High Dimensional Reachability Analysis: Addressing the Curse of Dimensionality in Formal Verification

Mo Chen
Claire Tomlin, Ed.

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2017-132
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2017/EECS-2017-132.html

July 26, 2017

High Dimensional Reachability Analysis:
Addressing the Curse of Dimensionality in Formal Verification

by

Mo Chen

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 Murat Arcak
Professor Andrew Packard

Summer 2017

High Dimensional Reachability Analysis:
Addressing the Curse of Dimensionality in Formal Verification

Mo Chen

Abstract

High Dimensional Reachability Analysis:
Addressing the Curse of Dimensionality in Formal Verification

by

Mo Chen
Doctor of Philosophy in Engineering - Electrical Engineering and Computer Sciences

University of California, Berkeley
Professor Claire Tomlin, Chair

Automation is becoming pervasive in everyday life, and many automated systems, such as unmanned aerial systems, autonomous cars, and many types of robots, are complex and safety-critical. Formal verification tools are essential for providing performance and safety guarantees for these systems. In particular, reachability analysis has previously been successfully applied to small scale control systems with general nonlinear dynamics under the influence of disturbances. Its exponentially scaling computational complexity, however, makes analyzing more complex, large scale systems intractable. Alleviating computation burden is in general a primary challenge in formal verification.

This thesis presents ways to tackle this "curse of dimensionality" from multiple fronts, bringing tractable verification of complex, practical systems such as unmanned aerial systems, autonomous cars and robots, and biological systems closer to reality. The theoretical contributions pertain to Hamilton-Jacobi (HJ) reachability analysis, with applications to unmanned aerial system. In addition, this thesis also explores two frontiers of HJ reachability by combining the formal guarantees of reachability with the computational advantages of optimization and machine learning, and with fast motion planning algorithms commonly used in robotics. The potential and benefits of the theoretical advances are demonstrated in numerous practical applications.

Contents

Acknowledgments

First, I would like to express special appreciation for the guidance I received from my thesis committee members Professor Andrew Packard, Professor Murat Arcak, and Professor Claire Tomlin. The wisdom you imparted on me has benefited me greatly in both tangible and intangible ways. In addition, this thesis would also not have been possible without many generous, caring, and fun people.

The first step to producing new results in research is learning how to do research; I am deeply grateful for receiving the guidance of Anil, Ruzena, Max, Young-Hwan, Suraj, Junkai, Frauke, Simon, Insoon, Zhengyuan, Chris, Meeko, Ian, Tomizuka, Jose, Karl, Andy, Murat, and Claire. Without you, I would not even be able to begin this thesis.

Of course, a thesis cannot be finished without its contents, and contents are only possible through research with collaborators. Whenever "we" appears in this thesis, I mean Kene, Somil, Glen, Aparna, Jaime, SooJean, Sylvia, Frank, Qie, Casey, Jennifer, Mahesh, Shankar, and Claire. Through these collaborations, I have learned that a great researcher is more often one with many collaborators rather than the stereotypical lone scientist.

Technical content and research vision is only a small part of a PhD journey. Uncountable intangible factors are needed for the courage to pursue a PhD, the creativity to create new challenges, and the confidence to propose seemingly foolish solutions. These qualities were jointly granted to me by countless many people: Kene, Somil, Lititia, Lucretia, Qi, Sylvia, Qie, Ian, Ying, Winnie, Meeko, Claire, Sally, Shicong, Jack, and everyone else I missed.

Chapter 1

Introduction

Autonomous systems have become increasingly pervasive in every day life. These systems include unmanned aerial systems, self-driving cars, and many other types of robots. By now, it goes without saying that these systems have many potential applications, limited only by our imagination. In the recent years, a tremendous amount of progress has been made in autonomous systems research, for example in sub-areas such as modeling, planning, and sensing and perception. In addition, the availability of computing power and hardware platforms today have also helped bridge the gap between theory and practical implementation.

Despite the recent success in automation, our use of robots and interactions with robots are still quite limited. For example, one of the current uses of unmanned aerial vehicles (UAVs) is surveying areas with very few people and no other air traffic. In general, robotic operations are restricted to controlled environments, and involve a single robot or a few robots. These robots also have limited interactions with other agents such as humans. There are likely many reasons for this, and one reason is simple: If we put many robots close to each other and to humans, we would not know for sure whether they will harm each other or harm humans.

The perspective of safety is crucial for enabling more effective use of autonomous systems, many of which are safe-critical systems. Safety critical systems are systems in which failure is extremely costly, or even fatal. Formal safety analysis will allow autonomous systems to become provably robust to changes in the environment and to other agents, as well as operate in much denser configurations. This would mean, for example, that thousands of UAVs could fly in an urban area. Safety analysis is also essential for allowing automation to interact closely and physically with humans.

1.1 Safety-Critical Autonomous Systems

On an intuitive level, maintaining safety could be simply avoiding an obstacle, such as a tree. Sometimes the obstacle may be an agent that can also control the

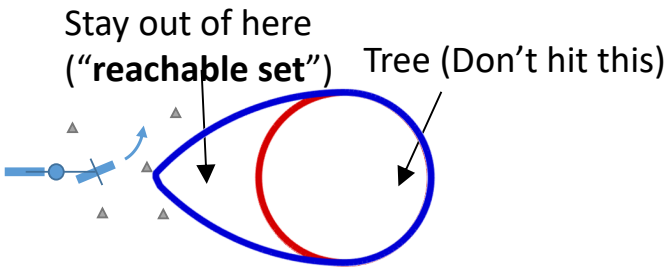


Figure 1.1: Reachability analysis quantifies when the bike needs to steer away to avoid the tree.

way it moves, like an aircraft. On a broader level, maintaining safety means keeping within a set of safe operating conditions. Staying out of obstacles is one specific example, but this concept is quite general. For example, safe operating conditions can be defined in terms of not only position, but also any other variables of interest such as velocities, angles, or even voltages or concentrations of chemicals, human comfort, and degree of trust in automation.

Verification of systems is challenging for many reasons. First, all possible system behaviors need to be accounted for. This makes most simulation-based approaches insufficient, and is where formal verification methods are needed. Many practical systems operate in complex environments. These systems are affected by disturbances such as weather conditions. The environments can be unpredictable, and may even contain adversarial agents. In addition, the systems evolve in continuous time with complex, nonlinear dynamics.

Perhaps the most difficult challenge of all is that these systems often have high-dimensional configuration spaces. High-dimensionality means that many variables are needed to describe the state of a system. This could occurs if the system of interest is very complex, or if there are many agents in the system, or both.

1.2 Reachability as a Safety Analysis Tool

The focus of this thesis is reachability analysis, one of the most powerful formal verification tools for guaranteeing performance and safety properties of systems. The idea is quite simple: Imagine riding a bike, and suppose that there is a tree in front. Obviously, we do not want to run into the tree. Figure 1.1 shows a simplified diagram in which we have the bike, and a circular area that represents the tree. The way we can avoid hitting the tree is to make sure to change our direction of travel early enough, while taking into account variables such as momentum and steering capabilities of the bike, and any disturbances like rough terrain that might affect steering.

Reachability analysis quantifies exactly what it means to steer away *early enough*. This is done by computing the backward reachable tube or in some cases a backward reachable set, a region that we must stay out of in order to be able to avoid the obstacle. In a more generalized setting, where wed like to keep our system within safe operating conditions, reachability analysis tells us how far away from the unsafe conditions the system needs to stay away from.

Reachability analysis has been studied extensively in the past several decades. It is a formal verification method, and accounts for all possible system behaviors, but is computationally expensive. So, many reachability methods are only applicable to certain classes of system dynamics, such as linear, or polynomial. Others approximate reachable sets using simple shapes such as polytopes, or only applicable to systems without control and disturbance variables. In fact, most methods have multiple of these restrictions.

This thesis builds upon and extends beyond the Hamilton-Jacobi (HJ) formulation of reachability, which is the most general formulation, without any of the above restrictions. It is applicable to general controlled nonlinear systems that involve disturbances or adversarial behaviors, and despite this, computes the exact reachable set rather than approximations. The trade off here is that HJ reachability is the most computationally expensive method.

In terms of the challenges in analyzing safety critical systems, HJ reachability is very well suited for addressing all except for one: Like every other formal verification method, computation burden makes HJ reachability intractable for high dimensional systems. This thesis presents several methods to alleviate this last challenge, which is referred to as the "curse of dimensionality".

In the case of HJ reachability, the computational complexity is exponential with respect to the number of system dimensions. This is depicted by Figure 1.2. Using HJ reachability, 1D and 2D reachable sets can be computed very quickly, and do not use much RAM. 3D reachable sets can take minutes to hours to compute, and require hundreds of megabytes of memory. 4D reachable sets typically take many hours to days to compute, and require many gigabytes of memory. Due to computation time and memory limitations, reachable sets of 5 or more dimensions have been considered intractable to compute prior to the work on system decomposition in this thesis.

Therefore, despite its recent success, reachability methods are in general intractable for high-dimensional systems. Unfortunately, these are the systems for which performance and safety guarantees are the most urgently needed, given the recent developments in automation and systems modeling. This thesis describes progress towards tractable formal verification of complex, high-dimensional systems via reachability analysis. The solutions presented involve three broad, complementary approaches:

1. **Structural solutions**. The behavior of multi-agent systems can be non-intuitive and difficult to monitor. In these cases, imposing various structural

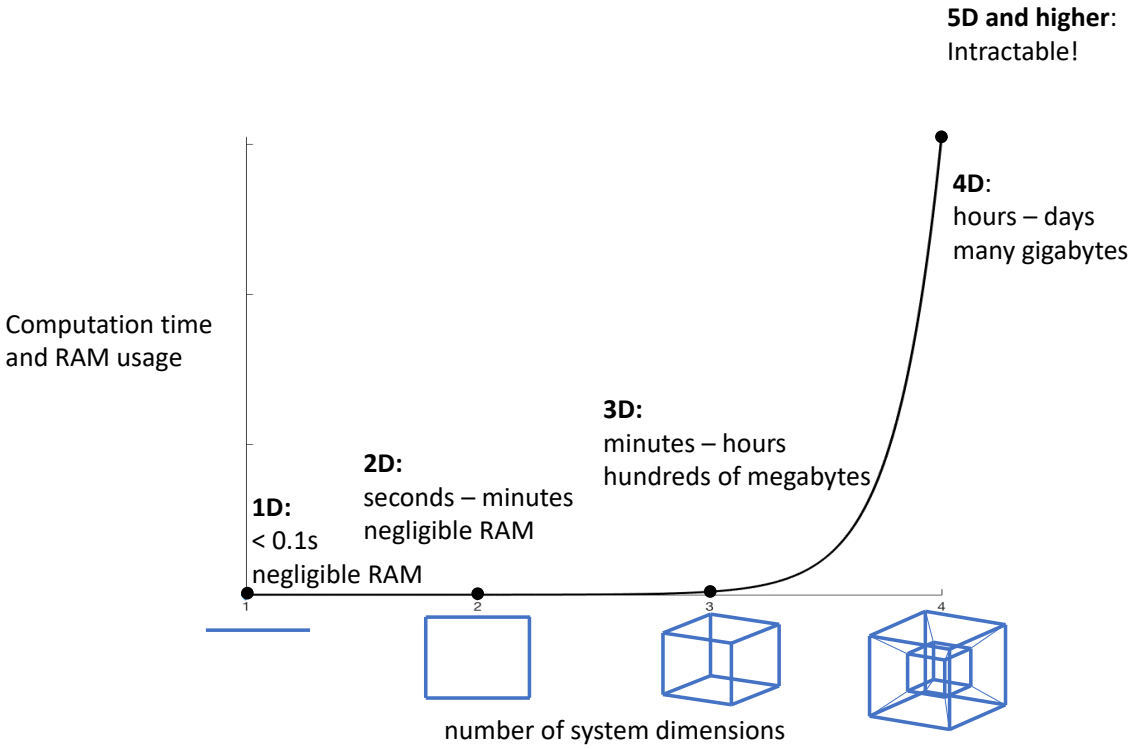


Figure 1.2: Illustration of computational complexity of HJ reachability.

assumptions, such as having air highways, to the system can significantly reduce problem complexity while allowing intuitive human participation.

2. **System decomposition**. For general high-dimensional systems, this thesis presents recently developed techniques to decompose a full dynamical system into multiple subsystems, reducing computation cost by many orders of magnitude and enabling previously intractable analyses.

3. **Frontiers of Reachability**. This thesis will also provide a first attempt of exploring the connection between machine learning and verification, as well as a first attempt of combining reachability analysis with fast planners in a modular fashion.

Chapter 2

Background

Hamilton-Jacobi (HJ) reachability analysis falls under the umbrella of optimal control problems and differential games, which 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 [16, 23, 60, 64, 111, 140, 142], and have been successfully applied to practical problems such as pairwise collision avoidance [111], aircraft in-flight refueling [54], vehicle platooning [35], and many others [18, 79]. With the recent growing interest in using safety-critical autonomous systems such as autonomous cars and unmanned aerial vehicles for civil purposes [**?**, 13, 19, 84, 116, 123], 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. 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 various definitions of the backward reachable tube (BRT), backward reachable set (BRS), forward reachable tube (FRT), or forward reachable set (FRS). This thesis mostly focuses on backward reachability. When the target set is a set of desired states, the BRT or 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 BRT or 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. Several frequently used formal definitions of BRSs and backward reachable tubes (BRTs) will be defined in Section 2.2.1.

In addition, HJ reachability is a powerful tool because BRTs and BRSs can be used for synthesizing both controllers that steer the system away from a set of unsafe states ("safety controllers") to guarantee safety, and controllers that steer the system into a set of goal states ("goal satisfaction controllers") to guarantee goal satisfaction. Unlike many formulations of reachability, the HJ formulations are flexible in terms of system dynamics, enabling the analysis of controlled nonlinear systems under disturbances. Furthermore, HJ reachability analysis is complemented by many numerical tools readily available to solve the associated HJ partial differential equation (PDE) [110, 122, 135]. However, the computation is done on a grid, making the problem complexity scale exponentially with the number of states, and therefore with the number of vehicles. Consequently, HJ reachability computations are intractable for large numbers of vehicles.

Besides the HJ formulation, there are many other methods related to reachability analysis. In general, none of the current methods, including the HJ formulation, simultaneously addresses all of the challenges that need to be overcome. For example, [58, 93] excel in determining whether system trajectories from a small set of initial conditions could potentially enter a set of unsafe states, but do not provide the BRS or BRT – the set of all initial states from which entering some target set is inevitable. Due to the challenges of computing BRSs and BRTs, the state-of-the-art methods need to make trade offs on different axes of considerations such as computational scalability, generality of system dynamics, existence of control and/or disturbance variables, and flexibility in representation of sets.

For example, the methods presented in [65, 71, 95, 96, 102] have had success in analyzing relatively high-dimensional affine systems using sets of pre-specified shapes, such as polytopes or hyperplanes. Other potentially less scalable methods are able to handle systems with the more complex dynamics [6, 39, 56, 65, 104]. Computational scalability varies among these different methods, with the most scalable methods requiring that the system dynamics do not involve control and disturbance variables. The work in [120] accounts for both control and disturbances, but is only applicable to linear systems. Methods that can account for general nonlinear systems such as [8] also sometimes represent sets using simple shapes such as polytopes, potentially sacrificing representation fidelity in favor of the other aspects mentioned earlier. Hamilton-Jacobi (HJ) formulations [16, 24, 105, 111] excel in handling general nonlinear dynamics, control and disturbance variables, and flexible set representations via a grid-based approach; however, these methods are the least computationally scalable. Still other methods make a variety of other assumptions to make desirable trade offs [42, 48, 73]. In addition, under some special scenarios, it may be possible to obtain small computational benefits while minimizing trade offs in other axes of consideration by exploiting system structure [31, 64, 89, 90, 114, 115].

In this chapter, we formalize the above notions and specialize in the HJ formulation although much of the content in the following chapters are agnostic to the reachability formulation.

2.1 System Dynamics

Let $s \in [-\infty, 0]$ be the time, $z \in \mathbb{R}^n$ be the system state, which evolves according to the ordinary differential equation (ODE)

$$\frac{dz(s)}{ds} = \dot{z}(s) = f(z(s), u(s), d(s)), u(s) \in \mathcal{U}, d(s) \in \mathcal{D} \tag{2.1}$$

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 and disturbance are respectively denoted by $u(s)$ and $d(s)$, with the control function $u(\cdot)$ and disturbance function $d(s)$ being respectively drawn from the set of measurable functions[1]:

$$u(\cdot) \in \mathbb{U}(t) = \{\phi : [t, 0] \to \mathcal{U} : \phi(\cdot) \text{ is measurable}\}$$
$$d(\cdot) \in \mathbb{D}(t) = \{\phi : [t, 0] \to \mathcal{D} : \phi(\cdot) \text{ is measurable}\}$$

where $\mathcal{U} \subset \mathbb{R}^{n_u}$ and $\mathcal{D} \subset \mathbb{R}^{n_d}$ are compact, and $t < 0$. 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, bounded, and Lipschitz continuous in[2] z for fixed u and d. Therefore, given $u(\cdot) \in \mathbb{U}, d(\cdot) \in \mathbb{D}$, there exists a unique trajectory solving (2.1) [41]. We will denote solutions, or trajectories of (2.1) starting from state z at time t under control $u(\cdot)$ and disturbance $d(\cdot)$ as $\xi(s; z, t, u(\cdot), d(\cdot)) : [t, 0] \to \mathbb{R}^n$. ξ satisfies (2.1) with an initial condition almost everywhere:

$$\begin{aligned} \frac{d}{ds}\xi(s; z, t, u(\cdot), d(\cdot)) &= f(\xi(s; z, t, u(\cdot), d(\cdot)), u(s), d(s)) \\ \xi(t; z, t, u(\cdot), d(\cdot)) &= z \end{aligned} \tag{2.2}$$

For time-invariant system dynamics, the time variables in trajectories can be shifted by any constant τ:

$$\xi(s; z, t, u(\cdot), d(\cdot)) = \xi(s + \tau; z, t + \tau, u(\cdot), d(\cdot)), \forall z \in \mathbb{R}^n \tag{2.3}$$

The interaction between disturbance and control is modeled using a differential game, as in [111]. We define a strategy for the disturbance as the mapping $\gamma : \mathcal{U} \to \mathcal{D}$

[1] A function $f : X \to Y$ between two measurable spaces (X, Σ_X) and (Y, Σ_Y) is said to be measurable if the preimage of a measurable set in Y is a measurable set in X, that is: $\forall V \in \Sigma_Y, f^{-1}(V) \in \Sigma_X$, with Σ_X, Σ_Y σ-algebras on X,Y.

[2] The notation (s) from variables such as z and u referring to function values will be omitted when the context is clear.

that determines a disturbance signal that reacts to the control signal based on the state. The mapping γ is drawn from only non-anticipative strategies $\gamma \in \Gamma(t)$, and write $d(\cdot) = \gamma[u](\cdot)$. Non-anticipative strategies is defined as follows:

$$\begin{aligned}\gamma \in \Gamma(t) := \{\mathcal{K} : \mathbb{U}(t) \to \mathbb{D}(t), u(r) = \hat{u}(r) \text{ for almost every } r \in [t, s] \\ \Rightarrow \mathcal{K}[u](r) = \mathcal{K}[\hat{u}](r) \text{ for almost every } r \in [t, s]\}\end{aligned} \tag{2.4}$$

Roughly speaking, this means that the disturbance may only react to current and past control signals. A detailed discussion of this information pattern can be found in [111].

2.2 Hamilton-Jacobi Reachability Analysis

In HJ reachability, we begin with the system dynamics given by an ordinary differential equation (ODE), and a target set which represents the set of unsafe states. We then solve the HJ equation to obtain various desired forms of reachable sets or tubes, which could represent states leading to danger. To avoid danger, the system may apply any control until it reaches the boundary of a reachable set. At the boundary, applying the optimal safety controller would guarantee avoidance. We present the most commonly used definitions in this section, and more specialized definitions in their respective sections.

2.2.1 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 $|t|$. 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]:

[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.

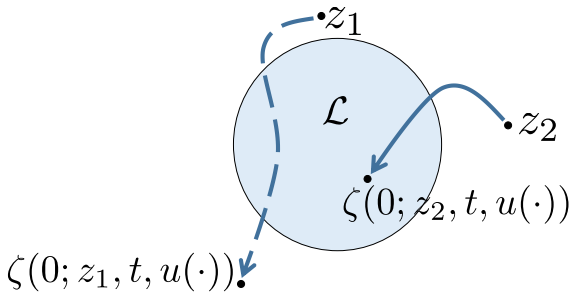


Figure 2.1: The difference between a BRS and a BRT. The dashed trajectory starts at z_1 and passes through $\mathcal{L}$ during the period $[t, 0]$, but exits $\mathcal{L}$ by the end of the time period. Therefore the z_1 is in the BRT, but not in the BRS. The solid trajectory starting from z_2 is in $\mathcal{L}$ at the end of the time period. Therefore, z_2 is in both the BRS and the BRT.

Definition 1 ***Maximal BRS***.

$$\mathcal{R}(t) = \{z : \forall \gamma \in \Gamma(t), \exists u(\cdot) \in \mathbb{U}, \xi(0; z, t, u(\cdot), \gamma[u](\cdot)) \in \mathcal{L}\}$$

Definition 2 ***Minimal BRS***.

$$\mathcal{A}(t) = \{z : \forall u(\cdot) \in \mathbb{U}, \xi(0; z, t, u(\cdot), \gamma[u](\cdot)) \in \mathcal{L}\}$$

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 any time* during the time horizon of duration $|t|$. Figure 2.1 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 any time within a specified time horizon. Formally, the two definitions of BRTs are as follows:

Definition 3 ***Maximal BRT***.

$$\bar{\mathcal{R}}(t) = \{z : \exists u(\cdot) \in \mathbb{U}, \exists s \in [t, 0], \xi(s; z, t, u(\cdot), \gamma[u](\cdot)) \in \mathcal{L}\}$$

Definition 4 ***Minimal BRT***.

$$\bar{\mathcal{A}}(t) = \{z : \forall u(\cdot) \in \mathbb{U}, \exists s \in [t, 0], \xi(s; z, t, u(\cdot), \gamma[u](\cdot)) \in \mathcal{L}\}$$

The terms "maximal" and "minimal" refer to the role of the optimal control [109]. 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.

2.2.2 Computing Reachable Sets and Tubes

HJ formulations such as [16, 24, 111, 142] cast the reachability problem as an optimal control or differential game problem and directly compute BRSs and BRTs in the full state space of the system. The numerical methods (such as [109]) for obtaining the optimal solution all involve solving an HJ PDE on a grid that represents a discretization of the state space. For the time-invariant case, we now summarize necessary details related to the HJ PDEs and what their solutions represent in terms of the cost function of the corresponding optimization problem.

Let the target set $\mathcal{L} \subseteq \mathbb{R}^n$ be represented by the implicit surface function $l(z)$ as $\mathcal{L} = \{z : l(z) \leq 0\}$. Such a function always exists since we can choose $l(\cdot)$ to be the signed distance function from $\mathcal{L}$. Consider the optimization problem

$$V_{\mathcal{R}}(t,z) = \sup_{\gamma[u](\cdot)\in\Gamma(t)} \inf_{u(\cdot)\in\mathbb{U}} l(\xi(0;z,t,u(\cdot),\gamma[u](\cdot))) \quad \text{subject to (2.2)} \tag{2.5}$$

with the optimal control being given by

$$u^*_{\mathcal{R}}(\cdot) = \arg \sup_{\gamma[u](\cdot)\in\Gamma(t)} \inf_{u(\cdot)\in\mathbb{U}} l(\xi(0;z,t,u(\cdot),\gamma[u](\cdot))) \tag{2.6}$$

The value function $V_{\mathcal{R}}(t,z)$ is the implicit surface function representing the maximal BRS: $\mathcal{R}(t) = \{z : V_{\mathcal{R}}(t,z) \leq 0\}$.

Similarly, consider the optimization problem

$$V_{\mathcal{A}}(t,z) = \inf_{\gamma[u](\cdot)\in\Gamma(t)} \sup_{u(\cdot)\in\mathbb{U}} l(\xi(0;z,t,u(\cdot),\gamma[u](\cdot))) \quad \text{subject to (2.2)} \tag{2.7}$$

with optimal control

$$u^*_{\mathcal{A}}(\cdot) = \arg \inf_{\gamma[u](\cdot)\in\Gamma(t)} \max_{u(\cdot)\in\mathbb{U}} l(\xi(0;z,t,u(\cdot),\gamma[u](\cdot))) \tag{2.8}$$

Analogously, we also have $\mathcal{A}(t) = \{z : V_{\mathcal{A}}(t,z) \leq 0\}$.

Fig. 2.2 provides a simple 2D example demonstrating the relationship between the target set, implicit surface function, BRS, and value function.

The value functions $V_{\mathcal{R}}(t,z)$ and $V_{\mathcal{A}}(t,z)$ are the viscosity solution [44, 45] of the HJ PDE

$$\begin{aligned} D_s V(s,z) + H(z, \nabla V(s,z)) = 0, \quad s \in [t,0] \\ V(0,z) = l(z) \end{aligned} \tag{2.9}$$

The Hamiltonian $H(z,\lambda)$ depends on the system dynamics and on the optimal control problem. For example, for the optimal control problem (2.5), the Hamiltonian is given by

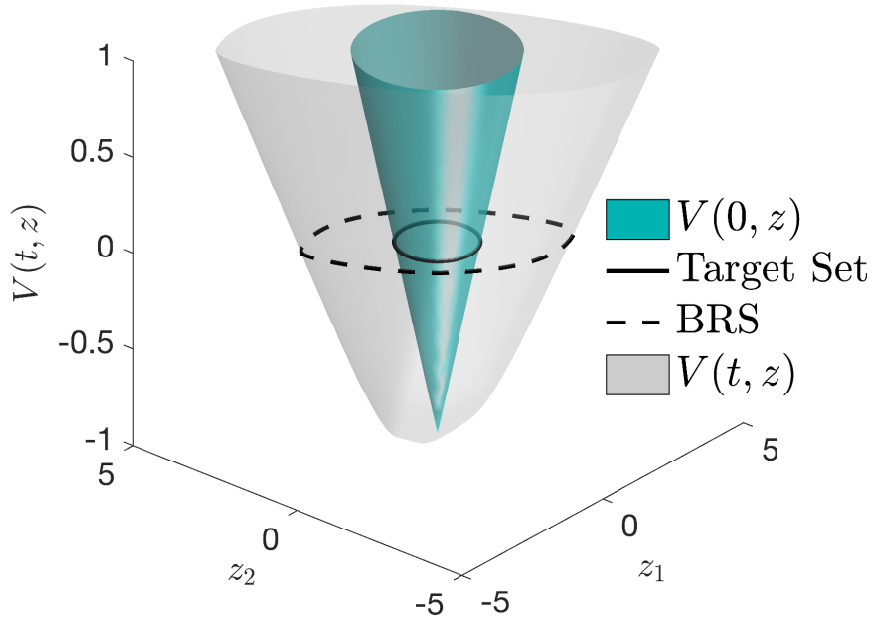


Figure 2.2: A 2D illustration of HJ reachability. The boundary of the target set $\mathcal{L}$ is shown as the solid black line. The blue surface represents the implicit surface function $l(z)$, which by (2.9) is equivalent to $V_{\pm}(0, z)$. The light gray surface shows the value function $V_{\pm}(t, z)$ at some $t < 0$. The corresponding BRS is the zero sub-level set of this function; the boundary of the BRS is shown as the dashed black line.

$$H(z,\lambda) = \min_{u\in\mathcal{U}} \max_{d\in\mathcal{D}} \lambda \cdot f(z,u,d) \tag{2.10}$$

Once the value function $V_{\mathcal{R}}$ is computed, the optimal control (2.6) can be obtained by the expression

$$u^*_{\mathcal{R}}(s) = \arg\min_{u\in\mathcal{U}} \max_{d\in\mathcal{D}} \nabla V_{\mathcal{R}}(s,z) \cdot f(z,u,d) \tag{2.11}$$

Similarly, for the optimal control problem (2.7), the Hamiltonian is given by

$$H(s,z,\lambda) = \max_{u\in\mathcal{U}} \min_{d\in\mathcal{D}} \lambda \cdot f(z,u,d) \tag{2.12}$$

and the optimal control is given by

$$u^*_{\mathcal{A}}(s) = \arg\max_{u\in\mathcal{U}} \min_{d\in\mathcal{D}} \nabla V_{\mathcal{A}}(s,z) \cdot f(z,u,d) \tag{2.13}$$

Furthermore, by the dynamic programming principle, the optimal value on optimal trajectories must be constant:

$$\begin{aligned} V_{\mathcal{R}}(s,\xi(s;z,\tau,u^*_{\mathcal{R}}(\cdot)) &= V_{\mathcal{R}}(\tau,z)\ \forall \tau, s\in[t,0] \\ V_{\mathcal{A}}(s,\xi(s;z,\tau,u^*_{\mathcal{A}}(\cdot)) &= V_{\mathcal{A}}(\tau,z)\ \forall \tau, s\in[t,0] \end{aligned} \tag{2.14}$$

For the BRT, several equivalent formulations have been proposed. For example, one can modify the value function to keep track of the minimum value of the function $l(\cdot)$ that the system trajectory achieves over some time horizon, so that (2.5) and (2.7) respectively become

$$\begin{aligned} \bar{V}_{\mathcal{R}}(t,z) &= \max_{\gamma[u](\cdot)\in\Gamma(t)} \min_{u(\cdot)\in\mathbb{U}} \min_{s\in[t,0]} l(\xi(0;z,t,u(\cdot),\gamma[u](\cdot))) \\ \bar{V}_{\mathcal{A}}(t,z) &= \min_{\gamma[u](\cdot)\in\Gamma(t)} \max_{u(\cdot)\in\mathbb{U}} \min_{s\in[t,0]} l(\xi(0;z,t,u(\cdot),\gamma[u](\cdot))) \end{aligned} \tag{2.15}$$

The reader is encouraged to read the details of this formulation in [64] which contains a very general time-varying reach-avoid framework, or other formulations such as [16, 24, 105, 111]. However, for this thesis it suffices to note that $\bar{V}_{\mathcal{R}}$ and $\bar{V}_{\mathcal{A}}$ are the viscosity solution of the following HJ variational inequality:

$$\begin{aligned} \min\{D_s\bar{V}(s,z) + H(z,\nabla\bar{V}(s,z)), l(z) - \bar{V}(s,z)\} = 0, \quad s\in[t,0] \\ \bar{V}(0,z) = l(z) \end{aligned} \tag{2.16}$$

where $H(z,\lambda)$ is given by (2.10), (2.12) for $\bar{V}_{\mathcal{R}}, \bar{V}_{\mathcal{A}}$ respectively.

Chapter 3

Unmanned Airspace Infrastructure

Unmanned aerial vehicles (UAVs) have in the past been mainly used for military operations [74, 139]; however, recently there has been an immense surge of interest in using UAVs for civil applications. Through projects such as Amazon Prime Air [10] and Google Project Wing [137], companies are looking to send UAVs into the airspace to not only deliver commercial packages, but also for important tasks such as aerial surveillance, emergency supply delivery, videography, and search and rescue [123]. In the future, the use of UAVs is likely to become more and more prevalent.

As a rough estimate, suppose in a city of 2 million people, each person requests a drone delivery every 2 months on average and each delivery requires a 30-minute trip for a UAV. This would equate to thousands of UAVs simultaneously in the air just from package delivery services. Applications of UAVs extend beyond package delivery; they can also be used, for example, to provide supplies or to respond to disasters in areas that are difficult to reach but require prompt response [13, 49]. As a result, government agencies such as the Federal Aviation Administration (FAA) and National Aeronautics and Space Administration (NASA) are also investigating unmanned aerial systems (UAS) traffic management (UTM) in order to prevent collisions among potentially numerous UAVs [84, 116, 123].

In this chapter, we present two different approaches for managing the airspace. First, in Section 3.1, we propose the concept of air highways, and provide a method for automatically designing an air highway network on which UAVs would fly in platoons. In Section 3.2, we propose the sequential trajectory planning method, potentially as a last-mile solution for off-the-highway management. Through large scale simulations, we provide a case study of our algorithm over the San Francisco Bay Area.

3.1 Air Highways and Unmanned Aerial Platoons

This section is an adaptation of the paper in [34].

In order to accommodate potentially thousands of vehicles simultaneously flying

in the air, additional structure is needed to allow for tractable analysis and intuitive monitoring by human beings. An air highway system on which platoons of vehicles travel accomplishes both goals. However, many details of such a concept need to be addressed. Due to the flexibility of placing air highways compared to building ground highways in terms of highway location, even the problem of air highway placement can be a daunting task. To address this, in the first part of this section, we propose a flexible and computationally efficient method based on [135] to perform optimal air highway placement given an arbitrary cost map that captures the desirability of having UAVs fly over any geographical location. We demonstrate our method using the San Francisco Bay Area as an example. Once air highways are in place, platoons of UAVs can then fly in fixed formations along the highway to get from origin to destination. The air highway structure greatly simplifies safety analysis, while at the same time allows intuitive human participation in unmanned airspace management.

A considerable body of work has been done on the platooning of ground vehicles [87]. For example, [108] investigated the feasibility of vehicle platooning in terms of tracking errors in the presence of disturbances, taking into account complex nonlinear dynamics of each vehicle. [75] explored several control techniques for performing various platoon maneuvers such as lane changes, merge procedures, and split procedures. In [100], the authors modeled vehicles in platoons as hybrid systems, synthesized safety controllers, and analyzed throughput. Reachability analysis was used in [66] to analyze a platoon of two trucks in order to minimize drag by minimizing the following distance while maintaining collision avoidance safety guarantees. Finally, [129] provided a method for guaranteeing string stability and eliminating accordion effects for a heterogeneous platoon of vehicles with linear time-invariant dynamics.

Previous analyses of a large number of vehicles typically do not provide safety and goal satisfaction guarantees to the extent that HJ reachability does; however, HJ reachability typically cannot be used to tractably analyze a large number of vehicles. In the second part of this section, we propose organizing UAVs into platoons, which provides a structure that allows pairwise safety guarantees from HJ reachability to better translate to safety guarantees for the whole platoon. With respect to platooning, we first propose a hybrid systems model of UAVs in platoons to establish the modes of operation needed for our platooning concept. Then, we show how reachability-based controllers can be synthesized to enable UAVs to successfully perform mode switching, as well as prevent dangerous configurations such as collisions. Finally, we show several simulations to illustrate the behavior of UAVs in various scenarios.

Overall, this section is not meant to provide an exhaustive solution to the unmanned airspace management problem. Instead, this section illustrates that the computation intractability of HJ reachability can be overcome using an air highway structure with UAVs flying in platoons. In addition, the results are intuitive, which can facilitate human participation in managing the airspace. Although many chal-

lenges not addressed in this section still need to be overcome, this section can provide a starting point for future research in large-scale UASs with safety and goal-satisfaction guarantees.

3.1.1 Air Highways

We consider air highways to be virtual highways in the airspace on which a number of UAV platoons may be present. UAVs seek to arrive at some desired destination starting from their origin by traveling along a sequence of air highways. Air highways are intended to be the common pathways for many UAV platoons, whose members may have different origins and destinations. By routing platoons of UAVs onto a few common pathways, the airspace becomes more tractable to analyze and intuitive to monitor. The concept of platoons will be proposed in Section 3.1.3; in this section, we focus on air highways.

Let an air highway be denoted by the continuous function $\mathbb{H} : [0, 1] \to \mathbb{R}^2$. Such a highway lies in a horizontal plane of fixed altitude, with start and end points given by $\mathbb{H}(0) \in \mathbb{R}^2$ and $\mathbb{H}(1) \in \mathbb{R}^2$ respectively. For simplicity, we assume that the highway segment is a straight line segment, and the parameter s indicates the position in some fixed altitude as follows: $\mathbb{H}(s) = \mathbb{H}(0) + s(\mathbb{H}(1) - \mathbb{H}(0))$. To each highway, we assign a speed of travel $v_{\mathbb{H}}$ and specify the direction of travel to be the direction from $\mathbb{H}(0)$ to $\mathbb{H}(1)$, denoted using a unit vector $\hat{d} = \frac{\mathbb{H}(1)-\mathbb{H}(0)}{\|\mathbb{H}(1)-\mathbb{H}(0)\|_2}$. As we will show in Section 3.1.3, UAVs use simple controllers to track the highway.

Air highways must not only provide structure to make the analysis of a large number of vehicles tractable, but also allow vehicles to reach their destinations while minimizing any relevant costs to the vehicles and to the surrounding regions. Such costs can for example take into account people, assets on the ground, and manned aviation, entities to which UAVs pose the biggest risks [123]. Thus, given an origin-destination pair (eg. two cities), air highways must connect the two points while potentially satisfying other criteria. In addition, optimal air highway locations should ideally be able to be recomputed in real-time when necessary in order to update airspace constraints on-the-fly, in case, for example, airport configurations change or certain airspaces have to be closed [123]. With this in mind, we now define the air highway placement problem, and propose a simple and fast way to approximate its solution that allows for real-time recomputation. Our solution based on solving the Eikonal equation can be thought of as converting a cost map over a geographic area in continuous space into a discrete graph whose nodes are waypoints joined by edges which are the air highways.

Note that the primary purpose of this section is to provide a method for the real-time placement of air highways. The specifics of determining the cost map based on population density, geography, weather forecast information, etc., as well as the criteria for when air highway locations need to be updated, is beyond the scope of this section.

In addition, if vehicles in the airspace are far away from each other, it may be reasonable for all vehicles to fly in an unstructured manner. As long as multiple-way conflicts do not occur, pairwise collision avoidance maneuvers would be sufficient to ensure safety. Unstructured flight is likely to result in more efficient trajectories for each individual vehicle. However, whether multiple-way conflicts occur cannot be predicted ahead of time, and are not guaranteed to be resolvable when they occur. By organizing vehicles into platoons, the likelihood of multiple-way conflicts is vastly reduced. Structured flight is in general less efficient for the individual vehicle, and this loss of efficiency can be thought of as a cost incurred by the vehicles in order ensure higher levels of safety.

In general, there may be many different levels of abstractions in the airspace. For larger regions such as cities, air highways may prove beneficial, and for a small region such as a neighborhood, perhaps unstructured flight is sufficiently safe. Further research is needed to better understand parameters such as the density of vehicles above which unstructured flight is no longer manageable, and other details like platoon size.

3.1.2 The Air Highway Placement Problem

Consider a map $c : \mathbb{R}^2 \to \mathbb{R}$ which defines the cost $c(p)$ incurred when a UAV flies over the position $p = (p_x, p_y) \in \mathbb{R}^2$. Given any position p, a large value of $c(p)$ indicates that the position p is costly or undesirable for a UAV to fly over. Locations with high cost could, for example, include densely populated areas and areas around airports. In general, the cost map $c(\cdot)$ may be used to model cost of interference with commercial airspaces, cost of accidents, cost of noise pollution, risks incurred, etc., and can be flexibly specified by government regulation bodies.

Let p^o denote an origin point and p^d denote a destination point. Consider a sequence of highways $\mathbb{S}_N = \{\mathbb{H}_1, \mathbb{H}_2, \ldots, \mathbb{H}_N\}$ that satisfies the following:

$$\begin{aligned} \mathbb{H}_1(0) &= p^o \\ \mathbb{H}_i(1) &= \mathbb{H}_{i+1}(0), i = 0, 1, \ldots, N-1 \\ \mathbb{H}_N(1) &= p^d \end{aligned} \tag{3.1}$$

The interpretation of the above conditions is that the start point of first highway is the origin, the end point of a highway is the start point of the next highway, and the end point of last highway is the destination. The highways $\mathbb{H}_1, \ldots, \mathbb{H}_N$ form a sequence of waypoints for a UAV starting at the origin p^o to reach its destination p^d.

Given only the origin point p^o and destination point p^d, there are an infinite number of choices for a sequence of highways that satisfy (3.1). However, if one takes into account the cost of flying over any position p using the cost map $c(\cdot)$, we arrive at the air highway placement problem:

$$\min_{\mathbb{S}_N, N} \left\{ \left(\sum_{i=1}^{N} \int_0^1 c(\mathbb{H}_i(s))ds \right) + R(N) \right\} \quad \text{subject to (3.1)} \tag{3.2}$$

where $R(\cdot)$ is a regularizer, such as $R(N) = N^2$.

The interpretation of (3.2) is that we consider air highways to be line segments of constant altitude over a region, and UAV platoons travel on these air highways to get from some origin to some destination. Any UAV flying on a highway over some position p incurs a cost of $c(p)$, so that the total cost of flying from the origin to the destination is given by the summation in (3.2). The air highway placement problem minimizes the cumulative cost of flying from some origin p^o to some destination p^d along the sequence of highways $\mathbb{S}_N = \{\mathbb{H}_1, \mathbb{H}_2, \ldots, \mathbb{H}_N\}$. The regularization term $R(N)$ is used to prevent N from being arbitrarily large.

3.1.2.1 The Eikonal Equation – Cost-Minimizing Path

Let $s_0, s_1 \in \mathbb{R}$, and let $\mathbb{P} : [s_0, s_1] \to \mathbb{R}^2$ be a path starting from an origin point $p^o = \mathbb{P}(s_0)$ and ending at a destination point $p^d = \mathbb{P}(s_1)$. Note that the sequence $\mathbb{S}_N$ in (3.2) is a piecewise affine example of a path $\mathbb{P}(s), s \in [s_0, s_1]$; however, a path $\mathbb{P}$ that is not piecewise affine cannot be written as a sequence of highways $\mathbb{S}_N$.

More concretely, suppose a UAV flies from an origin point p^o to a destination point p^d along some path $\mathbb{P}(s)$ parametrized by s. Then, $\mathbb{P}(s_0) = p^o$ would denote the origin, and $\mathbb{P}(s_1) = p^d$ would denote the destination. All intermediate s values denote the intermediate positions of the path, i.e. $\mathbb{P}(s) = p(s) = (p_x(s), p_y(s))$.

Consider the cost map $c(p_x, p_y)$ which captures the cost incurred for UAVs flying over the position $p = (p_x, p_y)$. Along the entire path $\mathbb{P}(s)$, the cumulative cost $C(\mathbb{P})$ is incurred. Define C as follows:

$$C(\mathbb{P}) = \int_{s_0}^{s_1} c(\mathbb{P}(s))ds \tag{3.3}$$

For an origin-destination pair, we would like to find the path such that the above cost is minimized. More generally, given an origin point p^o, we would like to compute the function V representing the optimal cumulative cost for any destination point p^d:

$$\begin{aligned} V(p^d) &= \min_{\mathbb{P}(\cdot), \mathbb{P}(s_1) = p^d} C(\mathbb{P}) \\ &= \min_{\mathbb{P}(\cdot), \mathbb{P}(s_1) = p^d} \int_{s_0}^{s_1} c(\mathbb{P}(s))ds \end{aligned} \tag{3.4}$$

It is well known that the viscosity solution [45] to the Eikonal equation (3.5) precisely computes the function $V(p^d)$ given the cost map c [9, 135]. Note that a single function characterizes the minimum cost from an origin p^o to *any* destination

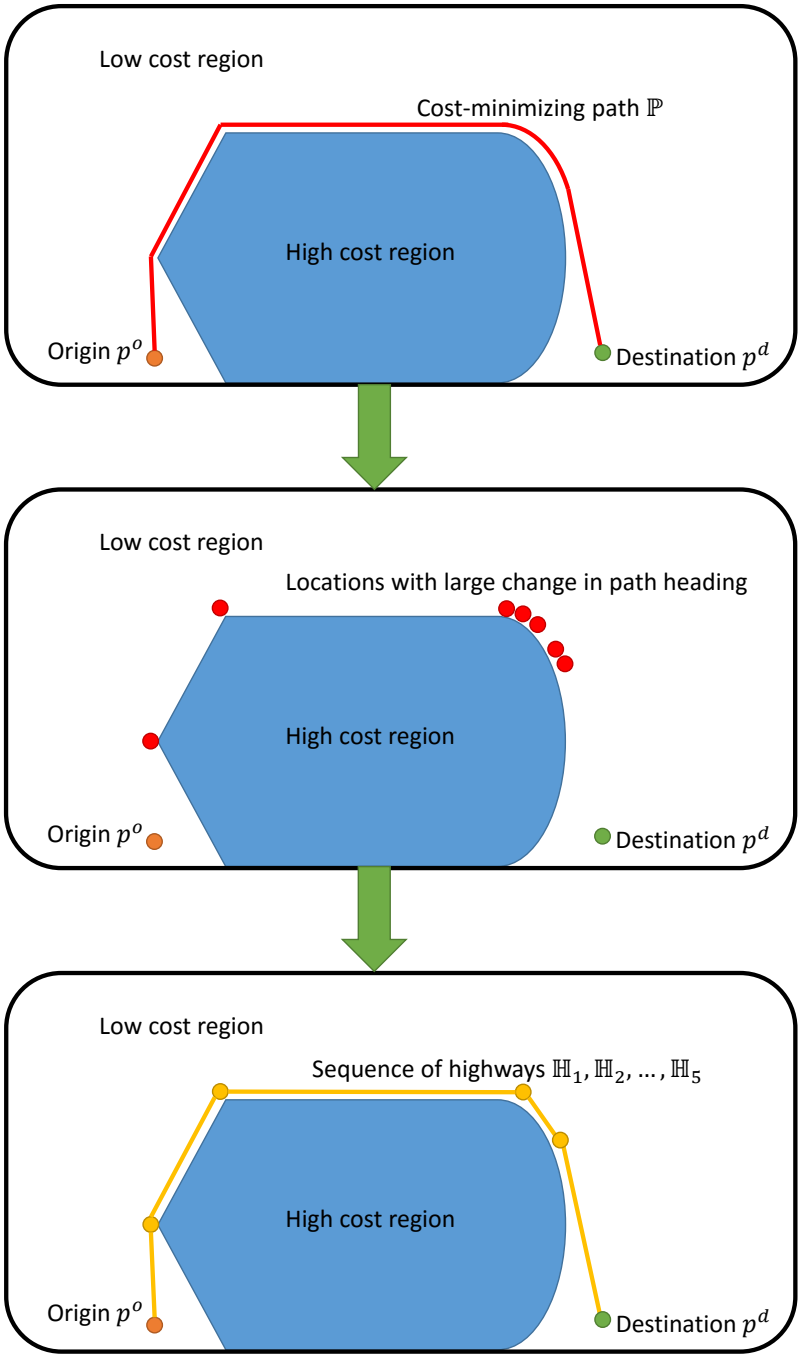


Figure 3.1: Illustration of the air highway placement procedure.

p^d. Once V is found, the optimal path $\mathbb{P}$ between p^o and p^d can be obtained via gradient descent.

$$\begin{aligned} c(p)|\nabla V(p)| &= 1 \\ V(p^o) &= 0 \end{aligned} \tag{3.5}$$

The Eikonal equation (3.5) can be efficiently computed numerically using the fast marching method [135]; each computation takes on the order of merely a second.

Note that (3.4) can be viewed as a relaxation of the air highway placement problem defined in (3.2). Unlike (3.2), the relaxation (3.4) can be quickly solved using currently available numerical tools. Thus, we first solve the approximate air highway placement problem (3.4) by solving (3.5), and then post-process the solution to (3.4) to obtain an approximation to (3.2).

Given a single origin point p^o, the optimal cumulative cost function $V(p^d)$ can be computed. Suppose M different destination points $p_i^d, i = 1, \ldots, M$ are chosen. Then, M different optimal paths $\mathbb{P}_i, i = 1, \ldots, M$ are obtained from V.

3.1.2.2 From Paths to Waypoints

Each of the cost-minimizing paths $\mathbb{P}_i$ computed from the solution to the Eikonal equation consists of a closely-spaced set of points. Each path $\mathbb{P}_i$ is an approximation to the sequence of highways $\mathbb{S}_{N_i}^i = \{\mathbb{H}_j^i\}_{i=1,j=1}^{i=M,j=N_i}$ defined in (3.2), but now indexed by the corresponding path index i.

For each path $\mathbb{P}_i$, we would like to sparsify the points on the path to obtain a collection of waypoints, $\mathcal{W}_{i,j}, j = 1, \ldots, N_i + 1$, which are the end points of the highways:

$$\begin{aligned} \mathbb{H}_j^i(0) &= \mathcal{W}_{i,j}, \\ \mathbb{H}_j^i(1) &= \mathcal{W}_{i,j+1}, \\ j &= 1, \ldots, N_i \end{aligned} \tag{3.6}$$

There are many different ways to do this, and this process will not be our focus. However, for illustrative purposes, we show how this process may be started. We begin by noting the path's heading at the destination point. We add to the collection of waypoints the first point on the path at which the heading changes by some threshold θ_C, and repeat this process along the entire path.

If there is a large change in heading within a small section of the cost-minimizing path, then the collection of points may contain many points which are close together. In addition, there may be multiple paths that are very close to each other (in fact, this behavior is desirable), which may contribute to cluttering the airspace with too many waypoints. To reduce clutter, one could cluster the points. Afterwards, each cluster of points can be replaced by a single point located at the centroid of the cluster.

To the collection of points resulting from the above process, we add the origin and destination points. Repeating the entire process for every path, we obtain waypoints for all the cost-minimizing paths under consideration. Figure 3.1 summarizes the entire air highway placement process, including our example of how the closely-spaced set of points on a path can be sparsified.

3.1.2.3 Simulation Results

To illustrate our air highway placement proposal, we used the San Francisco Bay Area as an example, and classified each point on the map into four different regions: "regions around airports", "highly populated cities", "water", and "other". Each region has an associated cost, reflecting the desirability of flying a vehicle over an area in the region. In general, these costs can be arbitrary and determined by government regulation agencies. For illustration purposes, we assumed the following categories and costs:

- Region around airports: $c_{\text{airports}} = b$,
- Cities: $c_{\text{cities}} = 1$,
- Water: $c_{\text{water}} = b^{-2}$,
- Other: $c_{\text{other}} = b^{-1}$.

This assumption assigns costs in descending order to the categories "regions around airports", "cities", "other", and "water". Flying a UAV in each category is more costly by a factor of b compared to the next most important category. The factor $b > 1$ is a tuning parameter that we adjusted to vary the relative importance of the different categories, and we used $b = 4$ in the figures below.

Figure 3.2 shows the San Francisco Bay Area geographic map, cost map, cost-minimizing paths, and contours of the value function V. The region enclosed by the black boundary represents "region around airports", which have the highest cost. The dark blue, yellow, and light blue regions represent the "cities", the "water", and the "other"' categories, respectively. We assumed that the origin corresponds to the city "Concord", and chose a number of other major cities as destinations.

A couple of important observations can be made here. First, the cost-minimizing paths to the various destinations in general overlap, and only split up when they are very close to entering their destination cities. This is intuitively desirable because having overlapping cost-minimizing paths keeps the number of distinct air highways low. Secondly, the contours, which correspond to level curves of the value function, have a spacings corresponding to the cost map: the spacings are large in areas of low cost, and vice versa. This provides insight into the placement of air highways to destinations that were not shown in this example.

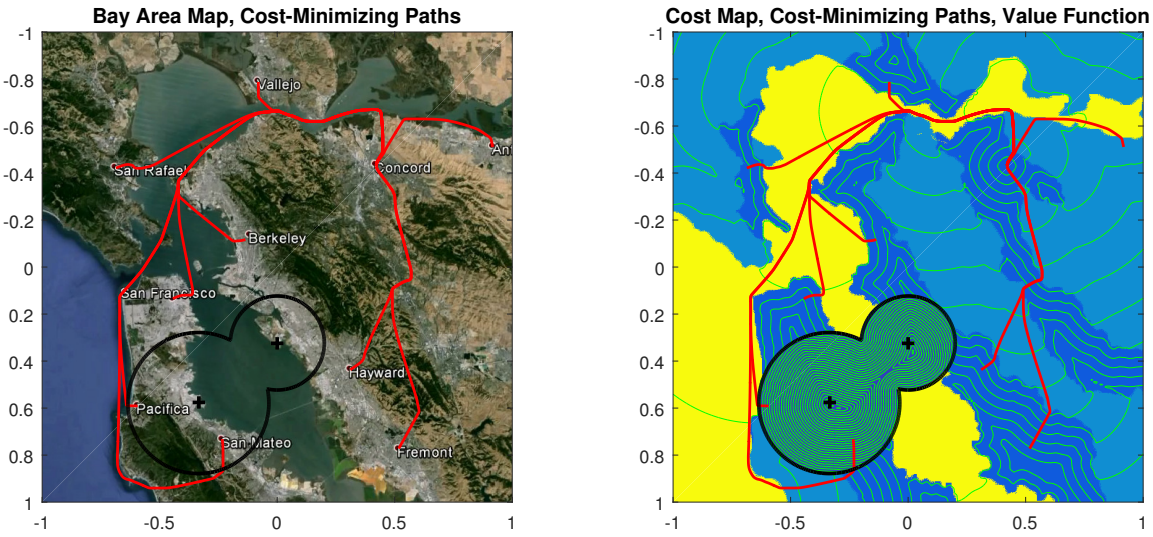


Figure 3.2: Cost-minimizing paths computed by the Fast Marching Method based on the assumed cost map of the San Francisco Bay Area.

Figure 3.3 shows the result of converting the cost-minimizing paths to a small number of waypoints. The left plot shows the waypoints, interpreted as the start and end points of air highways, over a white background for clarity. The right plot shows these air highways over the map of the Bay Area. Note that we could have gone further to merge some of the overlapping highways. However, the purpose of this section is to illustrate the natural occurrence of air highways from cost-minimizing paths; post-processing of the cost-minimizing paths, which serve as a guide for defining air highways, is not our focus.

3.1.2.4 Real-Time Highway Location Updates

Since (3.5) can be solved in approximately 1 second, the air highway placement process can be redone in real-time if the cost map changes at a particular time. This flexibility can be useful in any situation in which unforeseen circumstances could cause a change in the cost of a particular region of the airspace. For example, accidents or disaster response vehicles may result in an area temporarily having a high cost. On the other hand, depending on for instance the time of day, it may be most desirable to fly in different regions of the airspace, resulting in those regions temporarily having a low cost.

3.1.3 Unmanned Aerial Vehicle Platooning

Air highways exhibiting trunk routes that separate near destinations motivate the use of platoons which fly on these highways. The air highway structure along with

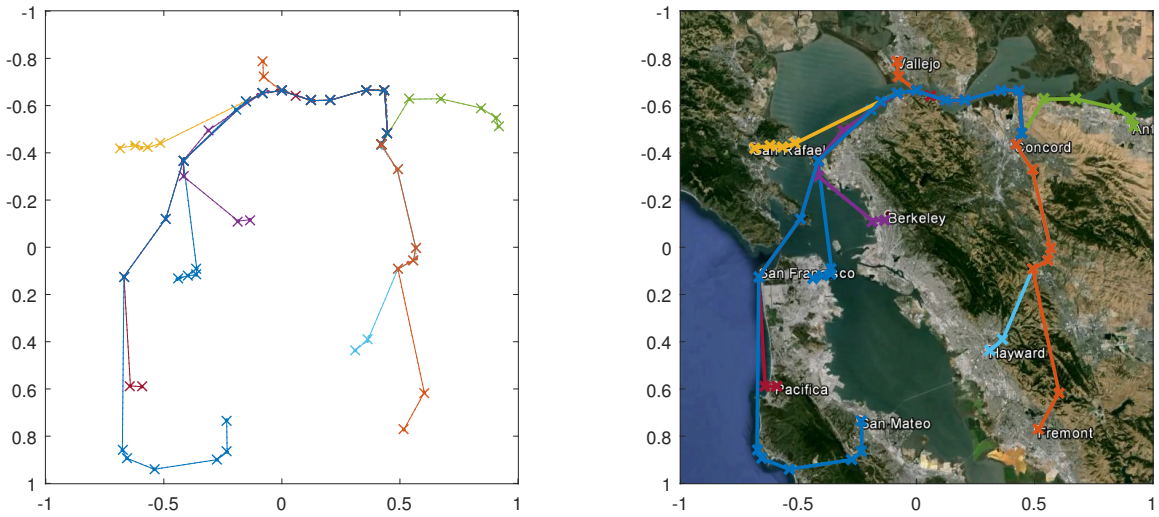


Figure 3.3: Results of conversion from cost-minimizing paths to highway way points.

the UAV platooning concept together enable the use of reachability to analyze safety and goal satisfaction properties. The structure reduces the likelihood of multiple-way conflicts, and makes pairwise analysis more indicative of the joint safety of all UAVs. In addition to reducing complexity, the proposed structure is intuitive, and allows human participation in the monitoring and management of the unmanned airspace.

Organizing UAVs into platoons implies that the UAVs cannot fly in an unstructured way, and must have a restricted set of controllers or maneuvers depending on the UAV's role in the airspace. To model UAVs flying in platoons on air highways, we propose a hybrid system whose modes of operations describe a UAV's role in the highway structure. For the hybrid system model, reachability analysis is used to enable successful and safe operation and mode transitions.

3.1.3.1 UAVs in Platoons

Vehicle Dynamics: The techniques we present here do not depend on the dynamics of the vehicles, as long as their dynamics are known. However, for concreteness, we assume that the UAVs are quadrotors that fly at a constant altitude under non-faulty circumstances. For the quadrotor, we use a simple model in which the x and y dynamics are double integrators:

$$\begin{aligned} \dot{p}_x &= v_x \\ \dot{p}_y &= v_y \\ \dot{v}_x &= u_x \\ \dot{v}_y &= u_y \\ |u_x|, |u_y| &\leq u_{\max} \end{aligned} \tag{3.7}$$

where the state $x = (p_x, v_x, p_y, v_y) \in \mathbb{R}^4$ represents the quadrotor's position in the x-direction, its velocity in the x-direction, and its position and velocity in the y-direction, respectively. The control input $u = (u_x, u_y) \in \mathbb{R}^2$ consists of the acceleration in the x- and y- directions. For convenience, we will denote the position and velocity $p = (p_x, p_y), v = (v_x, v_y)$, respectively.

In general, the problem of collision avoidance among N vehicles cannot be tractably solved using traditional dynamic programming approaches because the computation complexity of these approaches scales exponentially with the number of vehicles. Thus, in our present work, we will consider the situation where UAVs travel on air highways in platoons, defined in the following sections. The structure imposed by air highways and platooning enables us to analyze the safety and goal satisfaction properties of the vehicles in a tractable manner.

Vehicles as Hybrid Systems: We model each vehicle as a hybrid system [100, 101] consisting of the modes "Free", "Leader", "Follower", and "Faulty". Within each mode, a vehicle has a set of restricted maneuvers, including one that allows the vehicle to change modes if desired. The modes and maneuvers are as follows:

- Free:

  A Free vehicle is not in a platoon or on a highway, and its possible maneuvers or mode transitions are

  - remain a Free vehicle by staying away from highways
  - become a Leader by entering a highway to create a new platoon
  - become a Follower by joining a platoon that is currently on a highway

- Leader:

  A Leader vehicle is the vehicle at the front of a platoon (which could consist of only the vehicle itself). The available maneuvers and mode transitions are

  - remain a Leader by traveling along the highway at a pre-specified speed $v_{\mathbb{H}}$
  - become a Follower by merging the current platoon with a platoon in front
  - become a Free vehicle by leaving the highway

- Follower:

  A Follower vehicle is a vehicle that is following a platoon leader. The available maneuvers and mode transitions are

  - remain a Follower by staying a distance of d_{sep} behind the vehicle in front in the current platoon

  - remain a Follower by joining a different platoon on another highway
  - become a Leader by splitting from the current platoon while remaining on the highway
  - become a Free vehicle by leaving the highway

- Faulty:

  If a vehicle from any of the other modes becomes unable to operate within the allowed set of maneuvers, it transitions into the Faulty mode. Reasons for transitioning to the Faulty mode include vehicle malfunctions, performing collision avoidance with respect to another Faulty vehicle, etc. A Faulty vehicle is assumed to descend via a fail-safe mechanism after some pre-specified duration t_{faulty} to a different altitude level where it no longer poses a threat to vehicles on the air highway system.

  Such a fail-safe mechanism could be an emergency landing procedure such as those analyzed in [2, 43, 80]. Typically, emergency landing involves identifying the type of fault and finding feasible landing locations given the dynamics during the fault. We will omit these details and summarize them into t_{faulty}, the time required to exit the current altitude level.

The available maneuvers and associated mode transitions are summarized in Figure 3.4.

Suppose that there are N vehicles in total in the airspace containing the highway system. We will denote the N vehicles as $Q_i, i = 1 \ldots, N$. We consider a platoon of vehicles to be a group of M vehicles ($M \leq N$), denoted $Q_{P_1}, \ldots, Q_{P_M}, \{P_j\}_{j=1}^{M} \subseteq \{i\}_{i=1}^{N}$, in a single-file formation. When necessary, we will use superscripts to denote vehicles of different platoons: $Q_{P_i^j}$ represents the ith vehicle in the jth platoon.

For convenience, let $\mathcal{Q}_i$ denote the set of indices of vehicles with respect to which Q_i checks safety. If vehicle Q_i is a free vehicle, then it must check for safety with respect to all other vehicles, $\mathcal{Q}_i = \{j : j \neq i\}$. If the vehicle is part of a platoon, then it checks safety with respect to the platoon member in front and behind, $\mathcal{Q}_i = \{P_{j+1}, P_{j-1}\}$. Figure 3.5 summarizes the indexing system of the vehicles.

We will organize the vehicles into platoons travel along air highways. The vehicles maintain a separation distance of d_{sep} with their neighbors inside the platoon. In order to allow for close proximity of the vehicles and the ability to resolve multiple simultaneous safety breaches, we assume that when a vehicle exhibits unpredictable behavior, it will be able to exit the altitude range of the highway within a duration of t_{faulty}. Such a requirement may be implemented practically as an fail-safe mechanism to which the vehicles revert when needed.

Objectives: Given the above modeling assumptions, our goal is to provide control strategies to guarantee the success and safety of all the mode transitions. The theo-

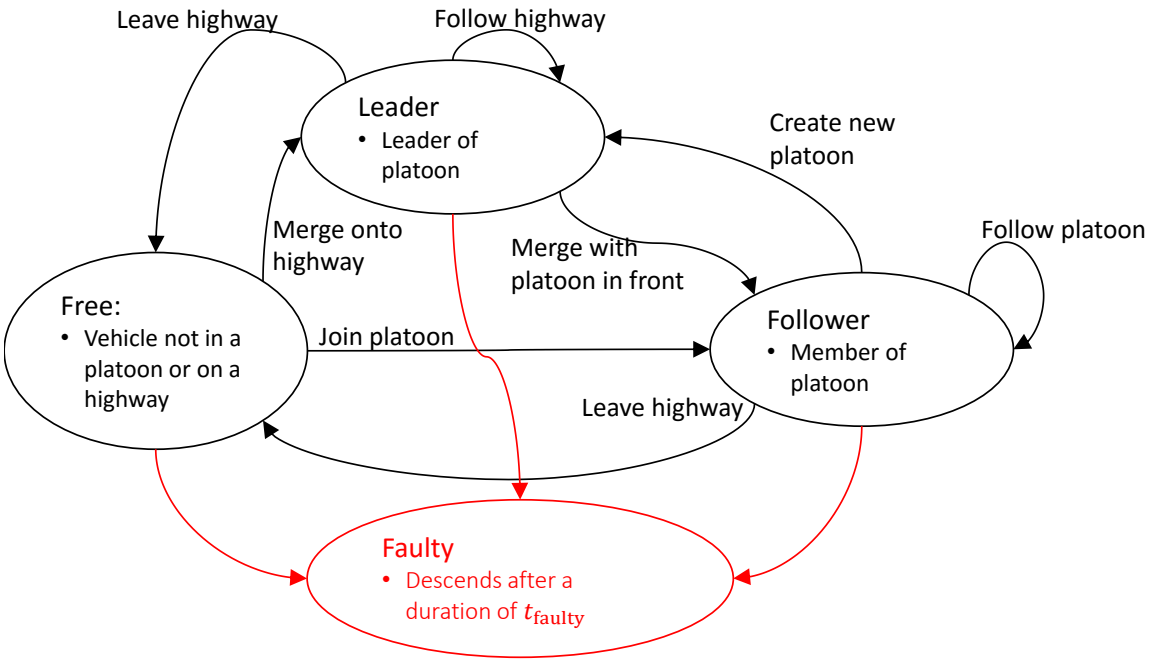


Figure 3.4: Hybrid modes for vehicles in platoons. Vehicles begin in the "Free" mode before they enter the highway.

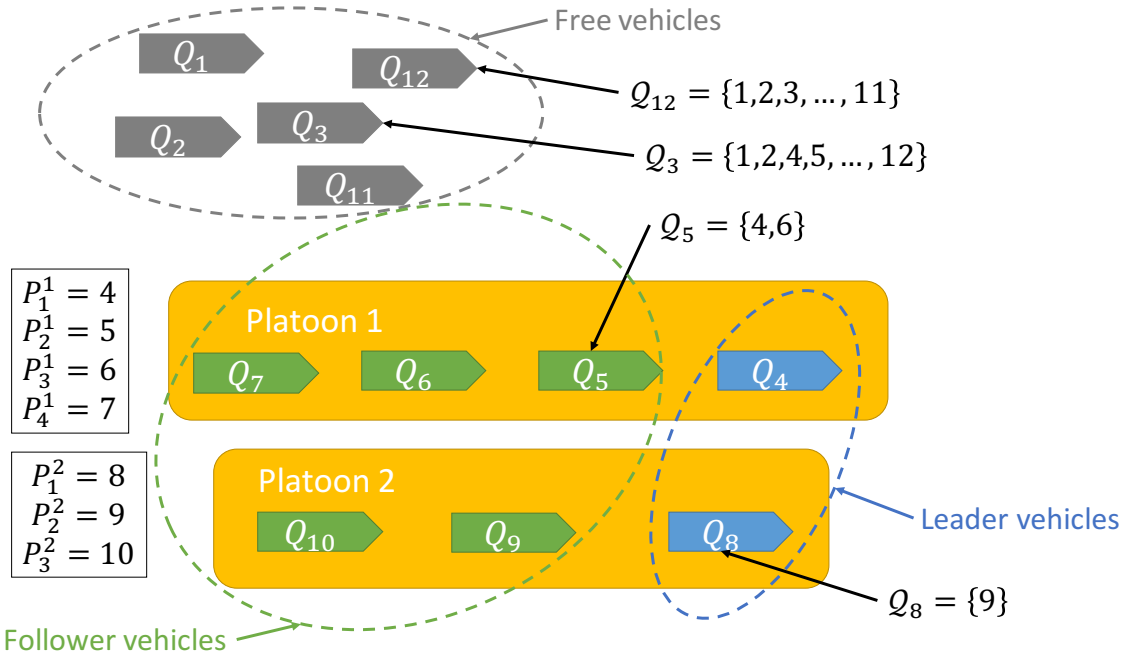


Figure 3.5: Notation for vehicles in platoons.

retical tool used to provide the safety and goal satisfaction guarantees is reachability. The BRSs we compute will allow each vehicle to perform complex actions such as

- merge onto a highway to form a platoon
- join a new platoon
- leave a platoon to create a new one
- react to malfunctioning or intruder vehicles

We also propose more basic controllers to perform other simpler actions such as

- follow the highway at constant altitude at a specified speed
- maintain a constant relative position and velocity with respect to the leader of a platoon

In general, the control strategy of each vehicle has a safety component, which specifies a set of states that it must avoid, and a goal satisfaction component, which specifies a set of states that the vehicle aims to reach. Together, the safety and goal satisfaction controllers guarantee the safety and success of a vehicle in the airspace

making any desired mode transition. In this section, these guarantees are provided using reachability analysis, and allow the multi-UAV system to perform joint maneuvers essential to maintaining structure in the airspace.

Relative Dynamics and Augmented Relative Dynamics: Besides Equation (3.17), we will also consider the relative dynamics between two quadrotors Q_i, Q_j. These dynamics can be obtained by defining the relative variables

$$\begin{aligned} p_{x,r} &= p_{x,i} - p_{x,j} \\ p_{y,r} &= p_{y,i} - p_{y,j} \\ v_{x,r} &= v_{x,i} - v_{x,j} \\ v_{y,r} &= v_{y,i} - v_{y,j} \end{aligned} \tag{3.8}$$

We treat Q_i as Player 1, the evader who wishes to avoid collision, and we treat Q_j as Player 2, the pursuer, or disturbance, that wishes to cause a collision. In terms of the relative variables given in (3.8), we have

$$\begin{aligned} \dot{p}_{x,r} &= v_{x,r} \\ \dot{p}_{y,r} &= v_{y,r} \\ \dot{v}_{x,r} &= u_{x,i} - u_{x,j} \\ \dot{v}_{y,r} &= u_{y,i} - u_{y,j} \end{aligned} \tag{3.9}$$

We also augment (3.8) with the velocity of Q_i, given in (3.10), to impose a velocity limit on the quadrotor.

$$\begin{aligned} \dot{p}_{x,r} &= v_{x,r} \\ \dot{p}_{y,r} &= v_{y,r} \\ \dot{v}_{x,r} &= u_{x,i} - u_{x,j} \\ \dot{v}_{y,r} &= u_{y,i} - u_{y,j} \\ \dot{v}_{x,i} &= u_{x,i} \\ \dot{v}_{y,i} &= u_{y,i} \end{aligned} \tag{3.10}$$

For our application, we will use several decoupled system models and utilize decomposition techniques [32, 33, 37], which enables real-time 4D BRT computations and tractable 6D BRT computations.

3.1.3.2 Reachability-Based Controllers

Reachability analysis is useful for constructing controllers in a large variety of situations. In order to construct different controllers, an appropriate target set needs to be defined depending on the goal of the controller. If one defines the target set to be a set of desired states, the BRS would represent the states that a system needs to first arrive at in order to reach the desired states. On the other hand, if the target set

represents a set of undesirable states, then the BRS would indicate the region of the state space that the system needs to avoid. In addition, the system dynamics with which the BRS is computed provide additional flexibility when using reachability to construct controllers.

Using a number of different target sets and dynamics, we now propose different reachability-based controllers used for vehicle mode transitions in our platooning concept.

Getting to a Target State: The controller used by a vehicle to reach a target state is important in two situations in the platooning context. First, a vehicle in the "Free" mode can use the controller to merge onto a highway, forming a platoon and changing modes to a "Leader" vehicle. Second, a vehicle in either the "Leader" mode or the "Follower" mode can use this controller to change to a different highway, becoming a "Leader" vehicle.

In both of the above cases, we use the dynamics of a single vehicle specified in (3.17). The target state would be a position $(\bar{p}_x, \bar{p}_y)$ representing the desired merging point on the highway, along with a velocity $(\bar{v}_x, \bar{v}_y) = v_{\mathbb{H}}$ that corresponds to the speed and direction of travel specified by the highway. For the reachability computation, we define the target set to be a small range of states around the target state $\bar{x}_H = (\bar{p}_x, \bar{p}_y, \bar{v}_x, \bar{v}_y)$:

$$\begin{aligned} \mathcal{L}_H = \{x : |p_x - \bar{p}_x| \leq r_{p_x}, |v_x - \bar{v}_x| \leq r_{v_x}, \\ |p_y - \bar{p}_y| \leq r_{p_y}, |v_y - \bar{v}_y| \leq r_{v_y}\}. \end{aligned} \tag{3.11}$$

Here, we represent the target set $\mathcal{L}_H$ as the zero sublevel set of the function $l_H(x)$, which specifies the terminal condition of the HJ PDE that we need to solve. Once the HJ PDE is solved, we obtain the BRS $\mathcal{V}_H(t)$ from the subzero level set of the solution $V_H(t, x)$. More concretely, $\mathcal{V}_H(T) = \{x : V_H(-T, x) \leq 0\}$ is the set of states from which the system can be driven to the target $\mathcal{L}_H$ within a duration of T.

Depending on the time horizon T, the size of the BRS $\mathcal{V}_H(T)$ varies. In general, a vehicle may not initially be inside the BRS $\mathcal{V}_H(T)$, yet it needs to be in order to get to its desired target state. Determining a control strategy to reach $\mathcal{V}_H(T)$ is itself a reachability problem (with $\mathcal{V}_H(T)$ as the target set), and it would seem like this reachability problem needs to be solved in order for us to use the results from our first reachability problem. However, practically, one could choose T to be large enough to cover a sufficiently large area to include any practically conceivable initial state. From our simulations, a suitable algorithm for getting to a desired target state is as follows:

1. Move towards $\bar{x}_H$ in pure pursuit with some velocity, until $V_H(-T, x) \leq 0$. In practice, this step consistently drives the system into the BRS.

2. Apply the optimal control extracted from $V_H(-T, x)$ according to (2.11) until $\mathcal{L}_H$ is reached.

Getting to a State Relative to Another Vehicle: In the platooning context, being able to go to a state relative to another moving vehicle is important for the purpose of forming and joining platoons. For example, a "Free" vehicle may join an existing platoon that is on a highway and change modes to become a "Follower". Also, a "Leader" or "Follower" may join another platoon and afterwards go into the "Follower" mode.

To construct a controller for getting to a state relative to another vehicle, we use the relative dynamics of two vehicles, given in (3.9). In general, the target state is specified to be some position $(\bar{p}_{x,r}, \bar{p}_{y,r})$ and velocity $(\bar{v}_{x,r}, \bar{v}_{y,r})$ relative to a reference vehicle. In the case of a vehicle joining a platoon that maintains a single file, the reference vehicle would be the platoon leader, the desired relative position would be a certain distance behind the leader, depending on how many other vehicles are already in the platoon; the desired relative velocity would be $(0, 0)$ so that the formation can be kept.

For the reachability problem, we define the target set to be a small range of states around the target state $\bar{x}_P = (\bar{p}_{x,r}, \bar{p}_{y,r}, \bar{v}_{x,r}, \bar{v}_{y,r})$:

$$\begin{aligned}\mathcal{L}_P = \{x : |p_{x,r} - \bar{p}_{x,r}| \leq r_{p_x}, |v_{x,r} - \bar{v}_{x,r}| \leq r_{v_x}, \\ |p_{y,r} - \bar{p}_{y,r}| \leq r_{p_y}, |v_{y,r} - \bar{v}_{y,r}| \leq r_{v_y}\}\end{aligned} \tag{3.12}$$

The target set $\mathcal{L}_P$ is represented by the zero sublevel set of the implicit surface function $l_P(x)$, which specifies the terminal condition of the HJ PDE (2.16). The zero sublevel set of the solution to (2.16), $V_P(-T, x)$, gives us the set of relative states from which a quadrotor can reach the target in the relative coordinates within a duration of T. In the BRS computation, we assume that the reference vehicle moves along the highway at constant speed, so that $u_j(t) = 0$. The following is a suitable algorithm for a vehicle joining a platoon to follow the platoon leader:

1. Move towards $\bar{x}_P$ in a straight line, with some velocity, until $V_P(-T, x) \leq 0$.
2. Apply the optimal control extracted from $V_P(-T, x)$ according to (2.11) until $\mathcal{L}_P$ is reached.

Avoiding Collisions: A vehicle can use a goal satisfaction controller described in the previous sections when it is not in any danger of collision with other vehicles. If the vehicle could potentially be involved in a collision within the next short period of time, it must switch to a safety controller. The safety controller is available in every mode, and executing the safety controller to perform an avoidance maneuver does not change a vehicle's mode.

In the context of our platooning concept, we define an unsafe configuration as follows: a vehicle is either within a minimum separation distance d to a reference vehicle in both the x and y directions, or is traveling with a speed above the speed

limit $v_{\max}$ in either of the x and y directions. To take this specification into account, we use the augmented relative dynamics given by (3.10) for the reachability problem, and define the target set as follows:

$$\mathcal{L}_S = \{x : |p_{x,r}|, |p_{y,r}| \leq d \vee |v_{x,i}| \geq v_{\max} \vee |v_{y,i}| \geq v_{\max}\} \tag{3.13}$$

We can now define the implicit surface function $l_S(x)$ corresponding to $\mathcal{L}_S$, and solve the HJ PDE (2.16) using $l_S(x)$ as the terminal condition. As before, the zero sublevel set of the solution $V_S(t, x)$ specifies the BRS $\mathcal{V}_S(t)$, which characterizes the states in the augmented relative coordinates, as defined in (3.10), from which Q_i *cannot* avoid $\mathcal{L}_S$ for a time period of t, if Q_j uses the worst-case control. To avoid collisions, Q_i must apply the safety controller according to (2.13) on the boundary of the BRS in order to avoid going into the BRS. The following algorithm wraps our safety controller around goal satisfaction controllers:

1. For a specified time horizon t, evaluate $V_S(-t, x_i - x_j)$ for all $j \in \mathcal{Q}(i)$.

   $\mathcal{Q}(i)$ is the set of quadrotors with which vehicle Q_i checks safety.

2. Use the safety or goal satisfaction controller depending on the values $V_S(-t, x_i - x_j), j \in \mathcal{Q}(i)$:

   If $\exists j \in \mathcal{Q}(i), V_S(-t, x_i - x_j) \leq 0$, then Q_i, Q_j are in potential conflict, and Q_i must use a safety controller; otherwise Q_i may use a goal satisfaction controller.

3.1.3.3 Other Controllers

Reachability was used in Section 3.1.3.2 for the relatively complex maneuvers that require safety and goal satisfaction guarantees. For the simpler maneuvers of traveling along a highway and following a platoon, many well-known classical controllers suffice. For illustration, we use the simple controllers described below.

Traveling along a highway: We use a model-predictive controller (MPC) for traveling along a highway; this controller allows the leader to travel along a highway at a pre-specified speed. Here, the goal is for a leader vehicle to track an air highway $\mathbb{H}(s), s \in [0, 1]$ while maintaining some constant velocity $v_{\mathbb{H}}$ specified by the highway. The highway and the specified velocity can be written as a desired position and velocity over time, $\bar{p}(t), \bar{v}(t)$. Assuming that the initial position on the highway, $s_0 = s(t_0)$ is specified, such a controller can be obtained from the following optimization problem over the time horizon $[t_0, t_1]$:

$$\begin{aligned} \text{minimize } & \int_{t_0}^{t_1} \big\{ \|p(t) - \bar{p}(t)\|_2 + \\ & \qquad \|v(t) - \bar{v}(t)\|_2 + 1 - s \big\} dt \\ \text{subject to vehicle dynamics } & (3.7) \\ & |u_x|, |u_y| \leq u_{\max}, |v_x|, |v_y| \leq v_{\max} \\ & s(t_0) = s_0, \dot{s} \geq 0 \end{aligned} \tag{3.14}$$

If we discretize time, the above optimization is becomes convex optimization over a small number of decision variables, and can be quickly solved.

Following a Platoon: Follower vehicles use a feedback control law tracking a nominal position and velocity in the platoon, with an additional feedforward term given by the leader's acceleration input; here, for simplicity, we assume perfect communication between the leader and the follower vehicles. This following law enables smooth vehicle trajectories in the relative platoon frame, while allowing the platoon as a whole to perform agile maneuvers by transmitting the leader's acceleration command $u_{P_1}(t)$ to all vehicles.

The i-th member of the platoon, Q_{P_i}, is expected to track a relative position in the platoon $r^i = (r^i_x, r^i_y)$ with respect to the leader's position p_{P_1}, and the leader's velocity v_{P_1} at all times. The resulting control law has the form:

$$u^i(t) = k_p \big[p_{P_1}(t) + r^i(t) - p^i(t) \big] + k_v \big[v_{P_1}(t) - v^i(t) \big] + u_{P_1}(t) \tag{3.15}$$

for some $k_p, k_v > 0$. In particular, a simple rule for determining $r^i(t)$ in a single-file platoon is given for Q_{P_i} as:

$$r^i(t) = -(i-1) d_{\text{sep}} \hat{d} \tag{3.16}$$

where d_{sep} is the spacing between vehicles along the platoon and $\hat{d}$ is the highway's direction of travel.

3.1.3.4 Summary of Controllers

We have introduced several reachability-based controllers, as well as some simple controllers. Pairwise collision avoidance is guaranteed using the safety controller, described in Section 3.1.3.2. As long as a vehicle is not in potential danger according to the safety BRSs, it is free to use any other controller. All of these other controllers are *goal satisfaction controllers*, and their corresponding mode transitions are shown in Figure 3.6.

The controller for getting to an absolute target state, described in Section 3.1.3.2, is used whenever a vehicle needs to move onto a highway to become a platoon leader. This controller guarantees the success of the mode transitions shown in blue in Figure 3.6.

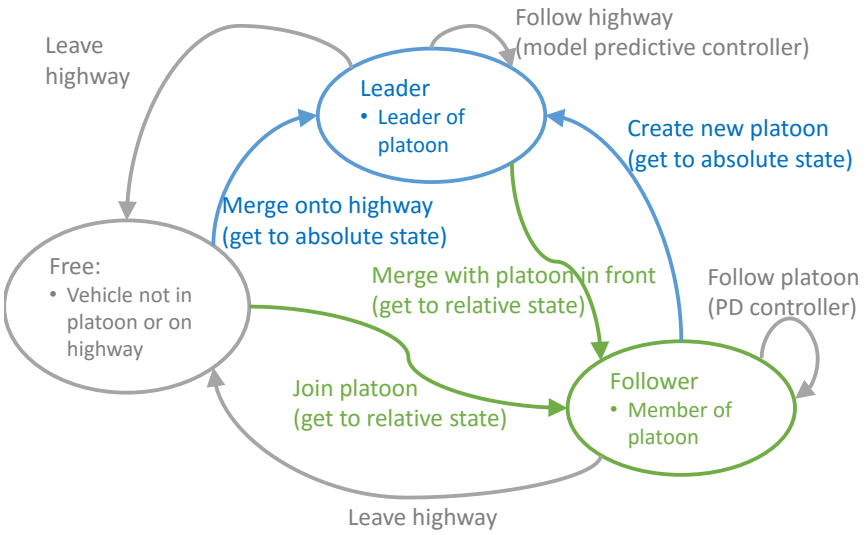


Figure 3.6: Summary of mode switching controllers. Reachability-based controllers are shown as the blue and green arrows.

The controller for getting to a relative target state, described in Section 3.1.3.2, is used whenever a vehicle needs to join a platoon to become a follower. This controller guarantees the success of the mode transitions shown in green in Figure 3.6.

For the simple maneuvers of traveling along a highway or following a platoon, many simple controllers such as the ones suggested in Section 3.1.3.3 can be used. These controllers keep the vehicles in either the Leader or the Follower mode. Alternatively, additional controllers can be designed for exiting the highway, although these are not considered in this section. All of these non-reachability-based controllers are shown in gray in Figure 3.6.

3.1.3.5 Safety Analysis

Under normal operations in a single platoon, each follower vehicle $Q_i, i = P_2, \ldots, P_{M-1}$ in a platoon checks whether it is in the safety BRS with respect to $Q_{P_{i-1}}$ and $Q_{P_{i+1}}$. So $\mathcal{Q}_i = \{P_{i+1}, P_{i-1}\}$ for $i = P_2, \ldots, P_{N-1}$. Assuming there are no nearby vehicles outside of the platoon, the platoon leader Q_{P_1} checks safety against Q_{P_2}, and the platoon trailer Q_{P_N} checks safety against $Q_{P_{N-1}}$. So $\mathcal{Q}_{P_1} = \{P_2\}, \mathcal{Q}_{P_N} = \{P_{N-1}\}$. When all vehicles are using goal satisfaction controllers to perform their allowed maneuvers, in most situations no pair of vehicles should be in an unsafe configuration. However, occasionally a vehicle Q_k may behave unexpectedly due to faults or malfunctions, in which case it may come into an unsafe configuration with another vehicle.

With our choice of $\mathcal{Q}_i$ and the assumption that the platoon is in a single-file formation, some vehicle Q_i would get near the safety BRS with respect to Q_k, where

Q_k is likely to be the vehicle in front or behind of Q_i. In this case, a "safety breach" occurs. Our synthesis of the safety controller guarantees the following: between every pair of vehicles Q_i, Q_k, if $V_S(-t, x_i - x_k) > 0$, then $\exists u_i$ to keep Q_i from colliding with Q_k for a desired time horizon t, despite the worst case (an adversarial) control from Q_k. Therefore, as long as the number of "safety breaches" is at most one for Q_i, Q_i can simply use the optimal control to avoid Q_k and avoid collision for the time horizon of t. Under the assumption that vehicles are able to exit the current altitude range within a duration of t_{faulty}, if we choose $t = t_{\text{faulty}}$, the safety breach would always end before any collision can occur.

Within a duration of t_{faulty}, there is a small chance that additional safety breaches may occur. However, as long as the total number of safety breaches does not exceed the number of affected vehicles, collision avoidance of all the vehicles can be guaranteed for the duration t_{faulty}. However, as our simulation results show, placing vehicles in single-file platoons makes the likelihood of multiple safety breaches low during the presence of one intruder vehicle.

In the event that multiple safety breaches occur for some of the vehicles due to a malfunctioning vehicle within the platoon or intruding vehicles outside of the platoon, vehicles that are causing safety breaches must exit the highway altitude range in order to avoid collisions. Every extra altitude range reduces the number of simultaneous safety breaches by 1, so K simultaneous safety breaches can be resolved using $K - 1$ different altitude ranges. The general process and details of the complete picture of multi-altitude collision avoidance is part of our future work.

The concept of platooning can be coupled with any collision avoidance algorithm that provides safety guarantees. In this section, we have only proposed the simplest reachability-based collision avoidance scheme. Existing collision avoidance algorithms such as [15] and [36] have the potential to provide safety guarantees for many vehicles in order to resolve multiple safety breaches at once. Coupling the platooning concept with the more advanced collision avoidance methods that provide guarantees for a larger number of vehicles would reduce the risk of multiple safety breaches.

Given that vehicles within a platoon are safe with respect to each other, each platoon can be treated as a single vehicle, and perform collision avoidance with other platoons when needed. The option of treating each platoon as a single unit can reduce the number of individual vehicles that need to check for safety against each other, reducing overall computation burden.

3.1.3.6 Numerical Simulations

In this section, we consider several situations that vehicles in a platoon on an air highway may commonly encounter, and show via simulations the behaviors that emerge from the controllers we defined in Sections 3.1.3.2 and 3.1.3.3.

Forming a Platoon: We first consider the scenario in which Free vehicles merge onto

an initially unoccupied highway. In order to do this, each vehicle first checks safety with respect to all other vehicles, and uses the safety controller if necessary, according to Section 3.1.3.2. Otherwise, the vehicle uses the goal satisfaction controller for getting to an absolute target set described in Section 3.1.3.2 in order to merge onto the highway, create a platoon, and become a Leader vehicle if there are no platoons on the highway. If there is already a platoon on the highway, then the vehicle would use the goal satisfaction controller for getting to a target set relative to the platoon leader as described in Section 3.1.3.2 to join the platoon and become a Follower.

For the simulation example, shown in Figure 3.7, the highway is specified by a line segment beginning at the origin. The five vehicles, $Q_1, Q_2, \ldots, Q_5$ are colored orange, purple, light blue, dark blue, and yellow, respectively.

The first two plots in Figure 3.7 illustrate the use of safety and goal satisfaction BRS for the first two vehicles. Since the goal satisfaction BRSs are in 4D and the safety BRSs are in 6D, we compute and plot their 2D slices based on the vehicles' velocities and relative velocities. All vehicles begin as Free vehicles, so they each need to take into account five different BRSs: four safety BRSs and one goal satisfaction BRS. For clarity, we only show the goal satisfaction BRS and the four safety BRSs for one of the vehicles.

For Q_1 (orange), an arbitrary point of entry on the highway is chosen as the target absolute position, and the velocity corresponding to a speed of 10 m/s in the direction of the highway is chosen as the target absolute velocity. This forms the target state $\bar{x}_H = (\bar{p}_x, \bar{v}_x, \bar{p}_y, \bar{v}_y)$, from which we define the target set $\mathcal{L}_H$ as in Section 3.1.3.2.

At $t = 4.2$, Q_1 (orange) is inside the goal satisfaction BRS for getting to an absolute state, shown as the dotted orange boundary. Therefore, it is "locked-in" to the target state $\bar{x}_H$, and follows the optimal control in (2.11) to $\bar{x}_H$. During the entire time, Q_1 checks whether it may collide with any of the other vehicles within a time horizon of t_{faulty}. To do this, it simply checks whether its state relative to each of the other vehicles is within the corresponding safety BRS. As an example, the safety BRS boundary with respect to Q_2 (purple) is shown as the orange dashed boundary around Q_2 (purple); Q_1 (orange) is safe with respect to Q_2 (purple) since Q_1 (orange) is outside of the boundary. In fact, Q_1 is safe with respect to all vehicles.

After completing merging onto the empty highway, Q_1 (orange) creates a platoon and becomes its leader, while subsequent vehicles begin to form a platoon behind the leader in the order of ascending distance to Q_1 (orange) according to the process described in Section 3.1.3.2. Here, we choose the target relative position $(\bar{p}_{x,r}, \bar{p}_{y,r})$ to be a distance d_{sep} behind the last reserved slot in the platoon, and the target relative velocity $(\bar{v}_{x,r}, \bar{v}_{y,r}) = (0, 0)$ with respect to the leader in order to maintain the platoon formation. This gives us the target set $\mathcal{L}_P$ that we need.

At $t = 8.0$, Q_2 (purple) is in the process of joining the platoon behind Q_1 (orange) by moving towards the target $\bar{x}_P$ relative to the position of Q_1 (orange). Note that $\bar{x}_P$ moves with Q_1 (orange) as $\bar{x}_P$ is defined in terms of the relative states

of the two vehicles. Since Q_2 is inside the goal satisfaction BRS boundary for joining the platoon (purple dotted boundary), it is "locked-in" to the target relative state $\bar{x}_P$, and begins following the optimal control in (2.11) towards the target as long as it stays out of all safety BRSs. For example, at $t = 5.9$, Q_2 (purple) is outside of the safety BRS with respect to Q_1 (orange), shown as the purple dashed boundary around Q_1 (orange). Again, from the other safety BRS boundaries, we can see that Q_2 is in fact safe with respect to all vehicles.

In the bottom plots of Figure 3.7, Q_1 (orange) and Q_2 (purple) have already become the platoon leader and follower, respectively. The rest of the vehicles follow the same process to join the platoon. All 5 vehicles eventually form a single platoon and travel along the highway together. As with the first two vehicles, the goal satisfaction controllers allow the remaining vehicles to optimally and smoothly join the platoon, while the safety controllers prevent collisions from occurring.

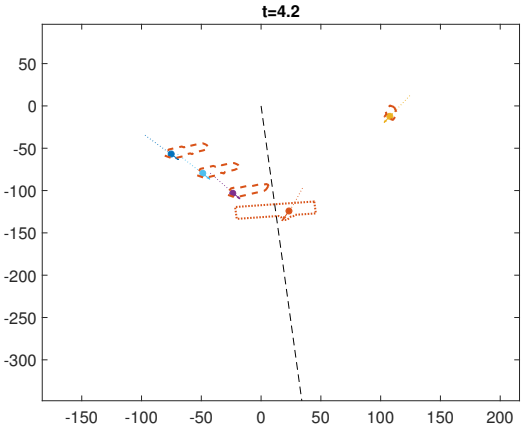
Intruder Vehicle: We now consider the scenario in which a platoon of vehicles encounters an intruder vehicle. To avoid collision, each vehicle checks for safety with respect to the intruder and any vehicles in front and behind of it in the platoon. If necessary, the vehicle uses the reachability-based safety controller to avoid collision, otherwise it uses the appropriate controller to travel on the highway if it is a leader, or follow the leader if it is a follower. After danger has passed, the vehicles in the platoon resume normal operation.

Figure 3.8 shows the simulation result. At $t = 9.9$, a platoon of four vehicles, $Q_i, i = 1, \ldots, 4$ (with $P_i = i$), travels along the highway shown. An intruder vehicle Q_5 (yellow) heads left, disregarding the presence of the platoon. At $t = 11.9$, the platoon leader Q_1 (red) detects that it has gone near the boundary of the safety BRS (not shown) with respect to the intruder Q_5 (yellow). In response, Q_1 (red) starts using the safety controller to optimally avoid the intruder according to (2.13); in doing so, it steers slightly off the highway.

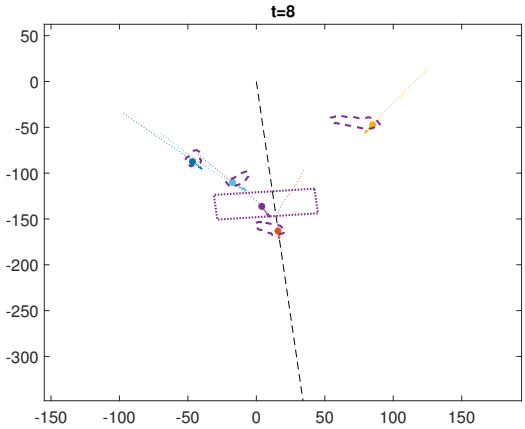
Note that although in this particular simulation, the intruder travels in a straight line, a straight line motion of the intruder was *not* assumed. Rather, the safety BRSs are computed assuming the worst case control of the intruder.

As the intruder Q_5 (yellow) continues to disregard other vehicles, the followers of the platoon also get near the respective boundaries of their safety BRSs with respect to the intruder. This occurs at $t = 13.9$, where the platoon "makes room" for the intruder to pass by to avoid collisions; all vehicles deviate from their intended path, which is to follow the platoon leader or the highway. Note that in this case, we have assumed that the platoon does not move as a unit in response to an intruder to show more interesting behavior.

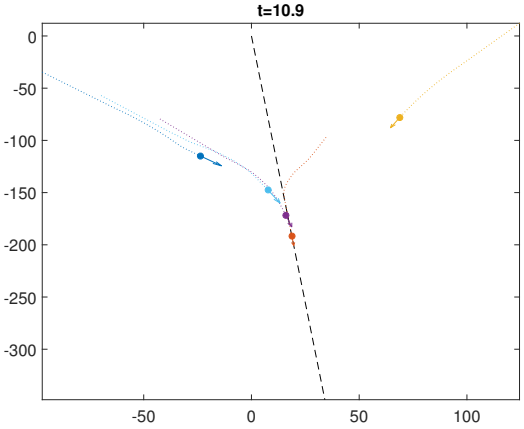
After the intruder has passed, eventually all vehicles become far away from any safety BRSs. When this occurs, the leader resumes following the highway, and the followers resume following the leader. At $t = 19.9$, the platoon successfully gets back onto the highway.



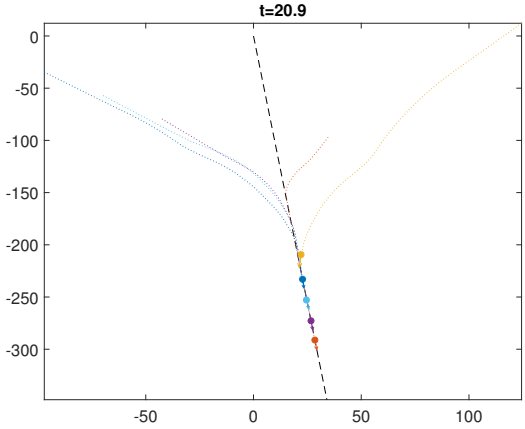
(a) The red vehicle is merging onto the highway while avoiding collisions.



(b) The purple vehicle is joining the platoon while avoiding collisions.

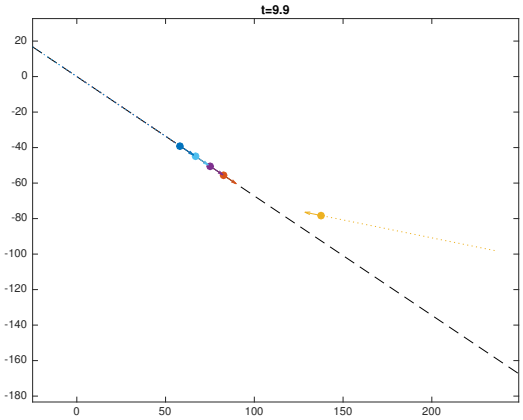


(c) The last three vehicles follow the same process to join the platoon.

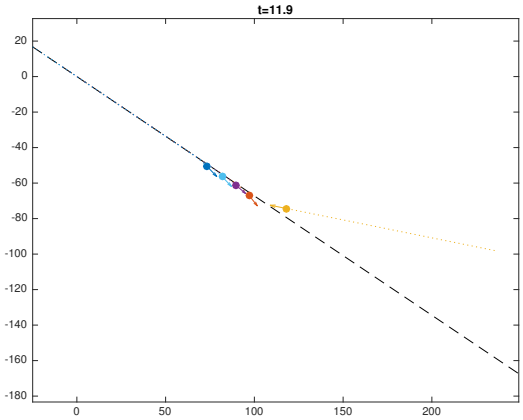


(d) All five vehicles have successfully joined the platoon and now travel on the highway together.

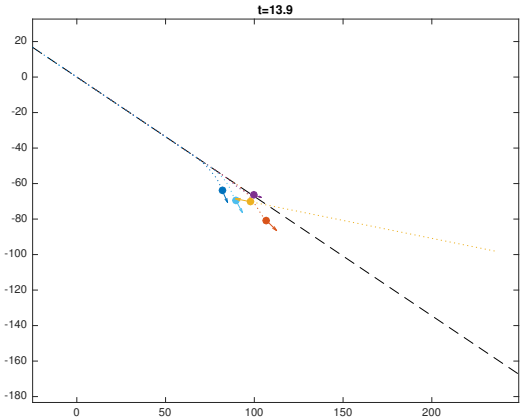
Figure 3.7: A simulation showing how five vehicles initially in the Free mode can form a platoon.



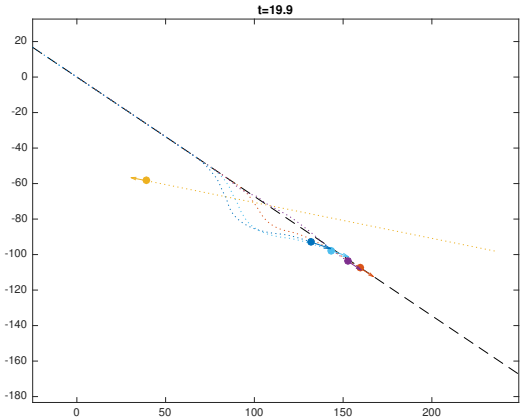
(a) A four-vehicle platoon travels along the highway. The yellow vehicle disregards the others.



(b) Vehicles begin avoidance maneuvers as they get near safety BRS boundaries.



(c) Safety controllers cause vehicles to spread out to "make room" for the intruder to pass.



(d) After danger has passed, the platoon resumes normal operation.

Figure 3.8: A simulation showing how a platoon of four vehicles reacts to an intruder.

Changing highways: In order to travel from origin to destination, a vehicle may need to change highways several times before exiting an air highway system. In this simulation, shown in Figure 3.9, two platoons are traveling on two different highways that intersect. When the platoons are near the highway intersection, two of the vehicles in the four-vehicle platoon change highways and join the other platoon.

The $t = 8.2$ plot shows the two platoons of vehicles traveling on the two air highways. One platoon has three vehicles, and the other has four vehicles. At $t = 12.3$, the yellow vehicle begins steering off its original highway in order to join the other platoon. In terms of the hybrid systems modes, the yellow vehicle transitions from the Leader mode to the Follower mode. At the same time, the green vehicle transitions from the Follower mode to the Leader mode, since the previous platoon leader, the yellow vehicle, has left the platoon. By $t = 16.9$, the yellow vehicle successfully changes highways and is now a follower in its new platoon.

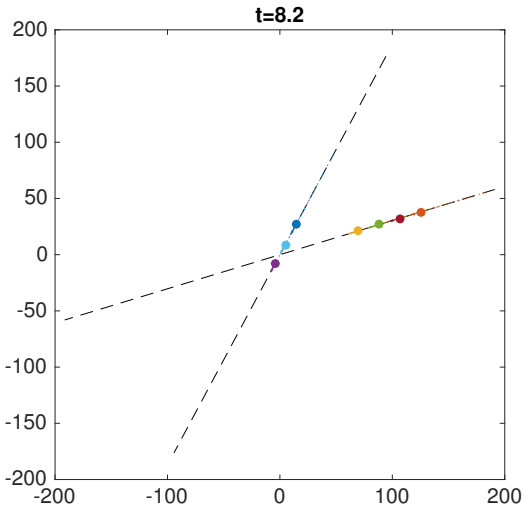
At $t = 16.9$, the dark red vehicle is in the process of changing highways. In this case, it remains in the Follower mode, since it is a follower in both its old and new platoons. While the dark red vehicle changes highways, the orange vehicle moves forward to catch up to its new platoon leader, the green vehicle. By $t = 23$, all the vehicles have finished performing their desired maneuvers, resulting in a two-vehicle platoon and a five-vehicle platoon traveling on their respective highways.

3.1.4 Conclusions

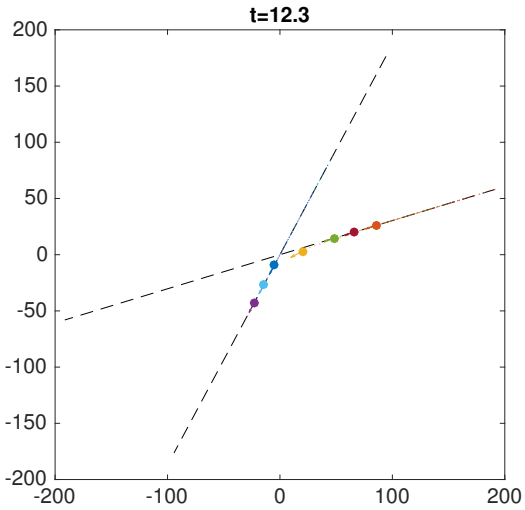
To address the important and urgent problem of the traffic management of unmanned aerial vehicles (UAVs), we proposed to have platoons of UAVs traveling on air highways. We showed how such an airspace structure leads to much easier safety and goal satisfaction analysis. We provided simulations which show that by organizing vehicles into platoons, many complex maneuvers can be performed using just a few different backward reachable sets.

For the placement of air highways over a region, we utilize the very intuitive and efficient fast marching algorithm for solving the Eikonal equation. Our algorithm allows us to take as input any arbitrary cost map representing the desirability of flying over any position in space, and produce a set of paths from any destination to a particular origin. Simple heuristic clustering methods can then be used to convert the sets of paths into a set of air highways.

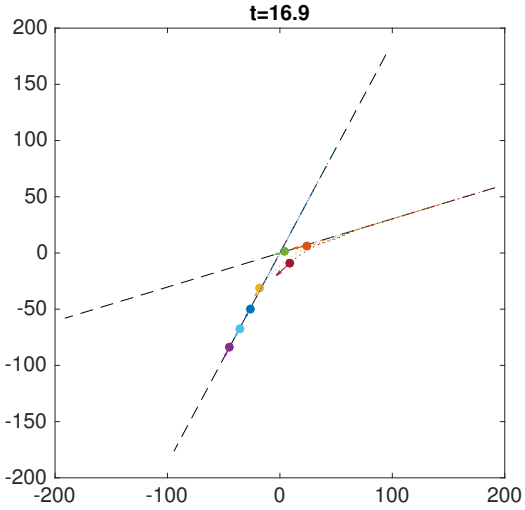
On the air highways, we considered platoons of UAVs modeled by hybrid systems. We show how various required platoon functions (merging onto an air highway, changing platoons, etc.) can be implemented using only the Free, Leader, and Follower modes of operation. Using HJ reachability, we proposed goal satisfaction controllers that guarantee the success of all mode transitions, and wrapped a safety controller around goal satisfaction controllers to ensure no collision between the UAVs can occur. Under the assumption that faulty vehicles can descend after a pre-specified duration, our safety controller guarantees that no collisions will occur in a single alti-



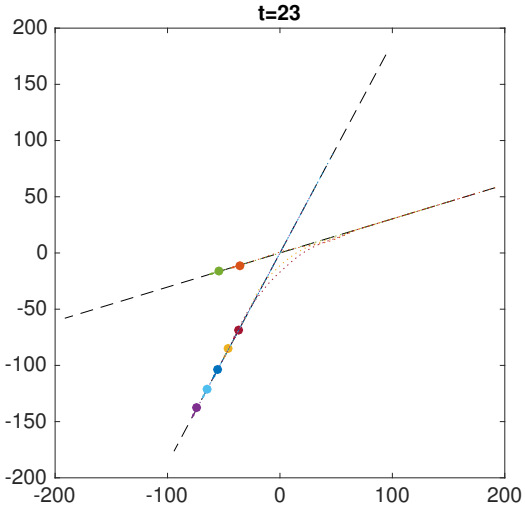
(a) A three-vehicle platoon and a four-vehicle platoon travel on their respective air highways.



(b) The yellow vehicle begins to join the new platoon; The green vehicle becomes a leader.



(c) The dark red vehicle joins a new platoon; the orange vehicle catches up to new platoon leader.



(d) New platoons now travel on their respective air highways.

Figure 3.9: A simulation showing two vehicles changing highways and joining a new platoon.

tude level as long as at most one safety breach occurs for each vehicle in the platoon. Additional safety breaches can be handled by multiple altitude ranges in the airspace.

3.2 Provably Safe and Scalable UAV Routing: A Case Study in San Francisco and the Bay Area

This section is an adaptation of the paper in [29].

One essential problem that needs to be addressed for the unmanned airspace to be successful is that of trajectory planning: how a group of vehicles in the same vicinity can reach their destinations while avoiding situations which are considered dangerous, such as collisions. Many previous studies address this problem under different assumptions. In some studies, specific control strategies for the vehicles are assumed, and approaches such as those involving induced velocity obstacles [27, 63, 141, 145] and involving virtual potential fields to maintain collision avoidance [40, 121] have been used. Methods have also been proposed for real-time trajectory generation [62], for path planning for vehicles with linear dynamics in the presence of obstacles with known motion [3], and for cooperative path planning via waypoints which do not account for vehicle dynamics [22]. Other related work include those which consider only the collision avoidance problem without path or trajectory planning. These results include those that assume the system has a linear model [20, 132, 138], rely on a linearization of the system model [7, 106], assume a simple positional state space [99], and many others [**?**, 77, 97].

However, to make sure that a dense group of UAVs can safely fly in the close vicinity of each other, we need the capability to flexibly plan provably safe and dynamically feasible trajectories without making strong assumptions on the vehicles' dynamics and other vehicles' motion. Moreover, any trajectory planning scheme that addresses collision avoidance must also guarantee both goal satisfaction and safety of UAVs despite disturbances caused by wind and communication faults [123]. Finally, the proposed scheme should scale well with the number of vehicles, as well as result in an intuitive airspace structure for humans to monitor and potentially adjust.

The problem of trajectory planning and collision avoidance under disturbances in safety-critical systems has been well-studied using HJ reachability analysis, which provides guarantees on goal satisfaction and safety of optimal system trajectories [16, 23, 24, 64, 105, 111]. Reachability-based methods are particularly suitable in the context of UAVs because of the hard guarantees that are provided. Despite its power, the approach becomes numerically intractable as the state space dimension increases, and therefore is not directly suitable for managing the next generation airspace, which is a large-scale system with a high-dimensional joint state space because of the high density of vehicles that needs to be accommodated [123].

To overcome this problem, the Sequential Trajectory Planning (STP) method was proposed in [30]. Here, vehicles are assigned a strict priority ordering. Higher-priority vehicles plan their trajectories without taking into account the lower-priority vehicles. Lower-priority vehicles treat higher-priority vehicles as moving obstacles. Under this assumption, time-varying formulations of reachability [24, 64] can be used to obtain the optimal and provably safe trajectories for each vehicle, starting from the highest-priority vehicle. Thus, the curse of dimensionality is overcome for the multi-vehicle trajectory planning problem at the cost of a mild structural assumption, under which the computation complexity scales just *linearly* with the number of vehicles.

Intuitively, STP algorithm allocates a space-time trajectory to each vehicle based on their priorities. The highest-priority vehicle gets to choose any space-time trajectory that does not collide with static obstacles, such as the optimal trajectory. The next vehicle's trajectory must not intersect with the trajectory of the highest-priority vehicle, and so on. Hence two vehicles can either follow same state tarjectory but at different times (referred to as *time-separated* trajectories here on) or follow different state trajectories but at the same time (referred to as *state-separated* trajectories here on), but not both. Finally, they can have different state trajectories at different times (referred to as *state-time separated* trajectories here on). So by design, STP algorithm ensures that the space-time trajectories of the vehicles do not intersect, and hence a safe transition to destination is guaranteed for all vehicles.

Authors in [14] and [28], respectively, extend STP to the scenarios where disturbances and adversarial intruders are present in the system, resolving some of the practical challenges associated with the basic STP algorithm in [30]. The focus of these works, however, have mostly been on the theoretical development of STP algorithm. The focus of this chapter is instead on demonstrating the potential of STP algorithm as a provably safe trajectory planning algorithm for large-scale systems. In particular,

- We simulate large-scale multi-vehicle systems in two different urban environments under the presence of disturbances in vehicles' dynamics. First, the STP algorithm is used for trajectory planning at a city level and then at a regional level. For city level planning, we consider the city of San Francisco in California, USA, and for regional level planning we consider a part of San Francisco Bay Area in California, USA. The main differences between these two case studies are that the city level planning needs to take into account static physical obstacles such as tall buildings, whereas the origins and destinations are farther apart at the regional level. In both cases, we demonstrate that STP algorithm is able to design provably-safe trajectories despite the disturbances.

- We demonstrate how different types of space-time trajectories emerge naturally out of STP algorithm between a given pair of origin and destination for different disturbance conditions and other problem parameters. These emerging behav-

iors, while being provably safe, are also intuitive and would facilitate human monitoring.

- We also show the reactivity of the control law obtained from STP algorithm. In other words, we show that the obtained control law is able to effectively counteract the disturbances in real-time without requiring any communication with other vehicles.

The rest of this section is organized as follows: in Section 3.2.1, we formally present the STP problem in the presence of disturbances. In Section 3.2.2, we present a brief review of time-varying reachability, the basic STP algorithm [30] in the absence of disturbances, and the Robust Trajectory Tracking (RTT) method [14] to account for disturbances. We also use this algorithm for all our simulations in this section. Simulation results are in Sections 3.2.3 and 3.2.4.

3.2.1 Hamilton-Jacobi Sequential Trajectory Planning

Consider N vehicles (also denoted as *STP vehicles*) which participate in the STP process $Q_i, i = 1, \ldots, N$. We assume their dynamics are given by

$$\begin{aligned} &\dot{z}_i = f_i(z_i, u_i, d_i), t \leq t_i^{\text{STA}} \\ &u_i \in \mathcal{U}_i, d_i \in \mathcal{D}_i, i = 1 \ldots, N \end{aligned} \tag{3.17}$$

where $z_i \in \mathbb{R}^{n_i}$, $u_i \in \mathcal{U}_i$ and $d_i \in \mathcal{D}_i$, respectively, represent the state, control and disturbance experienced by vehicle Q_i. We partition the state z_i into the position component $p_i \in \mathbb{R}^{n_p}$ and the non-position component $h_i \in \mathbb{R}^{n_i - n_p}$: $z_i = (p_i, h_i)$. We will use the sets $\mathbb{U}_i, \mathbb{D}_i$ to respectively denote the set of functions from which the control and disturbance functions $u_i(\cdot), d_i(\cdot)$ are drawn.

Each vehicle Q_i has initial state z_i^0, and aims to reach its target $\mathcal{L}_i$ by some scheduled time of arrival t_i^{STA}. The target in general represents some set of desirable states, for example the destination of Q_i. On its way to $\mathcal{L}_i$, Q_i must avoid a set of static obstacles $\mathcal{O}_i^{\text{static}} \subset \mathbb{R}^{n_i}$. The interpretation of $\mathcal{O}_i^{\text{static}}$ could be a tall building or any set of states that are forbidden for each STP vehicle. In addition to the static obstacles, each vehicle Q_i must also avoid the danger zones with respect to every other vehicle $Q_j, j \neq i$. The danger zones in general can represent any joint configurations between Q_i and Q_j that are considered to be unsafe. We define the danger zone of Q_i with respect to Q_j to be

$$\mathcal{Z}_{ij} = \{(z_i, z_j) : \|p_i - p_j\|_2 \leq R_c\} \tag{3.18}$$

whose interpretation is that Q_i and Q_j are considered to be in an unsafe configuration when they are within a distance of R_c of each other. In particular, Q_i and Q_j are said to have collided, ifif $(z_i, z_j) \in \mathcal{Z}_{ij}$.

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

In general, the above optimal trajectory planning problem must be solved in the joint space of all N STP vehicles. However, due to the high joint dimensionality, a direct dynamic programming-based solution is intractable. Therefore, authors in [30] proposed to assign a priority to each vehicle, and perform STP given the assigned priorities. Without loss of generality, let Q_j have a higher priority than Q_i if $j < i$. Under the STP scheme, higher-priority vehicles can ignore the presence of lower-priority vehicles, and perform trajectory planning without taking into account the lower-priority vehicles' danger zones. A lower-priority vehicle Q_i, on the other hand, must ensure that it does not enter the danger zones of the higher-priority vehicles $Q_j, j < i$; each higher-priority vehicle Q_j induces a set of time-varying obstacles $\mathcal{O}_i^j(t)$, which represents the possible states of Q_i such that a collision between Q_i and Q_j could occur.

It is straight-forward to see that if each vehicle Q_i is able to plan a trajectory that takes it to $\mathcal{L}_i$ while avoiding the static obstacles $\mathcal{O}_i^{\text{static}}$ and the danger zones of *higher-priority vehicles* $Q_j, j < i$, then the set of STP vehicles $Q_i, i = 1, \ldots, N$ would all be able to reach their targets safely. Under the STP scheme, trajectory planning can be done sequentially in descending order of vehicle priority in the state space of only a single vehicle. Thus, STP provides a solution whose complexity scales linearly with the number of vehicles, as opposed to exponentially with a direct application of dynamic programming approaches.

3.2.2 Background

In this section, we present the basic STP algorithm [30] in which disturbances are ignored and perfect information of vehicles positions is assumed. This simplification allows us to clearly present the basic STP algorithm. However, in presence of disturbances, it is no longer possible to commit to exact trajectories (and hence positions), since the disturbance $d_i(\cdot)$ is *a priori* unknown. Thus, disturbances and incomplete information significantly complicate the STP scheme. We next present the robust trajectory tracking algorithm [14] that can be used to make basic STP approach robust to disturbances as well as to an imperfect knowledge of other vehicles' positions. All of these algorithms use time-varying reachability analysis to provide goal satisfaction and safety guarantees; therefore, we start with an overview of time-varying reachability.

3.2.2.1 Time-Varying HJ Reachability

We will be using reachability analysis to compute a backward reachable set (BRS) $\mathcal{V}$ given some target set $\mathcal{L}$, time-varying obstacle $\mathcal{G}(t)$, and the Hamiltonian function H which captures the system dynamics as well as the roles of the control and disturbance. The BRS $\mathcal{V}$ in a time interval $[t, t_f]$ will be denoted by

$$\mathcal{V}(t, t_f) \quad \text{(backward reachable set)} \tag{3.19}$$

Several formulations of reachability are able to account for time-varying obstacles [24, 64] (or state constraints in general). For our application in STP, we utilize the time-varying formulation in [64], which accounts for the time-varying nature of systems without requiring augmentation of the state space with the time variable. In the formulation in [64], a BRS is computed by solving the following *final value* double-obstacle HJ VI:

$$\begin{aligned} \max\Big\{ \min\{D_t V(t,z) + H(t,z,\nabla V(t,z)), l(z) - V(t,z)\}, \\ - g(t,z) - V(t,z)\Big\} = 0, \quad t \le t_f \\ V(t_f, z) = \max\{l(z), -g(t_f, z)\} \end{aligned} \tag{3.20}$$

In (3.20), the function $l(z)$ is the implicit surface function representing the target set $\mathcal{L} = \{z : l(z) \le 0\}$. Similarly, the function $g(t, z)$ is the implicit surface function representing the time-varying obstacles $\mathcal{G}(t) = \{z : g(t, z) \le 0\}$. The BRS $\mathcal{V}(t, t_f)$ is given by

$$\mathcal{V}(t, t_f) = \{z : V(t, z) \le 0\} \tag{3.21}$$

Some of the reachability computations will not involve an obstacle set $\mathcal{G}(t)$, in which case we can simply set $g(t, z) \equiv \infty$ which effectively means that the outside maximum is ignored in (3.20).

The Hamiltonian, $H(t, z, \nabla V(t, z))$, depends on the system dynamics, and the role of control and disturbance. Whenever H does not depend explicit on t, we will drop the argument. In addition, the Hamiltonian is an optimization that produces the optimal control $u^*(t, z)$ and optimal disturbance $d^*(t, z)$, once V is determined. For BRSs, whenever the existence of a control ("$\exists u$") or disturbance is sought, the optimization is a minimum over the set of controls or disturbance. Whenever a BRS characterizes the behavior of the system for all controls ("$\forall u$") or disturbances, the optimization is a maximum. We will introduce precise definitions of reachable sets, expressions for the Hamiltonian, expressions for the optimal controls as needed for the many different reachability calculations we use.

3.2.2.2 STP Without Disturbances

In this section, we give an overview of the basic STP algorithm assuming that there is no disturbance and intruder affecting the vehicles. Although in practice, such assumptions do not hold, the description of the basic STP algorithm will introduce the notation needed for describing the subsequent, more realistic versions of STP. The majority of the content in this section is taken from [30].

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

$$\begin{aligned} &\dot{z}_i = f_i(z_i, u_i), t \leq t_i^{\text{STA}} \\ &u_i \in \mathcal{U}_i, \qquad i = 1 \ldots, N \end{aligned} \tag{3.22}$$

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

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

$$\mathcal{G}_i(t) = \mathcal{O}_i^{\text{static}} \cup \bigcup_{j=1}^{i-1} \mathcal{O}_i^j(t) \tag{3.23}$$

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

$$\mathcal{O}_i^j(t) = \{z_i : \|p_i - p_j(t)\|_2 \leq R_c\} \tag{3.24}$$

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

$$\begin{aligned} \mathcal{V}_i^{\text{basic}}(t, t_i^{\text{STA}}) = &\{y : \exists u_i(\cdot) \in \mathbb{U}_i, z_i(\cdot) \text{ satisfies (3.22)}, \\ &\forall s \in [t, t_i^{\text{STA}}], z_i(s) \notin \mathcal{G}_i(s), \\ &\exists s \in [t, t_i^{\text{STA}}], z_i(s) \in \mathcal{L}_i, z_i(t) = y\} \end{aligned} \tag{3.25}$$

The BRS $\mathcal{V}(t, t_i^{\text{STA}})$ can be obtained by solving (3.20) with $\mathcal{L} = \mathcal{L}_i$, $\mathcal{G}(t) = \mathcal{G}_i(t)$, and the Hamiltonian

$$H_i^{\text{basic}}(z_i, \lambda) = \min_{u_i \in \mathcal{U}_i} \lambda \cdot f_i(z_i, u_i) \tag{3.26}$$

The optimal control for reaching $\mathcal{L}_i$ while avoiding $\mathcal{G}_i(t)$ is then given by

$$u_i^{\text{basic}}(t, z_i) = \arg\min_{u_i \in \mathcal{U}_i} \lambda \cdot f_i(z_i, u_i) \tag{3.27}$$

from which the trajectory $z_i(\cdot)$ can be computed by integrating the system dynamics, which in this case are given by (3.22). In addition, the latest departure time t_i^{LDT} can be obtained from the BRS $\mathcal{V}(t, t_i^{\text{STA}})$ as $t_i^{\text{LDT}} = \arg\sup_t \{z_i^0 \in \mathcal{V}(t, t_i^{\text{STA}})\}$. In summary, the basic STP algorithm is given in Algorithm 1.

Algorithm 1: STP algorithm in the absence of disturbances

input : Set of vehicles $Q_i, i = 1, \ldots, N$ in the descending priority order;
Vehicle dynamics (3.22) and initial states z_i^0;
Vehicle destinations $\mathcal{L}_i$ and static obstacles $\mathcal{O}_i^{\text{static}}$.

output: Provably safe vehicle trajectories to respective destinations for all vehicles;
Goal-satisfaction controller $u^{\text{basic}}(\cdot)$ for all vehicles.

for $i = 1 : N$ **do**
- **Trajectory planning for** Q_i
- compute the total obstacle set $\mathcal{G}_i(t)$ given by (3.23). If $i = 1$, $\mathcal{G}_i(t) = \mathcal{O}_i^{\text{static}}\ \forall t$;
- compute the BRS $\mathcal{V}_i^{\text{basic}}(t, t_i^{\text{STA}})$ defined in (3.25).
- **Trajectory and controller of** Q_i
- compute the optimal controller $u_i^{\text{basic}}(\cdot)$ given by (3.27);
- determine the trajectory $z_i(\cdot)$ using vehicle dynamics (3.22) and the control $u_i^{\text{basic}}(\cdot)$;
- output the trajectory and optimal controller for Q_i.
- **Induced obstacles by** Q_i
- given the trajectory $z_i(\cdot)$, compute the induced obstacles $\mathcal{O}_k^i(t)$ given by (3.24) for all $k > i$.

3.2.2.3 Robust Trajectory Tracking (RTT)

In the basic STP algorithm, lower priority vehicles know the trajectories of all higher-priority vehicles. The region that a lower-priority vehicle needs to avoid is thus simply given by the danger zones around these trajectories; however, disturbances and incomplete information significantly complicate the STP scheme. Committing to exact trajectories is no longer possible, since the disturbance $d_i(\cdot)$ is *a priori* unknown.

Thus, the induced obstacles $\mathcal{O}_i^j(t)$ are no longer just the danger zones centered around positions. Fortunately, it is straight forward to apply the FaSTrack method outlined in Section 5.2 to robustify STP. In this section, we provide some intuitive details; for the formal mathematics of FaSTrack, please refer to Section 5.2.

Even though it is impossible to commit to and track an exact trajectory in the presence of disturbances, it may still be possible to instead *robustly* track a feasible *nominal* trajectory with a bounded error at all times. If this can be done, then the tracking error bound can be used to determine the induced obstacles. Here, computation is done in two phases: the *planning phase* and the *disturbance rejection phase*. In the planning phase, a nominal trajectory $z_{r,j}(\cdot)$ is computed that is feasible in the absence of disturbances. This planning is done for a reduced control set $\mathcal{U}^p \subset \mathcal{U}$, as some margin is needed to reject unexpected disturbances while tracking the nominal trajectory.

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

Let z_j and $z_{r,j}$ denote the states of the actual vehicle Q_j and the virtual evader, respectively, and define the tracking error $e_j = z_j - z_{r,j}$. When the error dynamics are independent of the absolute state as in (3.28) (and also (7) in [111]), we can obtain error dynamics of the form

$$\begin{aligned} \dot{e_j} &= f_{e_j}(e_j, u_j, u_{r,j}, d_j), \\ u_j &\in \mathcal{U}_j, u_{r,j} \in \mathcal{U}_j^p, d_j \in \mathcal{D}_j, \quad t \leq 0 \end{aligned} \tag{3.28}$$

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

$$\mathcal{E}_j = \{e_j : \|p_{e_j}\|_2 \leq R_{\text{EB}}\}, \tag{3.29}$$

where p_{e_j} denotes the position coordinates of e_j and R_{EB} is a design parameter. We next solve a reachability problem with its complement $\mathcal{E}_j^c$, the set of tracking errors violating the error bound, as the target in the space of the error dynamics. From $\mathcal{E}_j^c$, we compute the following BRS:

$$\begin{aligned} \mathcal{V}_j^{\text{EB}}(t,0) = \{y : \forall u_j(\cdot) \in \mathbb{U}_j, \exists u_{r,j}(\cdot) \in \mathbb{U}_j^p, \exists d_j(\cdot) \in \mathbb{D}_i, \\ e_j(\cdot) \text{ satisfies (3.28)}, e_j(t) = y, \\ \exists s \in [t,0], e_j(s) \in \mathcal{E}_j^c\}, \end{aligned} \tag{3.30}$$

where the Hamiltonian to compute the BRS is given by:

$$H_j^{\mathrm{EB}}(e_j, \lambda) = \max_{u_j \in \mathcal{U}_j} \min_{u_r \in \mathcal{U}_j^p, d_j \in \mathcal{D}_j} \lambda \cdot f_{e_j}(e_j, u_j, u_{r,j}, d_j). \tag{3.31}$$

Letting $t \to -\infty$, we obtain the infinite-horizon control-invariant set $\Omega_j := \lim_{t \to -\infty} \left(\mathcal{V}_j^{\mathrm{EB}}(t, 0)\right)^c$. If Ω_j is nonempty, then the tracking error e_j at flight time is guaranteed to remain within $\Omega_j \subseteq \mathcal{E}_j$ provided that the vehicle starts inside Ω_j and subsequently applies the feedback control law

$$\kappa_j(e_j) = \arg\max_{u_j \in \mathcal{U}_j} \min_{u_r \in \mathcal{U}_j^p, d_j \in \mathcal{D}_j} \lambda \cdot f_{e_j}(e_j, u_j, u_{r,j}, d_j). \tag{3.32}$$

The induced obstacles by each higher-priority vehicle Q_j can thus be obtained by:

$$\begin{aligned} \mathcal{O}_i^j(t) &= \{z_i : \exists y \in \mathcal{P}_j(t), \|p_i - y\|_2 \leq R_c\} \\ \mathcal{P}_j(t) &= \{p_j : \exists h_j, (p_j, h_j) \in \mathcal{M}_j(t)\} \\ \mathcal{M}_j(t) &= \Omega_j + z_{r,j}(t), \end{aligned} \tag{3.33}$$

where the "+" in (3.33) denotes the Minkowski sum[1]. Intuitively, if Q_j is tracking $z_{r,j}(t)$, then it will remain within the error bound Ω_j around $z_{r,j}(t)$ $\forall t$. This is precisely the set $\mathcal{P}_j(t)$. The induced obstacles can then be obtained by augmenting a danger zone around this set. Finally, we can obtain the total obstacle set $\mathcal{G}_i(t)$ using:

$$\mathcal{G}_i(t) = \mathcal{O}_i^{\mathrm{static}} \cup \bigcup_{j=1}^{i-1} \mathcal{O}_i^j(t) \tag{3.34}$$

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

$$\tilde{\mathcal{G}}_i(t) = \mathcal{G}_i(t) + \Omega_i. \tag{3.35}$$

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

$$\begin{aligned} \mathcal{V}_i^{\mathrm{rtt}}(t, t_i^{\mathrm{STA}}) = &\{y : \exists u_i(\cdot) \in \mathbb{U}_i^p, z_i(\cdot) \text{ satisfies (3.22)}, \\ &\forall s \in [t, t_i^{\mathrm{STA}}], z_i(s) \notin \tilde{\mathcal{G}}_i(t), \\ &\exists s \in [t, t_i^{\mathrm{STA}}], z_i(s) \in \tilde{\mathcal{L}}_i, z_i(t) = y\} \end{aligned} \tag{3.36}$$

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

The BRS $\mathcal{V}_i^{\text{rtt}}(t, t_i^{\text{STA}})$ can be obtained by solving (3.20) using the Hamiltonian:

$$H_i^{\text{rtt}}(z_i, \lambda) = \min_{u_i \in \mathcal{U}_i^p} \lambda \cdot f_i(z_i, u_i) \tag{3.37}$$

The corresponding optimal control for reaching $\tilde{\mathcal{L}}_i$ is given by:

$$u_i^{\text{rtt}}(t) = \arg\min_{u_i \in \mathcal{U}_i^p} \lambda \cdot f_i(z_i, u_i). \tag{3.38}$$

The nominal trajectory $z_{r,i}(\cdot)$ can thus be obtained by using vehicle dynamics in the absence of disturbances (given by (3.22)) with the optimal control $u_i^{\text{rtt}}(\cdot)$ given by (3.38). From the resulting nominal trajectory $z_{r,i}(\cdot)$, the overall control policy to reach $\mathcal{L}_i$ can be obtained via (3.32). The robust trajectory tracking method can be summarized in Algorithm 2.

3.2.3 City Environment Simulation

We now illustrate STP algorithm using a 50-vehicle UAV system where UAVs are flying through a city environment. This setup can be representative of many UAV applications, such as package delivery, aerial surveillance, etc.

3.2.3.1 Setup

We grid the City of San Francisco (SF) in California, USA, and use it as our position space, as shown in Fig. 3.10. Each box in Fig. 3.10 represents a 1 km^2 area of SF. The origin point for the vehicles is denoted by the blue star. Four different areas in the city are chosen as the destinations for the vehicles. Mathematically, the target sets $\mathcal{L}_i$ of the vehicles are circles of radius r in the position space, i.e. each vehicle is trying to reach some desired set of positions. In terms of the state space z_i, the target sets are defined as

$$\mathcal{L}_i = \{z_i : \|p_i - c_i\|_2 \leq r\} \tag{3.39}$$

where c_i are centers of the target circles. In this simulation, we use $r = 100$ m. The four targets are represented by four circles in Fig. 3.10. The destination of each vehicle is chosen randomly from these four destinations. Finally, some areas in downtown SF and the city hall are used as representative static obstacles for the STP vehicles, denoted by black contours in Fig. 3.10.

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

$$\begin{aligned} \dot{p}_{x,i} &= v_i \cos\theta_i + d_{x,i} \\ \dot{p}_{y,i} &= v_i \sin\theta_i + d_{y,i} \\ \dot{\theta}_i &= \omega_i, \\ \underline{v} \leq v_i \leq \bar{v},\ &|\omega_i| \leq \bar{\omega}, \\ \|(d_{x,i}, d_{y,i})\|_2 &\leq d_r \end{aligned} \tag{3.40}$$

Algorithm 2: Robust trajectory tracking algorithm (STP algorithm in the presence of disturbances)

input : Set of vehicles $Q_i, i = 1, \ldots, N$ in the descending priority order;
Vehicle dynamics (3.17) and initial states z_i^0;
Vehicle destinations $\mathcal{L}_i$ and static obstacles $\mathcal{O}_i^{\text{static}}$.

output: The nominal trajectories to respective destinations for all vehicles;
Goal-satisfaction controller for all vehicles.

for $i = 1 : N$ **do**
- **Computation of tracking error bound for** Q_i
- obtain the error dynamics given by (3.28);
- decide on a reduced control authority $\mathcal{U}_i^p$ for the planning phase, and choose a parameter R_{EB} to conservatively bound the tracking error;
- compute the BRS $\mathcal{V}_i^{\text{EB}}(t, 0)$ using (3.30);
- compute the Ω_i using the converged BRS $\mathcal{V}_i^{\text{EB}}$;
- **if** $\Omega_i \neq \emptyset$ **then**
  - the error bound on the tracking error is given by Ω_i;
- **else**
  - recompute the tracking error bound using a larger R_{EB};
- **Computation of obstacles for** Q_i
- determine the total obstacle set $\mathcal{G}_i(t)$, given in (3.23). In the case $i = 1$, $\mathcal{G}_i(t) = \mathcal{O}_i^{\text{static}}\ \forall t$;
- using Ω_i, determine the augmented obstacle set $\tilde{\mathcal{G}}_i(t)$, given in (3.35).
- **Trajectory planning for** Q_i
- compute the reduced target set $\tilde{\mathcal{L}}_i$;
- compute the BRS $\mathcal{V}_i^{\text{rtt}}(t, t_i^{\text{STA}})$ defined in (3.36).
- **Trajectory and controller of** Q_i
- compute the nominal controller $u_i^{\text{rtt}}(\cdot)$ given by (3.38);
- determine the nominal trajectory $z_{r,i}(\cdot)$ using vehicle dynamics (3.22) and the control $u_i^{\text{rtt}}(\cdot)$;
- the overall goal satisfaction controller for Q_i is given by (3.32);
- output the nominal trajectory and the optimal tracking controller for Q_i.
- **Induced obstacles by** Q_i
- given the trajectory $z_{r,i}(\cdot)$ and the tracking error bound Ω_i, compute the induced obstacles $\mathcal{O}_k^i(t)$ given by (3.33) for all $k > i$.

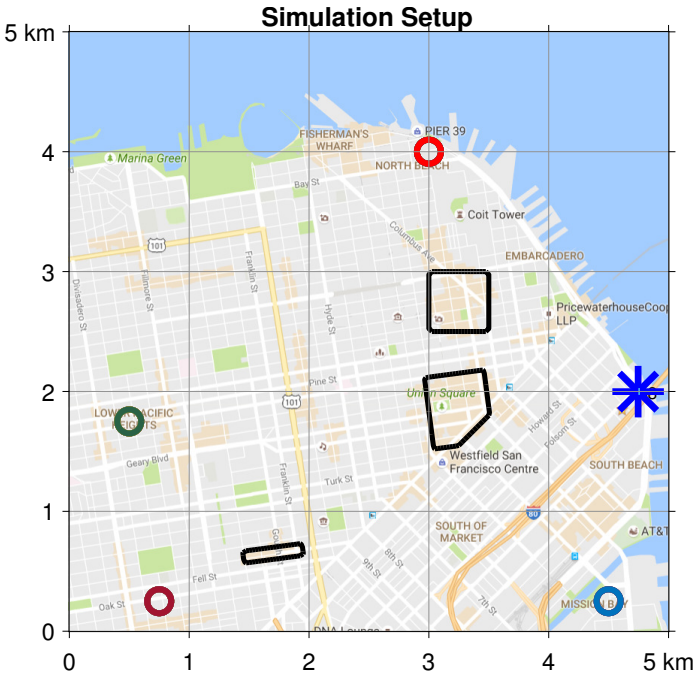


Figure 3.10: Simulation setup. A 25 km^2 area in the City of San Francisco is used as the space for the STP vehicles. STP vehicles originate from the blue star and go to one of the four destinations, denoted by circles. Tall buildings in the downtown area are used as static obstacles, represented by the black contours.

where $z_i = (p_{x,i}, p_{y,i}, \theta_i)$ is the state of vehicle Q_i, $p_i = (p_{x,i}, p_{y,i})$ is the position, θ_i is the heading, and $d = (d_{x,i}, d_{y,i})$ represents Q_i's disturbances, for example wind, that affect its position evolution. The control of Q_i is $u_i = (v_i, \omega_i)$, where v_i is the speed of Q_i and ω_i is the turn rate; both controls have a lower and upper bound. To make our simulations as close as possible to real scenarios, we choose velocity and turn-rate bounds as $\underline{v} = 0$ m/s, $\bar{v} = 25$ m/s, $\bar{\omega} = 2$ rad/s, aligned with the modern UAV specifications [1, 118]. The disturbance bounds are chosen to be either $d_r = 6$ m/s or $d_r = 11$ m/s. These conditions correspond to *moderate breeze* and *strong breeze* respectively on the Beaufort scale [144]. The scheduled times of arrival t_i^{STA} for all vehicles are chosen to be all 0 for a high UAV density condition. For medium and low density conditions, we separated the t_i^{STA} for the vehicles by 5 s and 10 s respectively, with the latest t_i^{STA} being 0. Note that we have used same dynamics and input bounds across all vehicles for clarity of illustration; however, STP can easily handle more general systems of the form in which the vehicles have different control bounds, t_i^{STA} and dynamics.

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

3.2.3.2 High UAV Density with 6 m/s Wind

As per Algorithm 2, we start with Q_1 and sequentially compute the optimal control policy κ_j and the latest departure time t_j^{LDT} for each vehicle. To compute the error bound $\mathcal{E}_j$ on the tracking error of vehicle j, we choose $R_{\text{EB}} = 5$ m and use the reduced control authority $\mathcal{U}_j^p = \{(v_{r,j}, \omega_{r,j}) : 11 \text{ m/s} \leq v_{r,j} \leq 13 \text{ m/s}, |\omega_{r,j}| \leq 1.2 \text{ rad/s}\}$. Given dynamics in (3.40), the error dynamics between Q_j and the virtual evader are given by [111]:

$$\begin{aligned} \dot{e}_{x,j} &= v_j \cos(e_{\theta,j}) - v_{r,j} + \omega_{r,j} e_{y,j} + d_{x,j} \\ \dot{e}_{y,j} &= v_{r,j} \sin(e_{\theta,j}) - \omega_{r,j} e_{x,j} + d_{y,j} \\ \dot{e}_{\theta,j} &= \omega_j - \omega_{r,j}, \end{aligned} \tag{3.41}$$

where $e_j = (e_{x,j}, e_{y,j}, e_{\theta,j})$ is the tracking error in the three states of Q_j. Given relative dynamics, we compute the BRS $\mathcal{V}^{\text{EB}}$ using (3.30) and evaluate the infinite-horizon control invariant set Ω_j. For all the BRS computations in this simulation, we use Level Set Toolbox [110]. In presence of moderate winds, the obtained tracking error bound is 5 m. This means that given any trajectory (which is a sequence of states over time) of vehicle, winds can at most cause a deviation of 5 m from this trajectory

at all times. Consequently, the vehicle will be within a distance of 5 m from the trajectory. Note that since all STP vehicles have same dynamics, the error bound is also same for all vehicles.

The optimal control policy for Q_j is thus given by $\kappa_j(e_j)$ in (3.32). However, to compute the relative state e_j, the nominal trajectory $z_{r,j}$ for Q_j is required. Using Ω_j, we compute the obstacles induced by the higher-priority vehicles for Q_j and the obstacle induced by Q_j for the lower-priority vehicles. These obstacles are given by (3.35) and (3.33) respectively. Note that since disturbance directly impacts the computation of tracking error bound, these obstacles also grow as disturbance magnitude increases. We will illustrate the effect of disturbance magnitude on the trajectories of vehicles in 3.2.3.3.

The nominal trajectory can now be obtained by computing $\mathcal{V}_j^{\text{rtt}}$ in (3.36) and executing the corresponding control policy u_j^{rtt} in (3.38), starting from the initial state z_j^0. Finally, the latest departure time t_j^{LDT} is given by $\arg\sup_t z_j^0 \in \mathcal{V}_j^{\text{rtt}}(t, t_j^{\text{STA}})$. It is important to note that since the BRS $\mathcal{V}_j^{\text{rtt}}$ is computed backwards starting from the scheduled time of arrival t_j^{STA}, (a) the latest departure time t_j^{LDT} directly depends on and varies with t_j^{STA} and (b) it directly impacts the obstacles that Q_j needs to avoid in its trajectory towards its destination. We will illustrate the effect of the scheduled time of arrival on the trajectories of vehicles also in 3.2.3.3.

The resulting trajectories of vehicles for $d_r = 6$ m/s and $t_j^{\text{STA}} = 0\ \forall j$ at different times are shown in Fig. 3.11. As evident from the figures, the vehicles remain clear of all the static obstacles (the black contours) and make steady progress towards reaching their destinations. The vehicles whose destinations are relatively closer need less time to travel to their destinations and depart later. Note that the shown trajectories are simulated under uniformly random disturbance (i.e., for every vehicle, the disturbance is uniformly sampled from a circle of radius $d_r = 6$ m/s at each time step), but the STP algorithm guarantees safety and reactivity despite the worst case disturbance, as we show later in this section.

The full trajectories of vehicles are shown in Fig. 3.12a. All vehicles reach their respective destinations. A zoomed-in version of Fig. 3.12a near the red target (Fig. 3.13) illustrates that vehicles are also outside each other's danger zones (circles around the vehicles) as required.

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

Although the plotted trajectories in Fig. 3.12a are for a particular realization of the (uniformly random) disturbances, the STP algorithm guarantees obstacle avoidance regardless of the realized disturbance. To illustrate this, we plot the trajectories

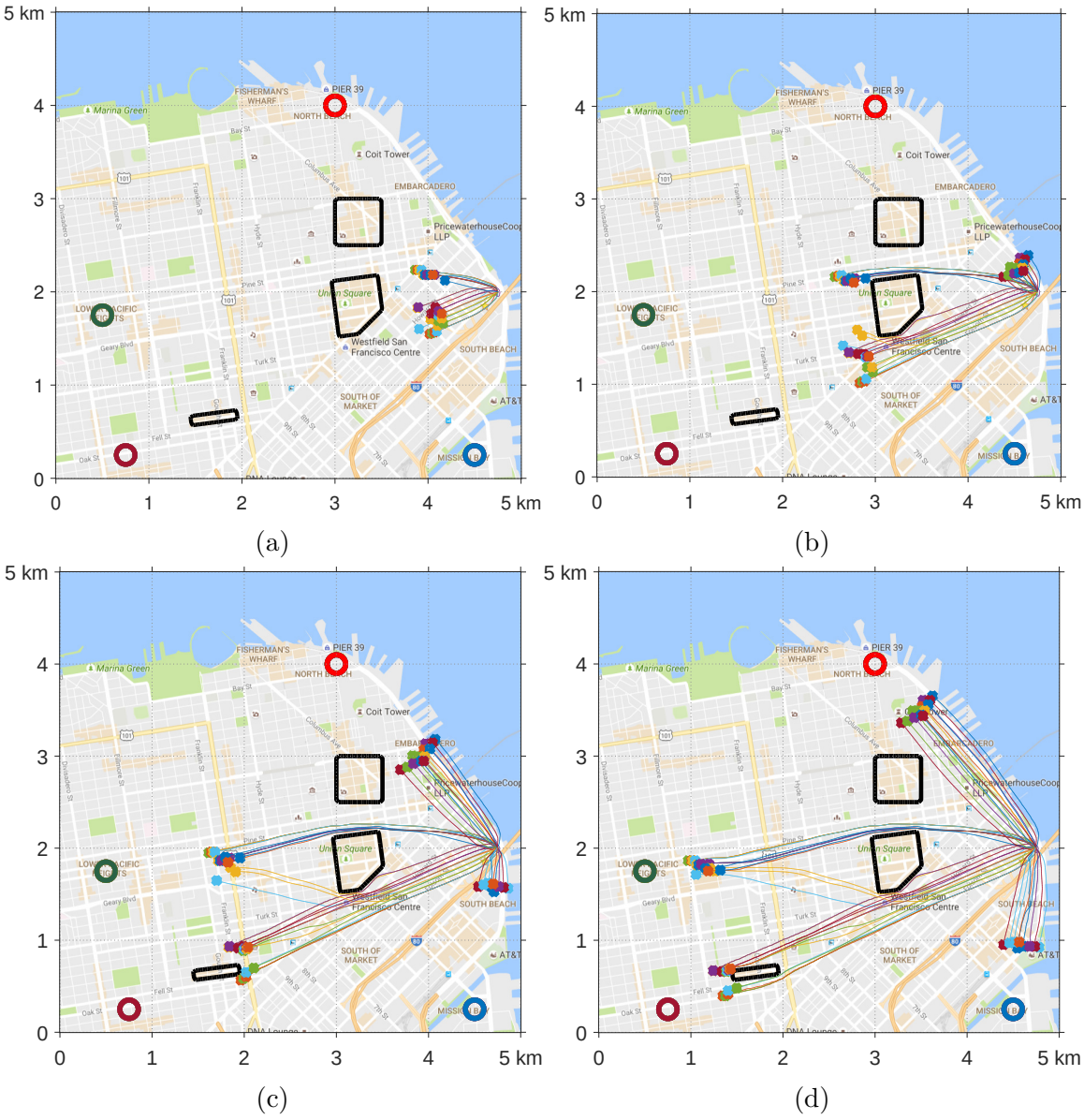
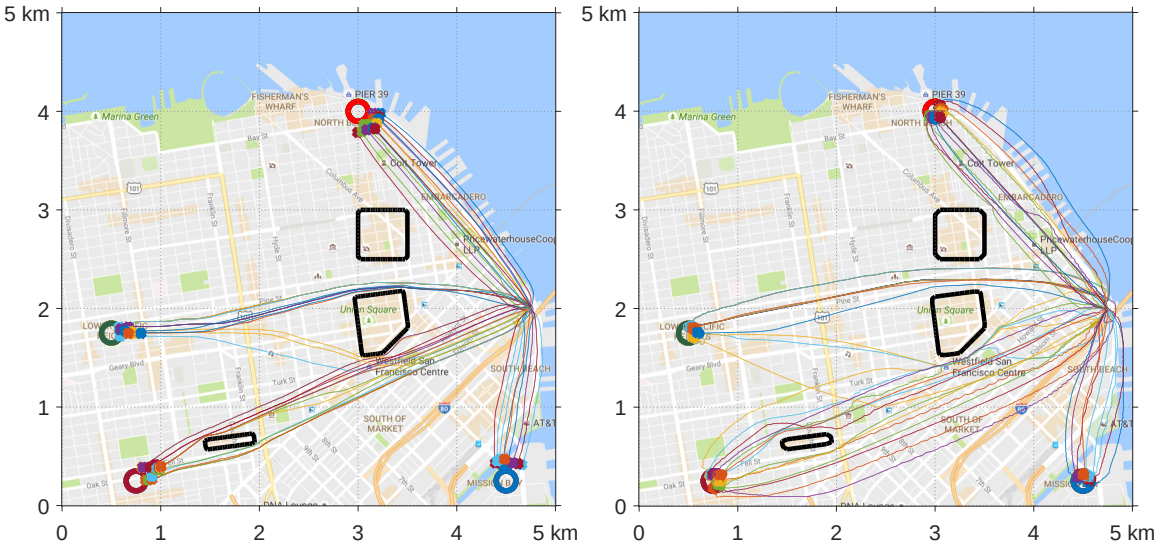


Figure 3.11: Snapshots of vehicle trajectories at different times for uniform disturbance with $d_r = 6$ m/s. The vehicles remain clear of all static obstacles despite the disturbance in the dynamics.



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

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

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

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

Figure 3.12: Effect of the disturbance magnitude and the scheduled times of arrival on vehicle trajectories. All trajectories are simulated under uniformly random disturbance. The relative separation in the scheduled times of arrival of vehicles determines the number of lanes between a pair of origin and destination, and more and more tarjectories become time-separated as this relative separation increases. The disturbance magnitude determines the relative separation between different lanes, and more and more tarjectories become state-separated as the disturbance increases.

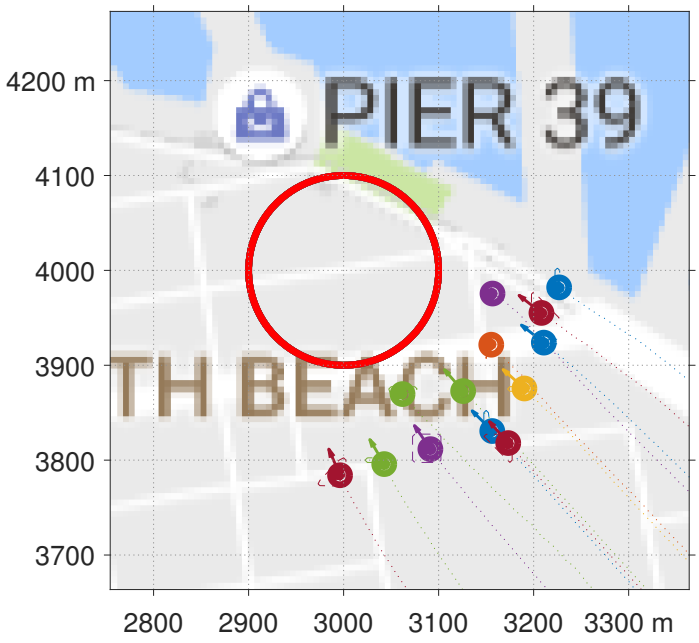


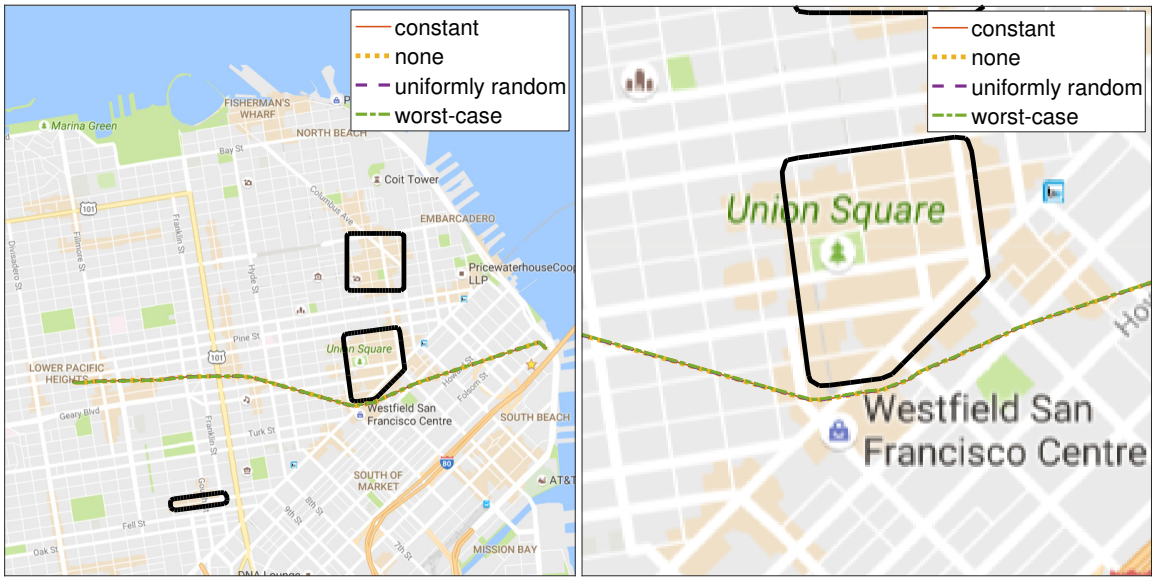
Figure 3.13: Zoomed-in version of vehicle trajectories near the red target in Fig. 3.12a. The STP algorithm ensures that the vehicles are outside each other's danger zones (circles around the vehicles).

of a particular vehicle (Q_{17}) for three different disturbance realizations – uniform, worst-case, and constant wind from the west direction – in Fig. 3.14. One can see that the trajectories are nearly indistinguishable even in the zoomed-in plot on the right.

Fig. 3.15 shows the minimum distance between Q_{17} and other vehicles. One can see that this distance is never below 10 m, which is the minimum required separation between the vehicles. The separation distance is small in the beginning because the vehicles need to depart as soon as possible while maintaining this minimum required separation.

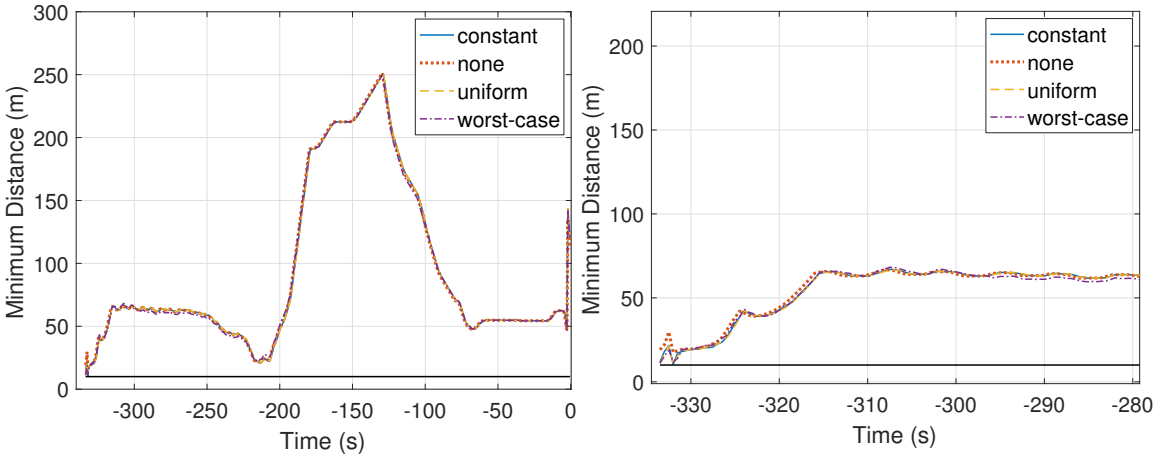
Fig. 3.16 shows the optimal control action in terms of the turn rate and the linear speed of each vehicle, demonstrating that the tracking control law reacts to tracking error due to disturbances in an intuitive way. Both subplots show the control law when the vehicle is facing east. Fig. 3.16a indicates, for example, that generally when the tracking error is such that the vehicle is too far to the right compared to the nominal position, the optimal tracking control is to turn left. Fig. 3.16b indicates, for example, that when the vehicle is too far ahead of the nominal position, it should slow down. The optimal turn rate and linear speed controls ensure that the vehicle never deviates from the nominal trajectory for more than 5 m.

The average trajectory computation time per vehicle is 2 seconds using a CUDA implementation of the level set toolbox on a desktop computer with a Core i7 5820K



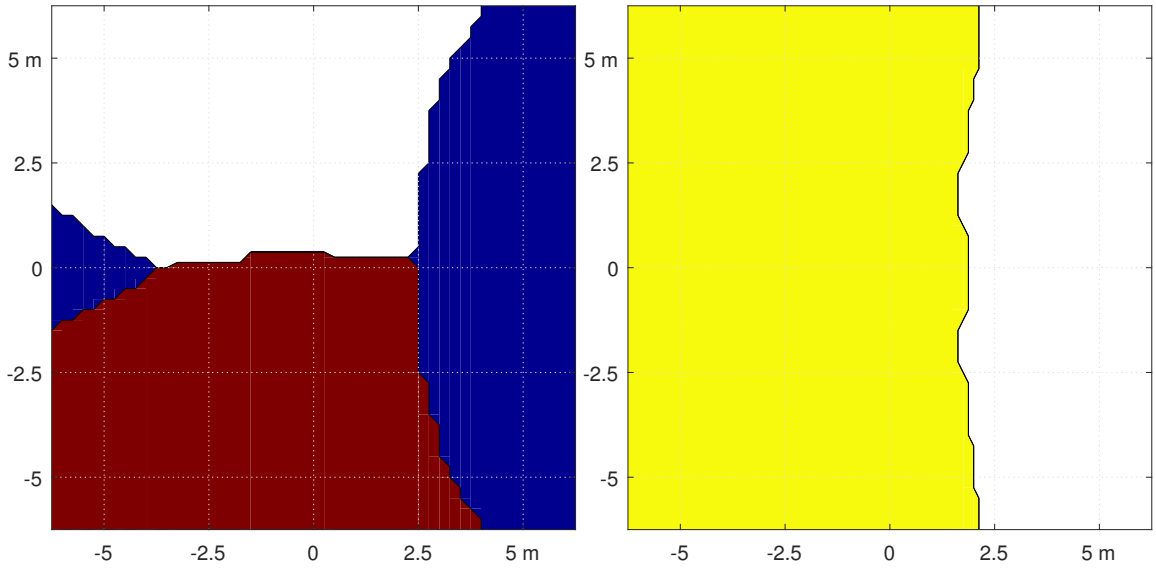
(a) (b)

Figure 3.14: Trajectories of Q_{17} under different types of bounded disturbances – constant wind from the west, no wind, uniformly random wind, and worse-case wind. The right plot zooms in on the beginning of the left plot.



(a) (b)

Figure 3.15: Minimum distance to other vehicles for Q_{17} under different types of bounded disturbances – constant wind from the west, no wind, uniformly random wind, and worse-case wind. The right plot zooms in on the beginning of the left plot.



(a) Turn rate control as a function of tracking error. White: turn right; red: turn left; blue: go straight.

(b) Linear speed control as a function of tracking error. Yellow: high speed; white: low speed.

Figure 3.16: Optimal tracking control law for each vehicle for a heading of zero degrees (facing towards the right).

processor and two GeForce GTX Titan X graphics processing units. Note that all this computation is done offline and the resulting optimal policy $\kappa_j(e_j)$ is obtained as a lookup table. In real time, neither any computation nor any communication between vehicles is required. Only a lookup table query is required, and this can be performed very quickly in real time. This illustrates the capability of STP as a provably safe trajectory planning algorithm for large multi-vehicle systems. Without CUDA, the computation time per vehicle is 33 seconds using a C++ implementation of the level set toolbox, and 200 seconds using a MATLAB implementation.

3.2.3.3 Effects of Disturbance and Scheduled Time of Arrival

In this section, we illustrate how the disturbance bound d_r in (3.40) and the relative t^{STA}s of vehicles affect the vehicle trajectories. For this purpose, we simulate the STP algorithm for four additional scenarios:

- Case-0: $d_r = 6$ m/s, $t_i^{\text{STA}} = 0\ \forall i$ (moderate breeze, high UAV density)
- Case-1: $d_r = 11$ m/s, $t_i^{\text{STA}} = 0\ \forall i$ (strong breeze, high UAV density)
- Case-2: $d_r = 6$ m/s, $t_i^{\text{STA}} = 5(i-1)\ \forall i$ (moderate breeze, medium UAV density)
- Case-3: $d_r = 11$ m/s, $t_i^{\text{STA}} = 5(i-1)\ \forall i$ (strong breeze, medium UAV density)

- Case-4: $d_r = 11$ m/s, $t_i^{\text{STA}} = 10(i-1)\ \forall i$ (strong breeze, low UAV density)

The interpretation $t_i^{\text{STA}} = 5(i-1)$ is that the scheduled time of arrival of any two consecutive vehicles is separated by 5 s, which represents a medium vehicle density scenario; a separation of 10 s represents a low vehicle density scenario. $d_r = 6$ m/s and $d_r = 11$ m/s correspond to the moderate breeze and strong breeze respectively on Beaufort wind force scale [144].

Intuitively, as d_r increases, it is harder for a vehicle to closely track a particular nominal trajectory, which results in a higher tracking error bound. As mentioned previously, with a 6 m/s wind speed, the tracking error bound is 5 m; however, with an 11 m/s wind speed, the tracking error bound becomes 35 m. Thus, the vehicles need to be separated more from each other in space to ensure that they do not enter each other's danger zones. This is also evident from comparing the results corresponding to Case-0 (Fig. 3.12a) and Case-1 (Fig. 3.12b). As the disturbance magnitude increases from $d_r = 6$ m/s (moderate breeze) to $d_r = 11$ m/s (strong breeze), the vehicles' trajectories get farther apart from each other. Since t^{STA} is same for all vehicles, the vehicles trajectories are still predominately *state-separated* trajectories.

We next compare Case-0 and Case-2. The difference between these two cases is that vehicles have a 5 second separation in their schedule times of arrival in Case-2. When vehicles Q_i and Q_j $(j > i)$ have same scheduled time of arrival and are going to the same destination, they are constrained to travel at the same time to make sure they reach the destination by the designated t^{STA}. However, since Q_i is high-priority, it gets access to the optimal trajectory (in terms of the total time of travel to destination) and Q_j has to settle for a relatively sub-optimal trajectory. Thus, all vehicles going to a particular destination take different trajectories creating a "band" of trajectories between the origin and the destination, as shown in Fig. 3.12a; the high-priority vehicles take a relatively straight trajectory between the origin and the destination whereas the low-priority vehicles take a (relatively sub-optimal) curved trajectory. If we think of an air highway between the origin and the destination, then vehicles take different lanes of that highway to reach the destination in Case-0. Thus, the trajectories of vehicles in this case are *state-separated*. However, when $t_j^{\text{STA}} > t_i^{\text{STA}}$, then Q_j is not bound to travel at the same time as Q_i; it can wait for Q_i to depart and take a shorter trajectory later on. Thus, vehicles travel in a single (optimal) lane in this case, as shown in Fig. 3.12c. In other words, they take the same trajectory to the destination, but at different times. Thus, the trajectories of vehicles in this case are *time-separated*.

Note that the exact number of lanes depends on *both* the disturbance and separation of scheduled times of arrival. As the disturbance increases, vehicles need to be separated more from each other to ensure safety. A larger arrival time difference between vehicles is also able to ensure this separation even if the vehicles were to take the same lane. As shown in Fig. 3.12d, a difference of 5 s in the t^{STA}'s is not suffi-

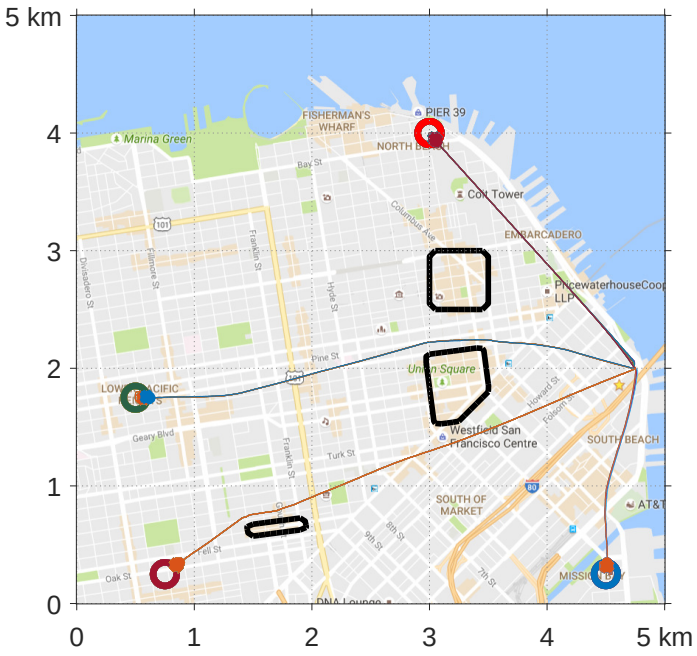


Figure 3.17: Vehicle trajectories for Case-4: $d_r = 11$ m/s, $t_i^{\text{STA}} = 10(i-1)$. Since different vehicles have different scheduled times of arrival, there is a single lane between every origin-destination pair.

cient to achieve a single lane behavior for stronger 11 m/s wind conditions. However, the number of lanes is significantly less than that in Case-1 (Fig. 3.12b). Finally, a separation of 10 s in t^{STA}'s ensure that we get the single lane behavior even in the presence of 11 m/s winds, leading to *time-separated* trajectories, as shown in Fig. 3.17. Videos of the simulations can be found at https://youtu.be/1ocaBGZqSAE.

Overall, the relative magnitude of disturbance and scheduled times of arrival separation determines the number of lanes and type of trajectories that emerge out of the STP algorithm. For a fixed disturbance magnitude, as the separation in the scheduled times of arrival of vehicles increases, the number of lanes between a pair of origin and destination decreases, and more and more trajectories become time-separated. On the other hand, for a fixed separation in the scheduled times of arrival of vehicles, as the disturbance magnitude increases, the number of lanes between a pair of origin and destination increases, and more and more trajectories become state-separated.

3.2.4 Multi-city Environment Simulation

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

3.2.4.1 Setup

We grid the San Francisco Bay Area in California, US and use it as our state space, as shown in Fig. 3.18. We consider the UAVs flying to and from four cities: Richmond, Berkeley, Oakland, and San Francisco. The blue region in Fig. 3.18 represents bay. This environment is different from the city environment in Section 3.2.3 in that now the UAVs need to fly for longer distances and through a high-density vehicle environment with strong winds, but have very few static obstacles like tall buildings.

Each box in Fig. 3.18 represents a 25 km^2 area. The vehicles are flying to and from the four cities indicated by the four circles. The origin and the destination of each vehicle is chosen randomly from these four cities. The vehicle dynamics are given by (3.40). We choose velocity and turn-rate bounds as $\underline{v} = 0$ m/s, $\bar{v} = 25$ m/s, $\bar{\omega} = 2$ rad/s. The disturbance bound is chosen as $d_r = 11$ m/s, which corresponds to *strong breeze* on Beaufort wind force scale [144]. The scheduled time of arrival t^{STA} for vehicles are chosen as $5(i-1)$ s.

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

3.2.4.2 Results

The trajectory planning for the vehicles is done using RTT algorithm, similar to that in Section 3.2.3.2. The resulting trajectories of vehicles are shown in Fig. 3.19a. Once again, the vehicles remain clear of all other vehicles and reach their respective destinations. Given the separation between the scheduled times of arrival, the trajectories are predominately *time-separated*, with roughly two lanes for each pair of cities (one for going from city A to city B and another for from city B to city A). A high-density of vehicles is achieved in the center since the 4 trajectories are intersecting in the center (Richmond-Oakland, Oakland-Richmond, Berkeley-San Francisco, San Francisco-Berkeley), but the STP algorithm ensures safety despite this high-density, as shown in the zoomed-in version of center at an intermediate time when a large number of vehicles are passing through the central region (Fig. 3.19b).

Finally, we simulate the system for the case where $t_i^{\text{STA}} = 0 \; \forall i$. As evident from Fig. 3.20, we get multiple lanes between each pair of cities in this case and trajectories become predominately state-separated, as we expect based on the discussion in Section 3.2.3.3.

The average computation time per vehicle is 4 minutes using a CUDA implementation of the Level Set Toolbox on a desktop computer with a Core i7 5820K

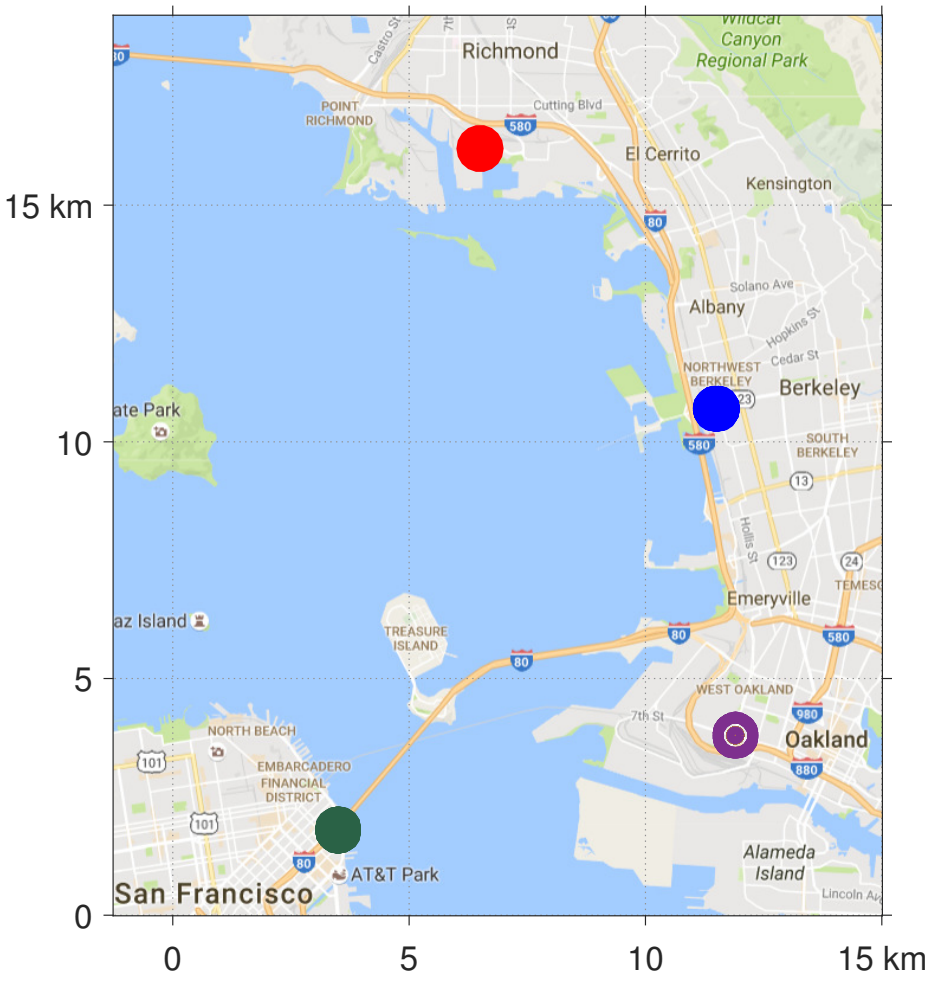
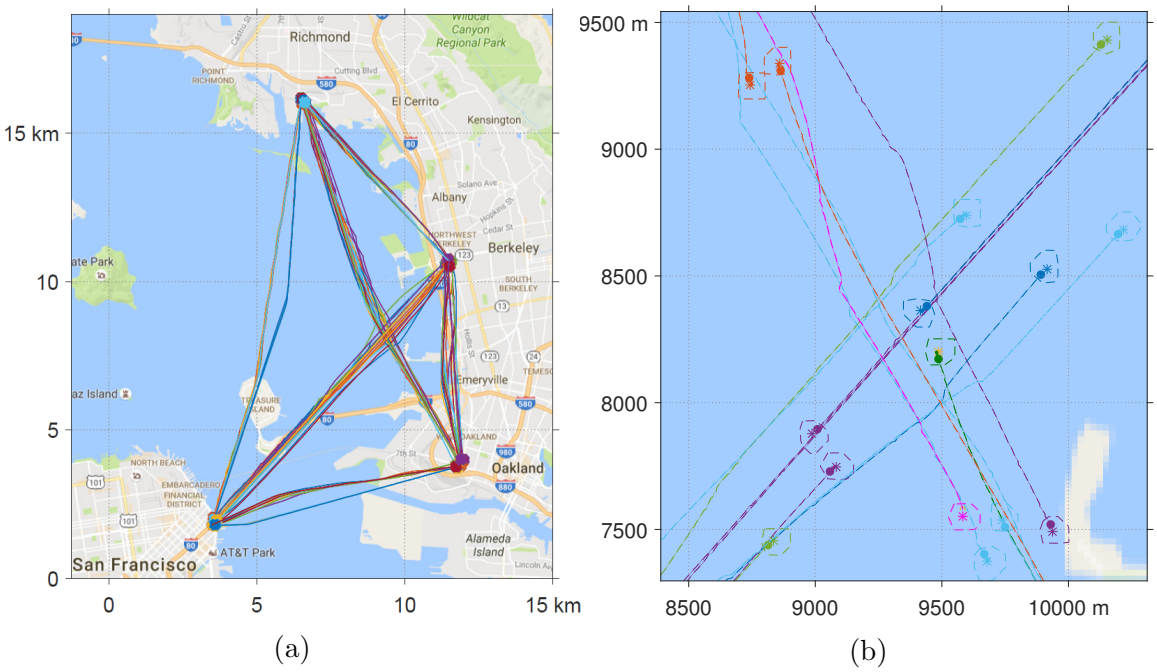


Figure 3.18: Multi-city simulation setup. A 300 km^2 area of San Francisco Bay Area is used as the state-space for vehicles. STP vehicles fly to and from the four cities indicated by the four circles. The simulations are performed under the strong winds condition with $d_r = 11$ m/s.



(a) (b)

Figure 3.19: (a) Trajectories obtained from the STP algorithm for the multi-city simulation with $d_r = 11$ m/s, $t_i^{\text{STA}} = 5(i-1)$. (b) Zoomed-in version of the central area. A high density of vehicles is achieved at the center because of the intersection of several trajectories; however, the STP algorithm still ensures that vehicles do not enter each other's danger zones and reach their destinations.

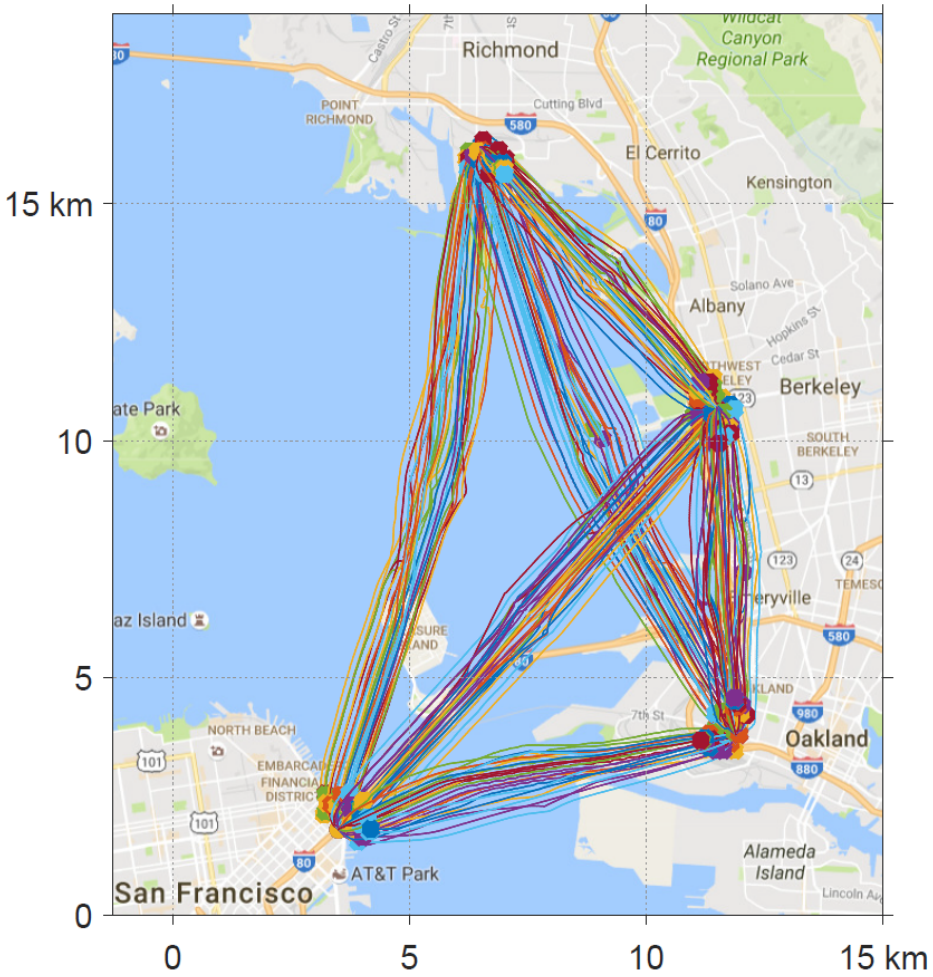


Figure 3.20: Vehicle trajectories for $d_r = 11$ m/s, $t_i^{\text{STA}} = 0$. Since different vehicles have same scheduled times of arrival, a multiple-lane behavior is observed between every pair of cities.

processor and two GeForce GTX Titan X graphics processing units. The computation time is much longer than in the previous simulation in SF because of the larger space over which planning is done. Once again all the computation is done offline and only a lookup table query is required in real-time, which can be performed very efficiently. This simulation illustrates the scalability and the potential of deploying the STP algorithm for provably safe trajectory planning for large multi-vehicle systems.

3.2.5 Conclusion

Provably safe multi-vehicle trajectory planning in an important problem that needs to be addressed to ensure that vehicles can fly in close proximity of each other. Recently, the STP algorithm was proposed for multi-vehicle trajectory planning problem that scales linearly with the number of vehicles. We illustrate the full potential of the algorithm by using it for large-scale multi-vehicle trajectory planning problems under different flying conditions. We demonstrate how different types of space-time trajectories emerge naturally out of the algorithm for different disturbance conditions and other problem parameters. The reactivity of the obtained controller is also demonstrated under different wind conditions.

Chapter 4

System Decomposition

There are several 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 BRT or 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 [98, 107, 114].

In this chapter, we introduce two new methods for decomposing a system into smaller subsystems. Both methods involve computing a BRS or BRT in lower-dimensional subspaces, and then reconstructing the full-dimensional BRS or BRT. Section 4.1 presents a method based on the new concept of "self-contained subsystems" common found in vehicle dynamics and mechanical systems; this method substantially reduces computational requirements without incurring any additional approximation errors, as long as the system dynamics are of a compatible form. Section 4.2 presents a complementary method based on the concept of virtual disturbances, which eliminates state coupling in system dynamics at the cost of conservative approximation error. Fortunately, the trade off between degree of conservatism and computational requirements can be tuned naturally.

4.1 Decomposition of Reachable Sets and Tubes for a Class of Nonlinear Systems

This section is an adaptation of the paper in [33].

In this section, 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 [26, 130]. 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 section is compatible with any methods that compute BRSs and BRTs, such as [6, 31, 39, 65, 115] and others mentioned earlier. In addition, when different decomposition methods are combined together, even more dimensionality reduction can be achieved. This section will be presented as follows:

- In Section 4.1.1 we introduce all the definitions needed for our proposed decomposition technique.
- In Sections 4.1.2 and 4.1.3 we present our theoretical results related to decomposing BRSs for systems involving a control variable, but *not* involving a disturbance variable.
- In Section 4.1.4 we show how BRTs can be decomposed.
- In Section 4.1.5 we demonstrate our decomposition method on high-dimensional systems.
- In Section 4.1.6 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 [111] throughout the subsections to validate our theory.

4.1.1 Problem Formulation

In this section, we seek to obtain the BRSs and BRTs in Definitions 1 to 4 in Section 2.2.1 via computations in lower-dimensional subspaces under the assumption that the system (2.1) can be decomposed into self-contained subsystems (SCS) (4.77). Such a decomposition is common, since many systems involve components that are loosely coupled. In particular, the evolution of position variables in vehicle dynamics is often weakly coupled though other variables such as heading.

4.1.1.1 Definitions

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 belong to both subsystems.

Under the above notation, the system dynamics (2.1) become

$$\begin{aligned} \dot{z}_1 &= f_1(z_1, z_2, z_c, u) \\ \dot{z}_2 &= f_2(z_1, z_2, z_c, u) \\ \dot{z}_c &= f_c(z_1, z_2, z_c, u) \end{aligned} \tag{4.1}$$

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* (4.1)*:*

$$\begin{aligned} \dot{z}_1 &= f_1(z_1, z_c, u) \\ \dot{z}_2 &= f_2(z_2, z_c, u) \\ \dot{z}_c &= f_c(z_c, u) \end{aligned} \tag{4.2}$$

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 (4.77) 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. When the subsystems are coupled through u, we say that the subsystems have "shared control".

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

$$\begin{bmatrix} \dot{p}_x \\ \dot{p}_y \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \omega \end{bmatrix}, \qquad \omega \in \mathcal{U} \tag{4.3}$$

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 subsystem states x_i and the subsystem controls w_i are

$$\begin{aligned} \dot{x_1} &= \begin{bmatrix} \dot{z_1} \\ \dot{z_c} \end{bmatrix} = \begin{bmatrix} \dot{p_x} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ \omega \end{bmatrix} \\ \dot{x_2} &= \begin{bmatrix} \dot{z_2} \\ \dot{z_c} \end{bmatrix} = \begin{bmatrix} \dot{p_y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\sin\theta \\ \omega \end{bmatrix} \\ u &= \omega \end{aligned} \tag{4.4}$$

where the overlapping state is θ, and the subsystem controls and their shared component is the control u itself. The control partitions u_1, u_2 do not exist, since the state partitions z_1, z_2 do not depend on the control. For more examples of systems decomposed into SCSs, see (4.56), (4.57) and other numerical examples in this section.

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. 4.1. 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

$$\text{proj}_i(z) = x_i = (z_i, z_c) \tag{4.5}$$

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

$$\text{proj}^{-1}(x_i) = \{z \in \mathcal{Z} : (z_i, z_c) = x_i\} \tag{4.6}$$

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

$$\text{proj}^{-1}(\mathcal{S}_i) = \{z \in \mathcal{Z} : \exists x_i \in \mathcal{S}_i, (z_i, z_c) = x_i\} \tag{4.7}$$

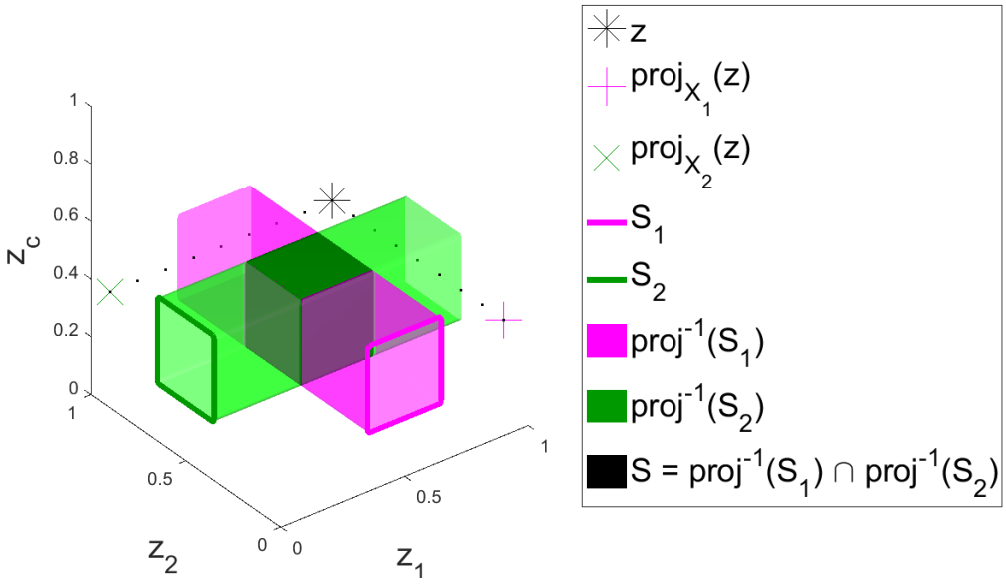


Figure 4.1: This figure shows the back-projection of sets in the z_1-z_c plane S_1 and the z_2-z_c plane (S_2) to the 3D space to form the intersection shown as the black cube (S). The figure also shows projection of a point z onto the lower-dimensional subspaces in the z_1-z_c and z_2-z_c planes.

Subsystem Trajectories: Since each subsystem in (4.77) is self-contained, we can denote the subsystem trajectories $\xi_i(s; x_i, t, u(\cdot))$. When needed, we will write the subsystem trajectories more explicitly in terms of the state partitions as $\xi_i(s; z_i, z_c, t, u(\cdot))$. The subsystem trajectories satisfy the subsystem dynamics and initial condition:

$$\begin{aligned} \frac{d}{ds}\xi_i(s; x_i, t, u(\cdot)) &= g_i(\xi_i(s; x_i, t, u(s)) \\ \xi_i(t; x_i, t, u(\cdot)) &= x_i \end{aligned} \tag{4.8}$$

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:

$$\text{proj}_i\big(\xi(s; z, t, u(\cdot))\big) = \xi_i(s; x_i, t, u(\cdot)) \tag{4.9}$$

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

4.1.1.2 Goals of This Section

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:

$$\mathcal{L} = \text{proj}^{-1}(\mathcal{L}_1) \cap \text{proj}^{-1}(\mathcal{L}_2) \tag{4.10}$$

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

$$\mathcal{L} = \text{proj}^{-1}(\mathcal{L}_1) \cup \text{proj}^{-1}(\mathcal{L}_2) \tag{4.11}$$

where the full target set is the union of the back-projections of subsystem target sets. Fig. 4.1 helps provide intuition for these concepts: applying (4.10) to $\mathcal{S}_1$ and $\mathcal{S}_2$ results in the black cube. Applying (4.11) 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. 4.3 and 4.5, 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 (4.77) and subsystem target sets $\mathcal{L}_i, i = 1, 2$, respectively:

$$\begin{aligned} \mathcal{R}_i(t) &= \{x_i : \exists u(\cdot), \xi_i(0; x_i, t, u(\cdot)) \in \mathcal{L}_i\} \\ \mathcal{A}_i(t) &= \{x_i : \forall u(\cdot), \xi_i(0; x_i, t, u(\cdot)) \in \mathcal{L}_i\} \end{aligned} \tag{4.12}$$

Subsystem BRTs are defined analogously:

$$\begin{aligned} \bar{\mathcal{R}}_i(t) &= \{x_i : \exists u(\cdot), \exists s \in [t, 0], \xi_i(s; x_i, t, u(\cdot)) \in \mathcal{L}_i\} \\ \bar{\mathcal{A}}_i(t) &= \{x_i : \forall u(\cdot), \exists s \in [t, 0], \xi_i(s; x_i, t, u(\cdot)) \in \mathcal{L}_i\} \end{aligned} \tag{4.13}$$

Given a system in the form of (4.77) with target set that can be represented by (4.10) or (4.11), 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 the following four cases is true:

$$\begin{aligned} &(4.10) \Rightarrow \mathcal{R}(t) = \text{proj}^{-1}(\mathcal{R}_1(t)) \cap \text{proj}^{-1}(\mathcal{R}_2(t)) \\ &(4.10) \Rightarrow \mathcal{A}(t) = \text{proj}^{-1}(\mathcal{A}_1(t)) \cap \text{proj}^{-1}(\mathcal{A}_2(t)) \\ &(4.11) \Rightarrow \mathcal{R}(t) = \text{proj}^{-1}(\mathcal{R}_1(t)) \cup \text{proj}^{-1}(\mathcal{R}_2(t)) \\ &(4.11) \Rightarrow \mathcal{A}(t) = \text{proj}^{-1}(\mathcal{A}_1(t)) \cup \text{proj}^{-1}(\mathcal{A}_2(t)) \end{aligned} \tag{4.14}$$

Table 4.1: Backward Reachable Set Decomposition

Section	**4.1.2**		**4.1.3**	
Shared Controls	**Yes**		**No**	
Shared Disturbance	**No**		**No**	
Target	**Intersection**	**Union**	**Intersection**	**Union**
Recover Max. BRS?	No	Yes, exact	Yes, exact	Yes, exact
Recover Min. BRS?	Yes, exact	No	Yes, exact	Yes, exact
Locations & Equation(s)	Thm 2, (4.16)	Thm 1, (4.15)	Prop 1, (4.31) Thm 2, (4.16)	Thm 1, (4.15) Prop 2, (4.32)

Section	**4.1.6.1**		**4.1.6.2**	
Shared Controls	**Yes**		**No**	
Shared Disturbance	**Yes**		**Yes**	
Target	**Intersection**	**Union**	**Intersection**	**Union**
Recover Max. BRS?	No	Yes, consrv	Yes, consrv	Yes, consrv
Recover Min. BRS?	Yes, consrv	No	Yes, consrv	Yes, consrv
Locations & Equation(s)	Cor 4, (4.67)	Cor 3, (4.65)	Cor 5, (4.68)	Cor 6, (4.70)

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 (4.77); in the case that there are no shared states this becomes a straightforward decoupled system.

Results related to BRSs are outlined for SCSs in Theorems 1 and 2. In the case that the subsystem controls do not share any components, Propositions 1 and 2 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 3 and 4, and Theorem 3.

- **Treatment of disturbances**. Finally, we investigate how the above theoretical results change in the presence of disturbances. In Section 4.1.6, we will show that slightly conservative BRSs and BRTs can still be obtained using our decomposition technique.

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

Table 4.2: BRT Results for Reconstruction from Tubes

Section	**4.1.4.1**			
Shared Controls	**Yes**		**No**	
Shared Disturbance	**No**		**No**	
Target	**Intersection**	**Union**	**Intersection**	**Union**
Recover Max. BRT?	No	Yes, exact	No	Yes, exact
Recover Min. BRT?	No	No	No	Yes, exact
Equation(s)	N/A	Prop 3, (4.37)	N/A	Prop 3, (4.37) Prop 3, (4.38)

Section	**4.1.6.3**			
Shared Controls	Yes		**No**	
Shared Disturbance	Yes		**Yes**	
Target	**Intersection**	**Union**	**Intersection**	Union
Recover Max. BRT?	No	Yes, conserv	No	Yes, conserv
Recover Min. BRT?	No	No	No	Yes, conserv
Equation(s)	N/A	Cor 7, (4.71)	N/A	Cor 7, (4.71) Cor 8, (4.72)

4.1.2 Self-Contained Subsystems

Suppose the full system (2.1) can be decomposed into SCSs given in (4.2). Then, the full-dimensional BRS can be reconstructed without incurring additional approximation errors from lower-dimensional BRSs in the situations stated in Theorem 1 and 2.

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* (2.1) *can be decomposed into the form of* (4.2)*, then*

$$\begin{aligned} &\mathcal{L} = proj^{-1}(\mathcal{L}_1) \cup proj^{-1}(\mathcal{L}_2) \\ &\Rightarrow \mathcal{R}(t) = proj^{-1}(\mathcal{R}_1(t)) \cup proj^{-1}(\mathcal{R}_2(t)) \end{aligned} \tag{4.15}$$

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 $\mathcal{L}$, it suffices to avoid the unsafe states in either subsystem, regardless of any coupling that exists between the subsystems. Theorem 2 formally states this intuitive result.*

Theorem 2 *Suppose that the full system in* (2.1) *can be decomposed into the form of* (4.2)*, then*

Table 4.3: BRT Results for Reconstruction from Sets

Section	**4.1.4.2**			
Shared Controls	**Yes**		**No**	
Shared Disturbance	**No**		**No**	
Target	**Intersection**	**Union**	**Intersection**	**Union**
Recover Max. BRT?	Yes, exact			
Recover Min. BRT?	Yes, exact*			
Equation(s)	Prop 4, (4.39) Thm 3, (4.41)			

Section	**4.1.6.3**			
Shared Controls	**Yes**		**No**	
Shared Disturbance	**Yes**		**Yes**	
Target	**Intersection**	**Union**	**Intersection**	**Union**
Recover Max. BRT?	Yes, conserv			
Recover Min. BRT?	Yes, exact*			
Equation(s)	Cor 9, (4.73) Thm 3, (4.41)			

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 (4.77); in the case that there are no shared states this becomes a straightforward decoupled system. We assume that exact sets are available to compute those BRTs that require the union of sets.
* the solution here can be found only if the minimum BRSs are non-empty for the entire time period.

$$\begin{aligned} &\mathcal{L} = \overline{proj}^{-1}(\mathcal{L}_1) \cap \overline{proj}^{-1}(\mathcal{L}_2) \\ &\Rightarrow \mathcal{A}(t) = \overline{proj}^{-1}(\mathcal{A}_1(t)) \cap \overline{proj}^{-1}(\mathcal{A}_2(t)) \end{aligned} \tag{4.16}$$

To prove the theorems, we need some intermediate results.

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

$$\bar{x}_i \in \mathcal{S}_i \Leftrightarrow \bar{z} \in \overline{proj}^{-1}(\mathcal{S}_i) \tag{4.17}$$

Proof of Lemma 1: 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 (4.7), 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 (4.7) 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 $\mathcal{S} = \overline{proj}^{-1}(\mathcal{S}_1) \cup \overline{proj}^{-1}(\mathcal{S}_2)$, then*

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

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

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

4.1.2.1 Proof of Theorem 1

We will prove the following equivalent statement:

$$\bar{z} \in \mathcal{R}(t) \Leftrightarrow \bar{z} \in \text{proj}^{-1}(\mathcal{R}_1(t)) \cup \text{proj}^{-1}(\mathcal{R}_2(t)) \tag{4.18}$$

Consider the relationship between the full system trajectory and subsystem trajectory in (4.9). Define $\bar{x}_i = \text{proj}_i(\bar{z})$ and $\xi_i(0; \bar{x}_i, t, u(\cdot)) = \text{proj}_i(\xi(0; \bar{z}, t, u(\cdot)))$.

We first prove the backward direction. By Corollary 1, (4.18) is equivalent to

$$\bar{x}_1 \in \mathcal{R}_1(t) \vee \bar{x}_2 \in \mathcal{R}_2(t) \tag{4.19}$$

WLOG, assume $\bar{x}_1 \in \mathcal{R}_1(t)$. By the subsystem BRS definition in (4.12), this is equivalent to

$$\exists u(\cdot), \xi_1(0; \bar{x}_1, t, u(\cdot)) \in \mathcal{L}_1 \tag{4.20}$$

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

For the forward direction, we begin with $\bar{z} \in \mathcal{R}(t)$, which by Definition 1 is equivalent to $\exists u(\cdot), \xi(0; \bar{z}, t, u(\cdot)) \in \mathcal{L}$. By Corollary 1, we then have

$$\exists u(\cdot), \xi_1(0; \bar{x}_1, t, u(\cdot)) \in \mathcal{L}_1 \vee \xi_2(0; \bar{x}_2, t, u(\cdot)) \in \mathcal{L}_2 \tag{4.21}$$

Finally, distributing "$\exists u(\cdot)$" gives (4.19). ■

4.1.2.2 Proof of Theorem 2

We will prove the following equivalent statement:

$$\bar{z} \notin \mathcal{A}(t) \Leftrightarrow \bar{z} \notin \text{proj}^{-1}(\mathcal{A}_1(t)) \cap \text{proj}^{-1}(\mathcal{A}_2(t)) \tag{4.22}$$

The above statement is equivalent to

$$\bar{z} \in \mathcal{A}^c(t) \Leftrightarrow \bar{z} \in \left[\text{proj}^{-1}(\mathcal{A}_1(t))\right]^c \cup \left[\text{proj}^{-1}(\mathcal{A}_2(t))\right]^c \tag{4.23}$$

By the Definition 2 (minimal BRS), we have that $\bar{z} \in \mathcal{A}^c(t)$ is equivalent to $\exists u(\cdot) \in \mathbb{U}, \xi(0; \bar{z}, t, u(\cdot)) \in \mathcal{L}^c$. 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 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 section of thesis. Table 4.4 summarizes the results from this subsection.

Table 4.4: BRS Results from Section 4.1.2

Shared Controls	**Yes**	
Shared Disturbance	**No**	
Target	**Intersection**	**Union**
Recover Max. BRS?	No	Yes, exact
Recover Min. BRS?	Yes, exact	No
Equation(s)	Thm 2, (4.16)	Thm 1, (4.15)

4.1.2.3 Numerical Example: The Dubins Car

The Dubins Car is a well-known system whose dynamics are given by (4.3). This system is only 3D, and its BRS can be tractably computed in the full-dimensional space, so we use it to compare the full formulation with our decomposition method. The Dubins Car dynamics can be decomposed according to (4.4). 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:

$$\mathcal{L} = \{(p_x, p_y, \theta) : |p_x|, |p_y| \leq 0.5\} \tag{4.24}$$

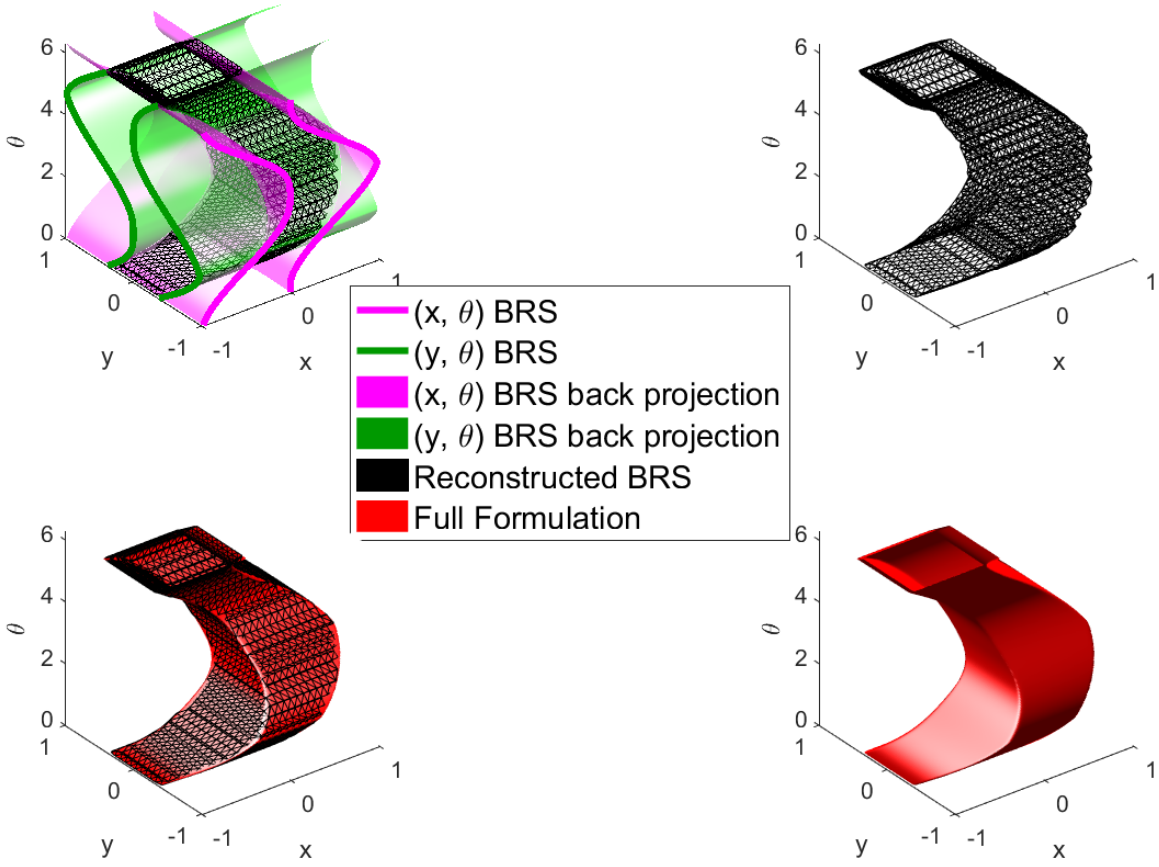


Figure 4.2: Comparison of the Dubins Car BRS $\mathcal{A}(t = -0.5)$ computed using the full formulation and via decomposition. Left top: BRSs in the lower-dimensional subspaces and how they are combined to form the full-dimensional BRS. Top right: BRS computed via decomposition. Bottom left: BRSs computed using both methods, superimposed, showing that they are indistinguishable. Bottom right: BRS computed using the full formulation.

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. 4.2 as the red surface which appears in the bottom subplots. To compute the BRS using our decomposition method, we write the unsafe set $\mathcal{L}$ as

$$\begin{aligned} \mathcal{L}_1 = \{(p_x, \theta) : |p_x| \leq 0.5\}, \mathcal{L}_2 = \{(p_y, \theta) : |p_y| \leq 0.5\} \\ \mathcal{L} = \text{proj}^{-1}(\mathcal{L}_1) \cap \text{proj}^{-1}(\mathcal{L}_2) \end{aligned} \tag{4.25}$$

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: $\mathcal{A}(t) =$

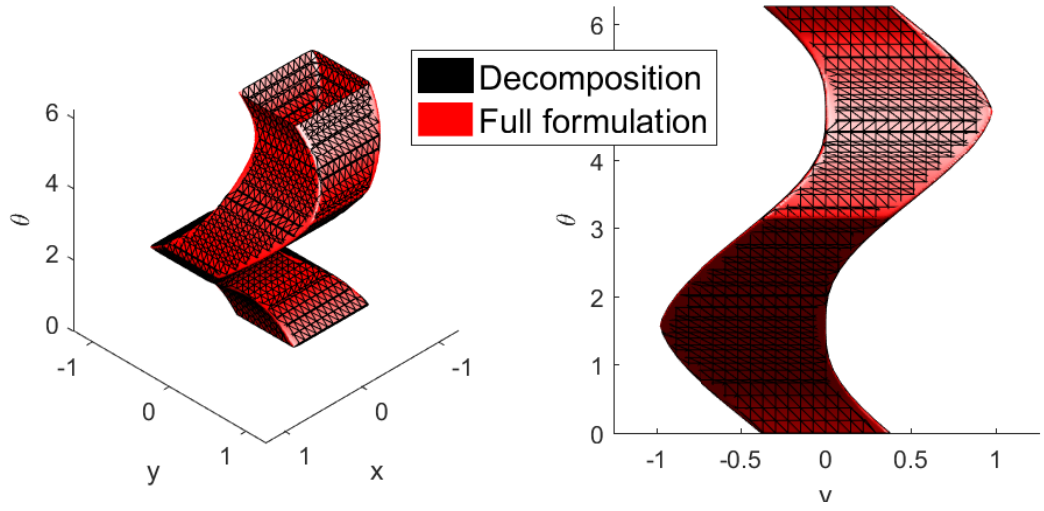


Figure 4.3: The Dubins Car BRS $\mathcal{A}(t = -0.5)$ computed using the full formulation and via decomposition, other view angles.

$\text{proj}^{-1}(\mathcal{A}_1(t)) \cap \text{proj}^{-1}(\mathcal{A}_2(t))$. The subsystem BRSs and their back-projections are shown in magenta and green in the top left subplot of Fig. 4.2. The reconstructed BRS is shown in the top left, top right, and bottom left subplots of Fig. 4.2 (black mesh).

In the bottom left subplot of Fig. 4.2, we superimpose the full-dimensional BRS computed using the two methods. We show the comparison of the computation results viewed from two different angles in Fig. 4.3. The results are indistinguishable.

Theorem 2 allows the computation to be performed in lower-dimensional subspaces, which is significantly faster. Another benefit of the decompositino 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. 4.4. The plot shows, in log-log scale, the computation time in seconds versus the number of grid points per dimension in the numerical computation. One can see that the direct computation of the BRS in 3D becomes very time-consuming as the number of grid points per dimension is increased, while the computation via decomposition hardly takes any time in comparison. Directly computing the BRS with 251 grid points per dimension in 3D took approximately 80 minutes, while computing the BRS via decomposition in 2D only took approximately 30 seconds! The computations were timed on a computer with an Intel Core i7-2600K processor and 16GB of random-access memory.

Fig. 4.5 illustrates Theorem 1. We chose the target set to be $\mathcal{L} = \{(p_x, p_y, \theta) : p_x \leq 0.5 \vee p_y \leq 0.5\}$, and computed the BRS $\mathcal{R}(t), t = -0.5$ via decomposition.

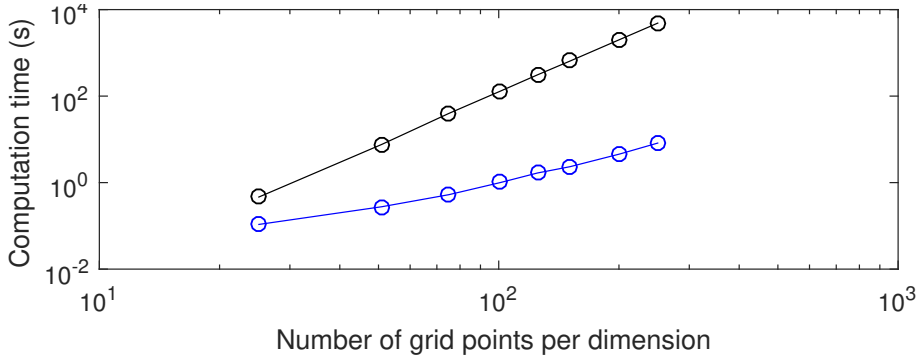


Figure 4.4: Computation times of the two methods in log scale for the Dubins Car. The time of the direct computation in 3D increases rapidly with the number of grid points per dimension. In contrast, computation times in 2D with decomposition are negligible in comparison.

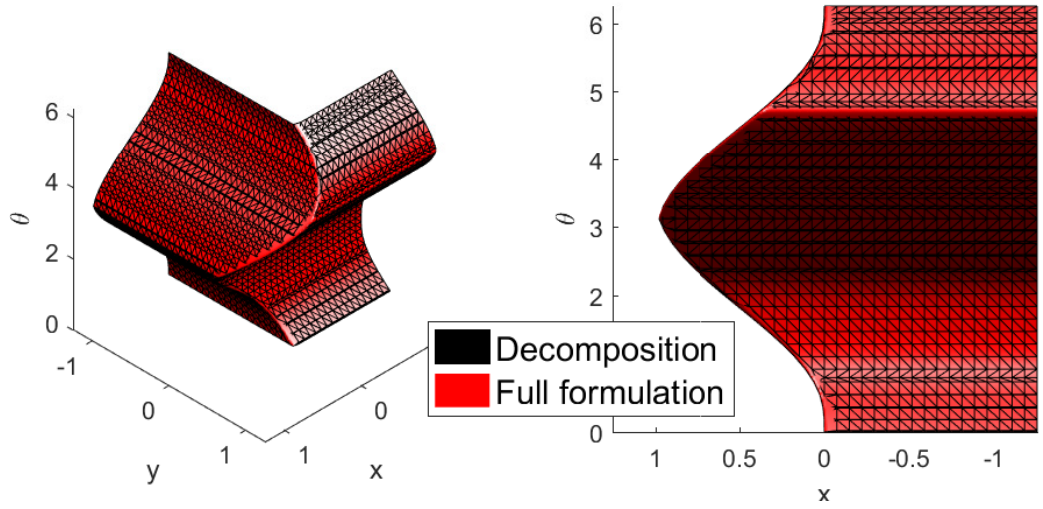


Figure 4.5: Comparison of the $\mathcal{R}(t)$ computed using our decomposition method and the full formulation. The computation results are indistinguishable. Note that the surface shows the boundary of the set; the set itself is on the "near" side of the left subplot, and the left side of the right subplot.

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 $\mathcal{L}_1 = \{(p_x, \theta) : p_x \leq 0.5\}, \mathcal{L}_2 = \{(p_y, \theta) : p_y \leq 0.5\}$.

4.1.3 SCSs with Decoupled Control

In this section, we consider a special case of (4.2) in which the subsystem controls do not have any shared components. The results from 4.1.2 still hold, and in addition we can state the results in Propositions 1 and 2. The special case of (4.2) is as follows:

$$\begin{aligned} \dot{z}_1 &= f_1(z_1, z_c, u_1) \\ \dot{z}_2 &= f_2(z_2, z_c, u_2) \\ \dot{z}_c &= f_c(z_c) \end{aligned} \tag{4.26}$$

where the subsystem controls do not have any shared components, so that we have $u = (u_1, u_2)$. Furthermore, we can define the trajectory of z_c as $\eta(s; z_c, t))$, which satisfies

$$\begin{aligned} \frac{d}{ds}\eta_c(s; z_c, t) &= f_c(\eta_c(s; z_c, t)) \\ \eta_c(t; z_c, t)) &= z_c \end{aligned} \tag{4.27}$$

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

$$\dot{z}_i = f_i(z_i, z_c, u_i) = f_i(z_i, u_i; \eta_c(s; z_c, t)) \tag{4.28}$$

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

$$\begin{aligned} &\frac{d}{ds}\eta_i(s; z_i, z_c, t, u_i(\cdot)) \\ &\qquad = f_i(\eta_i(s; z_i, z_c, t, u_i(\cdot)), u_i(s); \eta_c(s; z_c, t)) \\ &\qquad \eta_i(t; z_i, z_c, t, u_i(\cdot)) = z_i \end{aligned} \tag{4.29}$$

Therefore, the *subsystem trajectories* can be written as

$$\xi_i(s; x_i, t, u_i(\cdot)) = \big(\eta_i(s; z_i, z_c, t, u_i(\cdot)), \eta_c(s; z_c, t)\big) \tag{4.30}$$

Proposition 1 *Suppose that the full system in* (2.1) *can be decomposed into the form of* (4.26)*. Then,*

$$\begin{aligned} &\mathcal{L} = \mathit{proj}^{-1}(\mathcal{L}_1) \cap \mathit{proj}^{-1}(\mathcal{L}_2) \\ &\Rightarrow \mathcal{R}(t) = \mathit{proj}^{-1}(\mathcal{R}_1(t)) \cap \mathit{proj}^{-1}(\mathcal{R}_2(t)) \end{aligned} \tag{4.31}$$

Proposition 2 *Suppose that the full system in* (2.1) *can be decomposed into the form of* (4.26)*. Then,*

$$\begin{aligned} &\mathcal{L} = \overleftarrow{proj}^{-1}(\mathcal{L}_1) \cup \overleftarrow{proj}^{-1}(\mathcal{L}_2) \\ &\Rightarrow \mathcal{A}(t) = \overleftarrow{proj}^{-1}(\mathcal{A}_1(t)) \cup \overleftarrow{proj}^{-1}(\mathcal{A}_2(t)) \end{aligned} \tag{4.32}$$

Remark 3 *Systems with fully decoupled subsystems in the form of $x_1 = z_1, x_2 = z_2$ are a special case of* (4.26)*. A numerical example illustrating this case will be presented in the 10D Near-Hover Quadrotor example in Section 4.1.5.2.*

4.1.3.1 Proof of Proposition 1

We will prove the following equivalent statement:

$$\bar{z} \in \mathcal{R}(t) \Leftrightarrow \bar{z} \in \text{proj}^{-1}(\mathcal{R}_1(t)) \cap \text{proj}^{-1}(\mathcal{R}_2(t)) \tag{4.33}$$

By Definition 1 (maximal BRS), we have

$$\bar{z} \in \mathcal{R}(t) \Leftrightarrow \exists u(\cdot), \xi(0; \bar{z}, t, u(\cdot)) \in \mathcal{L} \tag{4.34}$$

Consider the relationship between the full system trajectory and subsystem trajectory in (4.9). Define

$$\begin{aligned} &\bar{x}_i = (\bar{z}_i, \bar{z}_c) = \text{proj}_i(\bar{z}), \text{ and} \\ &\xi_i(s; \bar{x}_i, t, u_i(\cdot)) = \text{proj}_i(\xi(0; \bar{z}, t, u(\cdot))) \end{aligned}$$

By (4.30) we can write

$$\big(\eta_i(s; \bar{z}_i, \bar{z}_c, t, u_i(\cdot)), \eta_c(s; \bar{z}_c, t)\big) = \text{proj}_i(\xi(0; \bar{z}, t, u(\cdot)))$$

Since $\mathcal{L} = \text{proj}^{-1}(\mathcal{L}_1) \cap \text{proj}^{-1}(\mathcal{L}_2)$, (4.34) is equivalent to

$$\begin{aligned} &\exists\, (u_1(\cdot), u_2(\cdot))\,, \\ &\quad \big(\eta_1(s; \bar{z}_1, \bar{z}_c, t, u_1(\cdot)), \eta_c(s; \bar{z}_c, t)\big) \in \mathcal{L}_1 \wedge \\ &\quad \big(\eta_2(s; \bar{z}_2, \bar{z}_c, t, u_2(\cdot)), \eta_c(s; \bar{z}_c, t)\big) \in \mathcal{L}_2 \\ &\qquad \text{(by Corollary 2)} \end{aligned} \tag{4.35}$$

$$\begin{aligned} \Leftrightarrow \quad &\exists u_1(\cdot), \big(\eta_1(s; \bar{z}_1, \bar{z}_c, t, u_1(\cdot)), \eta_c(s; \bar{z}_c, t)\big) \in \mathcal{L}_1 \wedge \\ &\exists u_2(\cdot), \big(\eta_2(s; \bar{z}_2, \bar{z}_c, t, u_2(\cdot)), \eta_c(s; \bar{z}_c, t)\big) \in \mathcal{L}_2 \\ &\text{(since subsystem controls do not share components)} \end{aligned} \tag{4.36}$$

$$\begin{aligned} \Leftrightarrow \quad &x_1 \in \mathcal{R}_1(t) \wedge x_2 \in \mathcal{R}_2(t) \\ &\text{(by definition of subsystem BRS in (4.12))} \end{aligned}$$

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

■

4.1.3.2 Proof of Proposition 2

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

Remark 4 *When the subsystem controls have shared components, the control chosen by each subsystem may not agree with the other. This is the intuition behind why the results of Propositions 1 and 2 only hold true when there are no shared components in the subsystem controls. Note that the theorems hold despite the state coupling between the subsystems.*

The results from this section are summarized in Table 4.5.

Table 4.5: BRS Results from Section 4.1.3

Shared Controls	**No**	
Shared Disturbance	**No**	
Target	**Intersection**	**Union**
Recover Max. BRS?	Yes, exact	Yes, exact
Recover Min. BRS?	Yes, exact	Yes, exact
Equation(s)	Prop 1, (4.31) Thm 2, (4.16)	Thm 1, (4.15) Prop 2, (4.32)

4.1.4 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 4.1.4.1, and then present a general method in which a BRT can be obtained via the union of BRSs in Section 4.1.4.2.

4.1.4.1 Full System BRTs From Subsystem BRTs

Intuitively, it may seem like the results related to BRSs outlined in Sections 4.1.2 and 4.1.3 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 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 3 *Suppose* (4.11) *holds, that is,*

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

Then, the full-dimensional BRT can be reconstructed from the lower-dimensional BRTs without incurring additional approximation errors. For systems with SCSs as in (4.2)*, we have*

$$\bar{\mathcal{R}}(t) = \widehat{proj}^{-1}(\bar{\mathcal{R}}_1(t)) \cup \widehat{proj}^{-1}(\bar{\mathcal{R}}_2(t)) \tag{4.37}$$

This can be 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 subsystem controls that do not share any components, we in addition have

$$\bar{\mathcal{A}}(t) = \widehat{proj}^{-1}(\bar{\mathcal{A}}_1(t)) \cup \widehat{proj}^{-1}(\bar{\mathcal{A}}_2(t)) \tag{4.38}$$

This can be proven by starting the proof of Proposition 2 from Definition 4 for the BRT $\bar{\mathcal{A}}(t)$ *instead of Definition 2 for the BRS* $\mathcal{A}(t)$*. Note that (4.38) does not necessarily hold for systems with shared controls.*

4.1.4.2 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 with BRTs, we lose information about the exact time that a trajectory enters a set. Instead, we provide a general method of obtaining the BRT that can *also* be used in the case of Section 4.1.4.1: We first compute the BRSs, and then take their union to obtain the BRT. For this case, we show that $\bar{\mathcal{R}}(t) = \bigcup_{s\in[t,0]} \mathcal{R}(s)$, and $\bar{\mathcal{A}}(t) = \bigcup_{s\in[t,0]} \mathcal{A}(s)$ when $\mathcal{A}(s) \neq \emptyset \; \forall s \in [t, 0]$. These results related to the indirect reconstruction of BRTs are given in Proposition 4 and Theorem 3

Proposition 4

$$\bigcup_{s\in[t,0]} \mathcal{R}(s) = \bar{\mathcal{R}}(t) \tag{4.39}$$

Theorem 3

$$\bigcup_{s\in[t,0]} \mathcal{A}(s) \subseteq \bar{\mathcal{A}}(t) \tag{4.40}$$

In addition, if $\forall s \in [t, 0], \mathcal{A}(s) \neq \emptyset$*, then*

$$\bigcup_{s\in[t,0]} \mathcal{A}(s) = \bar{\mathcal{A}}(t). \tag{4.41}$$

Propositions 4 and the first part of Theorem 3 are known [109], but we present them 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 4.1.2 and 4.1.3 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* (4.13)*:*

$$\bar{\mathcal{A}}_i(t) = \{x_i : \forall u(\cdot), \exists s \in [t, 0], \xi_i(s; x_i, t, u(\cdot)) \in \mathcal{L}_i\}$$
$$\bar{\mathcal{R}}_i(t) = \{x_i : \exists u(\cdot), \exists s \in [t, 0], \xi_i(s; x_i, t, u(\cdot)) \in \mathcal{L}_i\}$$

The set of "s" 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 4 and Theorem 3.*

4.1.4.3 Proof of Proposition 4

We start with Definition 1 (maximal BRS):

$$\mathcal{R}(t) = \{z : \exists u(\cdot) \in \mathbb{U}, \xi(0; z, t, u(\cdot)) \in \mathcal{L}\}$$

If some state z is in the union $\bigcup_{s\in[t,0]} \mathcal{R}(s)$, then there is some $s \in [t, 0]$ such that $z \in \mathcal{R}(s)$. Therefore, the union can be written as

$$\bigcup_{s\in[t,0]} \mathcal{R}(s) = \{z : \exists s \in [t, 0], \exists u(\cdot), \xi(0; z, s, u(\cdot)) \in \mathcal{L}\} \tag{4.42}$$

Suppose $z \in \bigcup_{s\in[t,0]} \mathcal{R}(s)$, then equivalently

$$\exists s \in [t, 0], \exists u(\cdot) \in \mathbb{U}, \xi(0; z, s, u(\cdot)) \in \mathcal{L} \tag{4.43}$$

Using (2.3), the time-invariance of the system, we can shift the trajectory time arguments by $t - s$ to get

$$\exists s \in [t, 0], \exists u(\cdot) \in \mathbb{U}, \xi(t - s; z, t, u(\cdot)) \in \mathcal{L} \tag{4.44}$$

Since $s \in [t, 0] \Leftrightarrow t - s \in [t, 0]$, we can equivalently write

$$\exists s \in [t, 0], \exists u(\cdot) \in \mathbb{U}, \xi(s; z, t, u(\cdot)) \in \mathcal{L} \tag{4.45}$$

We can swap the expressions $\exists s \in [t, 0]$ and $\exists u(\cdot) \in \mathbb{U}$ without changing meaning since both quantifiers are the same:

$$\exists u(\cdot) \in \mathbb{U}, \exists s \in [t, 0], \xi(s; z, t, u(\cdot)) \in \mathcal{L} \tag{4.46}$$

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

4.1.4.4 Proof of Theorem 3

We first establish $\bigcup_{s\in[t,0]} \mathcal{A}(s) \subseteq \bar{\mathcal{A}}(t)$. Consider Definition 2 (minimal BRS):

$$\mathcal{A}(t) = \{z : \forall u(\cdot) \in \mathbb{U}, \xi(0; z, t, u(\cdot)) \in \mathcal{L}\}$$

If some state z is in the union $\bigcup_{s\in[t,0]} \mathcal{A}(s)$, then $\exists s \in [t, 0]$ such that $z \in \mathcal{A}(s)$. Thus, the union can be written as

$$\bigcup_{s\in[t,0]} \mathcal{A}(s) = \{z : \exists s \in [t, 0], \forall u(\cdot), \xi(0; z, s, u(\cdot)) \in \mathcal{L}\} \tag{4.47}$$

Suppose $z \in \bigcup_{s\in[t,0]} \mathcal{A}(s)$, then

$$\exists s \in [t, 0], \forall u(\cdot) \in \mathbb{U}, \xi(0; z, s, u(\cdot)) \in \mathcal{L} \tag{4.48}$$

Using (2.3), the time-invariance of the system, we can shift the trajectory time arguments by $t - s$ to get

$$\exists s \in [t, 0], \forall u(\cdot) \in \mathbb{U}, \xi(t - s; z, t, u(\cdot)) \in \mathcal{L} \tag{4.49}$$

Since $s \in [t, 0] \Leftrightarrow t - s \in [t, 0]$, we can equivalently write

$$\exists s \in [t, 0], \forall u(\cdot) \in \mathbb{U}, \xi(s; z, t, u(\cdot)) \in \mathcal{L} \tag{4.50}$$

Let such an $s \in [t, 0]$ be denoted $\bar{s}$, then

$$\begin{aligned} &\forall u(\cdot) \in \mathbb{U}, \xi(\bar{s}; z, t, u(\cdot)) \in \mathcal{L} \\ &\Rightarrow \forall u(\cdot) \in \mathbb{U}, \exists s \in [t, 0], \xi(s; z, t, u(\cdot)) \in \mathcal{L} \end{aligned} \tag{4.51}$$

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

Next, given $\forall s \in [t, 0], \mathcal{A}(s) \neq \emptyset$, we show $\bigcup_{s\in[t,0]} \mathcal{A}(s) \supseteq \bar{\mathcal{A}}(t)$. Equivalently, we show

$$z \notin \bigcup_{s\in[t,0]} \mathcal{A}(s) \Rightarrow z \notin \bar{\mathcal{A}}(t) \tag{4.52}$$

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

$$\forall s \in [t,0], \forall u(\cdot) \in \mathbb{U}, \xi(s;\bar{z},t,u(\cdot)) \in \mathcal{A}(s) \tag{4.53}$$

since otherwise, we would have for some $\bar{s} \in [t,0]$,

$$\begin{aligned} &\exists u(\cdot) \in \mathbb{U}, \xi(\bar{s};\bar{z},t,u(\cdot)) \notin \mathcal{A}(\bar{s}) \\ &\Rightarrow \exists u(\cdot) \in \mathbb{U}, \xi(0;\xi(\bar{s};\bar{z},t,u(\cdot)),\bar{s},u(\cdot)) \notin \mathcal{L} \\ &\Leftrightarrow \exists u(\cdot) \in \mathbb{U}, \xi(0;\bar{z},t,u(\cdot)) \notin \mathcal{L} \end{aligned} \tag{4.54}$$

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

Given $z \notin \mathcal{A}(t)$, there exists some control $\bar{u}(\cdot)$ such that $\xi(0;z,t,\bar{u}(\cdot)) \notin \mathcal{L} = \mathcal{A}(0)$. Moreover, we must have $\forall s \in [t,0], \xi(s;z,t,\bar{u}(\cdot)) \notin \mathcal{L}$, since otherwise, we would have $\exists \hat{s}$ such that

$$\begin{aligned} &\xi(\hat{s};z,t,\bar{u}(\cdot)) \in \mathcal{L} = \mathcal{A}(0) \\ &\Rightarrow z = \xi(t;z,t,\bar{u}(\cdot)) \in \mathcal{A}(t-\hat{s}) \end{aligned} \tag{4.55}$$

which contradicts $z \notin \bigcup_{s\in(t,0)} \mathcal{A}(s)$.

Using time-invariance of the system dynamics, we have $\forall s \in [t,0], \xi(0;z,t-s,\bar{u}(\cdot)) \notin \mathcal{L}$, which is equivalent to $\forall s \in [t,0], \xi(0;z,s,\bar{u}(\cdot)) \notin \mathcal{L}$. Therefore, $\exists u(\cdot) \in \mathbb{U}, \forall s \in [t,0], \xi(0;z,s,\bar{u}(\cdot)) \notin \mathcal{L} \Leftrightarrow z \notin \bar{\mathcal{A}}(t)$. ■

Remark 6 *When $\exists s \in [t,0], \mathcal{A}(s) = \emptyset$, it is currently not known whether the union of the BRSs $\mathcal{A}(s)$ 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 equals to the BRT is an important future direction that we plan to investigate.*

Remark 7 *Note that Proposition 4 and Theorem 3 also hold for decoupled control.*

Table 4.6: BRT Results for Reconstruction from Tubes

Shared Controls	**Yes**		**No**	
Shared Dstb.	**No**		**No**	
Target	**Intersection**	**Union**	**Intersection**	**Union**
Recover Max. BRT?	No	Yes, exact	No	Yes, exact
Recover Min. BRT?	No	No	No	Yes, exact
Equation(s)	N/A, see Table 4.7	Prop 3, (4.37)	N/A, see Table 4.7	Prop 3, (4.37) Prop 3, (4.38)

Table 4.7: BRT Results for Reconstruction from Sets

Shared Controls	Yes		No	
Shared Disturbance	No		No	
Target	Intersection	Union	Intersection	Union
Recover Max. BRT?	Yes, exact			
Recover Min. BRT?	Yes, exact*			
Equation(s)	Prop 4, (4.39) Thm 3, (4.41)			

4.1.4.5 Numerical Results

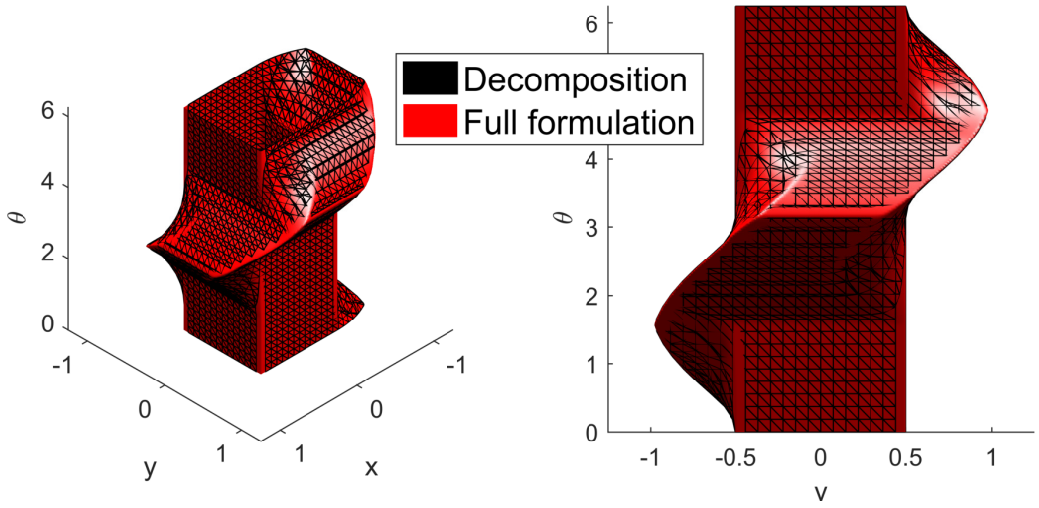


Figure 4.6: The BRT computed directly in 3D (red surface) and computed via decomposition in 2D (black mesh). Using our decomposition techinque, we first compute the BRSs $\mathcal{A}(s), s \in [-0.5, 0]$, and then obtained the BRT by taking their union.

We now revisit the Dubins Car, whose full system and subsystem dynamics are given in (4.3) and (4.4) respectively. Using the target set $\mathcal{L}$ given in (4.24) and writing $\mathcal{L}$ in the form of (4.25), we computed the BRT $\bar{\mathcal{A}}(t), t = -0.5$ by first computing $\mathcal{A}(s), s \in [-0.5, 0]$, and then taking their union.

Fig. 4.6 shows the BRT $\bar{\mathcal{A}}(t), t = -0.5$ computed directly in 3D and via decomposition. Since $\mathcal{A}(s) \neq \emptyset \ \forall s \in [-0.5, 0]$, the reconstruction does not incur any additional approximation errors.

4.1.5 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 in-

tractable to compute with previous methods to the best of our knowledge.

4.1.5.1 The 6D Acrobatic Quadrotor

In [69], 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

$$\begin{bmatrix} \dot{p}_x \\ \dot{v}_x \\ \dot{p}_y \\ \dot{v}_y \\ \dot{\phi} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} v_x \\ -\frac{1}{m} C_D^v v_x - \frac{T_1}{m} \sin\phi - \frac{T_2}{m} \sin\phi \\ v_y \\ -\frac{1}{m} \left(mg + C_D^v v_y\right) + \frac{T_1}{m} \cos\phi + \frac{T_2}{m} \cos\phi \\ \omega \\ -\frac{1}{I_{yy}} C_D^\phi \omega - \frac{l}{I_{yy}} T_1 + \frac{l}{I_{yy}} T_2 \end{bmatrix} \tag{4.56}$$

where x, y, and ϕ represent the quadrotor's horizontal, vertical, and rotational positions, respectively. Their derivatives represent the velocity with respect to each state. The control inputs T_1 and T_2 represent the thrust exerted on either end of the quadrotor, and the constant system parameters are m for mass, C_D^v for translational drag, C_D^ϕ for rotational drag, g for acceleration due to gravity, l for the length from the quadrotor's center to an edge, and I_{yy} for moment of inertia.

We decompose the system into the following subsystems:

$$x_1 = (p_x, v_x, \phi, \omega), \qquad x_2 = (p_y, v_y, \phi, \omega) \tag{4.57}$$

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:

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

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. 4.7 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. 4.7 are 3D slices in (v_x, v_y, ω) space at $p_x, p_y = 1.5, \phi = 1.5$ through different points in time. The sets grow darker as time propagates backward. The union of the BRSs is the BRT, shown as the gray surface.

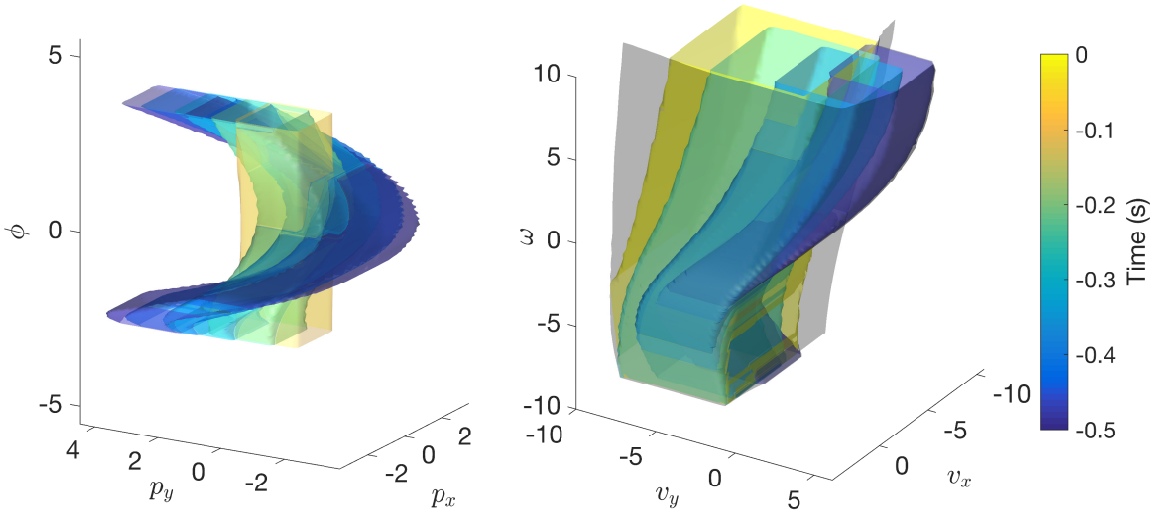


Figure 4.7: Left: 3D positional slices of the reconstructed 6D BRSs at $v_x = v_y = 1$, $\omega = 0$ at different points in time. The BRT cannot be seen in this image because it encompasses the entire union of BRSs. Right: 3D velocity slices of the reconstructed 6D BRSs at $x, y = 1.5$, $\phi = 1.5$ at different points in time. The BRT can be seen as the transparent gray surface that encompasses the sets.

4.1.5.2 The 10D Near-Hover Quadrotor

The 10D Near-Hover Quadrotor was used for experiments involving learning-based MPC [25]. Its dynamics are

$$\begin{bmatrix} \dot{p}_x \\ \dot{v}_x \\ \dot{\theta}_x \\ \dot{\omega}_x \\ \dot{p}_y \\ \dot{v}_y \\ \dot{\theta}_y \\ \dot{\omega}_y \\ \dot{p}_z \\ \dot{v}_z \end{bmatrix} = \begin{bmatrix} v_x + d_x \\ g \tan\theta_x \\ -d_1\theta_x + \omega_x \\ -d_0\theta_x + n_0 S_x \\ v_y + d_y \\ g \tan\theta_y \\ -d_1\theta_y + \omega_y \\ -d_0\theta_y + n_0 S_y \\ v_z + d_z \\ k_T T_z - g \end{bmatrix} \tag{4.58}$$

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:

$$x_1 = (p_x, v_x, \theta_x, \omega_x), x_2 = (p_y, v_y, \theta_y, \omega_y), x_3 = (p_z, v_z) \tag{4.59}$$

The target set is chosen to be

$$\begin{aligned} \mathcal{L} = \{(p_x, v_x, \theta_x, \omega_x, p_y, v_y, \theta_y, \omega_y, p_z, v_z) : \\ |p_x|, |p_y| \leq 1, |p_z| \leq 2.5\} \end{aligned} \tag{4.60}$$

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

$$\begin{aligned} \mathcal{L}_1 &= \{(p_x, v_x, \theta_x, \omega_x) : |p_x| \leq 1\} \\ \mathcal{L}_2 &= \{(p_y, v_y, \theta_y, \omega_y) : |p_y| \leq 1\} \\ \mathcal{L}_3 &= \{(p_z, v_z) : |p_z| \leq 2.5\} \end{aligned} \tag{4.61}$$

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 4.1.6.

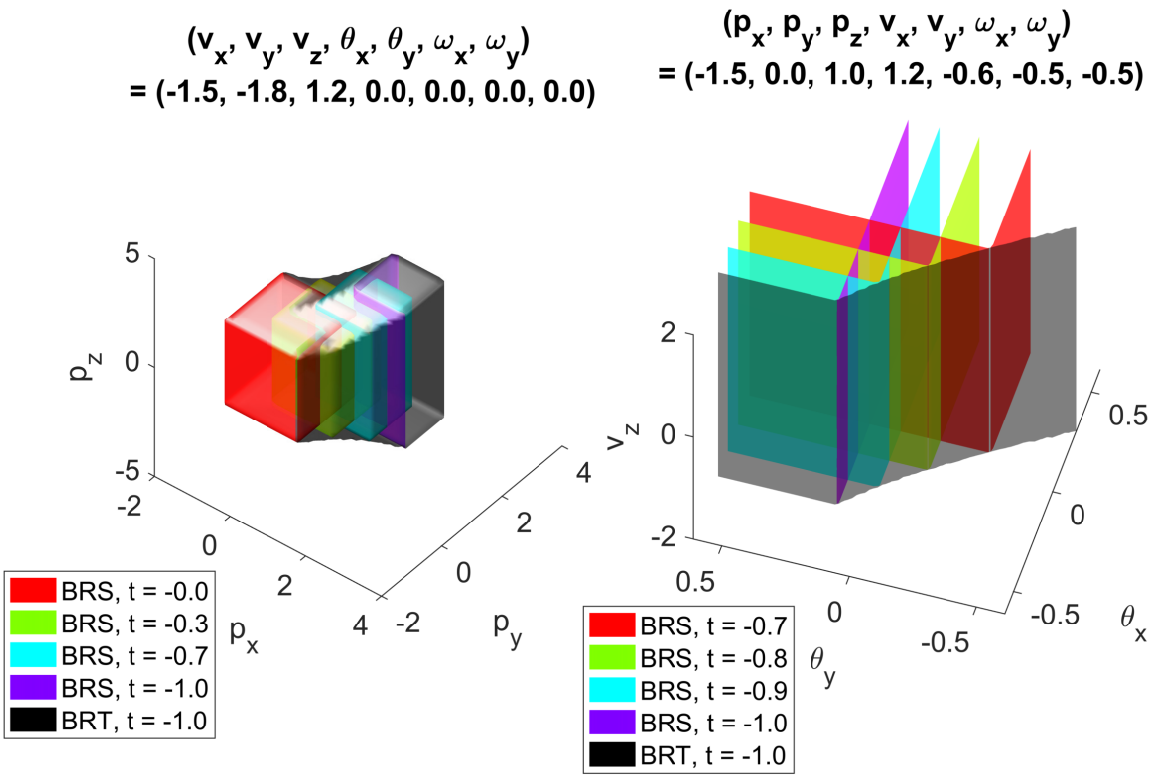


Figure 4.8: 3D slices of the 10D BRSs over time (colored surfaces) and BRT (black surface) for the Near-Hover Quadrotor. The slices are taken at the indicated 7D point.

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. 4.8, 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 4.

The right subplot of Fig. 4.8 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 (4.58) is needed for analyzing the angular behavior of the system.

4.1.6 Handling disturbances

Under the presence of disturbances, the full system dynamics changes from (2.1) to

$$\frac{dz}{ds} = \dot{z} = f(z, u, d), s \in [t, 0], u \in \mathcal{U}, d \in \mathcal{D} \tag{4.62}$$

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 [111], denoted $\Gamma(t)$. We denote the mapping from $u(\cdot)$ to $d(\cdot)$ as $\gamma[u](\cdot)$ as in [111]. The subsystems in (4.2) are now written as

$$\begin{aligned} \dot{z}_1 &= f_1(z_1, z_c, u, d) \\ \dot{z}_2 &= f_2(z_2, z_c, u, d) \\ \dot{z}_c &= f_c(z_c, u, d) \end{aligned} \tag{4.63}$$

In general, subsystem disturbances may have shared components. Whether this is the case is very important, as some of the results involving disturbances become stronger when the subsystem disturbances do not have shared components.

Trajectories of the system and subsystems are now denoted $\xi(s; z, t, u(\cdot), d(\cdot))$, $\xi_i(s; x_i, t, u(\cdot), d(\cdot))$, and satisfy conditions analogous to (2.2) and (4.8) respectively. We also need to incorporate the disturbance into the BRS and BRT definitions:

$$\begin{aligned} \bar{\mathcal{A}}(t) = \{z : \exists \gamma[u](\cdot), \forall u(\cdot), \exists s \in [t, 0], \\ \xi(s; z, t, u(\cdot), \gamma[u](\cdot)) \in \mathcal{L}\} \\ \bar{\mathcal{R}}(t) = \{z : \forall \gamma[u](\cdot), \exists u(\cdot), \exists s \in [t, 0], \\ \xi(s; z, t, \gamma[u](\cdot), d(\cdot)) \in \mathcal{L}\} \\ \mathcal{A}(t) = \{z : \exists \gamma[u](\cdot), \forall u(\cdot), \xi(0; z, t, u(\cdot), \gamma[u](\cdot)) \in \mathcal{L}\} \\ \mathcal{R}(t) = \{z : \forall \gamma[u](\cdot), \exists u(\cdot), \xi(0; z, t, u(\cdot), \gamma[u](\cdot)) \in \mathcal{L}\} \end{aligned} \tag{4.64}$$

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

4.1.6.1 Self-Contained Subsystems

Under the presence of disturbances, the results from Section 4.1.2 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* (2.1) *can be decomposed into the form of* (4.63)*, then*

$$\begin{aligned} &\mathcal{L} = proj^{-1}(\mathcal{L}_1) \cup proj^{-1}(\mathcal{L}_2) \\ &\Rightarrow \mathcal{R}(t) \supseteq proj^{-1}(\mathcal{R}_1(t)) \cup proj^{-1}(\mathcal{R}_2(t)) \end{aligned} \tag{4.65}$$

To solve this, in the proof of Theorem 1, (4.21) *becomes*

$$\begin{aligned} \forall \gamma[u](\cdot), \exists u(\cdot), \quad & \xi_1(0; \bar{x}_1, t, u(\cdot), d(\cdot)) \in \mathcal{L}_1 \vee \\ & \xi_2(0; \bar{x}_2, t, u(\cdot), d(\cdot)) \in \mathcal{L}_2 \end{aligned} \tag{4.66}$$

The expression "$\forall \gamma[u](\cdot), \exists u(\cdot)$" 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 z in the reconstructed BRS is guaranteed to be able to reach the target.

Corollary 4 *Suppose that the full system in* (2.1) *can be decomposed into the form of* (4.63)*, then*

$$\begin{aligned} & \mathcal{L} = proj^{-1}(\mathcal{L}_1) \cap proj^{-1}(\mathcal{L}_2) \\ & \Rightarrow \mathcal{A}(t) \subseteq proj^{-1}(\mathcal{A}_1(t)) \cap proj^{-1}(\mathcal{A}_2(t)) \end{aligned} \tag{4.67}$$

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 z outside of the reconstructed BRS is guaranteed to be able to avoid the target.

If the subsystem disturbances have no shared components, then (4.66) becomes

$$\begin{aligned} \forall \left(\gamma_1[u](\cdot), \gamma_2[u](\cdot)\right), \exists u(\cdot), \quad & \xi_1(0; \bar{x}_1, t, u(\cdot), \gamma_1[u](\cdot)) \in \mathcal{L}_1 \vee \\ & \xi_2(0; \bar{x}_2, t, u(\cdot), \gamma_2[u](\cdot)) \in \mathcal{L}_2 \end{aligned}$$

where $\gamma[u](\cdot)$ 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.

4.1.6.2 Subsystems with Decoupled Control

For systems with decoupled control, but shared disturbance in the subsystems, results from Section 4.1.6.1 still hold since the system dynamics structure is a special case of that in Section 4.1.6.1. In addition, results from Section 4.1.3 hold with some modifications. Propositions 1 and 2 need to be modified, and again the reconstructed BRS is now an approximation conservative in the right direction.

Corollary 5 *Suppose that the full system in* (2.1) *can be decomposed into the form of* (4.26)*, with the addition of shared disturbances. Then,*

$$\begin{aligned} & \mathcal{L} = proj^{-1}(\mathcal{L}_1) \cap proj^{-1}(\mathcal{L}_2) \\ & \Rightarrow \mathcal{R}(t) \supseteq proj^{-1}(\mathcal{R}_1(t)) \cap proj^{-1}(\mathcal{R}_2(t)) \end{aligned} \tag{4.68}$$

To prove this, we modify Proposition 1 by changing (4.35) *to*

$$\begin{aligned}&\forall\gamma[u](\cdot),\exists\,(u_1(\cdot),u_2(\cdot))\\&\quad\big(\eta_1(s;\bar{z}_1,\bar{z}_c,t,u_1(\cdot),\gamma[u](\cdot)),\eta_c(s;\bar{z}_c,t)\big)\in\mathcal{L}_1\wedge\\&\quad\big(\eta_2(s;\bar{z}_2,\bar{z}_c,t,u_2(\cdot),\gamma[u](\cdot)),\eta_c(s;\bar{z}_c,t)\big)\in\mathcal{L}_2\end{aligned}\tag{4.69}$$

The expression "$\forall\gamma[u](\cdot),\exists\,(u_1(\cdot),u_2(\cdot))$*" cannot be distributed to lead to a statement analogous to* (4.36)*. Hence, the forward direction of Proposition 1 does not hold, and conservativeness is introduced.*

By the same reasoning, the result of Proposition 2 changes to the following.

Corollary 6

$$\begin{aligned}&\mathcal{L}=proj^{-1}(\mathcal{L}_1)\cup proj^{-1}(\mathcal{L}_2)\\&\Rightarrow\mathcal{A}(t)\subseteq proj^{-1}(\mathcal{A}_1(t))\cup proj^{-1}(\mathcal{A}_2(t))\end{aligned}\tag{4.70}$$

In both cases, conservative approximations of the BRS can still be obtained.

Table 4.8: BRS Results from Subsections 4.1.6.1 & 4.1.6.2

Shared Controls	**Yes**		**No**	
Shared Dstb.	**Yes**		**Yes**	
Target	**Intersection**	**Union**	**Intersection**	**Union**
Recover Max. BRT?	No	Yes, consrv	Yes, consrv	Yes, consrv
Recover Min. BRT?	Yes, consrv	No	Yes, consrv	Yes, consrv
Equation(s)	Cor 4, (4.67)	Cor 3, (4.65)	Cor 5, (4.68)	Cor 6, (4.70)

4.1.6.3 Decomposition of Reachable Tubes

Under disturbances, the results from Section 4.1.4 carry over with modifications. For reconstruction from other BRTs, the arguments in Proposition 3 do not change. However, in the case where there are overlapping components in the subsystem disturbances, the reconstructed BRTs become conservative approximations:

Corollary 7 *Suppose our system has coupled control and disturbance as in* (4.63)*, then*

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

Corollary 8 *Suppose our system has subsystem controls that do not share any components, then*

$$\begin{aligned} &\mathcal{L} = \overline{proj}^{-1}(\mathcal{L}_1) \cup \overline{proj}^{-1}(\mathcal{L}_2) \\ &\Rightarrow \bar{\mathcal{A}}(t) \subseteq \overline{proj}^{-1}(\bar{\mathcal{A}}_1(t)) \cup \overline{proj}^{-1}(\bar{\mathcal{A}}_2(t)) \end{aligned} \tag{4.72}$$

For Proposition 4, 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* (4.63)*, then*

$$\begin{aligned} &\mathcal{L} = \overline{proj}^{-1}(\mathcal{L}_1) \cap \overline{proj}^{-1}(\mathcal{L}_2) \\ &\Rightarrow \bigcup_{s\in[t,0]} \mathcal{R}(s) \subseteq \bar{\mathcal{R}}(t) \end{aligned} \tag{4.73}$$

To show this, all arguments in the proof of Proposition 4 remain the same, except (4.42) *no longer implies* (4.46)*. Instead, the implication is unidirectional:*

$$\begin{aligned} &\exists s \in [t,0], \forall \gamma[u](\cdot), \exists u(\cdot), \xi(0; z, s, u(\cdot), \gamma[u](\cdot)) \in \mathcal{L} \\ &\Rightarrow \forall \gamma[u](\cdot), \exists u(\cdot), \exists s \in [t,0], \xi(s; z, t, u(\cdot), \gamma[u](\cdot)) \in \mathcal{L} \end{aligned} \tag{4.74}$$

This is due to the switching of the order of the expressions "$\exists s \in [t,0]$" and "$\gamma[u](\cdot)$". 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 4, all the arguments of Theorem 3 hold, since there no change of order of expressions involving existential and universal quantifiers.

Table 4.9: BRT Results for Reconstruction from Tubes

Shared Controls	**Yes**		**No**	
Shared Dstb.	**Yes**		**Yes**	
Target	**Intersection**	**Union**	**Intersection**	**Union**
Recover Max. BRT?	No	Yes, consrv	No	Yes, consrv
Recover Min. BRT?	No	No	No	Yes, consrv
Equation(s)	N/A	Cor 7, (4.71)	N/A	Cor 7, (4.71) Cor 8, (4.72)

Table 4.10: BRT Results for Reconstruction from Sets

Shared Controls	**Yes**		**No**	
Shared Disturbance	**Yes**		**Yes**	
Target	**Intersection**	**Union**	**Intersection**	**Union**
Recover Max. BRT?	Yes, conserv			
Recover Min. BRT?	Yes, exact*			
Equation(s)	Cor 9, (4.73) Thm 3, (4.41)			

* the solution here can be found only if the minimum BRSs are non-empty for the entire time period.

4.1.6.4 Dubins Car with Disturbances

Under disturbances, the Dubins Car dynamics are given by

$$\begin{bmatrix} \dot{p}_x \\ \dot{p}_y \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos\theta + d_x \\ v\sin\theta + d_y \\ \omega + d_\theta \end{bmatrix} \\ \omega \in \mathcal{U}, \quad (d_x, d_y, d_\theta) \in \mathcal{D} \tag{4.75}$$

with state $z = (p_x, p_y, \theta)$, 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 states x_i, controls w_i, and disturbances b_i are

$$\dot{x_1} = \begin{bmatrix} \dot{z_1} \\ \dot{z_c} \end{bmatrix} = \begin{bmatrix} \dot{p_x} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos\theta + d_x \\ \omega + d_\theta \end{bmatrix} \\ \dot{x_2} = \begin{bmatrix} \dot{z_2} \\ \dot{z_c} \end{bmatrix} = \begin{bmatrix} \dot{p_y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\sin\theta + d_y \\ \omega + d_\theta \end{bmatrix} \\ u_c = \omega = u \\ d_1 = d_x, d_2 = d_y, d_c = d_\theta \tag{4.76}$$

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.

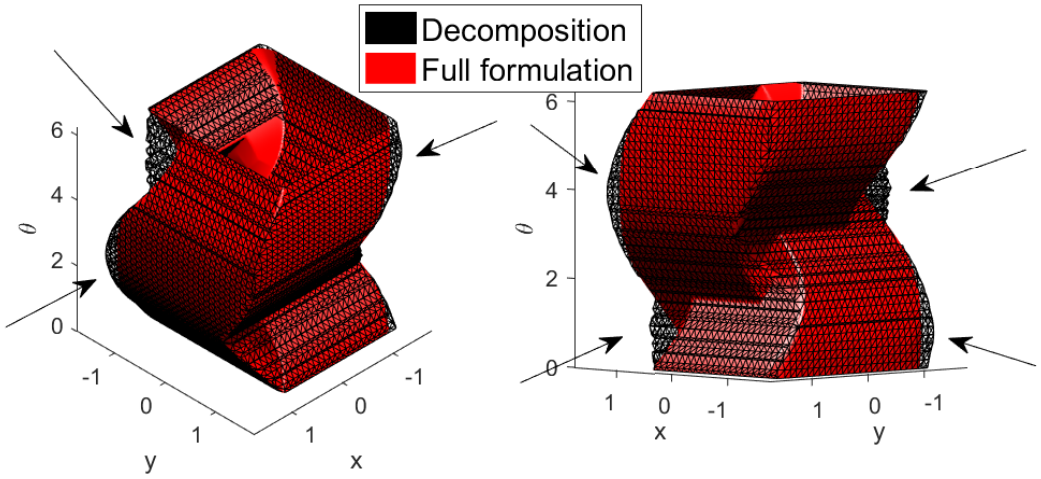


Figure 4.9: Minimal BRTs computed directly in 3D and via decomposition in 2D for the Dubins Car under disturbances with shared components. The reconstructed BRT is an over-approximation of the true BRT. The over-approximated regions of the reconstruction are indicated by the arrows.

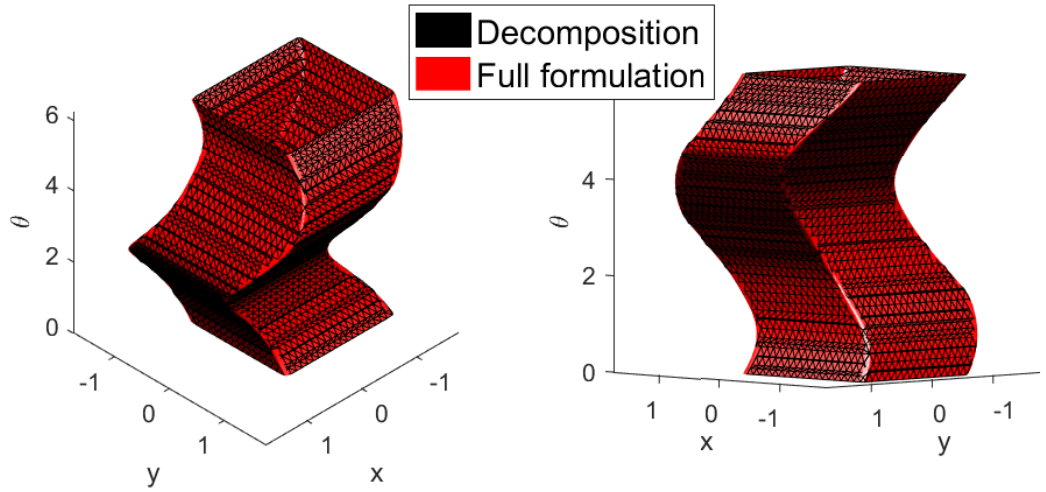


Figure 4.10: Minimal BRTs computed directly in 3D and via decomposition in 2D for the Dubins Car under disturbances *without* shared components. In this case, the BRT computed using decomposition matches the true BRT.

Fig. 4.9 compares the BRT $\bar{\mathcal{A}}(t), t = -0.5$ computed directly from the target set in (4.24), and using our decomposition technique from the subsystem target sets in (4.25). For this computation, we chose $\bar{d}_x, \bar{d}_y = 1, \bar{d}_\theta = 5$.

Since there is a shared component in the disturbances, the BRT computed using our decomposition technique becomes an over-approximation of the true BRT. One

can see the over-approximation by noting that the black set is not flush against the red set, as marked by the arrows in Fig. 4.9.

Fig. 4.10 shows the same computation with $\bar{d}_\theta = 0$, so that subsystem disturbances effectively have no shared components. In this case, one can see that the BRTs computed directly in 3D and via decomposition in 2D are the same.

4.1.7 Conclusions

In this section, 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 more 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.

4.2 Approximate Decomposition via State Decoupling Disturbances

This section is an adaptation of the paper in [31].

In this section, 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 [115] and [37], which exploit system structure. This method can also be combined with previous work such as [98, 107, 114]. 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].

4.2.1 Problem Formulation

Suppose that the state can be written as $z = (x, y)$ such that the control u and disturbance d can be written as $u = (u_x, u_y), d = (d_x, d_y)$, and such a decomposition

of the control leads to the following form of system dynamics:

$$\begin{aligned} \dot{x} = g(x, y, u_x, d_x) \quad \dot{y} = h(x, y, u_y, d_y) \\ u_x \in \mathcal{U}_x, u_y \in \mathcal{U}_y, \quad d_x \in \mathcal{D}_x, d_y \in \mathcal{D}_y, \quad t \in [-T, 0] \end{aligned} \tag{4.77}$$

where $x \in \mathbb{R}^{n_x}, y \in \mathbb{R}^{n_y}, n_x + n_y = n$, and g, h are components of the system dynamics that involve $(u_x, d_x), (u_y, d_y)$, 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, u_y and disturbance components d_x, d_y have independent control sets $\mathcal{U}_x, \mathcal{U}_y$ and disturbance sets $\mathcal{D}_x, \mathcal{D}_y$, respectively; note that we can also write $\mathcal{U} = \mathcal{U}_x \times \mathcal{U}_y$, and $\mathcal{D} = \mathcal{D}_x \times \mathcal{D}_y$. 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 (4.77), we would like to compute the BRS 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 BRS is defined as[1]

$$\begin{aligned} \mathcal{V}(T) = \{ & z_0 \in \mathbb{R}^n : \exists u(\cdot) \in \mathbb{U}, \forall \gamma \in \Gamma, \\ & z(\cdot) \text{ satisfies (4.77)}, z(-T) = z_0 \Rightarrow z(0) \in \mathcal{L} \} \end{aligned} \tag{4.78}$$

Standard HJ formulations exist for computing the BRS in the full dimensionality n [16, 23, 64, 111]. 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 [37, 115]. The goal of this section 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, 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 $\max(n_x, n_y)$ 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.

4.2.2 Decoupling via Virtual Disturbance

In the case where the dynamics are in the form of (4.77), one can treat y as a disturbance in the function g, and x as a disturbance in the function h, the system

[1] Similar definitions of BRSs and their relationships can be found in, for example, [109]

would become decoupled. Mathematically, (4.77) becomes the following:

$$\begin{array}{lll} \dot{x} = \hat{g}(x, y_d, u_x, d_x) & \dot{y} = \hat{h}(y, x_d, u_y, d_y) & \\ x \in \mathcal{X}, y_d \in \mathcal{Y} & x_d \in \mathcal{X}, y \in \mathcal{Y} & t \in [-T, 0] \\ u_x \in \mathcal{U}_x, d_x \in \mathcal{D}_x & u_y \in \mathcal{U}_y, d_y \in \mathcal{D}_y & \end{array} \tag{4.79}$$

where $\mathcal{X} \times \mathcal{Y}$ 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 (4.77), and produced approximate dynamics (4.79) that are decoupled, allowing us to do the computation in the space of each decoupled component.

Compared to the original system dynamics given in (4.77), the uncoupled dynamics given in (4.79) experiences a larger disturbance, since the y dependence of the function g and the x dependence on the function h are treated as disturbances. With the definition of BRS in (4.78), the approximate BRS computed using the dynamics (4.79) is an under-approximation of the true BRS. We formalize this in the proposition below.

Proposition 5 *Let $\mathcal{V}_x(T), \mathcal{V}_y(T)$ be the BRSs of the subsystem* (4.79) *from the target sets $\mathcal{L}_x, \mathcal{L}_y$, and let $\mathcal{V}(T)$ be the BRS of the system* (4.77) *from the target set $\mathcal{L}$. Then*[2]*, $\mathcal{L} = \mathcal{L}_x \cap \mathcal{L}_y \Rightarrow \mathcal{V}_x(T) \cap \mathcal{V}_y(T) \subseteq \mathcal{V}(T)$.*

Proof: It suffices to show that given any state $(x_0, y_0) = (x(-T), y(-T))$ such that x_0, y_0 are in the BRSs $\mathcal{V}_x(T), \mathcal{V}_y(T)$ for the system in (4.79), respectively, then (x_0, y_0) is in the BRS $\mathcal{V}(T)$ for the system in (4.77).

For convenience, we will use $x(\cdot) \in \mathbb{X}$ to denote $x(s) \in \mathcal{X}\ \forall s \in [-T, 0]$, with $y(\cdot) \in \mathbb{Y}$ having the analogous meaning. Applying the definition of BRS in (4.78) to the subsystems in (4.79), at the state $z_0 = (x_0, y_0)$ we have

1. $\exists u_x \in \mathbb{U}_x, \forall d_x \in \mathbb{D}_x, \forall y_d \in \mathbb{Y}, x(\cdot)$ satisfies (4.79), $x(0) \in \mathcal{L}_x$
2. $\exists u_y \in \mathbb{U}_y, \forall d_y \in \mathbb{D}_y, \forall x_d \in \mathbb{X}, y(\cdot)$ satisfies (4.79), $y(0) \in \mathcal{L}_y$

The above two conditions together imply

$$\begin{array}{r} \exists (u_x, u_y) \in \mathbb{U}_x \times \mathbb{U}_y, \forall (d_x, d_y) \in \mathbb{D}_x \times \mathbb{D}_y, \\ \forall (x_d, y_d) \in \mathbb{X} \times \mathbb{Y}, (x(\cdot), y(\cdot)) \text{ satisfies (4.79)}, \\ (x(0), y(0)) \in \mathcal{L} \end{array} \tag{4.80}$$

[2]Strictly speaking, $\mathcal{L}_x, \mathcal{L}_y, \mathcal{V}_x(T), \mathcal{V}_y(T)$ 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.

In particular, since $x(\cdot) \in \mathbb{X}, y(\cdot) \in \mathbb{Y}$, the above is true also when $x_d = x(\cdot), y_d = y(\cdot)$, so

$$\begin{aligned} \exists (u_x, u_y) \in \mathbb{U}_x \times \mathbb{U}_y, \forall (d_x, d_y) \in \mathbb{D}_x \times \mathbb{D}_y, \\ (x_d, y_d) = (x(\cdot), y(\cdot)), (x(\cdot), y(\cdot)) \text{ satisfies (4.79)}, \\ (x(0), y(0)) \in \mathcal{L} \end{aligned} \tag{4.81}$$

But if $x_d = x(\cdot), y_d = y(\cdot)$, then (4.79) becomes (4.77), thus

$$\begin{aligned} \exists (u_x, u_y) \in \mathbb{U}_x \times \mathbb{U}_y, \forall (d_x, d_y) \in \mathbb{D}_x \times \mathbb{D}_y, \\ (x(\cdot), y(\cdot)) \text{ satisfies (4.77)}, (x(0), y(0)) \in \mathcal{L}. \end{aligned} \tag{4.82}$$

■

By treating the coupled states as disturbance, the computation complexity reduces from $O(k^n T)$ for the full formulation to $O(k^{\max\{n_x, n_y\}} T)$ for the decoupled approximate system (4.79).

4.2.3 Disturbance Splitting

By treating coupling variables y and x 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 $\mathcal{X}$ and $\mathcal{Y}$ are large, the degree of conservatism can be reduced by splitting the disturbance x_d 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, $x_d \in \mathcal{X}$ can be split as follows:

$$\begin{aligned} x_d^i \in \mathcal{X}_i, i = 1, 2, \ldots, M \\ \text{where } \bigcup_{i=1}^{M} \mathcal{X}_i = \mathcal{X} \end{aligned} \tag{4.83}$$

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

$$\begin{aligned} \dot{x} &= g(x, y_d, u_x, d_x) & \dot{y} &= h(y, x_d^i, u_y, d_y) \\ u_x &\in \mathcal{U}_x, d_x \in \mathcal{D}_x & u_y &\in \mathcal{U}_y, d_y \in \mathcal{D}_y \\ y_d &\in \mathcal{Y} & x_d^i &\in \mathcal{X}^i \\ t &\in [-T, 0] & i &= 1, 2, \ldots, M_x \end{aligned} \tag{4.84}$$

from which a BRS can be computed in $\mathcal{X}^i \times \mathcal{Y}$. Since $\mathcal{X}^i \subseteq \mathcal{X}$, the uncoupling disturbance is reduced whenever the disturbance x_d is split. In addition, the uncoupling disturbance y_d can also be split into M_y, for a total of $M = M_x M_y$ 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. 4.12.

4.2.3.1 Examples of Decoupling System Dynamics

Our proposed method applies to any system of the form (4.77), as we will demonstrate with the example in Section 4.2.4. 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,

$$\dot{z} = \begin{bmatrix} \lambda & 1 & 0 & 0 \\ 0 & \lambda & 1 & 0 \\ 0 & 0 & \lambda & 1 \\ 0 & 0 & 0 & \lambda \end{bmatrix} z + Bu \tag{4.85}$$

If z_3 is treated as a disturbance in the equation for $\dot{z}_2$, we would obtain the decoupled components (z_1, z_2) and (z_3, z_4).

Lateral Quadrotor Dynamics near Hover [25]:

$$\dot{z} = \begin{bmatrix} z_2 \\ g \tan z_3 \\ z_4 \\ -d_0 z_3 - d_1 z_4 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ n_0 \end{bmatrix} u \tag{4.86}$$

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 (z_1, z_2) and (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 [25] 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 (4.86) into two 2D systems, each decoupled component is in the form $\dot{y} = g(y, u), \dot{x} = b(y)$. This is exactly the form of the dynamics in [115], 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.

4.2.4 Numerical Results

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

$$\begin{aligned} \dot{p}_x &= v\cos\psi & \dot{p}_y &= v\sin\psi \\ \dot{\psi} &= \omega & \dot{v} &= a \\ \underline{\omega} \le \omega &\le \bar{\omega} & \underline{a} \le a &\le \bar{a} \end{aligned} \tag{4.87}$$

where (p_x, p_y) represent the plane's position in the x and y directions, ψ represents the plane's heading, and v represents the plane's longitudinal velocity. The plane has a limited turn rate ω and a limited longitudinal acceleration a as its control variables. 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, ψ) and (p_y, v) as the decoupled components:

$$\begin{aligned} \dot{p}_x &= d_v \sin(\psi) & \dot{p}_y &= v\sin(d_\psi) \\ \dot{\psi} &= \omega & \dot{v} &= a \\ \underline{\omega} \le \omega &\le \bar{\omega} & \underline{a} \le a &\le \bar{a} \\ \underline{d_v} \le d_v &\le \bar{d_v} & \underline{d_\psi} \le d_\psi &\le \bar{d_\psi} \end{aligned} \tag{4.88}$$

We define the target set as a square of length 4 m centered at $(p_x, p_y) = (0, 0)$, described by $\mathcal{L} = \{(p_x, p_y, \psi, v) : |p_x|, |p_y| \le 2\}$. 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 $l(z)$ such that $l(z) \le 0 \Leftrightarrow x \in \mathcal{L}$. To analyze our newly decoupled system we must likewise decouple the target set by letting $\mathcal{L}_i, i = 1, 2$ be

$$\begin{aligned} \mathcal{L}_1 &= \{(p_x, \psi) : |p_x| \le 2\} \\ \mathcal{L}_2 &= \{(p_y, v) : |p_y| \le 2\} \end{aligned} \tag{4.89}$$

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

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

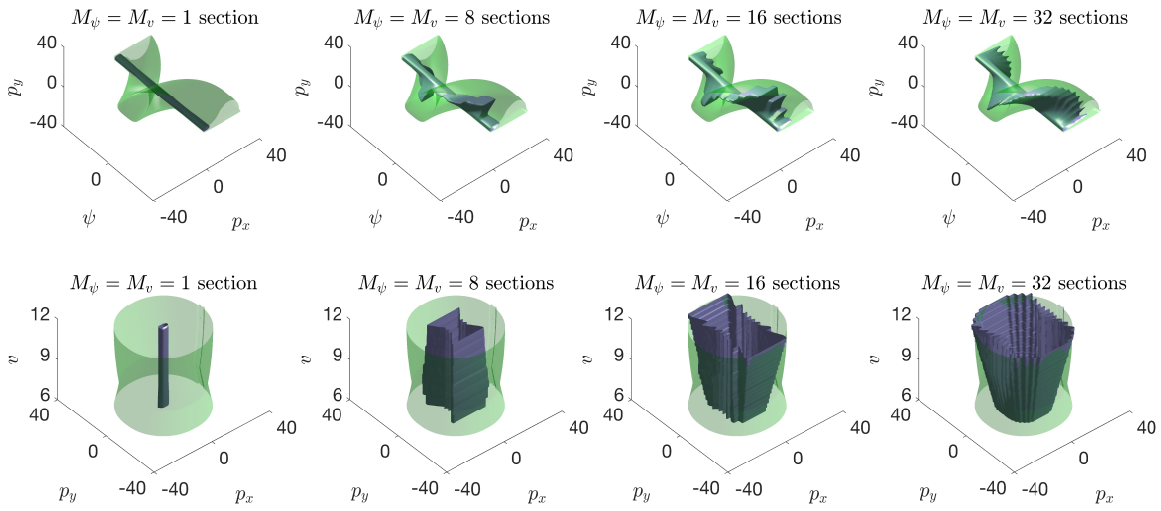


Figure 4.11: 3D slices of the BRSs across a range of M_v and M_ψ. The full formulation sets are in green, and the decoupled approximation sets are in gray. The top row shows 3D projections through all values of v. The top left plot shows this projection for $M_\psi = M_v = 1$ with M_ψ, M_v increasing 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.

4.2.4.1 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 despite worst possible disturbances. To analyze the BRS we vary the degree of conservatism using the disturbance splitting method described in section 4.2.3. After applying splitting, we arrive at the following piece-wise system:

$$\begin{aligned} \dot{p}_x &= d_v^i \sin(\psi) & \dot{p}_y &= v \sin(d_\psi^j) \\ \dot{\psi} &= \omega & \dot{v} &= a \\ \underline{\omega} \leq \omega &\leq \bar{\omega} & \underline{a} \leq a &\leq \bar{a} \\ \underline{d}_v^i \leq d_v^i &\leq \bar{d}_v^i & \underline{d}_\psi^j \leq d_\psi^j &\leq \bar{d}_\psi^j \\ i &= 1, 2, \ldots, M_v & j &= 1, 2, \ldots, M_\psi \end{aligned} \tag{4.90}$$

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. 4.11. 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 (4.78), 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

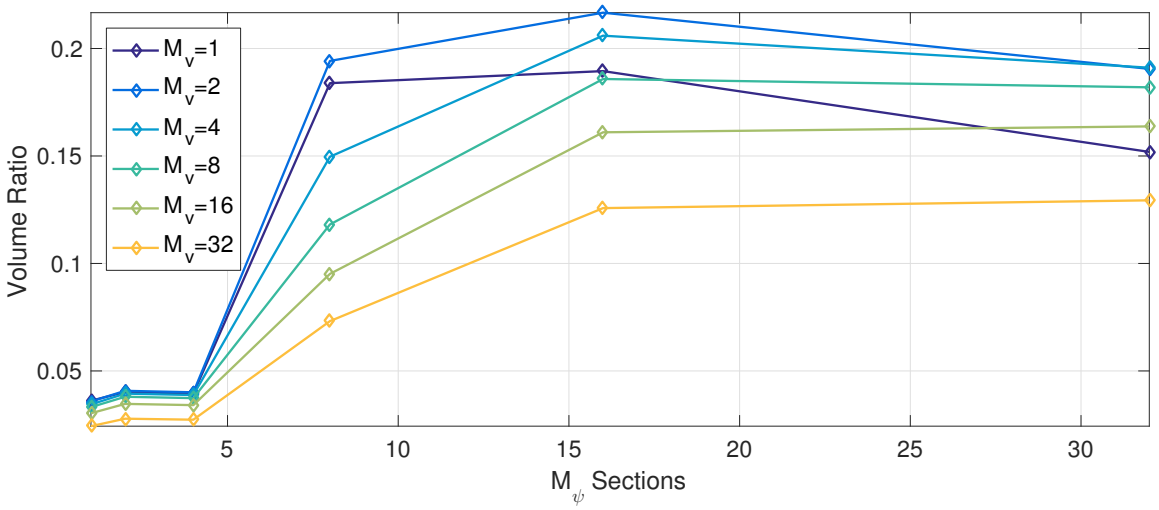


Figure 4.12: The vertical axis represents the ratio of the reconstructed BRS volume over the full formulation volume. The graph shows how this ratio changes as a function of number of disturbance sections. The highest volume ratio (and therefore least conservative BRS) was for $M_\psi = 16, M_v = 2$.

to right, each column of plots shows the decoupled approximations with an increasing number of split sections M_ψ and M_v.

4.2.4.2 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. 4.12 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 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 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

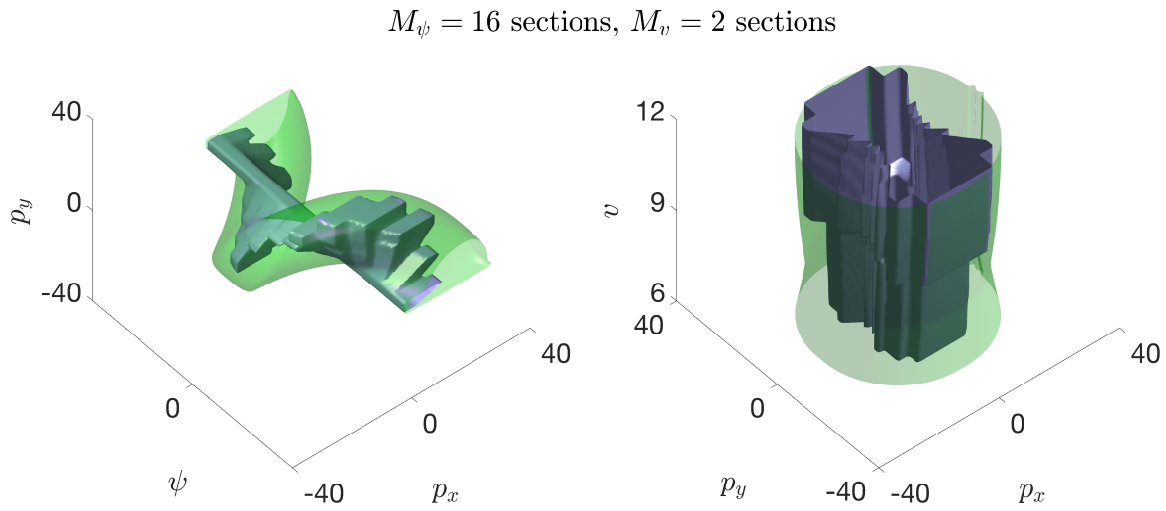


Figure 4.13: 3D slices of the BRS for $M_\psi = 16, M_v = 2$. This decoupled approximation provides the largest and least conservative under-approximation.

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. 4.13.

4.2.4.3 Computation Time Performance

In Fig. 4.14 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 4.2.4.2. 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 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.

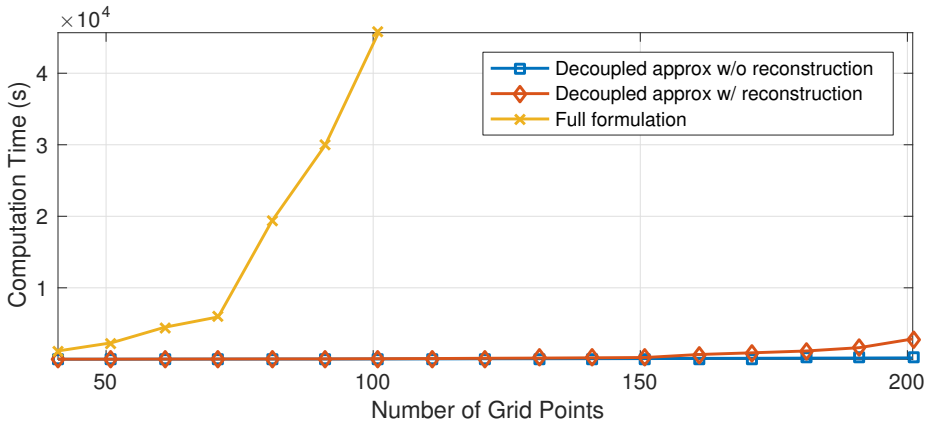


Figure 4.14: Computation time as a function of the number of grid points in each dimension. The full formulation (yellow curve) is orders of magnitude slower than the decoupled approximation. The decoupled approximation with reconstruction (red curve) takes a bit more time (and significantly more memory) than without reconstruction (blue curve).

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. 4.15 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.

4.2.5 Conclusions

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.

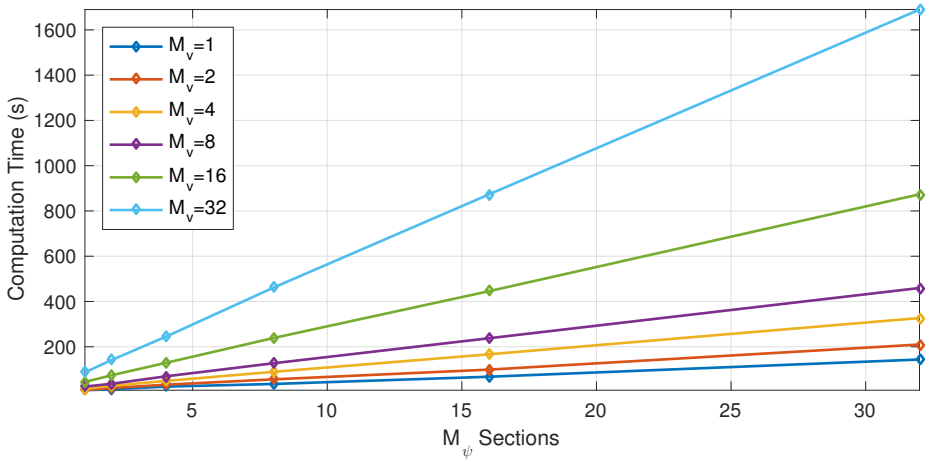


Figure 4.15: Computation time to compute decoupled approximation sets without reconstruction as a function of M_v and M_ψ. As the number of sections increases, the computation time increases approximately linearly.

Chapter 5

Frontiers in HJ Reachability Verification

In this chapter, we discuss two potential directions of HJ reachability analysis in detail. In Section 5.1, an initial idea of how a machine learning technique can be brought into the context of optimal control, and how formal guarantees can be maintained despite the use of a neural network. In Section 5.2, we present FaSTrack, a method for combining the formal guarantees of differential games and the real-time planning capabilities of robotic path planners such as rapidly-exploring random trees (RRT) to obtain the best of both worlds. Additional future directions will be discussed briefly in Chapter 6.

5.1 Using Neural Networks to Compute Approximate and Guaranteed Feasible HJB PDE Solutions

This section is an adaptation of the paper in [82].

In recent years, rapid progress in robotics and artificial intelligence has accelerated the need for efficient path-planning algorithms in high-dimensional spaces. In particular, there has been vast interest in the development of autonomous cars and unmanned aerial vehicles (UAVs) for civilian purposes [**?**, 13, 19, 61, 83, 123]. As such systems grow in complexity, development of algorithms that can tractably control them in high-dimensional state spaces are becoming necessary.

Many path planning problems can be cast as optimal control problems with initial and final state constraints. Dynamic programming-based methods for optimal control recursively compute controls using the Hamilton-Jacobi-Bellman (HJB) partial differential equation (PDE). Such methods suffer from space and time complexities that scale exponentially with the system dimension. Dynamic programming

is also the backbone for Hamilton-Jacobi (HJ) reachability analysis, which solves a specific type of optimal control problem and is a theoretically important and practically powerful tool for analyzing a large range of systems. It has been extensively studied in [16, 23, 60, 64, 105, 111, 140, 142], and successfully applied to many low-dimensional real world systems [35, 54, 111]. We use the reachability framework to validate our method.

To alleviate the curse of dimensionality, many proposed dynamic programming-based methods heavily restrict the system at hand [48, 56, 104]. Other less restrictive methods use projections, approximate dynamic programming, and approximate systems decoupling [31,107,114], each limited in flexibility, scalability, and degree of conservatism. There are also several approaches towards scalable verification [**?**, **?**, **?**, 59]; however, to the best of our knowledge, these methods do not extend easily to control synthesis. Direct and indirect methods for optimal control, such as nonlinear model predictive control [72], the calculus of variations [68], and shooting methods [12], avoid dynamic programming altogether but bring about other issues such as nonlinearity, instability, and sensitivity to initial conditions.

One primary drawback of dynamic programming-based approaches is the need to compute a value function over a large portion of the state space. This is computationally wasteful since the value function, from which the optimal controller is derived, is only needed along a trajectory from the system's initial state to the target set. A more efficient approach would be to only compute the value function local to the trajectory from the initial state to the target set. However, there is no way of knowing where such a trajectory will lie before thoroughly computing the value function.

Methods that exploit machine learning have great potential because they are state discretization-free and do not depend on the dynamic programming principle. Unfortunately, many machine learning techniques cannot make the guarantees provided by reachability analysis. For instance, [21] and [91] use neural networks (NNs) as nonlinear optimizers to synthesize trajectories which may not be dynamically feasible. The authors in [5] propose a supervised learning-based algorithm that depends heavily on feature tuning and design, making its application to high-dimensional problems cumbersome. In [55] and [119], the authors successfully use NNs for approximating the value function, but the approximation is not guaranteed to be conservative.

In this section, we attempt to combine the best features of both dynamic programming-based optimal control and machine learning using an NN-based algorithm. Our proposed grid-free method conservatively approximates the value function in only a region around a feasible trajectory. Unlike previous machine learning techniques, our technique guarantees a direction of conservatism, and unlike previous dynamic programming-based methods, our approach involves an NN that effectively finds the relevant region that requires a value function. Our contributions will be presented as follows:

(1) In Section 5.1.1, we summarize optimal control and the formalisms used for

this work. (2) In Section 5.1.2, we provide an overview of the full method and the core ideas behind each stage, as well as highlight the conservative guarantees of the method. (3) In Sections 5.1.3 and 5.1.4, we present the two phases of our proposed algorithm and the underlying design choices. (4) In Section 5.1.5, we illustrate our guaranteed-conservative approximation of the value function, and the resulting near-optimal trajectories for the Dubins Car. (5) In Section 5.1.6, we conclude and discuss future directions.

5.1.1 Problem Formulation

In this section, we will provide some definitions essential to express our main results. Afterwards, we will briefly discuss the goals of this section.

5.1.1.1 Optimal Control Problem

Consider a dynamical system governed by the following ordinary differential equation (ODE):

$$\begin{aligned} \dot{z} &= f(z,u), t \in [t_0, 0] \\ z(t_0) &= \bar{z}, \quad u \in \mathcal{U} \end{aligned} \tag{5.1}$$

Note that since the system dynamics (5.1) is time-invariant, we assume without loss of generality that the final time is 0.

Here, $z \in \mathbb{R}^n$ is the state of the system and the control function $u(\cdot)$ is assumed to be drawn from the set of measurable functions. Let us further assume that the system dynamics $f : \mathbb{R}^n \times \mathcal{U} \to \mathbb{R}^n$ are uniformly continuous, bounded, and Lipschitz continuous in z for fixed u. Denote the function space from which f is drawn as $\mathbb{F}$.

With these assumptions, given some initial state z, initial time t_0, and control function $u(\cdot) \in \mathbb{U}$, there exists a unique trajectory solving (3.17). We refer to trajectories of (3.17) starting from state z_1 and time t_1 as $\xi^f(t; z_1, t_1, u(\cdot))$, with $z_1 \in \mathbb{R}^n$ and $t_0 \leq t, t_1 \leq 0$. Trajectories satisfy an initial condition and (3.17) almost everywhere:

$$\begin{aligned} \frac{d}{dt}\xi^f(t; z_1, t_1, u(\cdot)) &= f(\xi^f(t; z_1, t_1, u(\cdot)), u(t)) \\ \xi^f(t_1; z_1, t_1, u(\cdot)) &= z_1 \end{aligned} \tag{5.2}$$

Note that we can use the trajectory notation to specify states that satisfy a final condition if $t \leq t_1$. In this section of the thesis this will often be the case, since our NN will be producing backward-time trajectories.

Consider the following optimal control problem with final state constraint[1]:

[1]For simplicity we constrain the final state to a single state; our method easily extends to the case with a set-based final state constraint, $z(0) \in \mathcal{L}$.

$$\begin{aligned} V(z,t) &= \min_{u(\cdot),t_0} \int_t^0 C(u(\tau))d\tau \\ \text{subject to } z(t_0) &= \bar{z}, z(0) = z_{\mathcal{L}} \end{aligned} \tag{5.3}$$

The value function $V(z,t)$ is typically obtained via dynamic programming-based approaches such as [17, 23, 64, 111, 122, 135, 146], and an appropriate HJB partial differential equation is solved backwards in time on a grid representing the discretization of states. Once $V(z,t)$ is found, the optimal control, which we denote $u^*(z,t)$, can be computed based on the gradient of $V(z,t)$:

$$u^*(z,t) = \arg\min_{u \in \mathcal{U}} \nabla V(z,t) \cdot f(z,u) \tag{5.4}$$

Unfortunately, the computational complexity of these methods scales exponentially with the state space dimension.

5.1.1.2 Goal

We seek to overcome the exponentially scaling computational complexity. Our approach is inspired by two inherent challenges that dynamic programming-based methods face. First, since only relatively mild assumptions are placed on the system dynamics (3.17), optimal trajectories are *a priori* unknown and could essentially trace out any arbitrary path in the state space. Dynamic programming ignores this issue by considering *all possible* trajectories. Second, in a practical setting, the system starts at some particular state z_0. Thus, the optimal control, and in particular $\nabla V(z,t)$ in (5.4), is needed only in a "corridor" along the optimal trajectory. However, since the optimal trajectory is *a priori* unknown, dynamic programming-based approaches resort to computing $V(z,t)$ over a very large portion of the state space so that the $\nabla V(z,t)$ is available regardless of where the optimal trajectory happens to be.

We propose a method that, in contrast to dynamic programming-based methods,

1. has a substantially faster computation time and smaller memory-usage;
2. is a flexible and general framework that can be applied to higher-order systems with just hyperparameter tuning;
3. generates an approximate value function $\hat{V}(z,t)$ from which a controller that drives the system to the target can be synthesized;
4. guarantees that $\hat{V}(z,t) \geq V(z,t)\ \forall x,t$, so that a direction of conservatism can be maintained despite the use of an NN.

We enforce 1) by avoiding operations that exhaustively search the state space. As seen in Section 5.1.3.2, the training and final data sets are either generated randomly

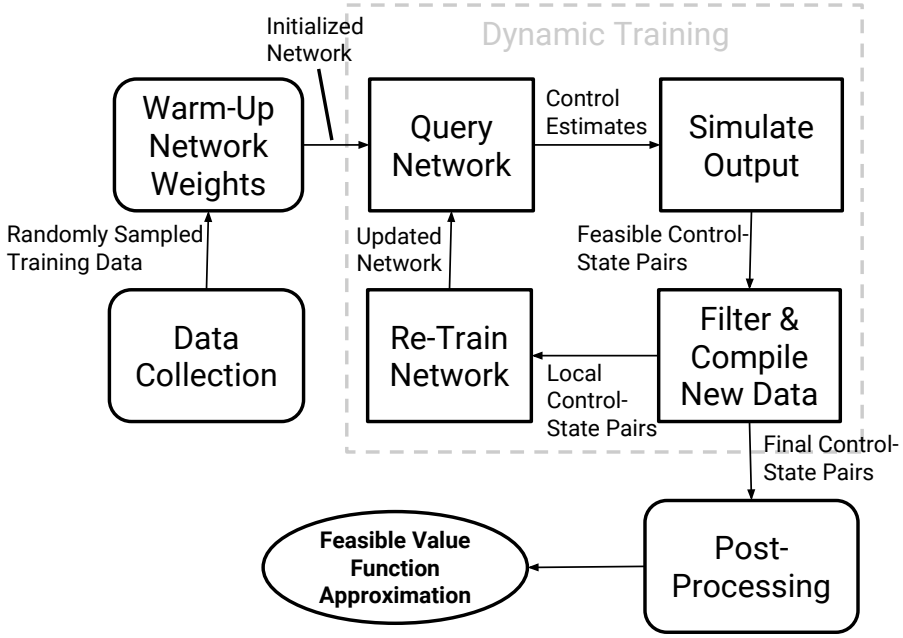


Figure 5.1: Stages of proposed method

or outputted by the NN, both constant time operations. Furthermore, we rely on NNs being universal function approximators [46] to make our method general to the system dynamics, thus satisfying 2). While our method does have some limitations, as discussed at the end of Section 5.1.3.1, we do not believe that these limitations are restrictive in the context of optimal control. Our post-processing of the NN outputs outlined in Section 5.1.4 satisfies 3). Finally, we use the dynamics of the system to ensure our final output satisfies 4); this is detailed in Section 5.1.3.2.

Our method overcomes the challenges faced by dynamic programming-based methods in two phases: the NN training phase, which allows the NN to learn the inverse backward system dynamics while also generating a dataset from which we can obtain a conservative approximation of the value function; and the controller synthesis phase, which uses the approximation to synthesize a controller to drive the system from its initial state to its target.

5.1.2 Overview

In this section we will briefly describe the different stages of our computational framework (depicted in Fig. 5.1). We leave the details of each stage to subsections of Section 5.1.3 and 5.1.4.

5.1.2.1 Pre-Training

In the pre-training phase (detailed in Section 5.1.3.2), we produce some initial datasets to prepare the NN for the dynamic training procedure. The dynamic training procedure is already designed to help the NN improve its performance iteratively; however, the pre-training phase speeds up the convergence by warming up the NN with sampled data of the system dynamics, f.

5.1.2.2 Dynamic Training

Our proposed dynamic training loop is tasked with both improving the approximation of the NN and producing the necessary data from which we extract our value function approximation. Fortunately, these two tasks are inherently tied to one another.

We pose our NN as an approximation of the inverse dynamics of the system (this is formally clarified in Section 5.1.3). In other words, if we query our NN with a set of states to check its approximation, then the NN will predict a set of controls to drive the system to those states (detailed in Section 5.1.3.2). We can check the correctness of this prediction by simulating the controls through f (detailed in Section 5.1.3.2). Even if this prediction is inaccurate, we can recycle the new data the NN produced through an attempted prediction, and add the data into our training set to re-train the NN. Since we simulated the NN's prediction to get the *actual* states the predicted control drives the system to, we now have corrective data with which we can re-train our NN. By doing this repeatedly, we iteratively train the NN with feedback. The data produced by the NN can be used to evaluate an approximate value of the value function using (5.3). Thus, with this dynamic training loop, we are able to produce data to construct an approximate value function while iteratively improving our NN's prediction capabilities. Furthermore, as the NN improves, the relevance of the data also improves.

To further encourage the process, in every iteration we additionally apply stochastic filters to our dataset that favor more local and optimal data (detailed in Section 5.1.3.2). This way, we can ensure the NN will not saturate or keep any unnecessary data for our final value function approximation.

5.1.2.3 Post-Training

Once dynamic training is complete, the NN will be able to make accurate predictions and our dataset will encompass the region over which an approximate value function is needed. After some post-processing over the dataset (detailed in Section 5.1.3.3), we will show that we can successfully extract a value function approximation from which we synthesize control to drive our system from our initial state to our goal state.

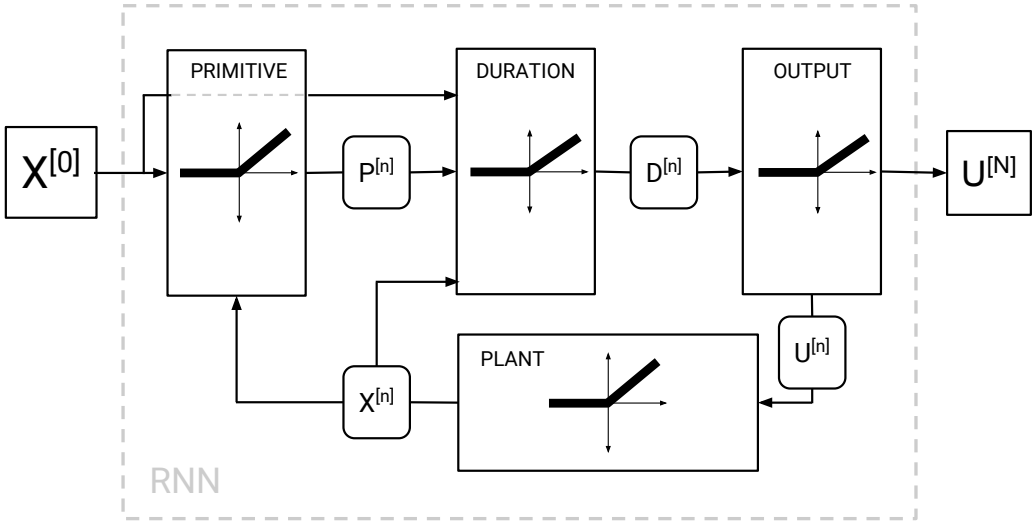


Figure 5.2: Four-layer RNN which takes input states and outputs discretized control.

5.1.3 Neural Network Training Phase

Given a target state $z_{\mathcal{L}}$ and assuming the system starts at some initial state $\bar{z}$, we want to design an NN that can produce a control function $u(\cdot)$ that drives the backward-time system from $z_{\mathcal{L}}$ to $\bar{z}$. More concretely, consider the inverse backward dynamics of the system, denoted $g_{z_{\mathcal{L}}} : \mathbb{R}^n \times \mathbb{F} \to \mathbb{U}$, and defined to be

$$g_{z_{\mathcal{L}}}(\bar{z}; -f(\cdot,\cdot)) = u^*(\cdot) \tag{5.5}$$

where the optimal control $u^*(\cdot)$ is defined for $t \leq 0$. For simplicity, we will write $g_{z_{\mathcal{L}}}$ as g from now on. Given some $u^*(\cdot)$ defined in the time interval $[t, 0]$, we have $\bar{z} = \xi^f(0; z_{\mathcal{L}}, t, u^*(\cdot))$.

Our NN is an approximation of g, and we will denote the NN as $\hat{g}$. Let $\hat{u}(\cdot)$ be the control produced by $\hat{g}$, and let the time interval for which $\hat{u}(\cdot)$ is defined be denoted $[-\hat{T}, 0]$. The primary tasks of our training procedure will be to: (1) iteratively improve our NN's training set so $\hat{g}$ approaches g in the region local to the path given by $\xi^f(0; z_{\mathcal{L}}, t, u^*(\cdot)), t \leq 0$; and (2) produce a dataset of states, their corresponding control and, in turn, the corresponding value approximation, which will be used for control synthesis.

5.1.3.1 Neural Network Architecture

Denote the maximum time horizon as $\bar{T}$. To reduce the space in which the NN needs to look for candidate control functions, we assume that $\hat{u}(\cdot)$ returned by the NN is composed of two finite sequences $\{u^j\}_{j=1}^K, \{\tau^j\}_{j=1}^K$ called the sequence of

control primitives and the sequence of time durations respectively. Mathematically, the control function $\hat{u}(\cdot)$ is of the form

$$\hat{u}(t) = \begin{cases} u^1, & t \in [-\sum_{j=1}^{K} \tau^j, -\sum_{j=2}^{K} \tau^j] \\ u^2, & t \in (-\sum_{j=2}^{K} \tau^j, -\sum_{j=3}^{K} \tau^j] \\ \cdots & \\ u^{K-1}, & t \in \left(-\tau^K - \tau^{K-1}, -\tau^K\right] \\ u^K, & t \in \left(-\tau^K, 0\right] \end{cases} \tag{5.6}$$

where we implicitly define $\hat{T} = \sum_{j=1}^{K} \tau^j$. Since we will only use the control to obtain the approximation $\hat{V}(\cdot, \cdot)$ and not for actually controlling the system, we do not need the generated controls to be extremely accurate or continuous, as we will explain in Sections 5.1.3.2 and 5.1.4.

We propose a rectified linear unit recurrent NN (RNN) with the following structure:

$$\begin{aligned} \text{Primitive Layer: } & P^{[n]} = \psi(W_P \cdot X^{[n]} + b_P) \\ \text{Duration Layer: } & D^{[n]} = \psi(W_{D_1} \cdot P^{[n]} + W_{D_2} \cdot X^{[n]} + b_D) \\ \text{Control Layer: } & U^{[n]} = W_L \cdot D^{[n]} + b_L \\ \text{Plant Layer: } & X^{[n+1]} = \psi(W_X \cdot U^{[n]} + b_X) \end{aligned}$$

where ψ is the positive rectifying function defined as $\psi(a) = \max(0, a)$ and $n = 0, 1, \ldots, N$. The input of the RNN is $\bar{z}$ ($X^{[0]} = \bar{z}$), and the output is some $\hat{u}(\cdot)$ ($U^{[N]} = \hat{u}(\cdot)$), that approximately brings the system from $\bar{z}$ to $z_{\mathcal{L}}$. The parameters we learn through training are the weights W_P, W_{D_1}, W_{D_2}, W_L, W_X and biases b_P, b_D, b_L, b_X. All weights and biases will be collectively denoted $\mathbb{W}$. For prediction, $\hat{u}(\cdot)$ and training example, $u(\cdot)$, we discretize both controls using (5.6), then the training is performed with mean-squared error (MSE), or

$$MSE = \frac{1}{n} \sum_{n=1}^{N} (\hat{u}^n - u^n)^2 \tag{5.7}$$

as the cost function, where $u(\cdot)$ is our training example output.

As already mentioned, the primitive layer takes a state as input and computes a control primitive. The duration layer takes in the primitive layer's output and the same input state, and outputs a time duration. This time duration is then passed through the control (also called output) linear layer, which outputs the sequences $\{u^j\}, \{\tau^j\}$ representing the control function $\hat{u}(\cdot)$. Afterwards, the control function is fed into the plant layer, which attempts to encode the backward dynamics $-f$. The plant's output state, $X^{[n+1]}$, is then fed back to the primitive layer.

Explicitly, the RNN can be written as

$$\hat{g}(\bar{z}, z_{\mathcal{L}}; -f(\cdot,\cdot), \mathbb{W}) = \hat{u}(\cdot) = U^{[N]} \tag{5.8}$$

where $\hat{u}(\cdot)$ is given in the form of the sequences $\{u^j\}, \{\tau^j\}$.

In the next section, we discuss the dynamic training procedure for this RNN.

5.1.3.2 Detailed Dynamic Training

Warm-up and Initial Training: We first train the RNN without knowledge of $\bar{z}$. For starters, we will require training examples in the form of $\{\hat{z}_i, \hat{u}_i(\cdot)\}$ that sufficiently capture the basic behaviors of the system dynamics. To do this, we randomly generate $\{\hat{z}_i\}$ using an accept-reject algorithm similar to the one in [55], which is described in Alg. 3.

Algorithm 3: Exponential Filter

Result: A
Inputs: z, $\mathcal{R}$, λ
compute z_{proj} via (5.9)
generate β uniformly from $[0, \lambda]$;
if $z \in \mathcal{R}$ **then**
  A = true;
else
  if $\beta \leq \lambda e^{-\lambda \|x - x_{\text{proj}}\|_2}$ **then**
    A = true;
  else
    A = false;
  end if
end if

Alg. 3 takes a state z, an accept region $\mathcal{R}$, and a decay rate λ as inputs. To use Alg. 3, we first compute z_{proj}, the Euclidean projection of the state z onto the set $\mathcal{R}$ as follows:

$$x_{\text{proj}} = \arg\min_{x'} \|x' - x\|_2 : x' \in \mathcal{R} \tag{5.9}$$

Using Alg. 3, we generate two training sets: one large dataset $\mathcal{D}_1$ and one small dataset $\mathcal{D}_2$. The large data set $\mathcal{D}_1$, used for supervised training of the plant layer (the weights W_X), is generated with a large accept region with a large number of accepted points. The smaller dataset $\mathcal{D}_2$, used to initialize the full RNN after the plant layer has been warmed up with $\mathcal{D}_1$, is generated with a smaller accept region and a relatively small number of accepted points.

Algorithm 4: Length Filter

Result: $\mathcal{X}$
Inputs: $\mathcal{X}$, λ_C, $\hat{C}(\mathcal{X})$, D
generate β uniformly from $[0, \lambda]$;
for $z_i \in \mathcal{X}$ **do**
  $\mathcal{X}_C \leftarrow \emptyset$;
  for $z_j \neq z_i \in \mathcal{X} : ||z_i - z_j||_2 \leq D$ **do**
    if $\beta > \lambda_C e^{-\lambda_C \hat{C}(z_j)}$ **then**
      $\mathcal{X}_C \leftarrow \mathcal{X}_C \cup x_j$;
    end if
  end for
  $\mathcal{X} \leftarrow \mathcal{X} \setminus \mathcal{X}_C$;
end for

Once the RNN's plant layer is warmed up with $\mathcal{D}_1$ and the full network has been trained on $\mathcal{D}_2$, the RNN is ready to be queried, setting the dynamic training loop into motion.

Neural Network Query: At the start of every training loop, the RNN is used to predict controls, $\{\hat{u}_i(\cdot)\}_{i=1}^N$, from a set of states $\{\bar{z}_i\}_{i=1}^N$.

For our training loop's first query, we uniformly sample M_ϵ states within a distance of ϵ to $\bar{z}$ to produce a set of states denoted $\mathcal{X}_\epsilon = \{\bar{z}_i\}_{i=1}^{M_\epsilon}$. Then, for all of the network queries, we query the RNN with the same $\mathcal{X}_\epsilon$ to get its current set of predictions, $\mathcal{V}_\epsilon = \{\hat{u}_i(\cdot)\}_{i=1}^{M_\epsilon}$. By using a set of states, $\mathcal{X}_\epsilon$, as opposed to using just a singular state, we capture more local trajectories and relax the need for every trajectory to lead exactly to a point location in the state space.

Simulate Output and Feasibility: In general, applying a control in $\mathcal{V}_\epsilon$ brings the backward system from $z_\mathcal{L}$ to some $\hat{z} \notin \mathcal{X}_\epsilon$. Thus, to find $\mathcal{V}_\epsilon$'s true set of resultant states, we simply apply each control in $\mathcal{V}\epsilon$ to f with $z_\mathcal{L}$ as the initial condition. This will yield $\mathcal{V}_\epsilon$'s true set of resultant states, which we denote as $\mathcal{X}'_\epsilon$.

This key step is what gives us our feasibilty guarantee. Since all of the data in our final output is a compilation of filtered data drawn from $(\mathcal{X}'_\epsilon, \mathcal{V}_\epsilon)$ at each training cycle, we know we have a dynamically feasible control for each state. In addition, since the controls are feasible, they must also be either optimal or suboptimal. **Therefore, when we compute values from our final set of controls using (5.3), these values must be strictly conservative.**

Filtering: Often the RNN will predict a $\mathcal{V}_\epsilon$ that lead to states far from our target. Since our method is intended to produce a value function approximation local to a relevant region of the state space, we apply an exponential accept-reject filter (Alg.

3) to $\mathcal{X}'_\epsilon$, with accept region $\mathcal{R}_\epsilon$ and decay rate λ_ϵ, to stochastically remove states that are not nearby. We let the remaining states and their corresponding controls be $(\mathcal{X}_{\text{new}}, \mathcal{V}_{\text{new}})$.

We choose the input accept region $\mathcal{R}_\epsilon$ and decay rate λ_ϵ provided to Alg. 3 such that the filter will generously accept states that could lie near a feasible path from $\bar{z}$ to $z_{\mathcal{L}}$. Though choosing a reasonable $\mathcal{R}_\epsilon$ for general high-dimensional systems with complicated dynamics before training could be difficult since we may be unable to provide even a generous guess of where in the state space optimal trajectories might lie, we can instead adjust the size or shape of $\mathcal{R}_\epsilon$ until the region begins accepting predictions from the RNN.

After finding $(\mathcal{X}_{\text{new}}, \mathcal{V}_{\text{new}})$, we can update our full training set on which we will train our RNN, for the next training iteration. We first improve our current training set, denoted as $(\mathcal{X}, \mathcal{V})$, by once again applying Alg. 3 with accept region $\mathcal{R}$ and decay rate λ. Then, we apply a second filter to all of our remaining data that favors controls with relatively low costs, this is detailed in Alg. 4 with inputs $\mathcal{X}$, the costs $\hat{C}(\mathcal{X})$, and decay rate λ_C and search radius D. We let the remaining data set from filtering $(\mathcal{X}, \mathcal{V})$ be called $(\mathcal{X}_{\text{old}}, \mathcal{V}_{\text{old}})$. Then, we compile our new training set for the next training cycle as $(\mathcal{X}, \mathcal{V}) = (\{\mathcal{X}_{\text{old}}, \mathcal{X}_{\text{new}}\}, \{\mathcal{V}_{\text{old}}, \mathcal{V}_{\text{new}}\})$.

5.1.3.3 Post-processing

In order to drive the system from $\bar{x}$ to $x_{\mathcal{L}}$, the value function, and in particular its gradient, is necessary at points between $\bar{z}$ and $z_{\mathcal{L}}$ along a dynamically feasible trajectory. Fortunately, this information can be computed from $\mathcal{X}$ and $\mathcal{V}$. Specifically, $\hat{z}_i \in \mathcal{X}$ and $\hat{u}_i(\cdot) \in \mathcal{V}$ produce a trajectory $\xi_i^f(0; \hat{z}_i, t, \hat{u}_i(\cdot)), t \in [\hat{T}, 0]$. From the trajectories, we can obtain M_i states on the trajectory by discretizing the time t into M_i points. We denote these states $z_{(i,j)}$, where the index i comes from the index of $\hat{z}_i \in \mathcal{X}$, and the index $j \in \{0, \ldots, M_i - 1\}$ indicates that the state is computed from the jth time point on the trajectory ξ_i^f. Mathematically, $\hat{z}_{(i,j)}$ is given as follows:

$$\begin{aligned} \hat{z}_{(i,j)} &= \xi_i^f(0; \hat{x}_i, t_j, \hat{u}_i(\cdot)), \\ t_j = -\frac{j\hat{T}_i}{M_i - 1}, j &= 0, 1, \ldots, M_i - 1 \end{aligned} \tag{5.10}$$

Once we have explicitly added data along the trajectories from our dataset, we now have a dataset that spans the local state space around and between $\bar{z}$ and $z_{\mathcal{L}}$. To get our value function approximation across our dataset, we can simply use (5.3). Explicitly, for state-control pair, $(\hat{x}_i, \hat{u}_i(\cdot)) \in \mathcal{X} \times \mathcal{V}$, and t_j, we have the approximate value function at states $x_{(i,j))}$ and times t_j:

$$\hat{V}(x_{(i,j)}, t_j) = \int_{t_j}^{0} C(\hat{u}_i(t))dt \tag{5.11}$$

5.1.4 Controller Synthesis Phase

After we obtain our value function approximation, we can synthesize control to drive our system from $\bar{z}$ to $z_{\mathcal{L}}$ using (5.4) with the appropriate gradient components of the value function. However, since our value function approximation is irregular and based in a point set, we will first define a special computation for obtaining the gradient.

To compute $(\nabla V)_i$, the ith component of the gradient at a given state z, we search above and below in the $\hat{i}$ direction for states in $\mathcal{X}$ within a hyper-cylinder of tunable radius r. We define the closest point within the hyper-cylinder above z as z_a and below as z_b. If some z_a and z_b exist, we compute the gradient at z as:

$$(\nabla V(z))_i = \frac{V(z_a) - V(z_b)}{(z_a)_i - z_b)_i} \tag{5.12}$$

If we have multiple states above but none below, we approximate the gradient using (5.12), with z_a being the closest state above and z_b being the second closest state above. A similar procedure holds if we have multiple states below but none above.

In general, we can use a finite element method to compute gradient values for a non-regular grid, which involves using shape functions as basis functions to interpolate the gradient values between nodes.

5.1.5 Dubins Car Example

5.1.5.1 Vehicle Dynamics

Consider the Dubins Car [57], with state $z = (p_x, p_y, \theta)$. (p_x, p_y) are the x and y positions of the vehicle, and θ is the heading of the vehicle. The system dynamics, assuming unit longitudinal speed, are

$$\begin{aligned} \dot{p}_x = \cos\theta,\ \dot{p}_y = \sin\theta \\ \dot{\theta} = u, \qquad |u| \leq 1 \end{aligned} \tag{5.13}$$

The control of the Dubins car is denoted u, and is constrained to lie in the interval $[-1, 1]$, the interpretation of which is that the vehicle has a maximum turn rate of 1 rad/s. In addition, we only accrue cost on our control with the duration of the control. Formally, this means that $C(u(t)) = 1$. We choose the Dubins car to illustrate our method because of the simple structure of the optimal controls. In addition, since the model is only 3D, we are able to verify our results by comparing them to the those obtained via HJ reachability.

For our example, we choose many different initial states $\bar{z}$ for the system. The target state $z_{\mathcal{L}}$ is chosen as the origin.

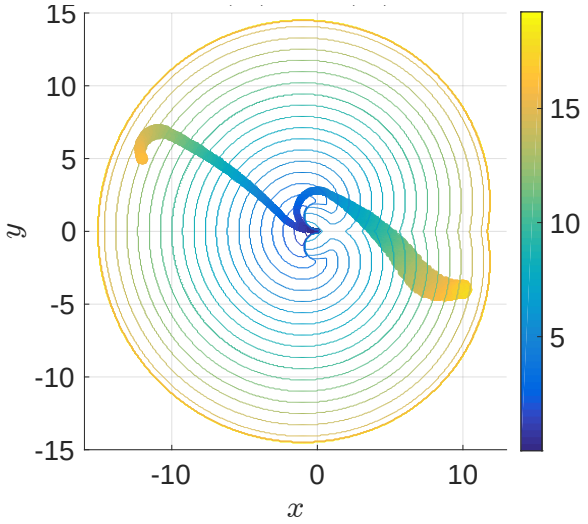


Figure 5.3: Comparison of the true value function $V(z, t_0^*)$ computed over a large portion of the state space, and the approximate value function from our NN-based approach $\hat{V}(z_{(i,j)}, t_j)$ on two different corridors containing the initial state $\bar{z}$, the target set $z_{\mathcal{L}}$, and a dynamically feasible trajectory. The contours are level sets of $V(z, t_0^*)$. Two different corridors in which $\hat{V}$ is computed resulting from two different values of $\bar{x}$ $((10, -4, -3), (-12, 5, 2))$ are also plotted using the same colormap.

5.1.5.2 Control Primitives

In [57], Dubins shows that all optimal trajectories of this system utilize controls that represent going straight or turning maximally left or right. Thus, the set of controls that are valid for generating optimal trajectories can be reduced down to three motion primitives, {'L', 'S', 'R'}, encoding the controls $u = 1$ (max left), $u = 0$ (straight), and $u = -1$ (max right) respectively. Though [57] additionally provides an algebraic solution to the optimal control problem, we purposefully do not leverage this result, as many interesting systems do not have such a simple method of deriving optimal control. Instead, we use five motion primitives, encoding the controls $u = 2$, $u = 1$, $u = 0$, $u = -1$, $u = -2$.

Following our notation in Section 5.1.3, we write a n length control sequence as $\{u^1, u^2, \ldots, u^n\}, \{\tau^1, \tau^2, \ldots, \tau^n\}$ where u^i denotes the ith control primitive and τ^i denotes the duration of ith control primitive.

5.1.5.3 Neural Network

Using the RNN architecture described in Section 5.1.3.1, we let $N = 3$, since we only need at most three control primitives. Since the controls and dynamics of Dubins car are simple, we have chosen the number of neurons in layers P, D, U, X to be $[10, 10, 6, 75]$, respectively. The NN is trained with the training functionality of

the MATLAB Neural Network ToolBox 2016a. The training function used for the full NN is resilient back-propagation and the performance function used is mean squared error.

5.1.5.4 Dynamic Training

For $\mathcal{D}_1$ and $\mathcal{D}_2$, we choose each u^i from the three possible values $\{-1, 0, 1\}$. The controls for $\mathcal{D}_1$ have durations, τ^i, uniformly sampled from $[0, \bar{T}]$ where $\bar{T} = 100$. $\mathcal{D}_2$'s control durations are also uniformly sampled in the same manner, but with $\bar{T} = 2\pi$.

For the dynamic training parameters, our query set $\mathcal{X}_\epsilon$ is generated with $\epsilon = 1$ and $M_\epsilon = 500$.

Filtering Algorithms: Throughout training, the exponential filtering process uses two accept regions. In early iterations, $\mathcal{R} = \mathcal{R}_c$ is defined as the cone of minimum size that contains $\mathcal{X}_\epsilon$, with the tip of the cone located at $\bar{x}$. The choice of using a conical filter to guide the neural net at first is based on the hypothesis that a trajectory taking the system from $\bar{z}$ to $x_{\mathcal{L}}$ is likely to stay in the cone $\mathcal{R}_c$. In later iterations, $\mathcal{R} = \mathcal{R}_s$ is chosen as $\mathcal{X}_\epsilon$. This spherical filter, centered at $x_{\mathcal{L}}$, helps to more finely guide the neural network to $x_{\mathcal{L}}$.

We also apply length filtering with a small distance parameter $D = 0.5$, since we would like to be comparing distance costs only between trajectories with similar end points.

Filter Decay Rate: Setting and Timing: Although $\mathcal{R}_\text{C}$ and $\mathcal{R}_\text{S}$ are chosen before the dynamic training process, the filtering of the training set $\mathcal{X}$ can still be adjusted while training. This is done by varying the parameters λ_C and λ_S. In early training iterations, we want to decrease λ_C slightly to ensure that we are not filtering out states needed for the NN to explore the state space. Once the NN has gained a better understanding of how to reach $\bar{z}$, we increase λ_C and λ_S slightly to further encourage the NN to drive states near $\bar{z}$. When the dataset is mostly near $\bar{z}$, we increase λ_S and λ_S significantly. The decay rate for the length filter, λ_C, is constant over iterations.

5.1.5.5 Dubins Car Results

Training Process: In Fig. 5.4a, 5.4b, 5.4c, the process by which the training set $\mathcal{X}$ iteratively changes from the initial training set $\mathcal{D}_2$ to encompassing $\mathcal{X}_\epsilon$ is shown. Here, the red states represent the set $\mathcal{X}_\epsilon$ and the black states represent the current states in $\mathcal{X}$. In the early iterations (Fig. 5.4a), the NN explores outward from the initial training set, frequently making mistakes, resulting in the states in $\mathcal{X}$ being very far away from $\bar{z}$. As the iteration number increases, the trajectory ambiguous training set $\mathcal{D}_2$ is gradually cut down, and eventually the NN begins to predict controls $\hat{u}_i(\cdot)$ that drive to states $\hat{z}_i$ in an arc that heavily intersects $\mathcal{X}_\epsilon$. This can be seen in Fig.

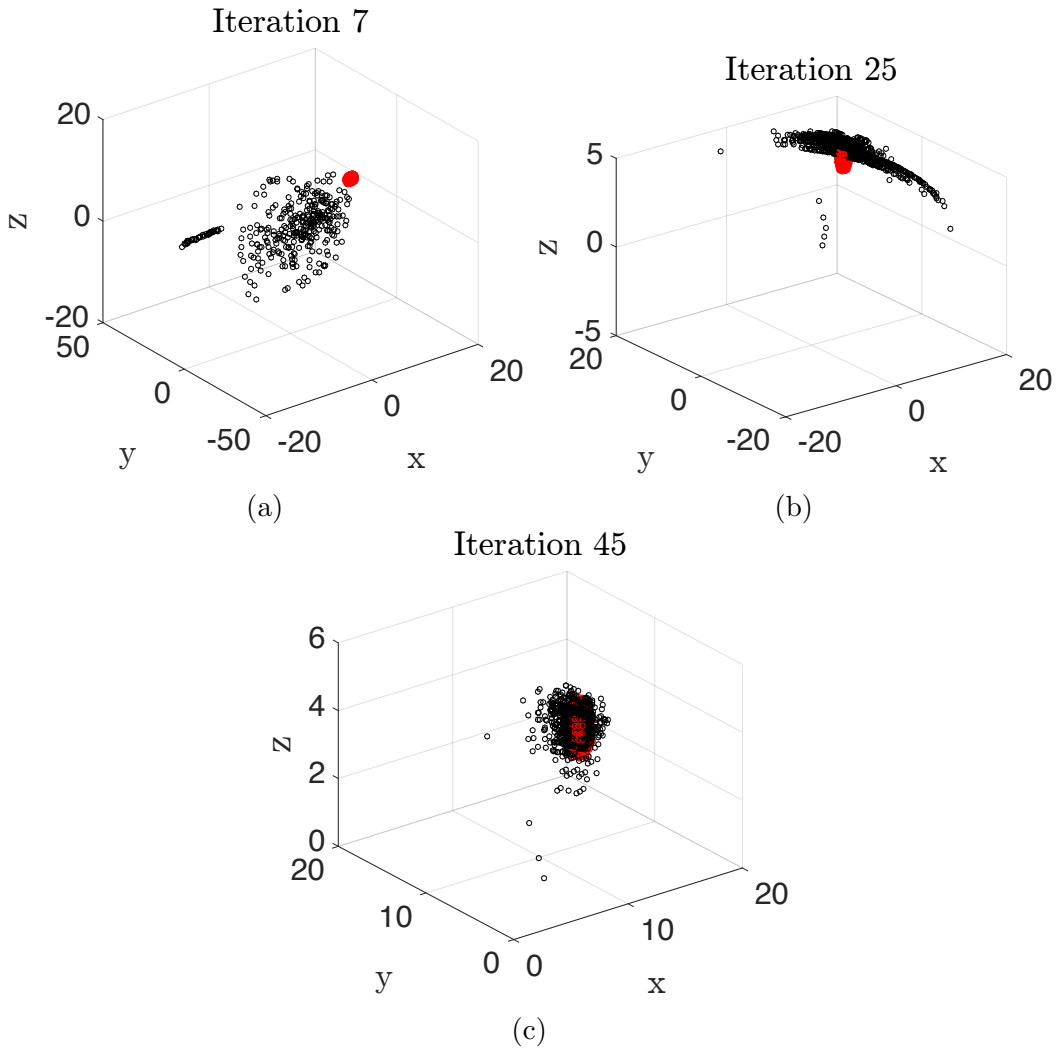
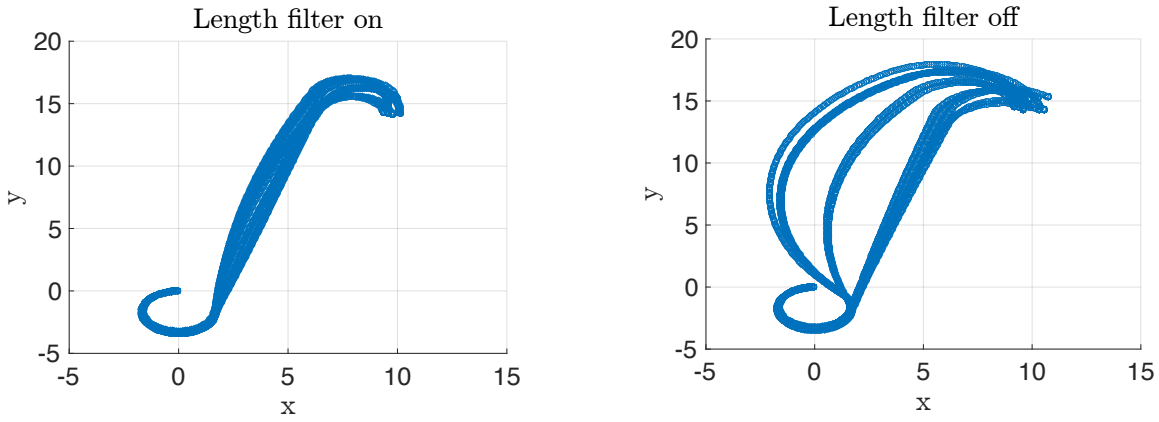


Figure 5.4: Evolution of $\mathcal{X}$ over many iterations.

5.4b. By the end of the training process, the $\mathcal{R}_O$ conical target filter prunes the states outside of the $\mathcal{X}_\epsilon$. This can be seen in Fig. 5.4c.

The length filter also enables our method to be robust to suboptimal training data. If we provide the neural net with a mixture of optimal and suboptimal training data, the length filter improves the quality of $\mathcal{X}$ by removing many states generated by using suboptimal control (Figure 5.5a), compared to without the length filter (Figure 5.5b).

Value Function Comparison: Using level set methods [111], we computed $V(x,t)$, and compared the true value function, $V(\bar{z}, t_0^*)$ and the approximate value function, $\hat{V}(\bar{z}, \hat{T})$ computed for several states in Table 5.1. t_0^* denotes the time component of the optimal solution of (5.3).



(a) (b)

Figure 5.5: Effect of the length filter on quality of $\mathcal{X}$. (a) shows $\mathcal{X}$ when the filter is on, and (b) shows $\mathcal{X}$ when it is off.

State $\bar{z}$	NN Cost $\hat{V}(\bar{z}, \hat{T})$	True Cost $V(\bar{z}, t_0^*)$
$(-12, 5, 2)$	26.51	14.84
$(-10, 0, 0)$	10.23	10.00
$(1, 1, 6)$	9.93	7.40
$(10, -4, -3)$	16.20	13.36

Table 5.1: Trajectory values (seconds)

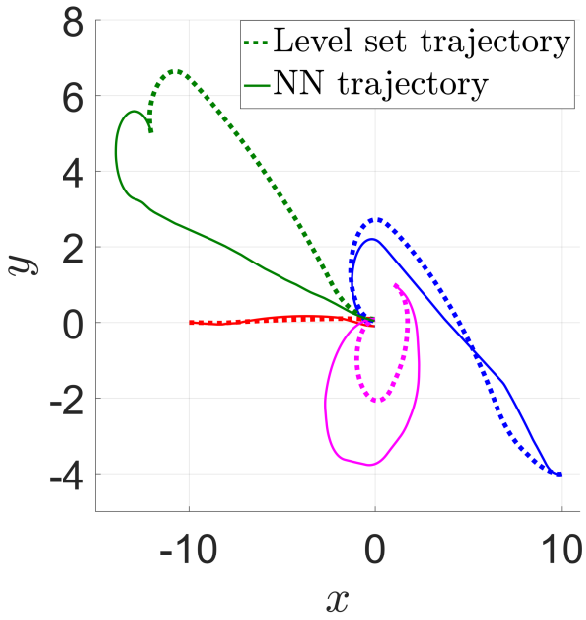


Figure 5.6: Trajectories generated using level set methods (dashed) and using our NN-based method (solid). Each color corresponds to a different initial state $\bar{x}$.

Computation Time: Synthesizing control using our NN-based approach allows for large time complexity improvements in comparison to using level set methods. On a 2012 MacBook Pro laptop, data generation requires approximately 3 minutes, and controller synthesis from this data and simulation requires 2 minutes on average. Since the region of the state space we are considering is quite large, and the target set is quite small (a singleton), the level set methods approach is intractable on this laptop, and requires 4 days on a desktop computer with a Core i7-5820K processor and 128 GB of RAM.

There are also large spatial savings by using the NN. For example, $\mathcal{X}$ and $\hat{V}$ for one particular corridor computed between $(10, -1, -3)$ and $(0, 0, 0)$ requires only 179 MB, while a reachable set computed over that horizon on a very low resolution grid requires approximately 7 GB.

As can be seen from this and the previous sections, using level set methods not only is more time-consuming compared to using our NN-based approach, but also does not guarantee a more shorter trajectory due to discretization error.

5.1.6 Conclusions and Future Work

Our NN-based grid-free method computes an upper bound of the optimal value function in a region of the state space that contains the initial state, the target set, and a feasible trajectory. By combining the strengths of dynamic programming-based and machine learning-based approaches, we greatly alleviate the curse of dimension-

ality while maintaining a desired direction of conservatism, effectively avoiding the shortcomings of both types of approaches.

Using a numerical example, we demonstrate that our approach can successfully generate a value function approximation in multiple test cases for the Dubins car. We are even able to approximate value function values in regions that are very far from the target set, a very computationally expensive task for dynamic programming-based approaches. Our approximate value function is able to drive the Dubins car from many different initial conditions to the target set.

Although our current results are promising, much more investigation is still needed to make our approach more practical and applicable to more scenarios. For example, better intuition for the choice of accept regions in the filtering process is needed to extend our approach to other systems. We currently plan to investigate applying our method to the 6D engine-out plane [2] as well as a 12D quadrotor model. In addition to path planning, we also hope to extend our theory to provide safety guarantees and robustness against disturbances. Such extensions are non-trivial due to the different roles that the control and disturbance inputs play in the system dynamics.

5.2 FaSTrack: a Modular Framework for Fast and Guaranteed Safe Motion Planning

This section is an adaptation of the paper in [76].

As unmanned aerial vehicles (UAVs) and other autonomous systems become more commonplace, it is essential that they be able to plan safe motion paths through crowded environments in real time. This is particularly crucial for navigating through environments that are *a priori* unknown. However, for many common dynamical systems, accurate and robust path planning can be too computationally expensive to perform efficiently. In order to achieve real-time planning, many algorithms use highly simplified model dynamics or kinematics, resulting in a tracking error between the planned path and the true high-dimensional system. This concept is illustrated in Fig. 5.7, where the path was planned using a simplified planning model, but the real vehicle cannot track this path exactly. In addition, external disturbances (e.g. wind) can be difficult to account for. Crucially, such tracking errors can lead to dangerous situations in which the planned path is safe, but the actual system trajectory enters unsafe regions.

We propose the modular tool FaSTrack: Fast and Safe Tracking, which models the navigation task as a sophisticated *tracking system* that pursues a simplified *planning system*. The tracking system accounts for complex system dynamics as well as bounded external disturbances, while the simple planning system enables the use of real-time planning algorithms. Offline, a precomputed pursuit-evasion game between the two systems is analyzed using Hamilton Jacobi (HJ) reachability analysis. This

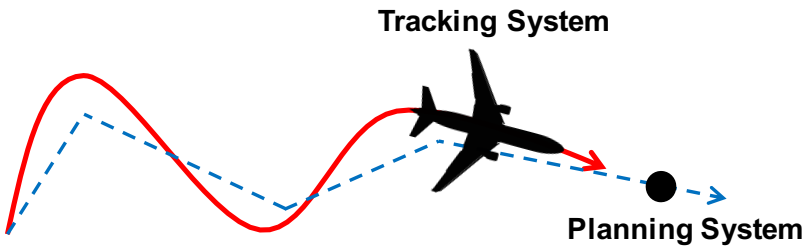


Figure 5.7: A planning system using a fast but simple model, followed by a tracking system using a dynamic model

results in a *tracking error function* that maps the initial relative state between the two systems to the *tracking error bound*: the maximum possible relative distance that could occur over time. This tracking error bound can be thought of as a "safety bubble" around the planning system that the tracking system is guaranteed to stay within. Because the tracking error is bounded in the relative state space, we can precompute and store a *safety control function* that maps the real-time relative state to the optimal safety control for the tracking system to "catch" the planning system. It is important to note that the offline computations are *independent* of the path planned in real-time; what matters are the relative states and dynamics between the systems, not the absolute state of the online path.

In the online computation, the autonomous system senses local obstacles, which are then augmented by the tracking error bound to ensure that no potentially unsafe paths can be computed. Next, a path or trajectory planner uses the simplified planning model to determine the next desired state. The tracking system then finds the relative state between itself and the next desired state. If this relative state is nearing the tracking error bound then it is plugged into the safety control function to find the instantaneous optimal safety control of the tracking system; otherwise, any controller may be used. In this sense, FaSTrack provides a *least-restrictive* control law. This process is repeated until the navigation goal is reached.

Because we designed FaSTrack to be modular, it can be used with existing fast path or trajectory planners, enabling motion planning that is rapid, safe, and dynamically accurate. In this section, we demonstrate this tool by computing the tracking error bound between a 10D quadrotor model affected by wind and a linear 3D kinematic model. Online, the simulated system travels through a static, windy environment with obstacles that are only known once they are within the limited sensing range of the vehicle. Combining this bound with a kinematic rapidly exploring random trees (RRT) fast path planner [94], [88], the system is able to safely plan and track a trajectory through the environment in real time.

5.2.1 Related Work

Motion planning is a very active area of research in the controls and robotics communities [78]. In this section we will discuss past work on path planning, kinematic planning, 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) [94], probabilistic road maps (PRM) [88], fast marching tree (FMT) [81], and many others [86, 92, 127] 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.

Motion planning for kinematic systems can also be accomplished through online trajectory optimization using methods such as TrajOpt [133] and CHOMP [125]. These methods can work extremely well in many applications, but are generally challenging to implement in real time for nonlinear dynamic systems due to the computational load.

Model predictive control (MPC) has been a very successful method for dynamic trajectory optimization in both academia and industry [124]. However, combining speed, safety, and complex dynamics is a difficult balance to achieve. Using MPC for robotic and aircraft systems typically requires model reduction to take advantage of linear programming or mixed integer linear programming [128, 143, 147]; robustness can also be achieved in linear systems [51, 126]. Nonlinear MPC is most often used on systems that evolve more slowly over time [52, 131], with active work to speed up computation [53, 117]. Adding robustness to nonlinear MPC is being explored through algorithms based on minimax formulations and tube MPCs that bound output trajectories with a tube around a nominal path (see [78] 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 [50,70]. Other methods include making use of safety funnels [103], or generating and choosing random trajectories at waypoints [85, 134]. The latter has been implemented successfully in many scenarios, but is risky in its reliance on finding randomly-generated safe trajectories.

Recent work has considered using offline Hamilton-Jacobi analysis to guarantee tracking error bounds, which can then be used for robust trajectory planning [14]. A similar new approach, based on contraction theory and convex optimization, allows computation of offline error bounds that can then define safe tubes around a nominal dynamic trajectory computable online [136].

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 [11].

The work presented here 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 static obstacles.

5.2.2 Problem Formulation

We seek to simultaneously plan and track a trajectory (or path converted to a trajectory) online and in real time. The planning is done using a kinematic or dynamic planning model. The tracking is done by a tracking model representing the autonomous system. The environment may contain static obstacles that are either known a priori or can be observed by the system within a limited sensing range (see Section 5.2.5). In this subsection we will define the tracking and planning models, as well as the goals of the thesis section.

5.2.2.1 Tracking Model

The tracking model is a representation of the autonomous 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 satisfy the ordinary differential equation:

$$\begin{aligned} \frac{ds}{dt} = \dot{s} = f(s, u_s, d), t \in [0, T] \\ s \in \mathcal{S}, u_s \in \mathcal{U}_s, d \in \mathcal{D} \end{aligned} \tag{5.14}$$

We assume that the system dynamics $f : \mathcal{S} \times \mathcal{U}_s \times \mathcal{D} \to \mathcal{S}$ are uniformly continuous, bounded, and Lipschitz continuous in s for fixed control u_s. The control function $u_s(\cdot)$ and disturbance function $d(\cdot)$ are drawn from the following sets:

$$\begin{aligned} u_s(\cdot) \in \mathbb{U}_s(t) = \{\phi : [0, T] \to \mathcal{U}_s : \phi(\cdot) \text{ is measurable}\} \\ d(\cdot) \in \mathbb{D}(t) = \{\phi : [0, T] \to \mathcal{D} : \phi(\cdot) \text{ is measurable}\} \end{aligned} \tag{5.15}$$

where $\mathcal{U}_s, \mathcal{D}$ are compact and $t \in [0, T]$ for some $T > 0$. Under these assumptions there exists a unique trajectory solving (5.14) for a given $u_s(\cdot) \in \mathcal{U}_s$ [**?**]. The trajectories of (5.14) that solve this ODE will be denoted as $\xi_f(t; s, t_0, u_s(\cdot))$, where $t_0, t \in [0, T]$ and $t_0 \leq t$. These trajectories will satisfy the initial condition and the ODE (5.14) almost everywhere:

$$\begin{aligned} \frac{d}{dt}\xi_f(t; s, t_0, u_s(\cdot)) = f(\xi_f(t; s, t_0, u_s(\cdot)), u_s(t)) \\ \xi_f(t; s, t, u_s(\cdot)) = s \end{aligned} \tag{5.16}$$

5.2.2.2 Planning Model

The planning model is used by the path or trajectory planner to solve for the desired path online. Kinematics or low-dimensional dynamics are typically used depending on the requirements of the 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 $s \in \mathcal{S}$. The dynamics similarly satisfy the ordinary differential equation:

$$\frac{dp}{dt} = \dot{p} = h(p, u_p), t \in [0, T], p \in \mathcal{P},\ \underline{u_p} \leq u_p \leq \overline{u_p} \tag{5.17}$$

Note that the planning model does not involve 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, including those that do not account for disturbances.

5.2.2.3 Goals of This Section

The goals of the section are threefold:

1. To provide a tool for precomputing functions (or look-up tables) to determine a guaranteed tracking error bound between tracking and planning models, and optimal safety controller for robust motion planning with nonlinear dynamic systems

2. To develop a framework for easily implementing this tool with fast real-time path and trajectory planners.

3. To demonstrate the tool and framework in an example using a high dimensional system

5.2.3 General Framework

The overall framework of FaSTrack is summarized in Figs. 5.8, 5.9, 5.10. The online real-time framework is shown in Fig. 5.8. At the center of this framework is the path or trajectory planner; our framework is agnostic to the planner, so any may be used (e.g. MPC, RRT, neural networks). We will present an example using an RRT planner in Section 5.2.6.

When executing the online framework, the first step is to sense obstacles in the environment, and then augment the sensed obstacles by a precomputed tracking error bound as described in Section 5.2.4. This tracking error bound is a safety margin that guarantees robustness despite the worst-case disturbance. Augmenting the obstacles by this margin can be thought of as equivalent to wrapping the planning system with a "safety bubble". These augmented obstacles are given as inputs to the planner

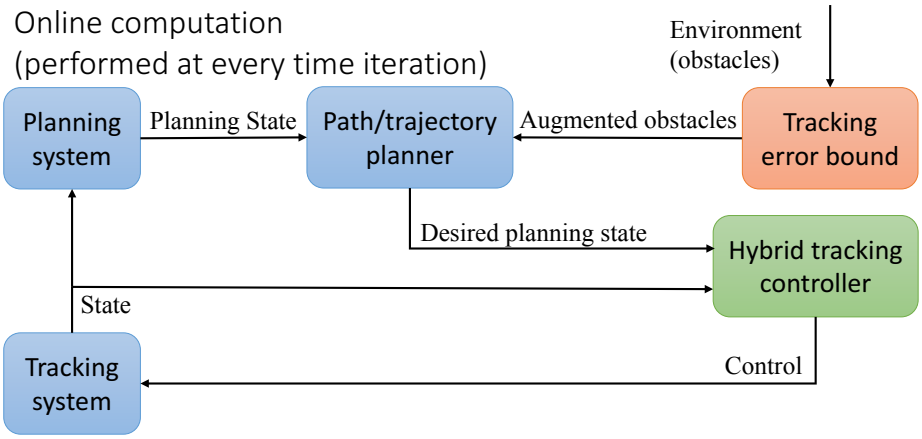


Figure 5.8: Online framework

along with the current state of the planning system. The planner then outputs the next desired state of the planning system.

The tracking system is a model of the physical system (such as a quadrotor). The hybrid tracking controller block takes in the state of the tracking system as well as the desired state of the planning system. Based on the relative state between these two systems, the hybrid tracking controller outputs a control signal to the tracking system. The goal of this control is to make the tracking system track the desired planning state as closely as possible.

The hybrid tracking controller is expanded in Fig. 5.9 and consists of two controllers: a 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. The safety controller consists of a function (or look-up table) computed offline via HJ reachability, and guarantees that the tracking error bound is not violated, *despite the worst-case disturbance*. In addition, the table look-up operation is computationally inexpensive. When the system is close to violating the tracking error bound, the safety controller must be used to prevent the violation. On the other hand, when the system is far from violating the tracking error bound, any controller (such as one that minimizes fuel usage), can be used. This control is used to update the tracking system, which in turn updates the planning system, and the process repeats.

To determine both the tracking error bound and safety controller functions/look-up tables, an offline framework is used as shown in Fig. 5.10. The planning and tracking system dynamics are plugged into an HJ reachability computation, which computes a value function that acts as the tracking error bound function/look-up table. The spatial gradients of the value function comprise the safety controller function/look-up table. These functions are independent of the online computations—

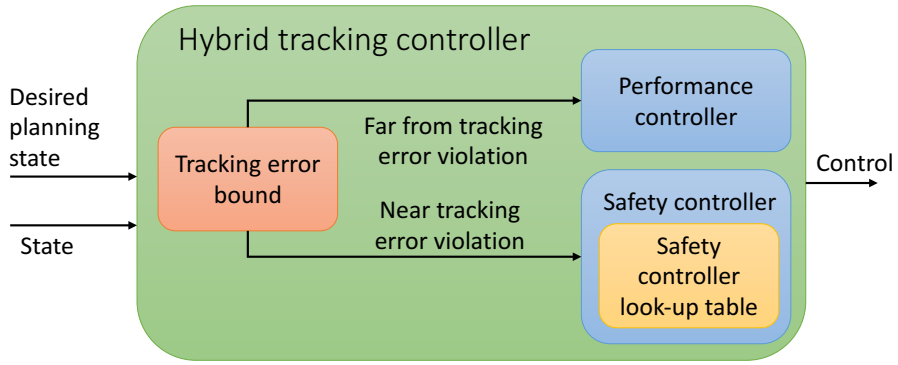


Figure 5.9: Hybrid controller

they depend only on the *relative* states and dynamics between the planning and tracking systems, not on the absolute states along the trajectory at execution time. In the following sections we will first explain the precomputation steps taken in the

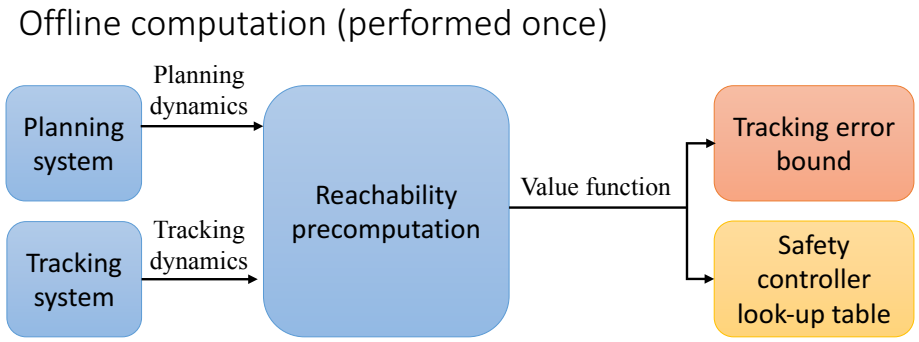


Figure 5.10: Offline framework

offline framework. We will then walk through the online framework and provide a complete example.

5.2.4 Offline Computation

The offline computation begins with setting up a pursuit-evasion game [38, 79] between the tracking system and the planning system, which we then analyze using HJ reachability. In this game, the tracking system will try to "capture" the planning system, while the planning system is doing everything it can to avoid capture. In reality the planner is typically not actively trying to avoid the tracking system, but this allows us to account for worst-case scenarios. If both systems are acting optimally in this way, we want to determine the largest relative distance that may occur over time. This distance is the maximum possible tracking error between the two systems.

5.2.4.1 Relative Dynamics

To determine the relative distance that may occur over time we must first define the relative states and dynamics between the tracking and planning models. The individual dynamics are defined in Section 5.2.2, equations (5.14) and (5.17). The relative system is found by fixing the planning model to the origin and finding the dynamics of the tracking model relative to the planning model, as shown below.

$$r = s - Qp, \qquad \dot{r} = g(r, u_s, u_p, d) \tag{5.18}$$

where Q matches the common states of s and p by augmenting the state space of the planning model (as shown in Section 5.2.6). The relative states r now represent the tracking states relative to the planning states. Similarly, Q^T projects the state space of the tracking model onto the planning model: $p = Q^T(s - r)$. This will be used to update the planning model in the online algorithm.

5.2.4.2 Formalizing the Pursuit-Evasion Game

Now that we have the relative dynamics between the two systems we must define a metric for the tracking error bound between these systems. We do this by defining an implicit surface function as a cost function $l(r)$ in the new frame of reference. Because the metric we care about is distance to the origin (and thus distance to the planning system), this cost function can be as simple as distance in position space to the origin. An example can be seen in Fig. 5.11-a, where $l(r)$ is defined for a 4D quadrotor model tracking a 2D kinematic planning model. The contour rings beneath the function represent varying level sets of the cost function. The tracking system will try to minimize this cost to reduce the relative distance, while the planning system will do the opposite.

Before constructing the differential game we must first determine the method each player must use for making decisions. We define a strategy for planning system as the mapping $\gamma_p : \mathcal{U}_s \to \mathcal{U}_p$ that determines a control for the planning model based on the control of the planning model. We restrict γ to draw from only non-anticipative strategies $\gamma_p \in \Gamma_p(t)$, as defined in [111]. We similarly define the disturbance strategy $\gamma_d : \mathcal{U}_s \to \mathcal{D}$, $\gamma_d \in \Gamma_d(t)$.

We want to find the farthest distance (and thus highest cost) that this game will ever reach when both players are acting optimally. Therefore we want to find a mapping between the initial relative state of the system and the maximum possible cost achieved over the time horizon. This mapping is through our value function, defined as

$$\begin{aligned} V(r,T) = \sup_{\gamma_p \in \Gamma_p(t), \gamma_d \in \Gamma_d(t)} \inf_{u_s(\cdot) \in \mathbb{U}_s(t)} \Big\{ \\ \max_{t \in [0,T]} l\Big(\xi_g(t; r, 0, u_s(\cdot), \gamma_p[u_s](\cdot), \gamma_d[u_s](\cdot))\Big)\Big\} \end{aligned} \tag{5.19}$$

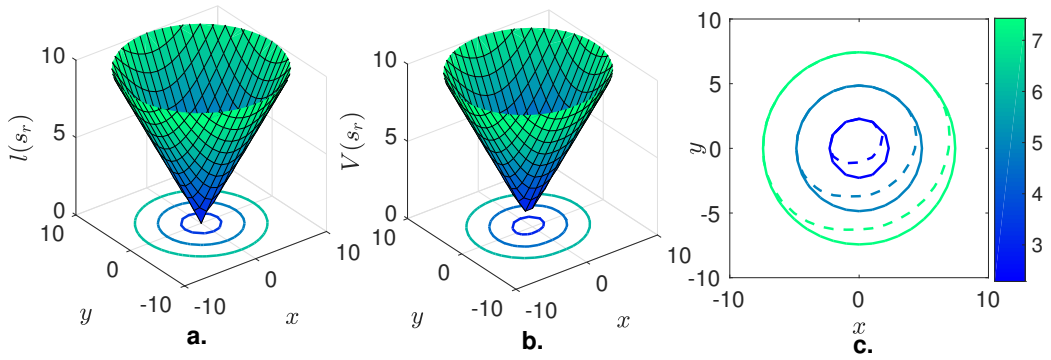


Figure 5.11: illustrative example of the precomputation steps for a 4D quadrotor model tracking a 2D kinematic planning model. All graphs are defined over a 2D slice of the 4D system. a) Cost function $l(r)$ defined on relative states as distance to the origin, b) Value function $V(r)$ computed using HJ reachability, c) Level sets of $l(r)$ (solid) and $V(r)$ (dashed). If the initial relative state is contained within the dashed set the system is guaranteed to remain within the corresponding solid set.

By implementing HJ reachability analysis we solve for this value function over the time horizon. If the control authority of the tracking system is powerful enough to always eventually reach the planning system, this value function will converge to an invariant solution for all time, i.e. $V_\infty(r) := \lim_{T\to\infty} V(r,T)$. An example of this converged value function is in Fig. 5.11-b. In the next section we will prove that the sub-level sets of this value function will map initial relative states to the guaranteed furthest possible tracking error over all time, as seen in Fig. 5.11-c.

In the context of the online framework, the value function $V_\infty(r)$ is the tracking error bound function. The spatial gradients of the value function, $\nabla V_\infty(r)$, comprise the safety controller function (as described in Section 5.2.5). When the framework is executed on a computer, these two functions are saved as look-up tables over a grid representing the state space of the relative system.

5.2.4.3 Invariance of Converged Value Function

Proposition 6 *Suppose that the value function converges, and define*

$$V_\infty(r) := \lim_{T\to\infty} V(r,T) \tag{5.20}$$

Then $\forall t_1, t_2$ *with* $t_2 \geq t_1$,

$$V_\infty(r) \geq V_\infty\Big(\xi_g^*(t_2; r, t_1)\Big), \text{ where} \tag{5.21}$$

$$\xi_g^*(t; r, 0) := \xi_g(t; r, 0, u_s^*(\cdot), u_p^*(\cdot), d^*(\cdot))) \tag{5.22}$$

$$\begin{aligned} u_s^*(\cdot) = \arg \inf_{u_s(\cdot)\in\mathbb{U}_s(t)} \Big\{ \\ \max_{t\in[0,T]} l(\xi_g(t; r, 0, u_s(\cdot), u_p^*(\cdot), d^*(\cdot)))\Big\} \end{aligned} \tag{5.23}$$

$$u_p^*(\cdot) := \gamma_p^*[u_s](\cdot) = \arg \sup_{\gamma_p \in \Gamma_p(t)} \inf_{u_s(\cdot) \in \mathbb{U}_s(t)} \Big\{ \max_{t \in [0,T]} l(\xi_g(t; r, 0, u_s(\cdot), \gamma_p[u_s](\cdot), d^*(\cdot))) \Big\} \tag{5.24}$$

$$d^*(\cdot) = \arg \sup_{\gamma_d \in \Gamma_d(t)} \sup_{\gamma_p \in \Gamma_p(t)} \inf_{u_s(\cdot) \in \mathbb{U}_s(t)} \Big\{ \max_{t \in [0,T]} l(\xi_g(t; r, 0, u_s(\cdot), \gamma_p[u_s](\cdot), \gamma_d[u_s](\cdot))) \Big\} \tag{5.25}$$

Proposition 6 proves that every level set of $V_\infty(r)$ is invariant under the following conditions:

1. The tracking system applies the optimal control which tries to track the planning system;
2. The planning system applies (at worst) the optimal control that tries to escape from the tracking system;
3. The tracking system experiences (at worst) the optimal disturbance that tries to prevent successful tracking.

In practice, conditions 2 and 3 may not hold; the result of this is only advantageous to the tracking system and will make it easier to stay within its current level set of $V_\infty(r)$, or to move to a smaller invariant level set of $V_\infty(r)$. The smallest invariant level set corresponding to the value $\underline{V} := \min_r V_\infty(r)$ can be interpreted as the smallest possible tracking error of the system. The tracking error bound is given by[2] the set $\mathcal{B} = \{r : V_\infty(r) \leq \underline{V}\}$. This tracking error bound in the planner's frame of reference is given by:

$$\mathcal{B}_p(s) = \{p : V_\infty(s - Qp) \leq \underline{V}\} \tag{5.26}$$

This is the tracking error bound that will be used in the online framework as shown in Fig. 5.8. Within this bound the tracking system may use any controller, but on the border of this bound the tracking system must use the safety optimal controller. We now prove Proposition 6.

Proof: Without loss of generality, assume $t_1 = 0$. By definition, we have

$$V_\infty(r) = \lim_{T \to \infty} \max_{t \in [0,T]} l(\xi_g^*(t; r, 0)) \tag{5.27}$$

By time-invariance, for some $t_2 > 0$,

$$\begin{aligned} V_\infty(r) &= \lim_{T \to \infty} \max_{t \in [-t_2, T]} l(\xi_g^*(t; r, -t_2)) \\ &\geq \lim_{T \to \infty} \max_{t \in [0,T]} l(\xi_g^*(t; r, -t_2)) \end{aligned} \tag{5.28}$$

[2]In practice, since V_∞ is obtained numerically, we set $\mathcal{B} = \{r : V_\infty(r) \leq \underline{V} + \epsilon\}$ for some suitably small $\epsilon > 0$

where the sub-interval $[-t_2, 0)$ has been removed in the last line. Next, by time invariance again, we have

$$\begin{aligned} \xi_g^*(t; r, -\tau) &= \xi_g^*(t; \xi_g^*(0; r, -t_2), 0) \\ &= \xi_g^*(t; \xi_g^*(t_2; r, 0), 0) \end{aligned} \tag{5.29}$$

Now, (5.28) implies

$$\begin{aligned} V_\infty(r) &\geq \lim_{T\to\infty} \max_{t\in[0,T]} l(\xi_g^*(t; \xi_g^*(t_2; r, 0), 0)) \\ &= V_\infty(\xi_g^*(t_2; r, 0)) \end{aligned} \tag{5.30}$$

■

Remark 8 *Proposition 6 is very similar to well-known results in differential game theory with a slightly different cost function [4], and has been utilized in the context of using the subzero level set of V_∞ as a backward reachable set for tasks such as collision avoidance or reach-avoid games [111]. In our work we do not assign special meaning to any particular level set, and instead consider all level sets at the same time. This effectively allows us to perform solve many simultaneous reachability problems in a single computation, thereby removing the need to check whether resulting invariant sets are empty, as was done in [14].*

5.2.5 Online Computation

Algorithm 5 describes the online computation. The inputs are the tracking error function $V_\infty(r)$ and the safety control look-up function $\nabla V_\infty(r)$. Note that when discretized on a computer these functions will be look-up tables; practical issues arising from sampled data control can be handled using methods such as [47,112,113] and are not the focus of this section.

Lines 1-3 initialize the computation by setting the planning and tracking model states (and therefore the relative state) to zero. The tracking error bound in the planning frame of reference is computed using (5.26). Note that by initializing the relative state to be zero we can use the smallest possible invariant $\mathcal{B}_p$ for the entire online computation. The tracking error bound block is shown on lines 5-6. The sensor detects obstacles $\mathcal{O}_{sense}$ within the sensing distance around the vehicle. The sensed obstacles are augmented by $\mathcal{B}_p(0)$ using the Minkowski sum. This is done to ensure that no unsafe path can be generated[3].

The path planner block (lines 7-8) takes in the planning model state p and the augmented obstacles $\mathcal{O}_{aug}$, and outputs the next state of the planning system p_{next}. The hybrid tracking controller block (lines 9-16) first computes the updated relative

[3]The minimum allowable sensing distance is $m = 2\mathcal{B}_p(0) + \Delta x$, where Δx is the largest step in space that the planner can make in one time step.

Algorithm 5: Online Trajectory Planning

Initialization:
$p = s = r = 0$
$\mathcal{B}_p(0) = \{p : V_\infty(0) \leq \underline{V}\}$
while planning goal is not reached **do**
  Tracking Error Bound Block:
  $\mathcal{O}_{aug} \leftarrow \mathcal{O}_{sense} + \mathcal{B}_p(0)$
  Path Planner Block:
  $p_{next} \leftarrow j(p, \mathcal{O}_{aug})$
  Hybrid Tracking Controller Block:
  $r_{next} = s - Qp_{next}$
  if r_{next} is on boundary $\mathcal{B}_p(0)$ **then**
    use safety controller: $u_s \leftarrow u_s^*$ in (5.31)
  else
    use performance controller:
    $u_s \leftarrow$ desired controller
  end if
  Tracking Model Block:
  apply control u_s to vehicle for a time step of Δt, save next state as s_{next}
  Planning Model Block:
  $p = Q^T s_{next}$
  check if p is at planning goal
  reset states $s = s_{next}, r = 0$
end while

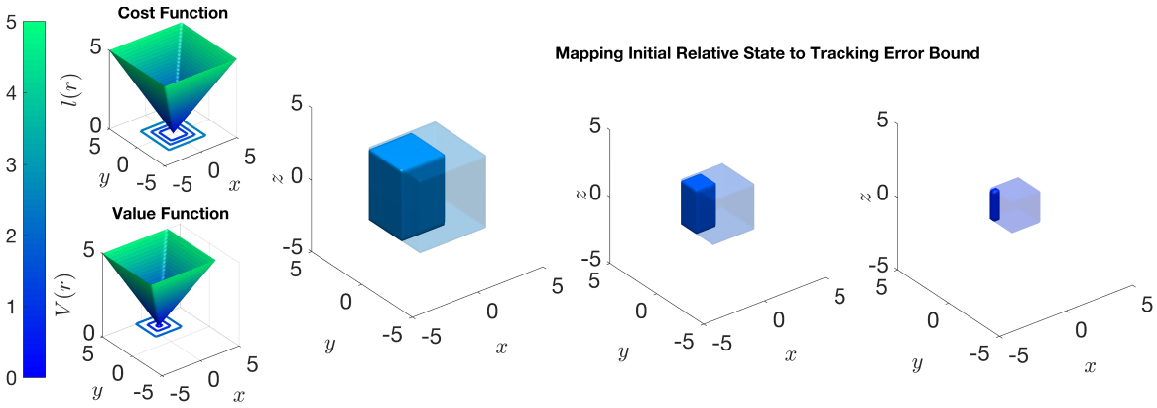


Figure 5.12: On the left are the cost and value functions over a 2D slice of the 10D relative state space, with contour lines showing three level sets of these functions. On the right are 3D projections of these level sets at the same slice $(v_x, v_y, v_z) = [1, -1, 1]$ m/s, $(\theta_x, \omega_x, \theta_y, \omega_y) = 0$. The solid boxes show initial relative states, and the transparent boxes show the corresponding tracking error bound. In practice we set the initial relative states to 0 to find the smallest invariant tracking error bound.

state r_{next}. If the r_{next} is on the tracking bound $\mathcal{B}_p(0)$, the safety controller must be used to remain within the safe bound. The safety control is given by:

$$u_s^* = \arg \min_{u_s \in \mathcal{U}_s} \max_{u_p \in \mathcal{U}_p, d \in \mathcal{D}} \nabla V(r_{next}) \cdot g(r_{next}, u_s, u_p, d) \tag{5.31}$$

For many practical systems (such as control affine systems), this minimization can be found extremely quickly.

If the relative state is not on the tracking boundary, a performance controller may be used. For the example in Section 5.2.6 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 17-18) for a time period of Δt. The next state is saved as s_{next}. This then updates the planning model state in the planning model block (lines 19-22). We repeat this process until the planning goal has been reached.

5.2.6 10D Quadrotor RRT Example

We demonstrate this framework with a 10D near-hover quadrotor developed in [25] tracking a 3D point source path generated by an RRT planner [94], [88]. First we perform the offline computations to acquire the tracking error bound and safety controller look-up tables. Next we set up the RRT to convert paths to simple 3D trajectories. Finally we implement the online framework to navigate the 10D system through a 3D environment with static obstacles.

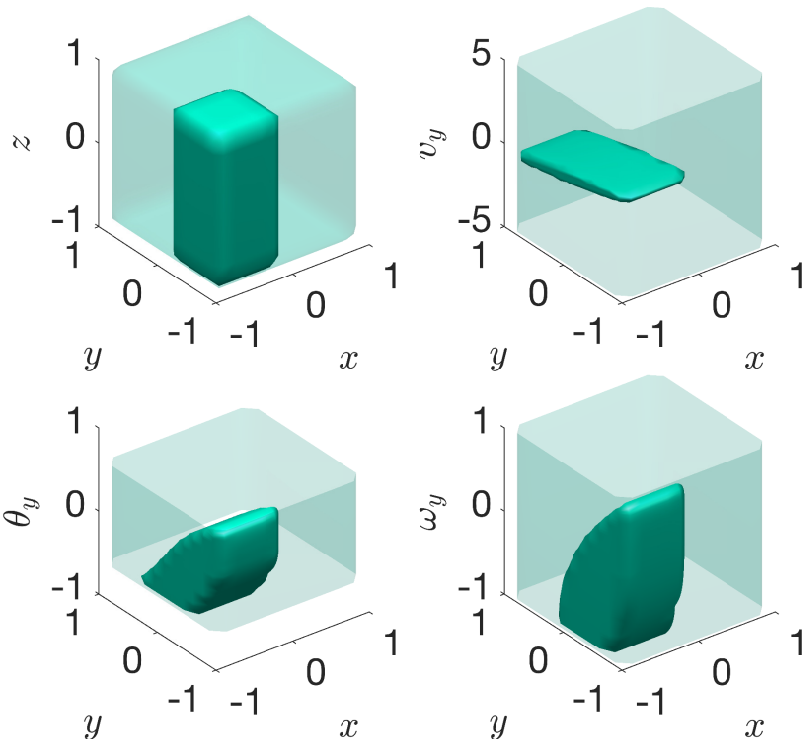


Figure 5.13: Various 3D slices of the 10D relative states (solid) and the corresponding tracking error bound (transparent)

5.2.6.1 Precomputation of 10D-3D system

First we use the 10D dynamics of the tracking quadrotors from (4.58), and define the 3D dynamics of a holonomic vehicle:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix} \tag{5.32}$$

where states (x, y, z) denote the position, (v_x, v_y, v_z) denote the velocity, (θ_x, θ_y) denote the pitch and roll, and (ω_x, ω_y) denote the pitch and roll rates. The controls of the 10D system are (a_x, a_y, a_z), where a_x and a_y represent the desired pitch and roll angle, and a_z represents the vertical thrust. The 3D system controls are (b_x, b_y, b_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. Note that the states of the 3D dynamics are a subset of the 10D state space; the matrix Q used in the online computation matches the position states of both systems. Next the relative dynamics between the two systems is defined using (5.18):

$$\begin{bmatrix} \dot{x_r} \\ \dot{v_x} \\ \dot{\theta_x} \\ \dot{\omega_x} \\ \dot{y_r} \\ \dot{v_y} \\ \dot{\theta_y} \\ \dot{\omega_y} \\ \dot{z_r} \\ \dot{v_z} \end{bmatrix} = \begin{bmatrix} v_x - b_x + d_x \\ g \tan \theta_x \\ -d_1 \theta_x + \omega_x \\ -d_0 \theta_x + n_0 a_x \\ v_y - b_y + d_y \\ g \tan \theta_y \\ -d_1 \theta_y + \omega_y \\ -d_0 \theta_y + n_0 a_y \\ v_z - b_z + d_z \\ k_T a_z - g \end{bmatrix} \tag{5.33}$$

The values for parameters d_0, d_1, n_0, k_T, g used were: $d_0 = 10, d_1 = 8, n_0 = 10, k_T = 0.91, g = 9.81$. The 10D control bounds were $|a_x|, |a_y| \leq 10$ degrees, $0 \leq a_z \leq 1.5g$ m/s^2. The 3D control bounds were $|b_x|, |b_y|, |b_z| \leq 0.5$ m/s. The disturbance bounds were $|d_x|, |d_y|, |d_z| \leq 0.1$ m/s.

Next we follow the setup in section 5.2.4 to create a cost function, which we then evaluate using HJ reachability until convergence to produce the invariant value function as in (5.19). Historically this 10D nonlinear relative system would be intractable for HJ reachability analysis, but using the decomposition method in Section 4.1 we can decompose this system into 3 subsystems (for each positional dimension). Doing this also requires decomposing the cost function; therefore we represent the cost function as a 1-norm instead of a 2-norm. This cost function as well as the resulting value function can be seen projected onto the x, y dimensions in Fig. 5.12.

Fig. 5.12 also shows 3D positional projections of the mapping between initial relative state to maximum potential relative distance over all time (i.e. tracking error

bound). If the real system starts exactly at the origin in relative coordinates, its tracking error bound will be a box of $\underline{V} = 0.81$ m in each direction. Slices of the 3D set and corresponding tracking error bounds are also shown in Fig. 5.13. We save the look-up tables of the value function (i.e. the tracking error function) and its spatial gradients (i.e. the safety controller function).

5.2.6.2 Online Planning with RRT and Sensing

Our precomputed value function can serve as a tracking error bound, and its gradients form a look-up table for the optimal tracking controller. These can be combined with any planning algorithm such as MPC, RRT, or neural-network-based planners in a modular way.

To demonstrate the combination of fast planning and provably robust tracking, we used a simple multi-tree RRT planner implemented in MATLAB modified from [67]. We assigned a speed of 0.5 m/s to the piecewise linear paths obtained from the RRT planner, so that the planning model is as given in (4.58). Besides planning a path to the goal, the quadrotor must also sense obstacles in the vicinity. For illustration, we chose a simple virtual sensor that reveals obstacles within a range of 2 m in the x, y, or z directions.

Once an obstacle is sensed, the RRT planner replans while taking into account all obstacles that have been sensed so far. To ensure that the quadrotor does not collide with the obstacles despite error in tracking, planning is done with respect to augmented obstacles that are "expanded" from the sensed obstacles by $\underline{V}$ in the x, y, and z directions.

On an unoptimized MATLAB implementation on a desktop computer with a Core i7-2600K CPU, each iteration took approximately 25 ms on average. Most of this time is spent on planning: obtaining the tracking controller took approximately 5 ms per iteration on average. The frequency of control was once every 100 ms.

Fig. 5.14 shows the simulation results. Four time snapshots are shown. The initial position is $(-12, 0, 0)$, and the goal position is $(12, 0, 0)$. The top left subplot shows the entire trajectory from beginning to end. In all plots, a magenta star represents the position of the planning model; its movement is based on the paths planned by RRT, and is modeled by a 3D holonomic vehicle with a maximum speed. The blue box around the magenta star represents the tracking error bound. The position of the tracking model is shown in blue. Throughout the simulation, the tracking model's position is always inside the tracking error, in agreement with Proposition 6. In addition, the tracking error bound never intersects with the obstacles, a consequence of the RRT planner planning with respect to a set of augmented obstacles (not shown). In the latter two subplots, one can see that the quadrotor appears to be exploring the environment briefly before reaching the goal. We did not employ any exploration algorithm; this exploration behavior is simply emerging from replanning using RRT whenever a new part (a 3 m^2 portion) of an obstacle is sensed.

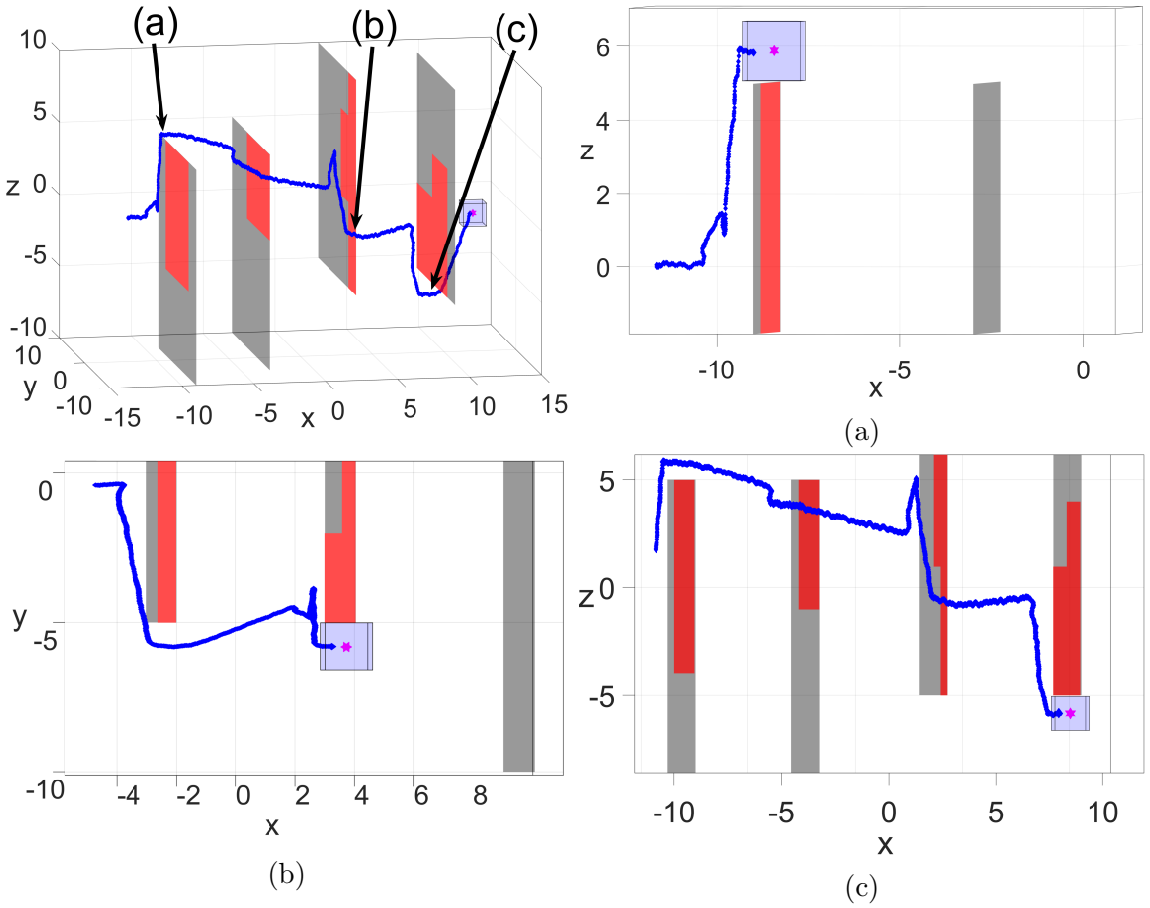


Figure 5.14: Numerical simulation. The tracking model trajectory is shown in blue, the planning model position in magenta, unseen obstacles in gray, and seen obstacles in red. The translucent blue box represents the tracking error bound. The top left subplot shows the entire trajectory; the other subplots zoom in on the positions marked in the top left subplot. The camera angle is also adjusted to illustrate our theoretical guarantees on tracking error and robustness in planning. A video of this simulation can be found at https://youtu.be/ZVvyeK-a62E

5.2.7 Conclusions and Future work

In this section we introduced our new tool FaSTrack: Fast and Safe Tracking. This tool can be used to add robustness to various path and trajectory planners without sacrificing fast online computation. So far this tool can be applied to unknown environments with a limited sensing range and static obstacles. We are excited to explore several future directions for FaSTrack in the near future, including exploring robustness for moving obstacles, adaptable error bounds based on external disturbances, and demonstration on a variety of planners.

Chapter 6

Conclusions and Future Work

Autonomous systems research has been tremendously successful recently, and now the perspective of safety is becoming very important, despite the difficulties of safety analysis. With the recent progress in high-dimensional verification, I think we are have made great initial progress in the path towards more pervasive and verified automation. If large-scale safety analysis could be combined with previous successes in the field in a modular way, we could have safe system design, planning, sensing, and learning, safe large-scale autonomous systems, and safe human-automation interaction in the near future.

There are many exciting areas of future research in verified automation, including combining machine learning and formal verification, modular combination of verification and planning, more sophisticated system decomposition, multi-agent coordination, and hardware validation. All of these are steps we can take towards more pervasive automation. Below, a few high level directions are provided.

Incorporating machine learning in verification: The preliminary work involving neural networks in Section 5.1 falls under the category of using machine learning techniques to augment verification techniques. This was an attempt at using machine learning to overcome computation burden in formal verification.

Incorporating verification in machine learning: As a start, verification techniques such as reachability can be applied to learned systems and learned controllers. This is not currently easy to do, and to address this difficulty, for safety-critical application we should also try to learn *verifiable systems and controllers*. The robotics community is now applying machine learning to identify many different system models as well as construct a variety of controllers. Currently, perhaps only a small portion of these are verifiable with current verification tools. For many applications, this may be good enough, but for safety-critical systems, machine learning should be done with verification in mind. For example, we could identify representations of system dynamics that are compatible with verification tools. Given current limitations in

verification, the identified system or controller would have to be low-dimensional, or have some decomposable structure.

More sophisticated verification techniques: The current popularity of machine learning motivates development of verification tools that are better suited in the context of learned systems. For example, having the ability to verify systems with partially known dynamics, having ways to analyze safety even when there is only partial state information, and developing more methods for high-dimensional systems would facilitate verification learned systems.

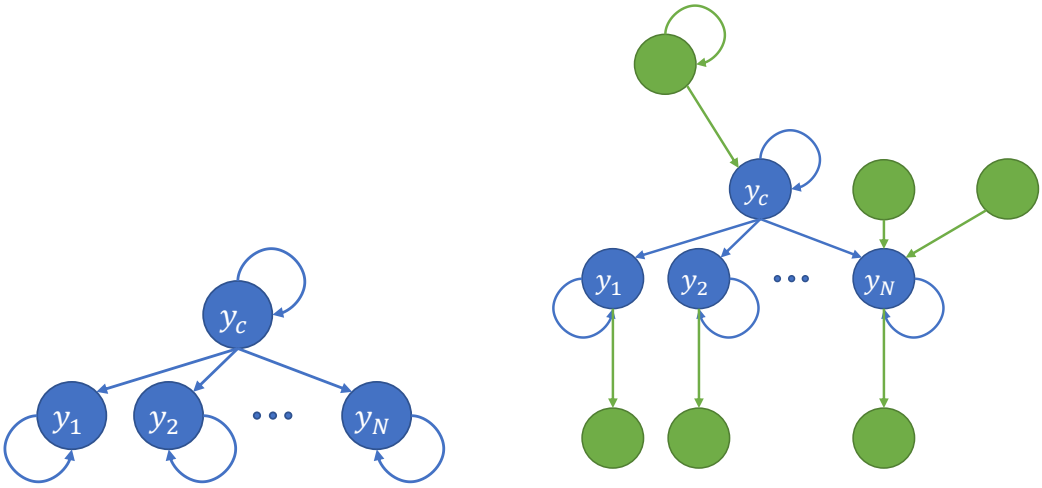
Parallelizing verification computations: Utilizing more computing power would also help verification tools become more relevant in analyzing learned systems, especially when the computing power is combined with high-dimensional techniques such as decomposition. Massively parallel computing and cloud computing are already widely used in machine learning, and verification methods can also take advantage of these resources. A first implementation of the HJ reachability method using GPU parallelization has already achieved a 80 times improvement in computation time.

Safe learning: A slightly different way in which formal verification would be very important is in safe learning. Often, practical systems require real-world data. For safety-critical systems, this data collection process must be done in a provably safe way.

More general system decomposition: This thesis provided two examples of system decomposition. In Section 4.2, a system component is linked unidirectionally to several others components, and all of these components may have self-links, as shown in Figure 6.1a. This simple structure has already enabled substantial dimensionality reduction. Section 4.2 provides a method to remove any edge at the expense of conservatism. With a graphical representation, one can easily imagine many other possible links, such as parent nodes, children nodes, or multiple parent nodes, shown in Figure 6.1b. The particular structures that we should investigate would be motivated by the structure of real systems, and perhaps those arising from other domains such as machine learning.

Decomposition in other contexts: The concept of decomposition is not just useful in reachability or verification, but would also be very beneficial in many other contexts. For example, decomposition methods would be useful in the contexts of partially observable Markov decision processes (POMDPs) and reinforcement learning, and the idea of decomposition is quite closely related to distributed and parallel computing.

Automatic system structure detection and decomposition: As we analyze



(a) Structure compatible with the decomposition technique in Section 4.1.

(b) A more complex system structure.

Figure 6.1: Graphical representations of dynamical systems.

systems that are more and more complex, we would have more complex graphical abstractions of dynamical systems. Eventually, we would need to develop methods for automatically decomposing a very complex system. Having more powerful decomposition tools would facilitate combining machine learning and verification on these systems.

Applications in unmanned airspace infrastructure: One possible application of both machine learning and verification is in unmanned airspace traffic management. This is a rapidly growing area, and multiple levels of abstractions of the airspace will likely be needed. For example, cities and larger regions could have separate traffic rules, and these different levels of abstractions would need to satisfy contracts between them.

Practical sensing and perception: On a smaller level, there are many practical considerations. For example, often algorithms designed in control theory are implemented on systems using motion capture systems to provide state information; however, this is not realistic for large-scale utilization. Sensors such as LIDAR and systems such as global positioning system (GPS) and computer vision systems would need to be used for navigation. Other practical challenges include delays, frequency of data, and even missing data.

Human-robot interactions: Humans will undoubtedly be participating in the control of autonomous or semi-autonomous systems. In this area, safety analysis would

be very useful for providing the right feedback, for example in the form of visual feedback or haptics, to guide humans away from danger. To this end, a modular combination of verification and planning such as FaSTrack, discussed in Section 5.2 is very promising.

Hardware implementation: All of the theory and algorithms that we develop can now be, and should be validated on many types of hardware platforms. This has never been easier before, with many different hardware platforms available from many different companies. For example, there is a variety of UAVs from companies such as Bitcraze, which offers the crazyflie, DJI, and Parrot, which sell larger quadrotors which may have different computation capabilities and sensors. Robotic arms are also now becoming more accessible, and would be especially useful for research in human-robot interactions. Robotic surgeries is now becoming a popular concept. Of course, depending on the application area there are other hardware platforms such as ground robotics and miniature autonomous cars. In fact, practically speaking, different types of autonomous systems will likely be interacting with each other.

Bibliography

[1] 3D Robotics. Solo Specs: Just the facts, 2015. https://news.3dr.com/solo-specs-just-the-facts-14480cb55722#.w7057q926 [retrieved May 9, 2017].

[2] Avishai Adler, Aharon Bar-Gill, and Nahum Shimkin. Optimal flight paths for engine-out emergency landing. In *Proceedings of the IEEE Chinese Control and Decision Conference*, pages 2908–2915, 2012. doi: 10.1109/CCDC.2012.6244461.

[3] A. Ahmadzadeh, N. Motee, A. Jadbabaie, and G. Pappas. Multi-vehicle path planning in dynamically changing environments. In *International Conference on Robotics and Automation*, pages 2449–2454, May 2009.

[4] Anayo K. Akametalu, Jaime F. Fisac, Jeremy H. Gillula, Shahab Kaynama, Melaine N. Zeilinger, and Claire J. Tomlin. Reachability-based safe learning with Gaussian processes. In *Proceedings of the IEEE Conference on Decision and Control*, Dec. 2014.

[5] Ross E Allen, Ashley A Clark, Joseph A Starek, and Marco Pavone. A machine learning approach for real-time reachability analysis. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2202–2208, Sep. 2014.

[6] M. Althoff. An Introduction to CORA 2015. In *Proceedings of the Workshop on Applied Verification for Continuous and Hybrid Systems*, 2015.

[7] Matthias Althoff and John M. Dolan. Set-based computation of vehicle behaviors for the online verification of autonomous vehicles. In *International Conference on Intelligent Transportation Systems*, pages 1162–1167, Oct. 2011.

[8] Matthias Althoff and Bruce H. Krogh. Reachability Analysis of Nonlinear Differential-Algebraic Systems. *IEEE Transactions Automatic Control*, 59(2):371–383, Feb. 2014.

[9] K. Alton and I.M. Mitchell. Optimal path planning under defferent norms in continuous state spaces. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 866–872. doi: 10.1109/ROBOT.2006.1641818.

[10] Amazon.com, Inc. Amazon prime air, 2016.

[11] Aaron D Ames, Jessy W Grizzle, and Paulo Tabuada. Control barrier function based quadratic programs with application to adaptive cruise control. In *Proceedings of the IEEE Conference on Decision and Control*, pages 6271–6278, 2014.

[12] Maria Soledad Aronna, J. Frédéric Bonnans, and Pierre Martinon. A shooting algorithm for optimal control problems with singular arcs. *Journal of Optimization Theory and Applications*, 158(2):419–459, 2013.

[13] AUVSI News. UAS Aid in South Carolina Tornado Investigation. `http://www.auvsi.org/blogs/auvsi-news/2016/01/29/tornado`, 2016.

[14] Somil Bansal, Mo Chen, Jaime F. Fisac, and Claire J. Tomlin. Safe Sequential Path Planning of Multi-Vehicle Systems Under Presence of Disturbances and Imperfect Information. *Proceedings of the AACC American Control Conference*, 2017.

[15] Somil Bansal*, Mo Chen*, and Claire J. Tomlin. Safe sequential path planning of multi-vehicle systems under presence of disturbances and measurement noise. In *Proceedings of the AACC American Control Conference*, 2017.

[16] E. N. Barron. Differential games with maximum cost. *Nonlinear Analysis*, 14(11):971–989, Jun. 1990.

[17] E.N. Barron and H. Ishii. The Bellman equation for minimizing the maximum cost. *Nonlinear Analysis: Theory, Methods & Applications*, 13(9):1067–1090, Sept. 1989. doi: 10.1016/0362-546X(89)90096-5.

[18] Alexandre M. Bayen, Ian M. Mitchell, Meeko Oishi, and Claire J. Tomlin. Aircraft autolander safety analysis through optimal control-based reach set computation. *AIAA Journal of Guidance, Control, and Dynamics*, 30(1), 2007.

[19] BBC News. Google plans drone delivery service for 2017. `http://www.bbc.co.uk/news/technology-34704868`, 2015.

[20] R.W. Beard and TW McLain. Multiple UAV cooperative search under collision avoidance and limited range communication constraints. In *Proceedings of the Conference on Decision and Control*, volume 1, pages 25–30, 2003.

[21] Yaar Becerikli, Ahmet Ferit Konar, and Tarq Samad. Intelligent optimal control with dynamic neural networks. *Neural Networks*, 16(2):251–259, Mar. 2003.

[22] J.S. Bellingham, M Tillerson, M Alighanbari, and J.P. How. Cooperative path planning for multiple UAVs in dynamic and uncertain environments. In *Proceedings of the Conference on Decision and Control*, volume 3, pages 2816–2822, Dec. 2002.

[23] Olivier Bokanowski, Nicolas Forcadel, and Hasnaa Zidani. Reachability and Minimal Times for State Constrained Nonlinear Problems without Any Controllability Assumption. *SIAM Journal on Control and Optimization*, 48(7):4292–4316, Jan. 2010. doi: 10.1137/090762075.

[24] Olivier Bokanowski and Hasnaa Zidani. MINIMAL TIME PROBLEMS WITH MOVING TARGETS AND OBSTACLES. *IFAC Proceedings Volumes*, 44(1):2589–2593, Jan. 2011.

[25] Patrick Bouffard. On-board model predictive control of a quadrotor helicopter: Design, implementation, and experiments. Master's thesis, University of California, Berkeley, 2012.

[26] Frank M. Callier and Charles A. Desoer. The System Representation R=[A,B,C,D], Part II. In *Linear System Theory*, pages 103–139. Springer, 1991.

[27] G.C. Chasparis and J.S. Shamma. Linear-programming-based multi-vehicle path planning with adversaries. In *Proceedings of the AACC American Control Conference*, pages 1072–1077, Jun. 2005.

[28] Mo Chen, Somil Bansal, Jaime F Fisac, and Claire J Tomlin. Robust sequential path planning under disturbances and adversarial intruder. *arXiv preprint arXiv:1611.08364*, 2016.

[29] Mo Chen, Somil Bansal, Ken Tanabe, and Claire J. Tomlin. Provably Safe and Robust Drone Routing via Sequential Path Planning: A Case Study in San Francisco and the Bay Area. May 2017.

[30] Mo Chen, Jaime F. Fisac, Shankar Sastry, and Claire J. Tomlin. Safe sequential path planning of multi-vehicle systems via double-obstacle Hamilton-Jacobi-Isaacs variational inequality. In *Proceedings of the European Control Conference*, pages 3304–3309, Jul. 2015.

[31] Mo Chen*, Sylvia Herbert*, and Claire J Tomlin. Fast Reachable Set Approximations via State Decoupling Disturbances. In *Proceedings of the IEEE Conference on Decision and Control*, 2016.

[32] Mo Chen, Sylvia Herbert, and Claire J Tomlin. Exact and Efficient Hamilton-Jacobi-based Guaranteed Safety Analysis via System Decomposition. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2017.

[33] Mo Chen, Sylvia L. Herbert, Mahesh S. Vashishtha, Somil Bansal, and Claire J. Tomlin. Decomposition of Reachable Sets and Tubes for a Class of Nonlinear Systems. *IEEE Transactions on Automatic Control (to appear)*, Nov. 2016.

[34] Mo Chen, Qie Hu, Jaime F. Fisac, Kene Akametalu, Casey Mackin, and Claire J. Tomlin. Reachability-Based Safety and Goal Satisfaction of Unmanned Aerial Platoons on Air Highways. *AIAA Journal of Guidance, Control, and Dynamics*, pages 1–14, Jan. 2017.

[35] Mo Chen, Qie Hu, Casey Mackin, Jaime Fisac, and Claire J. Tomlin. Safe platooning of unmanned aerial vehicles via reachability. In *Proceedings of the IEEE Conference on Decision and Control*, 2015.

[36] Mo Chen*, Jennifer Shih*, and Claire J. Tomlin. Multi-vehicle collision avoidance via reachability and mixed integer programming. In *Proceedings of the IEEE Conference on Decision and Control*, 2016.

[37] Mo Chen and Claire J. Tomlin. Exact and efficient Hamilton-Jacobi reachability for decoupled systems. In *Proceedings of the IEEE Conference on Decision and Control*, pages 1297–1303, 2015. doi: 10.1109/CDC.2015.7402390.

[38] Mo Chen, Zhengyuan Zhou, and Claire J. Tomlin. Multiplayer Reach-Avoid Games via Pairwise Outcomes. *IEEE Transactions on Automatic Control*, 62(3):1451–1457, Mar. 2017.

[39] Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. Flow*: An Analyzer for Non-linear Hybrid Systems. In *Proceedings of the Conference on Computer Aided Verification*, pages 258–263, 2013.

[40] Yao Li Chuang, Yuan R. Huang, Maria R. D'Orsogna, and Andrea L. Bertozzi. Multi-vehicle flocking: Scalability of cooperative control algorithms using pairwise potentials. In *International Conference on Robotics and Automation*, pages 2292–2299, Apr. 2007.

[41] Earl A. Coddington and Norman Levinson. *Theory of ordinary differential equations*. Tata McGraw-Hill Education, 1955.

[42] Samuel Coogan and Murat Arcak. Efficient finite abstraction of mixed monotone systems. In *Proceedings of the ACM International Conference Hybrid Systems: Computation and Control*, pages 58–67, 2015.

[43] Matthew Coombes, Wen-Hua Chen, and Peter Render. Reachability Analysis of Landing Sites for Forced Landing of a UAS. *Journal of Intelligent & Robotic Systems*, 73(1-4):635–653, Jan. 2014. doi: 10.1007/s10846-013-9920-9.

[44] Michael G. Crandall and Pierre-Louis Lions. Viscosity solutions of Hamilton-Jacobi equations. *Trans. Amer. Math. Soc.*, 277(1):1, Jan. 1983.

[45] Michael Grain Crandall, Lawrence Craig Evans, and Pierre-Louis Lions. Some properties of viscosity solutions of Hamilton-Jacobi equations. *Transactions of the American Mathematical Society*, 282(2):487–502, Apr. 1984. doi: S0002-9947-1984-0732102-X.

[46] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2(4):303–314, Dec. 1989.

[47] Charles Dabadie, Shahab Kaynama, and Claire J. Tomlin. A practical reachability-based collision avoidance algorithm for sampled-data systems: Application to ground robots. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, pages 4161–4168, Sept. 2014.

[48] Jérôme Darbon and Stanley Osher. Algorithms for overcoming the curse of dimensionality for certain HamiltonJacobi equations arising in control theory and elsewhere. *Research in the Mathematical Sciences*, 3(1):19, Dec. 2016.

[49] Wesley DeBusk. Unmanned Aerial Vehicle Systems for Disaster Relief: Tornado Alley. In *AIAA Infotech@Aerospace 2010*, Apr. 2010. AIAA paper number: 2010-3506, 10.2514/6.2010-3506.

[50] Debadeepta Dey, Kumar Shaurya Shankar, Sam Zeng, Rupesh Mehta, M Talha Agcayazi, Christopher Eriksen, Shreyansh Daftry, Martial Hebert, and J Andrew Bagnell. Vision and learning for deliberative monocular cluttered flight. In *Field and Service Robotics*, pages 391–409, 2016.

[51] Stefano Di Cairano and Francesco Borrelli. Reference tracking with guaranteed error bound for constrained linear systems. *IEEE Transactions on Automatic Control*, 61(8):2245–2250, 2016.

[52] Moritz Diehl, H Georg Bock, Johannes P Schlöder, Rolf Findeisen, Zoltan Nagy, and Frank Allgöwer. Real-time optimization and nonlinear model predictive control of processes governed by differential-algebraic equations. *Journal of Process Control*, 12(4):577–585, 2002.

[53] Moritz Diehl, Hans Joachim Ferreau, and Niels Haverbeke. Efficient numerical methods for nonlinear mpc and moving horizon estimation. In *Nonlinear model predictive control*, pages 391–417. 2009.

[54] Jerry Ding, Jonathan Sprinkle, S. Shankar Sastry, and Claire J. Tomlin. Reachability calculations for automated aerial refueling. In *Proceedings of the IEEE Conference on Decision and Control*, pages 3706–3712, 2008. doi: 10.1109/CDC.2008.4738998.

[55] Badis Djeridane and John Lygeros. Neural approximation of PDE solutions: An application to reachability computations. In *Proceedings of the IEEE Conference on Decision and Control*, pages 3034–3039, 2006.

[56] Tommaso Dreossi, Thao Dang, and Carla Piazza. Parallelotope Bundles for Polynomial Reachability. In *Proceedings of the ACM International Conference on Hybrid Systems: Computation and Control*, pages 297–306, 2016.

[57] L. E. Dubins. On Curves of Minimal Length with a Constraint on Average Curvature, and with Prescribed Initial and Terminal Positions and Tangents. *American Journal of Mathematics*, 79(3):497, Jul. 1957.

[58] Parasara Sridhar Duggirala, Sayan Mitra, Mahesh Viswanathan, and Matthew Potok. C2E2: A Verification Tool for Stateflow Models. In *Proc. Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, pages 68–82, 2015.

[59] Parasara Sridhar Duggirala, Matthew Potok, Sayan Mitra, and Mahesh Viswanathan. C2e2: A tool for verifying annotated hybrid systems. In *Proceedings of the ACM International Conference on Hybrid Systems: Computation and Control*, pages 307–308, 2015.

[60] L. C. Evans and P. E. Souganidis. Differential games and representation formulas for solutions of Hamilton-Jacobi-Isaacs equations. *Indiana University Mathematics Journal*, 33(5):773–797, 1984.

[61] Katie Fehrenbacher. Feds Say Safety Is the Key to the Future of Autonomous Cars. `http://fortune.com/2016/07/19/safety-feds-autonomous-cars/`.

[62] Feng-Li Lian and R. Murray. Real-time trajectory generation for the cooperative path planning of multi-vehicle systems. In *Conference on Decision and Control*, volume 4, pages 3766–3769, Dec. 2002.

[63] Paolo Fiorini and Zvi Shiller. Motion Planning in Dynamic Environments Using Velocity Obstacles. *The International Journal of Robotics Research*, 17(7):760–772, Jul. 1998.

[64] Jaime F. Fisac, Mo Chen, Claire J. Tomlin, and S. Shankar Sastry. Reach-avoid problems with time-varying dynamics, targets and constraints. In *Proceedings of the ACM International Conference Hybrid Systems: Computation and Control*, pages 11–20, 2015.

[65] Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. SpaceEx: Scalable Verification of Hybrid Systems. In *Proceedings of*

the International Conference on Computer Aided Verification, pages 379–395, 2011.

[66] Ather Gattami, Assad Al Alam, Karl Henrik Johansson, and Claire J Tomlin. Establishing Safety for Heavy Duty Vehicle Platooning: A Game Theoretical Approach. volume 44, pages 3818–3823, Jan. 2011. doi: 10.3182/20110828-6-IT-1002.02071.

[67] Gavin (Matlab community Contributor). Multiple Rapidly-exploring Random Tree (RRT), 2013.

[68] I. M. (Izrail Moiseevich) Gelfand, Richard A Silverman, and S. V. (Sergei Vasilevich) Fomin. *Calculus of variations*. Englewood Cliffs, N.J. : Prentice-Hall, rev. english ed. / translated and edited by richard a. silverman edition, 1963.

[69] Jeremy H Gillula, Gabriel M Hoffmann, Haomiao Huang, Michael P Vitus, and Claire J Tomlin. Applications of hybrid reachability analysis to robotic aerial vehicles. *International Journal of Robotics Research*, 30(3):335–354, Mar. 2011.

[70] Jeremy H Gillula, Haomiao Huang, Michael P Vitus, and Claire J Tomlin. Design of guaranteed safe maneuvers using reachable sets: Autonomous quadrotor aerobatics in theory and practice. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 1649–1654, 2010.

[71] Mark R. Greenstreet and Ian Mitchell. Integrating projections. In *Hybrid Systems: Computation and Control*, pages 159–174, 1998.

[72] Lars Grne and Jrgen Pannek. *Nonlinear Model Predictive Control: Theory and Algorithms*. Springer Publishing Company, Incorporated, 2013.

[73] Michael R. Hafner and Domitilla Del Vecchio. Computation of safety control for uncertain piecewise continuous systems on a partial order. In *Proceedings of the IEEE Conference on Decision and Control*, pages 1671–1677, 2009.

[74] Daniel L. Haulman. U.S. Unmanned Aerial Vehicles in Combat, 1991-2003. Technical report, Air Force Historical Research Agency, Maxwell Air Force Base, Alabama, 2003.

[75] K. J. Hedrick, G. Zhang, K. V. Narendran, K. s. Chang, Partners for Advanced Transit, Highways (Calif.), and Berkeley. Institute of Transportation Studies University of California. *Transitional Platoon Maneuvers in an Automated Highway System*. California PATH Program, Institute of Transportation Studies, University of California at Berkeley, 1992.

[76] Sylvia L. Herbert, Mo Chen, SooJean Han, Somil Bansal, Jaime F. Fisac, and Claire J. Tomlin. FaSTrack: a Modular Framework for Fast and Guaranteed Safe Motion Planning. In *Proceedings of the IEEE International Conference on Robotics and Automation*, May 2017.

[77] Gabriel M. Hoffmann and Claire J. Tomlin. Decentralized cooperative collision avoidance for acceleration constrained vehicles. In *Proceedings of the Conference on Decision and Control*, pages 4357–4363, 2008.

[78] Michael Hoy, Alexey S Matveev, and Andrey V Savkin. Algorithms for collision-free navigation of mobile robots in complex cluttered environments: a survey. *Robotica*, 33(03):463–497, 2015.

[79] Haomiao Huang, J. Ding, Wei Zhang, and C.J. Tomlin. A differential game approach to planning in adversarial scenarios: A case study on capture-the-flag. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1451–1456, 2011.

[80] Jinu Idicula, Kene Akametalu, Mo Chen, Claire Tomlin, Jerry Ding, and Loyd Hook. Where to Land: A Reachability Based Forced Landing Algorithm for Aircraft Engine Out Scenarios, Nov. 2015. Seedling Technical Seminar, NASA Dryden Flight Research Center, Report number DFRC-E-DAA-TN28194.

[81] Lucas Janson, Edward Schmerling, Ashley Clark, and Marco Pavone. Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions. *The International Journal of Robotics research*, 34(7):883–921, 2015.

[82] Frank Jiang, Glen Chou, Mo Chen, and Claire J. Tomlin. Using Neural Networks to Compute Approximate and Guaranteed Feasible Hamilton-Jacobi-Bellman PDE Solutions. Nov 2016.

[83] Joint Planning and Development Office. Unmanned Aircraft Systems (UAS) Comprehensive Plan: A Report on the Nation's UAS Path Forward. Technical report, 2013.

[84] Jointed Planning and Development Office (JPDO). Unmanned aircraft systems (UAS) comprehensive plan – a report on the nation's UAS path forward. Technical report, Federal Aviation Administration, 2013.

[85] Mrinal Kalakrishnan, Sachin Chitta, Evangelos Theodorou, Peter Pastor, and Stefan Schaal. Stomp: Stochastic trajectory optimization for motion planning. In *IEEE International Conference on Robotics and Automation*, pages 4569–4574, 2011.

[86] Sertac Karaman, Matthew R Walter, Alejandro Perez, Emilio Frazzoli, and Seth Teller. Anytime motion planning using the rrt. In *Proceedings of the IEEE International Conference onRobotics and Automation*, pages 1478–1483, 2011.

[87] Pooja Kavathekar and YangQuan Chen. Vehicle Platooning: A Brief Survey and Categorization. In *Proceedings of the ASME/IEEE International Conference on Mechatronic and Embedded Systems and Applications, Parts A and B*, volume 3, pages 829–845, 2011. doi: 10.1115/DETC2011-47861.

[88] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[89] Shahab Kaynama and Meeko Oishi. Schur-based decomposition for reachability analysis of linear time-invariant systems. In *Proceedings of the IEEE Conference on Decision and Control*, pages 69–74, Dec. 2009.

[90] Shahab Kaynama and Meeko Oishi. A Modified Riccati Transformation for Decentralized Computation of the Viability Kernel Under LTI Dynamics. *IEEE Transactions on Automatic Control*, 58(11):2878–2892, Nov. 2013.

[91] Byoung S. Kim and Anthony J. Calise. Nonlinear Flight Control Using Neural Networks. *AIAA Journal of Guidance, Control, and Dynamics*, 20(1):26–33, Jan. 1997.

[92] Marin Kobilarov. Cross-entropy motion planning. *The International Journal of Robotics Research*, 31(7):855–871, 2012.

[93] Soonho Kong, Sicun Gao, Wei Chen, and Edmund Clarke. dReach: δ-Reachability Analysis for Hybrid Systems. In *Proceedings of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 200–205, 2015.

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

[95] A.B. Kurzhanski and P. Varaiya. Ellipsoidal techniques for reachability analysis: internal approximation. *Syst. & Contr. Lett.*, 41(3):201–211, Oct. 2000.

[96] A.B. Kurzhanski and P. Varaiya. On Ellipsoidal Techniques for Reachability Analysis. Part II: Internal Approximations Box-valued Constraints. *Optimization Methods and Software*, 17(2):207–237, Jan. 2002.

[97] Emmett Lalish, Kristi A. Morgansen, and Takashi Tsukamaki. Decentralized reactive collision avoidance for multiple unicycle-type vehicles. In *Proceedings of the AACC American Control Conference*, pages 5055–5061, Jun. 2008.

[98] Jean B. Lasserre, Didier Henrion, Christophe Prieur, and Emmanuel Trélat. Nonlinear optimal control via occupation measures and lmi-relaxations. *SIAM Journal Control and Optimization*, 47(4):1643–1666, June 2008.

[99] Yucong Lin and Srikanth Saripalli. Collision avoidance for UAVs using reachable sets. In *International Conference on Unmanned Aircraft Systems*, pages 226–235, Jun. 2015.

[100] John Lygeros, Datta N. Godbole, and Shankar S. Sastry. Verified hybrid controllers for automated vehicles. *IEEE Transactions on Automatic Control*, 43(4):522–539, Apr. 1998.

[101] John Lygeros, Shankar Sastry, and Claire Tomlin. *Hybrid Systems: Foundations, advanced topics and applications*. Springer Verlag, 2012.

[102] John N. Maidens, Shahab Kaynama, Ian M. Mitchell, Meeko M K Oishi, and Guy A. Dumont. Lagrangian methods for approximating the viability kernel in high-dimensional systems. *Automatica*, 49(7):2017–2029, Jul. 2013.

[103] Anirudha Majumdar and Russ Tedrake. Funnel libraries for real-time robust feedback motion planning. *The International Journal of Robotics Research*, Jun. 2017.

[104] Anirudha Majumdar, Ram Vasudevan, Mark M Tobenkin, and Russ Tedrake. Convex optimization of nonlinear feedback controllers via occupation measures. 33(9):1209–1230, Aug. 2014.

[105] Kostas Margellos and John Lygeros. HamiltonJacobi Formulation for ReachAvoid Differential Games. *Transactions on Automatic Control*, 56(8):1849–1861, Aug. 2011.

[106] M Massink and N. De Francesco. Modelling free flight with collision avoidance. In *Proceedings of the International Conference on Engineering of Complex Computer Systems*, pages 270–279, 2001.

[107] J. S. McGrew, J. P. How, L. Bush, B. Williams, and N. Roy. Air combat strategy using approximate dynamic programming. *AIAA Guidance, Navigation, and Control Conference*, Aug. 2008.

[108] H. D. McMahon, K. J. Hedrick, and E. S. Shladover. Vehicle modelling and control for automated highway systems. In *American Control Conference, 1990*, pages 297–303, May 1990.

[109] Ian M. Mitchell. Comparing forward and backward reachability as tools for safety analysis. In *Proceedings of the ACM International Conference on Hybrid Systems: Computation and Control*, 2007.

[110] Ian M. Mitchell. The Flexible, Extensible and Efficient Toolbox of Level Set Methods. *Journal of Scientific Computing*, 35(2-3):300–329, Jun. 2008. doi: 10.1007/s10915-007-9174-4.

[111] Ian M. Mitchell, Alexandre M. Bayen, and Claire J. Tomlin. A time-dependent Hamilton-Jacobi formulation of reachable sets for continuous dynamic games. *IEEE Transactions on Automatic Control*, 50(7):947–957, Jul. 2005.

[112] Ian M. Mitchell, Mo Chen, and Meeko Oishi. Ensuring safety of nonlinear sampled data systems through reachability. *IFAC Proceedings Volumes*, 45(9):108–114, 2012.

[113] Ian M. Mitchell, Shahab Kaynama, Mo Chen, and Meeko Oishi. Safety preserving control synthesis for sampled data systems. *Nonlinear Analysis: Hybrid Systems*, 10(1):63–82, Nov. 2013.

[114] Ian M. Mitchell and Claire J. Tomlin. Overapproximating reachable sets by hamilton-jacobi projections. *Journal of Scientific Computing*, 19(1-3):323–346, 2003.

[115] I.M. Mitchell. Scalable calculation of reach sets and tubes for nonlinear systems with terminal integrators: a mixed implicit explicit formulation. In *Proceedings of the ACM International Conference Hybrid Systems: Computation and Control*, pages 103–112, 2011.

[116] National Aeronautics and Space Administration. Challenge is on to design sky for all, 2016. http://www.nasa.gov/feature/challenge-is-on-to-design-sky-for-all, [retrieved Feb. 12, 2016].

[117] Michael Neunert, Cédric de Crousaz, Fadri Furrer, Mina Kamel, Farbod Farshidian, Roland Siegwart, and Jonas Buchli. Fast nonlinear model predictive control for unified trajectory optimization and tracking. In *Proceedings of the International Conference on Robotics and Automation*, pages 1398–1404, 2016.

[118] New Atlas. Amazon Prime Air. http://newatlas.com/amazon-new-delivery-drones-us-faa-approval/36957/ [retrieved May 9, 2017].

[119] K. N. Niarchos and J. Lygeros. A Neural Approximation to Continuous Time Reachability Computations. In *Proceedings of the IEEE Conference on Decision and Control*, pages 6313–6318, 2006.

[120] Petter Nilsson and Necmiye Ozay. Synthesis of separable controlled invariant sets for modular local control design. In *Proceedings of the AACC American Control Conference*, pages 5656–5663, Jul. 2016.

[121] Reza Olfati-Saber and Richard M. Murray. DISTRIBUTED COOPERATIVE CONTROL OF MULTIPLE VEHICLE FORMATIONS USING STRUCTURAL POTENTIAL FUNCTIONS. *IFAC Proceedings Volumes*, 35(1):495–500, 2002.

[122] Stanley Osher and Ronald Fedkiw. *Level Set Methods and Dynamic Implicit Surfaces*. Springer-Verlag, 2002.

[123] Thomas Prevot, Joseph Rios, Parimal Kopardekar, John E. Robinson III, Marcus Johnson, and Jaewoo Jung. UAS Traffic Management (UTM) Concept of Operations to Safely Enable Low Altitude Flight Operations. In *Proceedings of the AIAA Aviation Technology, Integration, and Operations Conference*, Jun. 2016. AIAA paper number: 2016-3292, doi: 10.2514/6.2016-3292.

[124] S Joe Qin and Thomas A Badgwell. A survey of industrial model predictive control technology. *Control Engineering Practice*, 11(7):733–764, 2003.

[125] Nathan Ratliff, Matt Zucker, J Andrew Bagnell, and Siddhartha Srinivasa. Chomp: Gradient optimization techniques for efficient motion planning. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 489–494, 2009.

[126] Arthur Richards and Jonathan P How. Robust variable horizon model predictive control for vehicle maneuvering. *International Journal of Robust and Nonlinear Control*, 16(7):333–351, 2006.

[127] Charles Richter, Adam Bry, and Nicholas Roy. Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments. In *Robotics Research*, pages 649–666. 2016.

[128] Stefan Richter, Colin Neil Jones, and Manfred Morari. Computational complexity certification for real-time mpc with input constraints based on the fast gradient method. *IEEE Transactions on Automatic Control*, 57(6):1391–1403, 2012.

[129] Serban Sabau, Cristian Oara, Sean Warnick, and Ali Jadbabaie. Optimal Distributed Control for Platooning via Sparse Coprime Factorizations. *IEEE Transactions on Automatic Control*, 62(1):305–320, Jan. 2017.

[130] Shankar S. Sastry. Linearization by State Feedback. In *Nonlinear Systems: Analysis, Stability, and Control*, chapter 9, page 384. Springer-Verlag, 1999.

[131] Georg Schildbach and Francesco Borrelli. A dynamic programming approach for nonholonomic vehicle maneuvering in tight environments. In *Intelligent Vehicles Symposium (IV), 2016 IEEE*, pages 151–156, 2016.

[132] Tom Schouwenaars and Eric Feron. Decentralized Cooperative Trajectory Planning of Multiple Aircraft with Hard Safety Guarantees. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*, pages 2004–5141, Aug. 2004.

[133] John Schulman, Jonathan Ho, Alex X Lee, Ibrahim Awwal, Henry Bradlow, and Pieter Abbeel. Finding locally optimal, collision-free trajectories with sequential convex optimization. In *Robotics: Science and Systems*, volume 9, pages 1–10, 2013.

[134] Ulrich Schwesinger, Martin Rufli, Paul Furgale, and Roland Siegwart. A sampling-based partial motion planning framework for system-compliant navigation along a reference path. In *Intelligent Vehicles Symposium (IV)*, pages 391–396, 2013.

[135] J A Sethian. A fast marching level set method for monotonically advancing fronts. *Proceedings of the National Academy of Sciences*, 93(4):1591–1595, Feb. 1996. doi: 10.1073/pnas.93.4.1591.

[136] Sumeet Singh, Anirudha Majumdar, Jean-Jacques Slotine, and Marco Pavone. Robust online motion planning via contraction theory and convex optimization. *Proceedings of the IEEE Conference on Decision and Control*, 2017.

[137] Jack Stewart. Google tests drone deliveries in Project Wing trials, 2014.

[138] Dusan M. Stipanovic, Peter F. Hokayem, Mark W. Spong, and Dragoslav D. Siljak. Cooperative Avoidance Control for Multiagent Systems. *Journal of Dynamic Systems, Measurement, and Control*, 129(5):699, 2007.

[139] Brian P Tice. Unmanned Aerial Vehicles – The Force Multiplier of the 1990s. *Airpower Journal*, pages 41–55, 1991.

[140] C.J. Tomlin, J Lygeros, and S. Shankar Sastry. A game theoretic approach to controller design for hybrid systems. *Proceedings of the IEEE*, 88(7):949–970, Jul. 2000.

[141] Jur van den Berg, Ming Lin, and Dinesh Manocha. Reciprocal Velocity Obstacles for real-time multi-agent navigation. In *Proceedings of the International Conference on Robotics and Automation*, pages 1928–1935, May 2008.

[142] P. P. Varaiya. On the Existence of Solutions to a Differential Game. *SIAM Journal on Control*, 5(1):153–162, Feb. 1967. doi: 10.1137/0305009.

[143] Michael Vitus, Vijay Pradeep, Gabriel Hoffmann, Steven Waslander, and Claire Tomlin. Tunnel-milp: Path planning with sequential convex polytopes. In *AIAA guidance, navigation and control conference and exhibit*, page 7132, 2008.

[144] Wikipedia. Beaufort scale. https://en.wikipedia.org/wiki/Beaufort_scale [retrieved May 9, 2017].

[145] Albert Wu and Jonathan P. How. Guaranteed infinite horizon avoidance of unpredictable, dynamically constrained obstacles. *Autonomous Robots*, 32(3):227–242, Apr. 2012.

[146] Insoon Yang, Sabine Becker-Weimann, Mina J. Bissell, and Claire J. Tomlin. One-shot computation of reachable sets for differential games. In *Proceedings of the ACM International Conference on Hybrid systems: Computation and Control*, page 183, 2013.

[147] Melanie Nicole Zeilinger, Colin Neil Jones, and Manfred Morari. Real-time suboptimal model predictive control using a combination of explicit mpc and online optimization. *IEEE Transactions on Automatic Control*, 56(7):1524–1534, 2011.