}$. Safety is formally guaranteed through precomputation of the TEB along with a corresponding optimal tracking controller, in combination with augmentation of constraints based on this TEB. An illustration of our framework is shown in Fig. 6.1.

6.3 General Framework

Details of the framework are summarized in Figs. 6.2, 6.3, and 6.4. The purpose of the offline framework (Fig. 6.2) is to generate a TEB and corresponding optimal tracking controller that can be quickly and easily used by the online framework. The planning and tracking model dynamics are used in the reachability precomputation (described in sec. 6.4), whose solution is a value function that acts as the TEB function/look-up table. The gradients of the value function comprise the optimal tracking controller function/look-up table. These functions are independent of the online computations and environment – they depend only on the *relative system state* and dynamics between the planning and tracking models, not on absolute states along the trajectory at execution time.

Online, we start in the bottom-left corner of Fig. 6.3 to determine the tracking model's initial state (i.e. autonomous system's initial state). Based on this we initialize the plan-

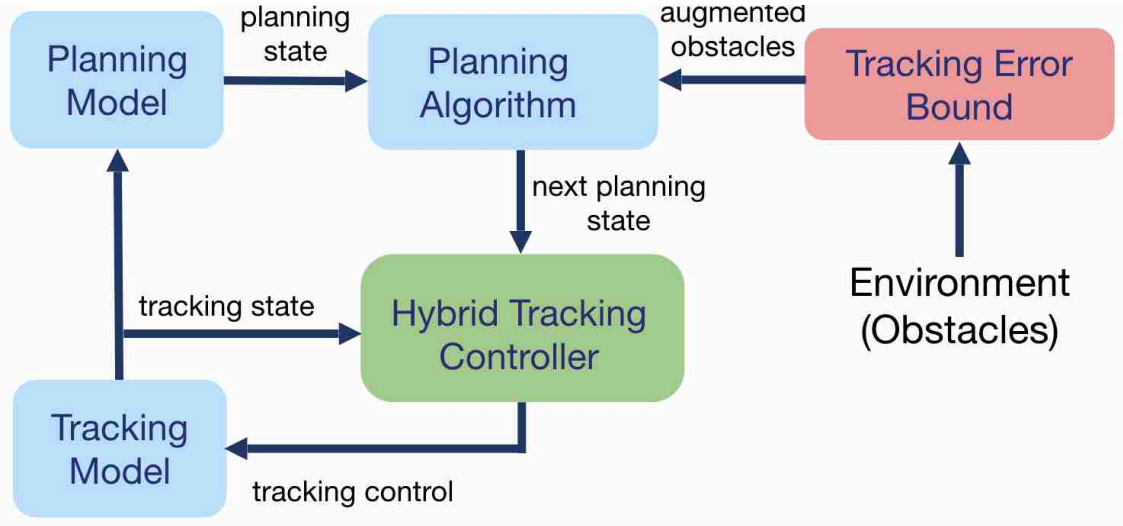


Figure 6.3: Online framework. Components from offline computation shown in red.

ning model such that the tracking model is within the TEB relative to the planning model. The state of the planning model is entered into a planning algorithm. Another input to the planning algorithm is the set of augmented constraints $\mathcal{C}_{p,\text{aug}}$. These are acquired by updating constraints $\mathcal{C}_p$ and accordingly updating the sensed constraints $\mathcal{C}_{p,\text{sense}}$ in the environment. This can be done, for example, by sensing the environment for obstacles. Next, $\mathcal{C}_{p,\text{sense}}$ is augmented by the precomputed TEB using the Minkowski difference to produce the augmented constraints $\mathcal{C}_{p,\text{aug}}$. [1]

In terms of obstacles in the environment, augmenting the constraints by this margin can be thought of as equivalent to wrapping the planning model of the system with a "safety bubble". The planning algorithm takes in the planning model state and augmented constraints, and then outputs a next desired state for the planning model towards $\mathcal{G}_{p,\text{contr}}$. The hybrid tracking controller block takes in this next planning model state along with the current state of the tracking model. Based on the relative state between these two models, the hybrid tracking controller outputs a control signal to the autonomous system. The goal of this control is to make the autonomous system track the desired planning state as closely as possible. This cycle continues as the planning algorithm moves towards $\mathcal{G}_{p,\text{contr}}$.

The hybrid tracking controller is expanded in Fig. 6.4 and consists of two controllers: an *optimal tracking controller* (also referred to as the safety controller) and a *performance controller*. In general, there may be multiple safety and performance controllers depending on various factors such as observed size of disturbances, but for simplicity we will just consider one safety and one performance controller in this chapter. The optimal tracking controller consists of a function (or look-up table) computed offline by solving a HJ variational inequality (VI) [73], and guarantees that the TEB is not violated, despite the worst-case disturbance

[1] For a faster computation we typically simply expand each obstacle by the maximum distance of the TEB in each dimension as a conservative approximation

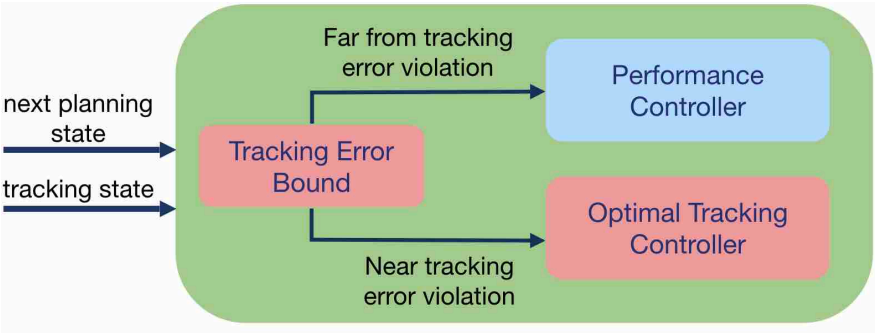


Figure 6.4: Hybrid controller. Components from offline computation are shown in red.

and worst-case planning control. Although the planning model in general does not apply the worst-case planning control, assuming the worst allows us to obtain a *trajectory-independent* TEB and an optimal tracking controller that is guaranteed safe. Note that the computation of the value function and optimal tracking controller is done offline; during online execution, the table look-up operation is computationally inexpensive.

When the system is close to violating the TEB, the optimal tracking controller must be used to prevent the violation. On the other hand, when the system is far from violating the TEB, any controller (such as one that minimizes fuel usage), can be used. This control is used to update the autonomous system's state, and the process repeats. In the following sections we will first explain the precomputation steps taken in the offline framework. We will then walk through the online framework. Finally, we will present three numerical examples.

6.4 Offline Computation

The offline computation begins with setting up a pursuit-evasion game [165, 126] between the tracking model and the planning model of the system. In this game, the tracking model will try to "capture" the planning model, while the planning model is doing everything it can to avoid capture. In reality the planning algorithm is typically not actively trying to avoid the tracking model, but this allows us to account for worst-case scenarios and more crucially, ensure that the TEB is *trajectory-independent*. If both systems are acting optimally in this way, we can determine the maximum possible tracking error between the two models, which is captured by the value function obtained from solving a Hamilton-Jacobi variational inequality, as described below.

Relative System Dynamics

To determine the relative distance over time, we must first define the relative system derived from the tracking (6.1) and planning (6.5) models. The relative system is obtained by fixing

the planning model to the origin and finding the dynamics of the tracking model relative to the planning model. Defining r to be the relative system state, we write

$$r = \Phi(s, p)(s - Qp) \tag{6.7}$$

where Q matches the common states of s and p by augmenting the state space of the planning model. The relative system states r represent the tracking system states relative to the planning states. The function Φ is a linear transform that simplifies the relative system dynamics to be of the form

$$\dot{r} = g(r, u_s, u_p, d), \tag{6.8}$$

which only depends on the relative system state r. A transform Φ that achieves the relative system dynamics in the form of (6.8) is often the identity map or the rotation map when the autonomous system is a mobile robot; therefore, in this chapter, we assume that a suitable Φ is available. For general dynamical systems, it may be difficult to determine Φ; a catalog of tracking and planning models with suitable transforms Φ, as well as a more detailed discussion of Φ, can be found in [161].

In addition, we define the error state e to be the relative system state *excluding* the absolute states of the tracking model, and the auxiliary states η to be the relative system state *excluding* the error state. Hence, $r = \begin{bmatrix} e, \eta \end{bmatrix}^\intercal$.

Example 6.3. *We must determine the relative system state between our 5D tracking and 3D planning models of the car. We define the relative system state to be* $(x_r, y_r, \theta_r, v, \omega)$*, such that the error state* $e = [x_r, y_r, \theta_r]^\intercal$ *is the position and heading of the 5D model in the reference frame of the 3D model, and the auxiliary state* $\eta = [v, \omega]^\intercal$ *represents the speed and turn rate of the 5D model. The relative system state* $r = \begin{bmatrix} e, \eta \end{bmatrix}^\intercal$*, tracking model state* s*, and planning model state* p *are related through* Φ *and* Q *as follows:*

$$\underbrace{\begin{bmatrix} x_r \\ y_r \\ \theta_r \\ v \\ \omega \end{bmatrix}}_{r} = \underbrace{\begin{bmatrix} \begin{bmatrix} \cos\hat{\theta} & \sin\hat{\theta} \\ -\sin\hat{\theta} & \cos\hat{\theta} \end{bmatrix} & \mathbf{0_{2\times 3}} \\ \mathbf{0_{3\times 2}} & \mathbf{I}_3 \end{bmatrix}}_{\Phi} \left(\underbrace{\begin{bmatrix} x \\ y \\ \theta \\ v \\ \omega \end{bmatrix}}_{s} - \underbrace{\begin{bmatrix} \mathbf{I}_3 \\ \mathbf{0_{2\times 3}} \end{bmatrix}}_{Q} \underbrace{\begin{bmatrix} \hat{x} \\ \hat{y} \\ \hat{\theta} \end{bmatrix}}_{p} \right), \tag{6.9}$$

where $\mathbf{0}, \mathbf{I}$ *denote the zero and identity matrices of the indicated sizes. Taking the time*

derivative, we obtain the following relative system dynamics:

$$\dot{r} = \begin{bmatrix} \dot{e} \\ \dot{\eta} \end{bmatrix} = \begin{bmatrix} \dot{x}_r \\ \dot{y}_r \\ \dot{\theta}_r \\ \dot{v} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} -\hat{v} + v\cos\theta_r + \hat{\omega} y_r + d_x \\ v\sin\theta_r - \hat{\omega} x_r + d_y \\ \omega - \hat{\omega} \\ a + d_a \\ \alpha + d_\alpha \end{bmatrix}. \tag{6.10}$$

More examples of relative systems are in Section 6.6.

Formalizing the Pursuit-Evasion Game

Given the relative system dynamics between the tracking and planning models, we would like to compute a guaranteed TEB between these models. This is done by first defining an error function $l(r)$ in the relative state space. One simple error function is the squared distance to the origin, which is shown in Fig. 6.5 (top left, blue hatch surface), and is used when one is concerned only with the tracking error in position.

When we would like to quantify the tracking error for more planning states (for example, error in angular orientation between the two models), the error function can be defined over these states as well. For example, the error function seen in Fig. 6.6 is defined in both position and velocity space; Fig. 6.10 shows yet another error function defined using the one-norm of the displacement between the two models. In our pursuit-evasion game, the tracking model tries to minimize the error, while the planning model and any disturbances experienced by the tracking model try to maximize.

Before constructing the pursuit-evasion game we must first define the method each player must use for making decisions. We define a strategy for planning model as the mapping $\gamma_p : \mathcal{U}_s \to \mathcal{U}_p$ that determines a planning control based on the tracking control. We restrict γ to non-anticipative strategies $\gamma_p \in \Gamma_p(t)$, as defined in [126]. We similarly define the disturbance strategy $\gamma_d : \mathcal{U}_s \to \mathcal{D}$, $\gamma_d \in \Gamma_d(t)$.

We compute the highest cost that this game will ever attain when both players are acting optimally. This is expressed through the following value function:

$$V(r,T) = \sup_{\gamma_p \in \Gamma_p(t), \gamma_d \in \Gamma_d(t)} \inf_{u_s(\cdot) \in \mathbb{U}_s(t)} \Big\{ \max_{t \in [0,T]} l\Big(\zeta_g(t; r, 0, u_s(\cdot), \gamma_p[u_s](\cdot), \gamma_d[u_s](\cdot))\Big)\Big\} \tag{6.11}$$

The value function can be computed via existing methods in HJ reachability analysis [126, 73]. Adapting the formulation in [73] and taking a convention of negative time in the backward reachability literature [38], we compute the value function by solving the HJ variational inequality

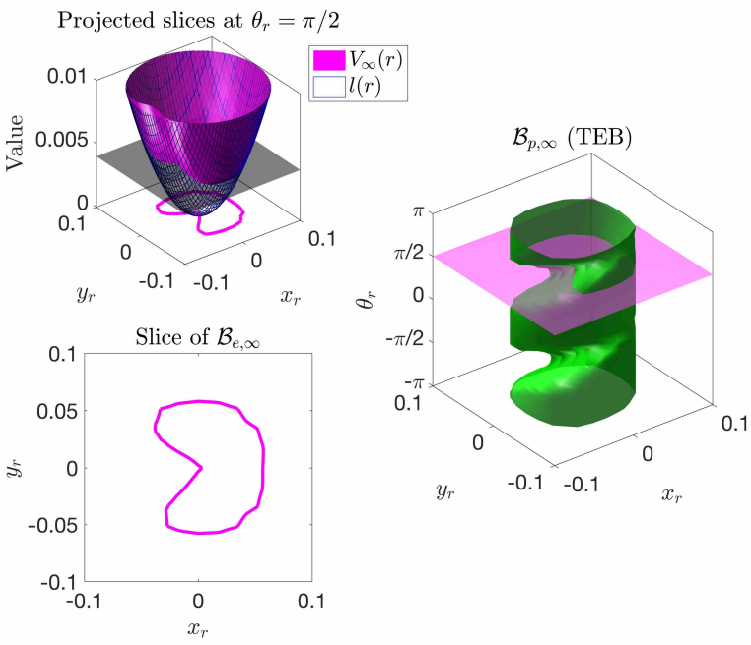


Figure 6.5: Value function and TEB for the running example in (6.9) and (6.10). Top Left: projected slice ($\theta_r = \pi/2$) of the error function (blue hatch) and converged value function (magenta). The minimum value $\underline{V}$ of the converged value function is marked by the black plane; the slice of the value function at $\underline{V}$ determines the TEB (pink set), also shown on the bottom left. Right: the full TEB (no longer projected) in the error states. Note that the slice shown on the bottom left corresponds to the slice marked by the magenta plane at $\theta_r = \pi/2$.

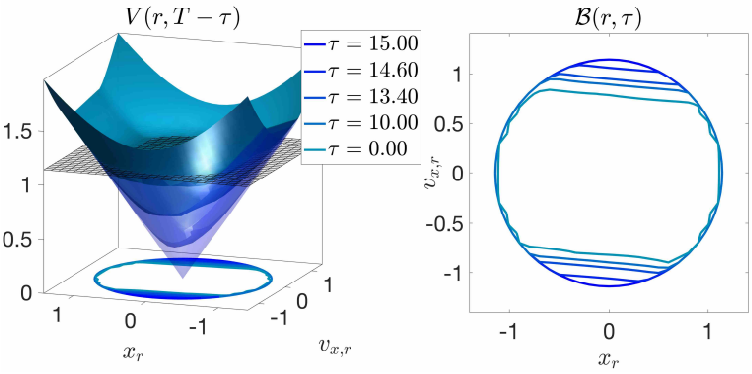


Figure 6.6: Time-varying value function (left) and TEBs (right) for the 8D quadrotor tracking 4D double integrator example in Section 6.6. The value function and the TEB varies with τ, which represents time into the future. The size of the TEB increases with τ because the disturbance and planning control may drive the error states farther and farther from the origin over time. The error states shown are the relative position x_r and velocity $v_{x,r}$. Note that $V(r, 0) = l(r)$.

$$\max\Big\{\frac{\partial \tilde{V}}{\partial t} + \min_{u_s\in\mathcal{S}} \max_{u_p\in\mathcal{P},d\in\mathcal{D}} \nabla\tilde{V}\cdot g(r,u_s,u_p,d), \\ l(r) - \tilde{V}(r,t)\Big\} = 0, \quad t\in[-T,0], \\ \tilde{V}(r,0) = l(r), \tag{6.12}$$

from which we obtain the value function, $V(r,t) = \tilde{V}(r,-t)$. There are many methods for solving this HJ variational inequality, including the level set toolbox [129].

If the planning model is "close" to the tracking model and/or if the control authority of the tracking model is powerful enough to always eventually remain within some distance from the planning model, this value function will converge to an invariant solution for all time, i.e. $V_\infty(r) := \lim_{T\to\infty} V(r,T)$. Because the planning model is user-defined, convergence can often be achieved by tuning the planning model.

However, there may be tracking-planning model pairs for which the value function does not converge. In these cases the value function provides a finite time horizon, time-varying TEB, an example of which is shown in Fig. 6.6. Thus, even when convergence does not occur we can still provide time-varying safety guarantees. In the rest of this chapter, we will focus on the more general time-varying TEB case for clarity, and leave discussion of the time-invariant TEB case in the Appendix. Our numerical examples will demonstrate both cases.

Example 6.4. *We will set the cost to $l(r) = x_r^2 + y_r^2$, squared distance to the origin in position space. This means we would like the system to stay within an (x_r, y_r) bound relative to the planning model, and ignore relative angle. We initialize equation (6.12) with this cost function and the relative system dynamics* (6.10)*. We propagate the HJ variational inequality using the level set method toolbox until convergence or until we reach the planning horizon. In this case the value function converges to V_∞ as seen in Fig. 6.7.*

In Section 6.4, we formally prove that sublevel sets of $V(r,t)$ provide the corresponding time-varying TEBs $\mathcal{B}(t)$ for the finite time horizon case. The analogous result for the time-invariant, infinite time horizon case is proven in the appendix.

The optimal tracking controller is obtained from the value function's spatial gradient [126, 73], $\nabla V(r,t)$, as

$$u_s^*(r,t) = \arg\min_{u_s\in\mathcal{U}_s} \max_{u_p\in\mathcal{U}_p,d\in\mathcal{D}} \nabla V(r,t)\cdot g(r,u_s,u_p,d) \tag{6.13}$$

To ensure the relative system remains within the TEB, we also note that the optimal (worst-case) planning control u_p^* and disturbance d^* can also obtained from $\nabla V(r,t)$ as follows:

$$\begin{bmatrix} u_p^* \\ d^* \end{bmatrix}(r,t) = \arg \max_{u_p \in \mathcal{U}_p, d \in \mathcal{D}} \nabla V(r,t) \cdot g(r, u_s^*, u_p, d) \tag{6.14}$$

For system dynamics affine in the tracking control, planning control, and disturbance, the optimizations in (6.13) and (6.14) are given analytically, and provide the optimal solution to (6.11). In practice, the gradient ∇V is saved as look-up tables over a grid representing the state space of the relative system.

Error Bound Guarantee via Value Function

Prop. 6 states the main theoretical result of this chapter[2]: every level set of $V(r,t)$ is invariant under the following conditions:

1. The tracking model applies the control in (6.13) which tries to track the planning model;
2. The planning model applies the control in (6.14) which tries to escape from the tracking model;
3. The tracking model experiences the worst-case disturbance in (6.14) which tries to prevent successful tracking.

In practice, since the planning control and disturbance are *a priori* unknown and are not directly controlled by the tracking model, conditions 2 and 3 may not hold. In this case, our theoretical results still hold; in fact, the absence of conditions 2 and 3 is advantageous to the tracking model and makes it "easier" to stay within its current level set of $V(r,t)$. The smallest level set corresponding to the value $\underline{V} := \min_r V(r,T)$ can be interpreted as the smallest possible tracking error of the system. The TEB is given by the set[3]

$$\mathcal{B}(\tau) = \{r : V(r, T-\tau) \leq \underline{V}\}. \tag{6.15}$$

Recall that we write the relative system state as $r = (e, \eta)$, where e, η are the error and auxiliary states. Therefore, the TEB in the error state subspace is given by projecting away the auxiliary states η in $\mathcal{B}(\tau)$:

$$\mathcal{B}_e(\tau) = \{e : \exists \eta, V(e, \eta, T-\tau) \leq \underline{V}\} \tag{6.16}$$

[2]The analogous infinite time horizon case is proven in the Appendix

[3]In practice, since V is obtained numerically, we set $\mathcal{B}(\tau) = \{r : V(r, T-\tau) \leq \underline{V} + \epsilon\}$ for some suitably small $\epsilon > 0$.

This is the TEB that will be used in the online framework as shown in Fig. 6.3. Within this bound the tracking model may use any controller, but on the boundary[4] of this bound the tracking model must use the optimal tracking controller. In general, the TEB is defined as a set in the error space, which allows the TEB to not only be in terms of position, but any state of the planning model such as velocity, as demonstrated in the example in Section 6.6.

We now formally state and prove the proposition.

Proposition 6. ***Finite time horizon guaranteed TEB.*** *Given* $t \in [0, T]$,

$$\forall t' \in [t, T],\ r \in \mathcal{B}(t) \Rightarrow \zeta_g^*(t'; r, t) \in \mathcal{B}(t'),\ \textit{where} \tag{6.17a}$$

$$\zeta_g^*(t'; r, t) := \zeta_g(t'; r, t, u_s^*(\cdot), u_p^*(\cdot), d^*(\cdot))), \tag{6.17b}$$

$$u_s^*(\cdot) = \arg \inf_{u_s(\cdot) \in \mathbb{U}_s(t)} \Big\{ \max_{t' \in [t,T]} l(\zeta_g(t'; r, t, u_s(\cdot), u_p^*(\cdot), d^*(\cdot)))\Big\}, \tag{6.17c}$$

$$u_p^*(\cdot) := \gamma_p^*[u_s](\cdot) = \arg \sup_{\gamma_p \in \Gamma_p(t)} \inf_{u_s(\cdot) \in \mathbb{U}_s(t)} \Big\{ \max_{t' \in [t,T]} l(\zeta_g(t'; r, t, u_s(\cdot), \gamma_p[u_s](\cdot), d^*(\cdot)))\Big\} \tag{6.17d}$$

$$d^*(\cdot) = \arg \sup_{\gamma_d \in \Gamma_d(t)} \sup_{\gamma_p \in \Gamma_p(t)} \inf_{u_s(\cdot) \in \mathbb{U}_s(t)} \Big\{ \max_{t' \in [t,T]} l(\zeta_g(t'; r, t, u_s(\cdot), \gamma_p[u_s](\cdot), \gamma_d[u_s](\cdot)))\Big\} \tag{6.17e}$$

Proof: We first show that given $t \in [0, T]$,

$$\forall t' \in [t, T],\ V(r, T - t) \geq V(\zeta_g^*(t'; r, t), T - t') \tag{6.18}$$

[4]Practical issues arising from sampled data control can be handled using methods such as [130, 132, 49] and are not the focus of this chapter.

This follows from the definition of value function.

$$
\begin{aligned}
V(r, T-t) &= \max_{\tau \in [0, T-t]} l(\zeta_g^*(\tau; r, 0)) && \text{(6.19a)} \\
&= \max\{ \max_{\tau \in [0, t'-t]} l(\zeta_g^*(\tau; r, 0)), \\
&\qquad \max_{\tau \in [t'-t, T-t]} l(\zeta_g^*(\tau; r, 0))\} && \text{(6.19b)} \\
&\geq \max_{\tau \in [t'-t, T-t]} l(\zeta_g^*(\tau; r, 0)) && \text{(6.19c)} \\
&= \max_{\tau \in [0, T-t']} l(\zeta_g^*(\tau; r, t-t')) && \text{(6.19d)} \\
&= \max_{\tau \in [0, T-t']} l(\zeta_g^*(\tau; \zeta_g^*(0; r, t-t'), 0)) && \text{(6.19e)} \\
&= \max_{\tau \in [0, T-t']} l(\zeta_g^*(\tau; \zeta_g^*(t'; r, t), 0)) && \text{(6.19f)} \\
&= V(\zeta_g^*(t'; r, t), T-t') && \text{(6.19g)}
\end{aligned}
$$

Explanation of steps:

- (6.19a), (6.19g): by definition of value function, after shifting the time interval in (6.17c) to (6.17e) from $[t, T]$ to $[0, T-t]$.
- (6.19b): rewriting $\max_{\tau \in [0, T-t]}$ by splitting up the time interval $[0, T-t]$ into $[0, t'-t]$ and $[t'-t, T-t]$
- (6.19c): ignoring first argument of the outside max operator
- (6.19d): shifting time reference by $t - t'$, since dynamics are time-invariant
- (6.19e): splitting trajectory $\zeta_g^*(\tau; r, t-t')$ into two stages corresponding to time intervals $[t-t', 0]$ and $[0, \tau]$
- (6.19f): shifting time reference in $\zeta_g^*(0; r, t-t')$ by t', since dynamics are time-invariant

Now, we finish the proof as follows:

$$
\begin{aligned}
r \in \mathcal{B}(t) &\Leftrightarrow V(r, T-t) \leq \underline{V} && \text{(6.20a)} \\
&\Rightarrow V(\zeta_g^*(t'; r, t), T-t') \leq \underline{V} && \text{(6.20b)} \\
&\Leftrightarrow \zeta_g^*(t'; r, t) \in \mathcal{B}(t'), && \text{(6.20c)}
\end{aligned}
$$

where (6.18) is used for the step in (6.20b). ■

Discussion

Worst-case assumptions

Prop. 6 assumes that the planning control u_p and disturbance d are optimally maximizing the value function V, and thereby increasing the size of the TEB $\mathcal{B}$. Despite this, (6.17a) still holds. In reality, u_p and d do not behave in a worst-case fashion, and it is often the case that when $t' \geq t$, we have $r \in \mathcal{B}(t) \Rightarrow \zeta_g(t'; r, t) \in \mathcal{B}(\tau)$ for some $\tau \leq t'$. Thus, one can "take advantage" of the suboptimality of u_p and d by finding the earliest τ such that $\zeta_g(t'; r, t) \in \mathcal{B}(\tau)$ in order to have the tighter TEB over a longer time-horizon.

Relationship to reachable sets

Prop. 6 is similar to well-known results in differential game theory with a different cost function [4], and has been utilized in the context of interpreting the subzero level set of V as a backward reachable set for tasks such as collision avoidance [126]. In this work we do not assign special meaning to any particular level set, and instead consider all level sets at the same time. This allows us to effectively solve many simultaneous reachability problems in a single computation, removing the need to check whether resulting invariant sets are empty, as was done in [18].

Relationship to control-Lyapunov functions

One interpretation of (6.16) is that $V(r, T-t)$ is a control-Lyapunov function for the relative dynamics between the tracking model and the planning model, and any level set of $V(r, T-t)$ is invariant. It should be noted that computing control-Lyapunov functions for general nonlinear systems is difficult. In addition, the relative system trajectories are guaranteed to remain within the initial level set despite the worst-case disturbance. In the absence of any information about, for example, the intent of the planning system, such a worst-case assumption is needed.

Example 6.5. *In this example we have computed a converged value function V_∞. The corresponding TEB can be found using* (6.29) *in the Appendix. We can similarly find the TEB projected onto the planning states using* (6.30)*. The minimum of the value function was approximately $\underline{V} = 0.004$, and the size of the TEB in (x_r, y_r) space is approximately* 0.065*. The converged value function and TEB can be seen in Fig. 6.7. The corresponding optimal tracking controller is obtained by plugging the gradients of our converged value function and our relative system dynamics into (6.27).*

6.5 Online Computation

Algorithm 1 describes the online computation. Lines 1 to 3 indicate that the value function $V(r, t'')$, the gradient ∇V from which the optimal tracking controller is obtained, as well as the TEB sets $\mathcal{B}, \mathcal{B}_e$ are given from offline precomputation. Lines 4-6 initialize the computation by setting the planning and tracking model states such that the relative system state is inside the TEB $\mathcal{B}$.

Algorithm 1: Online Trajectory Planning

1: **Given**:
2: $V(r, t''), t'' \in [0, T]$ and gradient $\nabla V(r, t'')$
3: $\mathcal{B}(t'), t' \in [0, T]$ from (6.15), and $\mathcal{B}_e$ from (6.16)
4: **Initialization**:
5: Choose p, s such that $r \in \mathcal{B}(0)$
6: Set initial time: $t \leftarrow 0$.
7: **while** Planning goal is not reached OR planning horizon is exceeded **do**
8: **TEB Block**:
9: Look for the smallest τ such that $r \in \mathcal{B}(\tau)$
10: $\mathcal{C}_{p,\text{aug}}(t + t') \leftarrow \mathcal{C}_{p,\text{sense}} \ominus \mathcal{B}_e(\tau + t')$
11: **Path Planning Block**:
12: $p_{\text{next}} \leftarrow \text{nextState}(p, \mathcal{C}_{p,\text{aug}})$
13: **Hybrid Tracking Controller Block**:
14: $r_{\text{next}} \leftarrow \Phi(s, p)(s - Qp_{\text{next}})$
15: **if** r_{next} is on boundary $\mathcal{B}_e(t)$ **then**
16: use optimal tracking controller: $u_s \leftarrow u_s^*$ in (6.13)
17: **else**
18: use performance controller:
19: $u_s \leftarrow$ desired controller
20: **end if**
21: **Tracking Model Block**:
22: apply control u_s to vehicle for a time step of Δt
23: **Planning Model Block**:
24: update planning state, $p \leftarrow p_{\text{next}}$, from Line 12
25: check if p is at planning goal
26: **Update time**:
27: $t \leftarrow t + \Delta t$
28: **end while**

The TEB block is shown on lines 8-10. The sensor detects obstacles, or in general constraints, $\mathcal{C}_{p,\text{sense}}(\cdot)$ within the sensing region around the vehicle. If the user allows the planning model to instantaneously stop, then for the static environments explored in this chapter the sensing region must be large enough to sense any obstacles within one TEB of

the planning algorithm. Thus, the set representing the minimum allowable sensing region[5] is $m = \{\mathcal{B}_e(T)\} \bigoplus FRS(\delta t)$, where $FRS(\delta t)$ is the forward reachable set of the planning model for one time step of planning (i.e. the largest step in space that the planning algorithm can make in one time step). When using sensors that perceive some fixed radius in position in all directions, the required radius is simply the maximum distance of the TEB summed with the maximum distance the planning algorithm can cover in one time step. Note that for time-varying TEBs with long time horizons this sensing requirement can be fairly restrictive depending on the maximum size of the time-varying TEB. If the planning model is not allowed to stop instantaneously, recursive safety can be ensured by methods such as [13, 76]. FaSTrack applied to dynamic environments with humans is explored in [72, 76], and is paired with work on sequential trajectory tracking [40] to handle multi-human, multi-robot environments [13].

Constraints are defined in the state space of the planning model, and therefore can represent constraints not only in position but also in, for example, velocity or angular space. One can either augment the constraints by the TEB, or augment the planning algorithm by the TEB. Augmenting either planning algorithms or constraints by some buffer is common practice in motion planning. The decision on which to augment falls to the user based on the planning method used. Augmenting the planning algorithm requires computing the intersection of sets between the TEB and the constraints (as done in [72, 13]). Augmenting the constraints instead requires using the Minkowski difference, denoted "$\ominus$." If the TEB is a complicated shape for which computing the Minkowski difference is difficult, one can reduce computational speed by simply expanding the constraints by the maximum distance of the TEB in each dimension (this will result in a more conservative approximation of the unsafe space).

The path planning block (lines 11-12) takes in the planning model state p and the augmented constraints $\mathcal{C}_{p,\text{aug}}$, and outputs the next state of the planning model p_{next} through the function nextState$(\cdot,\cdot)$. As mentioned, FaSTrack is agnostic to the planning algorithm used, so we assume that nextState$(\cdot,\cdot)$ has been provided. The hybrid tracking controller block (lines 13-20) first computes the updated relative system state r_{next}. If the r_{next} is on the boundary of the TEB $\mathcal{B}_e(0)$, the optimal tracking controller given in (6.27) must be used to remain within the TEB. If the relative system state is not on the tracking boundary, a performance controller may be used. For the example in Section 6.6 the safety and performance controllers are identical, but in general this performance controller can suit the needs of the individual applications.

The control u_s^* is then applied to the physical system in the tracking block (lines 21-22) for a time period of Δt. The next state is denoted s_{next}. Finally, the planning model state is updated to p_{next} in the planning model block (lines 23-25). We repeat this process until the planning goal has been reached.

[5] $T \to \infty$ for the infinite time horizon case.

6.6 Numerical examples

In this section, we demonstrate the FaSTrack framework in three numerical simulation examples that respectively represent dynamic programming- based, sampling- based, and optimization- based planning algorithms: (1) a 5D car tracking a 3D car model with the FSM planning algorithm, (2) a 10D quadrotor tracking a single integrator model with the RRT planning algorithm, and (3) an 8D quadrotor tracking a double integrator model with the MPC planning algorithm. In each example, obstacles in the environment are *a priori* unknown, and are revealed to the vehicle when they are "sensed," i.e. come within the minimum allowable sensing distance. Whenever the obstacle map is updated, the planning algorithm replans a trajectory in real time. In this chapter, the details of sensing are kept as simple as possible; we aim to only demonstrate our framework for real-time guaranteed safe planning and replanning. In general, any other planning algorithm can be used for planning in unknown environments, as long as planning and replanning can be done in real time.

For each example, we first describe the tracking and planning models. Next, we present the relative dynamics as well as the precomputation results. Afterwards, we briefly describe the planning algorithm and how obstacles are sensed by the vehicle. Finally, we show trajectory simulation results.

Running Example: 5D car-3D car example with FSM

For our first example, we continue our running example of the 5D tracking model and 3D planning model of an autonomous car. We will demonstrate the combination of fast planning and provably robust tracking by combining the fast sweeping method (FSM) [163] with our computed TEB. FSM is an efficient optimal control-based planning algorithm for car-like systems, and provides numerically convergent globally optimal trajectory in real time. In this example, we use FSM to perform real-time planning for the 3D kinematic car model, whose trajectory is tracked by the 5D car model.

Offline computation

As stated in Sec. 6.4, we computed the converged value function $V_\infty(r)$, which is shown in Fig. 6.7, with $\underline{V} = 0.004$, and $\mathcal{B}_{p,\infty} = 0.065$. The top left plot of Fig. 6.7 shows the TEB $\mathcal{B}_{p,\infty}$ in green. The tracking model must apply the optimal control when it is on the green boundary. The cross-sectional area of the TEB is the largest at $\theta_r = 0, \pi$, because at these θ_r values the 5D car model is either aligned with or opposite to the 3D car model. Since the 5D car is able to move both forward and backward, these two alignments make tracking the easiest. For the same reasoning, the cross sectional area is the smallest at $\theta_r = -\pi/2, \pi/2$, etc.

The magenta and cyan planes indicate slices of the TEB at $\theta_r = \pi/2, -3\pi/4$, respectively. With these θ_r values fixed, corresponding projections of the value function onto (x_r, y_r) space are shown in the top right and bottom left plots. Here, $\underline{V}$ is shown as the gray plane, with

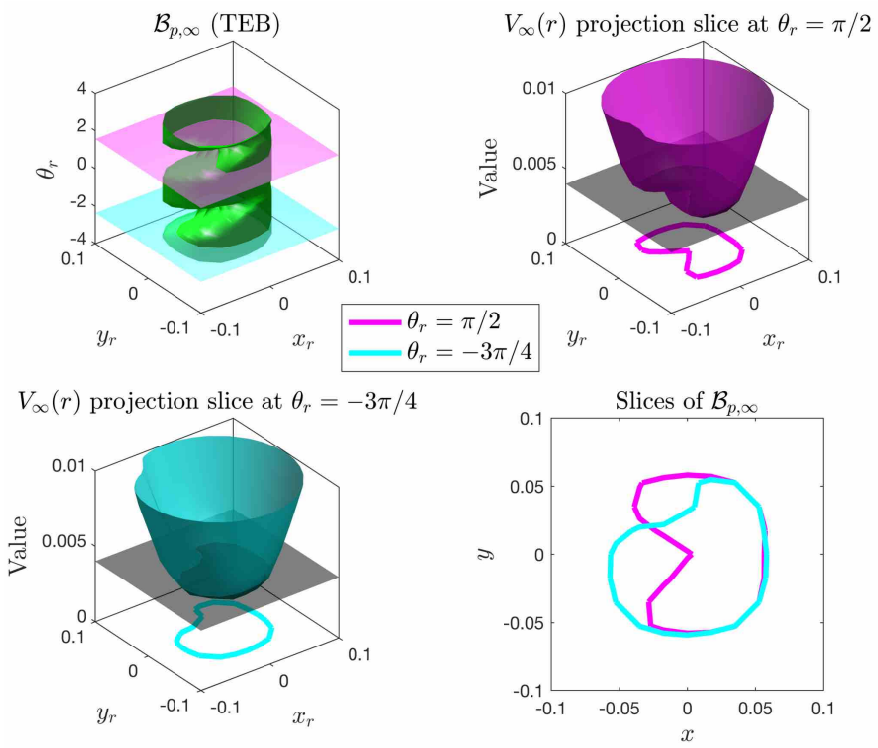


Figure 6.7: Infinite time horizon TEB (top left), two slices of the value function at $\theta_r = \pi/2, -3\pi/4$ (top right, bottom left), and corresponding TEB slices (bottom right) for the running example (5D car tracking 3D car) introduced in Section 6.6.

the intersection of the gray plane and the value function projection shown by the curve in the 0-level plane. These curves are slices of $\mathcal{B}_{p,\infty}$ at the $\theta_r = \pi/2, -3\pi/4$ levels.

Computation was done on a desktop computer with an Intel Core i7 5820K processor, on a $31 \times 31 \times 45 \times 27 \times 47$ grid, and took approximately 23 hours and required approximately 2 GB of RAM using a C++ implementation of level set methods for solving (6.12). A 5D computation is at the limit of computational tractability using the HJ method. Fortunately, FaSTrack is modular with respect to the method for computating the TEB, and we are exploring techniques for computing TEBs for higher-dimensional systems through sum-of-squares optimization [161] and approximate dynamic programming [150].

Online sensing and planning

The simulation showing the combination of tracking and planning is shown in Fig. 6.8. The goal of the system, the 5D car, is to reach the blue circle at $(0.5, 0.5)$ with a heading that is within $\pi/6$ of the direction indicated by the arrow inside the blue circle, $\pi/2$. Three initially unknown obstacles, whose boundaries are shown in dotted black, make up the constraints $\mathcal{C}_p$.

While planning a trajectory to the goal, the car also senses obstacles. For this example, we chose a simple virtual sensor that reveals obstacles within a range of 0.5 and in front of the vehicle within an angle of $\pi/6$, depicted as the light green fan. When a portion of the unknown obstacles is within this region, that portion is made known to the vehicle, and is shown in red. These make up the sensed constraints $\mathcal{C}_{p,\text{sense}}$. To ensure that the 5D car does not collide with the obstacles despite error in tracking, planning is done with respect to augmented constraints $\mathcal{C}_{p,\text{aug}}$, shown in dashed blue.

Given the current planning constraints $\mathcal{C}_{p,\text{aug}}$, the planning algorithm uses the 3D planning model to generate a trajectory, in real time using FSM, towards the goal. This plan is shown in dotted red. The 5D system robustly tracks the 3D system within the TEB in Fig. 6.7. Four time snapshots of the simulation are shown in Fig. 6.8. In the top left subplot, the system has sensed only a very small portion of the obstacles, and hence plans a trajectory through an unknown obstacle to the target. However, while tracking this initial trajectory, more of the L-shaped obstacle is made known to the system, and therefore the system plans around this obstacle, as shown in the top right subplot. The bottom subplots show the system navigating through sensed obstacles and reaching the goal at $t = 23.9$ s.

As explained in Fig. 6.4, when the tracking error is relatively large, the autonomous system uses the optimal tracking controller given by (6.27); otherwise, it uses a performance controller. In this simulation, we used a simple LQR controller on the linearized system when the tracking error is less than a quarter of the size of the TEB. In general, this switching condition is user-defined. The tracking error over time is shown in Fig. 6.9. The red dots indicate the time points at which the optimal tracking controller in (6.27) is used, and the blue dots indicate the time points at which the LQR controller is used. One can see that when the optimal tracking controller is used, the error stays below 0.05, well below the predicted TEB of 0.065, since the planning control and the disturbances are not being adversarial. The disturbance was chosen to be uniformly random within the chosen bounds.

The simulation was done in MATLAB on a desktop computer with an Intel Core i7 2600K CPU. Time was discretized in increments of 0.067 seconds, (15 Hz). Averaged over the duration of the simulation, planning with FSM took approximately 66 ms per iteration, and obtaining the tracking control from (6.27) took approximately 2 ms per iteration.

10D quadrotor-3D single integrator example with RRT

Our second example involves a 10D near-hover quadrotor [29] as the tracking model and a single integrator in 3D space as the planning model. Planning is done using RRT, a well-known sampling-based planning algorithm that quickly produces geometric paths from a starting position to a goal position [110, 98]. Paths given by the RRT planning algorithm are converted to time-stamped trajectories by placing a maximum velocity in each dimension along the generated geometric paths.

The dynamics of tracking model and of the 3D single integrator is as follows:

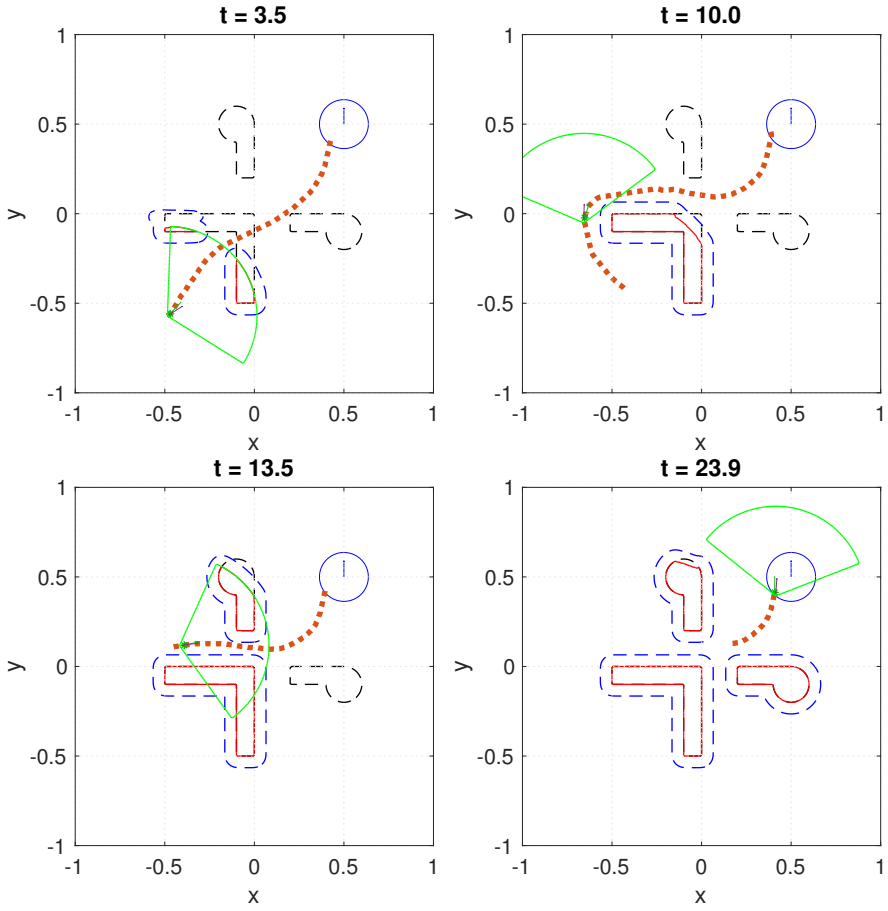


Figure 6.8: Simulation of the 5D-3D example. As the vehicle with 5D car dynamics senses new obstacles in the sensing region (light green), the 3D model replans trajectories, which are robustly tracked by the 5D system. Augmentation of the constraints resulting from the obstacles ensures safety of the 5D system using the optimal tracking controller.

$$\begin{bmatrix} \dot{x} \\ \dot{v}_x \\ \dot{\theta}_x \\ \dot{\omega}_x \\ \dot{y} \\ \dot{v}_y \\ \dot{\theta}_y \\ \dot{\omega}_y \\ \dot{z} \\ \dot{v}_z \end{bmatrix} = \begin{bmatrix} v_x + d_x \\ g \tan \theta_x \\ -d_1 \theta_x + \omega_x \\ -d_0 \theta_x + n_0 a_x \\ v_y + d_y \\ g \tan \theta_y \\ -d_1 \theta_y + \omega_y \\ -d_0 \theta_y + n_0 a_y \\ v_z + d_z \\ k_T a_z - g \end{bmatrix}, \quad \begin{bmatrix} \dot{\hat{x}} \\ \dot{\hat{y}} \\ \dot{\hat{z}} \end{bmatrix} = \begin{bmatrix} \hat{v}_x \\ \hat{v}_y \\ \hat{v}_z \end{bmatrix}, \tag{6.21}$$

where quadrotor states (x, y, z) denote the position, (v_x, v_y, v_z) denote the velocity, (θ_x, θ_y)

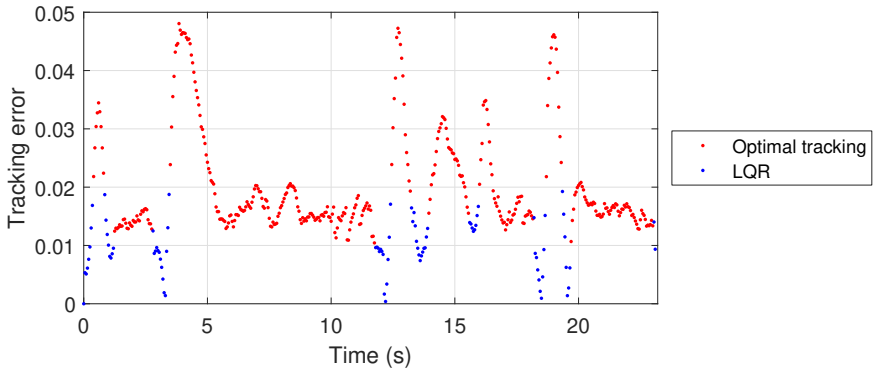


Figure 6.9: Tracking error bound over time for the 5D-3D example. The red dots indicate that the optimal tracking controller is used, while the blue dots indicate that an LQR controller for the linearized system is used. The hybrid controller switches from LQR to the optimal tracking controller whenever the error exceeds 0.02. The tracking error is always well below the predicted TEB of 0.065.

denote the pitch and roll, and (ω_x, ω_y) denote the pitch and roll rates. The controls of the 10D system are (u_x, u_y, u_z), where u_x and u_y represent the desired pitch and roll angle, and u_z represents the vertical thrust.

The 3D system controls are $(\hat{v}_x, \hat{v}_y, \hat{v}_z)$, and represent the velocity in each positional dimension. The disturbances in the 10D system (d_x, d_y, d_z) are caused by wind, which acts on the velocity in each dimension. The model parameters are chosen to be $d_0 = 10$, $d_1 = 8$, $n_0 = 10$, $k_T = 0.91$, $g = 9.81$, $|u_x|, |u_y| \leq \pi/9$, $u_z \in [0, 1.5g]$, $|\hat{v}_x|, |\hat{v}_y|, |\hat{v}_z| \leq 0.5$. The disturbance bounds were chosen to be $|d_x|, |d_y|, |d_z| \leq 0.1$.

Offline computation

We define the relative system states to consist of the error states, or relative position (x_r, y_r, z_r), concatenated with the rest of the state variables of the 10D quadrotor model. Defining $\Phi = \mathbf{I}_{10}$ and

$$Q = \begin{bmatrix} \begin{bmatrix} 1 \\ \mathbf{0}_{3\times 1} \end{bmatrix} & \mathbf{0}_{4\times 1} & \mathbf{0}_{4\times 1} \\ \mathbf{0}_{4\times 1} & \begin{bmatrix} 1 \\ \mathbf{0}_{3\times 1} \end{bmatrix} & \mathbf{0}_{4\times 1} \\ \mathbf{0}_{2\times 1} & \mathbf{0}_{2\times 1} & \begin{bmatrix} 1 \\ 0 \end{bmatrix} \end{bmatrix},$$

we obtain the following relative system dynamics:

$$\begin{bmatrix} \dot{x}_r \\ \dot{v}_x \\ \dot{\theta}_x \\ \dot{\omega}_x \\ \dot{y}_r \\ \dot{v}_y \\ \dot{\theta}_y \\ \dot{\omega}_y \\ \dot{z}_r \\ \dot{v}_z \end{bmatrix} = \begin{bmatrix} v_x - \hat{v}_x + d_x \\ g \tan \theta_x \\ -d_1 \theta_x + \omega_x \\ -d_0 \theta_x + n_0 u_x \\ v_y - \hat{v}_y + d_y \\ g \tan \theta_y \\ -d_1 \theta_y + \omega_y \\ -d_0 \theta_y + n_0 u_y \\ v_z - \hat{v}_z + d_z \\ k_T u_z - g \end{bmatrix}. \tag{6.22}$$

The relative system dynamics given in (6.22) is decomposable into three independent subsystems involving the sets of variables $(x_r, v_x, \theta_x, \omega_x)$, $(x_y, v_y, \theta_y, \omega_y)$, (z_r, v_z), allowing us to choose the error function to be also in the decomposable form of $l(r) = \max(x_r^2, y_r^2, z_r^2)$, so that we can solve (6.12) tractably since each subsystem is at most 4D [38].

The left subplot of Fig. 6.10 shows the projection of the value function V onto the (x_r, v_x) space resulting from solving (6.12) over an increasingly long time horizon. Starting from $\tau = 0$, we have that $l(r) = V(r, 0)$. As τ increases, the value function evolves according to (6.12), and eventually converges when τ reaches 3.5. This implies that $V_\infty(r) = V(r, \tau = 3.5)$, since we would still obtain the same function even if we let τ approach infinity. The horizontal plane shows $\underline{V} = 0.3$, which corresponds to a TEB of approximately 0.9.

The right subplot of Fig. 6.10 shows the $\underline{V} = 0.3$ level set of value function projection, which is the projection onto the (x_r, v_x) space, of the TEB $\mathcal{B}_{e,\infty}$. The range of x_r provides the TEB used for the planning algorithm, $\mathcal{B}_{p,\infty}$. The value function and TEB in the $(y_r, v_y, \theta_y, \omega_y)$ and (z_r, v_z) spaces are combined to form the 10D TEB, which is projected down to the 3D positional space. For conciseness, these value functions are not shown; however, one can see the resulting TEB in Fig. 6.11 and 6.12 as the translucent blue box.

Offline computations were done on a laptop with an Intel Core i7 4702HQ CPU using a MATLAB implementation of level set methods [129] used for solving (6.12). The 4D computations were done on a $61 \times 61 \times 41 \times 41$ grid, took approximately 12 hours, and required approximately 300 MB of RAM. The 2D computation in the (z_r, v_z) space was done on a 101×101 grid, took approximately 15 seconds, and required negligible RAM.

Online sensing and planning

The simulation involving the 10D quadrotor model tracking the 3D single integrator is shown in Fig. 6.11 and 6.12. Here, the system aims to start at $(x, y, z) = (-12, 0, 0)$ and reach $(12, 0, 0)$. To best test our method, we allow the wind disturbance to act adversarially, resulting in worst-case wind conditions. Three rectangular obstacles, which make up the constraints $\mathcal{C}_p$, are present and initially unknown. Before the obstacles are sensed by the system, they are shown in gray. As the 10D quadrotor senses obstacles (when they are within 1.5 units from the quadrotor), the component of the obstacle that is within sensing

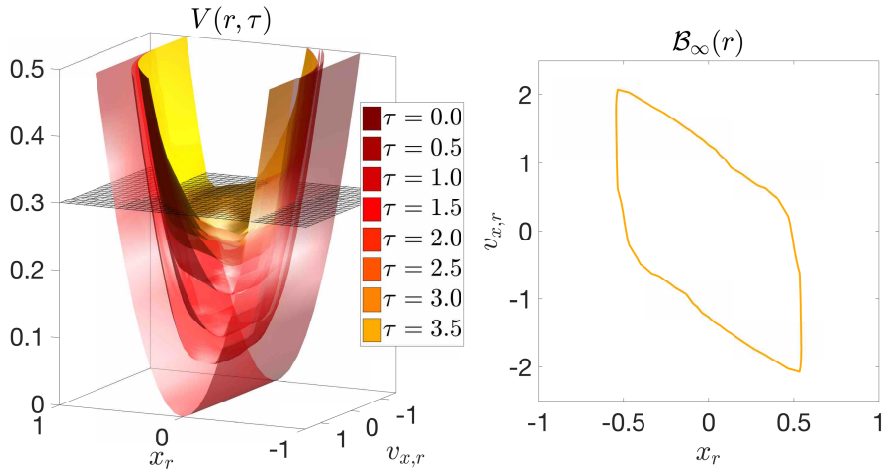


Figure 6.10: Left: snapshots in time of the value function $V(r, \tau)$ shown over dimensions x_r and $v_{x,r}$. Snapshots are from $\tau = 0$ s (transparent dark red surface on bottom) to convergence at $\tau = 3.5$ s (solid yellow surface on top). Right: 2D slice at $V_\infty(r) = 0.3$ (corresponding to gray slice on the left). This is the infinite horizon TEB, $\mathcal{B}_\infty(r)$ in the x_r and $v_{x,r}$ dimensions.

distance is revealed and shown in red. To demonstrate the flexibility of augmenting by the TEB, in this example we show the TEB augmenting the planning algorithm rather than the obstacles. Whenever new obstacles are revealed, the planning algorithm replans a trajectory to the goal while avoiding the augmented constraint set $\mathcal{C}_{p,\text{aug}}$. This happens in real time using RRT.

Fig. 6.11 shows the entire trajectory, with the end of the trajectory being close to the goal position. The planning model state is shown as a small green star, and the translucent box around it depicts the TEB: the tracking model position is guaranteed to reside within this box. Therefore, as long as the planning model plans in a way such that the TEB does not intersect with the obstacles, the tracking model is guaranteed to be safe. Due to the random nature of RRT, during the simulation the system appears to randomly explore to look for an unobstructed path to the obstacle; we did not implement any exploration algorithms.

Fig. 6.12 shows three different time snapshots of the simulation. At $t = 8$, the planning model has sensed a portion of the previously unknown obstacles, and replans, so that the path deviates from a straight line from the initial position to the goal position. The subplot showing $t = 47.7$ is rotated to show the trajectory up to this time from a more informative view angle. Here, the system has safely passed by the first planar obstacle, and is moving around the second. Note that the TEB never intersects the obstacles, implying that the tracking model is guaranteed to avoid collision with the obstacles, since it is guaranteed to stay within the TEB. At $t = 83.5$, the autonomous system safely passes by the last obstacle.

Fig. 6.13 shows the maximum tracking error, in the three positional dimensions over time. The red points indicate the time points at which the optimal tracking controller from Eq. (6.27) was used; this is the optimal tracking controller depicted in Fig. 6.4. The blue

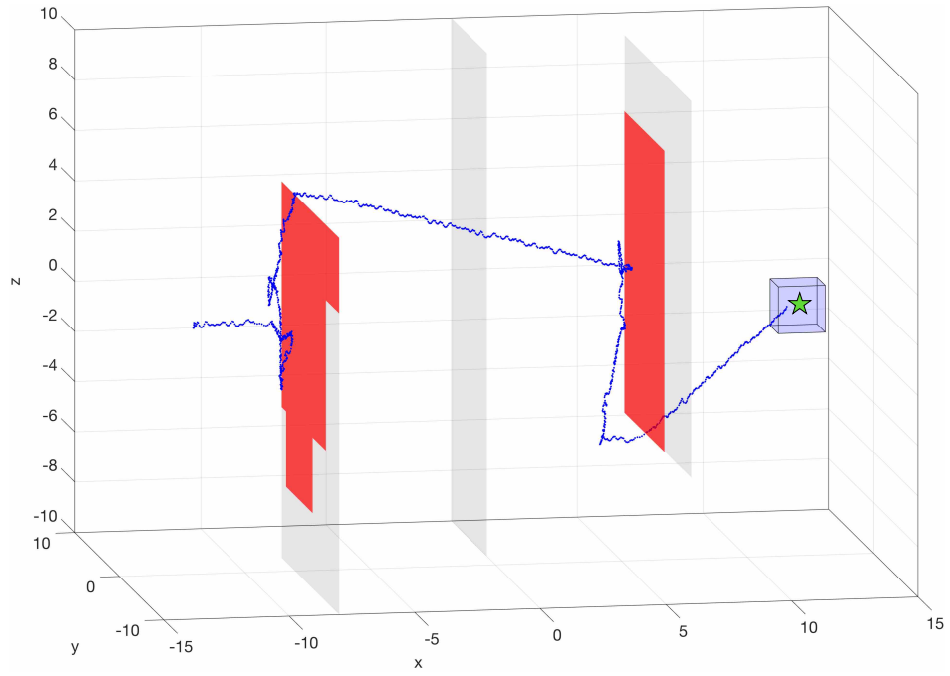


Figure 6.11: Simulation of the 10D quadrotor tracking (trajectory shown in blue) a 3D single integrator (position shown as green star inside blue box). The dimensions x, y, z represent the length, width, and height of the absolute state space. The system senses initially unknown obstacles (gray), which are revealed (revealed parts shown in red) as the system approaches them. Replanning is done in real time by RRT when new obstacles are sensed. The TEB is shown as the blue box, and is the set of positions that the 10D quadrotor is guaranteed to remain within.

points indicate the time points at which a performance controller, also depicted in Fig. 6.4, was used. For the performance controller, we used a simple proportional controller that depends on the tracking error in each positional dimension; this controller is used whenever the tracking error is less than a quarter of the TEB. From Fig. 6.13, one can observe that the tracking error is always less than the TEB implied by the value function.

The simulation was done in MATLAB on a desktop computer with an Intel Core i7 2600K CPU. The time was discretized in increments of 0.01. On average per iteration, planning with RRT using a simple multi-tree RRT planning algorithm implemented in MATLAB modified from [79] took 5 ms, and computing the tracking controller took 5.5 ms.

8D quadrotor-4D double integrator example with MPC

In this section, we demonstrate the online computation framework in Alg. 1 with an 8D quadrotor example and MPC as the online planning algorithm. Unlike in Sections 6.6 and 6.6,

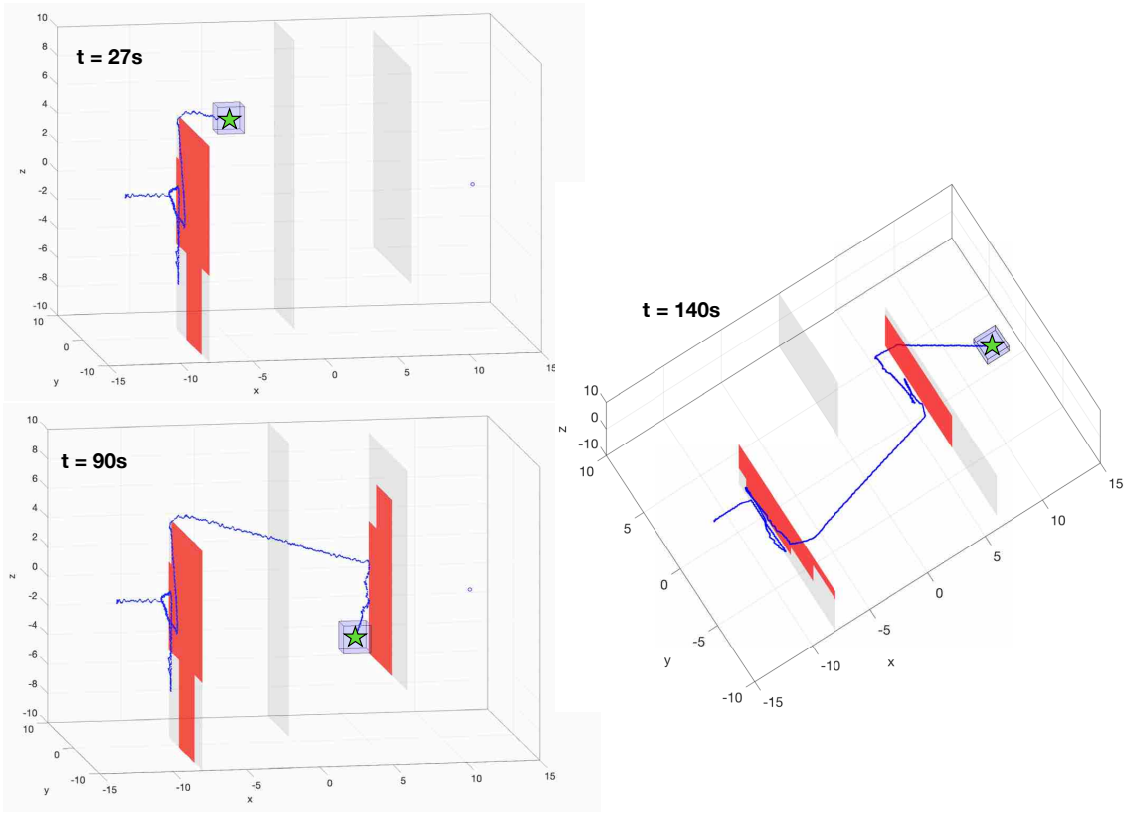


Figure 6.12: Three time snapshots of the simulation in Fig. 6.11. The full simulation can be seen at https://youtu.be/JwDrYG-oZSY.

we consider a planning-tracking model pair for which the value function does not converge, so that the computation instead provides a tvTEB. In addition, the TEB depends on both position and speed, as opposed to just position. This is to accommodate velocity bounds on the system.

First we define the 8D dynamics of the near-hover quadrotor, and the 4D dynamics of a double integrator, which serves as the planning model to be used in MPC:

$$\begin{bmatrix} \dot{x} \\ \dot{v}_x \\ \dot{\theta}_x \\ \dot{\omega}_x \\ \dot{y} \\ \dot{v}_y \\ \dot{\theta}_y \\ \dot{\omega}_y \end{bmatrix} = \begin{bmatrix} v_{x,s} + d_x \\ g \tan\theta_x \\ -d_1\theta_x + \omega_x \\ -d_0\theta_x + n_0 a_x \\ v_y + d_y \\ g \tan\theta_y \\ -d_1\theta_y + \omega_y \\ -d_0\theta_y + n_0 a_y \end{bmatrix}, \quad \begin{bmatrix} \dot{\hat{x}} \\ \dot{\hat{v}}_x \\ \dot{\hat{y}} \\ \dot{\hat{v}}_y \end{bmatrix} = \begin{bmatrix} \hat{v}_x \\ \hat{a}_x \\ \hat{v}_y \\ \hat{a}_y \end{bmatrix}, \tag{6.23}$$

where the states, controls, and disturbances are the same as the first 8 components of the

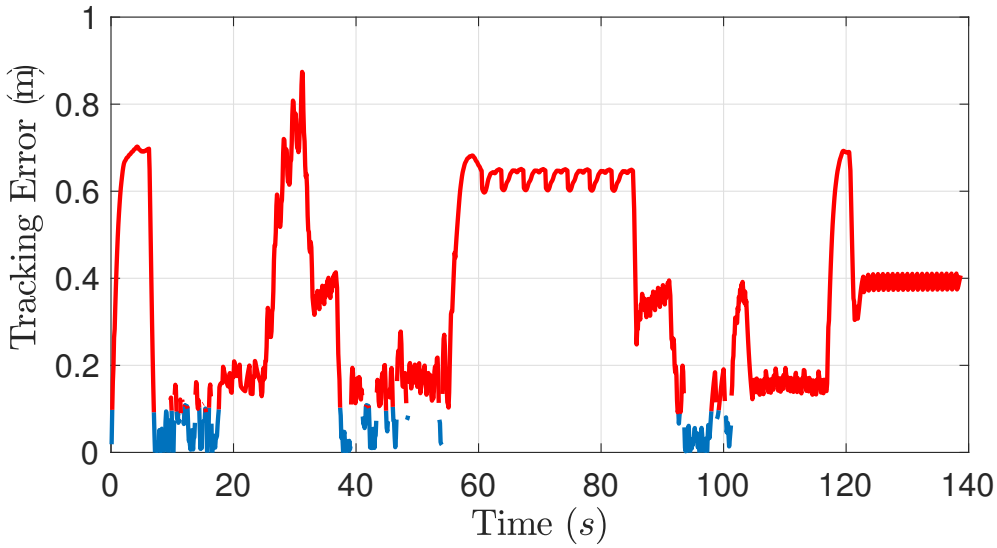


Figure 6.13: Tracking error over time for the 10D-3D example. The red dots indicate that the optimal tracking controller in (6.27) is used, while the blue dots indicate that an LQR controller for the linearized system is used. The tracking error stays below the predicted TEB of 0.9 m, despite worst-case wind.

dynamics in (6.21). The position $(\hat{x}, \hat{y})$ and velocity $(\hat{v}_x, \hat{v}_y)$ are the states of the 4D system. The controls are $(\hat{a}_x, \hat{a}_y)$, which represent the acceleration in each positional dimension. The model parameters are chosen to be $d_0 = 10$, $d_1 = 8$, $n_0 = 10$, $k_T = 0.91$, $g = 9.81$, $|u_x|, |u_y| \leq \pi/9$, $|\hat{a}_x|, |\hat{a}_y| \leq 1$, $|d_x|, |d_y| \leq 0.2$.

Offline precomputation

We define the relative system states to be the error states $(x_r, v_{x,r}, y_r, v_{y,r})$, which are the relative position and velocity, concatenated with the rest of the states in the 8D system. Defining $\Phi = \mathbf{I}_8$ and

$$Q = \begin{bmatrix} \begin{bmatrix} 1 \\ \mathbf{0}_{3\times 1} \end{bmatrix} & \mathbf{0}_{4\times 1} \\ \mathbf{0}_{4\times 1} & \begin{bmatrix} 1 \\ \mathbf{0}_{3\times 1} \end{bmatrix} \end{bmatrix},$$

we obtain the following relative system dynamics:

$$\begin{bmatrix} \dot{x}_r \\ \dot{v}_{x,r} \\ \dot{\theta}_x \\ \dot{\omega}_x \\ \dot{y}_r \\ \dot{v}_{y,r} \\ \dot{\theta}_y \\ \dot{\omega}_y \end{bmatrix} = \begin{bmatrix} v_{x,r} + d_x \\ g\tan\theta_x - \hat{a}_x \\ -d_1\theta_x + \omega_x \\ -d_0\theta_x + n_0 a_x \\ v_{y,r} + d_y \\ g\tan\theta_y - \hat{a}_y \\ -d_1\theta_y + \omega_y \\ -d_0\theta_y + n_0 a_y \end{bmatrix}. \tag{6.24}$$

As in the 10D-3D example in Section 6.6, the relative dynamics are decomposable into two 4D subsystems, and so computations were done in 4D space.

Fig. 6.6 shows the $(x_r, v_{x,r})$-projection of value function across several different times on the left subplot. The total time horizon was $T = 15$, and the value function did not converge. The gray horizontal plane indicates the value of $\underline{V}$, which was 1.14. Note that with increasing τ, $V(r, T-\tau)$ is non-increasing, as proven in Prop. 6. The right subplot of Fig. 6.6 shows the $(x_r, v_{x,r})$-projection of the tvTEB. At $\tau = 0$, the TEB is the smallest, and as τ increases, the size of TEB also increases, consistent with Proposition 6. In other words, the set of possible error states $(x_r, v_{x,r})$ in the relative system increases with time, which makes intuitive sense.

The tvTEB shown in Fig. 6.6 is used to augment planning constraints in the $\hat{x}$ and $\hat{v}_x$ dimensions. Since we have chosen identical parameters for the first four and last four states, the TEB in the $\hat{y}$ and $\hat{v}_y$ dimensions is identical.

On a desktop computer with an Intel Core i7 5820K CPU, the offline computation on a $81 \times 81 \times 65 \times 65$ grid with 75 time points took approximately 30 hours and required approximately 17 GB of RAM.

Online sensing and planning

We use the MPC design in [175] for online trajectory planning. It is formulated as a finite-time optimal control problem (OCP) given by

$$\begin{aligned} \underset{\{\bar{p}_k\}_{k=0}^{N}, \{\bar{u}_k\}_{k=0}^{N-1}}{\text{minimize}} \quad & \sum_{k=0}^{N-1} l(\bar{p}_k, \bar{u}_k) + l_f(\bar{p}_N - p_f) && \text{(6.25a)} \\ \text{subject to} \quad & \bar{p}_0 = p_{\text{init}}, && \text{(6.25b)} \\ & \bar{p}_{k+1} = h(\bar{p}_k, \bar{p}_k), && \text{(6.25c)} \\ & \bar{u}_k \in \mathbb{U}, \;\; \bar{p}_k \in \mathcal{C}_{p,\text{aug}}(t_k), && \text{(6.25d)} \end{aligned}$$

where $l(\cdot,\cdot)$ and $l_f(\cdot)$ are convex stage and terminal cost functions, the future states $\bar{p}_k$ and inputs $\bar{u}_k$ are decision variables, and N is the prediction horizon. The index $t_k = t_0 + k\Delta t_{\text{mpc}}$

denotes for the internal time steps used in MPC, with t_0 and Δt_{mpc} being the current time and the sampling interval of MPC, respectively. Note that N and Δt_{mpc} are chosen such that $t_0 + N\Delta t_{\text{mpc}} \leq T - \tau$ with τ given by Step 9 in Algorithm 1 and T defined for TEB in (6.15). States of the planning model are denoted by the variable $p = (\hat{x}, \hat{v}_x, \hat{y}, \hat{v}_y)$ with current state p_{init} and terminal state p_f. The planning control is denoted by the variable $u = (\hat{a}_x, \hat{a}_y)$. The dynamical system $h(\cdot, \cdot)$ is set to be a zero-order hold discretized model of the 4D dynamics in (6.23). The state and input constraints are $\mathcal{C}_{p,\text{aug}}(t_k)$ and $\mathbb{U}$. Note that the time-varying constraint $\mathcal{C}_{p,\text{aug}}(t_k)$ contains the augmented state constraints:

$$\bar{p}_k \in \mathbb{P}_k := \mathbb{P} \ominus \mathcal{B}_e(t_k), \tag{6.26}$$

where $\mathbb{P}$ denotes the original state constraint, and $\mathcal{B}_e(t_k)$ is the tracking error bound at t_k.

For this example, we represent the augmented constraints as the complement of polytopes, which makes the MPC problem non-convex. We follow the approach presented in [175] to compute a locally optimal solution that uses extra auxiliary variables for each non-convex constraint. We use a horizon $N = 8$ with sampling interval $\Delta t_{\text{mpc}} = 0.2$ s. The MPC replans every 0.8 s. The control frequency is 10 Hz, i.e. $\Delta t = 0.1$ for both the 4D planning and 8D tracking system.

The simulation showing the 8D quadrotor tracking the 4D double integrator is presented in Fig. 6.14. The quadrotor starts at $(2.0, 0.0)$ and seeks to reach the goal, i.e. the blue circle centered at $(9.0, 11.5)$ with a radius of 1.0. Three initially unknown polytopic obstacles make up the constraints $\mathcal{C}_p$.

The quadrotor has a circular sensing region with a radius of 6, colored in green. Unknown obstacles are marked with dotted black boundaries. As the quadrotor explores the environment, obstacles are detected once they are within the sensing range. The union of all sensed obstacles makes up the sensed constraints $\mathcal{C}_{p,\text{sense}}$, whose boundaries are colored in red. They are enclosed by the augmented obstacles shown as dashed polytopes with different colors, each representing a different augmentation of the original sensed obstacles. These illustrate the tvTEB in the position dimensions. The tvTEB is also defined in the velocity dimensions; the value of the bounds over time are shown in Fig. 6.6. Therefore, time-varying constraints $\mathcal{C}_{p,\text{aug}}(t_k)$ in (6.25d) are augmented accordingly in all four dimensions of the planning system.

The MPC planner takes as input the current state p_{init} of the 4D planning system and a list of augmented constraints $\mathcal{C}_{p,\text{aug}}(t_k)$. It then solves the OCP (6.25) in real time for a sequence of optimized control inputs, which steers the planning system towards the goal while avoiding the augmented obstacles. The current state of the planning system is represented as a green star. In front of it, the predicted trajectories in $(\hat{x}, \hat{y})$ space are plotted in dotted curves with the same color as the augmented obstacles considered at the time the MPC problem is solved. The traveled path of the planning system is shown as a solid grey curve. Here, unlike the standard MPC algorithm in which only the first element of the control inputs is used before replanning happens, our proposed MPC planner applies multiple control inputs to the 4D system in open-loop before it replans. This is due to the fact that the planning and tracking system use the same control frequency, i.e. they both update states every Δt

time steps as shown in Step 28 in Algorithm 1; and that the sampling interval of the planner Δt_{mpc} is in general larger than Δt.

The state of the 8D tracking model (6.24) is represented as a red circle. Using the hybrid tracking controller depicted in Fig. 6.4, the 8D system tracks the 4D planning system within the tvTEB in Fig. 6.6, which guarantees constraint satisfaction despite the tracking error at all times. The traveled path of the 8D system is shown as the solid black curve.

Fig. 6.14 also shows four time snapshots of the closed-loop simulation. The top left subplot shows the positional trajectories of the planning and tracking system at $t = 2.6$. At $t = 5.0$, the MPC planning algorithm speeds up and makes a sharp turn into a narrow corridor. This results in a significant deviation between the two trajectories. However, the tracking controller keeps the system within the TEB and no collision is incurred at this time, as shown in the top right subplot. A similar case can be observed in the lower left subplot. Finally, at $t = 13.4$ the quadrotor safely arrives at the destination. Note that the size of the augmented obstacles keeps changing from time step to time step, as indicated by the dashed polytopes with different colors and sizes in the snapshots. Meanwhile the MPC incorporates the updated information of augmented constraints $\mathcal{C}_{p,\text{aug}}(t_k)$ and plans safe trajectories.

Fig. 6.15 shows the tracking error in x_r over time. The red dots indicate the time points at which the optimal tracking controller from Eq. (6.13) is used. One may observe that the tracking error is always less than 0.45, well below the minimal TEB of 1.11 implied by the value function in Fig. 6.6.

Fig. 6.16 shows the tvTEB in the $v_{x,r}$ dimension used in the closed-loop simulation. Each time when replanning is performed via solving the MPC problem in Eq. (6.25), a sequence of 8 TEBs are determined by Step 9 in Algorithm 1, and used for constructing the augmented constraints in Eq. (6.25d). Each error bar and the red dot in the middle shows the range and mean value of the TEBs used within a single MPC planning loop. Enabling a tvTEB allows us to set smaller TEBs for the initial time steps, growing the TEB as the time horizon extends. If we were to use a fixed TEB, we would have to keep the error bound as the largest bound required over the time horizon; this can lead to fairly conservative movement from the planning algorithm. Note that the tvTEB is employed on full states including both position and velocity of the planning system, which allows us to reduce the conservativeness of the planner as much as possible.

We compare the simulation time where the tvTEB is used for planning with the case in that the TEB is fixed as a constant. The results are summarized in Table 6.1. One may observe that with the use of tvTEB the conservativeness in planning is reduced and a shorter flight time is achieved.

Table 6.1: Simulation Time with Constant and tvTEB.

Simulation case	Time (s)
Planning with constant TEB	14.6
Planning with tvTEB	13.4

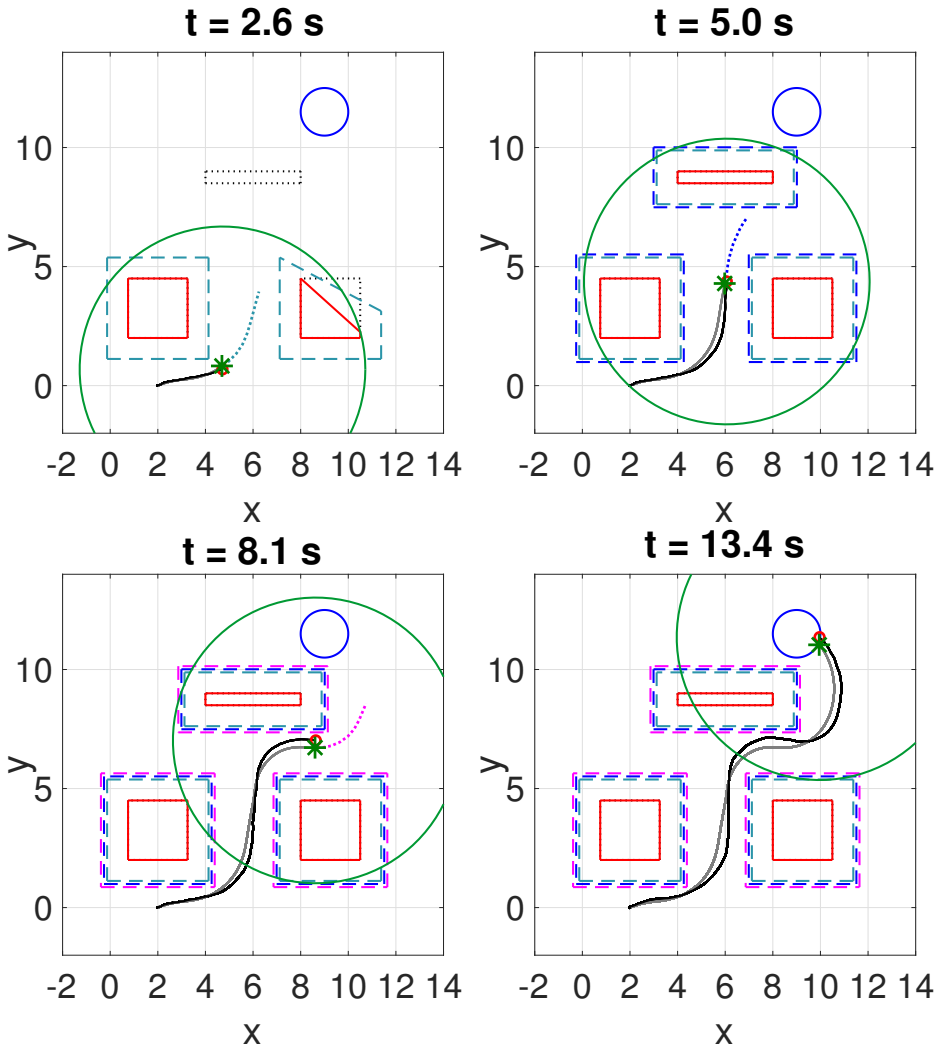


Figure 6.14: Simulation of the 8D-4D example. As the quadrotor with 8D dynamics (position shown as red circle and trajectory shown in black) senses new obstacles, the 4D planning system (position shown as green star and trajectory shown in grey)
replans the trajectory, which is robustly tracked by the 8D system. The time-varying augmented obstacles are plotted as dashed polytopes with different sizes and colors.

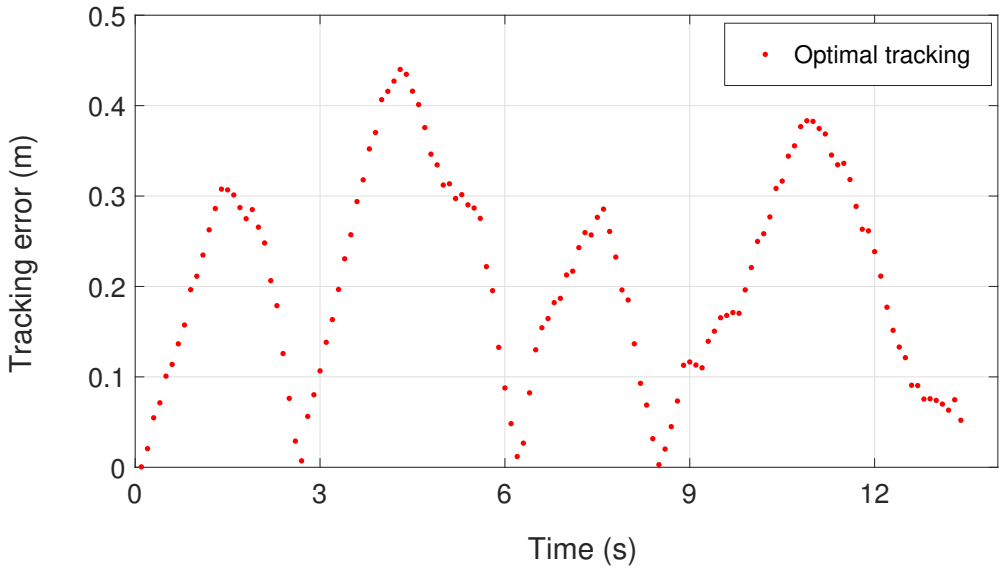


Figure 6.15: Tracking error in x_r over time for the 8D-4D example. The red dots represent the time steps when the optimal tracking controller in (6.13) is used. The tracking error remains lower than the minimal TEB of 1.11m at all times.

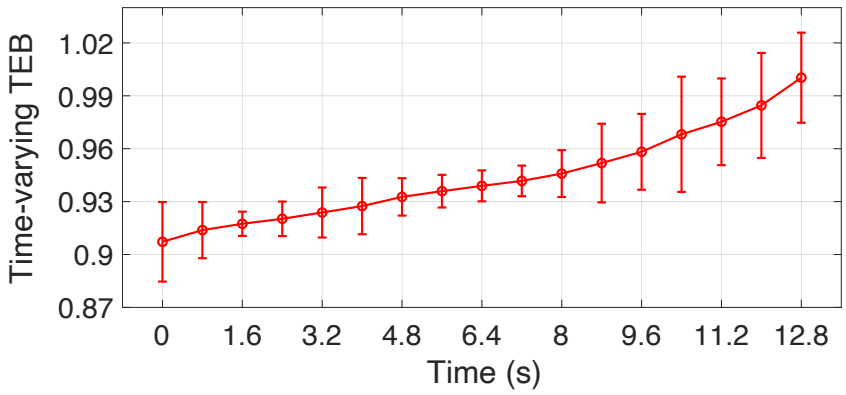


Figure 6.16: Time-varying TEB over time in the $v_{x,r}$ dimension used by the MPC planner. The error bars are plotted at time steps when the MPC replans. They characterize the range of TEBs used for each planning while the red dots shows the mean value.

Implementation of the MPC planner was based on MATLAB and `ACADO Toolkit` [89]. The nonlinear MPC problem was solved using an online active set strategy implemented in `qpOASES` [67]. All the simulation results were obtained on a laptop with Ubuntu 14.04 LTS operating system and a Core i5-4210U CPU. The average computation time for solving the MPC problem (6.25) was 0.37 s.

6.7 Chapter Summary

This chapter introduced FaSTrack: Fast and Safe Tracking, a framework for providing trajectory -independent tracking error bounds (TEB) for a tracking model representing an autonomous system, and a planning model representing a simplified model of the autonomous system. The TEB is obtained by analyzing a pursuit-evasion game between the two models, and obtaining the associated value function by solving a Hamilton- Jacobi (HJ) variational inequality.

We demonstrated the framework's utility in three representative numerical simulations involving a 5D car model tracking a 3D car model planning using the fast sweeping method, a 10D quadrotor model tracking a 3D single integrator model planning using rapid-exploring random trees, and an 8D quadrotor model tracking a 4D double integrator planning using model predictive control. We considered simulated environments with static obstacles, but FaSTrack has been demonstrated in hardware to safely navigate around environments with static obstacles [76] and moving human pedestrians [72].

There are still challenges and simplifications to address. The offline computation can be computationally prohibitive, a challenge we are working to address using techniques such as HJ reachability decomposition [38, 35], sophisticated optimization techniques [161], and approximate dynamic programming [150]. We are also interested in exploring methods to reduce conservativeness of the TEB by relaxing the worst-case assumption on the goals of the planning control. Additionally, identifying when a particular tracking-planning model will have a converged value function remains an open question. Finally, we currently assume both perfect sensing and a perfectly representative tracking model. We would like to alleviate these assumptions by bounding uncertainty from sensing error, and making online updates to the value function as information about the tracking model is improved.

By computing trajectory-independent TEB, our framework decouples robustness guarantees from planning, and achieves the best of both worlds – formal robustness guarantees which is usually computationally expensive to obtain, and real-time planning which usually sacrifices robustness. Combined with any planning method in a modular fashion, our framework enables guaranteed safe planning and replanning in unknown environments, among other potential applications.

Appendix: Infinite Time Horizon TEB

When the value function (6.11) converges, we write $V(T,r) := V_\infty(r)$. The optimal controller is then given by

$$u_s^*(r) = \arg\min_{u_s \in \mathcal{U}_s} \max_{u_p \in \mathcal{U}_p, d \in \mathcal{D}} \nabla V(r) \cdot g(r, u_s, u_p, d). \tag{6.27}$$

with the optimal planning control and disturbance given by

$$\begin{bmatrix} u_p^* \\ d^* \end{bmatrix}(r) = \arg\max_{u_p \in \mathcal{U}_p, d \in \mathcal{D}} \nabla V_\infty(r) \cdot g(r, u_s^*, u_p, d). \tag{6.28}$$

The smallest level set corresponding to the value $\underline{V}_\infty := \min_r V_\infty(r)$ can be interpreted as the smallest possible tracking error of the system, and the TEB is given by the set

$$\mathcal{B}_\infty = \{r : V_\infty(r) \leq \underline{V}_\infty\}. \tag{6.29}$$

with the TEB in error state subspace is given by

$$\mathcal{B}_{e,\infty} = \{e : \exists \eta, V_\infty(e, \eta) \leq \underline{V}_\infty\}. \tag{6.30}$$

In the online implementation in Alg. 1, one replaces all mentions of value function and TEB with their infinite time horizon counterpart, and skip Line 9. Finally, Prop. 7 provides the infinite time horizon result analogous to Prop. 6.

Proposition 7. ***Infinite time horizon guaranteed TEB****. Given $t \geq 0$, $\forall t' \geq t$, $r \in \mathcal{B}_\infty \Rightarrow \zeta_g^*(t'; r, t) \in \mathcal{B}_\infty$, with ζ_g^* defined the same way as in* (6.17b) *to* (6.17e)*.*

Proof: Suppose that the value function converges, and define

$$V_\infty(r) := \lim_{T \to \infty} V(r, T) \tag{6.31}$$

We first show that for all t, t' with $t' \geq t$,

$$V_\infty(r) \geq V_\infty(\zeta_g^*(t'; r, t)). \tag{6.32}$$

Without loss of generality, assume $t = 0$. Then, we have

$$\begin{aligned}
V_\infty(r) &= \lim_{T \to \infty} \max_{\tau \in [0,T]} l(\zeta_g^*(\tau; r, 0)) && \text{(6.33a)} \\
&= \lim_{T \to \infty} \max_{\tau \in [-t',T]} l(\zeta_g^*(\tau; r, -t')) && \text{(6.33b)} \\
&\geq \lim_{T \to \infty} \max_{\tau \in [0,T]} l(\zeta_g^*(\tau; r, -t')) && \text{(6.33c)} \\
&= \lim_{T \to \infty} \max_{\tau \in [0,T]} l(\zeta_g^*(\tau; \zeta_g^*(0; r, -t'), 0)) && \text{(6.33d)} \\
&= \lim_{T \to \infty} \max_{\tau \in [0,T]} l(\zeta_g^*(\tau; \zeta_g^*(t'; r, 0), 0)) && \text{(6.33e)} \\
&= V_\infty(\zeta_g^*(t'; r, 0)) && \text{(6.33f)}
\end{aligned}$$

Explanation of steps:

- (6.33a) and (6.19f): by definition of value function
- (6.33b): shifting time by $-t'$
- (6.33c): removing the time interval $[-t', 0)$ in the max operator
- (6.33d): splitting trajectory $\zeta_g^*(\tau; r, -t')$ into two stages corresponding to time intervals $[-t', 0]$ and $[0, \tau]$
- (6.33e): shifting time reference in $\zeta_g^*(0; r, -t')$ by t', since dynamics are time-invariant

Now, we finish the proof as follows:

$$
\begin{aligned}
r \in \mathcal{B}_\infty &\Leftrightarrow V_\infty(r) \leq \underline{V} && \text{(6.34a)}\\
&\Rightarrow V_\infty(\zeta_g^*(t'; r, t)) \leq \underline{V} && \text{(6.34b)}\\
&\Leftrightarrow \zeta_g^*(t'; r, t) \in \mathcal{B}_\infty, && \text{(6.34c)}
\end{aligned}
$$

where (6.32) is used for the step in (6.34b). ■

Chapter 7

Meta-Planning with FaSTrack

This chapter is based on the paper "Planning, Fast and Slow: A Framework for Adaptive Real-Time Safe Trajectory Planning" [76], written in collaboration with David Fridovich-Keil, Jaime Fisac, Sampada Deglurkar, and Claire J. Tomlin.

The navigation of autonomous dynamical systems through cluttered environments is a hard problem, particularly when there is a need for both speed and safety. Often, elements of the environment (such as obstacle locations) are also unknown *a priori*, further complicating the problem. Many popular methods exist for planning trajectories in such scenarios, but a key challenge lies in accounting for dynamic feasibility in real time while providing a safety guarantee. Some of the most common approaches in this space are sampling-based planners such as rapidly-exploring random trees (RRTs) [110]. Typically, these planners fall into one of two broad categories: *geometric* planners only attempt to find a *path* the system can take from its current position to the goal, while *kinodynamic* planners find a dynamically feasible *trajectory*, i.e. a path with associated time stamps that adheres to some known system dynamics.

Since the output of a geometric planner is not usually dynamically feasible, a common practice is to apply a feedback controller, e.g. a linear quadratic regulator (LQR), to attempt to track a geometric plan. Since the controller will not follow the plan perfectly, geometric plans are usually generated by assuming an ad hoc safety margin. This idea is illustrated in Fig. 7.1(a-b).

In practice, this safety margin is almost always a *conservative heuristic* chosen by the operator. However, the recently-developed Fast and Safe Tracking (FaSTrack) framework [88] provides a rigorous way to precompute a safety margin offline, given a model of the true system dynamics and a (possibly lower-dimensional) model of the online planner's dynamics. In the FaSTrack framework, a guaranteed maximum possible tracking error is computed between the tracking system model and the planning model. This tracking error bound (TEB) can also accommodate deviations due to external disturbances such as wind and time delays. The TEB is used to expand obstacles by a margin that *guarantees* safety. The offline

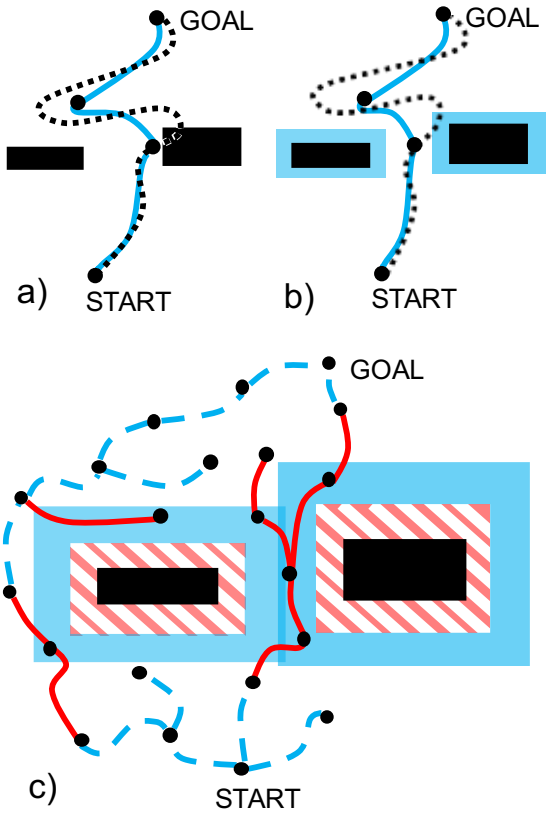


Figure 7.1: (a) A dynamical system (black, dotted) may not be able to track the output of a geometric planner (blue, solid), resulting in collision with an obstacle. (b) Often planners account for tracking error by heuristically augmenting obstacles; however, the system may still deviate from the planned path by more than this margin. (c) Schematic of meta-planner operation using fast (blue, dashed) and slow (red, solid) planning models with correspondingly large (blue, solid) and small (red, hatched) TEB-augmented obstacles.

precomputation also provides a computationally efficient safety controller that maps the relative state between the tracking system and the planned trajectory at any given time to the most effective control action for the tracking system to remain within the TEB. Hence, the online algorithm involves real-time planning using a fast, potentially low-dimensional planning model, and quickly computable robust optimal tracking of the planned trajectory using a higher-dimensional tracking model.

While FaSTrack makes no significant assumptions about the specific type of low-dimensional planner, in this work we focus our attention on geometric planners operating in the robot's configuration space. We observe that the resulting geometric paths can be interpreted as kinematic trajectories with a fixed maximum speed in each dimension. We emphasize that the restriction to geometric planners is pedagogical; like FaSTrack, our proposed meta-planning approach is more general and extends to more complex planning models.

One key drawback of FaSTrack is that the TEB can be overly conservative if the system is tracking a particularly difficult-to-track planning model. In this chapter we propose an extra layer to the core framework that allows combining multiple planning models with different maximum speeds, and hence different TEBs. We call this process *meta-planning*, and it effectively generates a tree of trajectories that switch between "faster" and "slower" planning models, as illustrated in Fig. 7.1(c). Faster planning models are able to navigate through the environment quickly, but their larger TEBs prevent them from threading narrow passages between obstacles. Slower planning models take more time to traverse the environment, but the correspondingly smaller TEBs allow them to maneuver more precisely near obstacles. By adaptively selecting the planning model in real time, our framework can trade off between speed of navigation and size of the TEB. Crucially, our meta-planning scheme can quickly and *safely* adapt to the presence of obstacles detected at motion time.

The main contributions of this chapter are the aforementioned real-time meta-planning algorithm for Fast and Safe Tracking, a constructive proof of safety, and a demonstration of the full algorithm both in simulation and hardware using a small quadrotor vehicle.

7.1 Related Work

Robust motion planning and trajectory optimization have been active areas of research in recent years. However, navigation that is both robust and fast is still a challenge. Sampling-based motion planners can be computationally efficient, but attempts to make them robust are generally heuristic. Other techniques for online dynamic navigation include model predictive control (MPC), which is extremely useful, particularly for linear systems. MPC is harder to use in real time for nonlinear systems due to the computational costs of solving for dynamic trajectories, though work to speed up computation is ongoing [54, 134]. Robustness can be achieved in linear systems [147, 53], and there is work on making MPC for nonlinear systems robust by using algorithms based on minimax formulations and tube MPCs that bound output trajectories with a tube around a nominal path (see [90] for references).

There are other techniques for robust navigation that take advantage of precomputation. Safety funnels can be constructed around motion primitives that can then be pieced together in real time [122]. Given a precomputed nominal dynamically feasible trajectory, contraction mapping can be used to make this nominal trajectory more robust to external disturbances in real time [160]. Finally, Hamilton-Jacobi (HJ) reachability analysis has been used for *offline* robust trajectory planning in fully known environments, providing guaranteed tracking error bounds under external disturbances [18].

The meta-planning aspect of this chapter was inspired by behavioral economist Daniel Kahneman's Nobel Prize winning work on "fast" (intuitive) and "slow" (deliberative) modes of cognitive function in the brain [94]. Thinking with the "fast system" is efficient, but more error-prone. Thinking with the "slow system" is less error-prone, but slower. The brain adaptively chooses which mode to be in to operate efficiently while minimizing error in scenarios where error can be disastrous. This act of deciding how much cognitive effort

to expend for a given task is called metareasoning [151], and can be useful for robotics. It may be desirable for a robot to plan and move swiftly whenever possible, but to operate more carefully when approaching a challenging region in the environment. Research in psychology has suggested that selecting between a limited number of discrete cognitive modes is computationally advantageous [125], which inspires the use of a discrete set of faster and slower planning models in our meta-planning algorithm. Our algorithm is able to trade off planner velocity and tracking conservativeness in a modular way while providing a strong theoretical safety guarantee.

7.2 Background

The FaSTrack framework can be used to plan and track a trajectory online and in real time. The real-time planning is done using a set of kinematic or dynamic planning models, and the physical system is represented by a dynamic tracking model that will attempt to follow the current planning model. The environment can contain static *a priori* unknown obstacles provided they can be observed by the system within a limited sensing range.[1] In this section we will define the tracking and planning models and their relation to one another, and present a brief overview of FaSTrack.

Tracking Model

The tracking model should be a realistic representation of the real system dynamics, and in general may be nonlinear and high-dimensional. Let s represent the state variables of the tracking model. The evolution of the dynamics satisfies the ordinary differential equation (ODE):

$$\begin{aligned} \frac{ds}{dt} = \dot{s} = f(s, u_s, d), t \in [0, T] \\ s \in \mathcal{S}, u_s \in \mathcal{U}_s, d \in \mathcal{D} \end{aligned} \tag{7.1}$$

The trajectories of (7.1) will be denoted as $\zeta_f(t; s_0, t_0, u_s(\cdot), d(\cdot))$, where $t_0, t \in [0, T]$ and $t_0 \leq t$. Under standard technical assumptions [88], these trajectories will satisfy the initial condition and the ODE (7.1) almost everywhere. For a running example we will consider a tracking model of a simple double-integrator with control u_s and disturbances $d = [d_v, d_a]^T$:

$$\begin{bmatrix} \dot{s}_x \\ \dot{s}_{vx} \end{bmatrix} = \begin{bmatrix} s_{vx} - d_v \\ u_s - d_a \end{bmatrix} \tag{7.2}$$

[1] In order to provide safety guarantees, the minimum allowable sensing distance in any direction is the length of the TEB's projection onto that direction, added to the largest distance the current planning reference could move while a new meta-plan is generated.

Planning Model

The planning model defines the class of trajectories generated by the motion planner. Let p represent the state variables of the planning model, with control u_p. The planning states $p \in \mathcal{P}$ are a subset of the tracking states $s \in \mathcal{S}$. FaSTrack is agnostic to the type of planner, as long it can be represented using a kinematic or dynamic model as follows:

$$\frac{dp}{dt} = \dot{p} = h(p, u_p), t \in [0, T], p \in \mathcal{P},\ \underline{u_p} \leq u_p \leq \overline{u_p} \tag{7.3}$$

This chapter focuses on geometric planners. Although geometric planners may not directly use a dynamical model, the paths they generate can be described by a point with direct velocity control. For example, a 1D geometric planner could be described as a point moving with a direct velocity controller: $\dot{p}_x = u_p$. Note that the planning model does not need a disturbance input. Disturbances need only be considered in the tracking model and not the planning model, since the latter only exists in the abstract as a reference for the former.

Relative Dynamics

The FaSTrack framework relies on using the relative dynamics between the tracking and planning models. The relative system may be derived by lifting the planner's state from $\mathcal{P}$ to $\mathcal{S}$ and subtracting:

$$r = s - Qp, \qquad \dot{r} = g(r, u_s, u_p, d) \tag{7.4}$$

Q is a matrix that matches the common states of s and p by augmenting the state space of the planning model. The relative states r now represent the tracking states relative to the planning states. Using our tracking and planning model examples from above we can define the dynamics of a double-integrator tracking a 1D point mass as:

$$\begin{bmatrix} \dot{r}_x \\ \dot{s}_{vx} \end{bmatrix} = \begin{bmatrix} s_{vx} - d_v - u_p \\ u_s - d_a \end{bmatrix} \tag{7.5}$$

The FaSTrack Framework

The FaSTrack framework, explained in detail in [88], consists of both an offline precomputation algorithm and an online planning algorithm. Together, these allow a nonlinear dynamic system to navigate through an *a priori* unknown environment with static obstacles, safely and in real time.

Offline, FaSTrack computes a tracking error bound (TEB) and a safety controller to stay inside this bound. The TEB is a safety margin that, when using the safety controller, guarantees robust tracking despite worst-case planner behavior and bounded disturbances. The safety controller operates on the *relative* state between tracker and planner, and is computed offline via HJ reachability analysis *in free space*. This is possible because the relative dynamics do not depend on the absolute state of the tracking system in the environment.

Since the tracker will always remain inside the TEB, as long as the TEB never intersects any obstacles, the free space relative dynamics will always apply.

Online, both at the start and whenever a new obstacle is sensed, an off-the-shelf planning algorithm—equipped with the precomputed TEB for collision-checking—generates a new trajectory. The tracking system may then apply the precomputed safety controller to track this planned trajectory in real time.

7.3 Meta-Planning

General Framework

In this work, we use the term *planner* to denote the conjunction of a planning algorithm and an associated planning model that it uses to generate timed trajectories.This chapter's main contribution to the FaSTrack framework is the introduction of a meta-planning algorithm to choose between a selection of planners $\{\pi_i\}_{i=1}^N$ with different maximum speeds and hence different TEBs at runtime. We first assume that planners are sorted in order of decreasing maximum speed and hence TEB size, and that the overall objective is to minimize the time to reach a specified goal point. This objective implies a preference for planners that can move faster, but also for planners that can safely navigate a more direct route even if they must do so at lower speed.

The core of the meta-planner is a random tree $\mathcal{T}$ inspired by RRT-style sampling-based planners [110], as shown in Fig. 7.1.[2] The obstacles are shown in black, and are augmented by the TEBs for two different planners. As in RRT, waypoints in $\mathcal{P}$ are sampled randomly from the environment and (potentially) connected with their nearest neighbor in $\mathcal{T}$. If the fast planner[3] (with the large blue TEB) finds a collision-free trajectory, the connection is established (dashed blue lines). Otherwise, the slow planner (with smaller red striped TEB and solid red lines) attempts to connect to the nearest neighbor. Upon success, the waypoint is inserted into $\mathcal{T}$, along with the trajectory generated by the planner to reach that waypoint from the nearest neighbor, and the associated safety controller to remain inside the TEB. If a waypoint is successfully inserted near the goal, a similar process ensues to attempt to find a trajectory between it and the goal point.

Once a valid "meta-plan" is found from start to goal, the meta-planner continues building $\mathcal{T}$ until a user-specified maximum runtime has elapsed, always retaining the best (shortest time) sequence of waypoints to the goal. Similar to Informed RRT* [78], the meta-planner immediately rejects samples which could not possibly improve upon the best available trajectory.[4]

[2]The choice of a tree topology is for convenience; any directed graph including the robot's current state would suffice.

[3]Note that by a *faster* planner we mean one with a higher associated maximum velocity, rather than smaller computation time.

[4]In Informed RRT*, planner velocities lie on the sphere leading to an elliptical geometry for valid samples. Since we assume a maximum speed *in each dimension*, valid samples lie in a distorted rhombicuboctahedron.

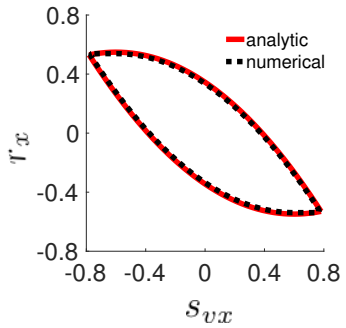


Figure 7.2: Invariant set that the double-integrator can remain in despite worst-case disturbance and planning control for the both numerical solution (dotted) and analytic solution (solid).

The key to meta-planning lies in ensuring safe switching between planners. This guarantee requires an offline computation to determine a safety margin for switching into successively slower planners (with smaller TEBs), as well as a safe switching control law. Online, we must be sure to plan with the appropriate safety margin at each step, and to "backtrack" if we detect the need for a switch to a slower planner. We will next explore the offline and online steps in detail.

Offline Reachability Analysis

There are two major components to the offline precomputation for the meta-planner. The first step is to compute the TEB and safety control look-up tables for each planner. This is done following the standard FaSTrack precomputation algorithm [88]. Fig. 7.2 shows the set of relative states in the x-subsystem that the tracker can remain within despite worst case planner behavior and external disturbance. The projection of this controlled invariant set onto the position axis comprises the x-TEB. For the double-integrator dynamics in (7.5), an analytic solution can also be found by applying the equations of constant-acceleration motion under the worst-case disturbance and the best associated control effort. The analytic controlled invariant set, consisting of two parabolic curves, is superimposed in Fig. 7.2. Such analytic solutions do not exist in general.

The second major component of the offline precomputation is to find the corresponding tracking bound and optimal controller for transitioning between planners. For the dynamics in (7.5), switching from a planner with a small TEB to one with a large TEB is safe by construction, because the large TEB *contains* the small TEB. Switching from a large TEB to a small one is more complicated.

To transition from a large TEB to a small TEB we must ensure that the relative state between the autonomous system and the planned path is within the small TEB by the time of the planner switch. FaSTrack provides the optimal control for staying within each bound individually, but does not provide the controller and bound required for reducing the tracking

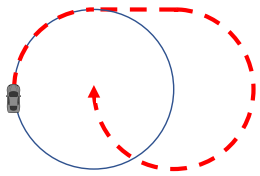


Figure 7.3: Example of a Dubins car that must leave its tight orbit in order to eventually move closer to the origin. This example illustrates why the switching safety bound may generally be larger than the tracking error bound.

error prior to a switch. Perhaps surprisingly, in general the tracker may first need to exit the large TEB before converging to the small TEB. Fig. 7.3 provides an intuitive example of this phenomenon. Here, a Dubins car moving at a fixed speed remains within radius R of the origin by turning at its maximum steering angle. In order for the car to reduce its distance to the origin, it must first exit the original circle to reorient itself towards the origin. In general we must precompute the set of states that the system may visit when transitioning from a large TEB to a small TEB, and the optimal control to achieve this transition. To do this we use HJ reachability analysis.

HJ reachability analysis provides a rigorous mechanism for analyzing the goal satisfaction of a system, and can be used to determine the backward reachable tube (BRT). The BRT is the set of all allowable initial states of a system such that it can enter a set of goal states within a given time interval. HJ reachability analysis can also be used in the context of differential pursuit-evasion games [91, 40]. Here, as in FaSTrack [88], we assume there is such a game between the tracking system and the planning system. In this game, the tracking system will try to "capture" the planning system, while the planning system is attempting to avoid capture. In practice, the planner is not actively trying to avoid the tracker, but this assumption accounts for unexpected, worst-case planner behavior. We want to determine the set of states that the tracking system may visit when transitioning from the larger TEB to the smaller TEB.

Before constructing the differential game we must first determine its information structure, i.e. how and when each player makes decisions. Since the relative dynamics between the tracker and planner are decoupled in their respective inputs, and we assume an additive disturbance, it is in fact irrelevant who "plays first" at each time instant, and the value of the game is well defined under feedback strategies.

For the system in the form of (7.4), we would like to compute the BRT of time horizon T, denoted $\mathcal{V}(T)$. Intuitively, $\mathcal{V}(T)$ is the set of states from which there exists a control strategy to drive the system into a target set $\mathcal{L}$ within a duration of T despite worst-case disturbances. Formally, the BRT is defined here as

$$\begin{aligned}\mathcal{V}(T) = \{r : &\exists u_s(\cdot) \in \mathbb{U}_s, \forall u_p(\cdot) \in \mathbb{U}_p, \forall d(\cdot) \in \mathcal{D}, \\ &r(\cdot) \text{ satisfies (7.4)}, \\ &\exists t \in [t_0 - T, t_0], \zeta(t; r, t_0, u_s(\cdot), d(\cdot)) \in \mathcal{L}\}\end{aligned} \tag{7.6}$$

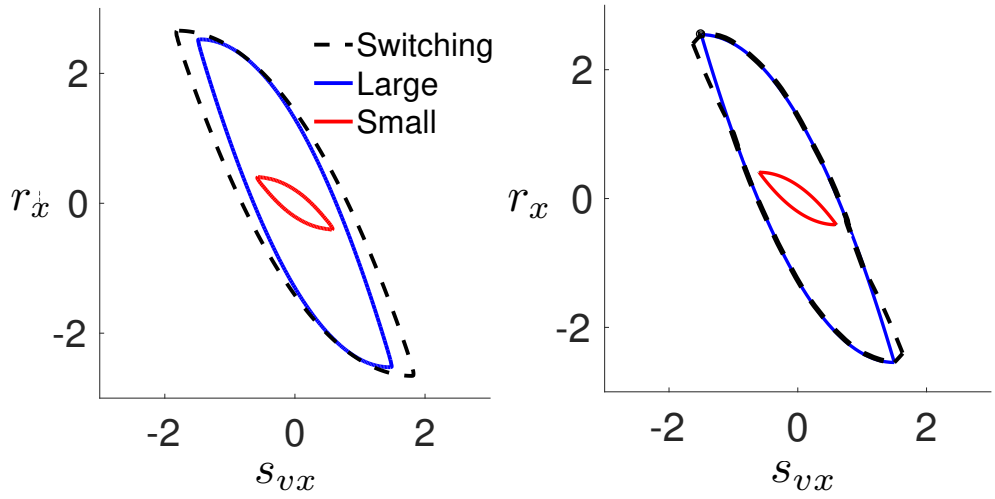


Figure 7.4: Visualizations of the x-subsystem's numerical (left) and analytic (right) controlled invariant sets for two different planners. The numerical SSB is guaranteed to over-approximate the minimal SSB.

where $\mathbb{U}_s, \mathbb{U}_p, \mathcal{D}$ denote the sets of feedback strategies for the tracker, planner and disturbance.

Standard HJ formulations exist for computing the BRT in general [19, 126, 26, 73], and more efficiently for decomposable systems [38]. Here the target $\mathcal{L}$ is the set of states represented by the smaller tracking error bound. Using the relative dynamics between the tracking model and the planning model associated with the smaller TEB, we evolve this set backwards in time. We stop the computation when the tube contains the set of states associated with the larger TEB. This BRT represents the set of states from which the system can enter the small TEB, as well as the states that the trajectories may enter along the way. By projecting this set onto the position dimensions we obtain a *switching safety bound* (SSB). We note that this is an *over-approximation* of the minimal SSB because it includes trajectories that do not originate inside the larger TEB's controlled invariant set. The SSB precomputation also generates the switching controller. Continuing our double-integrator example, Fig. 7.4a shows the controlled invariant sets associated with the larger and smaller TEBs, and the over-approximated set associated with the SSB. The same information computed analytically is shown in Fig. 7.4b, where the minimal SSB may be computed exactly.

Online Meta-Planning

At runtime, the meta-planner is in charge of constructing and maintaining a tree, $\mathcal{T}$, of waypoints connected via trajectories generated by the set of available planners. It is also responsible for re-planning whenever new information about the environment becomes available, i.e. when obstacles are detected.

The precomputed safety sets allow the meta-planner to reason quickly about dynamic

tracking feasibility as it builds $\mathcal{T}$. Using the precomputed TEBs, the meta-planner can determine which planners are safe to use in different regions of the environment. In addition, the SSBs allow the meta-planner to determine the validity of planner-to-planner transitions. The meta-planner's logic is detailed below and illustrated in Fig. 7.5.

Step 0: Root. The root node of $\mathcal{T}$ is initially set at the starting position of the tracking system. Since the system has an initial tracking error equal to zero, it is by definition inside of all the available TEBs. Later, if an obstacle has just been detected mid-trajectory, the new root node will be placed at the predicted position of the *planning* system after some allowed computation time (typically < 1 s) and the *tracking* system will only be guaranteed to be inside the TEB associated to the current edge of $\mathcal{T}$.

Step 1: Sample. The meta-planner constructs its tree $\mathcal{T}$ by sequentially sampling points uniformly at random from the environment and attempting to connect them to the nearest existing waypoint in the tree.

Step 2: Plan. By default, the meta-planner always tries to connect waypoints using the fastest planner π_1, which is also associated with the largest TEB. If π_1 does not find a collision-free trajectory, the meta-planner then attempts to use the second-fastest planner π_2, which has a smaller TEB. The meta-planner continues trying available planners in order of decreasing TEB size until one succeeds or all have failed (in which case it abandons this candidate waypoint and samples a new one).

Step 3: Virtual Backtrack. When a planner π_k succeeds in reaching a new point p from the nearest waypoint $w \in \mathcal{T}$, the meta-planner checks what planner was previously used to reach waypoint w from its parent $v \in \mathcal{T}$. If this preceding planner π_j had a larger TEB than the new planner (that is, if $j < k$), then p cannot be immediately added to $\mathcal{T}$. Instead, the meta-planner first needs to ensure that the tracking system will be able to safely transition into TEB_k *before* reaching w, so that it can then track π_k's plan from w to p while remaining inside its TEB. The meta-planner does this is by checking what planner π_i was used to reach w's parent v, and if $i < k$, using the safe switching bound $\text{SSB}_{i \to k}$ to collision-check the already-computed path $v \to w$. If $i \geq k$, there is no need to use a SSB and the path $v \to w$ is guaranteed to be safe under TEB_k, since it was already deemed safe under the larger TEB_j by π_j.

If the check is successful, this means that, instead of getting from v to w tracking the faster planner π_j, the system can follow an alternative trajectory, skipping π_j altogether and transitioning from the speed of π_i to the speed of π_k. This path is added to $\mathcal{T}$ as an *alternative* to the original $v \to w$ path: the more-slowly-reached w is a new node in $\mathcal{T}$, and p is added to $\mathcal{T}$ as a child of this new node.

If the check is unsuccessful, the meta-planner does not add p to the tree. Two different options for handling this possibility are as follows:

a) **Discard:** p is discarded and the meta-planner moves on to sample a new candidate point.

b) **Recursive Virtual Backtrack:** the meta-planner marks v as a waypoint that needs to be reached from its parent using a *slower* planner than the original π_i, so that safe

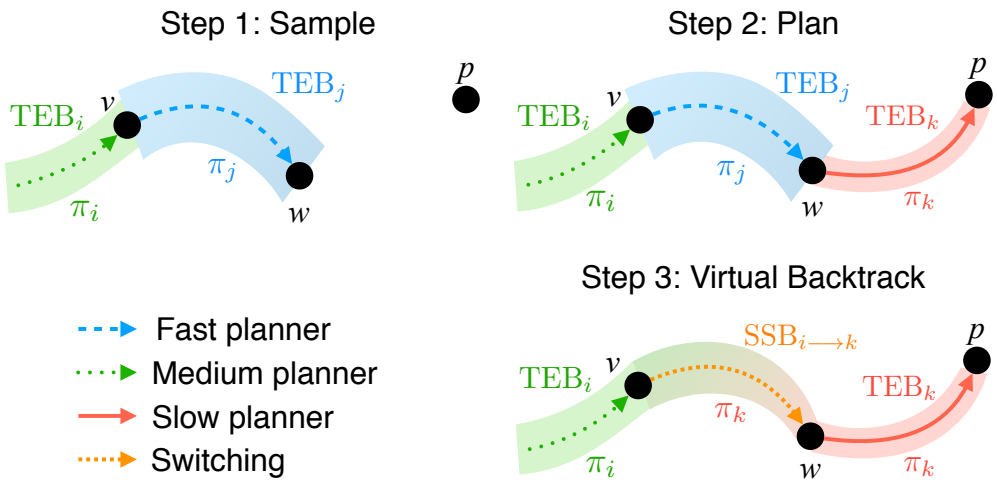


Figure 7.5: Illustration of the online meta-planning algorithm.

transition into TEB_k will be possible. This will always be the case if v is reached using π_j, since $v \to w$ is safe under $\text{TEB}_j \subset \text{TEB}_i$. Step 3 can then be repeated on v, and recursively applied (at worst) until the root of $\mathcal{T}$.

One alternative option for handling planner-switching failures is to prevent them altogether by always using SSBs instead of TEBs for the planning in Step 2. In particular, replacing TEB_i with $\text{SSB}_{i \to N}$ will ensure that planners will only attempt to add a candidate point p to the tree if it would not only be possible to reach p under this planner but also, if later deemed necessary, to do so while transitioning to the smallest TEB (so that subsequent nodes can be connected to it by any planner without the need for the backtracking verification in Step 3). The additional conservativeness introduced by this substitution depends on the relative tracker-planner dynamics, namely on how much larger $\text{SSB}_{i \to N}$ is than TEB_i.

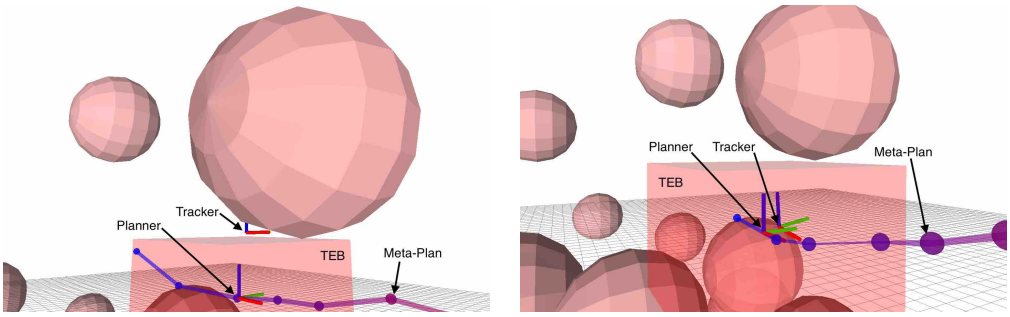
Remark 11. *In the case of a point-mass tracking model following a kinematic planner, we have* $SSB_{i \to j} = TEB_i$, $\forall j > i$, *and therefore this substitution does not need to be done explicitly nor does it introduce any additional conservativeness. The backtracking check in Step 3 is* always guaranteed to succeed.

Proposition 3. *Any plan generated by the meta-planner algorithm can be safely followed by the tracking system.*

Proof. The proof is by construction of the meta-planner, based on FaSTrack guarantees; we provide an outline here. A point is only added to the meta-planning tree if there exists a sequence of planned trajectories that reach the point such that (a) each planned trajectory can be tracked by the system with an error bounded by the associated TEB, and is clear of known obstacles by at least TEB, (b) each transition between planners can be followed by the system with an error bounded by the corresponding SSB, and is clear of known obstacles

by at least SSB, and (c) if new obstacles are detected, re-planning succeeds (at worst, a geometric planner can always reverse or stop) in time for the system to switch to the new plan before colliding. □

7.4 Results



(a) LQR controller. (b) Safety controller.

Figure 7.6: Simulated autonomous flight in a cluttered environment. Notice that when using LQR control the quadrotor leaves the TEB, but under optimal safety control it remains within the TEB. This is particularly important in the vicinity of obstacles.

We demonstrate our algorithm on a 6D near-hover quadrotor model tracking a suite of 3D geometric planners running BIT* [77] in the cluttered environment depicted in Fig. 7.6 with different maximum speeds in each dimension. The tracking[5] and planning models (for the i^{th} planner π_i) are given below in Eq. 7.7 (tracker at left, planner at right):

$$\begin{bmatrix} \dot{s}_x \\ \dot{s}_y \\ \dot{s}_z \\ \dot{s}_{vx} \\ \dot{s}_{vy} \\ \dot{s}_{vz} \end{bmatrix} = \begin{bmatrix} s_{vx} - d_{vx} \\ s_{vy} - d_{vy} \\ s_{vz} - d_{vz} \\ g\tan\theta - d_{ax} \\ -g\tan\phi - d_{ay} \\ T - g - d_{az} \end{bmatrix}, \qquad \begin{bmatrix} \dot{p}_x \\ \dot{p}_y \\ \dot{p}_z \end{bmatrix} = \begin{bmatrix} b_x^{(i)} \\ b_y^{(i)} \\ b_z^{(i)} \end{bmatrix} \tag{7.7}$$

Here $u_s = [\theta, \phi, T]^T$ and correspond to roll, pitch, and thrust. In all experiments, we set $-0.15 \text{ rad} \leq [\theta, \phi] \leq 0.15 \text{ rad}$ and $7.81\ m/s^2 \leq T \leq 11.81\ m/s^2$. Planner π_i's controls are $u_p = [b_x^{(i)}, b_y^{(i)}, b_z^{(i)}]$, each representing a fixed maximum speed in the given dimension. Due to the form of (7.7), the optimal safety controller will be bang-bang. However, it is only critical to apply the safety control at the *boundary* of the TEB. A smooth linear controller may be used in the interior, following a least-restrictive supervisory control law. The relative

[5]Note that we have assumed a zero yaw angle for the quadrotor. This is enforced in practice by using a strict feedback controller on yaw rate to regulate yaw to zero.

dynamics between the tracking and planning models are:

$$\begin{bmatrix} \dot{r}_x \\ \dot{r}_y \\ \dot{r}_z \\ \dot{r}_{vx} \\ \dot{r}_{vy} \\ \dot{r}_{vz} \end{bmatrix} = \begin{bmatrix} s_{vx} - d_{vx} - b_x^{(i)} \\ s_{vy} - d_{vy} - b_y^{(i)} \\ s_{vz} - d_{vz} - b_z^{(i)} \\ g \tan\theta - d_{ax} \\ -g \tan\phi - d_{ay} \\ T - g - d_{az} \end{bmatrix} \tag{7.8}$$

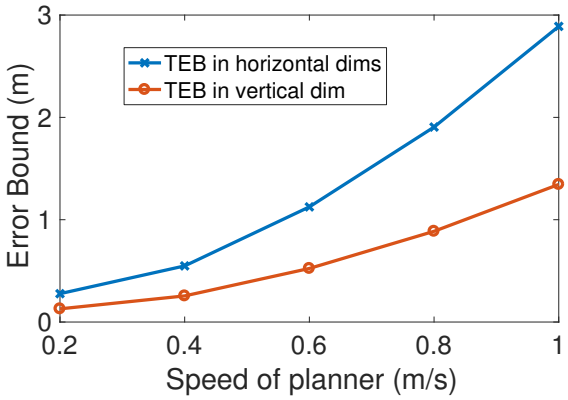


Figure 7.7: TEB vs. planner speed in each subsystem.

Equation (7.8) can be split into three 2D subsystems with states (x, v_x), (y, v_y), and (z, v_z) that are of the same form as the double-integrator example from Section 7.3. Note that the dynamics of the (x, v_x) and (y, v_y) subsystems are identical, and thus can be solved once and applied to each subsystem. By using decomposable HJ reachability [38] we compute the (x, v_x) set in 2 min 15 s and the (z, v_z) set in 2 min, for a total of a 4 min 15 s precomputation time. Fig. 7.7 shows the growth of $\mathcal{B}_p$ in each subsystem's position state as the planner speed in that dimension increases. Moreover, as explained in Section 7.3, the TEB for π_i is identical to the SSB for switching from $\pi_i \to \pi_j, j > i$.

Simulation

We implemented the meta-planning online algorithm within the robot operating system (ROS) [143] framework. We used the BIT* [77] geometric planner from the Open Motion Planning Library (OMPL) [162]. Code is written in C++ and is available as an open source ROS package.[6] Meta-planning typically runs in well under one second in a moderately cluttered environment.

Fig. 7.6 shows a snapshot of a simulated autonomous quadrotor flight in an artificial environment with spherical obstacles using trajectories generated by our algorithm. Initially,

[6]https://github.com/HJReachability/meta_fastrack

the obstacle locations and sizes are unknown to the quadrotor, but as soon as they come within the sensing radius (the size of which must adhere to the constraint discussed in Section 7.2) they are added to the meta-planner's internal environment model and used during re-planning.

In Fig. 7.6a we show what happens when the tracking controller is a standard LQR controller, while in Fig. 7.6b everything remains the same except that we apply the optimal controllers derived from the offline analysis in Section 7.3. Note that the LQR controller makes no guarantee about staying within the TEB, and hence it is unable to remain inside the TEB in the vicinity of the obstacle. The optimal controller, conversely, is guaranteed to remain in the TEB.

Hardware Demonstration

We replicated the simulation on a hardware testbed using the Crazyflie 2.0 open source quadrotor platform, shown in Fig. 7.8. We obtained position and orientation measurements at ~ 235 Hz from an OptiTrack infrared motion capture system. Given state estimates, we send control signals over a radio to the quadrotor at 100 Hz. As shown in our accompanying video,[7] the quadrotor successfully avoids the obstacles while remaining inside the TEB for each planner the meta-plan.

Fig. 7.9 shows the quadrotor's position over time recorded during a hardware demonstration. Note that the quadrotor stays well within the TEB throughout the flight even when the TEB changes size during planner switches.

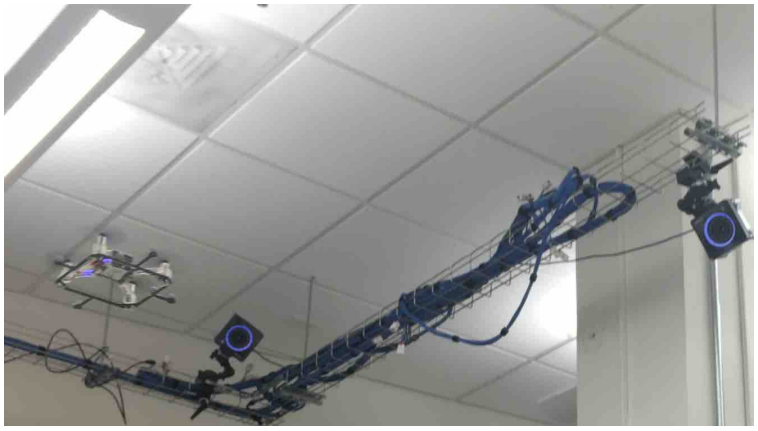


Figure 7.8: A Crazyflie 2.0 flying during our hardware demonstration. Two OptiTrack cameras are visible in the background.

[7] https://youtu.be/lPdXtR8Ar-E

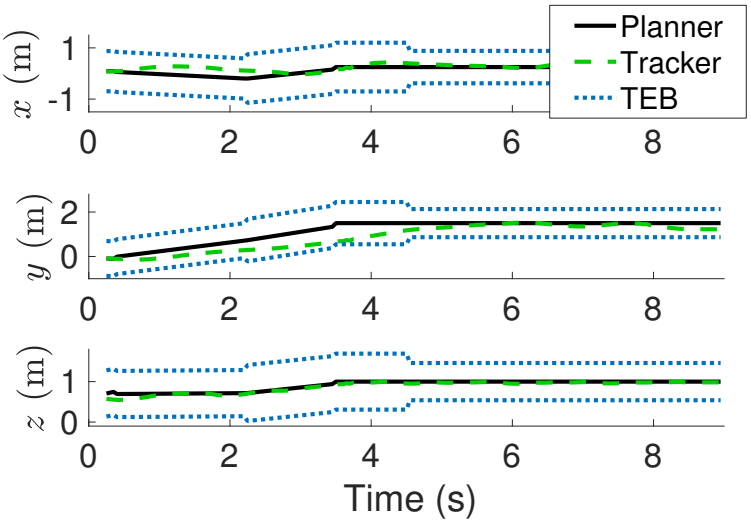


Figure 7.9: Position vs. time during a hardware demonstration.

7.5 Conclusions

We have proposed a novel meta-planning algorithm for using FaSTrack with multiple planners. The algorithm adaptively selects the fastest-moving planner that finds locally collision-free paths, and guarantees safe online transitions between these planners. The resulting meta-plans use more aggressive, faster-moving planners in open areas and more cautious, slower-moving planners near obstacles. We demonstrate meta-planning in simulation and in a hardware demonstration, using a quadrotor to track piecewise-linear trajectories at different top speeds.

The theory we develop here is general and can be applied to a wide variety of systems, including manipulators and other mobile robots. However, computing the TEB and SSB using HJ reachability can be challenging for these high-dimensional coupled systems. Ongoing work seeks to alleviate this challenge using other methods of computation such as sum of squares programming and neural network function approximators. Other promising directions include incorporating time-varying obstacle avoidance, further integration with OMPL and other planning libraries, providing adaptable error bounds based on external disturbances, and updating the tracking error bound online based on learned information about the tracking model.

Part III

Navigation in Multi-Agent Environments

Chapter 8

Single Robot, Single Human

This chapter is based on the papers "Probabilistically Safe Robot Planning with Confidence-Based Human Predictions" [72] and "Confidence-aware motion prediction for real-time collision avoidance" [75], written in collaboration with Jaime Fisac, Andrea Bajcsy, David Fridovich-Keil, Steven Wang, Claire J. Tomlin, and Anca D. Dragan.

Motion planning serves a key role in robotics, enabling robots to automatically compute trajectories that achieve the specified objectives while avoiding unwanted collisions. In many situations of practical interest, such as autonomous driving and UAV navigation, it is important that motion planning account not just for the current state of the environment, but also for its *predicted* future state. Often, certain objects in the environment may move in active, complex patterns that cannot be readily predicted using straightforward physics models; we shall refer to such complex moving objects as *agents*. Examples of agents considered in this chapter include pedestrians and human-driven vehicles. Predicting the future state of these agents is generally a difficult problem. Some of the key challenges include unclear and varying intents of other agents, mismatches between dynamics models and reality, incomplete sensor information, and interaction effects.

One popular approach to addressing the challenge of *a priori* unknown agent intent is to use rule-based or data-driven algorithms to predict *individual trajectories* for each agent, as in [155]. Alternatively, [180], [15], and [103] explicitly predict an agent's full state distribution over time; this representation may be better suited to capturing uncertainty in an agent's dynamics and the environment itself. [172] and [71] pose the prediction problem game-theoretically to model coupled human-robot interaction effects explicitly.

Unfortunately, a significant problem still remains: if an agent suddenly moves in a way that is not predicted, or not assigned sufficient probability, the robot may not react appropriately. For example, in Fig. 8.1 a pedestrian is walking around an obstacle which the robot, a quadcopter, cannot detect. To the robot, such behavior may be assigned very low probability, which could lead the robot to plan a dangerous trajectory. In this particular example, this inaccuracy caused the quadcopter to collide with the pedestrian (Fig. 8.1, left).

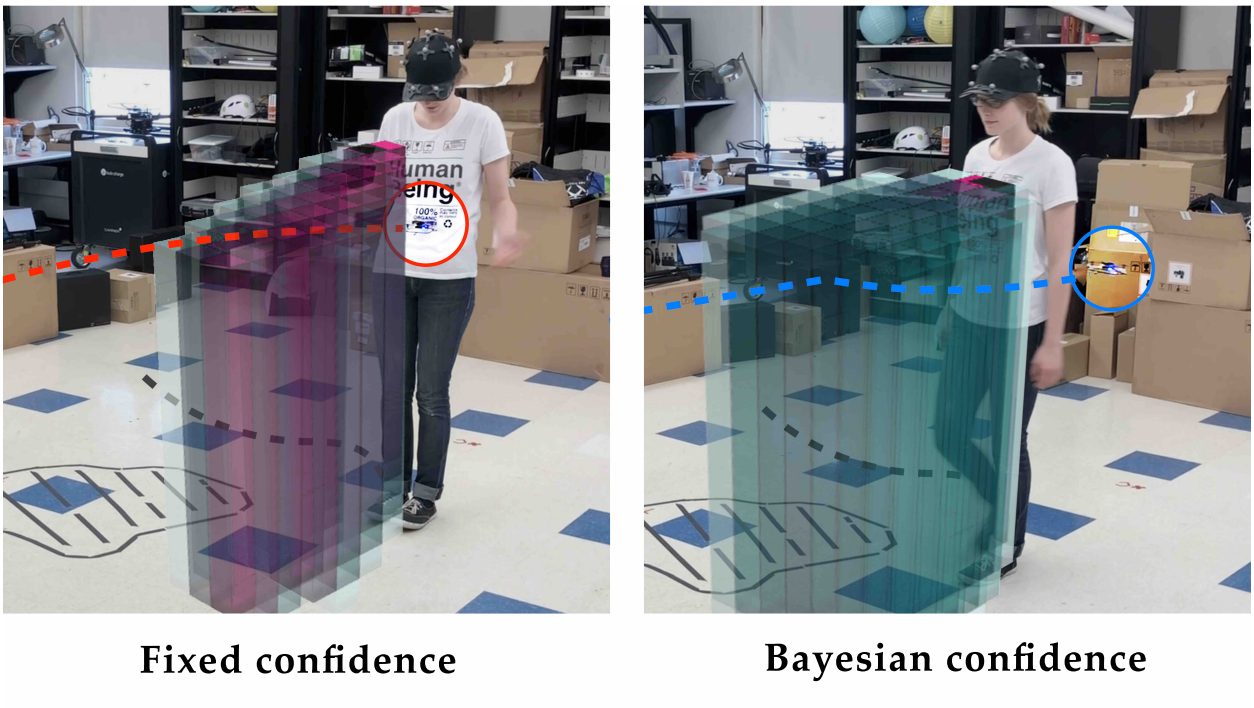


Figure 8.1: When planning around humans, accurate predictions of human motion (visualized here pink and blue, representing high and low probability respectively) are an essential prerequisite for safety. Unfortunately, these approaches may fail to explain all observed motion at runtime (e.g. human avoids unmodeled spill on the ground), leading to inaccurate predictions, and potentially, collisions (left). Our method addresses this by updating its *predictive model confidence* in real time (right), leading to more conservative motion planning in circumstances when predictions are known to be suspect.

To prepare for this eventuality, we introduce the idea of *confidence-aware prediction*. We argue that, in addition to predicting the future state of an agent, it is also crucial for a robot to assess the quality of the mechanism by which it is generating those predictions. That is, a robot should reason about how *confident* it is in its predictions of other agents before attempting to plan future motion. For computational efficiency, the quadcopter uses a simplified model of pedestrian dynamics and decision making. Thus equipped, it generates a time-varying probability distribution over the future state of the pedestrian, and plans trajectories to a pre-specified goal that maintain a low probability of collision. Fig. 8.1 (right) illustrates how this approach works in practice. The quadcopter maintains a Bayesian *belief* over its prediction confidence. As soon as the pedestrian moves in a way that was assigned low probability by the predictive model, the quadcopter adjusts its belief about the accuracy of that model. Consequently, it is less certain about what the pedestrian will do in the future. This leads the quadcopter's onboard motion planner, which attempts to find efficient trajectories with low probability of collision, to generate more cautious—and perhaps less

efficient—motion plans.

In order to improve the robustness of generated motion plans, we employ the recent FaSTrack framework from [88] for fast and safe motion planning and tracking. FaSTrack quantifies the maximum possible tracking error between a high-order dynamical model of the physical robot and the (potentially lower-order) dynamical model used by its motion planner. Solving an offline Hamilton-Jacobi reachability problem yields a guaranteed *tracking error bound* and the corresponding *safety controller*. These may be used by an out-of-the-box real-time motion planning algorithm to facilitate motion plans with strong runtime collision-avoidance guarantees.

The remainder of this chapter is organized as follows. Section 8.1 places this work in the context of existing literature in human motion modeling and prediction, as well as robust motion planning. Section 8.2 frames the prediction and planning problems more formally, and introduces a running example used throughout the chapter. Section 8.3 presents our main contribution: confidence-aware predictions. Section 8.5 showcases confidence-aware predictions in operation in several examples. Section 8.6 describes the application of the robust motion planning framework from FaSTrack to this setting, in which predictions are probabilistic. Section 8.7 explores a connection between our approach and reachability theory. Section 8.8 presents experimental results from a hardware demonstration. Finally, Section 8.9 concludes with a discussion of some of the limitations of our work and how they might be addressed in specific applications, as well as suggestions for future research.

8.1 Prior Work

Human Modeling and Prediction

One common approach for predicting human actions is to collect data from real-world scenarios and train a machine learning model via supervised learning. Such techniques use the human's current state, and potentially her prior state and action history, to predict future actions directly. [8], [56], [106], [114], and [85] demonstrate the effectiveness of this approach for inference and planning around human arm motion. Additionally, [85] focus on multi-step tasks like assembly, and [155] and [59] address the prediction problem for human drivers.

Rather than predicting actions directly, an alternative is for the robot to model the human as a rational agent seeking to maximize an unknown objective function. The human's actions up to a particular time may be viewed as Bayesian evidence from which the robot may infer the parameters of that objective. Assuming that the human seeks to maximize this objective in the future, the robot can predict her future movements, e.g. [135], [14], [180], and [11]. In this chapter, we build on this work by introducing a principled online technique for estimating confidence in such a learned model of human motion.

Safe Robot Motion Planning

Once armed with a predictive model of the human motion, the robot may leverage motion planning methods that plan around uncertain moving obstacles and generate real-time dynamically feasible and safe trajectories.

To avoid moving obstacles in real time, robots typically employ reactive and/or path-based methods. Reactive methods directly map sensor readings into control, with no memory involved, e.g. [22]. Path-based methods such as rapidly-exploring random trees from [96] and A* from [84] find simple kinematic paths through space and, if necessary, time. These path-based methods of planning are advantageous in terms of efficiency, yet, while they have in some cases been combined with probabilistically moving obstacles as in [9, 180], they do not consider the endogenous dynamics of the robot or exogenous disturbances such as wind. As a result, the robot may deviate from the planned path and potentially collide with obstacles. It is common for these plans to try to avoid obstacles by a heuristic margin of error. [88, 76] propose FaSTrack, a recent algorithm that provides a guaranteed tracking error margin and corresponding error-feedback controller for dynamic systems tracking a generic planner in the presence of bounded external disturbance. Our work builds upon FaSTrack to create an algorithm that can safely and dynamically navigate around uncertain moving obstacles in real time.

8.2 Problem Setup

We consider a single mobile robot operating in a shared space with a single human agent (e.g. a pedestrian or human-driven car). For simplicity, we presume that the robot has full knowledge of its own state and that of the human, although both would require online estimation in practice. As we present each formal component of this problem, we will provide a concrete illustration using a running example in which a quadcopter is navigating around a pedestrian.

Dynamical System Models and Safety

We will model the motion of both the human and the robot as the evolution of two dynamical systems. Let the state of the human be $x_H \in \mathbb{R}^{n_H}$, where n_H is the dimension of the human state space. We similarly define the robot's state, for planning purposes, as $x_R \in \mathbb{R}^{n_R}$. In general, these states could represent the positions and velocities of a mobile robot and a human in a shared environment, the kinematic configurations of a human and a robotic manipulator in a common workspace, or the positions, orientations, and velocities of human-driven and autonomous vehicles in an intersection. We express the evolution of these states over time as a family of ordinary differential equations:

$$\dot{x}_H = f_H(x_H, u_H) \ , \qquad \dot{x}_R \ = f_R(x_R, u_R) \tag{8.1}$$

where $u_H \in \mathbb{R}^{m_H}$ and $u_R \in \mathbb{R}^{m_R}$ are the control actions of the human and robot, respectively.

Example 8.1. *We introduce a running example for illustration purposes throughout the chapter. In this example we consider a small quadcopter that needs to fly to goal location $g_R \in \mathbb{R}^3$ in a room where a pedestrian is walking. For the purposes of planning, the quadcopter's 3D state is given by its position in space $x_R = [p_x, p_y, p_z]$, with velocity controls assumed decoupled in each spatial direction, up to $v_R = 0.25$ m/s. The human can only move by walking and therefore her state is given by planar coordinates $x_H = [h_x, h_y]$ evolving as $\dot{x}_H = [v_H \cos u_H, v_H \sin u_H]$. Intuitively, we model the human as moving with a fixed speed and controlling her heading angle. At any given time, the human is assumed to either move at a leisurely walking speed ($v_H \approx 1$ m/s) or remain still ($v_H \approx 0$).*

Ultimately, the robot needs to plan and execute an efficient trajectory to a pre-specified goal state (g_R), without colliding with the human. We define the keep-out set $\mathcal{K} \subset \mathbb{R}^{n_H} \times \mathbb{R}^{n_R}$ as the set of joint robot-human states to be avoided (for example, because they imply physical collisions). To avoid reaching this set, the robot must reason about the human's future motion when constructing its own motion plan.

Example 8.2. *In our quadcopter-avoiding-pedestrian example, $\mathcal{K}$ consists of joint robot-human states in which the quadcopter is flying within a square of side length $l = 0.3$ m centered around the human's location, while at any altitude, as well as any joint states in which the robot is outside the environment bounds defined as a box with a square base of side $L = 3.66$ m and height $H = 2$ m, regardless of the human's state.*

Robust Robot Control

Provided an objective and a dynamics model, the robot must generate a motion plan which avoids the keep-out set $\mathcal{K}$. Unfortunately, this safety requirement is difficult to meet during operation for two main reasons:

1. *Model mismatch.* The dynamical system model f_R will never be a perfect representation of the real robot. This mismatch could lead to unintended collision.

2. *Disturbances.* Even with a perfect dynamics model, there may be unobserved, external "disturbance" inputs such as wind or friction. Without accounting for these disturbances, the system is not guaranteed to avoid $\mathcal{K}$, even if the planned trajectory is pointwise collision-free.

To account for modelling error and external disturbances, we could in principle design a higher fidelity dynamical model directly in a robust motion planning framework. Unfortunately, however, real-time trajectory optimization in high dimensions can be computationally

burdensome, particularly when we also require some notion of robustness to external disturbance. Ideally we would like to enjoy the computational benefits of a planning with a lower-fidelity model while enforcing the safety constraints induced by the higher-fidelity model. To characterize this model mismatch, we consider a higher fidelity and typically higher-order dynamical representation of the robot, with state representation $s_R \in \mathbb{R}^{n_S}$. This dynamical model will also explicitly account for external disturbances as unknown bounded inputs, distinct from control inputs. In order to map between this higher fidelity "tracking" state s_R and the lower fidelity "planning" state x_R, we shall assume a known projection operator $\pi : \mathbb{R}^{n_S} \to \mathbb{R}^{n_R}$. Fortunately, we can plan in the lower-dimensional state space at runtime, and guarantee robust collision avoidance via an offline reachability analysis that quantifies the effects of model mismatch and external disturbance. This framework, called FaSTrack and first proposed by [88], is described in further detail in Section 8.6.

Example 8.3. *We model our quadcopter with the following flight dynamics (in the near-hover regime, at zero yaw with respect to a global coordinate frame):*

$$\begin{bmatrix} \dot{p}_x \\ \dot{p}_y \\ \dot{p}_z \end{bmatrix} = \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} \quad , \quad \begin{bmatrix} \dot{v}_x \\ \dot{v}_y \\ \dot{v}_z \end{bmatrix} = \begin{bmatrix} a_g \tan u_\theta \\ -a_g \tan u_\phi \\ u_T - a_g \end{bmatrix} \quad , \tag{8.2}$$

where $[p_x, p_y, p_z]$ is the quadcopter's position in space and $[v_x, v_y, v_z]$ is its velocity expressed in the fixed global frame. We model its control inputs as thrust acceleration u_T and attitude angles (roll u_ϕ and pitch u_θ), and denote the acceleration due to gravity as a_g. The quadcopter's motion planner generates nominal kinematic trajectories in the lower-dimensional $[p_x, p_y, p_z]$ position state space. Therefore we have a linear projection map $\pi(s_R) = [I_3, 0_3]s_R$, that is, x_R retains the position variables in s_R and discards the velocities.

Predictive Human Model

In order to predict the human's future motion, the robot uses its internal model of human dynamics, f_H. Under this modeling assumption, the human's future trajectory depends upon her choice of control input over time, $u_H(\cdot)$. Extensive work in econometrics and cognitive science, such as [171, 119, 14], has shown that human behavior—that is, u_H—can be well modeled by utility-driven optimization. Thus, the robot models the human as optimizing a reward function, $r_H(x_H, u_H; \theta)$, that depends on the human's state and action, as well as a set of parameters θ. This reward function could be a linear combination of features as in many inverse optimal control implementations (where the goal or feature weighting θ must be learned, either online or offline), or more generally learned through function approximators such as deep neural networks, where θ are the trained weights as in [68].

We assume that the robot has a suitable human reward function r_H, either learned offline from prior human demonstrations or otherwise encoded by the system designers. Thus

endowed with r_H, the robot can model the human's choice of control action as a probability distribution over actions conditioned on state. Under maximum-entropy assumptions [179] inspired by noisy-rationality decision-making models [14], the robot models the human as more likely to choose (discrete) actions u_H with high expected utility, in this case the state-action value (or Q-value):

$$P(u_H \mid x_H; \beta, \theta) = \frac{e^{\beta Q_H(x_H, u_H; \theta)}}{\sum_{\tilde{u}} e^{\beta Q_H(x_H, \tilde{u}; \theta)}} \ . \tag{8.3}$$

We use a temporally- and spatially-discretized version of human dynamics, $\tilde{f}_H$. These discrete-time dynamics may be found by integrating f_H over a fixed time step of Δt with fixed control u_H over the interval. Section 8.5 provides further details on this discretization.

Example 8.4. *The quadcopter's model of the human assumes that she intends to reach some target location $g_H \in \mathbb{R}^2$ in a straight line. The human's reward function is given by the distance traveled over time step Δt, i.e. $r_H(x_H, u_H; g_H) = -v_H \Delta t$, and human trajectories are constrained to terminate at g_H. The state-action value, parameterized by $\theta = g_H$, captures the optimal cost of reaching g_H from x_H when initially applying u_H for a duration Δt: $Q_H(x_H, u_H; g_H) = -v_H \Delta t - \|x_H + v_H \Delta t [\cos u_H, \sin u_H]^\top - g_H\|_2$.*

Often, the coefficient β is termed the *rationality coefficient*, since it quantifies the degree to which the robot expects the human's choice of control to align with its model of utility. For example, taking $\beta \downarrow 0$ yields a model of a human who appears "irrational," choosing actions uniformly at random and completely ignoring the modeled utility. At the other extreme, taking $\beta \uparrow \infty$ corresponds to a "perfectly rational" human, whose actions exactly optimize the modeled reward function. As we will see in Section 8.3, β can also be viewed as a measure of the robot's *confidence* in the predictive accuracy of Q_H.

Note that $Q_H(x_H, u_H; \theta)$ only depends on the human state and action and not on the robot's. Thus far, we have intentionally neglected discussion of human-robot interaction effects. These effects are notoriously difficult to model, and the community has devoted a significant effort to building and validating a variety of models, e.g. [167], [152]. In that spirit, we could have chosen to model human actions u_H as dependent upon robot state x_R in (8.3), and likewise defined Q_H to depend upon x_R. This extended formulation is sufficiently general as to encompass all possible (Markov) interaction models. However, in this work we explicitly do not model these interactions; indeed, one of the most important virtues of our approach is its robustness to precisely these sorts of modeling errors.

Probabilistically Safe Motion Planning

Ideally, the robot's motion planner should generate trajectories that reach a desired goal state efficiently, while maintaining safety. More specifically, in this context "safety" indicates that the physical system will never enter the keep-out set $\mathcal{K}$ during operation, despite human

motion and external disturbances. That is, we would like to guarantee that $(\pi(s_R), x_H) \notin \mathcal{K}$ for all time.

To make this type of strong, deterministic, *a priori* safety guarantee requires the robot to avoid the set of all human states x_H which could possibly be occupied at a particular time, i.e. the human's *forward reachable set*. If the robot can find trajectories that are safe for *any* possible human trajectory then there is no need to predict the human's next action. Unfortunately, the forward reachable set of the human often encompasses such a large volume of the workspace that it is impossible for the robot to find a guaranteed safe trajectory to the goal state. This motivates refining our notion of prediction: rather than reasoning about all the places where the human *could* be, the robot can instead reason about *how likely* the human is to be at each location. This probabilistic reasoning provides a guide for planning robot trajectories with a quantitative degree of safety assurance.

Our probabilistic model of human control input (8.3) coupled with dynamics model f_H allows us to compute a probability distribution over human states for every future time. By relaxing our conception of safety to consider only collisions which might occur with sufficient probability P_{th}, we dramatically reduce the effective volume of this set of future states to avoid. In practice, P_{th} should be chosen carefully by a system designer in order to trade off overall collision probability with conservativeness in motion planning.

The proposed approach in this chapter follows two central steps to provide a quantifiable, high-confidence collision avoidance guarantee for the robot's motion around the human. In Section 8.3 we present our proposed Bayesian framework for reasoning about the uncertainty inherent in a model's prediction of human behavior. Based on this inference, we demonstrate how to generate a real-time probabilistic prediction of the human's motion over time. Next, in Section 8.6 we extend a state-of-the-art, provably safe, real-time robotic motion planner to incorporate our time-varying probabilistic human prediction.

8.3 Confidence-Aware Human Motion Prediction

Any approach to human motion prediction short of computing a full forward reachable set must, explicitly or implicitly, reflect a model of human decision-making. In this work, we make that model explicit by assuming that the human chooses control actions in a Markovian fashion according to the probability distribution (8.3). Other work in the literature, such as [155], aims to learn a generative probabilistic model for human trajectories; implicitly, this training procedure distills a model of human decision making. Whether explicit or implicit, these models are by nature imperfect and liable to make inaccurate predictions eventually. One benefit of using an explicit model of human decision making, such as (8.3), is that we may reason directly and succinctly about its performance online.

In particular, the entropy of the human control distribution in (8.3) is a decreasing function of the parameter β. High values of β place more probability mass on high-utility control actions u_H, whereas low values of β spread the probability mass more evenly between different control inputs, regardless of their modeled utility Q_H. Therefore, β naturally quantifies

how well the human's motion is expected to agree with the notion of optimality encoded in Q_H. The commonly used term "rationality coefficient", however, seems to imply that discrepancies between the two indicate a failure on the human's part to make the "correct" decisions, as encoded by the modeled utility. Instead, we argue that these inevitable disagreements are primarily a result of the model's inability to fully capture the human's behavior. Thus, instead of conceiving of β as a rationality measure, we believe that β can be given a more pragmatic interpretation related to the accuracy with which the robot's model of the human is able to explain her motion. Consistently, in this chapter, we refer to β as *model confidence*.

An important related observation following from this interpretation of β is that the predictive accuracy of a human model is likely to change over time. For example, the human may change her mind unexpectedly, or react suddenly to some aspect of the environment that the robot is unaware of. Therefore, we shall model β as an unobserved, time-varying parameter. Estimating it in real-time provides us with a direct, quantitative summary of the degree to which the utility model Q_H explains the human's current motion. To do this, we maintain a Bayesian *belief* about the possible values of β. Initially, we begin with a uniform prior over β and over time this distribution evolves given measurements of the human's state and actions.

Real-time Inference of Model Confidence

We reason about the model confidence β as a hidden state in a hidden Markov model (HMM) framework. The robot starts with a prior belief b_-^0 over the initial value of β. In this work, we use a uniform prior, although that is not strictly necessary. At each discrete time step $k \in \{0, 1, 2, \dots\}$, it will have some belief about model confidence $b_-^k(\beta)$. [1] After observing a human action u_H^k, the robot will update its belief to b_+^k by applying Bayes' rule.

The hidden state may evolve between subsequent time steps, accounting for the important fact that the predictive accuracy of the human model may change over time as unmodeled factors in the human's behavior become more or less relevant. Since by definition we do not have access to a model of these factors, we use a naive "ϵ-static" transition model: at each time k, β may, with some probability ϵ, be re-sampled from the initial distribution b_-^0, and otherwise retains its previous value. We define the belief over the next value of β (denoted by β') as an expectation of the conditional probability $P(\beta' \mid \beta)$, i.e. $b_-^k(\beta') := \mathbb{E}_{\beta \sim b_+^{k-1}}[P(\beta' \mid \beta)]$. Concretely, this expectation may be computed as

$$b_-^k(\beta') = (1 - \epsilon) b_+^{k-1}(\beta') + \epsilon b_-^0(\beta') \ . \tag{8.4}$$

By measuring the evolution of the human's state x_H over time, we assume that, at every time step k, the robot is able to observe the human's control input u_H^k. This observed control

[1] To avoid confusion between discrete and continuous time, we shall use superscripts to denote discrete time steps (e.g. x_H^k) and parentheticals to denote continuous time (e.g. $x_H(t)$).

may be used as evidence to update the robot's belief b_-^k about β over time via a Bayesian update:

$$b_+^k(\beta) = \frac{P(u_H^k \mid x_H^k; \beta, \theta) b_-^k(\beta)}{\sum_{\tilde{\beta}} P(u_H^k \mid x_H^k; \tilde{\beta}, \theta) b_-^k(\tilde{\beta})} \ , \tag{8.5}$$

with $b_+^k(\beta) := P(\beta \mid x_H^{0:k}, u_H^{0:singlehuman_k})$ for $k \in \{0, 1, ...\}$, and $P(u_H^k \mid x_H^k; \beta, \theta)$ given by (8.3).

It is critical to be able to perform this update rapidly to facilitate real-time operation; this would be difficult in the original continuous hypothesis space $\beta \in [0, \infty)$, or even in a large discrete set. Fortunately, our software examples in Section 8.5 and hardware demonstration in Section 8.8 suggest that maintaining a Bayesian belief over a relatively small set of $N_\beta = 5$ discrete values of β distributed on a log scale achieves significant improvement relative to using a fixed value.

The "ϵ-static" transition model leads to the desirable consequence that old observations of the human's actions have a smaller influence on the current model confidence distribution than recent observations. In fact, if no new observations are made, successively applying time updates asymptotically contracts the belief towards the initial distribution, that is, $b_-^k(\cdot) \to b_-^0(\cdot)$. The choice of parameter ϵ effectively controls the rate of this contraction, with higher ϵ leading to more rapid contraction.

Human motion prediction

Equipped with a belief over β at time step k, we are now able to propagate the human's state distribution forward to any future time via the well-known Kolmogorov forward equations, recursively. In particular, suppose that we know the probability that the human is in each state x_H^κ at some future time step κ. We know that (according to our utility model) the probability of the human choosing control u_H^κ if she is in state x_H^κ is given by (8.3). Accounting for the otherwise deterministic dynamics model $\tilde{f}_H$, we obtain the following expression for the human's state distribution at the following time step $\kappa + 1$:

$$\begin{aligned} P(x_H^{\kappa+1}; \beta, \theta) = \sum_{x_H^\kappa, u_H^\kappa} & P(x_H^{\kappa+1} \mid x_H^\kappa, u_H^\kappa; \beta, \theta) \cdot \\ & P(u_H^\kappa \mid x_H^\kappa; \beta, \theta) P(x_H^\kappa; \beta, \theta) \ , \end{aligned} \tag{8.6}$$

for a particular choice of β. Marginalizing over β according to our belief at the current step time k, we obtain the overall occupancy probability distribution at each future time step κ:

$$P(x_H^\kappa; \theta) = \mathbb{E}_{\beta \sim b^k} P(x_H^\kappa; \beta, \theta) \ . \tag{8.7}$$

Note that (8.6) is expressed more generally than is strictly required. Indeed, because the only randomness in dynamics model $\tilde{f}_H$ originates from the human's choice of control input u_H, we have $P(x_H^{\kappa+1} \mid x_H^\kappa, u_H^\kappa; \beta, \theta) = \mathbb{1}\{x_H^{\kappa+1} = \tilde{f}_H(x_H^\kappa, u_H^\kappa)\}$.

8.4 Model Confidence with Auxiliary Parameter Identification

Thus far, we have tacitly assumed that the only unknown parameter in the human utility model (8.3) is the model confidence, β. However, often one or more of the auxiliary parameters θ are also unknown. These auxiliary parameters could encode one or more human goal states or intents, or other characteristics of the human's utility, such as her preference for avoiding parts of the environment. Further, much like model confidence, they may change over time.

In principle, it is possible to maintain a Bayesian belief over β and θ jointly. The Bayesian update for the hidden state (β, θ) is then given by

$$b_+^k(\beta,\theta) = \frac{P(u_H^k \mid x_H^k; \beta, \theta) b_-^k(\beta,\theta)}{\sum_{\tilde{\beta},\tilde{\theta}} P(u_H^k \mid x_H^k; \tilde{\beta}, \tilde{\theta}) b_-^k(\tilde{\beta},\tilde{\theta})} \ , \tag{8.8}$$

with $b_+^k(\beta,\theta) := P(\beta, \theta \mid x_H^{0:k}, u_H^{0:k})$ the running posterior and
$b_-^k(\beta,\theta) := P(\beta, \theta \mid x_H^{0:k-1}, u_H^{0:k-1})$ the prior at time step k.

This approach can be practical for parameters taking finitely many values from a small, discrete set, e.g. possible distinct modes for a human driver (distracted, cautious, aggressive). However, for certain scenarios or approaches it may not be practical to maintain a full Bayesian belief on the parameters θ. In such cases, it is reasonable to replace the belief over θ with a point estimate $\bar{\theta}$, such as the maximum likelihood estimator or the mean, and substitute that estimate into (8.6). Depending on the complexity of the resulting maximum likelihood estimation problem, it may or may not be computationally feasible to update the parameter estimate $\bar{\theta}$ at each time step. Fortunately, even when it is computationally expensive to estimate $\bar{\theta}$, we can leverage our model confidence as an indicator of when re-estimating these parameters may be most useful. That is, when model confidence degrades that may indicate poor estimates of $\bar{\theta}$.

8.5 Prediction Examples

We illustrate these inference steps with two sets of examples: our running pedestrian example and a simple model of a car.

Pedestrian model (running example)

So far, we have presented a running example of a quadcopter avoiding a human. We use a deliberately simple, purely kinematic model of continuous-time human motion:

$$\dot{x}_H = \begin{bmatrix} \dot{h}_x \\ \dot{h}_y \end{bmatrix} = \begin{bmatrix} v_H \cos u_H \\ v_H \sin u_H \end{bmatrix} \ . \tag{8.9}$$

However, as discussed in Section 8.2, the proposed prediction method operates in discrete time (and space). The discrete dynamics corresponding to (8.9) are given by

$$\begin{aligned} x_H^{k+1} - x_H^k &\equiv x_H(t+\Delta t) - x_H(t) \\ &= \begin{bmatrix} v_H \Delta t \cos u_H(t) \\ v_H \Delta t \sin u_H(t) \end{bmatrix}, \end{aligned} \tag{8.10}$$

for a time discretization of Δt.

Dubins car model

To emphasize the generality of our method, we present similar results for a different application domain: autonomous driving. We will model a human-driven vehicle as a dynamical system whose state x_H evolves as

$$\dot{x}_H = \begin{bmatrix} \dot{h}_x \\ \dot{h}_y \\ \dot{h}_\phi \end{bmatrix} = \begin{bmatrix} v_H \cos h_\phi \\ v_H \sin h_\phi \\ u_H \end{bmatrix}. \tag{8.11}$$

Observe that, while (8.11) appears very similar to (8.9), in this Dubins car example the angle of motion is a *state*, not a *control input*.

We discretize these dynamics by integrating (8.11) from t to $t+\Delta t$, assuming a constant control input u_H:

$$\begin{aligned} x_H^{k+1} - x_H^k &\equiv x_H(t+\Delta t) - x_H(t) = \\ &\begin{bmatrix} \frac{v_H}{u_H(t)} \left(\sin\left(h_\phi(t) + u_H(t)\Delta t\right) - \sin(h_\phi(t)) \right) \\ -\frac{v_H}{u_H(t)} \left(\cos\left(h_\phi(t) + u_H(t)\Delta t\right) - \cos(h_\phi(t)) \right) \\ u_H \Delta t \end{bmatrix} \end{aligned} \tag{8.12}$$

For a specific goal position $g = [g_x, g_y]$, the Q-value corresponding to state-action pair (x_H, u_H) and reward function $r_H(x_H, u_H) = -v_H \Delta t$ (until the goal is reached) may be found by solving a shortest path problem offline.

Accurate Model

First, we consider a scenario in which the robot has full knowledge of the human's goal, and the human moves along the shortest path from a start location to this known goal state. Thus, human motion is well-explained by Q_H.

The first row of Fig. 8.2 illustrates the probability distributions our method predicts for the pedestrian's future state at different times. Initially, the predictions generated by our Bayesian confidence-inference approach (right) appear similar to those generated by the low model confidence predictor (left). However, our method rapidly discovers that Q_H is an

accurate description of the pedestrian's motion and generates predictions that match the high model confidence predictor (center). The data used in this example was collected by tracking the motion of a real person walking in a motion capture arena. See Section 8.8 for further details.

Likewise, the first row of Fig. 8.3 shows similar results for a human-driven Dubins car model (in simulation) at an intersection. Here, traffic laws provide a strong prior on the human's potential goal states. As shown, our method of Bayesian model confidence inference quickly infers the correct goal and learns that the human driver is acting in accordance with its model Q_H. The resulting predictions are substantially similar to the high-β predictor. The data used in this example was simulated by controlling a Dubins car model along a pre-specified trajectory.

Unmodeled Obstacle

Often, robots do not have fully specified models of the environment. Here, we showcase the resilience of our approach to unmodeled obstacles which the human must avoid. In this scenario, the human has the same start and goal as in the accurate model case, except that there is an obstacle along the way. The robot is unaware of this obstacle, however, which means that in its vicinity the human's motion is not well-explained by Q_H, and $b(\beta)$ ought to place more probability mass on higher values of β.

The second rows of Fig. 8.2 and Fig. 8.3 illustrate this type of situation for the pedestrian and Dubins car, respectively. In Fig. 8.2, the pedestrian walks to an *a priori* known goal location and avoids an unmodeled spill on the ground. Analogously, in Fig. 8.3 the car swerves to avoid a large pothole. By inferring model confidence online, our approach generates higher-variance predictions of future state, but only in the vicinity of these unmodeled obstacles. At other times throughout the episode when Q_H is more accurate, our approach produces predictions more in line with the high model confidence predictor.

Unmodeled Goal

In most realistic human-robot encounters, even if the robot does have an accurate environment map and observes all obstacles, it is unlikely for it to be aware of all human goals. We test our approach's resilience to unknown human goals by constructing a scenario in which the human moves between both known and unknown goals.

The third row of Fig. 8.2 illustrates this situation for the pedestrian example. Here, the pedestrian first moves to one known goal position, then to another, and finally back to the start which was not a modelled goal location. The first two legs of this trajectory are consistent with the robot's model of goal-oriented motion, though accurate prediction does require the predictor to infer *which* goal the pedestrian is walking toward. However, when the pedestrian returns to the start, her motion appears inconsistent with Q_H, skewing the robot's belief over β toward zero.

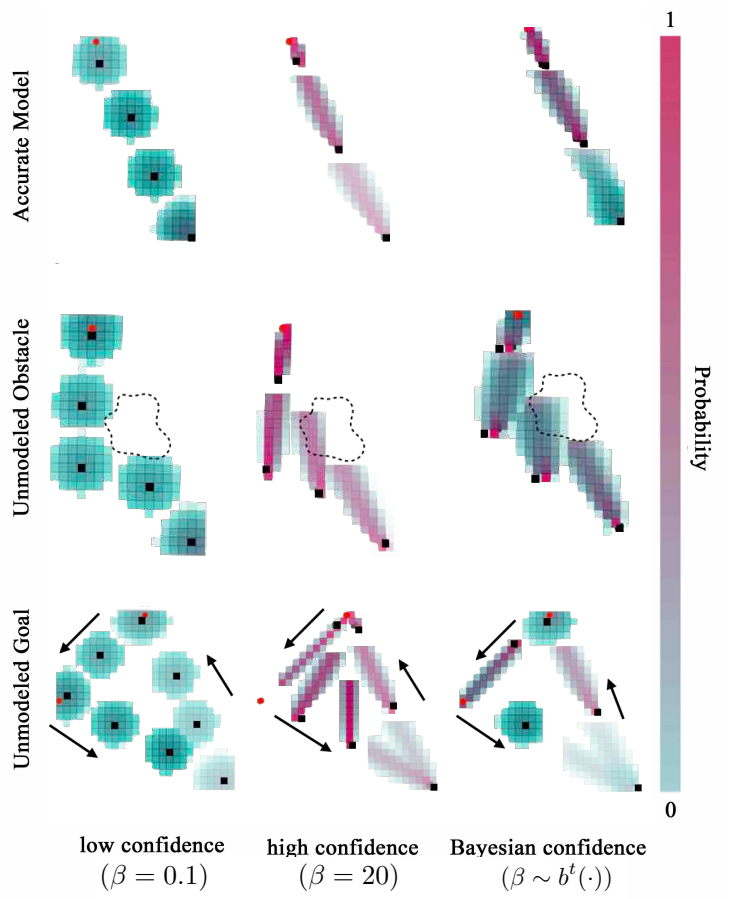


Figure 8.2: Snapshots of pedestrian trajectory and probabilistic model predictions. Top row: Pedestrian moves from the bottom right to a goal marked as a red circle. Middle row: Pedestrian changes course to avoid a spill on the floor. Bottom row: Pedestrian moves to one known goal, then to another, then to a third which the robot has not modeled. The first two columns show predictions for low and high model confidence; the third column shows the predictions using our Bayesian model confidence. For all pedestrian videos, see: `https://youtu.be/lh_E9rW-MJo`

Similarly, in the third row of Fig. 8.3 we consider a situation in which a car makes an unexpected turn onto an unmapped access road. As soon as the driver initiates the turn, our predictor rapidly learns to distrust its internal model Q_H and shift its belief over β upward.

8.6 Safe Probabilistic Planning and Tracking

Given probabilistic predictions of the human's future motion, the robot must plan efficient trajectories which avoid collision with high probability. In order to reason robustly about this probability of future collision, we must account for potential tracking errors incurred

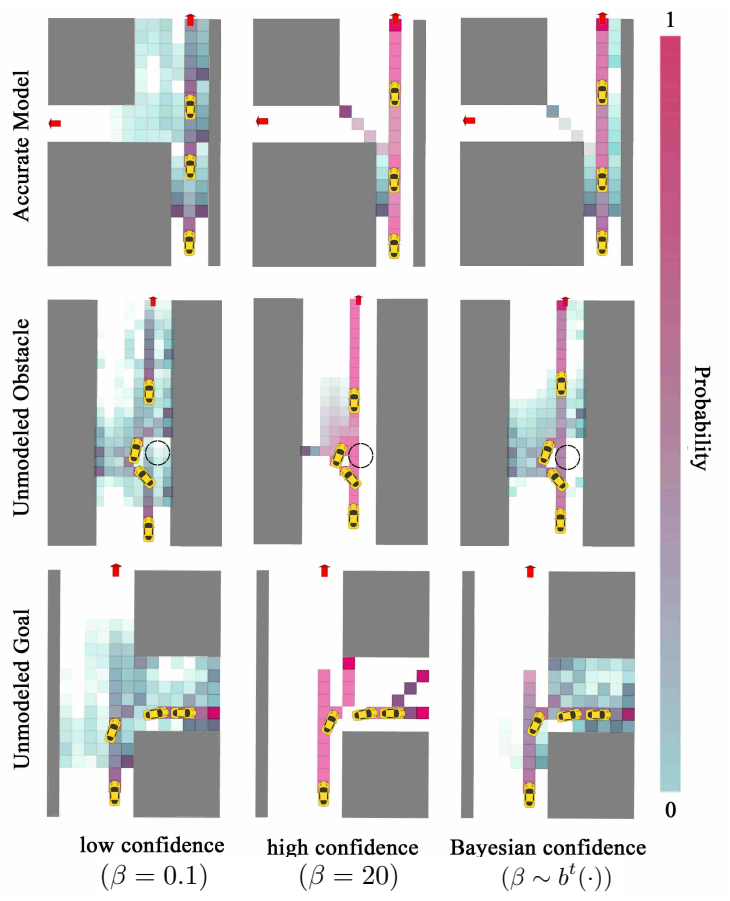


Figure 8.3: Snapshots of Dubins car and probabilistic predictions. Top row: Car moves straight ahead toward a known goal (red arrow), staying in its lane. Middle row: Car suddenly swerves to the left to avoid a pothole. Bottom row: Car turns to the right, away from the only known goal. The left and center columns show results for low and high confidence predictors, respectively, and the right column shows our approach using Bayesian inferred model confidence. For all Dubins car videos, see: `https://youtu.be/sAJKNnP42fQ`

by the real system as it follows planned trajectories. To this end, we build on the recent FaSTrack framework of [88], which provides control-theoretic robust safety certificates in the presence of deterministic obstacles, and extend it to achieve approximate probabilistic collision-avoidance.

Background: Fast Planning, Safe Tracking

Recall that x_R is the robot's state for the purposes of motion planning, and that s_R encodes a higher-fidelity, potentially higher-dimensional notion of state (with associated dynamics). The recently proposed FaSTrack framework from [88] uses Hamilton-Jacobi reachability anal-

ysis to quantify the worst-case tracking performance of the s_R-system as it follows trajectories generated by the x_R-system. For further reading on reachability analysis refer to [62], [126], and [17]. A byproduct of this FaSTrack analysis is an error feedback controller that the s_R system can use to achieve this worst-case tracking error. The tracking error bound may be given to one of many off-the-shelf real-time motion planning algorithms operating in x_R-space in order to guarantee real-time collision-avoidance by the s_R-system.

Formally, FaSTrack precomputes an optimal tracking controller, as well as a corresponding compact set $\mathcal{E}$ in the robot's planning state space, such that $\big(\pi(s_R(t)) - x_{R,\text{ref}}(t)\big) \in \mathcal{E}$ for *any* reference trajectory proposed by the lower-fidelity planner. This bound $\mathcal{E}$ is a trajectory tracking certificate that can be passed to an online planning algorithm for real-time safety verification: the dynamical robot is guaranteed to *always* be somewhere within the bound relative to the current planned reference point $x_{R,\text{ref}}(t)$. This tracking error bound may sometimes be expressed analytically; otherwise, it may be computed numerically using level set methods, e.g. [129]. Equipped with $\mathcal{E}$, the planner can generate safe plans by ensuring that the entire bound around the nominal state remains collision-free throughout the trajectory. Note that the planner only needs to know $\mathcal{E}$ and otherwise requires no explicit understanding of the high-fidelity model.

Example 8.5. *Since dynamics* (8.2) *are decoupled in the three spatial directions, the bound* $\mathcal{E}$ *computed by FaSTrack is an axis-aligned box of dimensions* $\mathcal{E}_x \times \mathcal{E}_y \times \mathcal{E}_z$*. For further details refer to [76].*

Robust Tracking, Probabilistic Safety

Unfortunately, planning algorithms for collision checking against deterministic obstacles cannot be readily applied to our problem. Instead, a trajectory's collision check should return the probability that it *might* lead to a collision. Based on this probability, the planning algorithm can discriminate between trajectories that are *sufficiently safe* and those that are not.

As discussed in Section 8.2, a safe online motion planner invoked at time t should continually check the probability that, at any future time τ, $(\pi(s_R(\tau)), x_H(\tau)) \in \mathcal{K}$. The tracking error bound guarantee from FaSTrack allows us to conduct worst-case analysis on collisions given a human state x_H. Concretely, if no point in the Minkowski sum $\{x_R + \mathcal{E}\}$ is in the collision set with x_H, we can guarantee that the robot is not in collision with the human.

The probability of a collision event for any point $x_R(\tau)$ along a candidate trajectory, assuming worst-case tracking error bound $\mathcal{E}$, can be upper-bounded by the total probability that $x_H(\tau)$ will be in collision with *any* of the possible robot states $\tilde{x}_R \in \{x_R(\tau) + \mathcal{E}\}$. For each robot planning state $x_R \in \mathbb{R}^{n_R}$ we define the set of human states in potential collision

with the robot:

$$\mathcal{H}_{\mathcal{E}}(x_R) := \{\tilde{x}_H \in \mathbb{R}^{n_H} : \exists \tilde{x}_R \in \{x_R + \mathcal{E}\}, (\tilde{x}_R, \tilde{x}_H) \in \mathcal{K}\} \ . \tag{8.13}$$

Example 8.6. *Given $\mathcal{K}$ and $\mathcal{E}$, $\mathcal{H}_{\mathcal{E}}(x_R)$ is the set of human positions within the rectangle of dimensions $(l + \mathcal{E}_x) \times (l + \mathcal{E}_y)$ centered on $[p_x, p_y]$. A human anywhere in this rectangle could be in collision with the quadcopter.*

The following result follows directly from the definition of the tracking error bound and a union bound.

Proposition 4. *The probability of a robot with worst case tracking error $\mathcal{E}$ colliding with the human at any trajectory point $x_R(\tau)$ is bounded above by the probability mass of $x_H(\tau)$ contained within $\mathcal{H}_{\mathcal{E}}(x_R(\tau))$.*

We shall consider *discrete-time* motion plans. The probability of collision along any such trajectory from current time step k to final step $k + K$ is upper-bounded by:

$$P_{\text{coll}}^{k:k+K} \leq \overline{P_{\text{coll}}^{k:k+K}} := 1 - \prod_{\kappa=k}^{k+K} P\Big(x_H^{\kappa} \notin \mathcal{H}_{\mathcal{E}}(x_R^{\kappa}) \mid x_H^{\kappa} \notin \mathcal{H}_{\mathcal{E}}(x_R^{s}), k \leq s < \kappa\Big) \ . \tag{8.14}$$

Evaluating the right hand side of (8.14) exactly requires reasoning about the joint distribution of human states over all time steps and its conditional relationship on whether collision has yet occurred. This is equivalent to maintaining a probability distribution over the exponentially large space of trajectories $x_H^{k:k+K}$ that the human might follow. As motion planning occurs in real-time, we shall resort to a heuristic approximation of (8.14).

One approach to approximating (8.14) is to assume that the event $x_H^{\kappa_1} \notin \mathcal{H}_{\mathcal{E}}(x_R^{\kappa_1})$ is independent of $x_H^{\kappa_2} \notin \mathcal{H}_{\mathcal{E}}(x_R^{\kappa_2}), \forall \kappa_1 \neq \kappa_2$. This independence assumption is equivalent to removing the conditioning in (8.14). Unfortunately, this approximation is excessively pessimistic; if there is no collision at time step κ, then collision is also unlikely at time step $\kappa + 1$ because both human and robot trajectories are continuous.

We shall refine this approximation by finding a tight lower bound on the right hand side of (8.14). Because collision events are correlated in time, we first consider replacing each conditional probability $P\big(x_H^{\kappa} \notin \mathcal{H}_{\mathcal{E}}(x_R^{\kappa}) \mid x_H^{s} \notin \mathcal{H}_{\mathcal{E}}(x_R^{s}), k \leq s < \kappa\big)$ by 1 for all $t > 0$. This effectively lower bounds $\overline{P_{\text{coll}}^{k:k+K}}$ by the worst case probability of collision at the current time step k:

$$\begin{aligned} \overline{P_{\text{coll}}^{k:k+K}} &\geq 1 - P\big(x_H^{k} \notin \mathcal{H}_{\mathcal{E}}(x_R^{k})\big) \\ &= P\big(x_H^{k} \in \mathcal{H}_{\mathcal{E}}(x_R^{k})\big) \ . \end{aligned} \tag{8.15}$$

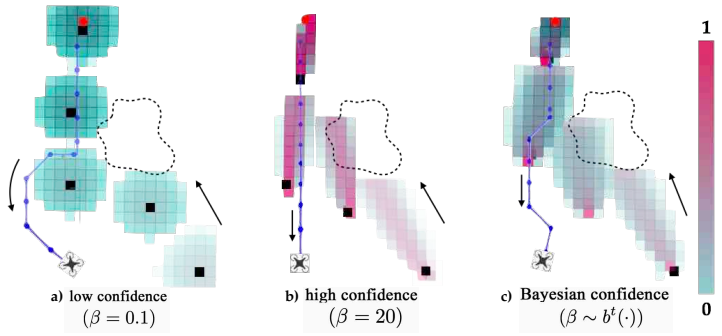


Figure 8.4: Scenario from the middle row of Fig. 8.2 visualized with robot's trajectory. When β is low and the robot is not confident, it makes large deviations from its path to accommodate the human. When β is high, the robot refuses to change course and comes dangerously close to the human. With inferred model confidence, the robot balances safety and efficiency with a slight deviation around the human.

This bound is extremely loose in general, because it completely ignores the possibility of future collision. However, note that probabilities in the product in (8.14) may be conditioned in any particular order (not necessarily chronological). This commutativity allows us to generate $K-k+1$ lower bounds of the form $\overline{P_{\text{coll}}^{k:k+K}} \geq P\big(x_H^\kappa \in \mathcal{H}_\mathcal{E}(x_R^\kappa)\big)$ for $\kappa \in \{k, \dots, k+K\}$.Taking the tightest of all of these bounds, we can obtain an informative, yet quickly computable, approximator for the sought probability:

$$P_{\text{coll}}^{k:k+K} \approx \max_{\kappa \in \{k:k+K\}} P\big(x_H^\kappa \in \mathcal{H}_\mathcal{E}(x_R^\kappa)\big) \ . \tag{8.16}$$

To summarize, (8.16) approximates the probability of collision over an entire trajectory with the highest marginal collision probability at *any point* in the trajectory. While this approximation will err on the side of optimism, we note that the robot's ability to continually replan as updated human predictions become available mitigates this potentially underestimated risk.

Safe Online Planning under Uncertain Human Predictions

This approximation of collision probability allows the robot to discriminate between valid and invalid candidate trajectories during motion planning. Using the prediction methodology proposed in Section 8.3, we may quickly generate, at every time t, the marginal probabilities in (8.16) at each future time $\kappa \in \{k, \dots, k+K\}$, based on past observations at times $0, \dots, k$. The planner then computes the instantaneous probability of collision $P\big(x_H^\kappa \in \mathcal{H}_\mathcal{E}(x_R^\kappa)\big)$ by integrating $P\big(x_H^\tau \mid x_H^{0:k}\big)$ over $\mathcal{H}_\mathcal{E}(x_R^\kappa)$, and rejects the candidate point x_R^κ if this probability exceeds P_{th}.

Note that for graph-based planners that consider candidate trajectories by generating a graph of time-stamped states, rejecting a candidate edge from this graph is equivalent to

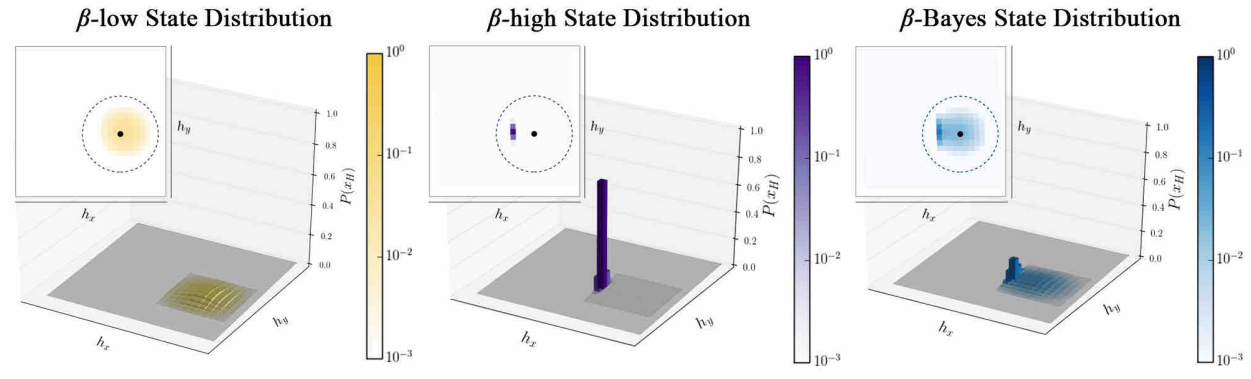


Figure 8.5: The human (black dot) is moving west towards a goal. Visualized are the predicted state distributions for one second into the future when using low, high, and Bayesian model confidence. Higher-saturation indicates higher likelihood of occupancy. The dashed circle represents the pedestrian's 1 second forward reachable set.

rejecting all further trajectories that would contain that edge. This early rejection rule is consistent with the proposed approximation (8.16) of $P_{\text{coll}}^{k:k+K}$ while preventing unnecessary exploration of candidate trajectories that would ultimately be deemed unsafe.

Throughout operation, the robot follows each planned trajectory using the error feedback controller provided by FaSTrack, which ensures that the robot's high-fidelity state representation s_R and the lower-fidelity state used for planning, x_R, differ by no more than the tracking error bound $\mathcal{E}$. This planning and tracking procedure continues until the robot reaches its desired goal state.

Example 8.7. *Our quadcopter is now required to navigate to a target position shown in Fig. 8.4 without colliding with the human. Our proposed algorithm successfully avoids collisions at all times, replanning to leave greater separation from the human whenever her motion departs from the model. In contrast, robot planning with fixed model confidence is either overly conservative at the expense of time and performance or overly aggressive at the expense of safety.*

8.7 Connections to Reachability Analysis

In this section, we present an alternative, complementary analysis of the overall safety properties of the proposed approach to prediction and motion planning. This discussion is grounded in the language of reachability theory and worst-case analysis of human motion.

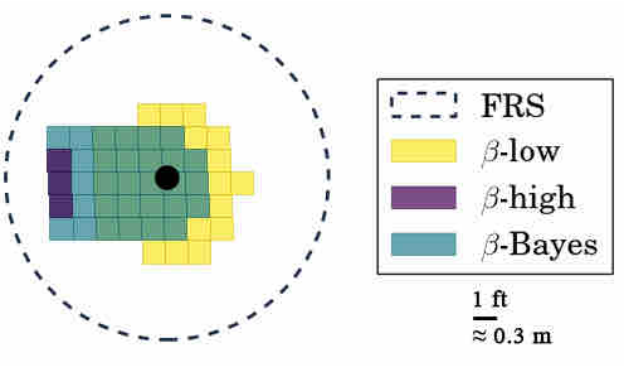


Figure 8.6: Visualization of the states with probability greater than or equal to the collision threshold, $P_{\text{th}} = 0.01$. The human's forward reachable set includes the set of states assigned probability greater than P_{th}. We show these "high probability" predicted states for predictors with fixed low and high β, as well as our Bayesian-inferred β.

Forward Reachable Set

Throughout this section, we frequently refer to the human's time-indexed forward reachable set. We define this set formally below.

Definition 6. *(Forward Reachable Set) For a dynamical system $\dot{x} = f(x, u)$ with state trajectories given by the function $\xi\big(x(0), t, u(\cdot)\big) =: x(t)$, the forward reachable set $\mathcal{V}_{FRS}(x, t)$ of a state x after time t has elapsed is*

$$\mathcal{V}_{FRS}(x, t) := \{x' : \exists u(\cdot), x' = \xi(x, t, u(\cdot))\} \ .$$

That is, a state x' is in the forward reachable set of x after time t if it is *reachable* via some applied control signal $u(\cdot)$.

Remark 12. *(Recovery of $\mathcal{V}_{FRS}$) For $P_{th} = 0$ and any finite β, the set of states assigned probability greater than P_{th} is identical to the forward reachable set, up to discretization errors. This is visualized for low, high, and Bayesian model confidence in Fig. 8.5.*

A Sufficient Condition for the Safety of Individual Trajectories

In Section 8.6, we construct an approximation to the probability of collision along a trajectory, which we use during motion planning to avoid potentially dangerous states. To make this guarantee of collision-avoidance for a motion plan even stronger, it would suffice to ensure that the robot never comes too close to the human's forward reachable set. More precisely, a planned trajectory is safe if $\{x_R(t) + \mathcal{E}\} \cap \mathcal{V}_{\text{FRS}}(x_H, t) = \emptyset$, for every state $x_R(t)$ along a motion plan generated when the human was at state x_H. The proof of this statement follows directly from the properties of the tracking error bound $\mathcal{E}$ described in Section 8.6.

While this condition may seem appealing, it is in fact highly restrictive. The requirement of avoiding the full forward reachable set is not always possible in confined spaces; indeed, this was our original motivation for wanting to predict human motion (see Section 8.2). However, despite this shortcoming, the logic behind this sufficient condition for safety provides insight into the effectiveness of our framework.

Recovering the Forward Reachable Set

Though it will not constitute a formal safety guarantee, we analyze the empirical safety properties of our approach by examining how our predicted state distributions over time relate to forward reachable sets. During operation, our belief over model confidence β evolves to match the degree to which the utility model Q_H explains recent human motion. The "time constant" governing the speed of this evolution may be tuned by the system designer to be arbitrarily fast by choosing the parameter ϵ to be small, as discussed in Section 8.3. Thus, we may safely assume that $b(\beta)$ places high probability mass on small values of β as soon as the robot observes human motion which is not well explained by Q_H.

Fig. 8.6 shows the sets of states with "high enough" ($> P_{\text{th}}$) predicted probability mass overlaid on the human's forward reachable set at time t, which is a circle of radius $v_H t$ centered on x_H for the dynamics in our running example. When β is high (10), we observe that virtually all of the probability mass is concentrated in a small number of states in the direction of motion predicted by our utility model. When β is low (0.05) we observe that the set of states assigned probability above our collision threshold P_{th} occupies a much larger fraction of the reachable set. A typical belief $b(\beta)$ recorded at a moment when the human was roughly moving according to Q_H yields an intermediate set of states.

Fig. 8.7 illustrates the evolution of these sets of states over time, for the unmodeled obstacle example of Section 8.5 in which a pedestrian avoids a spill. Each row corresponds to the predicted state distribution at a particular point in time. Within a row, each column shows the reachable set and the set of states assigned occupancy probability greater than $P_{\text{th}} = 0.01$. The color of each set of states corresponds to the value of β used by the low confidence and high confidence predictors, and the *maximum a posteriori* value of β for the Bayesian confidence predictor. The human's known goal state is marked by a red dot.

Interestingly, as the Bayesian model confidence decreases—which occurs when the pedestrian turns to avoid the spill at $t \approx 6$ s—the predicted state distribution assigns high probability to a relatively large set of states, but unlike the low-β predictor that set of states is oriented toward the known goal. Of course, had $b(\beta)$ placed even more probability mass on lower values of β then the Bayesian confidence predictor would converge to the low confidence one.

Additionally, we observe that, within each row as the prediction horizon increases, the area contained within the forward reachable set increases and the fraction of that area contained within the predicted sets decreases. This phenomenon is a direct consequence of our choice of threshold P_{th}. Had we chosen a smaller threshold value, a larger fraction of the forward reachable set would have been occupied by the lower-β predictors.

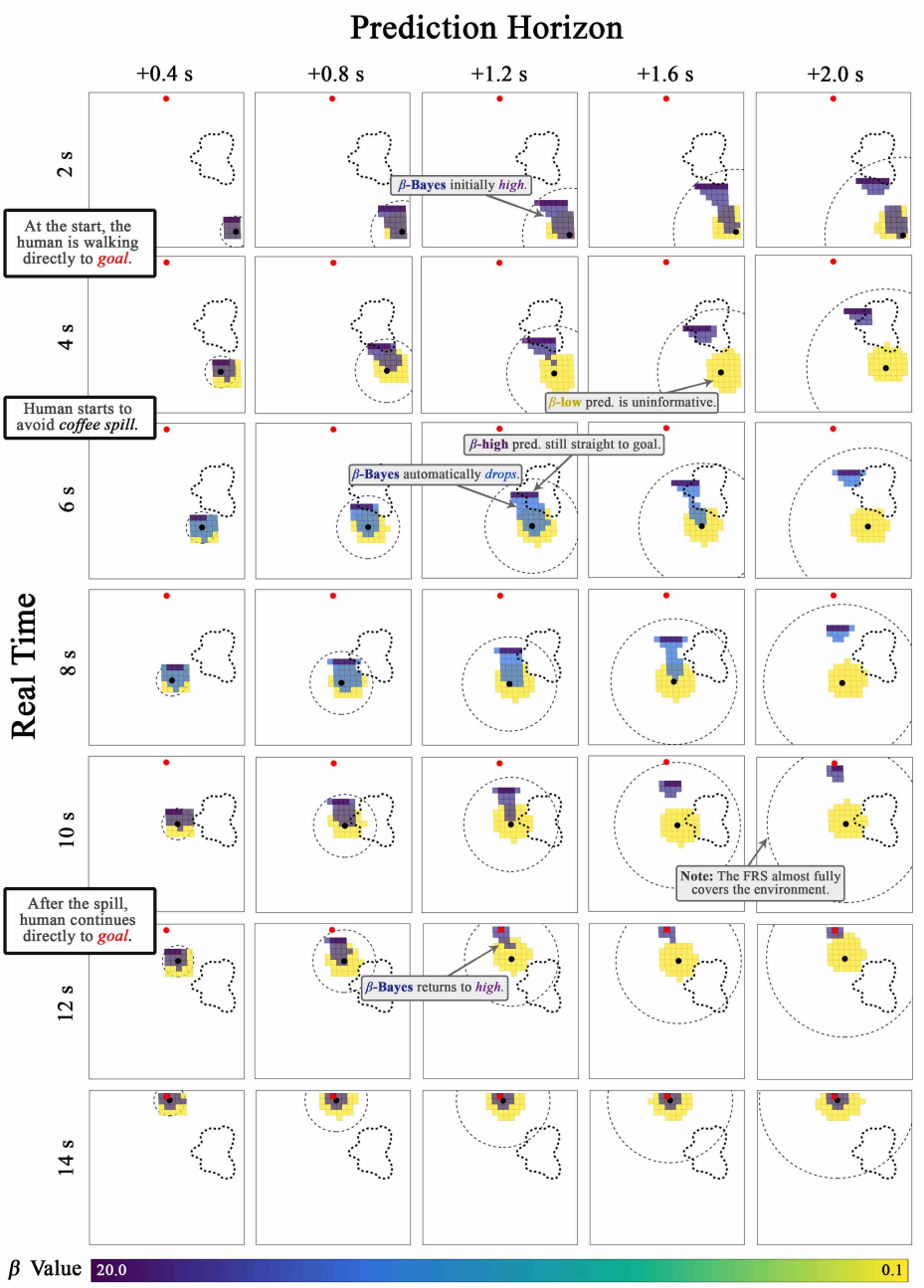


Figure 8.7: The human (black dot) is walking towards the known goal (red dot) but has to avoid an unmodeled coffee spill on the ground. Here we show the snapshots of the predictions at various future times (columns) as the human walks around in real time (rows). The visualized states have probability greater than or equal to $P_{\text{th}} = 0.01$. Each panel displays the human prediction under low confidence (in yellow), high confidence (in dark purple), and Bayesian confidence (colored as per the most likely β value), as well as the forward reachable set.

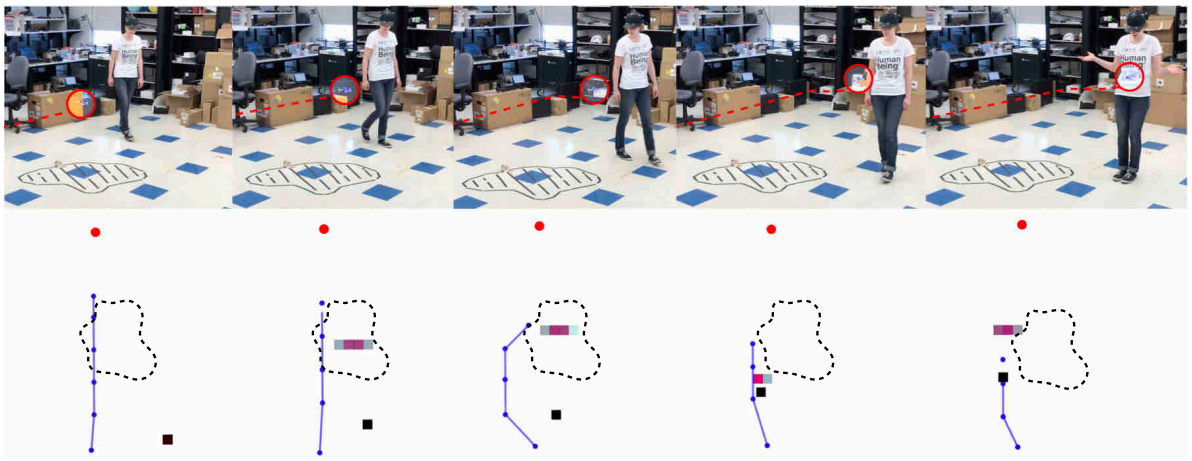


Figure 8.8: Predicting with fixed-β (in this case, $\beta = 20$) can yield highly inaccurate predictions (and worse, confidently inaccurate ones). The subsequent motion plans may not be safe; here, poor prediction quality leads to a collision.

This observation may be viewed prescriptively. Recalling the sufficient condition for safety of planned trajectories from Section 8.7, if the robot replans every T_{replan} seconds, we may interpret the fraction of $\mathcal{V}_{\text{FRS}}(\cdot, t + T_{\text{replan}})$ assigned occupancy probability greater than P_{th} by the low confidence predictor as a rough indicator of the safety of an individual motion plan, robust to worst-case human movement. As this fraction tends toward unity, the robot is more and more likely to be safe. However, for any $P_{\text{th}} > 0$, this fraction approaches zero for $T_{\text{replan}} \uparrow \infty$. This immediately suggests that, if we wish to replan every T_{replan} seconds, we can achieve a particular level of safety as measured by this fraction by choosing an appropriate threshold P_{th}.

In summary, confidence-aware predictions rapidly place high probability mass on low values of β whenever human motion is not well-explained by utility model Q_H. Whenever this happens, the resulting predictions encompass a larger fraction of the forward reachable set, and in the limit that $P_{\text{th}} \downarrow 0$ we recover the forward reachable set exactly. The larger this fraction, the more closely our approach satisfies the sufficient condition for safety presented in Section 8.7.

8.8 Hardware Demonstration

We implemented confidence-aware human motion prediction (Section 8.3) and integrated it into a real-time, safe probabilistic motion planner (Section 8.6), all within the Robot Operating System (ROS) software framework of [143]. To demonstrate the efficacy of our methods, we tested our work for the quadcopter-avoiding-pedestrian example used for illustration throughout this chapter. Human trajectories were recorded as (x, y) positions on the ground plane at roughly 235 Hz by an OptiTrack infrared motion capture system, and we

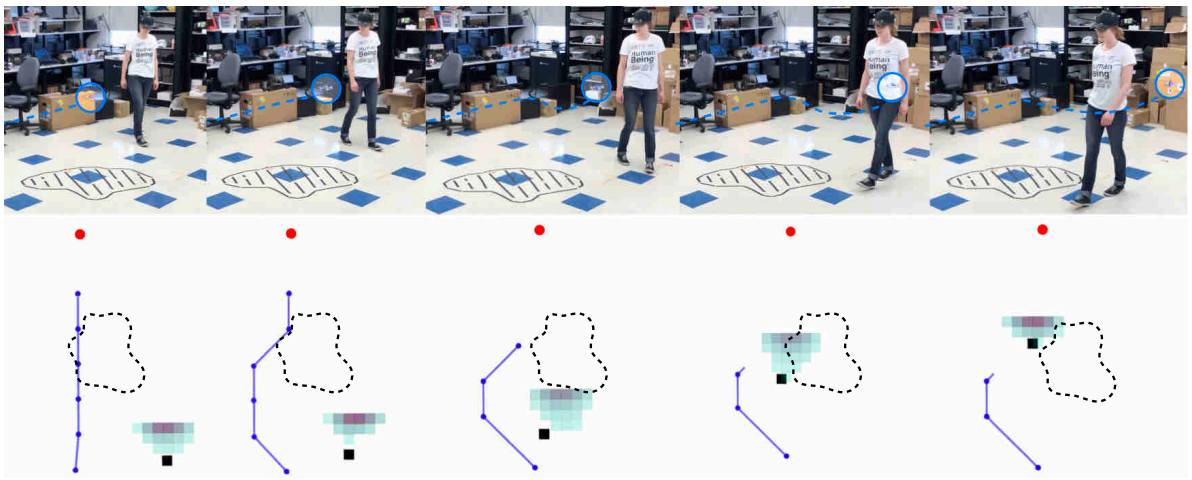


Figure 8.9: Inferring β leads to predicted state distributions whose entropy increases whenever the utility model Q_H fails to explain observed human motion. The resulting predictions are more robust to modeling errors, resulting in safer motion plans. Here, the quadcopter successfully avoids the pedestrian even when she turns unexpectedly.

used a Crazyflie 2.0 micro-quadcopter, also tracked by the OptiTrack system.[2]

Fig. 8.4 illustrates the unmodeled obstacle case from Section 8.5, in which the pedestrian turns to avoid a spill on the ground. Using a low model confidence results in motion plans that suddenly and excessively deviate from the ideal straight-line path when the pedestrian turns to avoid the spill. By contrast, the high confidence predictor consistently predicts that the pedestrian will walk in a straight line to her goal even when she turns; this almost leads to collision, as shown in detail in Fig. 8.8. Our proposed approach for Bayesian model confidence initially assigns high confidence and predicts that the pedestrian will walk straight to the goal, but when she turns to avoid the spill, the predictions become less confident. This causes the quadcopter to make a minor course correction, shown in further detail in Fig 8.9.

8.9 Chapter Summary

When robots operate in complex environments in concert with other agents, safety often depends upon the robot's ability to predict the agents' future actions. While this prediction problem may be tractable in some cases, it can be extremely difficult for agents like people who act with *intent*. In this chapter, we introduce the idea of *confidence-aware* prediction as a natural coping mechanism for predicting the future actions of intent-driven agents. Our approach uses each measurement of the human's state to reason about the accuracy of its internal model of human decision making. This reasoning about model confidence is

[2]We note that in a more realistic setting, we would require alternative methods for state estimation using other sensors, such as lidar and/or camera(s). A video recording may be found at `https://youtu.be/2ZRGxWknENg`.

expressed compactly as a Bayesian filter over the possible values of a single parameter, β, which controls the entropy of the robot's model of the human's choice of action. In effect, whenever the human's motion is not well-explained by this model, the robot predicts that the human could occupy a larger volume of the state space.

We couple this notion of confidence-aware prediction with a reachability-based robust motion planning algorithm, FaSTrack, which quantifies the robot's ability to track a planned reference trajectory. Using this maximum tracking error allows us to bound an approximation of the probability of collision along planned trajectories. Additionally, we present a deeper connection between confidence-aware prediction and forward reachable sets, which provides an alternative explanation of the safety of our approach. We demonstrate the proposed methodology on a ROS-based quadcopter testbed in a motion capture arena.

There are several important limitations of this work, which we summarize and discuss below.

Limitations: State Discretization

As presented, our approach to prediction requires a discrete representation of the human's state space. This can be tractable for the relatively simple dynamical models of human motion we consider in this work. Fortunately, one of the strongest attributes of confidence-aware prediction is that it affords a certain degree of robustness to modeling errors by design. Still, our approach is effectively limited to low-order dynamical models.

Limitations: FaSTrack Complexity

FaSTrack provides a strong safety guarantee vis-à-vis the maximum tracking error that could ever exist between a higher-fidelity dynamical model of the robot and a lower-order one used for motion planning. Unfortunately, the computational complexity of finding this maximum tracking error and the corresponding safety controller scales exponentially with the dimension of the high-fidelity model. In some cases, these dynamics are decomposable and analytic solutions exist, e.g. [76, 37], and in other cases conservative approximations may be effective, e.g. [35, 150].

Limitations: Boltzmann Distributional Assumption

We model the human's choice of control input at each time step as an independent, random draw from a Boltzmann distribution (8.3). This distributional assumption is motivated from the literature in cognitive science and econometrics and is increasingly common in robotics, yet it may not be accurate in all cases. Maintaining an up-to-date model confidence belief $b(\beta)$ can certainly mitigate this inaccuracy, but only at the cost of making excessively conservative predictions.

Limitations: Safety Certification

Our analysis in Section 8.7 makes connections to forward reachability in an effort to understand the safety properties of our system. As shown, whenever our confidence-aware prediction method detects poor model performance it quickly yields predictions that approximate the human's forward reachable set. Although this approximation is not perfect, and hence we cannot provide a strong safety certificate, the connection to reachability is in some sense prescriptive. That is, it can be used to guide the choice of collision probability threshold P_{th} and replanning frequency. However, even if we could provide a strong guarantee of collision-avoidance for a *particular* motion plan, that would not, in general, guarantee that future motion plans would be *recursively safe*. This recursive property is much more general and, unsurprisingly, more difficult to satisfy.

Future Directions

Future work will aim to address each of these shortcomings. We are also interested in extending our methodology for the multi-robot, multi-human setting; our preliminary results are reported in [13]. Additionally, we believe that our model confidence inference approach could be integrated with other commonly-used probabilistic prediction methods besides the Boltzmann utility model. Finally, we are excited to test our work in hardware in other application spaces, such as manipulation and driving.

Chapter 9

Multi-Robot, Multi-Human

This chapter is based on the paper "A Scalable Framework for Real-Time Multi-Robot, Multi-Human Collision Avoidance" [13], written in collaboration with Andrea Bajcsy, David Fridovich-Keil, Jaime Fisac, Sampada Deglurkar, Anca D. Dragan, and Claire J. Tomlin.

As robotic systems are increasingly used for applications such as drone delivery services, semi-automated warehouses, and autonomous cars, safe and efficient robotic navigation around humans is crucial. Consider the example in Fig. 9.1, inspired by a drone delivery scenario, where two quadcopters must plan a safe trajectory around two humans who are walking through the environment. We would like to guarantee that the robots will reach their goals without ever colliding with each other, any of the humans, or the static surroundings.[1]

This safe motion planning problem faces three main challenges: (1) controlling the nonlinear robot dynamics subject to external disturbances (e.g. wind), (2) planning around multiple humans in real time, and (3) avoiding conflicts with other robots' plans. Extensive prior work from control theory, motion planning, and cognitive science has led to the development of computational tools for rigorous safety analysis, faster motion planners for nonlinear systems, and predictive models of human agents. Individually, these problems are difficult—computing robust control policies, coupled robot plans, and joint predictions of multiple human agents are all computationally demanding at best and intractable at worst [126, 37, 108]. Recent work, however, has made progress in provably-safe real-time motion planning [88, 122, 161, 107], real-time probabilistic prediction of a human agent's motion [72, 180], and robust sequential trajectory planning for multi-robot systems [18, 36]. It remains a challenge to synthesize these into a real-time planning system, primarily due to the difficulty of joint planning and prediction for multiple robots and humans. There has been some work combining subsets of this problem [101, 167, 16, 109, 113, 5], but the full setting of real-time and robust multi-robot navigation around multiple humans remains underexplored.

[1] We use a motion capture system for state estimation—robustness with respect to sensor uncertainty is important but beyond the scope of this chapter.

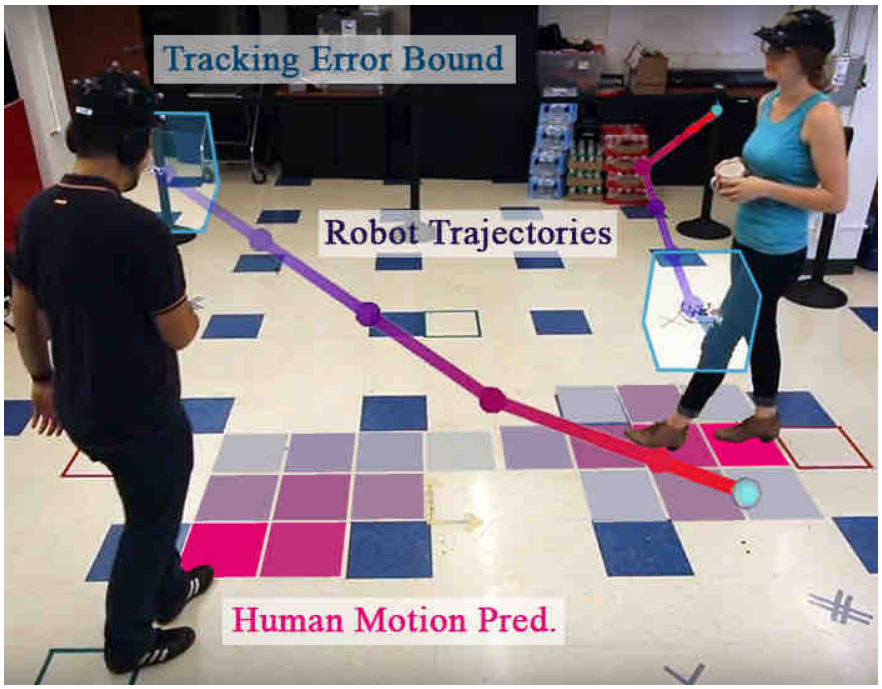


Figure 9.1: Hardware demonstration of real-time multi-agent planning while maintaining safety with respect to internal dynamics, external disturbances, and humans. The quadcopter's trajectories are visualized, and the tracking error bound is shown as a box around each quadcopter. The predictions of future human motion is shown in pink in front of each human.

Our main contributions are tractable approaches to joint planning and prediction, while still ensuring efficient, probabilistically-safe motion planning. Concretely, we:

1. achieve online prediction for multiple humans by using simplified models of human behavior that do not account for interaction effects. When people do interact with each other, our framework monitors confidence in the predictive model and automatically increases prediction variance when our assumptions are wrong,

2. generate safe and real-time multi-robot trajectories by planning sequentially and using a robust reachability-based controller with tracking guarantees, and

3. demonstrate our robust multi-human, multi-robot framework in hardware.

9.1 The Framework

Fig. 9.2 illustrates our overall framework. We introduce the components of the framework by incrementally addressing the three main challenges identified above. We first present the

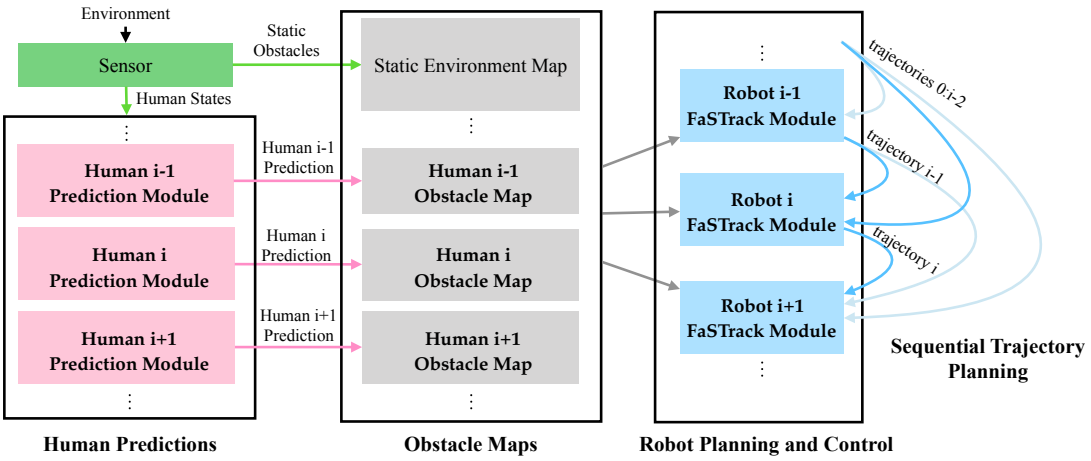


Figure 9.2: Our framework begins with sensor measurements of the environment state that is used to predict future human motion and construct an environment map. Our framework provides each robot with a trajectory and a set of controls that are robust to disturbances, other robots, and unmodeled human behavior.

robot planning and control module (Section 9.2), which is instantiated for each robot. Each robot uses a robust controller (e.g. the reachability-based controller of [88]) to track motion plans within a precomputed error margin that accounts for modeled dynamics and external disturbances. In order to generate safe motion plans, each robot will ensure that output trajectories are collision-checked with a set of obstacle maps, using the tracking error margin.

These obstacle maps include an *a priori* known set of static obstacles, as well as predictions of the future motion of any humans, which are generated by the **human predictions** module (Section 9.3). By generating these predictions, each robot is able to remain probabilistically safe with respect to the humans. To ensure tractability for multiple humans, we generate predictions using simplified interaction models. We subsequently adapt the predictions following a real-time Bayesian approach based on [72]. We leverage the property that individual predictions automatically become more uncertain whenever their accuracy degrades, and use this to enable our tractable predictions to be robust to unmodeled interaction effects.

Finally, to guarantee safety with respect to other robots, we carry out **sequential trajectory planning** (Section 9.4) by adapting the cooperative multi-agent planning scheme [18] to function in real time with the robust trajectories from the planning and control module. The robots generate plans according to a pre-specified priority ordering. Each robot plans to avoid the most recently generated trajectories from robots of higher priority, i.e. robot i must generate a plan that is safe with respect to the planned trajectories from robots $j, j < i$. This removes the computational complexity of planning in the joint state space of all robots at once.

9.2 Robot Planning and Control

To build a safe multi-robot navigation framework, each robot must be able to quickly generate and communicate the time-varying set of states they may occupy as they navigate to their goal. This tube must account for any possible deviation from planned trajectories that may occur due to environmental disturbances such as wind. To generate these plans we will first consider the canonical problem of planning through a static environment. Efficient algorithms for purely kinematic planning such as A* or rapidly-exploring random trees (RRT) [84, 96] excel at this task.

We now introduce robot dynamics and allow the environment to have external disturbances such as wind. Kinematic planners such as A* and RRT do not consider these factors when creating plans. In practice, however, these planners are often used to generate an initial trajectory, which may then be smoothed and tracked using a feedback controller such as a linear quadratic regulator (LQR). During execution, the mismatch between the planning model and the physical system can result in tracking error, which may cause the robot to deviate far enough from its plan to collide with an obstacle. To reduce the chance of collision, one can augment the robot by a heuristic error buffer; this induces a "safety bubble" around the robot used when collision checking. However, heuristically generating this buffer will not guarantee safety.

Several recent approaches address efficient planning while considering model dynamics and maintaining robustness with respect to external disturbances. These methods include robust motion primitives [122], model-predictive control [144] utilizes robust model-predictive control, parameterized reachable sets [107], and contraction theory [160].

In this chapter, we use FaSTrack [88, 76], a modular framework that computes a tracking error bound (TEB) via *offline* reachability analysis. This TEB can be thought of as a rigorous counterpart of the error-buffer concept introduced above. More concretely, the TEB is the set of states capturing the maximum relative distance (i.e. maximum tracking error) that may occur between the physical robot and the current state of the planned trajectory. We compute the TEB by formulating the tracking task as a pursuit-evasion game between the planning algorithm and the physical robot. We then solve this differential game using Hamilton-Jacobi reachability analysis. To ensure robustness, we assume (a) worst-case behavior of the planning algorithm (i.e. being as difficult as possible to track), and (b) that the robot is experiencing worst-case, bounded external disturbances. The computation of the TEB also provides a corresponding error-feedback controller for the robot to always remain inside the TEB. Thus, FaSTrack wraps efficient motion planners, and adds robustness to modeled system dynamics and external disturbances through the precomputed TEB and error-feedback controller. Fig. 9.4 shows a top-down view of a quadcopter using a kinematic planner (A*) to navigate around static obstacles. By employing the error-feedback controller, the quadcopter is guaranteed to remain within the TEB (shown in blue) as it traverses the A* path.

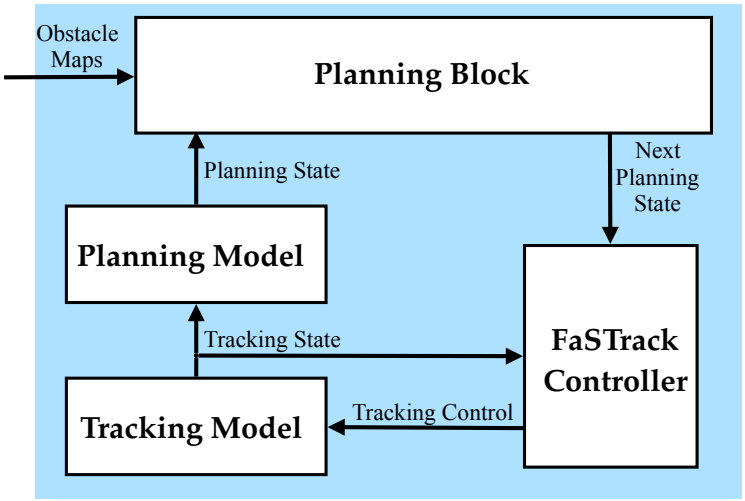


Figure 9.3: FaSTrack Module

FaSTrack Module

Assumptions: As stated above, we assume full state information and noiseless sensing of obstacles. To use FaSTrack, one needs a high-fidelity dynamical model of the system used for reference tracking, and a (potentially simpler) dynamic or kinematic model used by the planning algorithm. Using the relative dynamics between the tracking model and the planning model, the TEB and safety controller may be computed using any high-dimensional Hamilton-Jacobi reachability analysis technique [38, 35, 88, 161, 150].

Implementation: Fig. 9.3 describes the online algorithm for FaSTrack after the offline precomputation of the TEB and safety controller. We initialize the **planning module** to start within the TEB centered on the robot's current state. The planner then uses any desired planning algorithm (e.g. A*, or model predictive control) to find a trajectory from this initial state to a desired goal state. When collision-checking, the planning algorithm must ensure that the tube defined by the Minkowski sum of the TEB and the planned trajectory does not overlap any obstacles in the **obstacle map**.

The planning module provides the current planned reference state to the **FaSTrack controller**, which determines the relative state between the tracking model (robot) and planned reference (motion plan). The controller then applies the corresponding safe tracking control via a look-up table.

FaSTrack in the Framework

In the **robot planning and control** section of Fig. 9.2, each robot uses FaSTrack for robust planning and control. FaSTrack guarantees that each robot remains within its TEB-augmented trajectory.

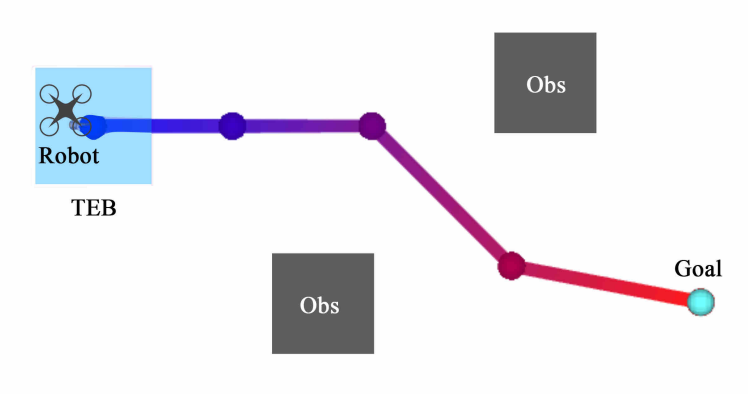


Figure 9.4: Top-down view of FaSTrack applied to a 6D quadcopter navigating a static environment. Note the simple planned trajectory (changing color over time) and the tracking error bound (TEB) around the quadcopter. This TEB is a 6D set that has been projected down to the position dimensions. Because we assume the quadcopter moves independently in (x, y, z), this projection looks like a box, making collision-checking very straightforward.

9.3 Human Predictions

Section 9.2 introduced methods for the fast and safe navigation of a single robot in an environment with deterministic, moving obstacles. However, moving obstacles—especially human beings—are not always best modeled as deterministic. For such "obstacles," robots can employ probabilistic predictive models to produce a distribution of states the human may occupy in the future. The quality of these predictions and the methods used to plan around them determine the overall safety of the system. Generating accurate real-time predictions for multiple humans (and, more generally, uncertain agents) is an open problem. A key challenge arises from the combinatorial explosion of interaction effects as the number of agents increases. Any simplifying assumptions, such as neglecting interaction effects, will inevitably cause predictions to become inaccurate. Such inaccuracies could threaten the safety of plans that rely on these predictions.

Our goal is to compute real-time motion plans that are based on up-to-date predictions of all humans in the environment, and at the same time maintain safety when these predictions become inaccurate. The confidence-aware prediction approach of [72] provides a convenient mechanism for adapting prediction uncertainty online to reflect the degree to which humans' actions match an internal model. This automatic uncertainty adjustment allows us to simplify or even neglect interaction effects between humans, because uncertain predictions will automatically result in more conservative plans when the observed behavior departs from internal modeling assumptions.

Human Prediction Module

Assumptions: In order to make any sort of collision-avoidance guarantees, we require a prediction algorithm that produces distributions over future states, and rapidly adjusts those predictions such that the actual trajectories followed by humans lie within the prediction envelope. There are many approaches to probabilistic trajectory prediction in the literature, e.g. [56, 85, 180, 114]. These methods could be used to produce a probabilistic prediction of the i-th human's state $x_i \in \mathbb{R}^{n_i}$ at future timesteps $\tau > t$, conditioned on the current state at time t.[2] However, these distributions may not always accurately predict the true trajectories of each human, especially when the models do not explicitly account for interaction effects. Fisac et. al. [72] provide an efficient mechanism for direct Bayesian updates to $P(x_i^\tau | x^{0:t})$ which automatically generates more conservative predictions when the human's motion deviates from the modeled structure.

Implementation: Fig. 9.5 illustrates the **human prediction module**. We use a maximum-entropy model as in [72, 68, 179], in which the dynamics of the i-th human are affected by actions u_i^t drawn from a Boltzmann probability distribution. This time-dependent distribution over actions implies a distribution over future states. Given a sensed state x_i^t of human i at time t, we invert the dynamics model to infer the human's action, u_i^t. Given this action, we perform a Bayesian update on the distribution of two parameters: θ_i, which describes the objective of the human (e.g. the set of candidate goal locations), and β_i, which governs the variance of the predicted action distributions. β_i can be interpreted as a natural indicator of *model confidence*, quantifying the model's ability to capture humans' current behavior [72]. Were we to model actions with a different distribution, e.g. a Gaussian process, the corresponding parameters could be learned from prior data [179, 180, 68], or inferred online [152, 72] using standard inverse optimal control (inverse reinforcement learning) techniques.

Once distributional parameters are updated, we produce a prediction over the future actions of human i through the following Boltzmann distribution:

$$P(u_i^t \mid x^t; \beta_i, \theta_i) \propto e^{\beta_i Q_i(x^t, u_i^t; \theta_i)} \ . \tag{9.1}$$

This model treats each human as more likely to choose actions with high expected utility as measured by the (state-action) Q-value associated to a certain reward function, $r_i(x, u_i; \theta_i)$. In general, this value function may depend upon the joint state x and the human's own action u_i, as well as the parameters θ_i, β_i. Finally, combining (9.1) with a dynamics model, these predicted actions may be used to generate a distribution over future states (see eq. (6) in [72]). In practice, we represent this distribution as a discrete occupancy grid. One such grid is visualized in Fig. 9.6.

By reasoning about each human's model confidence as a hidden state [72], our framework dynamically adapts predictions to the evolving accuracy of the models encoded in the set of state-action functions, $\{Q_i\}$. Uncertain predictions will force the planner to be more cautious whenever the actions of the humans occur with low probability as measured by (9.1).

[2] For simplicity, we will assume complete state observability.

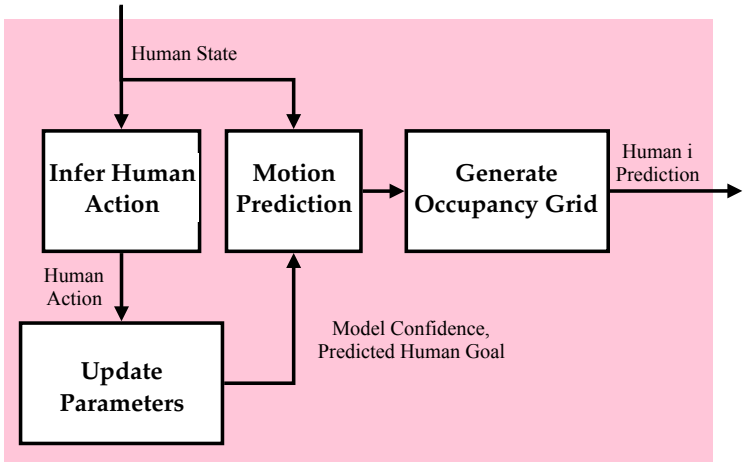


Figure 9.5: Human Prediction Module

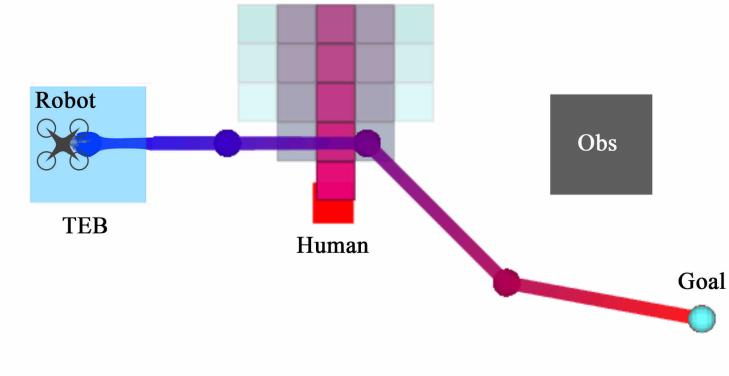


Figure 9.6: Our environment now has a human (red square). The robot models the human as likely to move north. Visualized on top of the human is the distribution of future states (pink is high, blue is low probability). Since the human is walking north and matching the model, the robot's predictions are confident that the human will continue northward and remain collision-free.

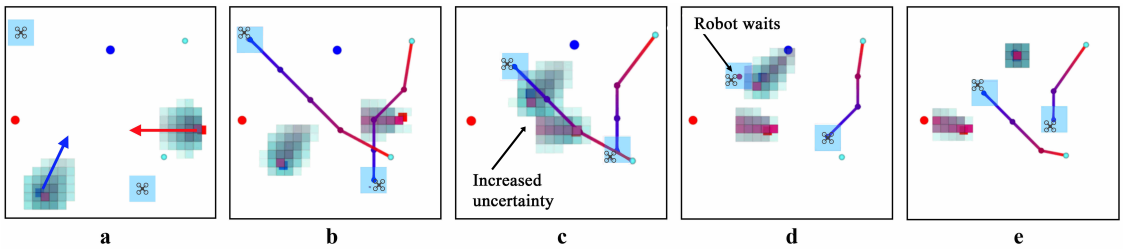


Figure 9.7: Birds-eye visualization of hardware demonstration from Fig. 9.1. (a) Two humans (red and blue) start moving towards their respective goals (also red and blue). Robot in lower right-hand corner has first priority, and robot in upper left-hand corner has second. The time-varying predictions of each human's future motion are visualized. (b) Robots plan trajectories to their goals based on the predictions, priority order, and are guaranteed to stay within the tracking error bound (shown in blue). (c) When the humans begin to interact in an unmodeled way by moving around each other, the future predictions become more uncertain. (d) The robots adjust their plans to be more conservative–note the upper-left robot waiting as the blue human moves past. (c) When the humans pass each other and the uncertainty decreases, the robots complete their trajectories.

Human Prediction in the Framework

The predicted future motion of the humans is generated as a probability mass function, represented as a time-indexed set of occupancy grids. These distributions are interpreted as an obstacle map by the FaSTrack module. During planning, a state is considered to be unsafe if the total probability mass contained within the TEB centered at that state exceeds a preset threshold, P_{th}. As in [72], we consider a trajectory to be unsafe if the maximum *marginal* collision probability at any individual state along it exceeds P_{th}.

When there are multiple humans, their state at any future time τ will generally be characterized by a joint probability distribution $P(x_1^\tau, ..., x_N^\tau)$.

Let p^τ be the planned state of a robot at time τ. We write $\text{coll}(p^\tau, x_i^\tau)$ to denote the overlap of the TEB centered at p^τ with the ith human at state x_i^τ. Thus, we may formalize the probability of collision with *at least one* human as:

$$P\big(\text{coll}(p^\tau, \{x_i^\tau\}_{i=1}^N)\big) = \\ 1 - \prod_{i=1}^{N} P\big(\neg\text{coll}(p^\tau, x_i^\tau) \mid \neg\text{coll}(p^\tau, \{x_j^\tau\}_{j=1}^{i-1})\big) \ . \tag{9.2}$$

Intuitively, (9.2) states that the probability that the robot is in collision at s^τ is one minus the probability that the robot is not in collision. We compute the second term by taking the product over the probability that the robot is not in collision with each human, given that the robot is not in collision with all previously accounted for humans. Unfortunately, it is generally intractable to compute the terms in the product in (9.2). Fortunately, tractable approximations can be computed by storing only the marginal predicted distribution of each

human at every future time step τ, and assuming independence between humans. This way, each robot need only operate with N occupancy grids:

$$P\big(\text{coll}(p^\tau, \{x_i^\tau\}_{i=1}^N)\big) \approx 1 - \prod_{i=1}^{N} \big(1 - P(\text{coll}(p^\tau, x_i^\tau))\big) \ . \tag{9.3}$$

Here we take the product over the probability that the robot is not in collision with each human, and then again take the complement to compute the probability of collision with any human. Note that when the predictive model neglects future interactions between multiple humans, (9.2) reduces to (9.3). If model confidence analysis [72] is used in conjunction with such models, each marginal distribution will naturally become more uncertain when interaction effects are significant.

Once a collision probability is computed, the planner can reject plans for which, at any time $\tau > t$, the probability of collision from (9.3) exceeds P_{th} for any future instant.

9.4 Planning for Multiple Robots

Thus far, we have shown how our framework allows a single robot to navigate in real-time through an environment with multiple humans while maintaining safety (at a probability of approximately P_{th}) and accounting for internal dynamics, external disturbances, and humans. However, in many applications (such as autonomous driving), the environment may also be occupied by other robots.

Finding the optimal set of trajectories for all robots in the environment would require planning over the joint state space of all robots. This very quickly becomes computationally intractable with increasing numbers of robots. Approaches for multi-robot trajectory planning often assume that the other vehicles operate with specific control strategies [173, 69, 33, 168] and involving virtual structures or potential fields to avoid collision [137, 45, 177]. These assumptions greatly reduce the dimensionality of the problem, but may not hold in general. An alternative strategy is for each robot to predict the future motion of all other robots, as in [178, 118, 2]. These techniques all require simple dynamics for each robot.

When robots can communicate with each other, methods for centralized and/or cooperative multi-agent planning allow for other scalable techniques [117, 166, 133]. One such method is sequential trajectory planning (STP) [41], which coordinates robust multi-agent planning using a sequential priority ordering. Priority ordering is commonly used in many multi-agent scenarios, particularly for aerospace applications [105]. In this work, we merge STP with FaSTrack to produce real-time planning for multi-agent systems.

Sequential Trajectory Planning

Assumptions: To apply STP, robots must be able to communicate trajectories and TEBs quickly and reliably over a network. We also assume that each robot can re-check its plan

for safety whenever the environment changes (e.g. humans move) or a higher-priority robot replans.

Implementation: STP addresses the computational complexity of coupled motion planning by assigning a priority order to the robots and allowing higher-priority robots to ignore the planned trajectories of lower-priority robots. The first-priority robot uses the **FaSTrack module** to plan a (time-dependent) trajectory through the environment while avoiding the **obstacle maps**. This trajectory is shared across the network with all lower-priority robots. The i-th robot augments the trajectories from robots $0 : i-1$ by their respective TEBs, and treats them as time-varying obstacles. The i-th robot determines a safe trajectory that avoids these time-varying tubes as well as the predicted state distributions of humans, and publishes this trajectory for robots $i{+}1 : n$. This process continues until all robots have computed their trajectory. Each robot replans as quickly as it is able; in our experiments, this was between 50–300 ms. Overall computational complexity is quadratic in the number of robots, and linear for each individual robot. Note that the method of assigning priority ordering is user-defined based on the objective (for example, ensuring that all robots reach their goal in minimum time vs. allowing certain "emergency" robots higher priority).

Sequential Trajectory Planning in the Framework

By combining this method with FaSTrack for fast individual planning, the sequential nature of STP does not significantly affect overall planning time. Note that STP does depend upon reliable, low-latency communication between the robots. If there are communication delays, techniques such as [51] may be used to augment each robot's TEB by a term relating to time delay.

9.5 Implementation and Experimental Results

We demonstrate our framework's feasibility in hardware with two robots and two humans, and its scalability in simulation with five robots and ten humans.

Hardware Demonstration

We implemented our framework in C++ and Python, using Robot Operating System (ROS) [143]. As shown in Fig. 9.1, we used Crazyflie 2.0 quadcopters as our robots, and two human volunteers. All computations for our hardware demonstration were done on a single laptop computer (specs: 31.3 GB of memory, 216.4 GB disk, Intel Core i7 @ 2.70GHz x 8), due to the limited onboard computational capability of the Crazyflie robots.[3] The position and orientation of robots and humans were measured at roughly 235 Hz by an OptiTrack infrared

[3]Despite running on a single computer, our implementation is fully decentralized within the ROS ecosystem.

motion capture system. The humans were instructed to move towards different places in the lab, while the quadcopters planned collision-free trajectories in three dimensions (x, y, z) using a time-varying implementation of A*. The quadcopters tracked these trajectories using the precomputed FaSTrack controller designed for a 6D near-hover quadcopter model tracking a 3D point [76]. Human motion was predicted 2 s into the future. Fig. 9.7 shows several snapshots of this scene over time. Note that the humans must move around each other to reach their goals—this is an unmodeled interaction affect. The predictions become less certain during this interaction, and the quadcopters plan more conservatively, giving the humans a wider berth. The full video of the hardware demonstration can be viewed in our accompanying video[4].

Framework Simulation Analysis

Due to the relatively small size of our motion capture arena, we demonstrate scalability of the framework through a large-scale simulation. As we increase human and robots in the environment, a robot may not always find a safe plan in their allocated planning time. In this case, we instruct the robot to find a safe location to hover with respect to the higher-priority robots until a path toward their goal is achievable. In this simulation, pedestrians are crossing through a $25 \times 20\text{m}^2$ region of the UC Berkeley campus. We simulate the pedestrians' motion using potential fields [82], which "pull" each pedestrian toward his or her own goal and "push" them away from other pedestrians and robots. These interaction effects between humans and robots are not incorporated into the state-action functions $\{Q_i\}$, and lead to increased model uncertainty (i.e., higher estimates of β_i) during such interactions. The fleet of robots must reach their respective goals while maintaining safety with respect to their internal dynamics, humans, and each other. We ran our simulation on a desktop workstation with an Intel i7 Processor and 12 CPUs operating at 3.3 GHz.[5] Our simulation took 98 seconds for all robots to reach their respective goals. Predictions over human motion took 0.15 ± 0.06 seconds to compute for each human. This computation can be done in parallel. Each robot took 0.23 ± 0.16 seconds to determine a plan. There was no significant difference in planning time between robots of varying priority. Robots used A* on a 2-dimensional grid with 1.5 m resolution, and collision checks were performed at 0.1 m along each trajectory segment. The resolution for human predictions was 0.25 m and human motion was predicted 2 s into the future.

9.6 Chapter Summary

In this chapter, we compose several techniques for robust and efficient planning together in a framework designed for fast multi-robot planning in environments with uncertain moving

[4] https://youtu.be/lJGRHNJ1_Wk

[5] The computation appears to be dominated by the prediction step, which we have not yet invested effort in optimizing.

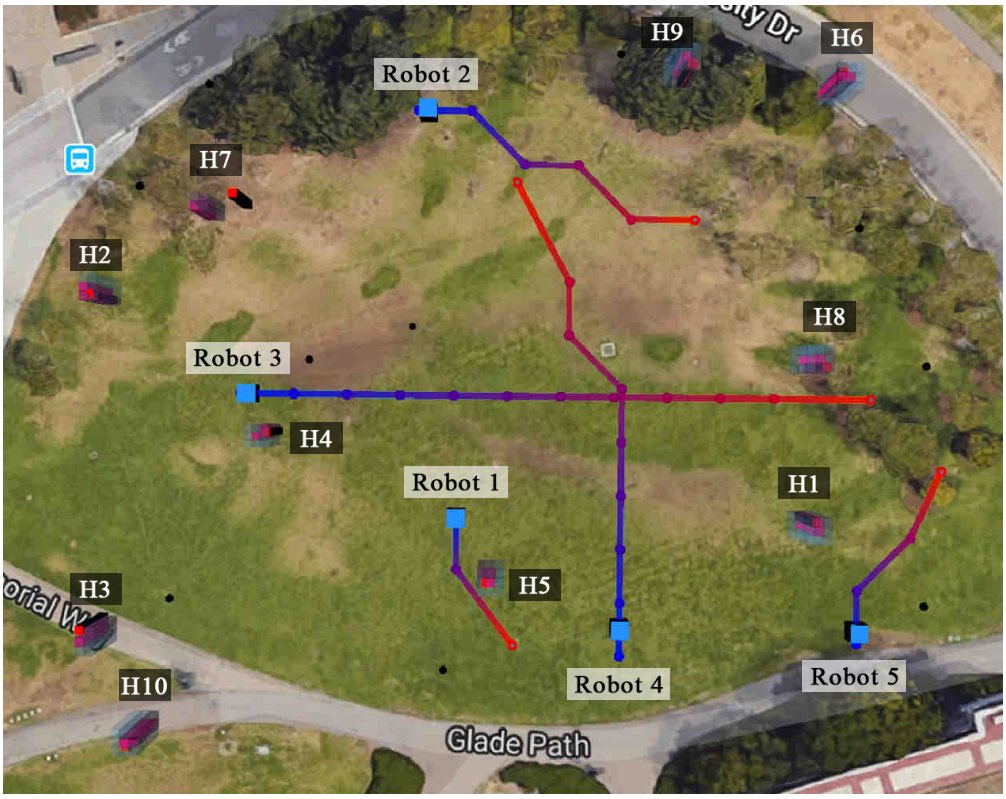


Figure 9.8: Simulation of 5 robots navigating among 10 humans. Simulated humans move according to a potential field, resulting in unmodeled interaction effects. However, our framework enables each robot to reach its goal safely.

obstacles, such as humans. Each robot generates real-time motion plans while maintaining safety with respect to external disturbances and modeled dynamics via the FaSTrack framework. To maintain safety with respect to humans, the robots use human state information to form probabilistic, adaptive predictions over their future trajectories. For efficiency, we model these humans' motions as independent, and to maintain robustness, we adapt prediction model confidence online. Finally, to remain safe with respect to other robots, we introduce multi-robot cooperation through STP, which relieves the computational complexity of planning in the joint state space of all robots by instead allowing robots to plan sequentially according to a fixed priority ordering.

We demonstrate our framework in hardware with two quadcopters navigating around two humans. We also present a larger simulation of five quadcopters and ten humans. To further demonstrate our framework's robustness, we are interested in exploring (a) non-grid based methods of planning and prediction, (b) the incorporation of sensor uncertainty, (c) optimization for timing and communication delays, and (d) recursive feasibility in planning. We are also interested in testing more sophisticated predictive models for humans, and other low-level motion planners.

Part IV

Examples, Codebases, and Conclusions

Chapter 10

HJ Reachability Examples and Codebases

10.1 HJ Reachability Computation Examples

Reach BRS

Recall the system from 2.1, with states $x = [p, v]^T$. In our example we will set the target set (i.e. goal) as $\mathcal{L} = \{(p, v) : |p| \leq 2, |v| < \infty\}$. The target function $l(x)$ will be a signed distance function to the boundary of $\mathcal{L}$. This set and target function can be seen in Fig. 2.3 in black.

We set the final time as $T = 0$, and set the terminal value function as $V(x(T), T) = l(x)$. We then update the value using the Hamilton-Jacobi-Bellman partial differential equation:

$$\begin{aligned} 0 &= D_t V + \min_{u \in [-1,1]} D_x V \cdot f(x, u), \\ -D_t V &= \min_{u \in [-1,1]} \left(\frac{\partial V}{\partial p} \frac{dp}{dt} + \frac{\partial V}{\partial v} \frac{dv}{dt} \right), \\ -D_t V &= \min_{u \in [-1,1]} \left(\frac{\partial V}{\partial p} v + \frac{\partial V}{\partial v} u \right), \\ -D_t V &= \frac{\partial V}{\partial p} v + \min_{u \in [-1,1]} \left(\frac{\partial V}{\partial v} u \right), \\ -D_t V &= \frac{\partial V}{\partial p} v - \left| \frac{\partial V}{\partial v} \right|. \end{aligned} \tag{10.1}$$

Starting with $V(x(T), T) = l(x)$, the partial gradients of the value function can be taken with respect to each state variable. This can intuitively be seen by looking at the slopes of $l(x)$ in Fig. 2.3:

$$\frac{\partial V}{\partial v} = 0, \text{ and}$$

$$\frac{\partial V}{\partial p} = 1 \text{ when } p > 0 \text{ and } \frac{\partial V}{\partial p} = -1 \text{ when } p < 0.$$

These gradients are plugged into the HJB PDE to update the value at each state backwards in time (thus the negative in $-D_t V$) [1]. The process repeats recursively until the initial time has been reached. The corresponding BRSs and value functions are visualized in Fig. 2.5 with initial time $t = -0.1$s (for the finite-time case) or $t = -\infty$s (for the infinite-time case).

Once the value at every time instant has been computed, the optimal control for a particular time instant can be determined by

$$\begin{aligned} u^*(x,t) &= \arg \inf_{u \in [-1,1]} D_x V \cdot f(x,u), \\ u^*(x,t) &= \arg \inf_{u \in [-1,1]} \left(\frac{\partial V}{\partial p} v + \frac{\partial V}{\partial v} u \right), \\ u^*(x,t) &= \arg \inf_{u \in [-1,1]} \left(\frac{\partial V}{\partial v} u \right), \\ u^*(x,t) &= -\left| \frac{\partial V}{\partial v} \right|. \end{aligned} \tag{10.2}$$

Note that in general the derivative $\frac{\partial V}{\partial v}$ must be with respect to the value function corresponding to that particular time instant, i.e. $\frac{\partial V}{\partial v}(t)$.

Avoid BRS

Avoid BRS for 2D Cart

The process of solving for a reach BRS (with the same dynamics and target set) is almost the same as an avoid BRS. Now the target set $\mathcal{L}$ is an obstacle, and the target function $l(x)$ can be considered a terminal reward function. We will use the same set and target function as in Example 10.1, shown in Fig. 2.3.

We set the final time as $T = 0$, and set the terminal value function as $V(x(T), T) = l(x)$. We then update the value using the Hamilton-Jacobi-Bellman partial differential equation:

$$\begin{aligned} 0 &= D_t V + \max_{u \in [-1,1]} D_x V \cdot f(x,u), \\ -D_t V &= \frac{\partial V}{\partial p} v + \left| \frac{\partial V}{\partial v} \right|. \end{aligned} \tag{10.3}$$

Starting with $V(x(T), T) = l(x)$, the partial gradients of the value function can be taken with respect to each state variable. These gradients are plugged into the HJB PDE to update the value at each state backwards in time. The process repeats recursively until the initial time has been reached. The corresponding BRSs and value functions are visualized in Fig. 2.5

[1] Note that there is a discontinuity in the gradient $\frac{\partial V}{\partial p}$ at $p = 0$. Refer to the viscosity solution in Section 2.2 for details.

with initial time $t = -0.1$s (for the finite-time case) or $t = -\infty$s (for the infinite-time case). Once the value at every time instant has been computed, the optimal control for a particular time instant can be determined by

$$\begin{aligned} u^*(x,t) &= \arg\max_{u\in[-1,1]} D_x V \cdot f(x,u), \\ u^*(x,t) &= \left|\frac{\partial V}{\partial v}\right|. \end{aligned} \tag{10.4}$$

Reach BRT

We will solve the same problem setup as in 10.1, but for a BRT. Plugging in the previously computed information, our HJI VI becomes:

$$0 = \min\left\{l(x) - V(x,t), D_t V\delta + \frac{\partial V}{\partial x_1}x_2\delta + \left|\frac{\partial V}{\partial x_2}\right|\delta\right\} \tag{10.5}$$

Initialize to $V(x,T) = l(x)$ and note as before that at time T,

$$\frac{\partial V}{\partial v} = 0,$$

$$\frac{\partial V}{\partial p} = 1 \text{ when } p > 0 \text{ and } \frac{\partial V}{\partial p} = -1 \text{ when } p < 0.$$

Let's focus on updating the value for the state $x = [2, -2]$ (i.e. at a position of 2m and velocity of -2m/s). At this state, $V(x,T) = l(x) = 0$ because it is on the boundary of the goal.
Begin by initializing the current value to $V(x, t+\delta)$. For example, here we set $V(x,t+\delta) = V(x,T)$. We will now solve for $V(x,t)$[2].

We will start by assuming the left hand side of eq. (10.5) **is active.** Then,

$$\begin{aligned} 0 &= l(x) - V(x,t) \\ V(x,t) &= l(x) = 0 \end{aligned} \tag{10.6}$$

Given this, we can solve for $D_t V$:

$$\begin{aligned} V(x,t) + D_t V\delta &= V(x,t+\delta), \\ 0 + D_t V\delta &= 0. \end{aligned} \tag{10.7}$$

[2]Alternatively you can leave the current value as $V(x,t)$ and solve for $V(x,t-\delta)$. The point we're solving for the value function one step backwards in time

Plugging $V(x,t)$ and D_tV into eq. 10.5:

$$\begin{aligned} 0 &= \min\left\{ l(x(t),t) - V(x,t), D_tV\delta + \frac{\partial V}{\partial x_1}x_2\delta + \left|\frac{\partial V}{\partial x_2}\right|\delta \right\} \\ 0 &= \min\left\{ 0-0, 0 + \frac{\partial V}{\partial x_1}x_2\delta + \left|\frac{\partial V}{\partial x_2}\right|\delta \right\} \end{aligned} \tag{10.8}$$

Plugging in our state $x = [2,-2]$ and the corresponding gradients $\frac{\partial V}{\partial x_1} = 1$, $\frac{\partial V}{\partial x_2} = 0$,

$$0 = \min\{0-0, 0 + (1)(-2)\delta + (0)\delta\}\, 0 = \min\{0, -2\delta\} = -2\delta \tag{10.9}$$

Since δ is positive, $0 \neq -2\delta$, so the left hand side must not be active.

Now assume that the right hand side of eq. 10.5 is active.
This will result in the HJI PDE. We can solve this using the steps in Example 10.1, resulting in:

$$\begin{aligned} -D_tV &= \frac{\partial V}{\partial p}v + \left|\frac{\partial V}{\partial v}\right|, \\ -D_tV &= (1)(-2) + 0 \\ D_tV &= 2 \end{aligned} \tag{10.10}$$

Given this, we can solve for $V(x,t)$:

$$\begin{aligned} V(x,t) + D_tV\delta &= V(x,t+\delta), \\ V(x,t) + 2\delta &= 0, \\ V(x,t) &= -2\delta. \end{aligned} \tag{10.11}$$

Plugging $V(x,t)$ and D_tV into eq. 10.5:

$$\begin{aligned} 0 &= \min\left\{ l(x(t),t) - V(x,t), D_tV + \frac{\partial V}{\partial x_1}x_2 + \left|\frac{\partial V}{\partial x_2}\right| \right\} \\ 0 &= \min\left\{ 0 - (-2\delta), 0 + \frac{\partial V}{\partial x_1}x_2 + \left|\frac{\partial V}{\partial x_2}\right| \right\} \\ 0 &= \min\{0 - (-2\delta), 2 + (1)(-2\delta) + (0)\} \\ 0 &= \min\{2\delta, 0\} \end{aligned} \tag{10.12}$$

This is valid, so the LHS must be active, meaning that $V(x,t) = -2\delta$. Repeat for all states. Once the value for all states for time t have been computed, move back one iteration in time. The final reach BRT can be seen in Fig.2.5. The optimal control is acquired using the same process as in the previous examples.

Reach BRT (Modified)

Solving for Reach BRT using (2.67):

$$V(x,t) = \min\left\{l(x(t),t), V(x,t+\delta) + \inf_{u(\cdot)} D_x V \cdot f(x,u)\delta\right\} \tag{10.13}$$

Initialize to $V(x,T) = l(x)$ and note as before that at time T,

$$\frac{\partial V}{\partial v} = 0,$$

$$\frac{\partial V}{\partial p} = 1 \text{ when } p > 0 \text{ and } \frac{\partial V}{\partial p} = -1 \text{ when } p < 0.$$

Begin by initializing the current value to $V(x(t+\delta), t+\delta)$. For example, here we set $V(x(t+\delta), t+\delta) = V(x(T), T)$. Let's focus on updating the value for the state $x = [2, -2]$ (i.e. at a position of 2m and velocity of -2m/s). At this state, $V(x,T) = l(x) = 0$ because it is on the boundary of the goal. We will fix this state and look at how the value changes backwards in time, i.e. we have $V(x(t+\delta), t+\delta) = V(x(t), t+\delta) = V([2,-2], t+\delta = 0$. We will now solve for $V(x,t)$[3].

$$\begin{aligned}
V(x(t),t) &= \min\left\{l(x(t)), V(x(t),t+\delta) + \inf_{u(\cdot)} D_x V \cdot f(x,u)\delta\right\} \\
V(x(t),t) &= \min\left\{0, 0 + \frac{\partial V}{\partial x_1} x_2 \delta + \left|\frac{\partial V}{\partial x_2}\right| \delta\right\} \\
V(x(t),t) &= \min\left\{0, 0 + (1)(-2)\delta + (0)\delta\right\} \\
V(x(t),t) &= \min\left\{0, -2\delta\right\} = -2\delta
\end{aligned} \tag{10.14}$$

Repeat for all states. Once the value for all states for time t have been computed, move back one iteration in time.The final reach BRT can be seen in Fig.2.5.

10.2 Computational Tools for HJ Reachability

This section is an overview of the computational tools used in this thesis that are publicly available.

The Level Set Toolbox (toolboxLS)

The level set toolbox (or *toolboxLS*) was developed by Professor Ian Mitchell [129] 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

[3]Alternatively you can leave the current value as $V(x,t)$ and solve for $V(x, t-\delta)$. The point we're solving for the value function one step backwards in time

any final-value HJ PDE. Since different reachable set computations can be ultimately posed as solving a final-value HJ PDE, 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 here: `https://www.cs.ubc.ca/~mitchell/ToolboxLS/`. 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 the Appendix and is also available at:`http://www.github.com/HJReachability/helperOC`.

The Berkeley Efficient API in C++ for Level Set methods (BEACLS) Toolbox

The Berkeley Efficient API in C++ for Level Set methods (*BEACLS*) Toolbox was developed by Ken Tanabe. This toolbox implements the functions from helperOC and toolboxLS in C++ for fast computation of reachability analyses. The library also uses GPUs for parallelizing different computations in the level set toolbox. The installation instructions and user guide can be found at: `http://www.github.com/HJReachability/beacls`. This GPU library has been used for large-scale multi-vehicle reachability problems, such as safe path planning.

FaSTrack: Fast and Safe Tracking

The FaSTrack framework implementation `https://github.com/HJReachability/fastrack` is composed of a precomputation section in MATLAB (built by myself), and a ROS online implementation in C++ (built largely by David Fridovich-Keil). This toolbox is built primarily with implementation on quadrotors in mind, and is paired with the crazyflie quadrotor flight toolbox `https://github.com/HJReachability/crazyflie_clean`, also built by David Fridovich-Keil.

Chapter 11

Summary and Conclusions

Below is a summary of the work in my dissertation, along with thoughts on where this work can and should continue in the future.

11.1 Making Theoretical Safety Analysis Practical: Scalable Safety Analysis for Learning and Adaptation

Theoretical guarantees of safety are a crucial starting point to achieve trustworthy real-world autonomy by formally verifying the safety of autonomous systems based on known information. However, this analysis is computationally expensive, and has been difficult to update in the face of changing environments and new information. Because of this "curse of dimensionality," tools like Hamilton-Jacobi analysis for guaranteeing safety cannot handle realistic complex models of autonomous systems, and cannot update analyses based on changing knowledge of the system and environment. For example, finding the optimal guaranteed safe trajectory for a 10D nonlinear quadcopter model to simply travel through a room with static obstacles has been completely intractable with current desktop computers.

Contributions

I have developed scalable techniques for theoretically sound safety guarantees that can reduce computation by orders of magnitude for high-dimensional systems (Part I). This allows us to use more realistic high-dimensional models of autonomous systems, resulting in better safety analysis. I have also developed techniques for efficiently updating analyses in the face of new information and changing assumptions on the system and the environment.

Future Vision

I plan to create an interdisciplinary effort to incorporate expertise in control theory, reinforcement learning, and formal methods to blend scalable tools performance-based control policies with theoretical safety analysis. Methods like Hamilton-Jacobi safety analysis verify all possible trajectories of a system to guarantee safety, but suffers from the curse of dimensionality. At the other extreme, techniques for efficient trajectory optimization and policy generation can produce aggressive maneuvers for complex autonomous systems in real time. To achieve scalability these methods typically reason only about expected outcomes and therefore lack an understanding of safety. I will work to explore and blend these techniques to make performance-based methods robust. For example, we can use the control policy generated by these high-dimensional techniques to seed safety analyses and inform crucial regions of the system's state space to formally verify. We can likewise use an initial safety analysis on a simple low-dimensional model of the system to guide the process of tuning the performance control policy.

11.2 Thinking Like Humans: Cognitive Science for Real-Time Decision-Making in Autonomous Systems

While scalability for theoretical guarantees is an important step towards safe real-world autonomy, these analyses depend on pre-defined assumptions about the system model and environment that may be wrong in practice. Therefore, we must equip autonomous systems with the ability to build upon theoretical safety guarantees by reasoning about uncertainty in the real world. This is where humans excel compared to current systems. We can reason efficiently in unstructured environments with limited computational resources. The field of cognitive science has produced models and frameworks of human learning and decision-making that can inform theory and algorithms for autonomous systems. Thrusts like DARPA's Common Sense initiative are pushing researchers to transfer this knowledge to AI systems for better performance, human interaction, and human augmentation.

Contributions

In my PhD work I have blended models of human decision-making from cognitive science with control theory and reinforcement learning to develop algorithms that mimic innate human abilities (Parts II,III). These techniques allow systems to navigate safely and efficiently, and reason over the uncertainty inherent to predicting humans and other agents. For example, the 10-dimensional nonlinear quadcopter can now use these algorithms combined with my scalability work to navigate in real-time through an a priori unknown environment while maintaining safety with respect to model mismatch, disturbances, human pedestrians, and other quadcopters.

Future Vision

I plan to adapt cognitive science for autonomous systems to (a) design algorithms that mimic advantageous human skills, (b) better understand and predict humans, and (c) design systems that complement human decision-making. Cognitive Science has followed a similar path to AI research over the past decades, from complete and computationally difficult "rational" models of humans, to fast heuristic models, to a blending of the two by exploring rational models with bounded computation. I will explore translating this work to autonomous systems.

11.3 Safety Throughout and Beyond Robotics

My work has the potential to impact robotic applications ranging from transportation, assisted living, healthcare, and industrial applications. In many of these applications, safety can be defined as preventing the intersection of the autonomous system and any unsafe obstacles. However, some applications have more nuanced notions of safety: for example, what is safe and unsafe for a surgical robot? When is bumping against a human tolerable (such as with a Roomba) or safety-critical (such as on a highway)? In my lab we will explore how to define and analyze safety throughout robotics. In addition to robotics and control, my work can be directly applied to other high-dimensional safety-critical systems in fields like biology, finance, and geology. My work also has implications for broader aspects of artificial intelligence, where safety is often critical but hard to define and maintain. By merging theory from safety analysis, formal methods, and reinforcement learning, I aim to apply these analyses beyond robotics applications.

11.4 Final Thoughts

In conclusion, autonomous systems are becoming increasingly complex, and are being deployed in unstructured real-world environments. Because of this, effective safety analysis for these systems is both more crucial and more challenging than ever before. I plan to lead an interdisciplinary research agenda directed at blending expertise from control theory, artificial intelligence, and cognitive science to tackle the challenge of safety for real-world autonomy. My lab will focus on creating scalable and adaptable tools to construct theoretical and practical safety guarantees that are backed by rigorous hardware and human-subject experiments.

Bibliography

[1] Amir Ali Ahmadi and Anirudha Majumdar. "DSOS and SDSOS optimization: more tractable alternatives to sum of squares and semidefinite optimization". In: *SIAM Journal on Applied Algebra and Geometry* 3.2 (2019), pp. 193–230. ISSN: 2470-6566.

[2] Ali Ahmadzadeh et al. "Multi-vehicle path planning in dynamically changing environments". In: *International Conference on Robotics and Automation (ICRA)*. IEEE, 2009, pp. 2449–2454. ISBN: 1424427886.

[3] Anayo K Akametalu et al. "A minimum discounted reward hamilton-jacobi formulation for computing reachable sets". In: *arXiv preprint arXiv:1809.00706* (2018).

[4] Anayo K Akametalu et al. "Reachability-based safe learning with Gaussian processes". In: *Conference on Decision and Control (CDC)*. 2014. ISBN: 978-1-4799-7746-8. DOI: 10.1109/CDC.2014.7039601.

[5] Daniel Althoff et al. "Safety assessment of robot trajectories for navigation in uncertain and dynamic environments". In: *Autonomous Robots* 32.3 (2012), pp. 285–302. ISSN: 0929-5593.

[6] Matthias Althoff. "An introduction to CORA 2015". In: *Workshop on Applied Verification for Continuous and Hybrid Systems*. 2015.

[7] Aaron D Ames, Jessy W Grizzle, and Paulo Tabuada. "Control barrier function based quadratic programs with application to adaptive cruise control". In: *Conference on Decision and Control*. IEEE. 2014, pp. 6271–6278.

[8] Heni Ben Amor et al. "Interaction primitives for human-robot cooperation tasks". In: *International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 2831–2837. ISBN: 1479936855.

[9] Georges S Aoude et al. "Probabilistically safe motion planning to avoid dynamic obstacles with uncertain motion patterns". In: *Autonomous Robots* 35.1 (2013), pp. 51–76. ISSN: 0929-5593.

[10] Anil Aswani et al. "Provably safe and robust learning-based model predictive control". In: *Automatica* 49.5 (2013), pp. 1216–1226. ISSN: 0005-1098.

[11] Haoyu Bai et al. "Intention-aware online POMDP planning for autonomous driving in a crowd". In: *International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 454–460. ISBN: 1479969230.

[12] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT press, 2008. ISBN: 0262304031.

[13] Andrea Bajcsy et al. "A scalable framework for real-time multi-robot, multi-human collision avoidance". In: *International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 936–943. ISBN: 153866027X.

[14] Chris L Baker, Joshua B Tenenbaum, and Rebecca R Saxe. "Goal inference as inverse planning". In: *Proceedings of the Annual Meeting of the Cognitive Science Society*. Vol. 29. 29. 2007.

[15] Tirthankar Bandyopadhyay et al. "Intention-aware motion planning". In: *Algorithmic foundations of robotics X*. Springer, 2013, pp. 475–491.

[16] Tirthankar Bandyopadhyay et al. "Intention-aware pedestrian avoidance". In: *Experimental Robotics*. Springer, 2013, pp. 963–977.

[17] Somil Bansal et al. "Hamilton-Jacobi reachability: A brief overview and recent advances". In: *Conference on Decision and Control (CDC)*. IEEE, 2017, pp. 2242–2253. ISBN: 1509028730.

[18] Somil Bansal et al. "Safe Sequential Path Planning of Multi-Vehicle Systems Under Presence of Disturbances and Imperfect Information". In: *American Control Conference (ACC)* (2017).

[19] E N Barron. "Differential Games with Maximum Cost". In: *Nonlinear Analysis: Theory, Methods & Applications* (1990), pp. 971–989.

[20] Andrew J Barry, Anirudha Majumdar, and Russ Tedrake. "Safety verification of reactive controllers for UAV flight in cluttered environments using barrier certificates". In: *International Conference on Robotics and Automation (ICRA)*. IEEE, 2012, pp. 484–490. ISBN: 1467314056.

[21] Alexandre M Bayen et al. "Aircraft Autolander Safety Analysis Through Optimal Control-Based Reach Set Computation". In: *Journal of Guidance, Control, and Dynamics* 30.1 (2007).

[22] Fethi Belkhouche. "Reactive path planning in a dynamic environment". In: *Transactions on Robotics* 25.4 (2009), pp. 902–911. ISSN: 1552-3098.

[23] Richard Bellman. "The Theory of Dynamic Programming". In: *Bulletin of the American Mathematical Society* (1954). ISSN: 02730979. DOI: 10.1090/S0002-9904-1954-09848-8.

[24] Calin Belta, Boyan Yordanov, and Ebru Aydin Gol. *Formal methods for discrete-time dynamical systems*. Vol. 89. Springer, 2017. ISBN: 331950763X.

[25] Dimitri P Bertsekas. "Dynamic Programming and Optimal Control 3rd Edition". In: *Control* (2010). DOI: 10.1.1.141.6891.

[26] Olivier Bokanowski, Nicolas Forcadel, and Hasnaa Zidani. "Reachability and minimal times for state constrained nonlinear problems without any controllability assumption". In: *SIAM Journal on Control and Optimization* (2010), pp. 1–24.

[27] Olivier Bokanowski and Hasnaa Zidani. "Minimal time problems with moving targets and obstacles". In: *IFAC Proceedings Volumes* 44.1 (2011), pp. 2589–2593. ISSN: 1474-6670.

[28] Matthew L Bolton, Ellen J Bass, and Radu I Siminiceanu. "Using formal verification to evaluate human-automation interaction: A review". In: *Transactions on Systems, Man, and Cybernetics: Systems* 43.3 (2013), pp. 488–503. ISSN: 2168-2216.

[29] Patrick Bouffard. "On-board Model Predictive Control of a Quadrotor Helicopter: Design, Implementation, and Experiments". PhD thesis. University of California, Berkeley, 2012.

[30] R R Burridge, A A Rizzi, and D E Koditschek. "Sequential Composition of Dynamically Dexterous Robot Behaviors". In: *International Journal of Robotics Research (IJRR)* 18.6 (June 1999), pp. 534–555. DOI: 10.1177/02783649922066385.

[31] Frank M Callier and Charles A Desoer. *Linear system theory*. Springer Science & Business Media, 1991. ISBN: 1461209579.

[32] Margaret P Chapman et al. "Reachability Analysis as a Design Tool for Stormwater Systems". In: *Conference on Technologies for Sustainability (SusTech)*. IEEE, 2018, pp. 1–8. ISBN: 1538677911.

[33] Georgios C Chasparis and Jeff S Shamma. "Linear-programming-based multi-vehicle path planning with adversaries". In: *American Control Conference (ACC)*. IEEE, 2005, pp. 1072–1077. ISBN: 0780390989.

[34] Mo Chen, Sylvia Herbert, and Claire J Tomlin. "Exact and Efficient Hamilton-Jacobi-based Guaranteed Safety Analysis via System Decomposition". In: *International Conference on Robotics and Automation (ICRA)*. 2017.

[35] Mo Chen, Sylvia Herbert, and Claire J Tomlin. "Fast reachable set approximations via state decoupling disturbances". In: *Conference on Decision and Control*. IEEE. 2016, pp. 191–196.

[36] Mo Chen, Jennifer C Shih, and Claire J Tomlin. "Multi-vehicle collision avoidance via hamilton-jacobi reachability and mixed integer programming". In: *Conference on Decision and Control (CDC)*. IEEE, 2016, pp. 1695–1700. ISBN: 1509018379.

[37] Mo Chen and Claire Tomlin. "Exact and Efficient Hamilton-Jacobi Reachability for Decoupled Systems". In: *Conference on Decision and Control*. Dec. 2015.

[38] Mo Chen et al. "Decomposition of Reachable Sets and Tubes for a Class of Nonlinear Systems". In: *Transactions on Automatic Control (TAC)* (2018). DOI: 10.1109/TAC.2018.2797194.

[39] Mo Chen et al. "Robust sequential path planning under disturbances and adversarial intruder". In: *arXiv preprint arXiv:1611.08364* (2016).

[40] Mo Chen et al. "Robust Sequential Trajectory Planning under Disturbances and Adversarial Intruder". In: *Transactions on Control Systems Technology* 27.4 (2019), pp. 1566–1582. ISSN: 1558-0865. DOI: 10.1109/TCST.2018.2828380.

[41] Mo Chen et al. "Safe Platooning of Unmanned Aerial Vehicles via Reachability". In: *Conference on Decision and Control*. 2015.

[42] Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. "Flow*: An analyzer for non-linear hybrid systems". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 8044 LNCS (2013), pp. 258–263. ISSN: 03029743. DOI: 10.1007/978-3-642-39799-8{_}18.

[43] Mo Chen* et al. "FaSTrack: a Modular Framework for Real-Time Motion Planning and Guaranteed Safe Tracking". In: *Transactions on Automatic Control (TAC)* (2020).

[44] Yat Tin Chow et al. "Algorithm for overcoming the curse of dimensionality for time-dependent non-convex Hamilton–Jacobi equations arising from optimal control and differential games problems". In: *Journal of Scientific Computing* 73.2-3 (2017), pp. 617–643. ISSN: 0885-7474.

[45] Yao-Li Chuang et al. "Multi-vehicle flocking: scalability of cooperative control algorithms using pairwise potentials". In: *International Conference on Robotics and Automation (ICRA)*. IEEE, 2007, pp. 2292–2299. ISBN: 1424406021.

[46] Earl A Coddington and Norman Levinson. *Theory of ordinary differential equations*. Tata McGraw-Hill Education, 1955. ISBN: 0070992568.

[47] Samuel Coogan, Murat Arcak, and Calin Belta. "Formal methods for control of traffic flow: Automated control synthesis from finite-state transition models". In: *Control Systems Magazine* 37.2 (2017), pp. 109–128. ISSN: 1066-033X.

[48] Michael G Crandall and Pierre-Louis Lions. "Viscosity Solutions of {Hamilton-Jacobi} Equations". In: *Transactions of the American Mathematical Society* 277.1 (1983), pp. 1–42.

[49] Charles Dabadie, Shahab Kaynama, and Claire J Tomlin. "A practical reachability-based collision avoidance algorithm for sampled-data systems: Application to ground robots". In: *International Conference on Intelligent Robots and Systems (IROS)*. 2014, pp. 4161–4168. ISBN: 9781479969340. DOI: 10.1109/IROS.2014.6943149.

[50] Jérôme Darbon and Stanley 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. ISSN: 2197-9847.

[51] Ankush Desai et al. "DRONA: a framework for safe distributed mobile robotics". In: *International Conference on Cyber-Physical Systems*. 2017, pp. 239–248.

[52] Debadeepta Dey et al. "Vision and learning for deliberative monocular cluttered flight". In: *Field and Service Robotics*. Springer. 2016, pp. 391–409.

[53] Stefano Di Cairano and Francesco Borrelli. "Reference tracking with guaranteed error bound for constrained linear systems". In: *IEEE Transactions on Automatic Control (TAC)* 61.8 (2015), pp. 2245–2250. ISSN: 0018-9286.

[54] Moritz Diehl, Hans Joachim Ferreau, and Niels Haverbeke. "Efficient numerical methods for nonlinear MPC and moving horizon estimation". In: *Nonlinear Model Predictive Control*. Springer, 2009, pp. 391–417.

[55] Moritz Diehl et al. "Real-time optimization and nonlinear model predictive control of processes governed by differential-algebraic equations". In: *Journal of Process Control* 12.4 (2002), pp. 577–585.

[56] Hao Ding et al. "Human arm motion modeling and long-term prediction for safe and efficient human-robot-interaction". In: *International Conference on Robotics and Automation (ICRA)*. IEEE, 2011, pp. 5875–5880. ISBN: 1612843859.

[57] Jerry Ding et al. "Reachability Calculations for Automated Aerial Refueling". In: *Conference on Decision and Control*. Cancun, Mexico, 2008.

[58] Tommaso Dreossi, Thao Dang, and Carla Piazza. "Parallelotope bundles for polynomial reachability". In: *International Conference on Hybrid Systems: Computation and Control (HSCC)*. 2016, pp. 297–306.

[59] Katherine Driggs-Campbell, Roy Dong, and Ruzena Bajcsy. "Robust, informative human-in-the-loop predictions via empirical reachable sets". In: *Transactions on Intelligent Vehicles* 3.3 (2018), pp. 300–309. ISSN: 2379-8904.

[60] Parasara Sridhar Duggirala et al. "C2E2: A verification tool for stateflow models". In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2015, pp. 68–82.

[61] Parasara Sridhar Duggirala et al. "Tutorial: Software tools for hybrid systems verification, transformation, and synthesis: C2e2, hyst, and tulip". In: *Conference on Control Applications (CCA)*. IEEE, 2016, pp. 1024–1029. ISBN: 1509007555.

[62] L.C. Evans and Panagiotis E. Souganidis. *Differential games and representation formulas for solutions of Hamilton-Jacobi-Isaacs equations*. 1984. DOI: 10.1512/iumj.1984.33.33040.

[63] Lawrence C Evans. *Partial differential equations*. Vol. 19. American Mathematical Soc., 2010. ISBN: 0821849743.

[64] Georgios E Fainekos et al. "Temporal logic motion planning for dynamic robots". In: *Automatica* 45.2 (2009), pp. 343–352. ISSN: 0005-1098.

[65] Chuchu Fan et al. "Automatic reachability analysis for nonlinear hybrid models with C2E2". In: *International Conference on Computer Aided Verification*. Springer, 2016, pp. 531–538.

[66] Jaime Fernandez Fisac. *Game-Theoretic Safety Assurance for Human-Centered Robotic Systems*. 2019.

[67] Hans Joachim Ferreau et al. "qpOASES: A parametric active-set algorithm for quadratic programming". In: *Mathematical Programming Computation* 6.4 (2014), pp. 327–363. ISSN: 1867-2949.

[68] Chelsea Finn, Sergey Levine, and Pieter Abbeel. "Guided cost learning: Deep inverse optimal control via policy optimization". In: *International Conference on Machine Learning (ICML)*. 2016, pp. 49–58.

[69] Paolo Fiorini and Zvi Shiller. "Motion planning in dynamic environments using velocity obstacles". In: *The International Journal of Robotics Research (IJRR)* 17.7 (1998), pp. 760–772. ISSN: 0278-3649.

[70] Jaime F Fisac et al. "Bridging hamilton-jacobi safety analysis and reinforcement learning". In: *International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 8550–8556. ISBN: 153866027X.

[71] Jaime F Fisac et al. "Hierarchical game-theoretic planning for autonomous vehicles". In: *International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 9590–9596. ISBN: 153866027X.

[72] Jaime F Fisac et al. "Probabilistically Safe Robot Planning with Confidence-Based Human Predictions". In: *Robotics: Science and Systems (RSS)* (2018).

[73] Jaime F Fisac et al. "Reach-Avoid Problems with Time-Varying Dynamics , Targets and Constraints". In: *IEEE Conference on Decision and Control (CDC)* (2015).

[74] Goran Frehse et al. "SpaceEx: Scalable verification of hybrid systems". In: *International Conference on Computer Aided Verification*. Springer, 2011, pp. 379–395.

[75] David Fridovich-Keil et al. "Confidence-aware motion prediction for real-time collision avoidance". In: *International Journal of Robotics Research (IJRR)* (2020). ISSN: 17413176. DOI: 10.1177/0278364919859436.

[76] David Fridovich-Keil et al. "Planning, fast and slow: A framework for adaptive real-time safe trajectory planning". In: *International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 387–394. ISBN: 1538630818.

[77] Jonathan D Gammell, Siddhartha S Srinivasa, and Timothy D Barfoot. "Batch informed trees (BIT*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs". In: *International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 3067–3074. ISBN: 1479969230.

[78] Jonathan D Gammell, Siddhartha S Srinivasa, and Timothy D Barfoot. "Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic". In: *International Conference on Intelligent Robots and Systems*. IEEE, 2014, pp. 2997–3004. ISBN: 1479969346.

[79] Gavin (Matlab community Contributor). *Multiple Rapidly-exploring Random Tree (RRT)*. 2013. URL: https://www.mathworks.com/matlabcentral/fileexchange/21443-multiple-rapidly-exploring-random-tree-rrt.

[80] Jeremy H Gillula et al. "Applications of hybrid reachability analysis to robotic aerial vehicles". In: *The International Journal of Robotics Research (IJRR)* 30.3 (2011), pp. 335–354. ISSN: 0278-3649.

[81] Jeremy H Gillula et al. "Design of guaranteed safe maneuvers using reachable sets: Autonomous quadrotor aerobatics in theory and practice". In: *International Conference on Robotics and Automation (ICRA)*. IEEE. 2010, pp. 1649–1654.

[82] Michael A Goodrich. "Potential fields tutorial". In: *Class Notes* 157 (2002).

[83] Mark R Greenstreet and Ian Mitchell. "Integrating projections". In: *International Workshop on Hybrid Systems: Computation and Control*. Springer, 1998, pp. 159–174.

[84] Peter E Hart, Nils J Nilsson, and Bertram Raphael. "A formal basis for the heuristic determination of minimum cost paths". In: *Transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107. ISSN: 0536-1567.

[85] Kelsey P Hawkins et al. "Probabilistic human action prediction and wait-sensitive planning for responsive human-robot collaboration". In: *International Conference on Humanoid Robots (Humanoids)*. IEEE, 2013, pp. 499–506. ISBN: 1479926191.

[86] Didier Henrion and Milan Korda. "Convex computation of the region of attraction of polynomial control systems". In: *Transactions on Automatic Control (TAC)* 59.2 (2013), pp. 297–312. ISSN: 0018-9286.

[87] Sylvia L. Herbert et al. "Reachability-Based Safety Guarantees using Efficient Initializations". In: *Conference on Decision and Control (CDC)* 2019-Decem.Cdc (2019), pp. 4810–4816. ISSN: 07431546. DOI: 10.1109/CDC40024.2019.9029575.

[88] Sylvia L Herbert et al. "FaSTrack: A modular framework for fast and guaranteed safe motion planning". In: *Conference on Decision and Control (CDC)*. IEEE, 2017, pp. 1517–1522. ISBN: 1509028730.

[89] Boris Houska, Hans Joachim Ferreau, and Moritz Diehl. "ACADO toolkit—An open-source framework for automatic control and dynamic optimization". In: *Optimal Control Applications and Methods* 32.3 (2011), pp. 298–312. ISSN: 0143-2087.

[90] Michael Hoy, Alexey S Matveev, and Andrey V Savkin. "Algorithms for collision-free navigation of mobile robots in complex cluttered environments: a survey". In: *Robotica* 33.03 (2015), pp. 463–497.

[91] Haomiao Huang et al. "A differential game approach to planning in adversarial scenarios: A case study on capture-the-flag". In: *International Conference on Robotics and Automation (ICRA)*. 2011, pp. 1451–1456.

[92] Lucas Janson et al. "Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions". In: *The International Journal of Robotics Research (IJRR)* 34.7 (2015), pp. 883–921.

[93] Susmit Jha et al. "Safe autonomy under perception uncertainty using chance- constrained temporal logic". In: *Journal of Automated Reasoning* 60.1 (2018), pp. 43–62.

[94] Daniel Kahneman. *Thinking, fast and slow*. Macmillan, 2011. ISBN: 0374275637.

[95] Mrinal Kalakrishnan et al. "STOMP: Stochastic trajectory optimization for motion planning". In: *International Conference on Robotics and Automation*. IEEE. 2011, pp. 4569–4574.

[96] Sertac Karaman and Emilio Frazzoli. "Sampling-based algorithms for optimal motion planning". In: *The International Journal of Robotics Research (IJRR)* 30.7 (2011), pp. 846–894. ISSN: 0278-3649.

[97] Sertac Karaman et al. "Anytime motion planning using the RRT". In: *International Conference on Robotics and Automation (ICRA)*. IEEE. 2011, pp. 1478–1483.

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

[99] Shahab Kaynama et al. "Computing the viability kernel using maximal reachable sets". In: *Hybrid Systems: Computation and Control (HSCC)*. 2012, pp. 55–64.

[100] Eric Shinwon Kim. *Constructive Formal Control Synthesis through Abstraction and Decomposition*. 2019.

[101] Ross A Knepper and Daniela Rus. "Pedestrian-inspired sampling-based multi-robot collision avoidance". In: *International Symposium on Robot and Human Interactive Communication*. IEEE, 2012, pp. 94–100. ISBN: 1467346063.

[102] Marin Kobilarov. "Cross-entropy motion planning". In: *The International Journal of Robotics Research (IJRR)* 31.7 (2012), pp. 855–871.

[103] Mykel J Kochenderfer et al. "Airspace encounter models for estimating collision risk". In: *Journal of Guidance, Control, and Dynamics* 33.2 (2010), pp. 487–499. ISSN: 0731-5090.

[104] Soonho Kong et al. "dReach: δ-reachability analysis for hybrid systems". In: *International Conference on TOOLS and Algorithms for the Construction and Analysis of Systems*. Springer, 2015, pp. 200–205.

[105] Parimal Kopardekar et al. "Unmanned aircraft system traffic management (UTM) concept of operations". In: (2016).

[106] Hema Swetha Koppula and Ashutosh Saxena. "Anticipating human activities for reactive robotic response." In: *International Conference on Intelligent Robots and Systems (IROS)*. Tokyo, 2013, p. 2071.

[107] Shreyas Kousik et al. "Bridging the Gap Between Safety and Real-Time Performance in Receding-Horizon Trajectory Design for Mobile Robots". In: *arXiv* (2018). URL: http://arxiv.org/abs/1809.06746.

[108] Henrik Kretzschmar, Markus Kuderer, and Wolfram Burgard. "Learning to predict trajectories of cooperatively navigating agents". In: *International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 4015–4020. ISBN: 1479936855.

[109] Thibault Kruse et al. "Human-aware robot navigation: A survey". In: *Robotics and Autonomous Systems* 61.12 (2013), pp. 1726–1743. ISSN: 0921-8890.

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

[111] Alexander B Kurzhanski and Pravin Varaiya. "Ellipsoidal techniques for reachability analysis: internal approximation". In: *Systems & control letters* 41.3 (2000), pp. 201–211. ISSN: 0167-6911.

[112] Alexander B Kurzhanski and Pravin 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. ISSN: 1055-6788.

[113] Chi-Pang Lam et al. "Human-centered robot navigation—Toward a harmoniously coexisting multi-human and multi-robot environment". In: *International Conference on Intelligent Robots and Systems*. IEEE, 2010, pp. 1813–1818. ISBN: 1424466768.

[114] Przemyslaw A Lasota and Julie A Shah. "Analyzing the effects of human-aware motion planning on close-proximity human–robot collaboration". In: *Human factors* 57.1 (2015), pp. 21–33. ISSN: 0018-7208.

[115] Jean B Lasserre et al. "Nonlinear Optimal Control via Occupation Measures and LMI-Relaxations". In: *SIAM Journal on Control and Optimization* 47.4 (June 2008), pp. 1643–1666. ISSN: 0363-0129. DOI: 10.1137/070685051.

[116] D Lee. "Google plans drone delivery service for 2017". In: *BBC News* (2016).

[117] Frank L Lewis et al. *Cooperative control of multi-agent systems: optimal and adaptive design approaches*. Springer Science & Business Media, 2013. ISBN: 1447155742.

[118] Feng-Li Lian and Richard Murray. "Real-time trajectory generation for the cooperative path planning of multi-vehicle systems". In: *Conference on Decision and Control (CDC)*. Vol. 4. IEEE, 2002, pp. 3766–3769. ISBN: 0780375165.

[119] R Duncan Luce. *Individual choice behavior: A theoretical analysis*. Courier Corporation, 2012. ISBN: 0486153398.

[120] John N Maidens et al. "Lagrangian methods for approximating the viability kernel in high-dimensional systems". In: *Automatica* 49.7 (2013), pp. 2017–2029. ISSN: 0005-1098.

[121] Anirudha Majumdar, Amir Ali Ahmadi, and Russ Tedrake. "Control design along trajectories with sums of squares programming". In: *International Conference on Robotics and Automation (ICRA)*. IEEE, 2013, pp. 4054–4061. ISBN: 1467356433.

[122] Anirudha Majumdar and Russ Tedrake. "Funnel libraries for real-time robust feedback motion planning". In: *International Journal of Robotics Research (IJRR)* 36.8 (July 2017), pp. 947–982. DOI: 10.1177/0278364917712421.

[123] Kostas Margellos and John Lygeros. "Hamilton-jacobi formulation for reach-avoid differential games". In: *Transactions on Automatic Control (TAC)* (2011). ISSN: 00189286. DOI: 10.1109/TAC.2011.2105730.

[124] James S McGrew et al. "Air-combat strategy using approximate dynamic programming". In: *Journal of Guidance, Control, and Dynamics* 33.5 (2010), pp. 1641–1654. ISSN: 0731-5090.

[125] Smitha Milli, Falk Lieder, and Thomas L Griffiths. "When does bounded-optimal metareasoning favor few cognitive systems?" In: *AAAI Conference on Artificial Intelligence*. 2017.

[126] I M Mitchell, A M Bayen, and C J Tomlin. "A time-dependent {Hamilton-Jacobi} formulation of reachable sets for continuous dynamic games". In: *Transactions on Automatic Control (TAC)* 50.7 (), pp. 947–957. DOI: 10.1109/TAC.2005.851439.

[127] Ian M Mitchell. "Comparing Forward and Backward Reachability As Tools for Safety Analysis". In: *International Conference on Hybrid Systems: Computation and Control (HSCC)*. 2007.

[128] Ian M 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 (HSCC)*. 2011, pp. 103–112. ISBN: 978-1-4503-0629-4.

[129] Ian M Mitchell. "The Flexible, Extensible and Efficient Toolbox of Level Set Methods". In: *Journal of Scientific Computing* 35.2-3 (2008), pp. 300–329. DOI: 10.1007/s10915-007-9174-4.

[130] Ian M Mitchell, Mo Chen, and Meeko Oishi. "Ensuring safety of nonlinear sampled data systems through reachability". In: *IFAC Proc. Volumes* 45.9 (2012), pp. 108–114. DOI: 10.3182/20120606-3-NL-3011.00104.

[131] Ian M Mitchell and Claire J Tomlin. "Overapproximating Reachable Sets by Hamilton-Jacobi Projections". In: *Journal of Scientific Computing* 19.1-3 (2003), pp. 323–346. ISSN: 0885-7474. DOI: 10.1023/A:1025364227563.

[132] Ian M Mitchell et al. "Safety preserving control synthesis for sampled data systems". In: *Nonlinear Analysis: Hybrid Systems* 10.1 (2013), pp. 63–82. DOI: 10.1016/j.nahs.2013.04.003.

[133] Thulasi Mylvaganam, Mario Sassano, and Alessandro Astolfi. "A differential game approach to multi-agent collision avoidance". In: *Transactions on Automatic Control (TAC)* 62.8 (2017), pp. 4229–4235. ISSN: 0018-9286.

[134] Michael Neunert et al. "Fast nonlinear model predictive control for unified trajectory optimization and tracking". In: *International Conference on Robotics and Automation (ICRA)*. IEEE. 2016, pp. 1398–1404.

[135] Andrew Y Ng and Stuart J Russell. "Algorithms for inverse reinforcement learning." In: *International Conference on Machine Learning (ICML)*. Vol. 1. 2000, p. 2.

[136] Petter Nilsson and Necmiye Ozay. "Synthesis of separable controlled invariant sets for modular local control design". In: *American Control Conference (ACC)*. IEEE, 2016, pp. 5656–5663. ISBN: 1467386820.

[137] Reza Olfati-Saber and Richard M Murray. "Distributed cooperative control of multiple vehicle formations using structural potential functions". In: *IFAC world congress*. Vol. 15. 1. Barcelona, Spain, 2002, pp. 242–248.

[138] Stanley Osher and R. Fedkiw. *Level Set Methods and Dynamic Implicit Surfaces*. 2006. ISBN: 1852337087.

[139] Pablo A Parrilo. "Structured semidefinite programs and semialgebraic geometry methods in robustness and optimization". PhD thesis. California Institute of Technology, 2000.

[140] Jointed Planning. *Development Office (JPDO), "Unmanned aircraft systems (UAS) comprehensive plan–a report on the nation's UAS path forward," Federal Aviation Administration*. Tech. rep. 2013.

[141] Thomas Prevot et al. "UAS traffic management (UTM) concept of operations to safely enable low altitude flight operations". In: *AIAA Aviation Technology, Integration, and Operations Conference*. 2016, p. 3292.

[142] S Joe Qin and Thomas A Badgwell. "A survey of industrial model predictive control technology". In: *Control engineering practice* 11.7 (2003), pp. 733–764.

[143] Morgan Quigley et al. "ROS: an open-source Robot Operating System". In: *ICRA workshop on open source software*. Vol. 3. 3.2. Kobe, Japan, 2009, p. 5.

[144] Saša V Raković. "Set theoretic methods in model predictive control". In: *Nonlinear Model Predictive Control*. Springer, 2009, pp. 41–54.

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

[146] Mark Reynolds. "Continuous temporal models". In: *Australian Joint Conference on Artificial Intelligence*. Springer, 2001, pp. 414–425.

[147] Arthur Richards and Jonathan P How. "Robust variable horizon model predictive control for vehicle maneuvering". In: *International Journal of Robust and Nonlinear Control* 16.7 (2006), pp. 333–351.

[148] Charles Richter, Adam Bry, and Nicholas Roy. "Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments". In: *Robotics Research*. 2016.

[149] Stefan Richter, Colin Neil Jones, and Manfred Morari. "Computational complexity certification for real-time MPC with input constraints based on the fast gradient method". In: *Transactions on Automatic Control (TAC)* 57.6 (2011), pp. 1391–1403. ISSN: 0018-9286.

[150] Vicenc Rubies Royo et al. "Classification-based Approximate Reachability with Guarantees Applied to Safe Trajectory Tracking". In: *International Conference on Robotics and Automation (ICRA)* (2019).

[151] Stuart Russell and Eric Wefald. "'Principles of Metareasoning". In: *Artificial intelligence* 49.1-3 (1992), pp. 361–395.

[152] Dorsa Sadigh et al. "Planning for autonomous cars that leverage effects on human actions." In: *Robotics: Science and Systems (RSS)*. Vol. 2. Ann Arbor, MI, USA, 2016.

[153] Shankar Sastry. *Nonlinear systems: analysis, stability, and control*. Vol. 10. Springer Science & Business Media, 1999. ISBN: 1475731086.

[154] Georg Schildbach and Francesco Borrelli. "A dynamic programming approach for nonholonomic vehicle maneuvering in tight environments". In: *Conference on Intelligent Vehicles Symposium (IV)*. IEEE. 2016, pp. 151–156.

[155] Edward Schmerling et al. "Multimodal probabilistic model-based planning for human-robot interaction". In: *International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 1–9. ISBN: 1538630818.

[156] John Schulman et al. "Finding Locally Optimal, Collision-Free Trajectories with Sequential Convex Optimization." In: *Robotics: Science and Systems (RSS)*. Vol. 9. 1. Citeseer. 2013, pp. 1–10.

[157] Ulrich Schwesinger et al. "A sampling-based partial motion planning framework for system-compliant navigation along a reference path". In: *Intelligent Vehicles Symposium (IV), 2013 IEEE*. IEEE. 2013, pp. 391–396.

[158] J. A. Sethian. "A fast marching level set method for monotonically advancing fronts". In: *National Academy of Sciences* 93.4 (1996), pp. 1591–1595.

[159] Seyed Mahdi Shavarani et al. "Application of hierarchical facility location problem for optimization of a drone delivery system: a case study of Amazon prime air in the city of San Francisco". In: *The International Journal of Advanced Manufacturing Technology* 95.9-12 (2018), pp. 3141–3153. ISSN: 0268-3768.

[160] Sumeet Singh et al. "Robust online motion planning via contraction theory and convex optimization". In: *International Conference on Robotics and Automation (ICRA)*. 2017. ISBN: 9781509046331. DOI: 10.1109/ICRA.2017.7989693.

[161] S Singh et al. "Robust Tracking with Model Mismatch for Fast and Safe Planning: an {SOS} Optimization Approach". In: *Workshop on Algorithmic Foundations of Robotics (WAFR)*. 2018.

[162] Ioan A Sucan, Mark Moll, and Lydia E Kavraki. "The open motion planning library". In: *Robotics & Automation Magazine (RAM)* 19.4 (2012), pp. 72–82. ISSN: 1070-9932.

[163] Ryo Takei and Richard Tsai. "Optimal Trajectories of Curvature Constrained Motion in the Hamilton–Jacobi Formulation". In: *Journal of Scientific Computing* 54.2-3 (2013), pp. 622–644. DOI: 10.1007/s10915-012-9671-y.

[164] Russ Tedrake et al. "LQR-trees: Feedback motion planning via sums-of-squares verification". In: *The International Journal of Robotics Research (IJRR)* 29.8 (2010), pp. 1038–1052. ISSN: 0278-3649.

[165] C J Tomlin, J Lygeros, and S Shankar Sastry. "A game theoretic approach to controller design for hybrid systems". In: *Proceedings of the IEEE* 88.7 (July 2000), pp. 949–970.

[166] Alejandro Torreño et al. "Cooperative multi-agent planning: A survey". In: *ACM Computing Surveys (CSUR)* 50.6 (2017), pp. 1–32. ISSN: 0360-0300.

[167] Peter Trautman and Andreas Krause. "Unfreezing the robot: Navigation in dense, interacting crowds". In: *International Conference on Intelligent Robots and Systems*. IEEE, 2010, pp. 797–803. ISBN: 1424466768.

[168] Jur Van den Berg, Ming Lin, and Dinesh Manocha. "Reciprocal velocity obstacles for real-time multi-agent navigation". In: *International Conference on Robotics and Automation (ICRA)*. IEEE, 2008, pp. 1928–1935. ISBN: 1424416469.

[169] Pravin Varaiya. "On the existence of solutions to a differential game". In: *SIAM Journal on Control and Optimization* 5.1 (1967), pp. 153–162.

[170] Michael Vitus et al. "Tunnel-milp: Path planning with sequential convex polytopes". In: *AIAA guidance, navigation and control conference and exhibit*. 2008, p. 7132.

[171] John Von Neumann, Oskar Morgenstern, and Harold William Kuhn. *Theory of games and economic behavior (commemorative edition)*. Princeton university press, 2007. ISBN: 0691130612.

[172] Zijian Wang, Riccardo Spica, and Mac Schwager. "Game theoretic motion planning for multi-robot racing". In: *Distributed Autonomous Robotic Systems*. Springer, 2019, pp. 225–238.

[173] Albert Wu and Jonathan P How. "Guaranteed infinite horizon avoidance of unpredictable, dynamically constrained obstacles". In: *Autonomous robots* 32.3 (2012), pp. 227–242. ISSN: 0929-5593.

[174] Melanie Nicole Zeilinger, Colin Neil Jones, and Manfred Morari. "Real-time suboptimal model predictive control using a combination of explicit MPC and online optimization". In: *Transactions on Automatic Control (TAC)* 56.7 (2011), pp. 1524–1534.

[175] Xiaojing Zhang, Alexander Liniger, and Francesco Borrelli. "Optimization-based collision avoidance". In: *Transactions on Control Systems Technology* (2020). ISSN: 1063-6536.

[176] Pengcheng Zhao, Shankar Mohan, and Ram Vasudevan. "Control synthesis for nonlinear optimal control via convex relaxations". In: *American Control Conference (ACC)*. IEEE, 2017, pp. 2654–2661. ISBN: 150905992X.

[177] Dingjiang Zhou, Zijian Wang, and Mac Schwager. "Agile coordination and assistive collision avoidance for quadrotor swarms using virtual structures". In: *Transactions on Robotics* 34.4 (2018), pp. 916–923. ISSN: 1552-3098.

[178] Dingjiang Zhou et al. "Fast, on-line collision avoidance for dynamic vehicles using buffered voronoi cells". In: *Robotics and Automation Letters (RAL)* 2.2 (2017), pp. 1047–1054. ISSN: 2377-3766.

[179] Brian D Ziebart et al. "Maximum entropy inverse reinforcement learning." In: *AAAI*. Vol. 8. Chicago, IL, USA, 2008, pp. 1433–1438.

[180] Brian D Ziebart et al. "Planning-based prediction for pedestrians". In: *International Conference on Intelligent Robots and Systems*. IEEE, 2009, pp. 3931–3936. ISBN: 1424438039.