Real-time Robotic Safety Set Blending Schemes

Charles Tang

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2021-61
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2021/EECS-2021-61.html

May 13, 2021

Copyright © 2021, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

Real-time Robotic Safety Set Blending Schemes

by Charles (Chuck) Tang

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

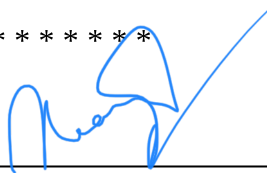
Committee:

Professor Claire Tomlin
Research Advisor

May 11, 2021

(Date)

* * * * * * *



Professor Shankar Sastry
Second Reader

May 12, 2021

(Date)

Real-time Robotic Safety Set Blending Schemes

by

Charles (Chuck) Tang

A thesis submitted in partial satisfaction of the

requirements for the degree of

Master of Science

in

Electrical Engineering and Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Claire Tomlin, Chair
Professor Shankar Sastry

Spring 2021

Real-time Robotic Safety Set Blending Schemes

Copyright 2021
by
Charles (Chuck) Tang

Abstract

Real-time Robotic Safety Set Blending Schemes

by

Charles (Chuck) Tang

Master of Science in Electrical Engineering and Computer Science

University of California, Berkeley

Professor Claire Tomlin, Chair

Modern day robots deployed in the ground or sky have to frequently navigate a priori unknown environments. In these scenarios, robots need to make goal-driven decisions while also satisfying safety constraints imposed by obstacles. Guaranteeing safe operation in unknown environments with limited-range sensors remains a challenging problem for autonomous vehicles. Recent work proposed a Hamilton-Jacobi reachability framework for computing safe operational regions for such scenarios. Unfortunately, controllers synthesized from this framework are jerky and uncomfortable when deployed on the robot. Furthermore, evaluating HJI algorithms in real-time scenarios remains difficult due to the curse of dimensionality. In this work, we implement the BEACLS_ROS toolbox which allows for real time computation of HJI safety sets for robotic navigation tasks. The state-of-the-art algorithms are benchmarked against MATLAB implementations and demonstrate a 6x speedup in computation time. We then explore bang bang control, unconstrained optimization, and constrained optimization formulations to blend the safety set and robot motion planner. We conclude by evaluating each blending scheme's ability to produce smooth, safe, feasible, and goal reaching trajectories for known and unknown environment point-goal navigation tasks.

Contents

Contents i

List of Figures ii

List of Tables iii

1 Introduction 1

2 Background 3
- 2.1 Problem Statement . . . 3
- 2.2 Safe Autonomous Vehicle Navigation . . . 4
- 2.3 Hamilton-Jacobi Reachability . . . 4
- 2.4 Efficient BRS Algorithms . . . 6
- 2.5 Software Tools for BRS . . . 6
- 2.6 Arbitration Policies . . . 8

3 Real-Time BRS Calculations 9
- 3.1 Running Example . . . 9
- 3.2 BEACLS_ROS . . . 10
- 3.3 Results and Discussion . . . 12

4 Blending Schemes for Safety Sets 13
- 4.1 Problem Set Up . . . 13
- 4.2 Blending Schemes . . . 18
- 4.3 Results . . . 23
- 4.4 Discussion . . . 25
- 4.5 Summary . . . 28

5 Conclusion and Future Work 29

Bibliography 30

List of Figures

3.1 A running example of BEACLS_ROS in action. The goal is for the turtlebot to reach the last green way point from the initial position. Big green circles denote the circular sensing region. Blue denotes the robot obstacle. Black is the turtlebot. Red is the safety set calculated by HJIPDE in real time. Note that for this example, we use a 61x61x31 grid and can still evaluate the safety set in real time. 10
3.2 The ROS C++ Data pipeline for BEACLS_ROS. Note that all modules in the BEACLS_ROS repo are colored while all black nodes represent the turtlebot nodes. The robotic navigation stack goes from turtlebot odometry state ⇒ turtlebot controls ⇒ environment occupancy grid update ⇒ beacls safety set ⇒ ROS visualization 11

4.2 Signed Distance Potential Cost Function J 16
4.3 Alpha Functions 22
4.4 **Robot trajectories with different blending schemes:** The robot is moving from the start position in *blue* to the goal position (*black* X). Several different blending schemes are plotted in different colors. Value Alpha (Red): Value Function Alpha (4.10), Sample Alpha (Blue): Sampled Alpha (4.9), Mo and Karen (Orange): Constrained Optimization (4.7), CDC (Pink): Bang Bang Control (2.9). The reach avoid trajectory in green is the HJI solution that we use as an optimal baseline. The pink outline is the BRS V. 24
4.5 **Trajectory Metrics** We logged several metrics of the robot's trajectory including the linear jerk, angular jerk, alpha blend probability, average safety score $V(x)$, distance to goal, and distance to optimal trajectory (reach avoid trajectory) 25

List of Tables

3.1 HJI Speed up in C++ 12

4.1 **Known Environment Results**: The success metric counts the number of point navigation scenarios that reached the goal location x^* within 0.5 meters. The crash metric counts the number of scenarios where the robot crashed into an obstacle. The timeout metric counts the number of scenarios where the robot ran for more than 1000 timesteps. The total number of test scenarios is 100. For the **67** scenarios in which all 4 algorithms succeeded, we measured the jerk, time taken, and number of safety controls each robot took. Jerk is measured in m/s^3. Time Taken is measured in seconds. Num Safety counts the total number of safety controls the robot took to navigate through the environment to the goal. 23

4.2 **Unknown Environment Results**: The success metric counts the number of point navigation scenarios that reached the goal location x^* within 0.5 meters. The crash metric counts the number of scenarios where the robot crashed into an obstacle. The timeout metric counts the number of scenarios where the robot ran for more than 1000 timesteps. The total number of test scenarios is 100. For the **22** scenarios in which all 4 algorithms succeeded, we measured the jerk, time taken, and number of safety controls each robot took. Jerk is measured in m/s^3. Time Taken is measured in seconds. Num Safety counts the total number of safety controls the robot took to navigate through the environment to the goal. 23

4.3 Blending Scheme Properties for Known Environment 26

4.4 Blending Scheme Properties for Unknown Environment 27

Acknowledgments

I would first like to thank my research mentors Andrea Bajcsy and Somil Bansal for guiding me through everything that I have learned in this year of research. Andrea and Somil's invaluable advice in formulating the right research questions and action-oriented approach to devising research experiments was crucial to the insights of this thesis. Their empathetic and insightful feedback pushed me to sharpen my thinking and brought my work to a higher level.

Next, I would like to thank Professor Claire Tomlin for providing a supportive, exciting, and fun lab environment to be in. It's quite rare for seniors to take on new research opportunities but I'm glad Professor Tomlin took a chance on me last year. I would also like to thank Professor Shankar Sastry for being a second reader for this thesis and teaching me fascinating robotics algorithms in his undergraduate class.

Finally, I would like to thank my parents for their unconditional love and support. Without them, none of this could have been possible.

Chapter 1

Introduction

Autonomous navigation is an important problem in robotics that has received significant attention in recent years. Developing a fully autonomous robot that can navigate in a priori unknown environments remains challenging due to several issues such as modelling external disturbances, generating smooth trajectories, and producing safe optimal controls. As a vehicle travels towards a goal and receives new knowledge about the environment, rigorous safety analysis is critical to ensure that the system does not enter into fatal collisions. Balancing this safety analysis with the goal oriented planning algorithm is a challenging task that involves important trade-offs in safety, latency, smoothness, and goal reaching capabilities.

One way to approach this problem is to create a geometrically consistent map used for collision-free, goal oriented motion planning methods. Unfortunately, the real-time generation of accurate maps is computationally expensive and challenging for real-world use cases. Alternative methods suggest using end-to-end deep learning based controllers to side-step this difficult map estimation problem. However, such approaches are both sample inefficient and not interpretable for safety guarantees.

Hamiltonian Jacobi Isaacs (HJI) analysis is another promising research direction which ensures strong safety guarantees for states that the robot can operate in without collision. HJI analysis accounts for both external disturbances of the environment and dynamic modelling errors, with its main drawback being that HJI suffers heavily from computational constraints. Recently, [3] proposed an efficient Hamilton-Jacobi-Isaacs (HJI) reachability framework that allows robots to navigate in *a-priori* unknown static environments in near real-time. To do so, [3] leveraged efficient data structures and warm starting techniques to alleviate the curse of dimensionality that traditional HJI suffers from. This thesis directly builds upon this line of research and re-implements the algorithms in [3] with a C++ framework that can take advantage of GPU speedups. We demonstrate a 6x speed up in comparison to the MATLAB baselines and improve the algorithms in [3] to run in real-time robotic navigation scenarios.

After efficiently computing the safety set, [3] suggest a least restrictive controller that only interferes with the goal reaching planner when the robot is on the safety set boundary. Although this switch-based controller is minimally invasive to the goal reaching capabilities of the robot, the "bang bang" controller produced by this framework is jerky and uncomfortable in practice. Other authors have suggested using the safety sets as a constraint in the optimization-based path planning framework [21]. Although provably safe and goal reaching, such solutions may not always be feasible in real world navigation tasks. In a similar optimization lens, [13, 33] has suggested blending different pieces of robot task information using unconstrained optimization methods, thereby discarding any safety guarantees. In this thesis, we extensively investigate different ways one can formulate the blending problem between safety sets and goal reaching planners. We study bang bang control, constrained optimization, and unconstrained optimization blending schemes and analyze the smoothness, feasibility, efficiency, and safety of each algorithm across a variety of point navigation scenarios.

The rest of this thesis is organized as follows. In Chapter 2, we give an overview of the key equations, toolboxes, and algorithms for efficiently calculating and using safety sets in robotic navigation tasks. In Chapter 3, we describe our software package BEACLS_ROS and demonstrate a robot planning with safety sets in a real time point navigation scenario. In Chapter 4, we explore how to arbitrate the safety set with the original robot planner and analyze the trade offs of different blending schemes. Lastly, in Chapter 5, we discuss future directions of research and summarize our findings. Overall, we describe our contributions as:

- Implementation of the BEACLS_ROS toolbox which enables real-time calculations of safety sets
- Demonstration of a 6x speed up for a robotic navigation task when baselined against MATLAB
- Exploration of different blending schemes between safety sets and original robot planners

Chapter 2

Background

This chapter is based in part on the papers "Efficient Reachability-Based Framework for Provably Safe Autonomous Navigation in Unknown Environments" [3] written by Andrea Bajcsy, Somil Bansal, Eli Bronstein, Varun Tolani, Claire J. Tomlin

2.1 Problem Statement

In this thesis, we study the problem of autonomous vehicle navigation in an indoor environment setting with a-priori unknown obstacles. Let x_0 and x^* denote the start and the goal state of the vehicle. The vehicle aims to navigate from x_0 to x^* in an *a priori* unknown environment, $\mathcal{E}$, whose map or topology is not available to the robot. The dynamics of the system are given by:

$$\begin{aligned} \dot{p}_x = v\cos\phi + d_x, \quad \dot{p}_y = v\sin\phi + d_y, \quad \dot{\phi} = \omega\,, \\ \underline{v} \leq v \leq \bar{v}, \quad |\omega| \leq \bar{\omega}, \quad |d_x|, |d_y| \leq d_r \end{aligned} \tag{2.1}$$

where $x := (p_x, p_y, \phi)$ is the state, $p = (p_x, p_y)$ is the position, ϕ is the heading, and $d = (d_x, d_y)$ is the disturbance experienced by the vehicle. The control of the vehicle is $u := (v, \omega)$, where v is the speed and ω is the turn rate. We assume that odometry is perfect (i.e. the exact vehicle state is available), sensor hits are perfect (i.e. the exact obstacle state is given), and obstacles are static. Working with state estimation, noisy sensors, and dynamic obstacles are important problems of their own right and we defer them for future work.

The vehicle has a sensor which at any given time exposes a region of the state space $\mathcal{S}_t \subset \mathbb{R}^n$, and provides a conservative estimate of the occupancy within $\mathcal{S}_t$. The sensor hits are aggregated over time into the binary occupancy grid $O_t \subset \mathbb{R}^n$, where 1 denotes and 0 marks free space. At any time t and state $x(t)$, the vehicle has a planner $\Pi(x(t), x^*, O_t)$, which outputs the control command $u(t)$ to be applied at time t. The planner replans every model predictive control (MPC) horizon T in order to update the planner trajectory $\xi^{planner}$. We note that $\xi_t := (x_t, u_t)\ \forall\ t \in [0 - T]$. Throughout this work, we test our algorithms with a spline planner [31] in order to generate smooth and continuous blended trajectories.

Given x_0, x^*, the planner Π, and the sensor $\mathcal{S}$, the goal of this thesis is to design a real time control mechanism that smoothly navigates the vehicle to the goal without collisions.

2.2 Safe Autonomous Vehicle Navigation

The point-goal navigation task described above serves as a test bed for autonomous vehicle navigation progress [22, 35]. The main goal of these tasks is to design a collision-free control policy that navigates the robot(s) to a destination goal given a set of potentially noisy sensor inputs.

Recent approaches have suggested using end to end large scale reinforcement learning methods [34, 18, 9] which have achieved state of the art results for environments without external disturbances. Model based control methods such as [7, 27] have successfully incorporated optimal control and model priors to supervise the neural network controller for similar point navigation tasks as well. Despite these neural network advancements, guaranteeing safety for these algorithms remain difficult due to the lack of interpret-ability of a neural network module.

To ensure safety, many current methods use control barrier functions [2, 36] to design stable controllers that navigate around obstacles. However, these methods assume that a recursively feasible collision-free path can be obtained despite the unknown environment, which may not be possible in real-world environments. Many other attempts at incorporating safety in path planning include sum-of-squares [19], linear temporal logic [20], and reactive synthesis approaches [28]. Our framework builds upon more stronger guarantees of safety which accounts for external disturbances and a-priori unknown environments using HJ-reachability analysis.

2.3 Hamilton-Jacobi Reachability

The safety framework that this thesis builds upon is Hamilton Jacobi (HJ) reachability analysis [23, 25]. HJ reachability analysis has been successfully applied in a variety of domains, such as aircraft auto-landing and safe multi-vehicle path planning [5, 4]. In this work, we will be using reachability analysis to compute a backward reachable set (BRS) $\mathcal{V}(\tau)$ given a set of unsafe states $\mathcal{L}$. Intuitively, $\mathcal{V}(\tau)$ is the set of states such that the system trajectories that start from this set can enter $\mathcal{L}$ within a time horizon of τ for some disturbance despite the best control efforts. In contrast, for any trajectory that starts from $\mathcal{V}^c(\tau)$, there exists a control such that the system trajectory will *never* enter $\mathcal{L}$, despite the worst-case disturbance. Here, $\mathcal{V}^c(\tau)$ represents the complement of the set $\mathcal{V}(\tau)$.

The computation of the BRS can be formulated as a differential game between the control and disturbance, which can be solved using the principle of dynamic programming. The cost

functional corresponding to this differential game is given by:

$$J(x,\tau,u(\cdot),d(\cdot)) = \inf_{s\in[\tau,0]} l(\xi^{u,d}_{x,\tau}(s)), \tag{2.2}$$

where $\xi^{u,d}_{x,\tau}(s)$ represents the system state at time s starting from state x at time τ and applying control $u(\cdot)$ with disturbance $d(\cdot)$. In (2.2), the function $l(x)$ is the implicit surface function representing the unsafe set $\mathcal{L} = \{x : l(x) \le 0\}$. Intuitively, J keeps track of whether the system trajectory even entered the unsafe set during the time horizon $[\tau, 0]$, and if so, the cost corresponding to that trajectory is negative.

The value function corresponding to the cost functional in (2.2) is given by:

$$V(\tau,x) = \min_{d(\cdot)\in\Gamma[u]} \max_{u(\cdot)} J(x,\tau,u(\cdot),d(\cdot)), \tag{2.3}$$

where Γ represents the set of non-anticipative strategies [25]. If the value function is negative for a given state, then starting from this state the system cannot avoid entering into the unsafe set eventually. Thus, the value function in (2.3) keeps track of all unsafe trajectories of the system, which in turn can be used to compute the safe trajectories for the system. For further details on this formulation, we refer the interested readers to [25, 4].

The value function in (2.3) can be obtained using dynamic programming, which yields a Hamilton Jacobi-Isaacs Variational Inequality (HJI-VI) [14, 23]. Ultimately, a BRS can be computed by solving the following final value HJI-VI:

$$\begin{aligned} &\min\{D_\tau V(\tau,x) + H(\tau,x,\nabla V(\tau,x)), l(x) - V(\tau,x)\} = 0 \\ &V(0,x) = l(x), \quad \tau \le 0. \end{aligned} \tag{2.4}$$

Here, $D_\tau V(\tau,x)$ and $\nabla V(\tau,x)$ denote the time and space derivatives of the value function $V(\tau,x)$ respectively. The Hamiltonian, $H(\tau,x,\nabla V(\tau,x))$, encodes the role of system dynamics, control, and disturbance, and is given by

$$H(\tau,x,\nabla V(\tau,x)) = \max_{u\in\mathcal{U}_i} \min_{d\in\mathcal{D}} \nabla V(\tau,x)\cdot f(x,u,d). \tag{2.5}$$

Once the value function $V(\tau,x)$ is computed, the BRS, and consequently, the set of safe states are given by

$$\mathcal{V}(\tau) = \{x : V(\tau,x) \le 0\}, \tag{2.6}$$

$$\mathcal{W}(\tau) = \mathcal{V}^c(\tau) = \{x : V(\tau,x) > 0\}. \tag{2.7}$$

HJI reachability also provides the optimal control to keep the system in the safe set and is given by

$$u^*(\tau,x) = \arg\max_{u\in\mathcal{U}} \min_{d\in\mathcal{D}} \nabla V(\tau,x)\cdot f(x,u,d). \tag{2.8}$$

In fact, the system can safely apply any control as long as it is not at the boundary of the unsafe region. If the system reaches the boundary of $\mathcal{V}(\tau)$, the control in (2.8) steers the system away from the unsafe states. This least restrictive controller provided by HJI reachability is also the basis for ensuring safety in a least restrictive fashion that we employ in our framework.

Several challenges come up when one seeks to extend HJ reachability analysis for safe robotic navigation problems. First, calculating the BRS in real-time is difficult due to the curse of dimensionality [8] that HJI-VI suffers from. Second, integrating HJI-VI optimal controls with other planners in a switching fashion [3] creates high jerk trajectories that are uncomfortable for the user. In the next sections, we summarize the most relevant literature that aims to tackle these two issues.

2.4 Efficient BRS Algorithms

There is a long history of prior work that aims to alleviate the curse of dimensionality that comes with solving the HJI equations. Decomposition schemes [10], efficient initializiations [16], offline methods [15], and using neural networks for safety sets [6] are all promising research directions that have effectively sped up the BRS calculation for specific HJI tasks. This thesis builds upon the safety framework in [3] which sped up BRS calculations for indoor robots to near real time applications.

The key intuition in [3] is that the most recent HJI calculation at time $t - \delta$ is locally similar to the new HJI calculation that needs to be done at time t. Thus, one can use the previous safety set $\mathcal{V}^{last}$ to warm start the computation for the current safety set $\mathcal{V}^{new}$. This idea is similar to warm starting in reinforcement learning and transfer learning in computer vision where one initializes the weights of an algorithm with a prior that is close to the final solution. Another speed up introduced in [3] employs a queue based update algorithm to remove unnecessary computations for value function states which remain unchanged during each value function iteration. Our work directly extends these two ideas mentioned above into a faster programming language codebase in C++.

More widely accessible and well-maintained software packages have been a key enabler to the algorithmic advances presented above. We next summarize the relevant software packages for solving the HJI equations that provide performance improvements in software and hardware.

2.5 Software Tools for BRS

Level Set Toolbox

The level set toolbox (or toolboxLS) is the default toolbox implemented in MATLAB to solve any final-value HJ PDE. Since different reachable set computations can be ultimately posed as solving a final-value HJ-PDE, the level set toolbox is equipped to compute various types of reachable sets [1]. Information on how to install and use toolboxLS can be found at https://www.cs.ubc.ca/~mitchell/ToolboxLS/. This toolbox can be incorporated with the Hamilton Jacobi optimal control toolbox helperOC [17] for a subset of dynamical systems such as the Dubins Car model. An introduction to the helperOC can be found at: https:

//github.com/HJReachability/helperOC/blob/master/tutorial.m for quick start guide. The safe navigation framework in [3] uses the helperOC and level set toolbox frameworks for HJI computations and can be found at https://github.com/abajcsy/safe_navigation.

BEACLS

The Berkeley Efficient API in C++ for Level Set methods (BEACLS) Toolbox implements the functions from helperOC and toolboxLS for fast computation of reachability analyses. It works by using GPUs to parallelize different computations in the level set toolbox. Specifically, BEACLS divides the value function computation into separate regions which each have individual threads, much similar to how matrix multiplication is sped up in GPU's. This GPU library has been used for large-scale multi-vehicle reachability problems, such as safe path planning [11]. The installation instructions and user guide can be found at http://www.github.com/HJReachability/beacls.

BEACLS_ROS

The BEACLS_ROS Toolbox is one of the main contributions of this thesis. It was developed to provide a bridge between the Robot Operating System (ROS) and BEACLS so that modern robotics applications could leverage efficient BRS computations for safety planning. The library can be found at https://github.com/HJReachability/beacls_ros. Within this thesis, we demonstrate real-world examples of robots running through ROS and leveraging the BEACLS library to plan safe trajectories around obstacles.

Optimized DP

The optimized dp toolbox [29] was built by Prof. Mo Chen at Simon Fraser University, with the goal of having an easy python-based user interface to set up a reachability problem. The python interface wraps the fast C++ BEACLS code via HeteroCL in order to provide an intuitive scripting language entry point to the verbose C++ code. All software for this codebase can be found at: https://github.com/SFU-MARS/optimized_dp.

FlowStar

The flowstar toolbox [12] was built by Prof. Sriram Sankaranarayanan at CU Boulder with the goal of computing safe sets for large numbers of states that do not have control and disturbance inputs. It uses scalable methods that lead to an over approximation of the safety set. All software for this codebase can be found at: https://flowstar.org/

2.6 Arbitration Policies

The safety framework [3] that this thesis builds upon combines the BRS $\mathcal{V}$ described in previous sections with the default planner Π in a switch control manner. Concretely,

$$u(t) = \begin{cases} \Pi(x(t), x^*, O_t), & \text{if } x(t) \in \mathcal{W}_t \\ u^*(t, x(t)), & \text{otherwise} \end{cases} \tag{2.9}$$

If $V(x) < 0$, then the robot chooses to take the optimal safety control u^* described in equation (2.8) until the robot decides to replan for a new trajectory using the goal reaching planner Π. This bang bang control leads to trajectories that have high jerk and are uncomfortable for users. In this section, we explore a series of alternative ways that blend the BRS $\mathcal{V}$, safety control u^*, and $\xi^{planner}$.

Wang [33] proposes to *add* the value function of the BRS to the original cost function J for the planner as seen in Equation (2.10).

$$J(x) = (1 - \lambda) J_{planner}(x) + \lambda V(x) \tag{2.10}$$

However, finding the correct λ between the value function and original objective is difficult in practice and does not generalize well across novel environments. Alternatively, Leung [21] suggests to use the value function as a *constraint* for the planner's original optimization algorithm as described in Equation (2.11).

$$\max_{x_0 \dots x_T} \sum_{t=0}^{T} J(x_t) \quad \text{s.t.} \quad V(x_t) > \epsilon \ \ \forall t \tag{2.11}$$

A cone of controls are taken such that the vehicle safety score only increases once the robot enters an unsafe state where $V(x) \leq \epsilon$. Although optimal if HJI correctly models the full state and dynamics of all unknown obstacles, such a constrained optimization approach may suffer from being too closely constrained and thereby infeasible in practice. Volpe [30] suggests a blending mechanism between *velocity profiles* for a similar task of robotic manipulation where:

$$u_{blend} = \alpha * u_a + (1 - \alpha) * u_b \tag{2.12}$$

We explored such a control space blending mechanism in our initial experiments and found that blending velocities at earlier time steps would adversely affect the controls at later time steps thereby rendering the trajectory meaningless. In a similar notion, Dragan [13] suggests to arbitrate between *trajectories* of different agents for the shared autonomy task of human and robot teleoperation as described in Equation (2.13).

$$x_{blend} = \alpha * x_a + (1 - \alpha) * x_b \tag{2.13}$$

In our work, we extensively explore a series of arbitration schemes using this idea and analyze the properties of blending in trajectory space.

Chapter 3

Real-Time BRS Calculations

In this chapter we introduce the BEACLS_ROS repository, a real-time C++ ROS library that integrates the BEACLS toolbox with the Robot Operating System (ROS). BEACLS_ROS allows for efficient computations of backwards reachable sets and reusable code modules for robot navigation tasks.

3.1 Running Example

To illustrate our toolkit, we introduce a simple running example: a 3-dimensional Dubins car system with disturbances added to the velocity. The dynamics of the system are given by:

$$\begin{aligned} \dot{p}_x = v\cos\phi + d_x, \quad \dot{p}_y = v\sin\phi + d_y, \quad \dot{\phi} = \omega\,, \\ \underline{v} \leq v \leq \bar{v}, \quad |\omega| \leq \bar{\omega}, \quad |d_x|, |d_y| \leq d_r \end{aligned} \tag{3.1}$$

where $x := (p_x, p_y, \phi)$ is the state, $p = (p_x, p_y)$ is the position, ϕ is the heading, and $d = (d_x, d_y)$ is the disturbance experienced by the vehicle. The control of the vehicle is $u := (v, \omega)$, where v is the speed and ω is the turn rate. Both controls have a lower and upper bound, which for this example are chosen to be $\underline{v} = 0.1m/s$, $\bar{v} = 1m/s$, $\bar{\omega} = 1rad/s$. The disturbance bound is chosen as $d_r = 0.1m/s$.

The vehicle start and goal state are given by $x_0 = [-2, -2, \pi/2]$ and $x^* = [2, 2, -\pi/2]$. The square grid environment that the robot lives in ranges from $[-5, -5, -\pi]$ to $[5, 5, \pi]$. There is an rectangular obstacle in the environment which is not known to the vehicle beforehand. The rectangular obstacle's lower left corner lies at $[-1.5, 0, -\pi]$ and upper right corner at $[3.5, 2, \pi]$. The robot has a circular sensing region of radius 1.5 which allows it to detect free and occupied regions. For this simple benchmark, we use a hand-coded trajectory ξ^{hand} and planner Π^{hand} that navigates the robot to the next individual waypoint. The goal of the robot is to follow the green waypoints until it arrives at the final one.

For safety calculation demonstrations, we replan for the safety set using the algorithms described in [3] as soon as we receive new sensor updates from the environment. A snapshot of the robot navigation task is shown in Figure 3.1.

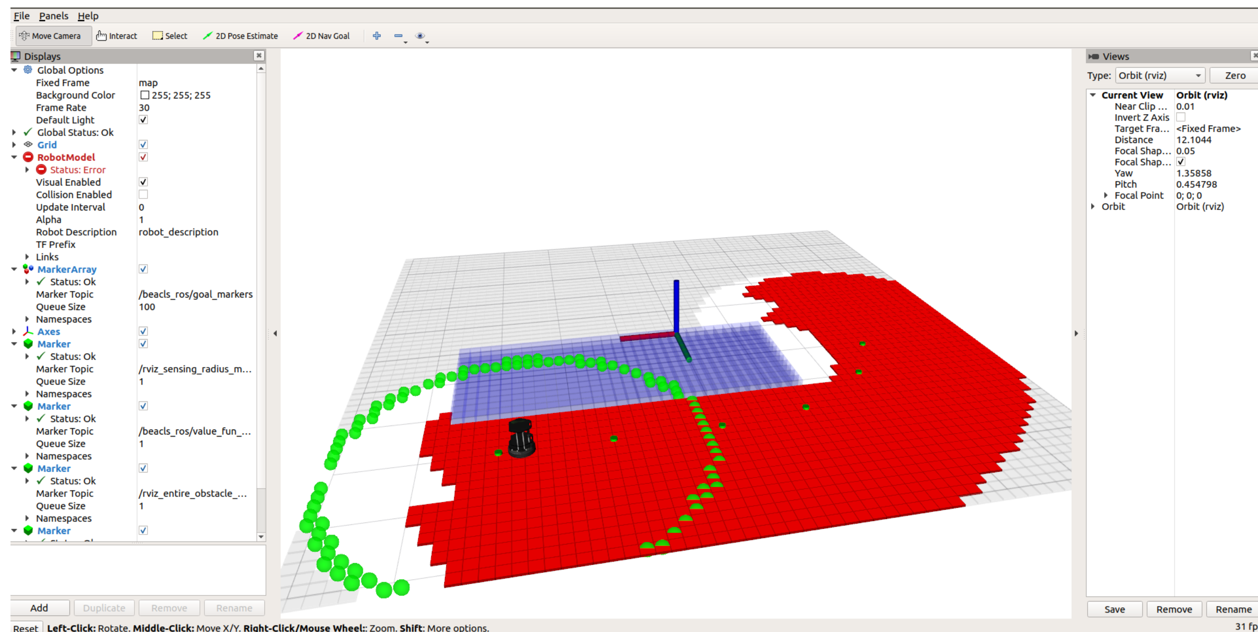


Figure 3.1. A running example of BEACLS_ROS in action. The goal is for the turtlebot to reach the last green way point from the initial position. Big green circles denote the circular sensing region. Blue denotes the robot obstacle. Black is the turtlebot. Red is the safety set calculated by HJIPDE in real time. Note that for this example, we use a 61x61x31 grid and can still evaluate the safety set in real time.

3.2 BEACLS_ROS

To calculate safety sets efficiently, we use BEACLS a powerful library written in C++. In a CPU setting, BEACLS provides a 6x speed up [11] in comparison to the native MATLAB helperOC_dev computations. This 6x speed up is crucial for using safety sets in real-world robotic applications. Unfortunately, BEACLS suffers from two issues: difficulty of usage and lack of tooling for ROS modules. These two issues makes it hard for robotics users to interface with this powerful toolbox.

BEACLS_ROS aims to address these shortcomings by using a single roslaunch file to seamlessly integrate different robotic modules such as the safety set calculation. All parts of the robotic stack such as the simulation odometry state, sensed environment occupancy grid, planning algorithm, safety set calculation, and visualization node can be launched with one roslaunch script. Communication between different ROS nodes is achieved through different ROS topics and messages as can be seen in Figure 3.2. The BEACLS_ROS toolbox is designed for usability so that users only need to edit one line in the yaml config files to change

experiment hyperparameters (e.g. grid size, disturbance, vehicle dynamics, etc.). Another design principle of BEACLS_ROS is modularity. By employing roslaunch scripts, the user can easily swap out different modules such as an RRT planner vs a hand coded trajectory with a single line change. This improved modularity and usability of BEACLS_ROS prevents the need for the user to modify source code and rebuild the workspace every time they make an experiment change.

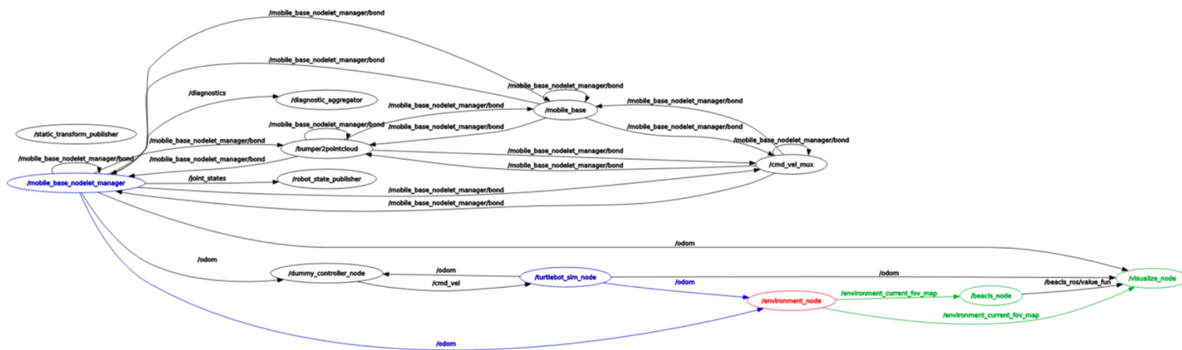


Figure 3.2. The ROS C++ Data pipeline for BEACLS_ROS. Note that all modules in the BEACLS_ROS repo are colored while all black nodes represent the turtlebot nodes. The robotic navigation stack goes from turtlebot odometry state ⇒ turtlebot controls ⇒ environment occupancy grid update ⇒ beacls safety set ⇒ ROS visualization

The current BEACLS codebase reimplements the three safety set algorithms presented in the safe navigation framework [3]: HJI-VI, warm start, and local Q. For these three algorithms, we expect to see a 6x speed up in computation time when baselined against the safe navigation paper's MATLAB implementation.

To implement the algorithms correctly in software: legacy code modifications to BEACLS, improved infrastructure tooling, and rigorous integration tests were key. First, additional features in BEACLS such as early stopping and minVWithL were added to the BEACLS toolkit. Next, getting local Q to work in C++ required modifying several of the original BEACLS toolkit functions which took a bulk of the implementation time. Another crucial step towards getting the algorithms running was porting the BEACLS codebase to run on docker, and therefore any machine. Surprisingly, the biggest speed up in development velocity was switching to VSCode which allowed for IDE based debugging of ROS tests. To validate correctness, all C++ algorithm value functions were sanity checked against their corresponding matlab implementation for a deterministic example.

3.3 Results and Discussion

As expected, we see the predicted 6x speed up in warm start and HJI for the running BEACLS_ROS example as seen in table 3.1 below. We only saw a 1.4x speed up for the Local Q update algorithm which we infer arises from the overhead when BEACLS copies around a queue to parallelize the value function iteration.

Method	Average Run Time (seconds)	Matlab Speed Up
C++ HJI	2.43	6.64x
C++ Warm	0.72	5.67x
C++ Local Q	0.55	1.40x
Matlab HJI	16.17	1.0x
Matlab Warm	4.13	1.0x
Matlab Local Q	0.77	1.0x

Table 3.1. HJI Speed up in C++

Here are three demo videos of the HJI, warm_start, and local_q update algorithms for computing safety sets in real time robotic navigation tasks. Some areas of future work for the BEACLS_ROS toolbox is to add in GPU benchmarks, NVIDIA GPU docker support, and more high dimensional dynamic models for safety set calculations.

Chapter 4

Blending Schemes for Safety Sets

4.1 Problem Set Up

Overview

In this chapter, we explore a series of model predictive control (MPC) based algorithms that tackle the point navigation task. At each MPC planning iteration, the robot is given a goal reaching planner and a safety aware BRS. We aim to explore several ways the robot can plan with these two pieces of information such that the blended trajectory is safe, goal reaching, feasible, and smooth.

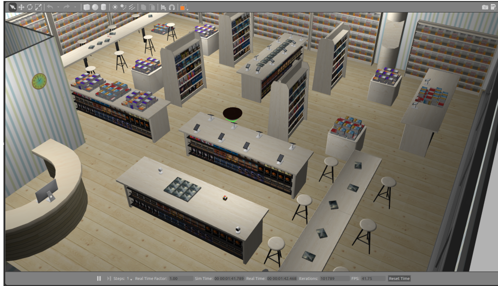
Task Set Up

The robot begins at start position x_0 and wants to navigate to goal position x^*. To accomplish this, we employ the MPC framework where we replan towards the goal every $T = 1.5$ seconds. The time discretization $dt = 0.1$ seconds and therefore the robot takes 15 time steps every MPC iteration. We terminate with success once the robot reaches within 0.5 meters of the goal x^*. We terminate with failure once the robot takes over 1000 time steps or crashes into an obstacle.

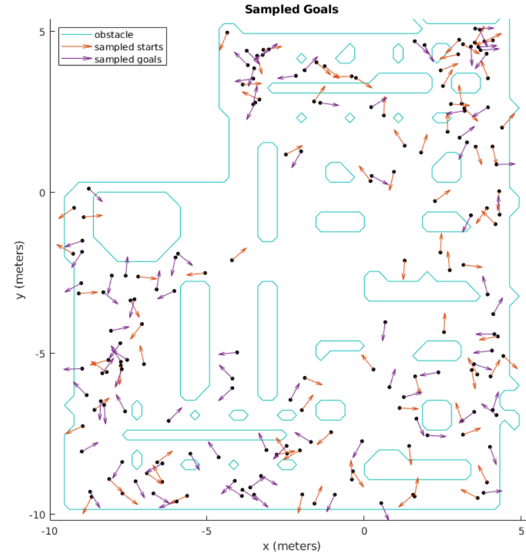
The robot lives in a realistic Amazon robomaker bookstore world shown in Figure 4.1a. The bookstore world lies between the range of $[-10, -5.4, -\pi]$ to $[10, 5.4, \pi]$ measured in meters and is discretized by $51 \times 51 \times 16$ grid cells. In the unknown environment, we use a lidar sensing region of radius 2. Thus, at every MPC planning iteration, the robot attains a new binary occupancy grid O_t with 15 new lidar hits. Using the binary occupancy grid O_t, the robot can recalculate the BRS $\mathcal{V}$ using the efficient HJI algorithms described in Section 2.5. To generate 100 test bed point-goal navigation scenarios, we randomly sampled 100 random start and goal locations that lie at least 10 meters away from one another as seen in Figure 4.1b.

For the robots dynamics model, we use the 3D Dubins Car model described in Equa-

tion (3.1) with the exact same hyper-parameters. To calculate the BRS, we use the helperOC_dev toolbox referenced in Section 2.5 with a disturbance of 0.1.



(a) AWS Robomaker Bookstore environment



(b) Sampled Goals for Bookstore Map

Metrics

We define a series of metrics in this section to quantitatively measure the smoothness, efficiency, and safeness of our robot's trajectories. **Smoothness** is measured by the average jerk or $\frac{d^3x}{dt^3}$ of the robot's trajectories. **Efficiency** is measured by the total number of trajectories where the robot timed out and the average number of time steps the robot took to successfully reach the goal. **Safety** is measured by the total number of trajectories where the robot crashed and the average number of safety controls the robot took to successfully reach the goal. We aim to optimize for each of the five metrics listed above in order to maximize smoothness, efficiency, and safety.

Planner

For our experiments, we chose to use a spline planner [32] as it provides a one to one mapping between the end robot position x_T and candidate spline planner trajectory $\xi^{candidate}$. This planner choice provides us a simple interface to implement our optimization based blending schemes as we can iterate over the set of all dynamically feasible candidate splines $\mathcal{X}$ to find the optimal spline $\xi^{planner}$. Furthermore, the differential flatness of our dynamics model enables the spline planner to generate smooth and continuous trajectories.

On a high level, the spline planner works by starting at position x_0 with velocity v_0 and enumerates over the set of all dynamically feasible candidate splines $\mathcal{X}$ that the robot can reach. The spline planner selects the optimal spline trajectory $\xi^{planner}$ whose states accumulate the least signed distance cost J as described in Equation (4.1).

$$\begin{aligned} \min_{x_0 \dots x_T} \sum_{t=0}^{T} J(x_t) \\ \text{s.t.} \quad x_t = x_{t-1} + f(x_{t-1}, u_{t-1}) \\ \xi^t = (x_t, u_t) \\ \xi \in \mathcal{X} \end{aligned} \tag{4.1}$$

More concretely, $\mathcal{X}$ is the set of dynamically feasible splines that start from state x_0, velocity v_0, and time $t_0 = 0$ and end in a discrete environment grid state x_f, velocity v_f, after time $t_f = T$ seconds. To obtain a spline trajectory from the constraints mentioned above, we fit a third order spline to the evolution of $p_x(t)$ and $p_y(t)$ and detail the initial and terminal constraints as seen in Equation (4.2).

$$\begin{aligned} p_x(t) = a_1 t^3 + b_1 t^2 + c_1 t + d \\ p_y(t) = a_2 t^3 + b_2 t^2 + c_2 t + d \\ \text{s.t.} \quad p_x(0) = p_{x_0},\ p_x(t_f) = p_{x_f},\ \dot{p_x}(0) = v_0 \cos(\phi_0),\ \dot{p_x}(t_f) = v_f \cos(\phi_f) \\ p_y(0) = p_{y_0},\ p_y(t_f) = p_{y_f},\ \dot{p_y}(0) = v_0 \sin(\phi_0),\ \dot{p_y}(t_f) = v_f \sin(\phi_f) \end{aligned} \tag{4.2}$$

We can solve these set of equations and find the values of a_1, b_1, c_1, d_1 and a_2, b_2, c_2, d_2.

$$\begin{bmatrix} a_1 \\ b_1 \\ c_1 \\ d_1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ t_f^3 & t_f^2 & t_f & 1 \\ 0 & 0 & 1 & 0 \\ 3t_f^2 & 2t_f & 1 & 0 \end{bmatrix}^{-1} \begin{bmatrix} p_{x_0} \\ p_{x_f} \\ v_0 \cos(\phi_0) \\ v_f \cos(\phi_f) \end{bmatrix} \tag{4.3}$$

$$\begin{bmatrix} a_2 \\ b_2 \\ c_2 \\ d_2 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ t_f^3 & t_f^2 & t_f & 1 \\ 0 & 0 & 1 & 0 \\ 3t_f^2 & 2t_f & 1 & 0 \end{bmatrix}^{-1} \begin{bmatrix} p_{y_0} \\ p_{y_f} \\ v_0 \sin(\phi_0) \\ v_f \sin(\phi_f) \end{bmatrix} \tag{4.4}$$

After finding the coefficients for $p_x(t)$ and $p_y(t)$, we can then leverage the differential flatness of the Dubins Car dynamics model to solve for $\phi(t)$, $v(t)$, and $\omega(t)$ as seen in Equation (4.5).

$$\begin{aligned} \phi(t) = \tan^{-1}\left(\frac{p_y(t)}{p_x(t)}\right) \\ v(t) = \sqrt{p_y(t)^2 + p_x(t)^2} \\ \omega(t) = \frac{\ddot{p_y}(t)\dot{p_x}(t) - \ddot{p_x}(t)\dot{p_y}(t)}{p_y(t)^2 + p_x(t)^2} \end{aligned} \tag{4.5}$$

Finally, we can create our candidate spline trajectory $\xi^{candidate}$ from $x(t), u(t)$ where $x(t) = (p_x(t), p_y(t), \phi(t))$ and $u(t) = (v(t), \omega(t))$. If $\xi^{candidate}$ satisfies all dynamic constraints where $x_t = x_{t-1} + f(x_{t-1}, u_{t-1})$ for regularly interleaved time intervals between t = 0 to T, we can then add $\xi^{candidate}$ to the candidate set of dynamically feasible splines $\mathcal{X}$.

To choose the optimal spline $\xi^{planner}$ from the set $\mathcal{X}$, one can solve an optimization problem that minimizes the cost of a spline trajectories' positions over a potential field J as described in Equation (4.1). In fact, J can be comprised of a sum of different potential fields. For our running example, J is equivalent to Equation (4.6) where $J_{obs}(x)$ penalizes states in obstacles only and $J_{goal}(x)$ gives higher cost to states that are further from the goal.

$$J(x) = J_{goal}(x) + \lambda J_{obs}(x) \tag{4.6}$$

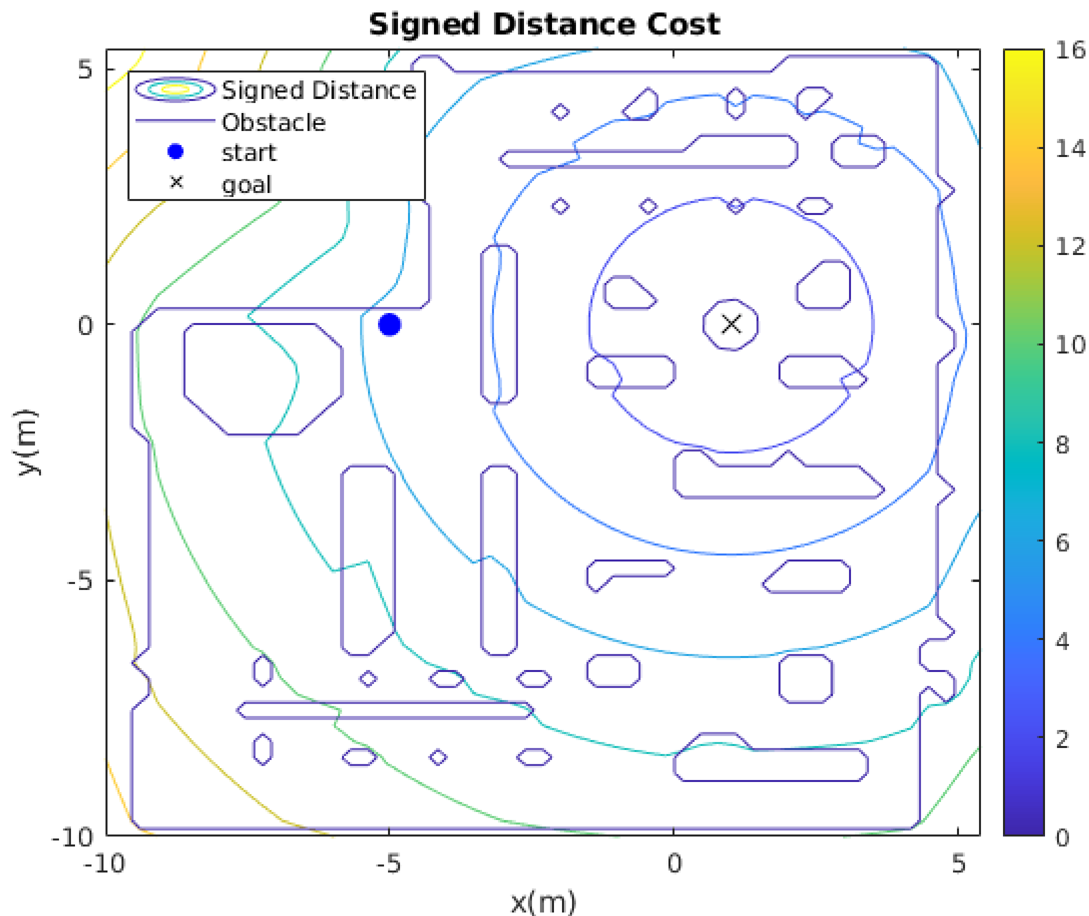


Figure 4.2. Signed Distance Potential Cost Function J

In practice, we note that this spline algorithm described above suffers from fatal singularity conditions. To address this, we add the 3rd order spline parameterizations for time

described in [32] to our planner. We left out such details for notational brevity and encourage interested readers to read the original reference for a more robust spline planner implementation. We also notice that this MPC based spline planner still crashes quite often when $\lambda = 1$ (i.e. J_{goal} and J_{obs} are weighted equally). Thus, this spline planner serves as an ideal test bed for our safety and blending framework. Finally, we reiterate that our blending framework is planner agnostic and any planner would have been viable.

Blending MPC Pipeline

In the model predictive control replan loop, we first query the spline planner derived above to get the trajectory $\xi^{planner}$. Next, if we are in an unknown environment, we resolve for the BRS as described in (2.4) with the new occupancy grid O_t. In the known environment, the BRS is already solved for in the first MPC iteration. With the BRS $\mathcal{V}$, value function V, and planner trajectory $\xi^{planner}$, we can now blend these three pieces of information into a new trajectory ξ^{blend}. The robot can then execute this trajectory ξ^{blend} until T timesteps are up or the robot enters an unsafe state where $V(x) < 0$. In this unsafe state, the robot switches to only taking safety controls for the remainder of the MPC horizon as described in Equation (2.9). To summarize, Algorithm 1 captures the main logic of our robot navigation pipeline.

Algorithm 1: Blending MPC Pipeline

1. x_0, x^*: The start and goal states
2. O_0: The binary occupancy grid at x_0
3. $\Pi(\cdot, x^*, O_0)$: The planner for the vehicle
4. $\mathcal{V}$: Calculate the initial backwards reachable set using O_0 and (2.4)
5. T: The control horizon
6. **while** *the vehicle is not at the goal* **do**
7. Obtain the current sensor observation $\mathcal{S}_t$
8. Update the binary occupancy grid O_t
9. Apply the least restrictive control $u(t)$ given by (2.9)
10. **for** ***every*** T *seconds* **do**
11. Replan the trajectory $\xi^{planner} \longleftarrow \Pi(\cdot, x^*, O_t)$
12. Recalculate the BRS $\mathcal{V}$ using O_t and (2.4)
13. Blend the trajectory ξ^{blend} using $\xi^{planner}$, Π, $\mathcal{V}$, V and blending schemes in 4.2

4.2 Blending Schemes

Overview

We want to blend the information between the safety-preserving optimal controls u^* encoded in the value function V, the BRS safety set $\mathcal{V}$, and planner trajectory $\xi^{planner}$ while also ensuring efficient, smooth, and minimum-cost trajectories ξ^{blend}. This question is very much an open problem and below we outline five candidate approaches for how to leverage the value function information at planning time.

Approach 1: Bang Bang Control

Prior work [3] has tried using a switch-based control scheme to incorporate the safety set with the default planner. The blending scheme first employs the default controls of the original plan until the robot reaches an unsafe state. Then, the robot *switches* to only taking the optimal safety controls of the BRS for the remainder of the MPC loop. This switching logic can be succintly explained by Equation (2.9).

This bang bang control approach achieves safety by design and is minimally invasive to the goal-reaching planner. The drawbacks of this approach is that there is high jerk when one switches from the spline planner to the safety trajectory which can ultimately have negative effects on the robot's sensing ability, hardware, and any humans that may be in the loop.

Approach 2: Value Function Constrained Optimization

Other approaches have tried using the value function as a constraint inside the original planner's optimization procedure [21]. In this case, the motion planner is imbued with the reachability value function. When determining a cost-minimizing plan, the robot must choose states which ensure that the robot never enters the unsafe, sub-zero level set of the value function. Note that in practice, it's often useful to allow for some small tolerance ϵ near the zero-level set due to model mismatch.

$$\begin{aligned} \min_{x_0 \dots x_T} \sum_{t=0}^{T} J(x_t) \\ \text{s.t.} \quad V(x_t) > \epsilon \\ x_t = x_{t-1} + f(x_{t-1}, u_{t-1}) \\ \xi^t = (x_t, u_t) \\ \xi \in \mathcal{X} \end{aligned} \tag{4.7}$$

This approach ensures safety through its value function constraint and finds the minimally invasive yet goal reaching trajectory. It has low jerk because all blended trajectories are splines. For known environments, we hypothesize that this blending scheme will perform close to optimal as it solves for a short horizon approximation of the reach avoid

HJI problem which has optimal guarantees [24]. In unknown environments, we expect this blending scheme to produce infeasible trajectories for some scenarios as the robot may be over-constrained by the value function requirements.

Approach 3: Unconstrained Optimization

An unconstrained blending formulation would alleviate the issues of feasibility that constrained optimization suffers from with drawback being that any safety guarantees would no longer be valid. To address this, we can do a post-processing of the unconstrained optimization solution to "correct" any of the unsafe parts of the trajectory. Below we detail three different ways we can formulate an unconstrained optimization problem to arbitrate between a safety trajectory ξ^{safety} and goal reaching trajectory $\xi^{planner}$.

Constant Alpha Blending

The value function V and planner trajectory $\xi^{planner}$ live in different spaces. Thus, to effectively blend the two pieces of information, a safety trajectory ξ^{safety} can be created which lies in the trajectory space of $\xi^{planner}$. To generate ξ^{safety}, one can unroll the optimal control u_t^* starting at the robot's position x_0 for T time steps as seen in Algorithm 2.

Algorithm 2: Unroll Safety Trajectory

1. x_0: The current position of the robot
2. V: The most recent value function
3. T: The control horizon
4. $t \longleftarrow 0$: The current time stamp
5. $\xi^{safety} \longleftarrow \{\}$: Initialize empty safety trajectory
6. **while** $t < T$ **do**
7. $u_t^* = \text{getOptCtrl}(V, x_t)$: Get optimal safety control at current state
8. $\xi^{safety} = \{(x_t, u_t^*)\} + \xi^{safety}$: Add state and control to safety trajectory
9. $x_{t+1} = x_t + f(x_t, u_t^*)$: Update state with optimal safe control
10. $t \longleftarrow t + 1$: Update time stamp

Next, one can blend the safety trajectory ξ^{safety} and goal reaching trajectory $\xi^{planner}$ in

an unconstrained optimization problem as seen in Equation (4.8).

$$
\begin{aligned}
\min_{x_0^{blend} \dots x_T^{blend}} \sum_{t=0}^{T} \alpha \| x_t^{blend} - x_t^{planner} \|_2^2 + (1-\alpha) \| x_t^{blend} - x_t^{safety} \|_2^2 \\
\text{s.t.} \quad x_t^{blend} = x_{t-1}^{blend} + f(x_{t-1}^{blend}, u_{t-1}^{blend}) \\
\xi_t^{blend} = (x_t^{blend}, u_t^{blend}) \\
\xi_t^{planner} = (x_t^{planner}, u_t^{planner}) \\
\xi_t^{safety} = (x_t^{safety}, u_t^{safety}) \\
\xi^{blend} \in \mathcal{X}
\end{aligned}
\tag{4.8}
$$

If we linearize the dynamics and exclude the spline constraints, this unconstrained optimization problem can be recast as a canonical LQR problem [26]. The α blending parameter controls how much we want to stay near the original planned trajectory $\xi_{planner}$ versus the safe trajectory ξ_{safe} and lies between $[0, 1]$. If $\alpha = 0$, the robot will always choose the safety trajectory and remain stuck in a loop. If $\alpha = 1$, we return to the bang bang control scheme described in Section 4.2. In practice, we found that no constant alpha was robust enough to account for all real world point navigation scenarios. Thus, we seek to improve upon this unconstrained optimization formulation with the following two approaches.

Sampled Alpha Blending

In order to guarantee the safety of the trajectories produced by alpha blending, one can sample α from $[0 - 1]$ in a decreasing order and re-solve Equation (4.8) until the blended trajectory ξ^{blend} is safe. This is similar to using the value function as a constraint as we saw in Section 4.7, yet different due to the fact that we are resolving an unconstrained optimization problem with each sampled α. This blending scheme searches through the space of optimization loss functions by decreasing α until we arrive at a loss function whose solution ξ_{blend} satisfies the safety constraint. Specifically, the optimization problem we solve

for is shown in Equation (4.9).

$$\begin{aligned}\max_{\alpha} \min_{x_0^{blend} \dots x_T^{blend}} \sum_{t=0}^{T} \alpha \| x_t^{blend} - x_t^{planner} \|_2^2 + (1-\alpha) \| x_t^{blend} - x_t^{safety} \|_2^2 \\ \text{s.t.} \quad V(x_t^{blend}) \geq 0 \\ 0 \leq \alpha \leq 1 \\ x_t^{blend} = x_{t-1}^{blend} + f(x_{t-1}^{blend}, u_{t-1}^{blend}) \\ \xi_t^{blend} = (x_t^{blend}, u_t^{blend}) \\ \xi_t^{planner} = (x_t^{planner}, u_t^{planner}) \\ \xi_t^{safety} = (x_t^{safety}, u_t^{safety}) \\ \xi^{blend} \in \mathcal{X}\end{aligned} \tag{4.9}$$

This approach achieves safety by constraint and finds the minimally invasive alpha to blend the BRS $\mathcal{V}$, value function V, and goal reaching trajectory $\xi^{planner}$. For known and unknown environments, we expect this blending scheme to perform close to optimal due to its sampling nature of alpha that finds the minimally invasive optimization problem whose solution is a safe trajectory.

Value Function Alpha Blending

Another way one might blend the safety trajectory ξ^{safety} and goal reaching trajectory $\xi^{planner}$ is to solve the unconstrained optimization problem described in Equation (4.8) with alpha derived from the value function $V(x)$. Intuitively, the value function is a good proxy of when we should compare against the safety trajectory ξ^{safety} or planned trajectory $\xi^{planner}$. Specifically, if we are in unsafe state where $V(x)$ *is low*, we want to penalize staying away from the original safe trajectory ξ^{safety} so α *should be low*. Conversely, if we are in very safe states where $V(x)$ *is high*, we don't need to stay close to the safety trajectory so α *should be high*. Thus, the value function based optimization problem can be formulated in Equation (4.10) as:

$$\begin{aligned}\min_{x_0^{blend} \dots x_T^{blend}} \sum_{t=0}^{T} \alpha_t \| x_t^{blend} - x_t^{planner} \|_2^2 + (1-\alpha_t) \| x_t^{blend} - x_t^{safety} \|_2^2 \\ \text{s.t.} \quad \alpha_t = g(V(x_t)) \\ x_t^{blend} = x_{t-1}^{blend} + f(x_{t-1}^{blend}, u_{t-1}^{blend}) \\ \xi_t^{blend} = (x_t^{blend}, u_t^{blend}) \\ \xi_t^{planner} = (x_t^{planner}, u_t^{planner}) \\ \xi_t^{safety} = (x_t^{safety}, u_t^{safety}) \\ \xi^{blend} \in \mathcal{X}\end{aligned} \tag{4.10}$$

The value function output $V(x)$ lies in a continuous space R of all real numbers while α lies between $[0, 1]$. To account for this, we investigated a series of alpha functions g that map a value function output to the space $[0, 1]$ as seen in Figure 4.3

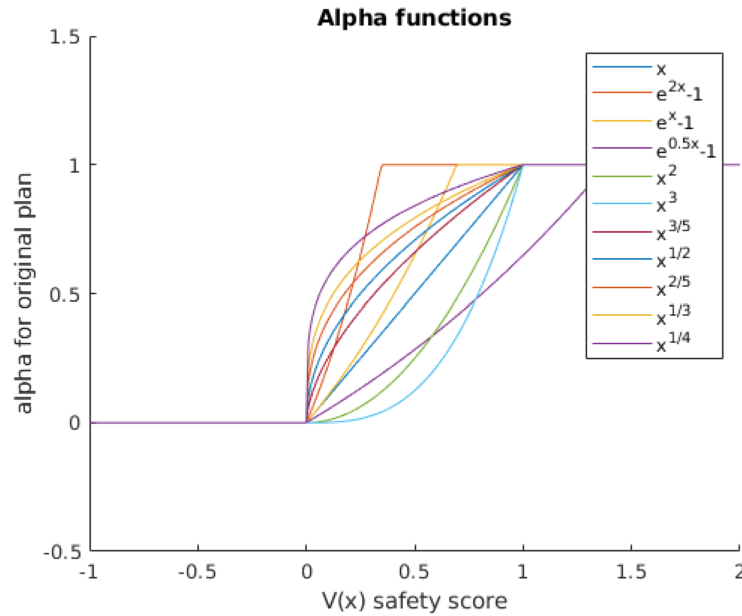


Figure 4.3. Alpha Functions

Experimentally, we found the optimal alpha function in known environments to be $g(x) = \max(\min(x^{\frac{2}{5}}, 1), 0)$ and the optimal alpha function in unknown environments to be $g(x) = \max(\min(x^{\frac{1}{4}}, 1), 0)$. Thus, for all future results, we use the aforementioned g functions to map value function outputs to alphas α.

4.3 Results

Table 4.1 shows the results of the algorithms mentioned above tested with 100 sampled scenarios in a known AWS bookstore environment. Table 4.2 shows the results of the algorithms tested with 100 sampled scenarios in an unknown AWS bookstore environment. Figure 4.4 shows an example of a robot navigating in this simulated bookstore world with different blending schemes. Figure 4.5 shows the key metrics we profile for a robot trajectory.

Table 4.1. **Known Environment Results**: The success metric counts the number of point navigation scenarios that reached the goal location x^* within 0.5 meters. The crash metric counts the number of scenarios where the robot crashed into an obstacle. The timeout metric counts the number of scenarios where the robot ran for more than 1000 timesteps. The total number of test scenarios is 100. For the **67** scenarios in which all 4 algorithms succeeded, we measured the jerk, time taken, and number of safety controls each robot took. Jerk is measured in m/s^3. Time Taken is measured in seconds. Num Safety counts the total number of safety controls the robot took to navigate through the environment to the goal.

Method	**Success**	**Crash**	**Timeout**	**Jerk**	**Time Taken**	**Num Safety**
Bang Bang (2.9)	82	14	4	0.33	22.75	41.5
Constrained Optimization (4.7)	**87**	**1**	**12**	**0.15**	**20.49**	**5.55**
Sampled Alpha (4.9)	81	12	7	0.27	22.14	31.15
Value Function Alpha (4.10)	85	7	8	0.28	23.30	32.75

Table 4.2. **Unknown Environment Results**: The success metric counts the number of point navigation scenarios that reached the goal location x^* within 0.5 meters. The crash metric counts the number of scenarios where the robot crashed into an obstacle. The timeout metric counts the number of scenarios where the robot ran for more than 1000 timesteps. The total number of test scenarios is 100. For the **22** scenarios in which all 4 algorithms succeeded, we measured the jerk, time taken, and number of safety controls each robot took. Jerk is measured in m/s^3. Time Taken is measured in seconds. Num Safety counts the total number of safety controls the robot took to navigate through the environment to the goal.

Method	**Success**	**Crash**	**Timeout**	**Jerk**	**Time Taken**	**Num Safety**
Bang Bang (2.9)	74	12	14	0.133	8.04	40.75
Constrained Optimization (4.7)	67	**2**	32	0.085	**7.735**	8.21
Sampled Alpha (4.9)	**82**	4	**14**	0.075	11.56	**7.3**
Value Function Alpha (4.10)	34	5	61	**0.073**	17.91	21.25

For the curious reader, one may look at the unknown environment results, known environment results, unknown environment plots, and known environment plots for a more

concrete understanding of each blending scheme's performance. All blending scheme plots have been grouped by success, crash, and time limit exceeded (TLE) outcomes so that one can get a general sense of an blending scheme's failure and success modes.

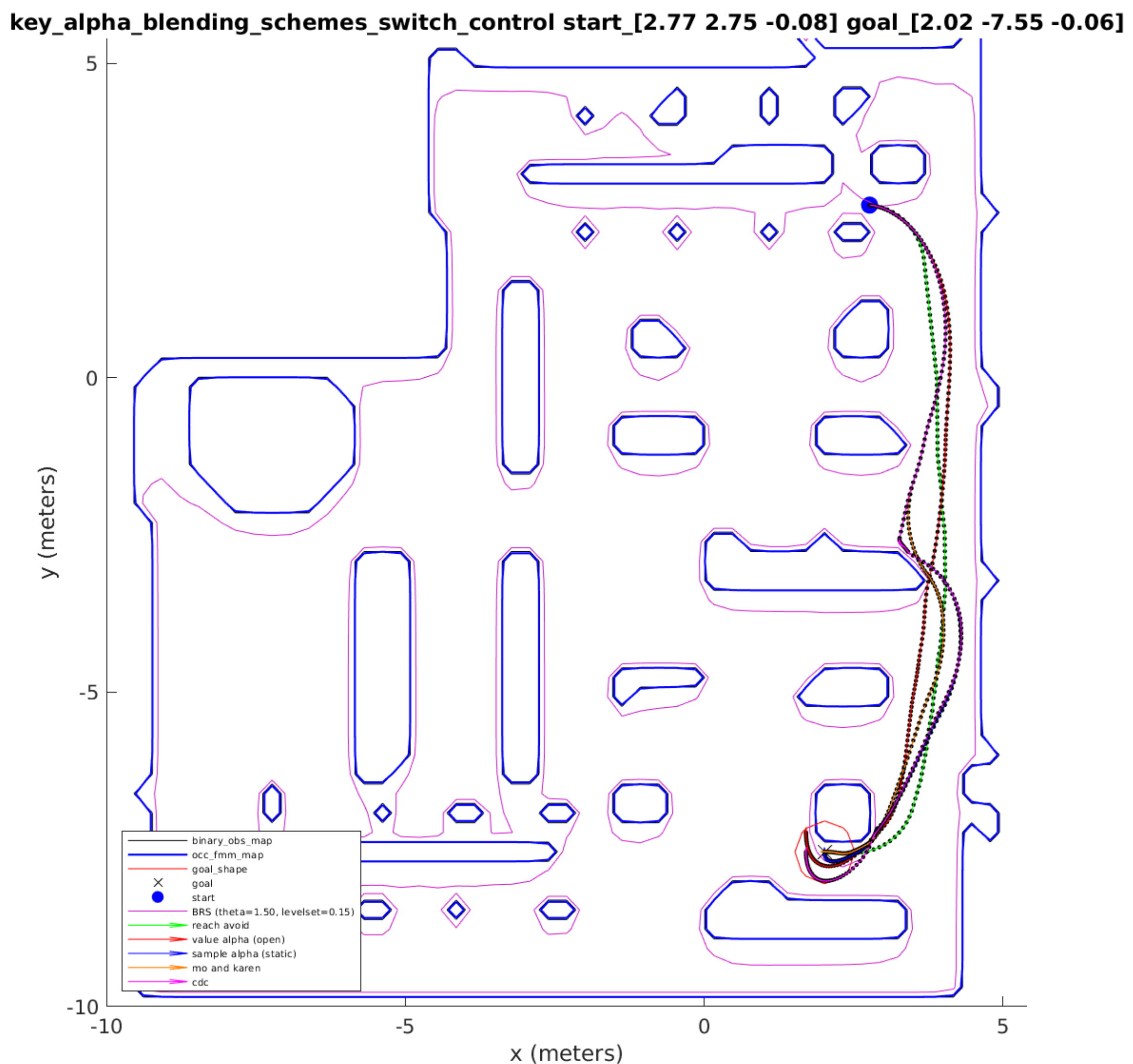


Figure 4.4. **Robot trajectories with different blending schemes:** The robot is moving from the start position in *blue* to the goal position (*black X*). Several different blending schemes are plotted in different colors. Value Alpha (Red): Value Function Alpha (4.10), Sample Alpha (Blue): Sampled Alpha (4.9), Mo and Karen (Orange): Constrained Optimization (4.7), CDC (Pink): Bang Bang Control (2.9). The reach avoid trajectory in green is the HJI solution that we use as an optimal baseline. The pink outline is the BRS V.

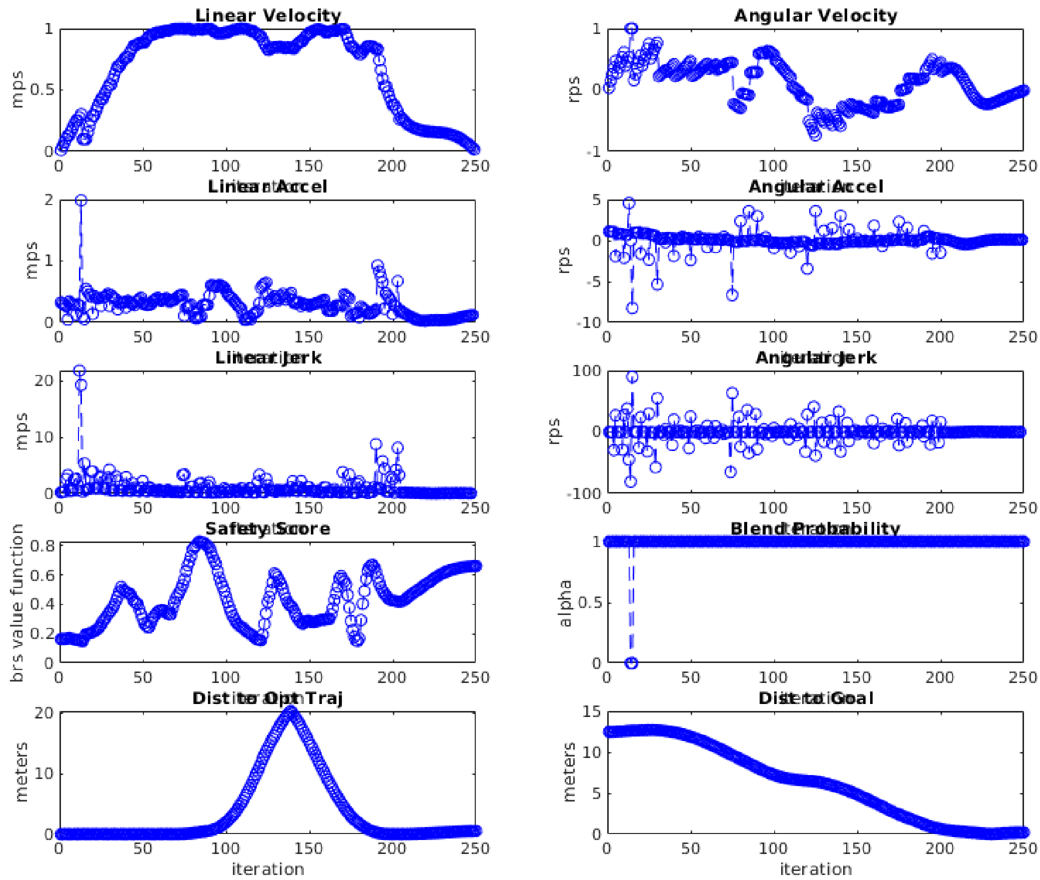


Figure 4.5. **Trajectory Metrics** We logged several metrics of the robot's trajectory including the linear jerk, angular jerk, alpha blend probability, average safety score $V(x)$, distance to goal, and distance to optimal trajectory (reach avoid trajectory)

4.4 Discussion

Known Environments

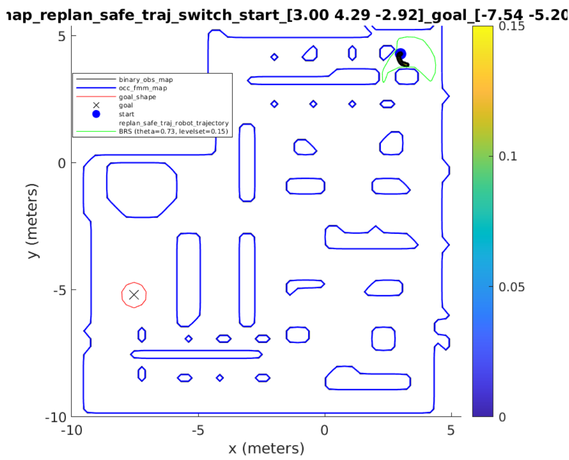
The constrained optimization blending scheme achieved the highest success rate of 87% in the known environment. Since constrained optimization plans around the obstacles, it took significantly less safety controls than unconstrained optimization methods. However, all algorithms reached greater than 80% success rate in the known environment indicating that this environment may be too simple for conclusive results. As expected, the bang bang control blending scheme had the highest jerk among all algorithms. Table 4.3 summarizes the key properties of this section's algorithms for a known environment.

Algorithm	Low Jerk	Safe Trajectory	Goal Reaching
Bang Bang (2.9)	✗	✓	✓
Constrained Optimization (4.7)	✓	✓	✓
Sampled Alpha (4.9)	✓	✓	✓
Value Function Alpha (4.10)	✓	✗	✓

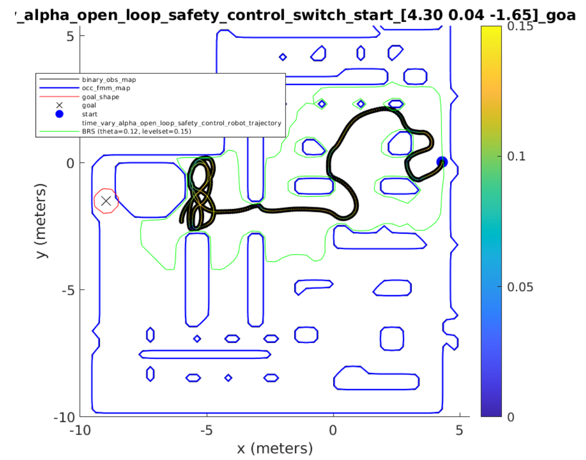
Table 4.3. Blending Scheme Properties for Known Environment

Unknown Environments

The sampled alpha blending scheme achieved the highest success rate of 82% in the unknown environment. We expect this is cause this blending scheme finds the minimally invasive unconstrained optimization problem that blends both the safety and goal reaching planner. The constrained optimization blending scheme timed out more frequently in the unknown environment due to the blending scheme being over constrained as seen in Figure 4.6a. We also witnessed the value function alpha blending scheme timing out significantly more frequently with a lot of scenarios getting stuck in local minimums as seen in Figure 4.6b.



(a) Robot being overly constrained due to the constrained optimization blending scheme



(b) Robot getting stuck in local minimum for the value function alpha blending scheme

Table 4.4 summarizes the key properties of this section's algorithms for an unknown environment.

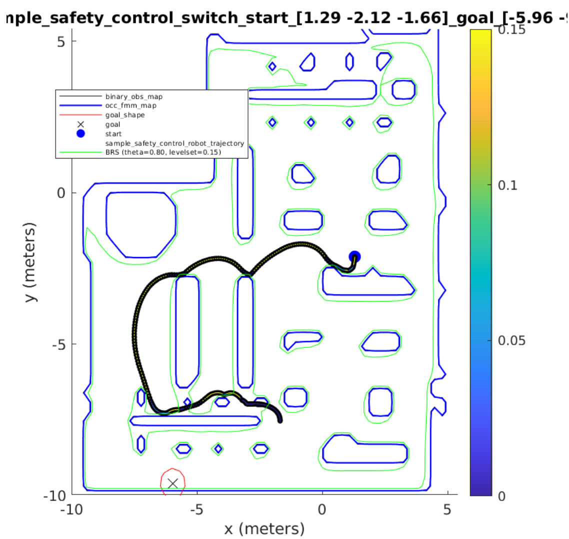
Algorithm	Low Jerk	Safe Trajectory	Goal Reaching	Feasible
Bang Bang (2.9)	✗	✓	✓	✓
Constrained Optimization (4.7)	✓	✓	✓	✗
Sampled Alpha (4.9)	✓	✓	✓	✓
Value Function Alpha (4.10)	✓	✗	✓	✗

Table 4.4. Blending Scheme Properties for Unknown Environment

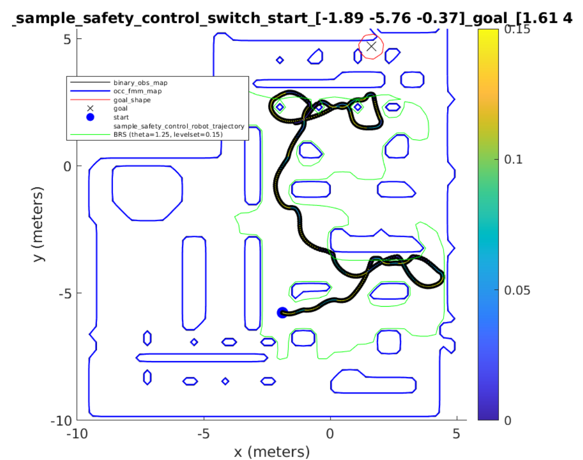
Failure Modes

Two failure modes that we saw in our point navigation experiments included robots crashing and robots exceeding the 1000 time step limit.

HJI-analysis based navigation has theoretical guarantees that the robot should never crash. However, in real world environments, we witnessed several collisions similar to the examples seen in Figure 4.7a and Figure 4.7b. We investigated all crash scenarios and found that the robot collisions only occur near obstacles of size 1, 2, or 3. We also ran these experiments on a simpler environment with a lack of small obstacles and report no crashes. This indicates that HJI safety guarantees display numerical instabilities near small obstacles for it's current helperOC_dev implementation described in Section 2.5. We suggest padding small obstacles to reduce numerical instabilities for future experiments.



(a) Robot Crash Example 1



(b) Robot Crash Example 2

Another issue lies in path planning algorithms getting stuck in infinite loops near tight corridors and long objects. This can be seen in examples such as Figure 4.6b. Fixing this issue would require some sort of memory for the robot's previous history and is an interesting direction for future work.

4.5 Summary

In this chapter, we explored a series of bang bang control, constrained optimization, and unconstrained optimization blending schemes that incorporated the safety BRS $\mathcal{V}$, value function V, and spline planners Π to produce a new trajectory ξ^{blend}. Interesting future work includes adding sensor noise to the dynamics models, incorporating uncertainty within the occupancy grid values, and testing these blending algorithms with different planners.

Chapter 5

Conclusion and Future Work

Providing safe, efficient, smooth, and goal-reaching planners for real-world autonomous systems operating in *a priori* unknown environments is a challenging but important problem. In this thesis, we first introduced the BEACLS_ROS toolkit that aims to speed up BRS calculations for safe robotic navigation. Next, we explored how the BRS can be blended with different planners to yield varying degrees of smooth, safe, feasible, and goal reaching policies. Finally, we demonstrated our approaches on an exhaustive test bed of 3D simulation scenarios.

Many promising future directions emerge from this thesis. For starts, one big assumption that we make in this thesis is that sensor measurements are accurate. Extending uncertainty into state estimation would be necessary for bringing this thesis' framework into real world navigation tasks. Next, extensions to bridge the gap between this thesis' static environment assumption with the dynamic, multi-agent real world is another promising direction to research. Finally, many failure cases in the point navigation tasks came from the difficulty of long term planning with HJI-VI. More investigation into successful long term planning algorithms will help drive the error rates of our algorithms down and will be vital to deploying robotic navigation planners in the real world.

Bibliography

[1] *A Toolbox of Level Set Methods*. URL: https://www.cs.ubc.ca/~mitchell/ToolboxLS/.

[2] Ayush Agrawal and Koushil Sreenath. "Discrete Control Barrier Functions for Safety-Critical Control of Discrete Systems with Application to Bipedal Robot Navigation". In: July 2017. DOI: 10.15607/RSS.2017.XIII.073.

[3] Andrea Bajcsy et al. "An Efficient Reachability-Based Framework for Provably Safe Autonomous Navigation in Unknown Environments". In: *2019 IEEE 58th Conference on Decision and Control (CDC)*. 2019, pp. 1758–1765. DOI: 10.1109/CDC40024.2019.9030133.

[4] S. Bansal et al. "Hamilton-Jacobi Reachability: A Brief Overview and Recent Advances". In: *CDC*. 2017.

[5] S. Bansal et al. "Safe sequential path planning under disturbances and imperfect information". In: *ACC*. 2017.

[6] Somil Bansal and Claire Tomlin. *DeepReach: A Deep Learning Approach to High-Dimensional Reachability*. 2020. arXiv: 2011.02082 [cs.RO].

[7] Somil Bansal et al. *Combining Optimal Control and Learning for Visual Navigation in Novel Environments*. 2019. arXiv: 1903.02531 [cs.RO].

[8] R. Bellman, R.E. Bellman, and Karreman Mathematics Research Collection. *Adaptive Control Processes: A Guided Tour*. Princeton Legacy Library. Princeton University Press, 1961. ISBN: 9780691079011. URL: https://books.google.com/books?id=POAmAAAAMAAJ.

[9] Mariusz Bojarski et al. *End to End Learning for Self-Driving Cars*. 2016. arXiv: 1604.07316 [cs.CV].

[10] Mo Chen et al. *Decomposition of Reachable Sets and Tubes for a Class of Nonlinear Systems*. 2017. arXiv: 1611.00122 [math.OC].

[11] Mo Chen et al. *Provably Safe and Robust Drone Routing via Sequential Path Planning: A Case Study in San Francisco and the Bay Area*. 2017. arXiv: 1705.04585 [cs.SY].

[12] Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. "Flow*: An Analyzer for Non-linear Hybrid Systems". In: *Computer Aided Verification*. Ed. by Natasha Sharygina and Helmut Veith. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 258–263.

[13] A. Dragan and S. Srinivasa. "A policy-blending formalism for shared control". In: *The International Journal of Robotics Research* 32 (2013), pp. 790–805.

[14] J. Fisac et al. "Reach-avoid problems with time-varying dynamics, targets and constraints". In: *HSCC*. 2015.

[15] David Fridovich-Keil et al. *Planning, Fast and Slow: A Framework for Adaptive Real-Time Safe Trajectory Planning*. 2018. arXiv: 1710.04731 [cs.SY].

[16] Sylvia L. Herbert et al. "Reachability-Based Safety Guarantees using Efficient Initializations". In: *CoRR* abs/1903.07715 (2019). arXiv: 1903.07715. URL: http://arxiv.org/abs/1903.07715.

[17] HJReachability. *HJReachability/helperOC*. URL: https://github.com/HJReachability/helperOC.

[18] Dong Ki Kim and Tsuhan Chen. *Deep Neural Network for Real-Time Autonomous Indoor Navigation*. 2015. arXiv: 1511.04668 [cs.CV].

[19] S. Kousik et al. "Bridging the Gap Between Safety and Real-Time Performance in Receding-Horizon Trajectory Design for Mobile Robots". In: *arXiv preprint* (2018).

[20] Karen Leung, Nikos Aréchiga, and Marco Pavone. *Back-propagation through Signal Temporal Logic Specifications: Infusing Logical Structure into Gradient-Based Methods*. 2021. arXiv: 2008.00097 [eess.SY].

[21] Karen Leung et al. "On infusing reachability-based safety assurance within planning frameworks for human–robot vehicle interactions". In: *The International Journal of Robotics Research* 39.10-11 (2020), pp. 1326–1345. DOI: 10.1177/0278364920950795. eprint: https://doi.org/10.1177/0278364920950795. URL: https://doi.org/10.1177/0278364920950795.

[22] Manolis Savva* et al. "Habitat: A Platform for Embodied AI Research". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019.

[23] K. Margellos and J. Lygeros. "Hamilton-Jacobi Formulation for Reach–Avoid Differential Games". In: *IEEE Trans. on Automatic Control* 56.8 (2011), pp. 1849–1861.

[24] Kostas Margellos and John Lygeros. *Hamilton-Jacobi formulation for reach-avoid differential games*. 2009. arXiv: 0911.4625 [math.OC].

[25] I. Mitchell, A. Bayen, and C. J. Tomlin. "A time-dependent Hamilton-Jacobi formulation of reachable sets for continuous dynamic games". In: *IEEE Trans. on automatic control* 50.7 (2005), pp. 947–957.

[26] Anderson Brian D O. and John B. Moore. *Optimal control: linear quadratic methods*. Dover Publications, 2007.

[27] Charles Richter, William Vega-Brown, and Nicholas Roy. "Bayesian Learning for Safe High-Speed Navigation in Unknown Environments". In: Jan. 2018, pp. 325–341. ISBN: 978-3-319-60915-7. DOI: 10.1007/978-3-319-60916-4_19.

[28] S. Sarid, Bingxin X., and H. Kress-gazit. "Guaranteeing high-level behaviors while exploring partially known maps". In: *RSS*. 2012.

[29] Sfu-Mars. *SFU-MARS/optimized_dp*. URL: https://github.com/SFU-MARS/optimized_dp.

[30] R. Volpe. "Task space velocity blending for real-time trajectory generation". In: *[1993] Proceedings IEEE International Conference on Robotics and Automation*. 1993, 680–687 vol.2. DOI: 10.1109/ROBOT.1993.291880.

[31] R. Walambe et al. "Optimal trajectory generation for car-type mobile robot using spline interpolation". In: *IFAC-PapersOnLine* 49.1 (2016), pp. 601–606.

[32] Rahee Walambe et al. "Optimal Trajectory Generation for Car-type Mobile Robot using Spline Interpolation**This work is carried out under the research project grant sanctioned under the WOS-A scheme by Department of Science and Technology (DST), Govt. of India." In: *IFAC-PapersOnLine* 49.1 (2016). 4th IFAC Conference on Advances in Control and Optimization of Dynamical Systems ACODS 2016, pp. 601–606. ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2016.03.121. URL: https://www.sciencedirect.com/science/article/pii/S2405896316301215.

[33] Xinrui Wang, Karen Leung, and M. Pavone. "Infusing Reachability-Based Safety into Planning and Control for Multi-agent Interactions". In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2020), pp. 6252–6259.

[34] Erik Wijmans et al. *DD-PPO: Learning Near-Perfect PointGoal Navigators from 2.5 Billion Frames*. 2020. arXiv: 1911.00357 [cs.CV].

[35] Fei Xia et al. "Gibson env: real-world perception for embodied agents". In: *Computer Vision and Pattern Recognition (CVPR), 2018 IEEE Conference on*. IEEE. 2018.

[36] Xiangru Xu et al. "Robustness of Control Barrier Functions for Safety Critical Control**This work is partially supported by the National Science Foundation Grants 1239055, 1239037 and 1239085." In: *IFAC-PapersOnLine* 48.27 (2015). Analysis and Design of Hybrid Systems ADHS, pp. 54–61. ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2015.11.152. URL: https://www.sciencedirect.com/science/article/pii/S2405896315024106.