Enabling Safe and Robust Control of Robots via Hamilton-Jacobi Reachability

by

Anjian Li

B.Sc., Beijing Normal University, 2017

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

in the
School of Computing Science
Faculty of Applied Sciences

© Anjian Li 2020
SIMON FRASER UNIVERSITY
Spring 2020

Copyright in this work rests with the author. Please ensure that any reproduction or re-use is done in accordance with the relevant national copyright legislation.

Approval

Name:	**Anjian Li**
Degree:	**Master of Science (Computing Science)**
Title:	**Enabling Safe and Robust Control of Robots via Hamilton-Jacobi Reachability**
Examining Committee:	**Chair:** Angel Chang Assistant Professor
	Mo Chen Senior Supervisor Assistant Professor
	Angelica Lim Supervisor Assistant Professor
	Manolis Savva External Examiner Assistant Professor
Date Defended:	**March 31, 2020**

Abstract

As autonomous robots become pervasive in daily life, it is important to ensure they successfully accomplish the task while being safe from collisions. These desired behaviours require both perception of the environment and robust control of the robot. Traditional optimal control method Hamilton-Jacobi (HJ) Reachability can formally verify the safety of the robot, but requires an *a priori* known map and is computationally intractable for high dimensional systems. Machine learning is widely used in machine perception, and recently, End-to-End method has been proposed to bridge the perception and control for robotics. However, it suffers from data inefficiency and lack of robustness when applied to robotics tasks.

To address the above challenges, firstly we propose a theoretical improvement on approximating HJ Reachability. Our novel system decomposition technique largely reduces the computation complexity without introducing much conservatism. Both formal mathematical proof and numerical examples are provided to demonstrate its efficiency and guaranteed-safe property. We also present the first HJ Reachability analysis on 6D bicycle model that is previously considered intractable.

Secondly, we apply HJ Reachability to learning-based visual navigation in indoor office environment, where a convolutional neural network (CNN) processes the visual input and predicts waypoint that leads to the goal. We propose a novel cost function for waypoint evaluation and generation based on HJ Reachability analysis, and uses disturbances in dynamics to model CNN's prediction error. Compared to state-of-the-art, our method shows more robust behaviours when navigating in narrow spaces demonstrated in both the simulation and hardware experiment in SFU buildings.

Keywords: Hamilton-Jacobi Reachability, machine learning, visual navigation, optimal control

Dedication

To my supportive parents
and especially,
my beloved grandma

Acknowledgements

First of all I want to especially thank my supervisor Prof. Mo Chen for all the guidance and support through my Master study. Mo led me to the field of robotics and control, and taught me how to do research with his wisdom and great patience. There were days that he discussed with me the details in a paper on the whiteboard for hours, which I will never forget. It was also magical that every time I walked out of his office after the meeting, I got inspired for the work we do and was really motivated in research. I also want to thank Prof. Angelica Lim, Prof. Manolis Savva and Prof. Angel Chang, and Prof. Claire Tomlin for all the valuable discussions that not only enriched my knowledge but also broadened my horizon.

Without my wonderful collaborators, I cannot finish the work in this thesis. Somil had a lot of insights about how vision and learning can be combined with robotics and control. Varun offered great help on simulation and hardware experiment. George was always kind and helped run a lot of real robot experiments.

Then I want to thank my lab folks who make this Master a happy journey. Xubo and I were the first two members of the MARS lab and he helped me in every way. As a beginner to robotics and even computer science, I cannot imagine how to tackle all the difficulties without his generous help. I appreciate the time that we hung out with Zhenqi and Kai, played basketball and had a wonderful meal, and also the happy lunch time with Zhitian. Jack was the real big brother in the lab, and the coffee time will definitely be missed by everyone around, along with Geoff's jokes. Sepehr was always kind to everyone and willing to help. If I had questions about school life, Payam was also the good one to talk to, and thanks for inviting me to your birthday party, it was great! Pratik had tons of knowledge about great food and spots for travel. Jimin also introduced a lot of delicious food and tourist attractions ... There are still a lot of people to mention, including but not limited to Rakesh, Juan, Minh, Atefeh, Dorsa, Taher, Sriraj, George, Francis, Kefan, Bita and Faraz.

Finally I want to thank my mom and dad, my grandma and grandpa to encourage me to pursue the things I love. I also want to thank my partner Qian for always supporting me through this journey.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Recently, autonomous systems including self-driving cars, unmanned aerial vehicles (UAV), service robot and etc., started to gain popularity in our daily life. As there will be more robot-to-robot and robot-to-human interaction, safety becomes an important topic when these robots are applied to the real world. For safe-critical systems like drones, any collision can bring huge damage to humans and the environment. For multi-agent systems, a team of robot can work together effectively only when they won't collide with each other. Therefore, robust algorithms are urgently need for autonomous systems to safely operate in the physical world.

Hamilton-Jacobi (HJ) Reachability is a formal verification tool widely used in robotic safety analysis for controlled nonlinear systems experiencing adversarial disturbances [16]. Given a target set that represents a set of unsafe states or goal states, the Backward Reachable Tube (BRT) specifies the states from which a system is guaranteed to enter the target under the worst-case disturbance.

Nonetheless, computing BRTs requires state space discretization, which suffers from exponential computational time and space complexity with respect to the state space dimension, known as "curse of dimensionality". In addition, full observation of the environment is a necessary input to the BRT computation, which may be hard to obtain accurately in reality.

Deep learning is widely used in robot perception of the environment. In visual navigation tasks, end-to-end (E2E) method learns to directly map the current image to the control signal of the robot [25, 61]. In [8], waypoint navigation (WayPtNav) is proposed, where a Convolutional Neural Network (CNN) predicts a waypoint that is used by the dynamics model for planning and tracking a trajectory to the waypoint.

However, the CNN inevitably makes prediction errors, ultimately leading to collisions, especially when the robot is navigating through cluttered and tight spaces. In addition, it is often inefficient to collect large amount of training data for real robots.

In Chapter 2, we focus on addressing "curse of dimensionality" in HJ Reachability method. We first propose a State Dependency Graph to represent the system dynamics, and then decompose the full system in a way that only dependent states are included in each subsystem, and "missing" states are treated as bounded disturbance. Thus for a large variety of dynamics in robotics, BRTs can be quickly approximated in lower-dimensional chained subsystems, while conservatism is preserved in the right direction to guarantee safety. We demonstrate our method with numerical experiments on the 4D Quadruple Integrator, and the 6D Bicycle, an important car model that has been intractable to analyze to the best of our knowledge.

In Chapter 3, we propose a novel visual navigation framework using a hybrid approach based on [8], where a CNN processes the current image observation to predict a waypoint, and HJ reachability generates robust supervision. Specifically, by modeling the prediction error of the CNN as disturbances in dynamics, the proposed method generates waypoints that are robust to these disturbances, and consequently to the prediction errors. Moreover, using globally optimal HJ reachability analysis leads to predicting waypoints that are time-efficient and do not exhibit greedy behavior. Through simulations and experiments on a hardware testbed, we demonstrate the advantages of the proposed approach for navigation tasks where the robot needs to navigate through cluttered, narrow indoor environments.

In Chapter 4, we summarize the work in this thesis.

Chapter 2

Guaranteed-Safe Approximate Reachability via State Dependency-Based Decomposition

As the popularity of mobile autonomous systems rapidly grows in daily life, the importance of safety of these systems substantially increases as well. Especially for safe-critical systems like self-driving cars, drones, and etc., collision avoidance is indispensable, since any crash can lead to serious damage to human, other autonomous agents and the environment. Thus, formal verification is urgently needed to analyze safety properties of these systems.

Optimal control and differential game theory are well-studied for safe-critical systems [9, 10, 40, 45]. These methods are ideal for analyzing controlled nonlinear systems under the influence of adversarial disturbances. One powerful tool for safety verification is reachability analysis, which not only characterizes the safe states of the systems, but also provides safety controllers [2, 22, 34]. It has been widely used in trajectory planning [26, 35, 50, 55], air traffic management [13, 14], and multi-agent collision avoidance [15, 18].

In HJ Reachability, by computing Backward Reachable Tube (BRT), one can easily quantify the states and controls that guarantee safety [16]. In a collision avoidance scenario, given system dynamics and the unsafe states, the BRT represents the states from which reaching the unsafe states is inevitable within a specified time horizon under the worst-case disturbance.

There is a variety of reachability analysis methods. [3, 22] focus on analytic solutions, which are fast to compute, but require specific types of targets, e.g. polytopes or hyperplanes. Some other techniques have strong assumptions such as linear dynamics [34, 39], or dynamics that do not include any control and disturbance [17]. HJ reachability is the most flexible method that accommodates nonlinear dynamics and arbitrary shapes of target sets; however, such flexibility requires level set methods [42, 48], in which value functions are stored on grid points in the discretized state space. Such an approach suffers from the curse of dimensionality.

Previously, several approaches have been proposed to reduce the computation burden for HJ reachability. Projection methods have been investigated to approximate BRTs in lower-dimension

$$\begin{tikzcd} z_1 \arrow[r] & z_2 \arrow[r] & z_3 \arrow[r] & z_4 \end{tikzcd}$$

(a)

$$\begin{tikzcd} z_1 \arrow[r] & z_2 \arrow[d, dashed] & z_2 \arrow[r] & z_3 \arrow[d, dashed] & z_3 \arrow[r] & z_4 \\ & z_3 & & z_4 & & \end{tikzcd}$$

(b)

Figure 2.1. (a) The State Dependency Graph for 4D Integrator. (b) A decomposed State Dependency Graph for 4D Integrator. The light blue vertices and edges indicate the missing states and their dependencies. Our system method approximates the BRT of the full-dimensional system in Fig. 2.1(a) by concurrently computing a sequence of BRTs for the subsystems in Fig. 2.1(b).

spaces [46], but it can be difficult to choose which dimensions to project to, and the results can be overly conservative at times. Integrator structures have been analyzed in [44] to reduce dimensionalilty by one. The state decoupling disturbances method [11] treats certain states as disturbance, thus subsystems can be decoupled and computed on lower-dimensional for goal-reaching problems. An exact system decomposition method [12] has also been used for reducing computation burden without incurring approximation errors. However, this approach is only applicable if there exist self-contained systems, which can sometimes be restrictive.

In this chapter, we propose a novel system decomposition method that exploits state dependency information in the system dynamics in a more sophisticated, multi-layered manner compared to previous works. Our approach involves representing the system dynamics using a directed dependency graph, and decomposing the full system into subsystems based on this graph. This is done in a way that allows computation of BRT over-approximations to be tractable yet not overly conservative. A set of lower-dimensional BRTs is computed concurrently: "missing" states in each subsystem are treated as disturbances, the range of which are bounded by the other BRT approximations being computed.

Our method is applicable to a large variety of system dynamics, especially those with a loosely coupled, "chained" structure. It is easy to combine our method with other decomposition techniques to achieve even more dimensionality reduction. Beyond that, our method also offers a flexible way of adjusting the trade-off between computational complexity and degree of conservatism. This means that our method is adaptable and relevant regardless of the amount of computational resources available.

Organization:

- In Section 2.1, we introduce the background on HJ reachability and projection operations of value functions.

- In Section 2.2, we first present how our method decomposes dynamical systems into several smaller subsystems. Then, we discuss how BRT over-approximations can be computed given these subsystems. Finally, we present a proof of correctness, computation complexity analysis, and a discussion on target set selection.
- In Section 2.3, we present numerical results for the 4D Quadruple Integrator and 6D Bicycle.
- In Section 2.4, we make brief concluding remarks and suggest future research directions.

2.1 Background

HJ reachability is a powerful tool for guaranteed-safety analysis of nonlinear system dynamics, compatible with arbitrary shapes of target sets, which represent the safe or unsafe states. Given a target set, minimal BRTs can be used to specify the states that will inevitably lead to collision; safety is guaranteed for all states outside of the BRT. In this section, we present the necessary setup for HJ reachability computation, and introduce the projection operations used in our method.

2.1.1 System Dynamics

Let $z \in \mathbb{R}^n$ represent the state and s represent time. The system dynamics is described by the following ODE:

$$\begin{aligned} \dot{z} = \frac{\mathrm{d}z}{\mathrm{d}s} = f(z, u, d), \qquad & s \in [s_0, 0], s_0 \leq 0 \\ & u \in \mathcal{U}, d \in \mathcal{D} \end{aligned} \tag{2.1}$$

The $u(\cdot)$ and $d(\cdot)$ denote the control function and disturbance function. For any fixed u and d, the dynamics $f : \mathbb{R}^n \times \mathcal{U} \times \mathcal{D} \to \mathbb{R}^n$ is assumed to be uniformly continuous, bounded and Lipschitz continuous with respect to all arguments; thus, a unique solution to (2.1) exists given u and d.

The solution for (2.1), or trajectory, is denoted as $\zeta(s; z, s_0, u(\cdot), d(\cdot)) : [s_0, 0] \to \mathbb{R}^n$, which starts from state z at time s_0 under control u and disturbance d. ζ satisfies (2.1) almost everywhere with initial condition:

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}s}\zeta(s; z, s_0, u(\cdot), d(\cdot)) = f(\zeta(s; z, s_0, u(\cdot), d(\cdot)), u(s), d(s)), \\ \zeta(s_0; z, s_0, u(\cdot), d(\cdot)) = z. \end{aligned} \tag{2.2}$$

The control and disturbance have opposing objectives, and are modeled as opposing players in a differential game. Following [45], we let $d(\cdot) = \gamma[u](\cdot)$, where γ is drawn from only nonanticipative strategies.

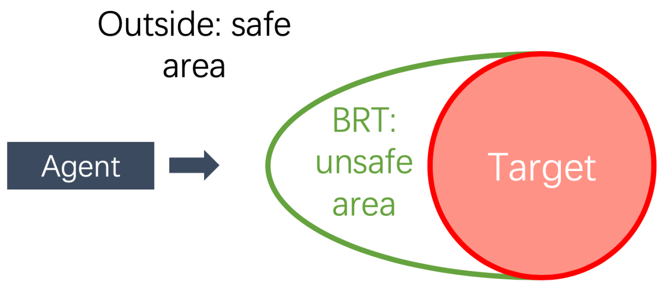


Figure 2.2. An example of a target and its BRT. To avoid the target, the agent should stay outside of the BRT.

2.1.2 Hamilton-Jacobi Reachability

Given a target $\mathcal{T}$ to avoid, the BRT is the set of states from which there exists a disturbance such that entering the target during the time horizon of duration $|s_0|$ is inevitable despite the best control. This is illustrated in Fig. 2.2.

Therefore, agents can remain safe for some time horizon by staying outside of the BRT of the corresponding duration. The definition of the minimal BRT $\bar{\mathcal{A}}(s)$ is as follows:

$$\begin{aligned}\bar{\mathcal{A}}(s) = \{z : \exists d(\cdot) \in \mathbb{D}, \forall u(\cdot) \in \mathbb{U}, \exists s \in [s_0, 0],\\ \zeta(s; z, s_0, u(\cdot)) \in \mathcal{T}\}\end{aligned} \tag{2.3}$$

In the HJ formulation, the target set is represented as the sub-level set of some function $l(z)$, where $z \in \mathcal{T} \Leftrightarrow l(z) \leq 0$. Then, the HJ formulation of the reachability problem becomes the differential game problem below:

$$V(z,s) := \min_{d(\cdot)} \max_{u(\cdot)} \min_{s\in[s_0,0]} l(\zeta(0; z, s, u(\cdot), d(\cdot))) \tag{2.4}$$

The value function $V(z, s)$ can be obtained as the viscosity solution of the following HJ partial differential equation: (2.4):

$$\begin{aligned}\min\{D_s V(z,s) + H(z, \nabla V(z,s)), V(z,0) - V(z,s)\} = 0,\\ V(z,0) = l(z), s \in [s_0, 0]\end{aligned} \tag{2.5}$$

where

$$H(z, \nabla V(z,s)) = \min_{d(\cdot)} \max_{u(\cdot)} \nabla V(z,s)^\top f(z,u) \tag{2.6}$$

The level set method [42] is a computation tool to solve (2.5) in the discretized state space. Recently, toolboxes [43, 57] have been developed to take pre-defined system dynamics and target sets as input, and numerically compute BRTs using the level set method.

2.1.3 Projection

We present two kinds of projection operations that are used to manipulate value functions of BRTs between high dimension spaces and their low dimension subspaces. Let $V(x, y) : \mathbb{R}^{n_x+n_y} \to \mathbb{R}$ be a value function in $n_x + n_y$ dimension space, where $x \in \mathbb{R}^{n_x}$ and $y \in \mathbb{R}^{n_y}$.

Given a BRT represented by $V(x, y)$, we define the projected BRT in its n_x dimension subspace by the value $W(x) : \mathbb{R}^{n_x} \to \mathbb{R}$, where

$$W(x) = \min_{y} V(x, y) \tag{2.7}$$

Given $W(x)$ in n_x-dimensional space, we define the value function $V(x, y)$ representing the back projected BRT as

$$V(x, y) = W(x),\ \forall y \in \mathbb{R}^{n_y}. \tag{2.8}$$

2.2 Methodology

Reachability analysis relies on an accurate model of the robotic system under consideration, but accurate models tend to be high-dimensional. This often makes HJ reachability intractable due to the curse of dimensionality.

In this section, we first present a novel method to decompose the dynamical system in (2.1) into several coupled subsystems, based on a State Dependency Graph. Then, we provide an algorithm for computing BRTs with these subsystems. Finally, we provide the proof of correctness of our method, analyze its computational time and space complexity, and discuss the constraints for target sets.

2.2.1 System decomposition

State Dependency Graph

Let S be the set of states, $S = \{z_i\}_{i=1}^n$. We first define the notion of "state dependency": for some states $z_i, z_j \in S$, z_i depends on z_j in $f(z, s)$ means $\frac{\mathrm{d}z_i}{\mathrm{d}s}$ is a function of z_j.

In order to clarify these dependency relationships between each state of S in system dynamics $f(z, s)$, we define a directed State Dependency Graph $G = (S, E)$ based on the system dynamics. The set of vertices is denoted S, and contains all state variables. If some state component $z_i \in S$ depends on z_j in $f(z, s)$, then the graph G would have a directed edge from z_i to z_j, $(z_i, z_j) \in E$.

Often, high-dimensional dynamics contain chains of integrators. Thus, we consider a running example: 4D Quadruple Integrator, whose states $z = (z_1, z_2, z_3, z_4) \in \mathbb{R}^4$. The dynamics are as follows:

$$\begin{bmatrix} \dot{z_1} \\ \dot{z_2} \\ \dot{z_3} \\ \dot{z_4} \end{bmatrix} = \begin{bmatrix} z_2 + d \\ z_3 \\ z_4 \\ u \end{bmatrix}, u \in \mathcal{U}, d \in \mathcal{D}, \tag{2.9}$$

where u and d denote the control and disturbance. For the system in (2.9), $S = \{z_1, z_2, z_3, z_4\}$, $E = \{(z_1, z_2), (z_2, z_3), (z_3, z_4)\}$, and its State Dependency Graph $G = (V, E)$ is shown in Fig. 2.1(a).

Choosing coupled subsystems

Given the State Dependency Graph G and the computational space constraint that each subsystem can be at most p-dimensional, we can decompose the full system S with state z into several coupled subsystems $S_1, S_2, \ldots, S_m$ with subsystem states denoted as $x_1, x_2, \ldots, x_m$ respectively, with the following properties:

- In every subsystem, each state should depend on or be depended on by at least one other state.
- Every subsystem should include no more than p states.
- Subsystems should be chained: each subsystem should share at least one state with another subsystem.

As a result, the decomposed system is represented by connected subgraphs of G each representing a subsystem. Let S_i be the set of state variables included in the i^{th} subsystem, and let S_i^c be the set of states that are not included in i^{th} subsystem, i.e. $\forall i, S_i^c = S \setminus S_i$.

For example, suppose that one requires the maximum dimensionality of subsystems to be two, $p = 2$. We can decompose the 4D Quadruple Integrator into 3 subsystems with $S_1 = \{z_1, z_2\}, S_2 = \{z_2, z_3\}, S_3 = \{z_3, z_4\}$. In terms of the subsystem state variables, we have $x_1 = (z_1, z_2)$, $x_2 = (z_2, z_3)$, and $x_3 = (z_3, z_4)$. The result of the decomposition is shown in Eq. (2.10), and the corresponding State Dependency Graph representing subsystems is illustrated in Fig. 2.1(b).

$$\begin{aligned}
S_1 : \dot{x_1} &= \begin{bmatrix} \dot{z_1} \\ \dot{z_2} \end{bmatrix} = \begin{bmatrix} z_2 + d \\ z_3 \end{bmatrix}, d \in \mathcal{D}, z_3(s) \in R_{z_3}(z_2, s) \\
S_2 : \dot{x_2} &= \begin{bmatrix} \dot{z_2} \\ \dot{z_3} \end{bmatrix} = \begin{bmatrix} z_3 \\ z_4 \end{bmatrix}, z_4(s) \in R_{z_4}(z_3, s) \\
S_3 : \dot{x_3} &= \begin{bmatrix} \dot{z_3} \\ \dot{z_4} \end{bmatrix} = \begin{bmatrix} z_4 \\ u \end{bmatrix}, u \in \mathcal{U}
\end{aligned} \tag{2.10}$$

In each subsystem, there may be zero or more missing state components. For the 4D Quadruple Integrator, the missing state of S_1 is z_3 since $z_3 \notin S_1$. To guarantee safety, we assume the worst case for the missing states by treating them as virtual disturbances, which leads to an over-approximated BRT that is conservative in the right direction [46].

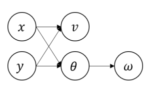
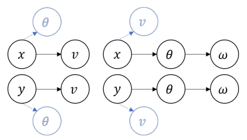
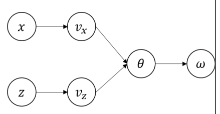
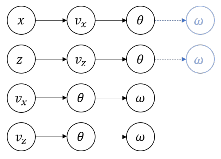
To avoid excessive conservatism, the virtual disturbances are not drawn from the whole computation range. Instead, we will compute the value functions of all the subsystems concurrently to able to access the up-to-date approximate BRTs of other subsystems. Since all the subsystems are chained, one can determine the bounds of virtual disturbances by searching the value functions of other subsystems that contain the corresponding state.

Formally, consider some subsystem S_i, and let $R_{z_j}(x_i, s)$ denote the range of the missing state $z_j \notin S_i$. Suppose $z_j \in S_k$ with S_k being chained with S_i, $S_k \cap S_i \neq \emptyset$. Note that k is not unique, as z_j may be a state of many different subsystems. Furthermore, let $W_k(x_k, s)$ be the value function for the subsystem S_k at the time s. Then, $R_{z_j}(x_i, s)$ is determined from $W_k(x_k, s)$ as follows:

$$R_{z_j}(x_i, s) = \{z_j | W_k(x_k, s) \leq 0, \ \forall k \text{ such that } z_j \in S_k \wedge S_k \cap S_i \neq \emptyset\} \tag{2.11}$$

Besides reducing dimensionality, our method also provides a simple way to adjust the trade off between computational burden and degree of conservatism. Depending on different requirements for computational time, computational space, and approximation accuracy, one can easily switch between having higher-dimensional subsystems for which BRTs are slower to compute but more accurate, and having lower-dimension subsystems for which BRTs are faster to compute but more conservative.

Table 2.1. Decomposition suggestions for 5D Car and 6D Planar Quadrotor

System configuration	System dynamics	State Dependency Graph	Decomposed State Dependency Graph	Time and space
5D Car (x, y)-position θ - heading v - speed ω - turn rate u_a - accel. control u_α - ang. accel. control	$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\theta}\\ \dot{v}\\ \dot{\omega}\end{bmatrix} = \begin{bmatrix}v\cos\theta\\ v\sin\theta\\ \omega\\ u_a\\ u_\alpha\end{bmatrix}$		 θ x v y v θ v x θ ω y θ ω v	Ground truth: both $O(k^5)$ Decomposition: $O(k^4)$ and $O(k^3)$
6D Planar Quadrotor (x, y)-position (v_x, v_z) - velocity θ - pitch ω - pitch rate u_T - thrust control u_τ - ang.accel.control	$\begin{bmatrix}\dot{x}\\ \dot{z}\\ \dot{v_x}\\ \dot{v_z}\\ \dot{\theta}\\ \dot{\omega}\end{bmatrix} = \begin{bmatrix}v_x\\ v_z\\ -u_T\sin\theta\\ u_T\cos\theta - g\\ \omega\\ u_\tau\end{bmatrix}$		 x v_x θ ω z v_z θ ω v_x θ ω v_z θ ω	Ground truth: both $O(k^6)$ Decomposition: $O(k^4)$ and $O(k^3)$

Algorithm 1 Approximating full-dimensional BRTs with chained subsystems

```
Require: System dynamics $f(z, u, d)$ described as (2.1) and a function $l(z)$ representing the target
    set $\mathcal{T}$
 1: Initialize the full-dimensional final time value function $V(z, 0)$ as (2.5)
 2: Decompose the entire system into chained subsystems $S_1, S_2, S_3, ...S_n$, based on Section 2.2.1
 3: Initialize the final time value functions $W_i(x_i, 0)$ for each subsystem $S_i$ based on (2.12)
 4: for $(s = 0; s \geq s_0; s = s - \Delta s)$ do
 5:     for each subsystem $S_i$
 6:         Find the range $R_{z_j}(x_i, s)$ of the missing states $z_j$ based on (2.11)
 7:         Obtain $W_i(x_i, s)$ by solving the HJ equation in (2.13)
 8: end for
 9: Obtain the approximated $V(z, s_0)$ based on (2.15)
10: Obtain the approximated full-dimensional BRT from the zero sub-level set of $V(z, s_0)$
11: For any time $s$, obtain the optimal controller as (2.16) and (2.17)
```

Although in general it may not be possible to decompose arbitrary dynamical systems in the form of (2.1) in a way that saves computation time, our approach is very flexible and can often successfully decompose many realistic system dynamics. We demonstrate the method by decomposing the high-dimensional, tightly coupled 6D Bicycle in Section 2.3.2. We also present decomposition suggestions for two other common system dynamics, 5D car [51] and 6D planar quadrotor [55], in TABLE 2.1.

2.2.2 Backward Reachable Tube Computation

We now present the procedure for over-approximating BRTs with low-dimensional chained subsystems $S_1, \dots, S_m$. Given the target set $\mathcal{T}$ and the corresponding final condition to the HJ PDE (2.5), $l(z) = V(z, 0)$, we project the full-dimensional BRT onto the subspace of each subsystem S_i, and initialize the final time value function $W_i(x_i, 0)$ for the subsystem S_i using the projection operation in (2.7) as follows:

$$W_i(x_i, 0) = \min_{z_i \in S_i^c} V(z, 0) \tag{2.12}$$

Then, given $W_i(x_i, t)$ for some t, we compute the value function $W_i(x_i, s)$ backwards in time for each subsystem following standard HJ PDE theory and level-set methods, while treating missing variables as virtual disturbances with appropriate bounds. For each time step $s \in [t - \Delta s, t]$, $W_i(x_i, s)$ is the viscosity solution of the following HJ partial differential equation:

$$\begin{aligned} \min\{D_s W_i(x_i, s) + H(x_i, \nabla W_i(x_i, s)), \\ W_i(x_i, 0) - W_i(x_i, s)\} = 0, \end{aligned} \tag{2.13}$$

The Hamiltonian is given by

$$H(x_i, \nabla W_i(x_i, s)) = \min_{\substack{d \in \mathcal{D} \\ z_k \in R_{z_k}(x_i, s), \\ \forall z_k \in S_i^c}} \max_{u \in \mathcal{U}} \sum_{z_j \in S_i} \frac{\partial W_i(x_i, s)}{\partial z_j} \cdot \frac{\partial z_j}{\partial s} \tag{2.14}$$

where $R_{z_i}(x_i, s)$ is the range of missing states $\{z_k\}$ given in (2.11).

This procedure starts at $t = -\Delta s$, and finishes when $W_i(x_i, s_0)$ is obtained. Finally, we take the maximum of all the $W_i(x_i, s_0)$ as the over-approximation of the full-dimensional initial time $V(z, s_0)$:

$$V(z, s_0) = \max_i \ W_i(x_i, s_0) \tag{2.15}$$

In general for any time s, we also have

$$V(z, s) = \max_i \ W_i(x_i, s). \tag{2.16}$$

The optimal controller is given by

$$u^*(s) = \operatorname*{argmax}_u \nabla V(z, s)^\top f(z, u). \tag{2.17}$$

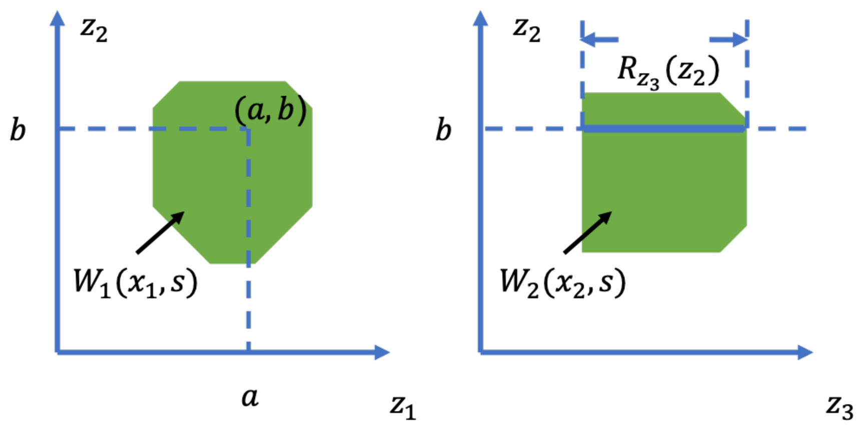


Figure 2.3. Searching missing states for 4D Quadruple Integrator. Left: a BRT described by $W_1(x_1, s)$ for subsystem S_1. Right: a BRT described by $W_2(x_2, s)$ for subsystem S_2. When solving on the grid point of $(z_1, z_2) = (a, b)$ in subsystem S_1 at time s, we find the range of missing state $R_{z_3}(z_2, s)$ inside the BRT from subsystem S_2.

The computation process is summarized in **Algorithm 1**.

Consider our running example, the 4D Quadruple Integrator in (2.9) and its subsystems in (2.10). In particular, consider the BRT computation for subsystem $S_1 = \{z_1, z_2\}$. At the time s, the HJ equation in (2.13) for subsystem S_1 becomes

$$\min\left\{\frac{\partial W_1(x_1,s)}{\partial s} + \min_{z_3\in R_{z_3}(z_2,s)}\left(\frac{\partial W_1(x_1,s)}{\partial z_1}z_2 + \frac{\partial W_1(x_1,s)}{\partial z_2}z_3\right), W_1(x_1,0) - W_1(x_1,s)\right\} = 0 \tag{2.18}$$

Here for the subsystem S_1 in (2.10), given z_2, we are able to find the range of z_3 in the subsystem S_2. Let $W_2(x_2, s)$ be the value function for the subsystem S_2 at the time s, the range of z_3 given z_2 will be defined as $R_{z_3}(z_2, s)$:

$$R_{z_3}(z_2, s) := \{z_3 | W_2(x_2, s) \leq 0\} \tag{2.19}$$

A graphical interpretation of (2.19) is in Fig. 2.3. For a specific grid point of $(z_1, z_2) = (a, b) \in \mathbb{R}^2$ in the subsystem S_1, the range of the missing state $R_{z_3}(z_2, s)$ can be drawn from the subsystem S_2 with the corresponding $z_2 = b$.

2.2.3 Proof and Discussions

Proof of Correctness

In this section, we show that the BRT generated from our method is an over-approximation of the true BRT obtained from (2.5). Let $V_i(z, s)$ denote the full-dimensional value function that back projected from $W_i(x_i, s)$ at the time s, based on the projection operation in (2.8):

$$V_i(z, s) = W_i(x_i, s), \forall z_i \in S_i^c \tag{2.20}$$

Because at any time s, we maximize over $W_i(x_i, s)$ to obtain the over-approximation, to prove the following Theorem 1 is sufficient to prove that each approximate value function V_i is no larger than the true value function V.

Theorem 1. *For any subsystem S_i at any time step $s \in [t - \Delta s, t]$, $V_i(z, s) \leq V(z, s)$*

Proof. We prove this by mathematical induction. For any subsystem S_i, we first show that the Theorem at final time $s = 0$ is true. Then we prove that for any time step $s \in [t - \Delta s, t]$, if $V_i(z, t) \leq V(z, t)$, we will have $V_i(z, t - \Delta s) \leq V(z, t - \Delta s)$.

At the final time $s = 0$, $W_i(x_i, 0)$ is initialized as (2.12) and $V_i(z, s)$ is initialized as (2.20), thus trivially we have

$$V_i(z, 0) \leq V(z, 0). \tag{2.21}$$

For any time step $s \in [t - \Delta s, t]$, $V(z, s)$ is the viscosity solution of (2.5) with final value $V(z, t)$. Let $\tilde{V}_i(z, t - \Delta s)$ be the viscosity solution of (2.5) with final value $V_i(z, t)$. Since $V_i(z, t) \leq V(z, t)$, we have

$$\tilde{V}_i(z, t - \Delta s) \leq V(z, t - \Delta s) \tag{2.22}$$

Let $W_i(x_i, s)$ be the viscosity solution of (2.13) at $s \in [t - \Delta s, t]$ with final value $W_i(x_i, t)$. When solving $W_i(x_i, t - \Delta s)$, the Hamiltonian $H(x_i, \nabla W_i(x_i, s))$ is computed as (2.14). For comparison, when solving $\tilde{V}_i(z, t - \Delta s)$, the Hamiltonian $H(z, \nabla \tilde{V}_i(z, s))$ is computed as (2.6).

Because $V_i(z, t)$ is back projected from $W_i(x_i, t)$ as (2.20), in $H(z, \nabla V_i(z, s))$ we have $\frac{\partial V_i(z,t)}{\partial z_i} = 0, \forall z_i \in S_i^c$. In addition, missing states are treated as disturbances in $H(x_i, \nabla W_i(x_i, s))$, so $H(x_i, \nabla W_i(x_i, s)) = \min_{\forall z_i \in S_i^c} H(z, \nabla \tilde{V}_i(z, s))$. Therefore,

$$H(x_i, \nabla W_i(x_i, s)) \leq H(z, \nabla \tilde{V}_i(z, s)), \forall z_i \in S_i^c. \tag{2.23}$$

Thus we obtain

$$W_i(x_i, t - \Delta s) \leq \tilde{V}_i(z, t - \Delta s), \forall z_i \in S_i^c. \tag{2.24}$$

Because $V_i(z, t - \Delta s)$ is back projected from $W_i(x_i, t - \Delta s)$ as (2.20), we have

$$V_i(z, t - \Delta s) \leq \tilde{V}_i(z, t - \Delta s). \tag{2.25}$$

Finally, for any time step $s \in [t, t - \Delta s]$, we combine (2.22) and (2.25) and obtain

$$V_i(z, t - \Delta s) \leq V(z, t - \Delta s). \tag{2.26}$$

□

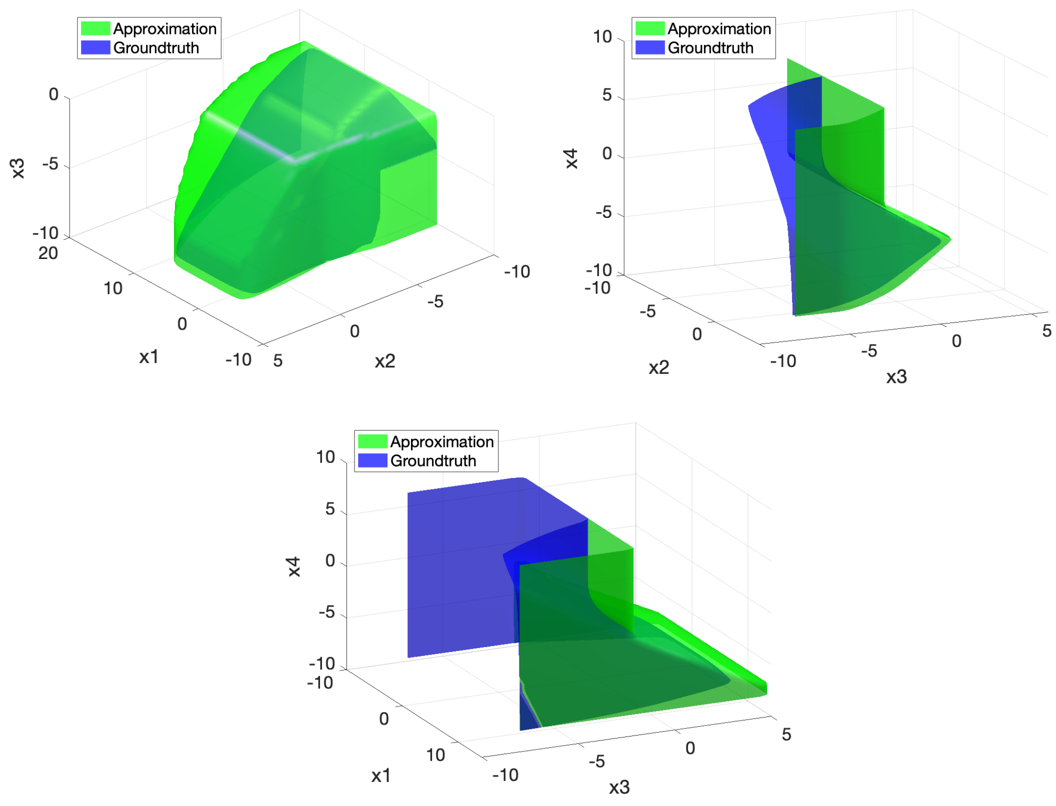


Figure 2.4. Comparison of ground truth BRTs (blue) and our approximated BRTs (green) for 4D Quadruple Integrator at $s = -1$. From top left, top right to bottom are 3D slices at $z_4 = -2, z_1 = 2.6, z_2 = -4.2$. There are few numerical errors.

Computation complexity

Let k be the number of grid points in each dimension for the numerical computation. The computational space complexity is determined by the largest dimension of subsystems. If each subsystem S_i has N_i states, $k^{\max_i N_i}$ grid points are needed to store the value function. Overall, the space complexity is $O(k^{\max_i N_i})$.

For computation time, there are two non-trivial parts: solving the HJ PDE and searching the missing states. The HJ PDE is solved on a grid with $O(k^{N_i})$ grid points. If this search is done over an M_i-dimensional grid for subsystem S_i, then these nested loops have a time complexity of $O(k^{N_i+M_i})$. Overall, the upper limit of the computation time will be the longest time among all subsystems, $O(k^{\max_i\{N_i+M_i\}})$. The computational complexity of specific numerical examples can be found in Section 2.3.

Target sets

Our decomposition technique has two constraints of the target sets. First, for each subsystem, there should be a clear boundary of the target, so that the subsystem value function can provide a virtual disturbance bound to other subsystems. Second, due to shared controls and disturbances in subsystems, the entire target should be the intersections of all targets from each subsystem to ensure the conservative approximation of BRT, according to [12].

2.3 Numerical Experiments

We demonstrate our method on the running example, 4D Quadruple Integrator, and on the higher-dimensional, heavily coupled 6D Bicycle model [55]. For the 4D Quadruple Integrator, we compare the approximate BRT computed using our method to ground truth BRT obtained from the full-dimensional computation, showing that our method maintains the safety guarantee without introducing much conservatism. For 6D Bicycle, we present for the first time a conservative but still practically useful BRT in a realistic simulated autonomous driving scenario. To the best of our knowledge, this was previously intractable. All the experiment are implemented on an AMD Ryzen 9 3900X 12-Core Processor with ToolboxLS [43] and helperOC toolbox.

2.3.1 4D Quadruple Integrator

The system dynamics of the 4D Quadruple Integrator is given in (2.9). Using our method, we decompose the system into subsystems shown in (2.10). Starting from the target set $\mathcal{T}$ in (2.27), we compute the approximate BRT for a time horizon of 1.0 second,

$$\mathcal{T} := \{(z_1, z_2, z_3, z_4) \mid -6 < z_1 < 6, z_2 < -4, z_3 < -2\} \tag{2.27}$$

In Fig. 2.4, we visualize the 4D BRT through 3D slices at the initial time $s = -1.0$. From top left, top right to bottom, our approximated BRTs (green) and the ground truth BRTs (blue) are shown, at the slices of $z_4 = -2$, $z_1 = 2.6$, and $z_2 = -4.2$ respectively. The results show that our approximated BRTs are similar in shape to the ground truth BRT while being a little bigger, which indicates that our results are conservative in the right direction: if a state is outside of the approximate BRT, it is guaranteed to be safe. There are few exceptions due to numerical errors.

For the 4D Quadruple Integrator, the largest subsystem has two states, so the computation space is $O(k^2)$. When solving subsystems S_1 and S_2, for any fixed z_1 and z_2, we should search the missing state z_2 and z_3 from the BRT of subsystem S_2 and S_3. Thus, the computation time is $O(k^3)$. To compare, computing ground truth BRTs for 4D Quadruple Integrator in the full dimension space will cost $O(k^4)$ both on space and time.

In our experiment, it takes 2.5 seconds to compute approximation from decomposition, while it takes 420 seconds to compute the ground truth in full dimension.

2.3.2 6D Bicycle

We now examine the proposed decomposition method on a practical example involving the 6D Bicycle model, and illustrate the utility of our method on decomposing high-dimensional system and heavily coupled systems. To the best of our knowledge, this is the first practically usable minimal

BRT computation for 6D Bicycle, a model widely used to approximate the behaviour of four-wheeled vehicles such as autonomous cars.

Problem Setup

The system dynamics is given in (2.28). X and Y denote position in the global frame, ψ denotes the orientation angle with respect to the X axis [1], v_x and v_y denote the longitudinal and lateral velocities, and ω denotes the angular speed. The controls are δ_f and a_x, which represent the steering angle and longitudinal acceleration, respectively.

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{\psi} \\ \dot{v_x} \\ \dot{v_y} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} v_x \cos\psi - v_y \sin\psi \\ v_x \sin\psi + v_y \cos\psi \\ \omega \\ \omega v_y + a_x \\ -\omega v_x + \frac{2}{m}(F_{c,f}\cos\delta_f + F_{c,r}) \\ \frac{2}{I_z}(l_f F_{c,f} - l_r F_{c,r}) \end{bmatrix} \tag{2.28}$$

To decompose 6D Bicycle, we hope to achieve the best accuracy within the current computation resources, hence we set the space and time limits to be $O(k^4)$. Based on the State Dependency Graph for 6D Bicycle in Fig. 2.5(a), we choose the subsystems in (2.29) with the corresponding decomposed State Dependency Graph shown in Fig. 2.5(b), which requires $O(k^3)$ space and $O(k^4)$ time complexity.

$$\begin{aligned} &x_1 = (X, v_x, v_y), x_2 = (Y, v_x, v_y), x_3 = (X, \psi), \\ &x_4 = (Y, \psi), x_5 = (v_x, v_y, \omega), x_6 = (\psi, \omega) \end{aligned} \tag{2.29}$$

We design the target set $\mathcal{T}$ with respect to X, Y, ψ and v_x in (2.30). This situation can be represented as a one way road surrounded by an open area in a parking lot as Fig. 2.6. In the one way road, only a positive forward speed and a forward orientation range is allowed [2].

$$\mathcal{T} := \{(X, Y, \psi, v_x, v_y, \omega) \mid -6 < X < 6, -2 < Y < 2, \psi < 7\pi/4, v_x < 0\} \tag{2.30}$$

We compute the BRT for a time horizon of 2 seconds.

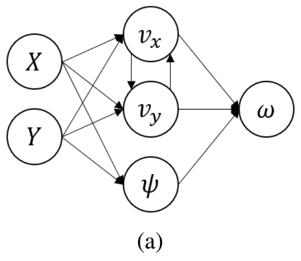
BRT Result

To visualize the 6D BRT at $t = -2.0$, we present several 3D slices in Fig. 2.7 and Fig. 2.8.

Fig. 2.7 shows the BRT at the slice of $\psi = \pi/4, \omega = -1.1$, $v_y = 0$ (left) and 18 (right), indicating the range of v_x to avoid for different X and Y. As shown, the farther from the area

[1] Computation bound for ψ is $[\pi/4, 9\pi/4]$

[2] Combined with the computation bound, the safe orientation range are $[-\pi/4, \pi/4]$



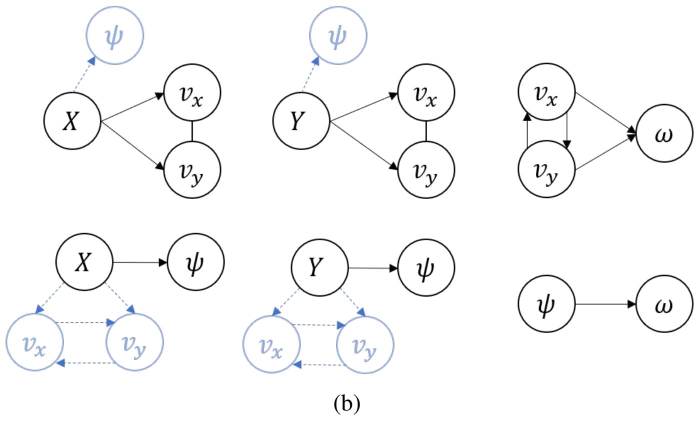


Figure 2.5. (*a*) The State Dependency Graph for 6D Bicycle. (*b*) A decomposed State Dependency Graph for 6D Bicycle. The light blue vertices and edges indicate the missing states and their dependency.

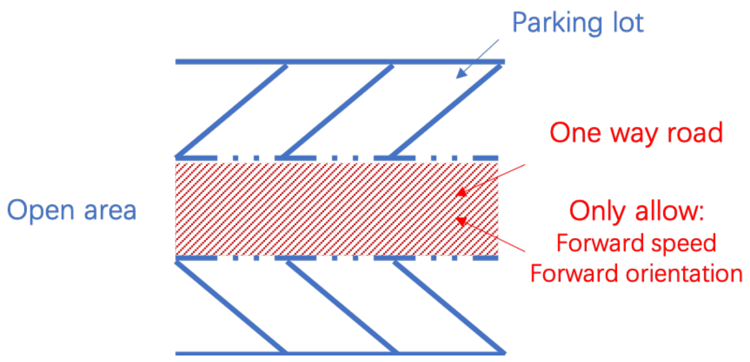


Figure 2.6. A target set example for 6D Bicycle. Inside a parking lot, there is a one-way road surrounded by open areas, where only a positive forward speed and a forward orientation range are allowed.

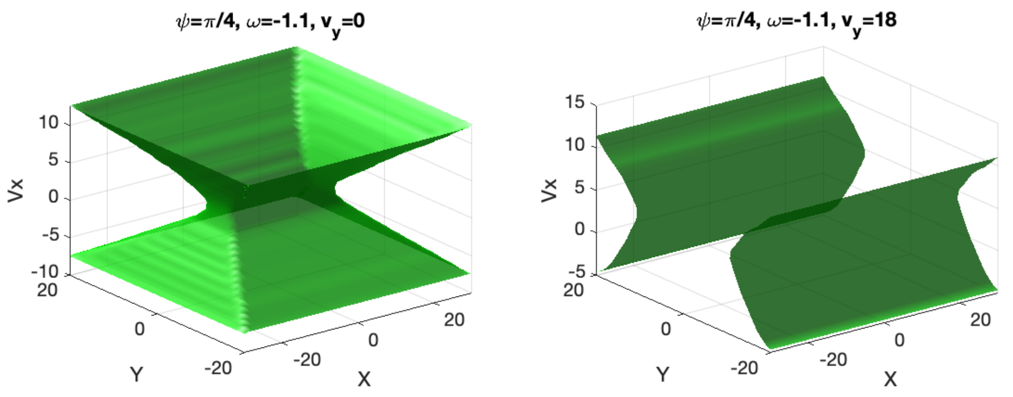


Figure 2.7. 3D slices of (X, Y, v_x) from 6D BRT at $s = -2$. Left: slice at $\psi = \pi/4, \omega = -1.1, v_y = 0$. Right: slice at $\psi = \pi/4, \omega = -1.1, v_y = 18$

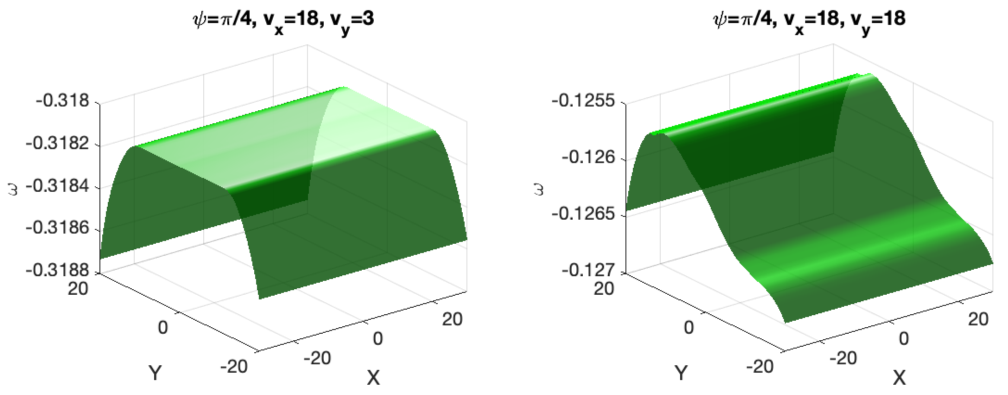


Figure 2.8. 3D slices of (X, Y, ω) from 6D BRT at $s = -2$. Left: slice at $\psi = \pi/4, v_x = 18, v_y = 3$. Right: slice at $\psi = \pi/4, v_x = 18, v_y = 18$

$\{(x, y)| -6 < x < 6, -2 < y < 2)\}$, the smaller set of v_x needs to be avoided to maintain safety. This is because if the agent is far from the unsafe positions, it has more time and space to slow down and adjust v_x. In comparison, in the right plot, the agent has a larger $v_y = 18$, and thus has a larger BRT in v_x to avoid, especially in the Y direction.

Fig. 2.8 shows the BRTs at the slice of $\psi = \pi/4, v_x = 18, v_y = 3$ (left) and 18 (right), indicating the range for ω to avoid on the computation area of X and Y. We get similar results as above: the farther from the target area $\{(x, y)| -6 < x < 6, -2 < y < 2)\}$, the smaller sets of ω one needs to avoid, due to more time and space for adjustment. In the right plot, $v_y = 18$, a larger value; thus more distance is needed to adjust ψ. As a result, the BRT is larger.

In our experiment, it takes 17 minutes to compute the approximated BRT with the decomposition method.

Safety-Preserving Trajectories

In Figs. 2.9 and 2.10, we present the evolution of the BRT over time, and illustrate that trajectories synthesized using Eq. (2.17) are guaranteed safe when starting outside the approximated BRTs, and may enter the targets when starting inside the approximated BRTs.

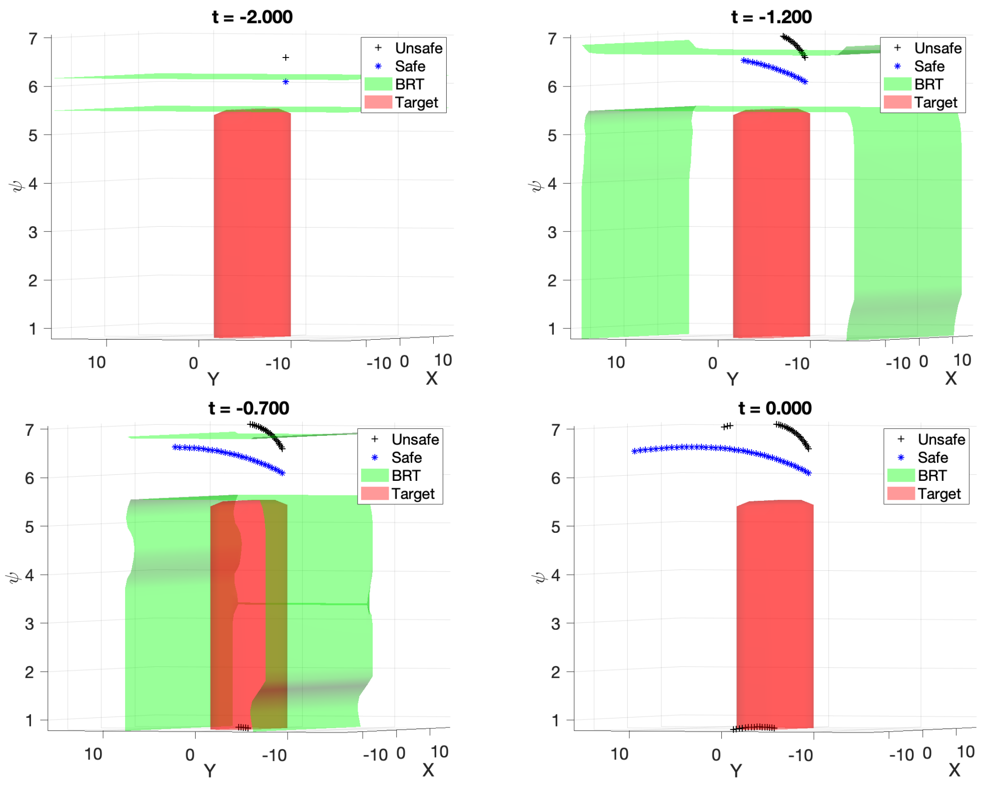


Figure 2.9. Comparison of a safe trajectory (blue) and an unsafe trajectory (black) of 6D Bicycle in (X, Y, ψ) space within the time horizon of $2s$. The safe trajectory starts from outside the BRT (green), and successfully avoids all BRTs and the targets (red) within $2s$. The unsafe trajectory starts from inside the BRT and finally hits the target from bottom.

In Fig. 2.7, we present the trajectories in (X, Y, ψ) space. The safe initial condition (blue) starts from outside the BRT (green), while the unsafe one (black) starts inside. The initial states of $(v_x, v_y, \omega) = (-10, 1, 0.8)$ are the same for both agents. As time moves forward, the blue trajectory can always stay outside of the BRT at the corresponding time, and avoids the target (red) during the time horizon of two seconds. However, the unsafe trajectory enters the target from the bottom of the plot at $s = -0.8$ (the ψ dimension is periodic).

In Fig. 2.8, we present the trajectories in (ω, ψ) space. The initial states of $(X, Y, v_x, v_y) = (10, 0, -6, 1)$ are the same for both agents. In the same setting, the safe trajectory (blue) can stay outside the BRT (green) and the target (red) within time horizon of two seconds, but the unsafe trajectory (black) enters the target from right side at $s = -0.2$.

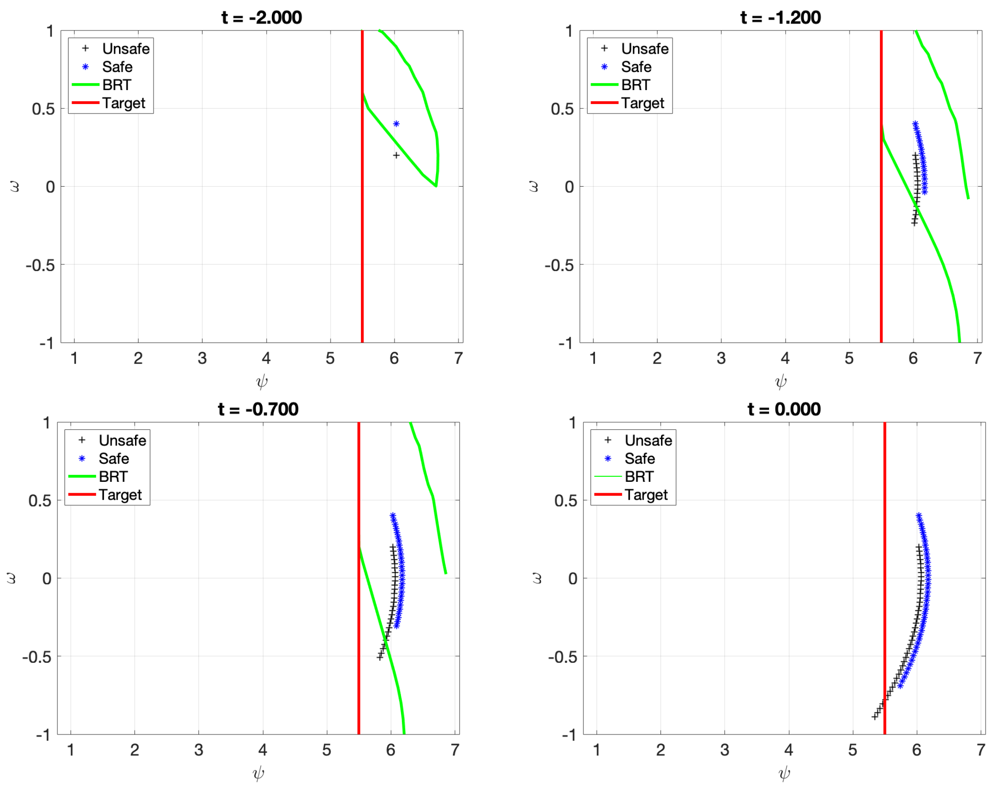


Figure 2.10. Comparison of a safe trajectory (blue) and an unsafe trajectory (black) of 6D Bicycle in (ω, ψ) space within the time horizon of $2s$. The safe trajectory starts from outside of the BRT (green), and successfully avoids all BRTs and the obstacle (red) within two seconds. The unsafe trajectory starts from inside the BRT and finally hits the target from the right side.

2.4 Conclusion

We propose a decomposition method that largely alleviates the computation complexity for approximating minimal BRTs, without introducing much conservatism. Our method has a superior advantage of solving sparse, high-dimensional integrator, and is able to handle a large variety of nonlinear system dynamics. We also provide a simple way of making trade-off between computation space, speed and performance, which will benefit when more computation resources are gained.

In the future, we are interested in investigating techniques that allow under-approximation of BRTs, so that maximal BRT, where the targets are goal states to reach, can be approximated by taking the union of BRTs from subsystems. Besides, we hope to explore more techniques such as in [37] to overcome the constraints for target sets when computing minimal BRTs.

Chapter 3

Generating Robust Supervision for Learning-Based Visual Navigation Using Hamilton-Jacobi Reachability

Autonomous navigation is fundamental to control and robotics. Following the success of deep learning, visual navigation has gained popularity. One appeal of visual navigation – which involves using one or more cameras and computer vision to perceive the environment to reach navigational goals – is that cameras are cheap, light weight, and ubiquitous.

Typically, a geometric map of the environment is used for navigation [23,36,58]; however, real-time map generation can be challenging in texture-less environments or in the presence of transparent, shiny objects, or strong ambient lighting [1]. In contrast, end-to-end (E2E) learning approaches have been used for locomotion [24,28,29,54] and goal-point navigation [25,32,33,49,61] that side-step this explicit map estimation step, but suffer from data inefficiency and lack of robustness [52]. Consequently, a number of papers seek to combine the best of learning with optimal control for high-speed navigation [6,27,38,41,47,53], race-track driving [19,20], and drone racing [30,31]. In particular, [8] combines ideas from optimal control and computer vision by having a convolutional neural network (CNN) predict waypoints instead of control signals, and using optimal control to obtain the control for reaching the waypoints. This hybrid approach greatly improved generalizability: a CNN trained in simulation could be successfully deployed on a real robot without additional training or tuning. However, the inevitable errors in waypoint predictions during the test time could lead to unintended robot trajectories, ultimately resulting in collisions with the obstacles. This is particularly problematic when the robot needs to navigate through cluttered environments or narrow openings, as the error margin in such scenarios is often small.

Contributions: In this chapter, we build on the framework in [8] and propose a novel reachability-based method to generate robust waypoints for supervising the CNN. Our key insight is to model the CNN prediction error as "disturbance" in the system dynamics and generate waypoints that are optimal under the worst-case disturbances, ensuring robustness despite the prediction errors. In the context of other work in the safe learning literature such as [21] and [5], which wrap reachability-

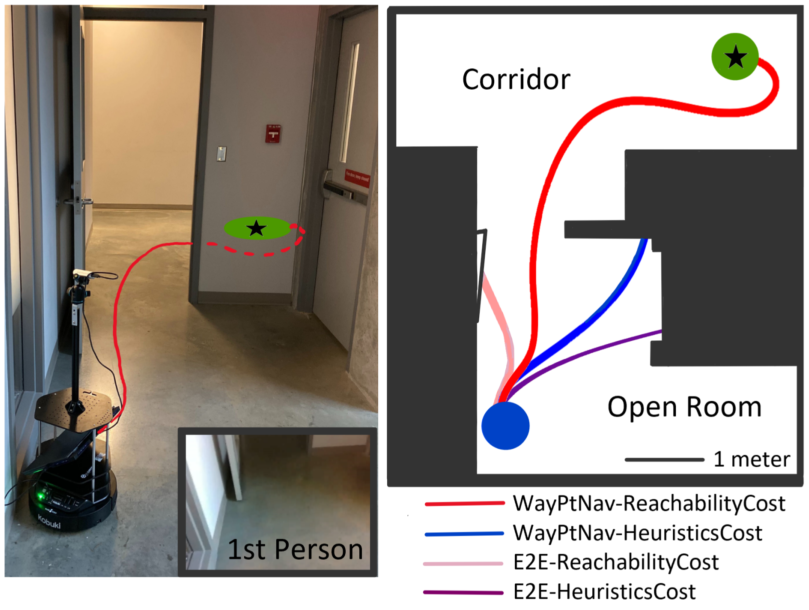


Figure 3.1. Hardware experiment 1. On the left, 3rd and 1st person views of the Turtlebot2 testbed are shown along with a trajectory (red) to the goal (green). On the right, a birds-eye view of the environment is displayed. The Turtlebot's starting location is shown as a blue circle.

based safety controllers around policies being learned, we provide an alternative that pre-emptively uses disturbances when generating training data to alleviate the effect of CNN prediction errors.

Unlike [8], which relies on distance-based heuristics, our method involves computing value functions by solving static Hamilton-Jacobi (HJ) [7] partial differential equations (PDEs). The value functions represent the time until goal-reaching and time until collision despite the worst-case disturbances, given system dynamics and a known environment. These value functions are combined into a cost map that precisely quantifies the quality of waypoints and considers all possible combinations of states by construction. This leads to less greedy navigation behavior and significant improvement in the success rate during the test time.

Overall, our approach leads to less greedy navigation behaviors that are robust to CNN prediction errors. Through simulations and real-world experiments we demonstrate that practical scenarios such as safely moving through narrow doorways become possible with the proposed approach.

3.1 Problem Setup

We consider the problem of autonomous navigation in *a priori* unknown static environment. Starting from an initial position, the robot needs to reach a goal position $p^* = (x^*, y^*)$. We model our ground vehicle as a four-dimensional (4D) system with the following dynamics:

$$\dot{x} = v\cos\psi, \quad \dot{y} = v\sin\psi, \quad \dot{v} = a, \quad \dot{\psi} = \omega\,, \tag{3.1}$$

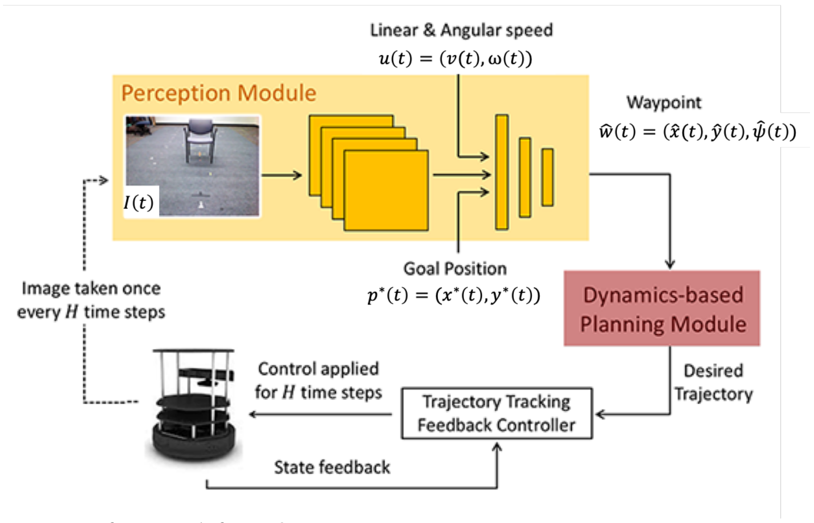

Linear & Angular speed
$u(t) = (v(t), \omega(t))$
Perception Module
$I(t)$
Waypoint
$\hat{w}(t) = (\hat{x}(t), \hat{y}(t), \hat{\psi}(t))$
Goal Position
$p^*(t) = (x^*(t), y^*(t))$
Image taken once every H time steps
Dynamics-based Planning Module
Desired Trajectory
Control applied for H time steps
Trajectory Tracking Feedback Controller
State feedback

Figure 3.2. LB-WayPtNav framework from [8].

where the state $z(t)$ at time t consists of the position $(x(t), y(t))$, speed $v(t)$, and heading $\psi(t)$. The control is acceleration and turn rate, $u(t) := (a(t), \omega(t))$. The robot is equipped with a forward-facing, monocular RGB camera mounted at a fixed height and oriented at a fixed pitch. At time t, the robot receives an RGB image of the environment $\mathcal{E}$, $I(t) = I(\mathcal{E}, z(t))$, the state $z(t)$, and the target position $p^* = (x^*, y^*)$. The objective is to obtain a control policy that uses these inputs to guide the robot to within a certain distance of p^*.

3.2 Background

We build upon the learning-based waypoint approach to navigation (LB-WayPtNav) proposed in [8]. However, unlike LB-WayPtNav, we use a HJ Reachability-based framework to generate supervision data. We now provide a brief overview of LB-WayPtNav and HJ reachability.

3.2.1 LB-WayPtNav

LB-WayPtNav combines a learning-based perception module with a dynamics model-based planning and control module for navigation in *a priori* unknown environments (see Fig. 3.2).

Perception module. The perception module is implemented as a CNN that takes as input a 224×224 pixel RGB image, $I(t)$, captured from the onboard camera, the target position, p^*, specified in the vehicle's current coordinate frame, and vehicle's current linear and angular speed, $(v(t), \omega(t))$, and outputs the desired state or a waypoint $\hat{w}(t) := (\hat{x}(t), \hat{y}(t), \hat{\psi}(t))$.

Planning and control module. Given a waypoint $\hat{w}(t)$, the system dynamics in Eqn. (3.1) are used to plan a spline-based trajectory to $\hat{w}(t)$, starting from the current state of the vehicle. This leads to a smooth, dynamically feasible, and computationally efficient trajectory to the waypoint. An LQR-based feedback controller tracks the trajectory. The commands generated by the LQR controller are executed on the system for H seconds, and then a new image is used to generate another waypoint and trajectory. This process is repeated until the robot is within a certain distance of p^*.

3.2.2 The Time-to-Reach Problem in Reachability Analysis

Consider a dynamical system described by $\dot{z}(t) = f(z(t), u(t), d(t)), z(0) = z_0$, where $z \in \mathbb{R}^n$ is the state, and $u \in \mathcal{U}, d \in \mathcal{D}$ are the control and disturbance respectively. In this chapter, the control represents the actions a robot can take to change its state, and the disturbance primarily models CNN prediction errors. Following [60] and [56], we define the time-to-reach (TTR) value function denoted $V_{\mathrm{R}}(z)$, which represents the minimum time required to drive the system to the goal set Γ_G while avoiding the obstacle set Γ_O, despite disturbances. The control minimizes this time and disturbance maximizes: $V_{\mathrm{R}}(z) = \max_{d(\cdot)\in\mathbb{D}} \min_{u(\cdot)\in\mathbb{U}} \min\{t | z(t) \in \Gamma_G \wedge \forall s \in [0,t], z(s) \notin \Gamma_O\}$. We also define the time-to-collision (TTC) value function, $V_{\mathrm{C}}(z)$, which represents the maximum time until collision with the obstacle set Γ_O assuming that the control is optimally avoiding obstacles under the worst-case disturbance: $V_{\mathrm{C}}(z) = \min_{d(\cdot)\in\mathbb{D}} \max_{u\in\mathbb{U}} \min\{t | z(t) \in \Gamma_O\}$. Applying the dynamic programming principle, we can obtain $V_{\mathrm{R}}(\cdot)$ and $V_{\mathrm{C}}(z)$ respectively as the viscosity solution for the following stationary HJ PDEs:

$$\begin{aligned} &V_{\mathrm{R}}(z) = 0 \text{ in } \Gamma_G, \quad V_{\mathrm{R}}(z) = \infty \text{ in } \Gamma_O, \quad \max_{u\in\mathcal{U}} \min_{d\in\mathcal{D}} \{-\nabla V_{\mathrm{R}}(z)^\top f(z,u,d) - 1\} = 0 \text{ otherwise} \\ &V_{\mathrm{C}}(z) = 0 \text{ in } \Gamma_O, \quad \min_{u\in\mathcal{U}} \max_{d\in\mathcal{D}} \left\{-\nabla V_{\mathrm{C}}(z)^\top f(z,u,d) - 1\right\} = 0 \text{ otherwise} \end{aligned} \tag{3.2}$$

3.3 Reachability-based Supervision for Waypoints

In the perception module presented in Fig. 3.2, from the start position, the robot sequentially observes the image $I(t)$, linear speed $v(t)$ and angular speed $\omega(t)$ at time t to predict a waypoint $\hat{w}$ with a CNN. This waypoint prediction is conducted at a fixed replan frequency until the robot reaches the goal Γ_G, defined to be positions within certain distance to p^*.

To generate supervision for safe and efficient waypoints, we propose a novel reachability expert to autonomously navigate in simulation and collect training data. Specifically, given an obstacle map M_{obs} and a goal area Γ_G, one can compute corresponding TTR and TTC value maps, which are integrated in the cost function of a model predictive control (MPC) optimization problem. By solving this MPC problem, the optimal waypoint $\hat{w}$ is obtained, and at time t, the image $I(t)$ is rendered, and linear and angular speed $\{v(t), \omega(t)\}$ are measured in simulation environment. Finally, we repeat the above procedure in different navigation tasks until sufficient data-label pairs $\{(I, v, \omega), \hat{w}\}$ are obtained for the training dataset. The entire procedure is illustrated in Fig. 3.3.

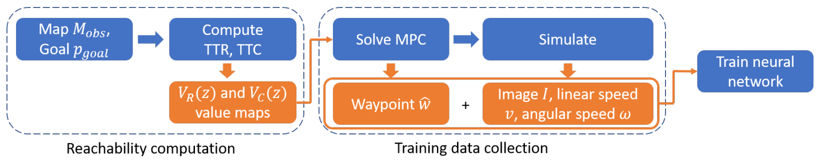


Figure 3.3. Workflow of training data generation using reachability expert.

3.3.1 TTR and TTC Computations

We add disturbances to Eqn. (3.1) to describe the expert's system dynamics

$$\dot{x} = v\cos\psi + d_x, \quad \dot{y} = v\sin\psi + d_y, \quad \dot{v} = a, \quad \dot{\psi} = \omega \ + d_\psi, \tag{3.3}$$

$$v \in [0, \bar{v}], \quad a \in [-\bar{a}, \bar{a}], \quad \omega \in [-\bar{\omega}, \bar{\omega}], \quad d_x^2 + d_y^2 \leq \bar{d}_{xy}^2, \quad d_\psi \in [-\bar{d}_\psi, \bar{d}_\psi] \tag{3.4}$$

with states $z = (x, y, v, \psi)$, controls $u = (a, \omega)$ and disturbances $d = (d_x, d_y, d_\psi)$. Also, $\bar{a}$, $\bar{\omega}$, and $\bar{d}_\psi$ are upper bounds for acceleration, angular speed and disturbances in turn rate, and $\bar{d}_{xy}$ is the circular upper bound for disturbances in x and y components of speed.

For every navigation task, we initialize a goal position p^* on an known obstacle map M_{obs}. We define the goal area Γ_G to be $\Gamma_G = \{(x, y, \psi, v) : \sqrt{(x-p^*)^2 + (y-p^*)^2} \leq c\}$, and the map M_{obs} as Γ_O. $V_{\mathrm{R}}(z)$ and $V_{\mathrm{C}}(z)$ are then computed based on Eqn. (3.2). $V_{\mathrm{R}}(z)$ and $V_{\mathrm{C}}(z)$ guide the reachability expert in a 4D state space for goal reaching and collision avoidance.

Incorporating worst-case disturbances leads to $V_{\mathrm{R}}(z)$ and $V_{\mathrm{C}}(z)$ having more conservative values, which results in less greedy expert trajectories in a nuanced manner. Greediness often makes the robot incapable of going around obstacles and having a view of the environment that informs the robot about possible routes to the goal. Crucially, the conservatively safe trajectories address the prediction errors from neural networks, since a minor deviation will not lead to collision. Note that M_{obs} is assumed to be known only during training; no such assumption is made during the test time, during which the robot only relies on onboard sensors for navigation.

3.3.2 Waypoint Supervision Generation

We use an MPC framework to generate waypoints and expert trajectories. To achieve efficient and safe navigation, we trade off between reaching the goal faster and staying further from obstacles. Thus we design a novel cost function, ReachabilityCost J, to be a combination of TTR and TTC:

$$J(z) = V_{\mathrm{R}}(z) + \alpha(\bar{V}_{\mathrm{C}} - V_{\mathrm{C}}(z)), \tag{3.5}$$

where $\bar{V}_{\mathrm{C}}$ is the computational upper bound for TTC, and α is the scale factor.

The expert uses MPC to plan, in a receding horizon manner, a trajectory of time horizon H until the goal is reached. Starting from $t = 0$, in every MPC optimization problem, we discretize the time horizon $[t, t + H]$ to be $\{t_i | t + i\Delta t, i \in \{0, 1, ..., N\}\}$. At any time index i , we denote $z^{[i]} := z(t_i)$

and $J^{[i]} := J(t_i)$. Then the following MPC problem is sequentially solved in $[t, t+H]$:

$$\underset{\substack{z^{[0]},\ldots,z^{[N]},\\ u^{[0]},\ldots,u^{[N]}}}{\text{minimize}} \sum_{i=0}^{N} J^{[i]}(z^{[i]})$$

$$\begin{aligned}\text{subject to } & x^{[i+1]} = x^{[i]} + \Delta t v^{[i]} \cos\psi^{[i]}, \quad y^{[i+1]} = y^{[i]} + \Delta t v^{[i]} \sin\psi^{[i]}, \quad \psi^{[i+1]} = \psi^{[i]} + \Delta t w^{[i]},\\ & v^{[i+1]} = v^{[i]} + \Delta t a^{[i]}, \quad v^{[i]} \in [0, \bar{v}], \quad \omega^{[i]} \in [-\bar{\omega}, \bar{\omega}],\\ & z^{[0]} = z(t), \quad z^{[N]} = z(t+H), \quad u^{[0]} = u(t), \quad u^{[N]} = u(t+H)\end{aligned} \tag{3.6}$$

To solve Eqn. (3.6) in $[t, t+H]$, the reachability expert first samples a local waypoint grid $\hat{\boldsymbol{w}}$ in the heading direction as possible final states $z^{[N]}$: $\hat{\boldsymbol{w}} := (\hat{\boldsymbol{x}}^{[N]}, \hat{\boldsymbol{y}}^{[N]}, \hat{\boldsymbol{\psi}}^{[N]})$, and compute dynamically feasible spline trajectories to each waypoint using differential flatness of Eqn. (3.3) ([59]). Next, the expert filters out the invalid waypoints whose trajectory violates control constraints. Finally, the solution trajectory $\boldsymbol{z}$ with the minimum cost is chosen, and the corresponding waypoint $\hat{w} = (\hat{x}^{[N]}, \hat{y}^{[N]}, \hat{\psi}^{[N]})$ is added to the training data set along with the image $I(t)$ and speeds $v(t), \omega(t)$ at time t. By solving the MPC problem many times, we obtain the expert dataset $S = \{(I_k(t), v_k(t), \omega_k(t)), (\hat{x}_k^{[N]}, \hat{y}_k^{[N]}, \hat{\psi}_k^{[N]})\}_{k=1}^{M}$, where k is the index of the MPC problem, and M is the total number of data points (and the total number of MPC problems solved).

3.4 Summary of Simulation Results

With the generated expert dataset S, we train a CNN that implicitly learns how to predict good waypoints given the current input image and robot's system dynamics. Then, we test our model in a novel environment in simulation without an *a priori* known map.

Dataset: We use Stanford large-scale 3D Indoor Spaces dataset [4] as our simulation environment, which are 3D scans of real world buildings. Two buildings are used for data generation and training; the 3rd *held-out* building is used as the test environment, which has significant differences in the object appearance and layout. For navigation tasks in training and testing, we sample various start and goal positions that require the robot to go through narrow openings.

Implementation details: We train the CNN in Fig. 3.2 with 150k data points from the reachability expert, $M = 150$k. The mean squared error loss is used and optimized using the Adaptive Moment Estimation (ADAM) algorithm with a learning rate of 10^{-4} and weight decay of 10^{-6}.

Metrics: We use both statistics and trajectory plots to present the test results. For statistics, we use success rate, average time to reach the goal area (for successful tasks), acceleration and jerk to measure the quality of trajectories. With trajectory plots, we analyze the robot's specific behaviors.

Baselines: We compare our approach with the HeuristicsCost designed in [8] for the MPC framework:

$$J^{heuristic}(z) := (\max\{0, \lambda_1 - d^{obs}(x,y)\})^3 + \lambda_2 (d^{goal}(x,y))^2 \tag{3.7}$$

Table 3.1. **Quantitative Comparisons in Simulation:** We compute four metrics-success rate, average time to reach the goal, acceleration and jerk-on 200 navigation tasks with a replan frequency of 4Hz. The proposed method, WayPtNav-ReachabilityCost, is most successful at completing novel navigation tasks. Without the disturbances incorporated in the dynamics, ReachabilityCost takes the shortest time to reach the goal, but the success rate largely drops because of prediction errors. For E2E learning, the success rate is generally lower and the trajectories are less smooth (indicated by high average jerk).

Agent	**Success (%)**	**Time taken (s)**	**Acceleration (m/s^2)**	**Jerk (m/s^3)**
WayPtNav-ReachabilityCost	**63.82**	21.00 ±8.00	**0.06 ±0.01**	**0.94 ±0.13**
WayPtNav-HeuristicsCost	52.26	18.82 ±5.66	0.07 ±0.02	1.06 ±0.15
WayPtNav-ReachabilityCost-NoDstb	49.24	**16.19 ±4.8**	0.07 ±0.01	0.98 ±0.16
E2E-ReachabilityCost	8.04	19.55 ±4.72	0.07 ±0.01	2.16 ±0.30
E2E-HeuristicsCost	31.66	25.56 ±9.85	0.26 ±0.06	9.06 ±1.94

where $d^{obs}(x, y)$ is the distance to the nearest obstacle, $d^{goal}(x, y)$ is the distance to the goal, and λ_1 and λ_2 are scaling factors. We compare our ReachabilityCost in Eqn. (3.5) to the HeuristicsCost in Eqn. (3.7) for two different navigation frameworks: waypoint navigation (WayPtNav) and end-to-end (E2E) learning, resulting in a total of 4 methods. WayPtNav maps the current image and controls inputs to the waypoint (as in Fig. 3.2), where E2E learning directly outputs the control commands given the same inputs. We also compare against an additional baseline, ReachabilityCost-NoDstb, that does not incorporate disturbances in the system dynamics during the data generation.

3.4.1 Expert Performance

We select $\bar{v} = 0.6$ m/s, $\bar{\omega} = 1.1$ rad/s, and $\bar{a} = 0.4$ m/s^2 to match the specifications of the Turtlebot 2 used in the hardware experiments (Sec. 3.6). We set $\bar{d}_{xy} = 0.05$ m/s and $\bar{d}_{\psi} = 0.15$ rad/s to account for prediction errors, and $\alpha = 30$ to prioritize collision avoidance. All expert trajectories are generated according to Section 3.3.2, where the replanning is done every 1.5s to collect training data. In Fig. 3.6 (a) to (c), we compare the expert trajectories obtained by HeuristicsCost and ReachabilityCost. The reachability expert uses the full system dynamics for optimizing waypoints. As a result, it maintains an appropriate orientation and speed when going through the narrow openings. Moreover, due to the presence of disturbances, it takes a conservative path, always staying near the middle of narrow openings, resulting in collision-free trajetcories even when there is prediction error. In contrast, HeuristicsCost takes a greedier path to approach the goal. To address CNN prediction error, [8] use an obstacle padding which makes narrow openings impossible to enter.

3.4.2 Test Results

We compare the different methods in Table 3.1. WayPtNav-ReachabilityCost achieves the highest success rate and least acceleration and jerk. WayPtNav-ReachabilityCost-NoDstb takes the shortest time to reach the goal, but experiences a notable drop on the success rate.

E2E learning results in lower success rate and more jerky trajectories for both methods. Notably, ReachabilityCost has a significant lower success rate with E2E learning compared to WayPtNav.

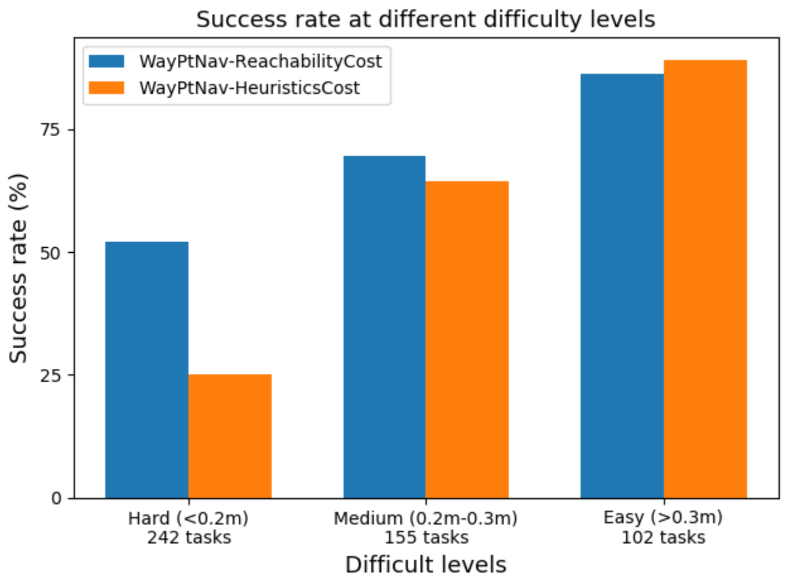


Figure 3.4. Success rate in tasks of different difficulties. Difficulties are assessed according to minimum distance to obstacles d_{min} along trajectories, divided into Hard ($d_{min} < 0.2$ m, 242 tasks), Medium (0.2 m $\leq d_{min} \leq$ 0.3 m, 155 tasks) and Easy ($d_{min} > 0.3$ m, 102 tasks). ReachabilityCost has significant advantage in Hard tasks.

3.5 Analysis of Simulation Results

3.5.1 Comparison with WayPtNav-HeuristicsCost

As discussed in Sec. 3.4.1, the reachability expert is less greedy and more robust, which enables it to navigate through cluttered environments and narrow openings. This results in a higher success rate during test time, as shown in Fig. 3.4, where we compare the two methods on 500 navigation tasks with varying difficulties. Here, we measure the difficulty of a task by the opening size that the robot must navigate through on its way to the goal. ReachabilityCost has much higher success rate on the "Hard" level, indicating that it makes robot more adept at maneuvering in narrow environments.

3.5.2 Effect of Replanning Frequency

Replanning frequency represents how often the robot predicts new waypoints. Fig. 3.5 shows the success rate on 200 navigation tasks for 5 different replanning frequencies: 0.67 Hz, 1 Hz, 2 Hz, 4 Hz and 6.67 Hz. As we increase the frequency, the success rate improves and reaches the peak at 4 Hz for both experts. Above 4 Hz, the success rate drops dramatically.

In general, a higher replanning frequency helps the learning system react faster to the unknown environment; however, if it is too high, the lack of visual memory in the WayPtNav framework results in myopic decisions, leading to a drop in the success rate. ReachabilityCost benefits more from the higher replanning frequencies compared to HeuristicsCost, as the greedy behaviors from the latter tend to drive the robot to cut corners, often leading to situations that are hard to recover from.

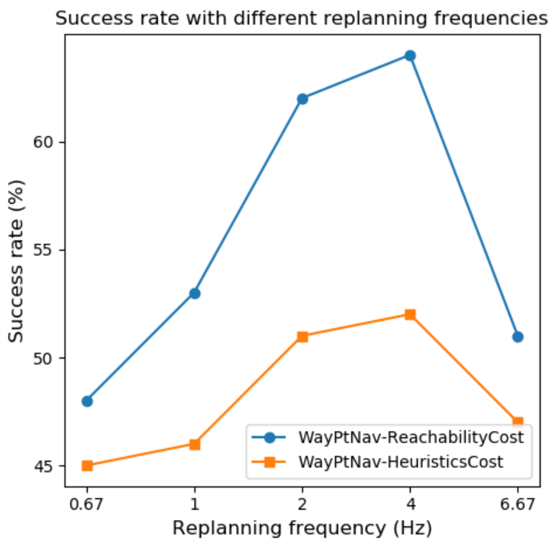


Figure 3.5. Success rates improve for both methods as the replanning frequency is increased (up to 4 Hz). ReachabilityCost benefits considerably more with higher replanning frequencies.

3.5.3 Effect of Adding Disturbances

Adding disturbances in the dynamics is crucial for improving test performance: it not only accounts for dynamics uncertainties, but also models neural network prediction errors. Without disturbances, trajectory can be "too optimal" so that a little deviation of robot's state or minor errors from the CNN results in collision. This is evident from Table 3.1, where the success rate drops from 63.82% to 49.24% in the absence of disturbances.

We examine reachability expert trajectories with and without the disturbances in Fig. 3.6 (b) and (c). Without disturbances, the expert chooses a path close to the obstacles, while with disturbances, the expert stays near the middle of the road. Although both experts succeed in reaching the goal, disturbances lead to more robust trajectories, which translate to test time as shown in Fig. 3.6 (d) and (e)). Without disturbances, the robot tries to avoid the wall but fails, while with disturbances, the robot is able to stay in the middle of the opening, and pass through a very narrow doorway.

3.5.4 Comparison with E2E Learning

Our conclusions here are consistent with the findings in [8] – the model-based approach (WayPtNav) leads to a higher success rate and significantly smoother trajectories. The success rate of E2E learning declines further for the reachability expert as the control profiles are even more nuanced, making it challenging to learn them.

3.5.5 Failure Cases

Since we do not construct a map or have any visual memory in this framework, the robot struggles in the navigation scenarios where it needs to "backtrack". In addition, when the room layout is too different from the training time, the CNN fails to predict good waypoints.

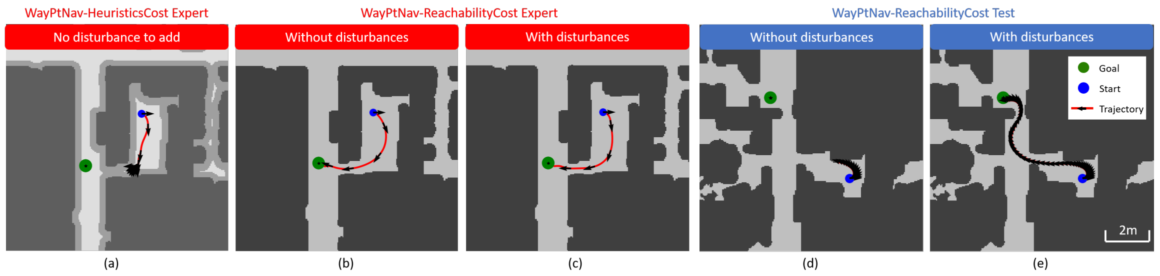


Figure 3.6. Trajectory comparison. (a), (b), and (c) are expert trajectories; (d) and (e) are test trajectories. In (a), the baseline expert starts with a greedier path and fails to enter the narrow opening due to hard obstacle padding (medium grey) used during waypoint optimization. Comparing (b) and (c), the reachability expert can safely go through the openings, but is more likely to stay in the middle of the road with disturbances incorporated in dynamics. This ability is transferred to test times in (d) and (e), where the robot with disturbances added is more resistant to the prediction errors from the neural network and manages to take a collision free path to the goal.

3.6 Hardware Experiments

We tested our framework on a Turtlebot 2 hardware testbed (Fig. 3.1), using monocular RGB images from the onboard camera for navigation. We tested the ReachabilityCost and HeuristicsCost CNNs directly on the Turtlebot without any additional training or fine-tuning.

Experiments were carried out in 3 separate areas of Simon Fraser University, each of which were absent from the training set. An outline of each experiment and the trajectories taken by the Turtlebot with each CNN are shown in Figures 3.1 and 3.7. Video footage of all experiments can be found at https://www.youtube.com/playlist?list=PLUBop1d3Zm2uDGGfGrjWiSjrSlzo5vWMs.

Each scenario required the Turtlebot to traverse narrow spaces and doorways, both of which have shown a low success rate for the HeuristicsCost CNN. For the first scenario, the Turtlebot maneuvered through a narrow doorway, and then around the nook behind the door to reach the goal (Fig. 3.1). The second and third scenarios required the Turtlebot to move from an open room into a narrow corridor and from a cluttered environment into a hallway, respectively (Fig. 3.7).

For each scenario, the HeuristicsCost CNN was unable to maneuver through the doorways, and collided with the wall at full speed, while the ReachabilityCost CNN successfully navigated through the doors and reached the goal. The ability to navigate through narrow environments and doorways is a key improvement seen in the ReachabilityCost CNN. For both methods, the neural networks trained using end-to-end learning were unable to reach the goal.

3.7 Conclusion

In this chapter, we present a novel method to generate training data for waypoint prediction in visual navigation, using reachability analysis. Our method helps the neural network to learn to predict efficient and safe waypoints given system dynamics and observations of the environment. We also use disturbances in dynamics to model neural network prediction errors and greatly enhance its robustness. Simulation and real robot experiments demonstrate higher success rate and smoother

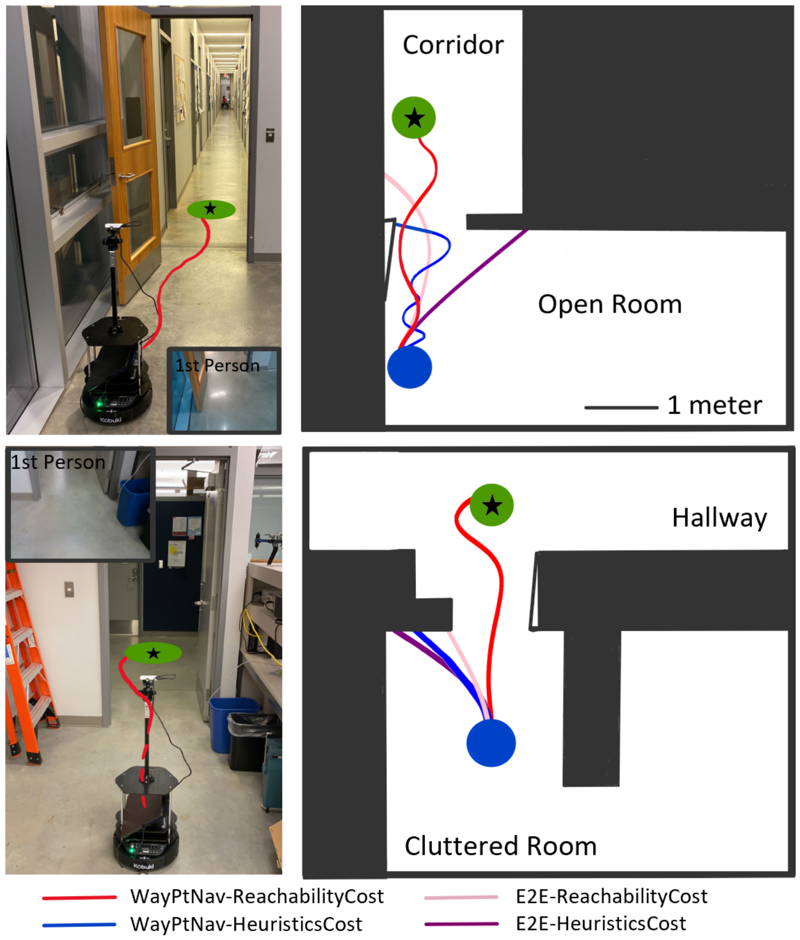


Figure 3.7. Experimental scenarios 2 (top) and 3 (bottom), shown in 3rd and 1st person along with birds-eye view. The goal is marked in green, and the trajectory taken by the ReachabilityCost CNN to the goal is shown in red. The Turtlebot's starting location is shown as a blue circle.

trajectories in navigation tasks with our method, which crucially enables the robot to pass through narrow passages such as doorways. Immediate future work includes adding memory in our navigation framework, investigating the data mismatch between training and test scenarios to better transfer the expert performance in test time, and incorporating human agents.

Chapter 4

Conclusion

To enable safe operation of autonomous systems in the real world, one often requires both effective perception of the environment and robust control of the robot. Hamilton-Jacobi Reachability is a widely used optimal control tool to verify robotic safety. In this thesis, we first work on improving HJ Reachability theory to address the “curse of dimensionality” via state dependency-based system decomposition. Then we apply the HJ Reachability theory to learning-based visual navigation and obtain robust and safe control for robots despite CNN’s prediction errors.

In the first work, we propose a novel system decomposition techniques to reduce the computation complexity of HJ Reachability. By exploiting state dependency information in dynamics and treating missing states as disturbances in subsystems, the approximate reachability maintains the guaranteed-safe property without introducing much conservatism. Both formal mathematical proof and numerical examples are provided for the correctness and efficiency of our method. We also provide the first HJ Reachability analysis of 6D bicycle model which was previously considered intractable.

In the second work, we propose a novel reachability-based method to generate waypoint supervision that better considers system dynamics information than [8]. A novel cost function is designed for waypoint evaluation that combines time-to-reach and time-to-collision functions via solving HJ partial differential equations. By adding disturbances to the robot’s dynamics to address CNN’s prediction error, we enhance the robustness of the learning framework and ensure safe navigation of robot in narrow spaces.

Disturbances play significant roles in both work, but in different ways. In the first work, robot safety is guaranteed via considering worst-case disturbances in system dynamics. In addition, missing states are treated as disturbance to secure the over-approximation of BRT. In the second work, we model the disturbances in system dynamics as CNN’s prediction error, which results in more conservative behaviours in training data, e.g. staying in the middle of the road, and thus leads to robust control in test time.

In summary, this thesis investigates HJ Reachability-based method for robotic safety. The proposed work reduces the computation complexity in HJ Reachability theory, and combines HJ Reachability with deep learning to enable robust visual navigation on real robots.

Bibliography

[1] Faraj Alhwarin, Alexander Ferrein, and Ingrid Scholl. IR stereo kinect: improving depth images by combining structured light with IR stereo. In *PRICAI*, 2014.

[2] Matthias Althoff and Bruce H Krogh. Reachability analysis of nonlinear differential-algebraic systems. *IEEE Transactions on Automatic Control*, 59(2):371–383, 2013.

[3] Matthias Althoff, Olaf Stursberg, and Martin Buss. Computing reachable sets of hybrid systems using a combination of zonotopes and polytopes. *Nonlinear analysis: HHybrid Systems*, 4(2):233–249, 2010.

[4] Iro Armeni, Ozan Sener, Amir R Zamir, Helen Jiang, Ioannis Brilakis, Martin Fischer, and Silvio Savarese. 3d semantic parsing of large-scale indoor spaces. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[5] Andrea Bajcsy, Somil Bansal, Eli Bronstein, Varun Tolani, and Claire J. Tomlin. An Efficient Reachability-Based Framework for Provably Safe Autonomous Navigation in Unknown Environments. In *Proc. Conf. Decision and Control*, 2019.

[6] Mayank Bansal, Alex Krizhevsky, and Abhijit Ogale. Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst. *arXiv preprint arXiv:1812.03079*, 2018.

[7] Somil Bansal, Mo Chen, Sylvia Herbert, and Claire J. Tomlin. Hamilton-Jacobi reachability: A brief overview and recent advances. In *Proc. IEEE Conf. on Decision and Control*, 2017.

[8] Somil Bansal, Varun Tolani, Saurabh Gupta, Jitendra Malik, and Claire Tomlin. Combining optimal control and learning for visual navigation in novel environments. *arXiv preprint arXiv:1903.02531*, 2019.

[9] EN Barron. Differential games with maximum cost. *Nonlinear analysis: Theory, methods & applications*, 14(11):971–989, 1990.

[10] Olivier Bokanowski and Hasnaa Zidani. Minimal time problems with moving targets and obstacles. *IFAC Proceedings Volumes*, 44(1):2589–2593, 2011.

[11] Mo Chen, Sylvia Herbert, and Claire J Tomlin. Fast reachable set approximations via state decoupling disturbances. In *Proc. IEEE Conf. on Decision and Control*. IEEE, 2016.

[12] Mo Chen, Sylvia Herbert, Mahesh Vashishtha, Somil Bansal, and Claire Tomlin. Decomposition of reachable sets and tubes for a class of nonlinear systems. *IEEE Trans. Autom. Control*, 63(11):3675–3688, 2018.

[13] 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 J. Guidance, Control, and Dynamics*, 40(6):1360–1373, 2017.

[14] Mo Chen, Qie Hu, Casey Mackin, Jaime F Fisac, and Claire J Tomlin. Safe platooning of unmanned aerial vehicles via reachability. In *Proc. IEEE Conf. Decision and Control*, 2015.

[15] Mo Chen, Jennifer C Shih, and Claire J Tomlin. Multi-vehicle collision avoidance via hamilton-jacobi reachability and mixed integer programming. In *Proc. IEEE Conf. on Decision and Control*, 2016.

[16] Mo Chen and Claire J Tomlin. Hamilton–jacobi reachability: Some recent theoretical advances and applications in unmanned airspace management. *Annual Review of Control, Robotics, and Autonomous Systems*, 1:333–358, 2018.

[17] Jérôme Darbon and Stanley Osher. Algorithms for overcoming the curse of dimensionality for certain hamilton–jacobi equations arising in control theory and elsewhere. *Research in the Mathematical Sciences*, 3(1):19, 2016.

[18] Aparna Dhinakaran, Mo Chen, Glen Chou, Jennifer C Shih, and Claire J Tomlin. A hybrid framework for multi-vehicle collision avoidance. In *Proc. IEEE Conf. on Decision and Control*, 2017.

[19] Paul Drews, Grady Williams, Brian Goldfain, Evangelos A. Theodorou, and James M. Rehg. Aggressive deep driving: Combining convolutional neural networks and model predictive control. In *CoRL*, 2017.

[20] Paul Drews, Grady Williams, Brian Goldfain, Evangelos A Theodorou, and James M Rehg. Vision-based high-speed driving with a deep dynamic observer. *IEEE Robotics and Automation Letters*, 2019.

[21] Jaime F. Fisac, Anayo K. Akametalu, Melanie N. Zeilinger, Shahab Kaynama, Jeremy Gillula, and Claire J. Tomlin. A General Safety Framework for Learning-Based Control in Uncertain Robotic Systems. *IEEE Transactions on Automatic Control*, 64(7):2737–2752, Jul. 2019.

[22] 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 *Int. Conf. on Computer Aided Verification*, 2011.

[23] J. Fuentes-Pacheco, J. Ruiz-Ascencio, and J. M. Rendón-Mancha. Visual simultaneous localization and mapping: a survey. *Artificial Intelligence Review*, 2015.

[24] Dhiraj Gandhi, Lerrel Pinto, and Abhinav Gupta. Learning to fly by crashing. In *IROS*, 2017.

[25] Saurabh Gupta, Varun Tolani, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. Cognitive mapping and planning for visual navigation. *arXiv preprint arXiv:1702.03920*, 2017.

[26] 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 *Proc. IEEE Conf. on Decision and Control*, 2017.

[27] Sunggoo Jung, Sunyou Hwang, Heemin Shin, and David Hyunchul Shim. Perception, guidance, and navigation for indoor autonomous drone racing using deep learning. *IEEE Robotics and Automation Letters*, 2018.

[28] Gregory Kahn, Adam Villaflor, Vitchyr Pong, Pieter Abbeel, and Sergey Levine. Uncertainty-aware reinforcement learning for collision avoidance. *arXiv preprint arXiv:1702.01182*, 2017.

[29] Katie Kang, Suneel Belkhale, Gregory Kahn, Pieter Abbeel, and Sergey Levine. Generalization through simulation: Integrating simulated and real data into deep reinforcement learning for vision-based autonomous flight. *arXiv preprint arXiv:1902.03701*, 2019.

[30] Elia Kaufmann, Mathias Gehrig, Philipp Foehn, René Ranftl, Alexey Dosovitskiy, Vladlen Koltun, and Davide Scaramuzza. Beauty and the beast: Optimal methods meet learning for drone racing. In *ICRA*, 2019.

[31] Elia Kaufmann, Antonio Loquercio, Rene Ranftl, Alexey Dosovitskiy, Vladlen Koltun, and Davide Scaramuzza. Deep drone racing: Learning agile flight in dynamic environments. In *CoRL*, 2018.

[32] Arbaaz Khan, Clark Zhang, Nikolay Atanasov, Konstantinos Karydis, Vijay Kumar, and Daniel D Lee. Memory augmented control networks. In *ICLR*, 2018.

[33] Dong Ki Kim and Tsuhan Chen. Deep neural network for real-time autonomous indoor navigation. *arXiv preprint arXiv:1511.04668*, 2015.

[34] Alexander B Kurzhanski and Pravin Varaiya. On ellipsoidal techniques for reachability analysis. part ii: Internal approximations box-valued constraints. *Optimization methods and software*, 17(2):207–237, 2002.

[35] Benoit Landry, Mo Chen, Scott Hemley, and Marco Pavone. Reach-avoid problems via sum-or-squares optimization and dynamic programming. In *2018 IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 2018.

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

[37] Donggun Lee, Mo Chen, and Claire J Tomlin. Removing leaking corners to reduce dimensionality in hamilton-jacobi reachability. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 9320–9326. IEEE, 2019.

[38] Antonio Loquercio, Ana I Maqueda, Carlos R del Blanco, and Davide Scaramuzza. Dronet: Learning to fly by driving. *IEEE Robotics and Automation Letters*, 3(2):1088–1095, 2018.

[39] John N Maidens, Shahab Kaynama, Ian M Mitchell, Meeko MK Oishi, and Guy A Dumont. Lagrangian methods for approximating the viability kernel in high-dimensional systems. *Automatica*, 49(7):2017–2029, 2013.

[40] Kostas Margellos and John Lygeros. Hamilton–jacobi formulation for reach–avoid differential games. *IEEE Trans. Autom. control*, 56(8):1849–1861, 2011.

[41] Xiangyun Meng, Nathan Ratliff, Yu Xiang, and Dieter Fox. Neural autonomous navigation with Riemannian motion policy. *arXiv preprint arXiv:1904.01762*, 2019.

[42] Ian Mitchell and Claire J Tomlin. Level set methods for computation in hybrid systems. In *Int. Workshop on Hybrid Systems: Computation and Control*, pages 310–323, 2000.

[43] Ian M Mitchell. A toolbox of level set methods, 2009.

[44] Ian M Mitchell. Scalable calculation of reach sets and tubes for nonlinear systems with terminal integrators: a mixed implicit explicit formulation. In *Proc. ACM Int. Conf. on Hybrid systems: computation and control*, 2011.

[45] Ian M Mitchell, Alexandre M Bayen, and Claire J Tomlin. A time-dependent hamilton-jacobi formulation of reachable sets for continuous dynamic games. *IEEE Trans. Autom. control*, 50(7):947–957, 2005.

[46] Ian M Mitchell and Claire J Tomlin. Overapproximating reachable sets by hamilton-jacobi projections. *J. Scientific Computing*, 19(1-3):323–346, 2003.

[47] Matthias Müller, Alexey Dosovitskiy, Bernard Ghanem, and Vladen Koltun. Driving policy transfer via modularity and abstraction. *arXiv preprint arXiv:1804.09364*, 2018.

[48] S Osher, R Fedkiw, and K Piechor. Level set methods and dynamic implicit surfaces, 2004.

[49] Yunpeng Pan, Ching-An Cheng, Kamil Saigol, Keuntaek Lee, Xinyan Yan, Evangelos Theodorou, and Byron Boots. Agile off-road autonomous driving using end-to-end deep imitation learning. In *RSS*, 2018.

[50] Céline Parzani and Stéphane Puechmorel. On a hamilton-jacobi-bellman approach for coordinated optimal aircraft trajectories planning. *Optimal Control Applications and Methods*, 39(2):933–948, 2018.

[51] Rajesh Rajamani. *Vehicle dynamics and control*. Springer Science & Business Media, 2011.

[52] Benjamin Recht. A tour of reinforcement learning: The view from continuous control. *Annual Review of Control, Robotics, and Autonomous Systems*, 2018.

[53] Charles Richter, William Vega-Brown, and Nicholas Roy. Bayesian learning for safe high-speed navigation in unknown environments. In *Robotics Research*, pages 325–341. Springer, 2018.

[54] Fereshteh Sadeghi and Sergey Levine. (CAD)^{2}RL: Real single-image flight without a single real image. In *RSS*, 2017.

[55] Sumeet Singh, Mo Chen, Sylvia L Herbert, Claire J Tomlin, and Marco Pavone. Robust tracking with model mismatch for fast and safe planning: an sos optimization approach. *arXiv preprint arXiv:1808.00649*, 2018.

[56] R. Takei, R. Tsai, Haochong Shen, and Y. Landa. A practical path-planning algorithm for a simple car: a Hamilton-Jacobi approach. In *Proc. American Control Conference*, 2010.

[57] Ken Tanabe and Mo Chen. Beacls library, 2019.

[58] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005.

[59] Rahee Walambe, Nipun Agarwal, Swagatu Kale, and Vrunda Joshi. Optimal trajectory generation for car-type mobile robot using spline interpolation. *IFAC*, 49(1):601 – 606, 2016.

[60] Insoon Yang, Sabine Becker-Weimann, Mina J. Bissell, and Claire J. Tomlin. One-shot computation of reachable sets for differential games. In *Proc. ACM Int. Conf. Hybrid Systems: Computation and Control*, 2013.

[61] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *ICRA*, 2017.