UNIVERSITY OF CALIFORNIA SAN DIEGO

Efficient Control Barrier Function Refinement Using Local Hamilton Jacobi Reachability

A thesis submitted in partial satisfaction of the
requirements for the degree Master of Science

in

Engineering Sciences (Mechanical Engineering)

by

Alex Toofanian

Committee in charge:

Professor Sylvia Herbert, Chair
Professor Jorge Cortes
Professor Sicun Gao

2023

The thesis of Alex Toofanian is approved, and it is acceptable in quality and form for publication on microfilm and electronically.

University of California San Diego

2023

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT OF THE THESIS

Efficient Control Barrier Function Refinement Using Local Hamilton Jacobi Reachability

by

Alex Toofanian

Master of Science in Engineering Sciences (Mechanical Engineering)

University of California San Diego, 2023

Professor Sylvia Herbert, Chair

Safe control policies for nonlinear dynamic systems are notoriously challenging to derive. Hamilton-Jacobi reachability analysis (HJ reachability) provides guaranteed safety in the form of the optimal control policy, but its compute cost scales exponentially with state space. Neural learning provides an alternative approach with its compute scaling only against problem complexity, but yields only approximate results. Recently, neural policies in the form of control barrier functions (CBFs) have been used to warmstart HJ reachability, yielding a guaranteed safe result more efficiently. However, a significant amount of compute is still spent to shape the CBF into the HJ reachability result. This paper introduces HJ Boundary Marching, which adapts the mechanics of warmstarted HJ reachability to refine erroneous control barrier function boundaries

by minimally reshaping them to the nearest interior control-invariant boundary. This yields a guaranteed safe CBF for the same set as HJ reachability, with up to two orders of magnitude faster compute. A demonstration is provided on a 4-dimensional system with a learned neural CBF.

Chapter 1

Introduction

As autonomous systems become more entwined with modern life, their assured safe operation becomes an increasingly pressing research frontier. To guarantee safety, it must be definitively shown that a controlled system will continue to operate within the intended design space, despite any adversarial or unfortunate circumstances. In practice, it is computationally expensive to assess the set of all possibilities in order to claim that all circumstances have been considered and the design is safe. Algorithms for this robust approach, such as those used in Hamilton-Jacobi reachability analysis (HJ reachability) [1], become exponentially intractable as the dimension of the configuration space increases. More generally, safety analysis often relies on producing value functions which encode a measure of safety for a given state, and inform safe control. Control barrier functions (CBFs) are one such kind of function [2].

To address this intractability, controls engineers have begun to employ neural learning [3] to yield approximate control barrier functions. Importantly, the neural problem can be formulated such that its computational cost hinges not on problem dimensionality, but on the underlying problem complexity. However, the produced barriers are only approximate, since neural networks require infinite data and infinite size to become exact. This effectively trades the exact, heavy rigor of HJ reachability for the increased speed of approximate learning.

On the most recent frontier, researchers are pursuing the best of both worlds by piecing these two approaches together. "Warmstarting" HJ reachability with a control barrier function [4,

5] has been shown to efficiently yield a guaranteed safe result. However, current reachability compute techniques are still inefficient in how they leverage the information from close CBF initializations; the solvers update globally when only a small subset of the barrier may need updating, and they pursue the exact HJ reachability result rather than an adjacent but verifiable CBF.

This paper introduces an algorithm which refines erroneous control barrier function boundaries by minimally contracting them to the nearest interior control-invariant boundary. IT achieves this by adapting the standard HJ reachability framework, a formal safety technique based on discrete space dynamic programming, to focus solely on uncertified boundary regions. Through the use of an iterative algorithm, we can ensure that the proposed method converges to a barrier that is guaranteed to be forward invariant, and thus a verified CBF. By focusing only on the boundary, we achieve our result in up to 2 orders of magnitude faster compute than warmstarted HJ reachability. To demonstrate the effectiveness of our approach, we first apply it to a 2D active cruise control system as an illustrative example, and then extend it to a 4D vertical quadcopter with a candidate neural CBF.

Chapter 2

Related Works

2.1 Efficient Hamilton Jacobi Reachability

Hamilton Jacobi reachability analysis [1] is the standard workhorse used to provide numerically exact safety assurances through dynamic programming. However, its compute requirements scale exponentially with problem dimension, which severely limits its use on high dimensional systems. Active research on HJ reachability primarily focuses on addressing its gross computational requirements.

Critical for our application, it has been shown that reachability may be "warmstarted" [4]; by initializing with an appropriate guess, the total amount of compute time may be decreased. Time saved varies significantly on problem structure and the quality of the supplied initialization. Notably, for small local inaccuracies in the initialization, standard algorithms will wastefully perform compute over the entire space, rather than just in the region of need.

Local reachability [6] acknowledges that minor changes in the environment should only drive a change in analysis to those regions which are connected to the change. Therefore, the region of state space necessary to compute over can begin at the changed set and its neighbors, and grow following each update iteration. This improves on warmstarting by not requiring extensive compute over regions that are already accurate. The marching algorithm defined in this paper takes inspiration from this approach, but introduces an even stronger filter on which states need be updated in order to propagate the necessary information.

For systems of certain advantageous underlying form, alternative solutions have been proposed. Where possible, system decomposition [7] attempts to address the curse of dimensionality by conservatively splitting the system to separate, lower dimension problems and solving over that space. Alternatively, if the system is linear, Hopf optimization [8] rather than dynamic programming allows for exceedingly efficient results by using gridless approaches.

Each of these methods somewhat mitigates the exponential compute cost that HJ reachability suffers from, while preserving the exact safety guarantees that make the method desirable. Two fronts are observed, one optimizing the compute algorithm (warmstarting, local reachability), and another reworking the underlying assumptions on the system (system decomposition, linearization). The research provided in this paper is of the former variety, presenting a reduced compute approach for safety.

2.2 Connecting Reachability to Control Barrier Functions

The recently popularized field of control barrier functions (CBF) [2] seeks to provide similar safety assurances as Hamilton Jacobi reachability by defining a value function which prescribes a control-invariant safe set, and which defines the necessary invariance-enforcing control to remain within the safe set. They leave the robust mechanisms of Hamilton Jacobi reachability for a more abstract approach, and are satisfied with resolving not the entire set of safe states, but some subset which itself is necessarily safe. However, the control barrier function framework provides no general constructive method. Connecting control barrier functions, which have a more flexible form, and reachability, which has a robust constructive formulation, has opened doors for wider verifiably safe compute methods.

Control Barrier Value Functions [9] links the methodical architecture of HJ reachability to the control barrier function framework, providing a constructive method for obtaining barriers with desirable safety properties, that satisfy the control barrier function requirements. With this interface opened, it was next shown that the principles of warmstarting extend to the CBVF;

candidate control barrier functions, constructed imperfectly through whatever abstract means, may be treated as an efficient initialization and refined with reachability [5].

2.3 Learning for Safe Control

Neural learning approaches have been employed as an alternative construction method for safety certificates. The results can be significantly more computationally efficient; suffering less from the curse of dimensionality, and rather scaling with the complexity of the underlying value function. Dawson [3] defines a canonical way to formulate CBFs, and Bansal [10] goes as far as to learn the true HJ reachability result.

However, neural barriers can be imperfect. The ability of a neural network to resolve an adequate CBF hinges on appropriate network structure to meet the underlying problem, and extensive sampling and training. Therefore, in practice, neural barrier functions are often inaccurate and will have leaky boundary regions. Recent work to bound the gradients of neural networks, used to analyze against antagonistic examples [11], has tangential benefits for the neural CBF community by enabling the analytical certification of the barrier [12].

Chapter 3

Preliminaries

3.1 Safety for Dynamic Systems

The mathematical framework through which we may model the maneuvering of an agent through its environment is the dynamic system. We assume the system has state $x \in \mathbb{R}^n$, which changes in time following the differential equation:

$$\dot{x} = f(x,\, u), \qquad u \in \mathcal{U} \tag{3.1}$$

where u is a user-controlled input from the compact set of possible inputs $\mathcal{U}$. The control signal over time, $u(\cdot)$, is necessary to fully define the time evolution of this system, and is assumed to be drawn from the set of measurable functions $\mathbb{U} : [t, 0] \to \mathcal{U}$. Notably, the evolution of this system is not dependent on the system time t; a necessary restriction for the infinite-time dynamic programming solutions pursued in this paper.

It is assumed that the flow field $f : \mathbb{R}^n \times \mathcal{U} \to \mathbb{R}^n$ is uniformly continuous in time and Lipschitz in x for fixed u. These assumptions yield a unique solution of system dynamics for a given starting state and control signal, referred to as the trajectory:

$$\xi(\tau;\, x,\, u(\cdot)) \tag{3.2}$$

This notation is read as the state $\xi \in \mathbb{R}^n$ reached at time τ by beginning at state x and following

$u(\cdot)$ over time τ.

This dynamic system operates over the space defined by $X \in \mathbb{R}^n$. In practice, the controls engineer may define a subset of space which is undesirable, $\mathcal{L} \subset X$. Safety, in the sense used throughout this paper and the safe controls community at large, is the assurance that the trajectory ξ never enters the undesirable set $\mathcal{L}$. For any given control signal, the set of states for which this is true is the safe set $S := \{x \mid \xi(\tau;\, x, u(\cdot)) \notin \mathcal{L} \;\forall\; \tau\}$. The maximal set of states from which there exists at least one control option that preserves safety is identified as the viability kernel $\mathcal{K}$.

A barrier function may be used to define a forward invariant set C for an autonomous, or fixed policy, system $f(x)$. A barrier function is a continuously differentiable function h for autonomous dynamics $f(x)$ if there exists a non-negative scalar γ such that:

$$C := \{x \mid h(x) \geq 0\}, \qquad \dot{h}(x) := L_{f(x)}h(x) \geq -\gamma h(x) \tag{3.3}$$

This has direct application to safety, since if the invariant set C does not intersect with the failure set $\mathcal{L}$, then C is a safe set S. This framework is extended to the controlled system $f(x,u)$ as a control barrier function [2]:

$$\sup_{u \in \mathcal{U}} \dot{h}(x) := \sup_{u \in \mathcal{U}} L_{f(x,u)}h(x) \geq -\gamma h(x) \tag{3.4}$$

While the existence of a barrier function provides a safety assurance for an autonomous system (or equivalently, a controlled system with a fixed policy), the existence of a control barrier function provides the general assurance that safe policies do exist for the system. Moreover, the control barrier function may used as a safety preserving filter by solving the following

optimization problem:

$$\begin{aligned} u^*(x) = \arg\min_u \quad & \|u - \pi(x)\|_R^2 \\ \text{subject to} \quad & \dot{h}(x) + \gamma h(x) \geq 0 \\ & u \in \mathcal{U} \end{aligned} \tag{3.5}$$

Where $\pi(x)$ is some nominal, safety-agnostic policy. Following this, even if no nominal policy is provided, the control barrier function still yields a safety preserving control, and for this reason may itself be viewed as a safety preserving policy.

3.2 Hamilton Jacobi Reachability

Hamilton Jacobi Reachability is a constructive method that can be used for generating a control barrier function $h(x)$ which encodes the optimal safe control policy and the viability kernel $\mathcal{K}$. Beginning with a failure set $\mathcal{L} := \{x \mid \ell(x) < 0\} \subset X$, where $\ell(x)$ is a reward function (often denoting proximity to the failure set, such as signed distance), the HJ reachability value function is defined as

$$V(x,t) = \max_{u \in \mathcal{U}_{[t,0]}} \min_{\tau \in [t,0]} \ell(\xi(\tau; x, u(\cdot))) \tag{3.6}$$

which finds the control signal $u(\cdot)$ that maximizes the minimum value of ℓ obtained over the trajectory ξ. For an infinite time horizon (as $t \to -\infty$), the 0-superlevel set of $V^*(x) = \lim_{t \to -\infty} V(x,t)$ defines the infinite-time viability kernel $\mathcal{K} := \{x \mid V^*(x) \geq 0\}$.

The focus of our work is solely on persistent safety, matching the CBF formulation. Therefore we drop the explicit time-dependence. To solve for the HJ reachability value function, we adopt a discrete state space and use dynamic programming to perform value iteration on the initialized value function [13]:

$$V_k(x) = V_k(x) + \Delta_k \min\{0, \max_{u \in \mathcal{U}} L_{f(x,u)} V_k(x)\} \tag{3.7}$$

where the subscript k denotes the iteration of the algorithm. In practice, Δ_k is dynamically updated based on the size of the Hamiltonian, $H(x)$.[1]

$$H(x) = \max\{0, \max_{u \in \mathcal{U}} L_{f(x,u)} V_k(x)\} \tag{3.8}$$

The outer minimization in the Hamiltonian forces the maximum Hamiltonian to be zero, such that values can only ever reduce, and upholding the minimum safety over time requirement from equation 3.6 as established in [14].

3.3 Learning-based Safety Analysis for Control

Neural networks can be employed to learn control policies and barrier functions. Standard policy optimization algorithms such as Soft Actor-Critic (SAC) [15] (as used in section 5.2) exist to learn a control policy $g(\cdot) : X \to \mathcal{U}$ using a reward function that encourages goal reaching and penalizes constraint violations. After the training of neural policy converges, the agent should show reasonably good behavior in reaching the goal while avoiding obstacles. To certify this policy, is is useful to train a barrier function for the fixed-policy system.

For the barrier used in section 5.2, a set of trajectories is sampled from randomized initial positions in the state space. The states of trajectories that reach the goal without constraint violation form the safe training set X_s, and those that violates safety constraints form the unsafe training set X_u. After labeling the safe and unsafe states, the barrier is trained using the following loss function:

$$L(\theta) = w_s \frac{1}{N_s} \sum_{i=0}^{N_s} \phi(-B(x_s^i)) + w_u \frac{1}{N_u} \sum_{i=0}^{N_u} \phi(B(x_u^i)) + w_l \frac{1}{N_s} \sum_{i=0}^{N_s} \phi(-\mathcal{L}_f B(x_s^i) - \gamma B(x_s^i))) \tag{3.9}$$

where $\phi(x) = \max(x, 0)$, and w_s, w_u, w_l, γ are positive parameters for balancing the weights of

[1]The Hamiltonian typically would only refer to the interior $L_{f(x,u)} V_k(x)$ term. However, for the sake of conciseness we adapt it here to include the full term which governs the value update.

the different components of the loss. The three terms in $L(\theta)$ correspond to the satisfaction of the three standard conditions for establishing barrier functions. For instance, the first term measures the average values of safe states with negative barrier function outputs, and it reaches zero only when the barrier function value is positive on all sampled "safe" states. Overall, $L(\theta)$ is always non-negative, and it reaches global minimum $L(\theta) = 0$ if and only if the barrier conditions in Equation 3.3 are completely satisfied.

Learned barriers are by no means perfect, since they require infinite training data and appropriate structure to verifiably have $L(\theta) = 0$. Accordingly, an almost-barrier function is a relaxation of the standard notion of barrier certificates by allowing regions on the safety barrier to be uncertified; certification meaning that the system is quantitatively known to meet the barrier condition (equation 3.4) at the particular state, x, s.t:

$$C := \{x \mid \sup_{u \in \mathcal{U}} L_{f(x,u)} h(x) \geq -\gamma h(x)\}. \tag{3.10}$$

This relaxed notion of barrier functions no longer implies safety in the standard sense. There is now a "leaky" area of the zero-level set from which the system may not meet the barrier condition, and therefore may cross over to the unsafe area X_u. Techniques exist to analytically bound neural network gradients, which facilitates the certification of neural almost-control barrier functions [12, 16, 17, 18, 19].

Chapter 4

HJ Reachability Boundary March

Algorithm 1. Hamilton Jacobi Boundary March

Require: $V_0(x) = \ell(x)$, **Optional:** C, set of oracle-certified cells

$\hat{Q}_0 \leftarrow \{x \mid |V_0(x)| < \varepsilon\}$ ▷ get boundary cells

$Q_0 \leftarrow \hat{Q}_0 \setminus C$ ▷ remove oracle-certified cells

$Q_k \leftarrow Q_0,\ V_k \leftarrow V_0$ ▷ initialize loop

while Q_k is not empty **do** ▷ Q_k not empty, boundary may be leaky!

  $H(x) \leftarrow \min\{0, \max_{u \in \mathcal{U}} L_{f(x,u)} V_k(x)\},\ \forall\, x \in Q_k$ ▷ hamiltonians over Q_k

  $Q_k^- \leftarrow \{x \mid H(x) < 0,\ \forall\, x \in Q_k\ \}$ ▷ leaky boundary cells, if none then done!

  $V_{k+1}(x) \leftarrow V_k(x) + \Delta_k H(x),\ \forall\, x \in Q_k^-$ ▷ decrease values where leaky

  $Q_{k+1} \leftarrow dilate(Q_k^-) \cap \{x \mid |V_{k+1}(x)| < \varepsilon\}$ ▷ unsafe boundary of V_{k+1} neighbors Q_k^-

  $Q_k \leftarrow Q_{k+1},\ V_k \leftarrow V_{k+1}$ ▷ increment loop

end while

if $V(x) < 0\ \forall\, x$ **then** ▷ Loop ended because boundary does not exist.

  return Failure ▷ No Valid CBF Found.

else

  return $V_k(x)$ ▷ Boundary exists and is not leaky, $V_k(x)$ is a CBF!

4.1 Overview

The Hamilton-Jacobi Boundary March of Algorithm 1 resolves the maximal control invariant subset of an arbitrary initial set by minimally and locally updating the corresponding value function. Specifically, the algorithm obtains the viability kernel and associated value function while updating the minimum cumulative number of grid cells. As demonstrated in Section 5, for nearly accurate almost-barrier functions, the proposed algorithm provides significant speedups, allowing formal safety analysis on systems of up to 2 orders of magnitude greater than conventional methods.

The key focus of this algorithm is to perform the minimal number of cell value updates to reveal the viability kernel, with only the information achieved through the Hamilton-Jacobi reachability computation. Rather than updating over the entire grid, the algorithm updates only the subset of cells which carry the driving "unsafe" information. These cells will naturally begin on the boundary of the failure set, then propagate inward delineating the unsafe frontier. In this process, if there is no viability kernel for the system, the boundary will contract until it becomes the empty set. In the expected case where there is a viability kernel, the boundary will contract until the unsafe frontier vanishes, indicating that the updated boundary has reached the control-invariant viability kernel.

The algorithm (alg 1) begins with an initial value function $V_0(x) = \ell(x)$; this should be a well structured target function such that the failure set $\mathcal{L} := \{x \mid \ell(x) < 0\}$. For approximate CBF initializations, the zero-levelset of $V(x)$ may contain some unsafe, "leaky" regions, Q, from which the dynamics will inevitably flow across the boundary into unsafe space. By oracle (step 2), or by computing the Hamiltonian (equation 3.8) over the boundary (step 5-6), we can identify these unsafe cells. Furthermore, by decreasing unsafe cell values in accordance with the Hamiltonian, we act to exclude them and only them from the running viability kernel (step 7). At each iteration, the boundary will contract, with a corresponding new potentially leaky region which requires assessment (step 8). This process is depicted for one iteration on a running

example system in figure 4.1. The algorithm iterates following this principle until a boundary with no leaky region is found. This boundary either prescribes a valid CBF for the viability kernel, or the empty set if no viability kernel exists for the system.

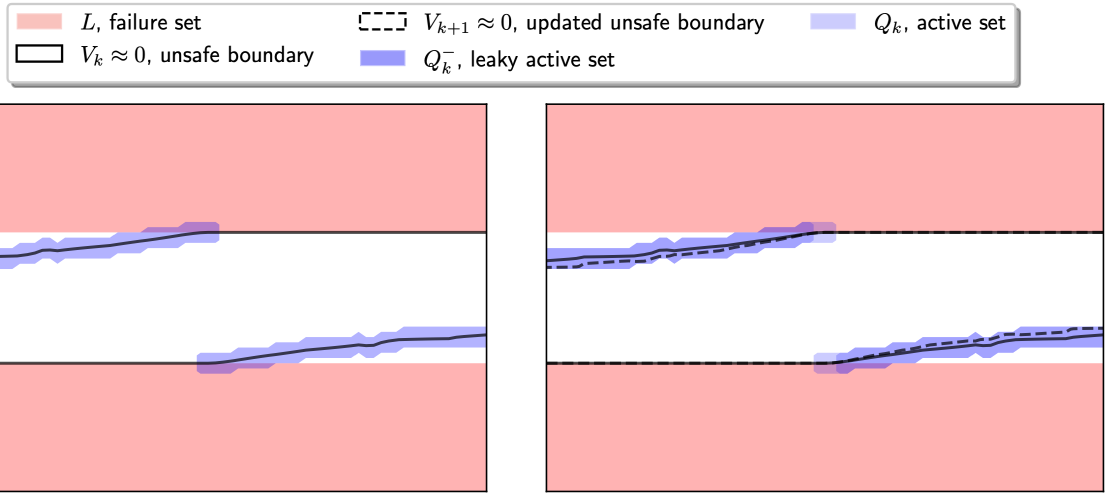
The result is a value function which resolves the true viability kernel from traditional HJ reachability. Compared with the value function from standard HJ reachability, the resulting function from Algorithm 1 will have steeper gradients around the boundary, flat gradients outside, and under-evaluated values elsewhere. Because this approach does not break any of the fundamental rules of value iteration and dynamic programming, we claim that the zero levelset is the true viability kernel, and the gradients at the boundary may still be used to preserve set invariance – therefore the refined value function is a valid CBF for the viability kernel.

4.2 Discussion and Possible Applications

The Boundary Marching algorithm is effectively a lighter Hamilton Jacobi reachability formulation which skims a small subset of the state space to find the minimally restrictive control for the viability kernel.

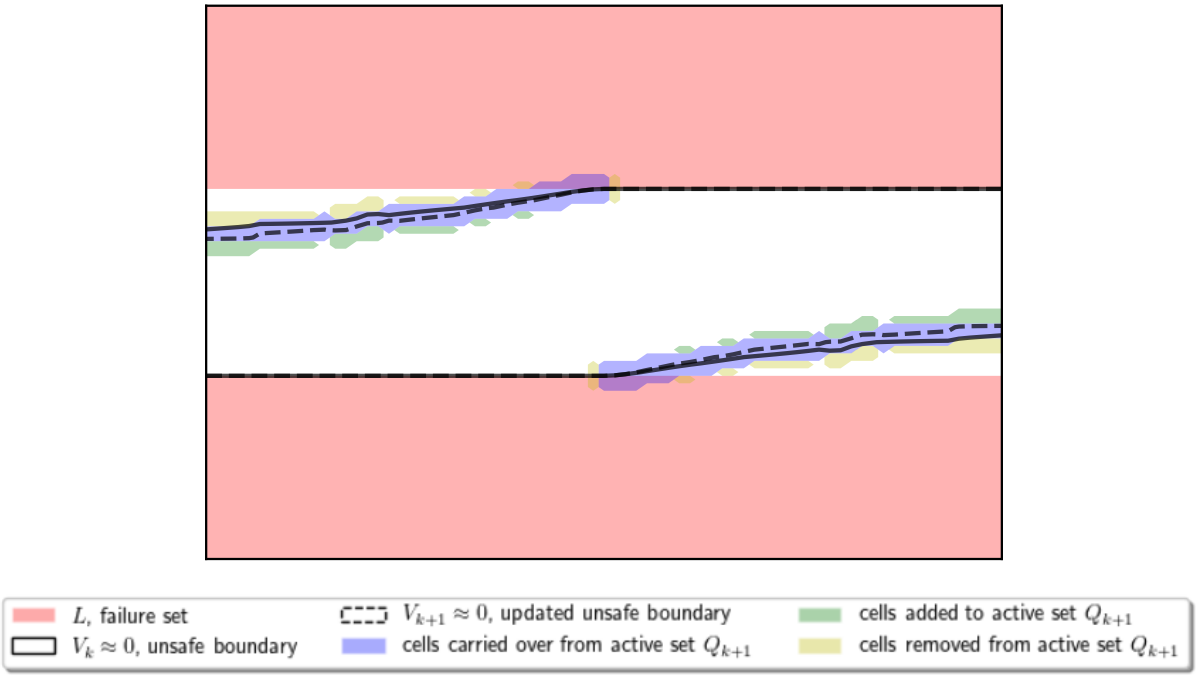
As the primary motivating application for this work, we find CBF verification can be rapidly performed using the Boundary March. The initial CBF boundary $\{x \mid h(x) < 0\}$ may be used as the failure set $\mathcal{L}$, and only one iteration of the march is necessary to show whether the boundary is truly control invariant. If not, successive iterations will yield a CBF refined to the nearest interior barrier. A demonstration of application is provided in section 5.

It is theorized that the algorithm also may be used to speed up the standard HJ reachability compute time by using the Boundary March to provide an initial topology of the value space. For example, in a near future where adaptive, nonuniform grids are leveraged in discrete space reachability analysis, it will be useful to perform a quick Boundary March to find the viability kernel to better inform the grid construction, before computing the full reachability result. Lacking the adaptive grid technology, a demonstration is not provided in this paper.



(a) A subset of the boundary is flagged as active, where hamiltonians will be computed.

(b) Unsafe cells have their values updated, moving the unsafe boundary.



(c) The active set marches to follow the unsafe frontier.

Figure 4.1. Single-step illustration of the Hamilton Jacobi Boundary March on an arbitrary system.

4.3 Proof

What follows is a proof of Algorithm 1, which provides an approximate dynamic programming approach to constructing a control barrier function for the viability kernel of a given system. It does this by marching an unsafe boundary frontier inward from the failure set, until the frontier collapses, indicating all cells on the running boundary are safe. By computing only on the unsafe boundary rather than the full grid, the compute requirements are significantly reduced, allowing problems of up to two orders of magnitude greater complexity to be analyzed.

Theorem 1. *The Boundary March algorithm yields a value function which has a 0-superlevel set equivalent to the viability kernel corresponding to the failure set, i.e.:*

$$\{x \mid V(x) \geq 0\} = K$$

.

Theorem 2. *The Boundary March algorithm yields a control barrier function, i.e.:*

$$\max_{u \in \mathcal{U}} L_{f(x,u)} V(x) \geq -\gamma V(x)$$

Proof.

Begin with an initial value function $V_0(x)$ that comes from the set of viable target functions $\ell(x)$, such that:

$$V_0(x) = \ell(x)$$

Lemma 1. *If the set B excludes only unsafe cells, the viability kernel of L is a subset of B.*

Define the zero-superlevelset of $V_0(x)$ as the initial kernel $B_0 := \{x \mid V_0(x) \geq 0\}$. By Lemma 1, the viability kernel is a subset of B_0.

Next, extract the cells which intersect with the boundary:

$$Q_0 := \{x \mid |V_0(x)| < \varepsilon\}$$

(At this point, the marching loop begins, so the subscripts k and $k+1$ are adopted to denote the current and the next iteration: $Q_k \leftarrow Q_0,\ V_k \leftarrow V_0,\ B_k \leftarrow B_0$.)

Lemma 2. *The set B is control-invariant if and only if all cells on its boundary have non-negative Hamiltonian.*

By Lemma 2, Q_k is the minimum set of cells which if certified would indicate that B_k is control-invariant. To determine if these cells are safe, we perform a two-phase certification step. First, we allow an opportunity for the oracle to certify safe cells in Q_k, by providing a list of cells C_k, such that:[1]

$$H_k(x) \geq 0,\ \forall\, x \in C_k$$

$$Q_k := Q_k \setminus C_k$$

After the optional oracle, what remains in Q_k is the subset of boundary that may be unsafe. The second phase of certification comes from HJ reachability, where the Hamiltonian (equation 3.8) is computed directly over those cells in Q_k:

$$H_k(x) = \max\{0, \max_{u \in \mathcal{U}} L_{f(x,u)} V_k(x)\},\ \forall\, x \in Q_k$$

With the Hamiltonian computed, it is trivial to flag Q_k^- as the set of boundary cells which may be unsafe.

$$Q_k^- := \{x \mid H_k(x) < 0,\ \forall\, x \in Q_k\}$$

[1] In Algorithm 1, oracle certification is performed only a single time, prior to the loop. This is because, in practice, we only expect to have oracle certification for the initial value function. Barring unforeseen advancements in certification, it would be both expensive and inefficient to generate a new oracle certification for each $V_k(x)$ (by whatever means, such as in [12]). However, it is theoretically acceptable, and shown here as such.

Lemma 3. *If $\hat{V}_{k+1}$ is the value function updated globally, and V_{k+1} is the value function updated locally, $\hat{V}_{k+1} \leq V_{k+1}$ and $\hat{B}_{k+1} \subseteq B_{k+1}$.*

Lemma 4. *Warmstarting HJ reachability property [4]: Initializations greater than than the optimal value function will contract to the optimal value function.*

By Lemmas 3 and 4, we can locally update the value function over Q_k^- to yield a contracted value function which still is a conservative initialization, and therefore retains the viability kernel, upholding Lemma 1.

$$V_{k+1}(x) = V_k(x) + \Delta_k H_k(x), \ \forall \ x \in Q_k^-$$

Lemma 5. *For Lipschitz dynamics, flows follow continuous trajectories. Therefore, the combined set of failure and unsafe cells is compact.*

To iterate the algorithm, it is necessary to define a new set of possible unsafe states Q_{k+1} which correspond to V_{k+1}, to continue to uphold Lemma 2. By Lemma 5, Q_{k+1} must exist within Q_k^- and its immediate neighbors, restricted to the boundary cells $\{x \mid |V_{k+1}(x)| < \varepsilon\}$.

$$Q_{k+1} \leftarrow dilate(Q_k^-) \ \cap \ \{x \mid |V_{k+1}(x)| < \varepsilon\}$$

This concludes the iteration k, and sets up iteration $k+1$ with the same initial premise of Lemmas 1 and 2. At each iteration, the value function has a contracted 0-superlevel set B_k which contains all safe cells, and will continue to contract until Q_k^- is empty, indicating that B_k is either empty or control-invariant. If B_k contains all safe cells and is control invariant, it is necessarily the viability kernel, and therefore V_k is a control barrier function for the viability kernel.

□

Chapter 5

Experiments

The algorithm is implemented in the the *refineNCBF* python package (*github.com/toofanian/refineNCBF*). This package uses the backend Hamilton Jacobi reachability solver from Optimized Dynamic Programming [20], with modifications made to allow selectively updating cells in the value iteration process (*github.com/toofanian/optimized_dp*).

5.1 Boundary Marching for Active Cruise Control CBF Construction

Active cruise control serves as a useful demo problem since it has intuitive dynamics and both dimensions can be visualized. Here, we define a failure set, and allow the Boundary March and HJ Reachability to independently compute a control barrier value function result, then compare the two approaches.

5.1.1 Problem Setup

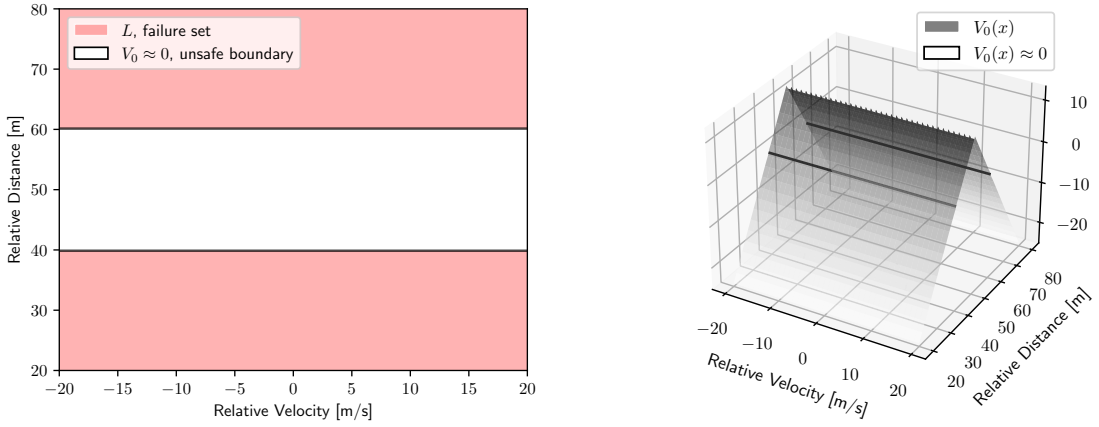
The simplified active cruise control system follows the dynamics:

$$\dot{v} = \frac{u}{m}, \qquad \dot{r} = v \tag{5.1}$$

where v is the relative velocity between the lead car and the agent car, r is the relative distance between the lead car and agent car, u is the acceleration control. Higher fidelity effects such as

Table 5.1. Parameters for the tested configuration of active cruise control (equation 5.1).

Active Cruise Control Parameters				
$u_{\min}$	$u_{\max}$	m	$r_{\min}$	$r_{\max}$
-4856 N	4856 N	1650 kg	40 m	60 m



(a) State space, with failure set.

(b) Initial value function.

Figure 5.1. Initialization for Active Cruise Control Reachability Analysis.

friction are ignored with no major impact on system behavior. For this demonstration, the agent car is required to stay within some range $(r_{\min},\ r_{\max})$ of the lead car. The acceleration control u is bounded between $(u_{\min},\ u_{\max})$, which makes the safety problem nontrivial. Exact parameters for the configured experiment are provided in table 5.1.

Figure 5.1a shows the state space with the failure set marked, while figure 5.1b shows the initial value function (signed distance to the failure set). No oracle certification is used (Algorithm 1, step 2), so the Boundary March is initialized along the entire boundary span. Intuitively, we can expect that when the agent car is too far from the lead car and has negative relative velocity (bottom-left region in figure 5.1a), it must be able to accelerate sufficiently fast elsewise it will fall too far behind. The HJ reachability and Boundary Marching solution should reflect this and similar behavior.

5.1.2 Results

The resultant viability kernel is separately computed using standard HJ reachability [1] and using the Boundary March, with final value result shown in figure 5.2. The yielded zero-superlevelsets of both algorithms closely match. Since the HJ reachability zero-superlevelset is the known to be the ground-truth viability kernel, this affirmatively demonstrates the ability of the Boundary March to resolve the viability kernel. However, since the Boundary March is restricted to only updating values along the running boundary, values elsewhere are undercomputed. Outside the boundary, where the march has skimmed over, is a region with near-zero gradients. Inside the boundary, where the march has never evaluated, are the original signed distance values. Thus, the control barrier value function yielded by the march is not optimal in terms of guaranteeing feasibility of the CBF-QP when inside the safe set, but is still safety preserving.

Successive iterations of each algorithm are shown for HJ reachability and Boundary Marching in figures 5.3 and 5.4 respectively. Each approach shows a similar evolution of the running kernel. In both, the kernel initially contracts from the upper-left and lower-right. The difference comes in the activated cells: the standard algorithm computes globally, agnostic of which cell updates contain significant information, whereas the marching algorithm strictly follows the flow of unsafety from the failure set.

Critically, the Boundary March is able to resolve a control barrier value function for the viability kernel with significantly less compute than required by HJ reachability. Figure 5.5 compares the total hamiltonians computed for the two approaches, and shows an order of magnitude reduction afforded by the Boundary March while maintaining kernel accuracy.

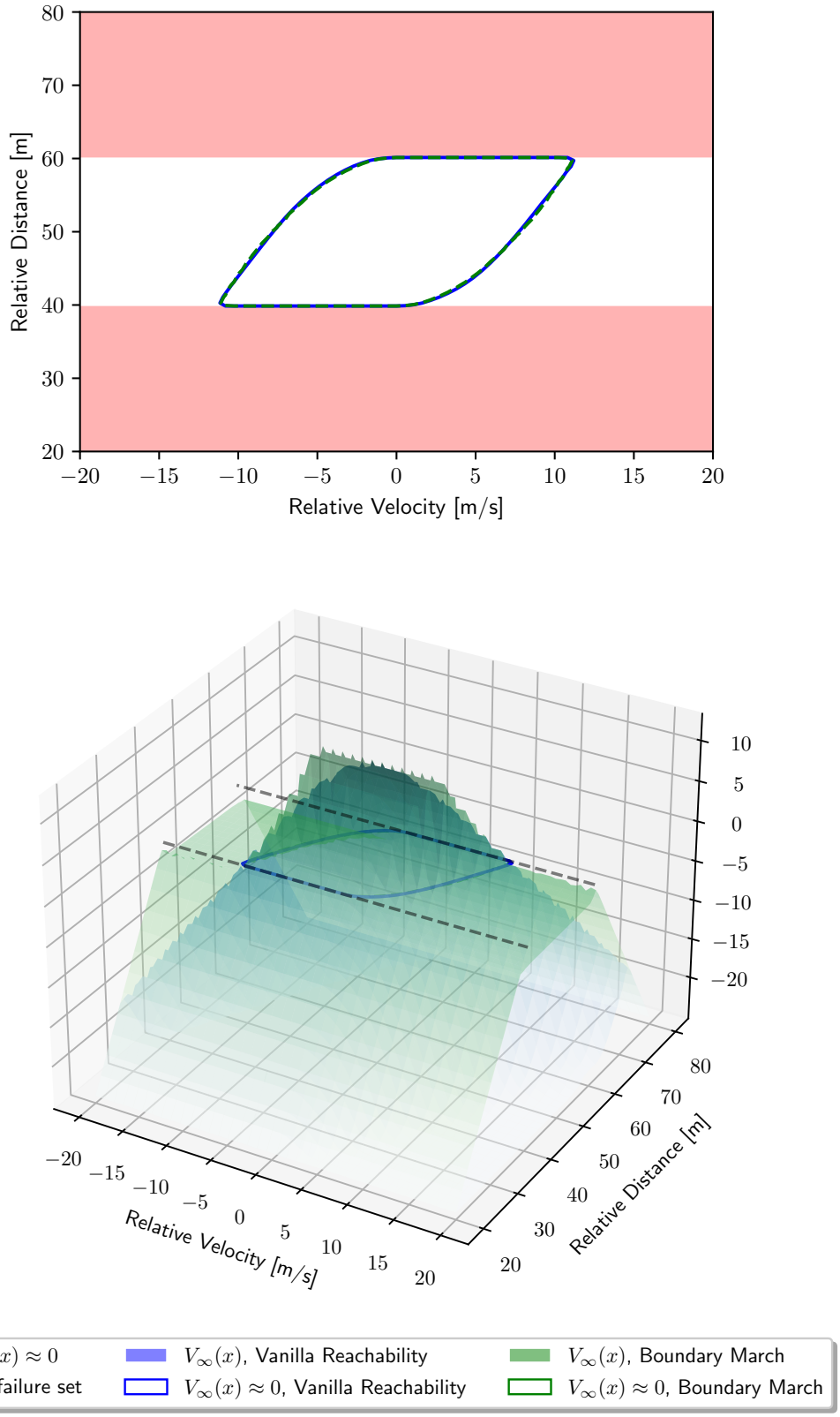
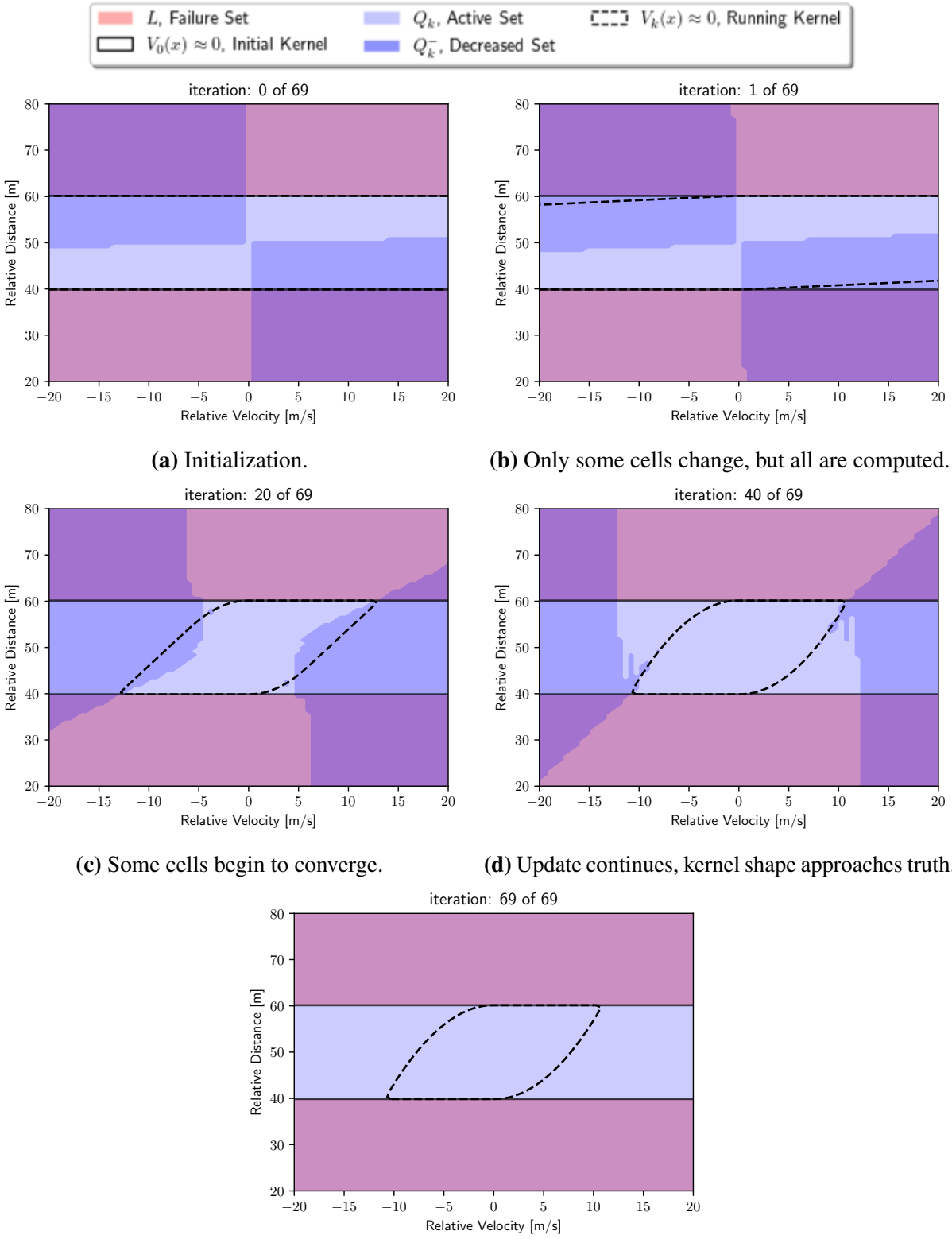


Figure 5.2. Converged Value Function Comparison for Active Cruise Control Reachability Analysis.



(a) Initialization.

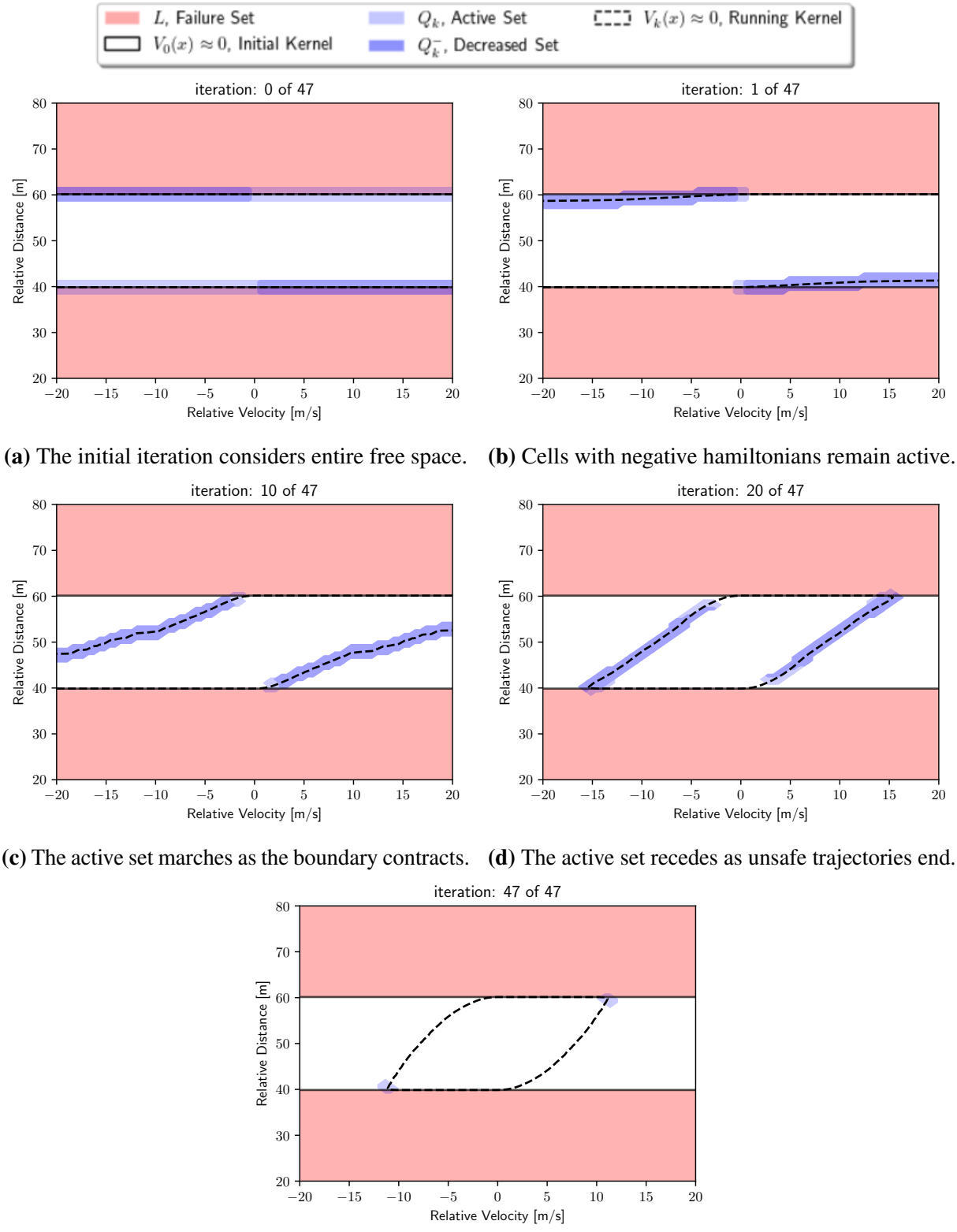
(b) Only some cells change, but all are computed.

(c) Some cells begin to converge.

(d) Update continues, kernel shape approaches truth.

(e) Update converges to viability kernel.

Figure 5.3. Vanilla HJ Reachability resolves the Active Cruise Control viability kernel



(a) The initial iteration considers entire free space. **(b)** Cells with negative hamiltonians remain active.

(c) The active set marches as the boundary contracts. **(d)** The active set recedes as unsafe trajectories end.

(e) The march converges.

Figure 5.4. HJ Boundary March resolves the Active Cruise Control viability kernel

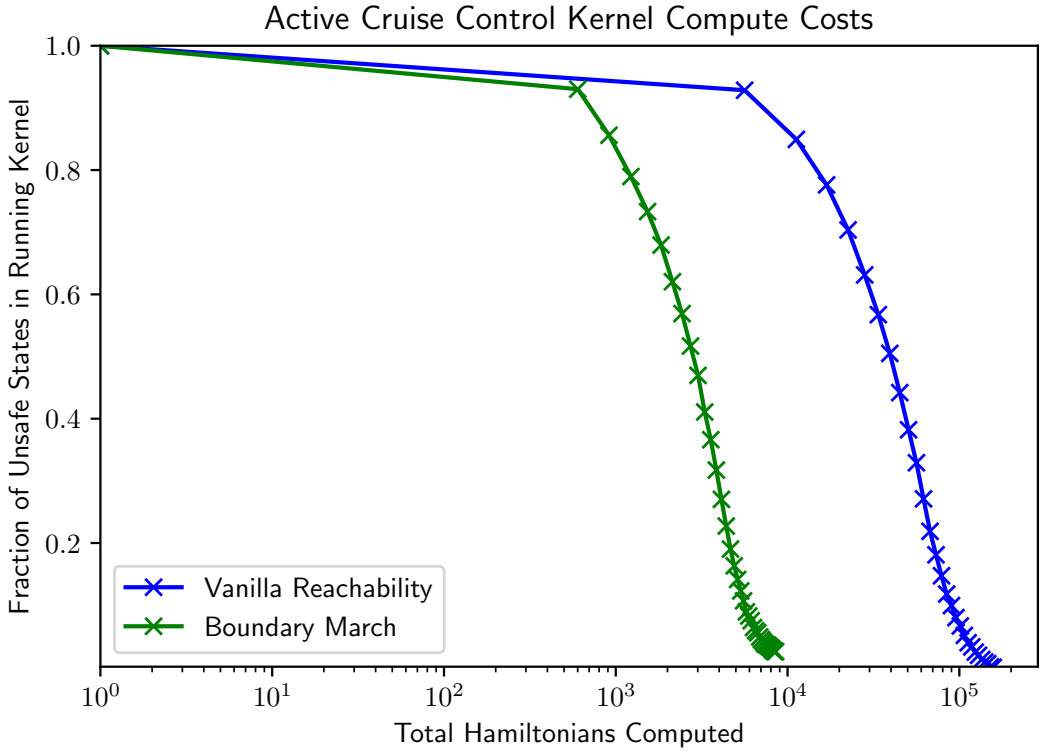


Figure 5.5. Hamiltonian computes required to resolve the active cruise control viability kernel using either Boundary March or global reachability. Computed on a 75x75 uniform grid.

5.2 Boundary Marching for Vertical Quadcopter Neural Control Barrier Function Refinement

Here, we train a neural control barrier function, and use it as an initialization to the Boundary March. This same initialization is also supplied to HJ reachability, and the results are compared. This is the primary motivating application, since the Boundary Marching algorithm benefits maximally from a close boundary initialization, and there is great need for efficient neural barrier certification methods.

5.2.1 Problem Setup

The vertical quadcopter dynamics are as follows:

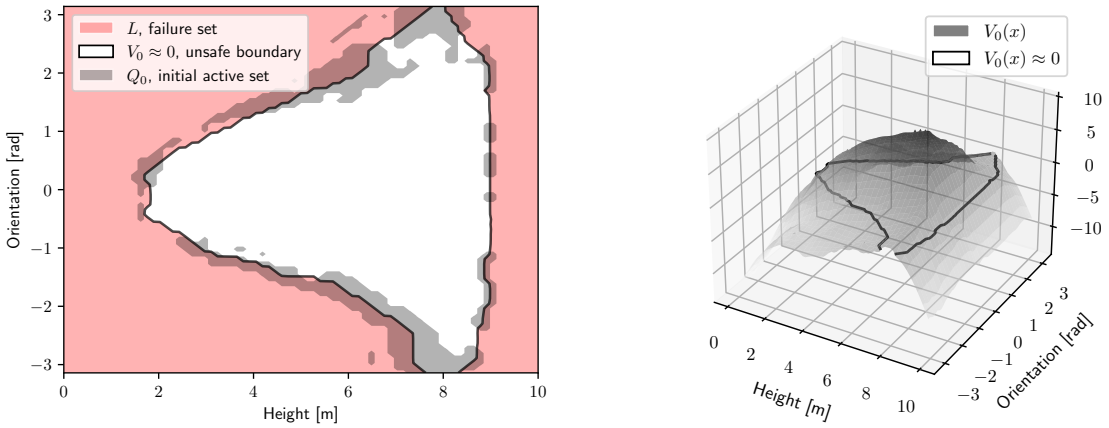
$$\begin{bmatrix} \dot{z} \\ \dot{v}_z \\ \dot{\phi} \\ \dot{\omega} \end{bmatrix} = \dot{X} = \begin{bmatrix} v_z \\ \frac{(T_1+T_2)\cos(\phi)-C_D^v v_z}{m} - g \\ \omega \\ \frac{(T_2-T_1)l-C_D^\phi \omega}{Iyy} \end{bmatrix} = \begin{bmatrix} v_z \\ \frac{-C_D^v v_z}{m} - g \\ \omega \\ \frac{-C_D^\phi \omega}{Iyy} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ \frac{\cos(\phi)}{m} & \frac{\cos(\phi)}{m} \\ 0 & 0 \\ -\frac{l}{I_{yy}} & \frac{l}{I_{yy}} \end{bmatrix} \underbrace{\begin{bmatrix} T_1 \\ T_2 \end{bmatrix}}_{u} \tag{5.2}$$

where z is the vertical position, v_z is the vertical velocity, ϕ is the roll angle, and ω is the roll velocity. Propeller torque is controlled directly as $u := [T_1,\ T_2]$, and the dynamics are subject to the parameters: mass m, vertical drag coefficient C_D^v, inertial drag coefficient C_D^ϕ, gravity g, moment of inertia I_{yy}, and length from propellers to center l. The quadcopter is required to hover within a prescribed height range $(r_{\min},\ r_{\max})$, with control bounded between $(u_{\min},\ u_{\max})$. Exact parameters for the configured experiment are provided in table 5.2.

A high-performing neural policy, $\pi(x)$, was developed for the vertical quadcopter system

Table 5.2. Parameters for the tested configuration of vertical quadcopter (equation 5.2)

Vertical Quadcopter Parameters									
$u_{\min}$	$u_{\max}$	m	C_D^V	C_D^ϕ	g	I_{yy}	l	$z_{\min}$	$z_{\max}$
0 N	18.45 N	2.5 kg	0.25	0.02255	9.81 m/s^2	1 kg·m^2	1 m	1 m	9 m



(a) State space, with failure set and initial active set.

(b) Initial value function.

Figure 5.6. Initialization for Warmstarted Quadcopter Vertical Reachability Analysis. Sliced at $v = 0$ and $\omega = 0$.

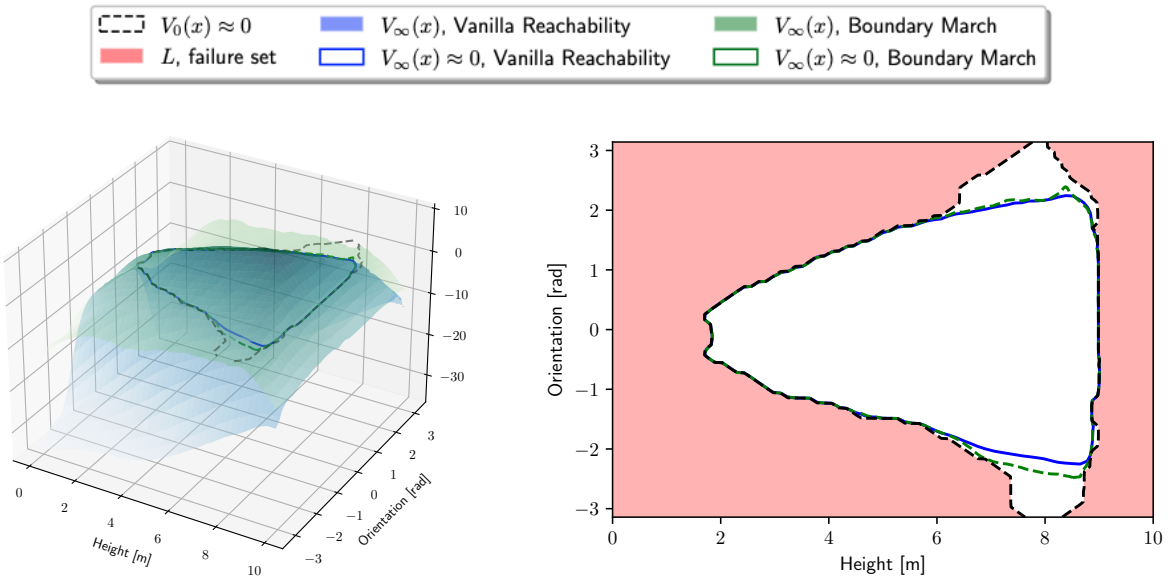
using the Soft Actor-Critic standard policy optimization algorithm [15]. The reward function encourages goal achievement while penalizing collisions with obstacles. Next, a data-driven approach was used to train a neural barrier function, $h(x)$, on the fixed-policy system, $f(x, u = \pi(x))$, using a loss function (equation 3.9) that specifies the desired characteristics of the barrier functions. The trained barrier function was analyzed to identify "leaky" areas on the boundary through a certification process which leverages (1) neural network analysis tools to identify network derivative over-estimations in a small neighborhood around the boundary [16, 17, 18, 19], and (2) sampling-based estimation to determine the system dynamics in those regions. Those regions with positive lower bounds are certified, C. To initialize the reachability problem, the failure set is defined as 0-sublevelset of this barrier, $\mathcal{L} := \{x \mid h(x) < 0\}$, and the certified cells are used to define a reduced initial active set $Q_0 := \hat{Q}_0 \setminus C$; both shown in figure 5.6a. The initial values are reinitialized from $h(x)$ to signed distance to $\mathcal{L}$, as shown for a portion of the 4-dimensional state space in figure 5.6b.[1]

[1]This reinitialization is theoretically unnecessary, but in practice is more assured to provide well behaved gradients and thereby a more useful $\ell(x)$ than what may exist in the discretized neural barrier function. This is a shortcut used in this test, rather than a necessity.

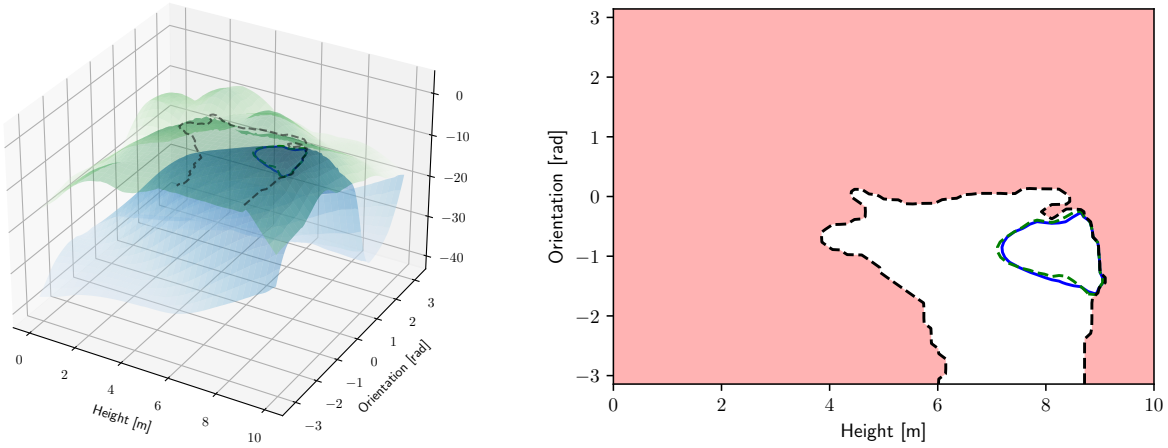
5.2.2 Results

The resultant viability kernel is separately computed using vanilla HJ reachability [1] the Boundary Marching algorithm (1), with the final value results compared in figure 5.7. The 0-superlevelsets of both algorithms closely match, though less-so than in the active cruise control case (5.1). The kernel is still substantially refined from the neural control barrier function initialization, as especially shown in figure 5.7b. This inaccuracy is believed to be due to the increased sensitivity of the Boundary Marching algorithm to numerical error, due to its steeper value gradients. Additional discussion is provided in section 5.3.

The compute for the HJ reachability update was performed on our server in 2207 seconds, with the first iteration (including compile) taking 182 seconds. The compute for the Boundary March was performed in 520 seconds, with the first iteration taking 88 seconds. This time includes several unoptimized convenience functions called every iteration, and can be improved considerably. The core solver time is linearly proportional to the number of hamiltonians performed, so a near 10x speed-up is possible, as shown in figure 5.8.



(a) Sliced at $v = 0$ and $\omega = 0$.



(b) Sliced at $v \ll 0$ and $\omega \gg 0$.

Figure 5.7. Converged Value Function Comparison for Warmstarted Quadcopter Vertical Reachability Analysis.

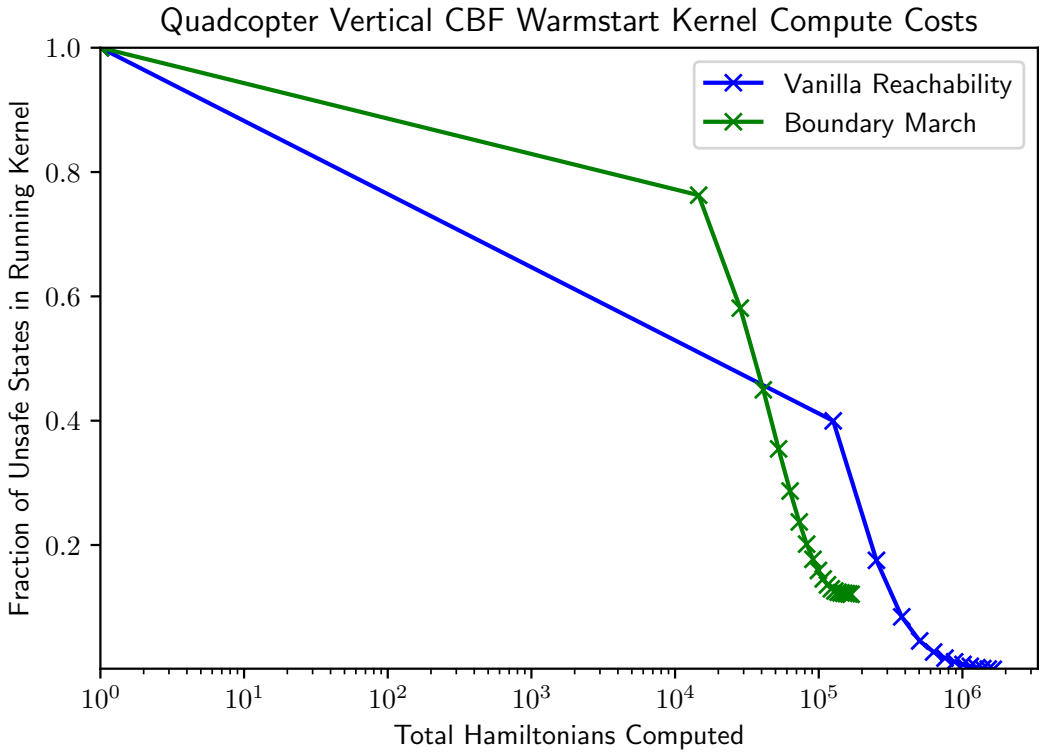


Figure 5.8. Hamiltonians required for convergence for Boundary March and global reachability. Computed on a 75x41x75x41 uniform grid.

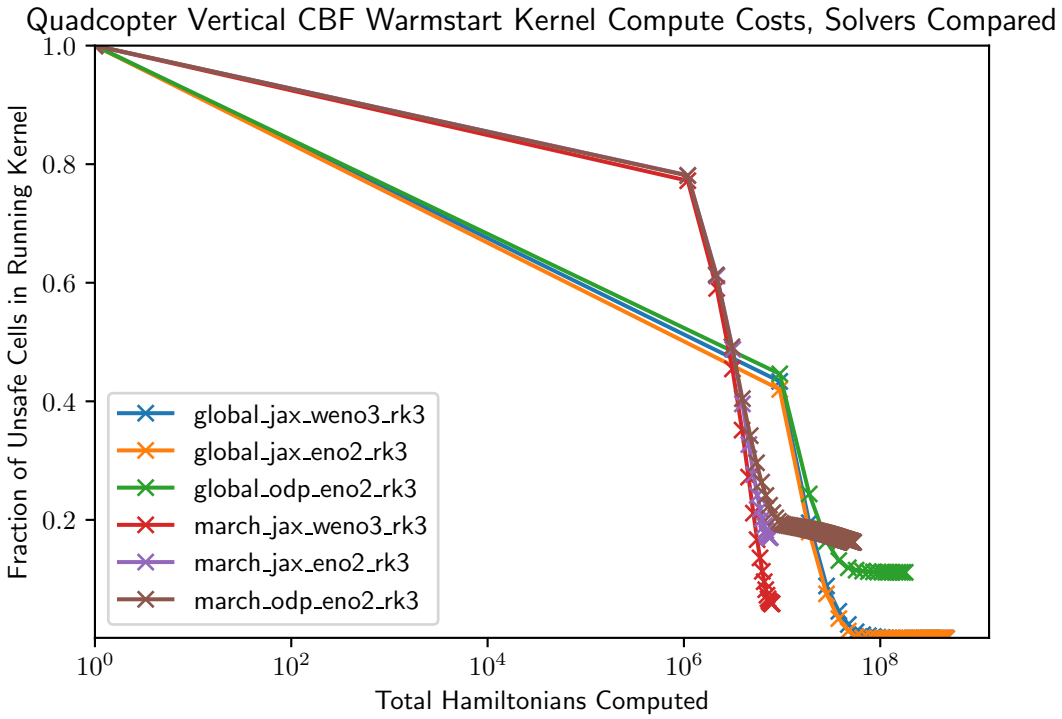


Figure 5.9. Comparison of solver performance between implemented schemes. Shows potential for improved accuracy with additional backend work.

5.3 Comparison Between Solvers

For the vertical quadcopter case (section 5.2), we found that the Boundary March result was not as accurate as with active cruise control. However, we remain confident in the algorithm in principle. Rather, we suspect that low backend solver accuracy raises numerical issues that prevent high performance on larger or more complicated dynamic systems. As a case study, we implemented the algorithm in the *hj_reachability* package (*github.com/toofanian/hj_reachability*) in addition to *optimized_dp*. This provided access to higher accuracy solvers, and yielded better performance, as shown in figure 5.9. However, the heavily parallelized implementation in *hj_reachability* prevented us from achieving actual compute time improvements. Ultimately, all discrete space solutions to ODEs are only as accurate as the solver they are based upon, so we are not troubled by the results, and we look forward to the continued development of HJ reachability solvers.

Chapter 6

Conclusion

HJ Reachability is a robust and powerful tool for deriving guaranteed safe control policies. Its results are most desirable, but come at the cost of exponentially scaling compute. The HJ Boundary March (Algorithm 1) builds on a tradition of research to make HJ reachability more efficient. It provides significant compute speedup, up to 2 orders of magnitude, by retaining the mechanics of warmstarted HJ reachability, but skimming to apply them only over a targeted subset of space. What remains is the critical, safety-enforcing result, without the claim of optimality.

The author anticipates opportunity for future work along these lines, both in the application of the Boundary March and in its improvement. Namely, with the compute gains shown, it should now be possible to compute a guaranteed safe control barrier function on up to 2 dimensions greater than the previous state of the art. Furthermore, the application of this algorithm as an efficient value-topology scan may even allow the Boundary March to directly participate in efficiently computing the standard optimal HJ reachability result. Regarding the algorithm itself, there is opportunity for implementation on higher accuracy solvers. Lastly, and most abstractly, there may be potential for a queue-based active set Q implementation, rather than the batch-based Q used here, which with some guiding heuristic may further optimize the contraction of the unsafe frontier.

Bibliography

[1] Somil Bansal, Mo Chen, Sylvia Herbert, and Claire J. Tomlin. Hamilton-jacobi reachability: A brief overview and recent advances, 2017.

[2] Aaron D. Ames, Samuel Coogan, Magnus Egerstedt, Gennaro Notomista, Koushil Sreenath, and Paulo Tabuada. Control barrier functions: Theory and applications, 2019.

[3] Charles Dawson, Sicun Gao, and Chuchu Fan. Safe control with learned certificates: A survey of neural lyapunov, barrier, and contraction methods for robotics and control. *IEEE Transactions on Robotics*, pages 1–19, 2023.

[4] Sylvia L. Herbert, Shromona Ghosh, Somil Bansal, and Claire J. Tomlin. Reachability-based safety guarantees using efficient initializations, 2019.

[5] Sander Tonkens and Sylvia Herbert. Refining control barrier functions through hamilton-jacobi reachability, 2022.

[6] 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, 2019.

[7] Mo Chen, Sylvia Herbert, and Claire J. Tomlin. Exact and efficient hamilton-jacobi-based guaranteed safety analysis via system decomposition, 2016.

[8] Jérôme Darbon and Stanley Osher. Algorithms for overcoming the curse of dimensionality for certain hamilton-jacobi equations arising in control theory and elsewhere, 2016.

[9] Jason J. Choi, Donggun Lee, Koushil Sreenath, Claire J. Tomlin, and Sylvia L. Herbert. Robust control barrier-value functions for safety-critical control, 2021.

[10] Somil Bansal and Claire Tomlin. Deepreach: A deep learning approach to high-dimensional reachability, 2020.

[11] Tsui-Wei Weng, Huan Zhang, Pin-Yu Chen, Jinfeng Yi, Dong Su, Yupeng Gao, Cho-Jui Hsieh, and Luca Daniel. Evaluating the robustness of neural networks: An extreme value theory approach, 2018.

[12] Zhizhen Qin, Tsui-Wei Weng, and Sicun Gao. Quantifying safety of learning-based self-driving control using almost-barrier functions. 2022.

[13] S. Herbert. *Safe Real-World Autonomy in Uncertain and Unstructured Environments*. PhD thesis, 2020.

[14] I.M. Mitchell, A.M. Bayen, and C.J. Tomlin. A time-dependent hamilton-jacobi formulation of reachable sets for continuous dynamic games. *IEEE Transactions on Automatic Control*, 50(7):947–957, 2005.

[15] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. 2018.

[16] Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. Efficient neural network robustness certification with general activation functions. *Advances in Neural Information Processing Systems*, 31:4939–4948, 2018.

[17] Kaidi Xu, Zhouxing Shi, Huan Zhang, Yihan Wang, Kai-Wei Chang, Minlie Huang, Bhavya Kailkhura, Xue Lin, and Cho-Jui Hsieh. Automatic perturbation analysis for scalable certified robustness and beyond. *Advances in Neural Information Processing Systems*, 33, 2020.

[18] Hadi Salman, Greg Yang, Huan Zhang, Cho-Jui Hsieh, and Pengchuan Zhang. A convex relaxation barrier to tight robustness verification of neural networks. *Advances in Neural Information Processing Systems*, 32:9835–9846, 2019.

[19] Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and J Zico Kolter. Beta-CROWN: Efficient bound propagation with per-neuron split constraints for complete and incomplete neural network verification. *Advances in Neural Information Processing Systems*, 34, 2021.

[20] Minh Bui, George Giovanis, Mo Chen, and Arrvindh Shriraman. Optimizeddp: An efficient, user-friendly library for optimal control and dynamic programming, 2022.