Refining Almost-Safe Value Functions on the Fly

Anonymous authors

***Abstract*—Control Barrier Functions (CBFs) are a powerful tool for ensuring robotic safety, but designing or learning valid CBFs for complex systems is a significant challenge. While Hamilton-Jacobi Reachability provides a formal method for synthesizing safe value functions, it scales poorly and is typically performed offline, limiting its applicability in dynamic environments. This paper bridges the gap between offline synthesis and online adaptation. We introduce REFINECBF for refining an approximate CBF—whether analytically derived, learned, or even unsafe—via warm-started HJ reachability. We then present its computationally efficient successor, HJ-PATCH, which accelerates this process through localized updates. Both methods guarantee the recovery of a safe value function and can ensure monotonic safety improvements during adaptation. Our experiments validate our framework's primary contribution: in-the-loop, real-time adaptation, in simulation (with detailed value function analysis) and on physical hardware. Our experiments on ground vehicles and quadcopters show that our framework can successfully adapt to sudden environmental changes, such as new obstacles and unmodeled wind disturbances, providing a practical path toward deploying formally guaranteed safety in real-world settings.**

I. INTRODUCTION

The widespread adoption of learning-based modules for perception and control has unlocked new capabilities in robotics, from dynamic locomotion [1] to complex navigation [2]. However, their effectiveness is often tightly coupled to their training conditions, limiting their ability to adapt to uncertain or evolving scenarios. This lack of adaptability presents a significant barrier to deployment in safety-critical applications, where environmental changes or distribution shifts can lead to catastrophic failures. This necessitates safety modules capable of adapting to changing conditions online in real-time, directly at the control level.

A promising approach for realizing such a module is the safety filter: a component that monitors the actions from a decision-making module and intervenes only when necessary to prevent failure [3], [4]. Many of these filters are built upon a safety value function, a scalar field over the state space designed to encode safety information. Control Barrier Functions (CBF) [5] are a prominent example. Typically, the function's value at a given state indicates the system's safety margin, while its gradient delineates the set of control actions that maintain or improve safety. The filter leverages the function's value and gradient to constrain the nominal control input to the set of safe control actions at the current state. However, the central challenge is designing a valid safe value function, i.e., one that guarantees persistent trajectory safety from purely pointwise enforcement.

This challenge has been approached from two main directions, each with significant drawbacks:

1) **Constructive methods**, such as Hamilton-Jacobi (HJ) Reachability [6], offer formal guarantees by numerically solving for the value function. However, they scale poorly with state dimensionality.
2) **Approximate methods**, which leverage function approximators like neural networks, offer scalability but sacrifice formal guarantees and do not generalize easily.

Crucially, both paradigms struggle to adapt to real-world variations encountered post-deployment, such as sim-to-real gaps or unexpected changes in system dynamics.

In this work, we argue that the constructive and approximate methods for generating safe value functions are not mutually exclusive but are, in fact, complementary. We propose to bridge the gap between formal guarantees and scalability by using a data-driven approximation as a warm start for a formal, constructive refinement process that leverages HJ Reachability.

A. Related work

Safety is a cornerstone of robotics research, with methods ranging from optimal control and model predictive control (MPC) [7] to formal verification [8]. While optimal control can formally incorporate safety constraints, solving these problems for complex, nonlinear systems is often computationally intractable, sacrificing guarantees for real-time performance. In contrast, optimal control-inspired approaches, such as constrained Reinforcement Learning, penalize constraint violations or provide guarantees solely in expectation, limiting their use in safety-critical settings. This has led to the rise of safety filters, standalone modules that minimally modify a primary policy's control output to ensure safety [4].

A popular approach to designing these filters is to use a safety value function—a scalar field over the state space that encodes safety information. Current research on these value functions largely falls into two complementary categories.

a) Control Barrier Functions (CBFs): CBFs provide a principled way to enforce safety online. By constraining the rate at which a system can approach a failure boundary, a CBF can be used to formulate a simple, pointwise optimization problem that minimally modifies a nominal control input to guarantee safety [9]. This enforcement mechanism is highly efficient to solve for control-affine systems, for which it reduces to a quadratic program.

However, the primary challenge for CBFs is characterizing safety—that is, finding a valid CBF in the first place. While they can be derived analytically for some systems, this is difficult for complex nonlinear dynamics, especially under external disturbances. To address this, learning-based approaches have been proposed to approximate CBFs from data [10], [11]. These methods, while scalable, often lack formal guarantees and their reliability depends heavily on the quality and density of the training data. Approaches like backup CBFs [12] provide stronger guarantees but introduce significant compu-

tational complexity. Additionally, their overall performance hinges on the expert design of a fixed "backup" policy.

b) Hamilton-Jacobi Reachability (HJR): In contrast to CBFs, HJ Reachability provides a formal method to characterize safety. It formulates a worst-case optimal control problem to compute the set of all states from which failure is inevitable, yielding a value function with strong safety guarantees for general nonlinear systems with disturbances [13].

However, this approach faces two significant hurdles. First, its traditional solution involves numerically solving a Hamilton-Jacobi partial differential equation (PDE) on a state-space grid. This method suffers from the curse of dimensionality, as its computational complexity grows exponentially with the number of state variables, typically limiting them to systems with fewer than six state dimensions [6]. Numerous works have sought to improve the scalability of the original formulation directly by leveraging problem structure (e.g., via warm-starting [14], decomposition [15]) or using learning-based approximations to eliminate state-space gridding (e.g., DeepReach [16], and reinforcement learning [17]). However, they often do so at the cost of approximation errors, extensive pre-computation, or overly conservative behavior. Second, and more critical for online use, HJR is typically used as a least-restrictive safety filter [18], where the optimal control is enforced only when near the safety boundary, often leading to either conservative or chattering control behavior.

In contrast, another family of methods sidesteps the PDE entirely by trading generality for computational tractability. These HJ-inspired approaches solve a related but more constrained problem, such as using Sum-of-Squares (SoS) optimization for systems with polynomial dynamics [19], [20] or using simpler geometric representations like zonotopes [21] which are efficient to propagate but may be overly conservative for general nonlinear systems. In contrast to the direct HJ approaches, these methods are often not deployed online with a safety filter, but e.g., generate safe "funnels" around trajectories [22] or leverage pre-computed sets for collision checking [23], [24].

c) Bridging the Gap: Summarizing, CBF-based methods offer efficient enforcement but rely on a safe value function, while HJR-based methods provide formal characterization but typically lack efficient enforcement. For example, [25] leverages HJ-based supervised rollouts for learning a CBF [25]. Choi et al. [26] first established a direct link between CBFs and HJR using Control Barrier Value Functions (CBVFs). They bridge this gap by using HJR-like computations (with discounting) to construct a valid CBVF; unfortunately, this approach inherits the high computational cost of HJR and does not leverage approximate solutions.

Alternatively, REFINECBF [27] and its extension HJ-PATCH [28], leveraged approximate initial value functions (from e.g., a learned CBF) to warmstart a HJR-based formal refinement process. Summarized, these methods leverage an initial approximation (and an adaptive updating scheme) for faster convergence. However, they construct the value function offline for a given environment and are limited to numerical simulations. This work instead focuses on adapting value functions online to adapt to real-world changes to the environment or the system, while retaining the safety and convergence guarantees of [27] and [28].

B. Contributions

This work presents a formally-grounded adaptive safety framework that enables online, in-the-loop adaptation to sensed changes in the system and environment. Our algorithmic extensions to REFINECBF and HJ-PATCH rely on iteratively refining any given value function until convergence while adapting to changes in the system and the environment. Specifically, our contributions are:

- A theoretical extension of the control-only REFINECBF [27] method to formally guarantee safety for systems subject to unknown but bounded external disturbances.
- Unified algorithmic framework spanning offline and in-the-loop refinement, including theoretical analysis and implications for online deployment.
- A rigorous comparative analysis against typical CBF approaches (disjoint CBFs, backup CBFs), highlighting the benefits of principally refining online in realistic simulated experiments with real-time requirements.
- Extensive, novel experiments on hardware (mobile robots and quadcopter) that demonstrate real-time adaptation to online detection of unforeseen obstacles and wind disturbances.

C. Organization

The remainder of this paper is organized as follows. Section II reviews the necessary background on value function-based safety. We then introduce our core algorithms and their theoretical guarantees: REFINECBF in Section III and its computationally efficient counterpart, HJ-PATCH, in Section IV IV. We validate our framework through extensive experiments, presenting detailed simulations in Section V, and hardware demonstrations on multiple robotic platforms in Section VI. Summarizing, Section VII provides use-cases and future work, followed by a Conclusion in Section VIII.

II. PRELIMINARIES

We consider a control- and disturbance affine dynamical system

$$\dot{x} = F(x, u, d) \triangleq f(x) + g(x)u + w(x)d, \tag{1}$$

which describes a wide range of robots, with state $x \in \mathcal{X} \subset \mathbb{R}^n$, input $u \in \mathcal{U} \subset \mathbb{R}^m$, and disturbance $d \in \mathcal{D} \subset \mathbb{R}^o$, where $\mathcal{U}$ and $\mathcal{D}$ are polytopes. We assume the dynamics $F : \mathbb{R}^n \times \mathcal{U} \times \mathcal{D} \mapsto \mathbb{R}^n$ are uniformly continuous, bounded, and Lipschitz continuous in x. The Lipschitz continuity of the dynamics ensures the existence and uniqueness of system trajectories $\mathbf{x}_{x,t}^{\mathbf{u},\mathbf{d}}(s)$ starting from state x at time t under a control signal $\mathbf{u}$ and disturbance signal $\mathbf{d}$. We denote the set of measurable functions $\mathbf{u} : \mathbb{R}_{\geq t} \to \mathcal{U}$ and $\mathbf{d} : \mathbb{R}_{\geq t} \to \mathcal{D}$ as $\mathbb{U}$ and $\mathbb{D}$, representing the allowed control and disturbance signals.

Let $\mathcal{L} \subseteq \mathcal{X}$ represent the constraint set, i.e. the set of non-failure states of the system (1). An element $x \in \mathcal{X}$ is considered instantaneously *safe* if $x \in \mathcal{L}$. Its complement,

$\mathcal{L}^C$, denotes the set of failure states. The goal of a safety filter is to keep the system within the constraint set $\mathcal{L}$. In this work, the disturbance $\mathbf{d}$ is determined in reaction to the control signal in the form of a disturbance strategy $\mathfrak{d} : \mathbb{U} \mapsto \mathbb{D}$. We restrict $\mathfrak{d}$ to draw from a *nonanticipative strategy*, which prohibits using future knowledge of the control signal to preserve causality [29], with Ξ the set of all such strategies.

A foundational concept for guaranteeing safety is control invariance.

Definition 1 (Robust Control Invariant Set)**.** *A set $\mathcal{C} \subseteq \mathcal{X}$ is a robust control invariant set if for any state $x \in \mathcal{C}$ and initial time $t \in \mathbb{R}$, there exists a control signal $\mathbf{u} \in \mathbb{U}$ such that for all disturbance strategies $\mathfrak{d} \in \Xi$, the system trajectory remains in the set, i.e., $\mathbf{x}_{x,t}^{\mathbf{u},\mathfrak{d}}(s) \in \mathcal{C}$ for all $s \geq t$.*

By this definition, the empty set $\emptyset$ is also considered control invariant, as the required condition holds vacuously. For any given constraint set $\mathcal{L}$, we are often interested in finding the largest possible safe region within it. This is known as the viability kernel.

Definition 2 (Viability Kernel, [30])**.** *For any set $\mathcal{K}$, let the viability kernel $\mathcal{S}(\mathcal{K})$ be such that $\mathcal{S}(\mathcal{K}) \subseteq \mathcal{K}$ and if there exists $\mathcal{C} \subseteq \mathcal{K}$ such that $\mathcal{C}$ is control invariant, then $\mathcal{C} \subseteq \mathcal{S}(\mathcal{K})$, hence $\mathcal{S}(\mathcal{K})$ is the largest control invariant set in $\mathcal{K}$.*

A. Control Barrier Functions

Let a value function $h : \mathcal{X} \mapsto \mathbb{R}$ be Lipschitz continuous and let $\mathcal{H} := \{x \mid h(x) \geq 0\} \subseteq \mathcal{X}$ be the 0-superlevel set of h. We define the Lie derivative $L_F h(x,u,d) := \langle \frac{\partial h}{\partial x}, F(x,u,d) \rangle$ and the Hamiltonian $L_F^* h(x) := \inf_{d \in \mathcal{D}} \sup_{u \in \mathcal{U}} L_F h(x)$. For non-continuously differentiable functions, the Lie derivative can be formulated through the Clarke generalized gradient [31].

Nagumo's theorem [32] provides a condition for control invariance: assume $\frac{\partial h}{\partial x} \neq 0$ for all $x \in \partial\mathcal{H} := \{x \mid h(x) = 0\}$, then $\mathcal{H}$ is control invariant if and only if

$$L_F^* h(x) \geq 0 \text{ for all } x \in \partial\mathcal{H}. \tag{2}$$

In this paper, a safe value function is considered to be any function h that satisfies Nagumo's theorem, and whose 0-superlevel set $\mathcal{H}$ is a subset of $\mathcal{L}$, $\mathcal{H} \subseteq \mathcal{L}$. The CBF condition extends Nagumo's theorem from the boundary to the interior of the set $\mathcal{H}$ with a Lyapunov-like condition, ensuring the state does not approach the boundary too quickly.

Definition 3 (Control Barrier Function [9])**.** *Let $\mathcal{H}$ denote the 0-superlevel set of a continuously differentiable value function $h : \mathcal{X} \mapsto \mathbb{R}$, then $h(x)$ is a control barrier function for* (1) *if there exists an extended class $\mathcal{K}$ function α and a set $\mathcal{C}$ with $\mathcal{H} \subseteq \mathcal{C} \subset \mathbb{R}^n$ such that*

$$L_F^* h(x) \geq -\alpha(h(x)) \tag{3}$$

for all $x \in \mathcal{C}$.

A CBF defines a control invariant set $\mathcal{H}$ [33]. For completeness, we define the state-dependent allowable control range for a CBF as follows:

$$\mathcal{G}_h(x) := \left\{ u \in \mathcal{U} \;\middle|\; \min_{d \in \mathcal{D}} L_F h(x,u,d) + \alpha(h(x)) \geq 0 \right\} \tag{4}$$

Any choice of control law for which $u \in \mathcal{G}$ will ensure that the system (1) remains in $\mathcal{H}$ indefinitely.

The popularity of CBFs can be attributed to the ease with which they can be used in a safety filter online through an optimization problem. At each time step, a nominal (safety-agnostic) policy $\hat{\pi}(x)$ is passed through a filter that solves:

$$\begin{aligned} u^*(x) = \; \arg\min_{u} \quad & \|u - \hat{\pi}(x)\|_2^2 \\ \text{subject to} \quad & L_g h(x) u \geq -\gamma h(x) - L_f h(x) \\ & \qquad - \min_{d \in \mathcal{D}} L_w h(x) d \\ & u \in \mathcal{U}. \end{aligned} \tag{5}$$

In this paper we assume $\alpha(x) = \gamma x$, for $\gamma > 0$. As the disturbance and control are independent for (1), we can first solve for the disturbance which decreases the value function most through an exhaustive search of its vertices, with the resulting optimization problem to solve for u^* being a quadratic program (QP), which can be solved in real time (> 100 Hz) for typical robotics applications.

The main challenge of this approach, however, lies in finding a valid function h for which the QP (5) is feasible for all states x in its 0-superlevel set. To precisely discuss the quality of candidate CBFs often used in practice, we classify CBFs into 4 possible categories:

TABLE I: The 4 possible categories of CBFs and their properties.

	Characterizes safety: $\mathcal{H} \cap \mathcal{L} = \emptyset$	Characterizes control invariance: (5) $\forall x$	Non-intrusive safety filter
Invalid CBF	✓	✗	✓/✗
Unsafe CBF	✗	✓	✓
Conservative (safe) CBF	✓	✓	✗
Desired (safe) CBF	✓	✓	✓

This work proposes methods to refine a candidate CBF into a desired, provably safe one. The resulting functions, which are constructed using tools from HJ reachability theory, are technically Control Barrier Value Functions (CBVFs), which we detail in the following section.

B. HJR-based Control Barrier Value Functions

Given the aforementioned constraint set $\mathcal{L}$, we formulate an associated constraint function, $\ell(x)$, such that $\ell(x) \geq 0$ if and only if $x \in \mathcal{L}$, e.g., a (weighted) signed-distance function. For a desired decay rate λ, see e.g. (5), we define the associated CBVF [26]:

$$h_\lambda(x,t) = \min_{\mathfrak{d} \in \Xi} \max_{\mathbf{u} \in \mathbb{U}} \min_{s \in [t,0]} e^{\lambda(s-t)} \ell\left(\mathbf{x}_{x,t}^{\mathbf{u},\mathfrak{d}}(s)\right). \tag{6}$$

We note that h_λ is uniquely defined for each fixed value of $\lambda \geq 0$, and for $\lambda = 0$ its definition matches the definition of the standard HJ reachability value function formulation [34].

It is important to note that any CBF is a CBVF, while the inverse does not hold. We introduce the following remark:

Remark 1 (Differentiating a CBVF from a CBF)**.** *In addition to satisfying Eq.* (3) *for all states, a CBF $h(x)$ is, by definition (Def. 3), also continuously differentiable, i.e. $h(x) \in C^1$. In contrast, a CBVF is differentiable almost everywhere, see [26].*

With a value function defined, the system can maintain safety during deployment by incorporating the value function into a safety filter. Similarly to CBFs, we can define the admissible control set that retains (finite time) safety, where we distinguish between the offline decay rate λ and online decay rate γ.

$$\mathcal{G}^{\gamma}_{h_\lambda}(x,s) = \left\{ u \in \mathcal{U} \mid \min_{d\in\mathcal{D}} \dot{h}_\lambda(x,s) + \gamma h_\lambda(x,s) \geq 0 \right\} \quad (7)$$

Importantly, the discount factor used to compute the CBVF offline λ does not require matching the CBVF decay rate online γ; Specifically:

Proposition 1 (Forward completeness and safety with larger online discount rate)**.** *Applying a discount factor γ online* (5) *for a CBVF that is constructed* (6) *with $\lambda \leq \gamma$ maintains control invariance of the safe set.*

Proof. By inspection of Eq (7), if $\gamma \geq \lambda$, we have that $\mathcal{G}^{\gamma}_{h_\lambda}(x,s) \supseteq \mathcal{G}^{\lambda}_{h_\lambda}(x,s) \neq \emptyset$ for all $x \in \mathcal{H}$ and $s \in [t,0]$. Hence, applying a larger discount rate γ online maintains pointwise feasibility. In addition, at the boundary of the safe set, $x \in \partial\mathcal{H}$, we have that $h_\lambda = 0$ for all $\lambda \geq 0$ (see [26]), hence $\mathcal{G}^{\gamma}_{h_\lambda}(x,s) = \mathcal{G}^{\lambda}_{h_\lambda}(x,s)$ by inspection of (7). Combined, this ensures the same safety guarantees are maintained when applying a larger discount rate γ online. □

This enables setting $\lambda = 0$ during construction and applying any (positive) decay rate γ online. This is particularly useful as applying a non-zero discount factor offline is limited to finite-time safety problems [26], whereas a discount factor $\lambda = 0$ holds for infinite-time safety, which this work focuses on. Additionally, considering $\lambda = 0$ allows us to freely choose any $\gamma > 0$ for safety enforcement online while providing infinite-time safety guarantees. The associated admissible control set for a converged h, i.e. $D_t h = 0$, is as follows:

$$\mathcal{G}^{\gamma}(x) = \left\{ u \in \mathcal{U} \mid \min_{d\in\mathcal{D}} L_F h(x) + \gamma h(x) \geq 0 \right\}. \quad (8)$$

C. Solutions to HJR value functions

To solve for (6), we present the associated continuous solution and its derived time discretized dynamic programming-based solution. For readability, we will drop the exponential term λ, i.e. $\lambda = 0$, and refer to the equivalent formulas in [26] for the formulation including λ. The value function for the $\gamma = 0$ case is defined as in [34]:

$$h(x,t) = \min_{\mathfrak{d}\in\Xi} \max_{\mathbf{u}\in\mathbb{U}} \min_{s\in[t,0]} \ell\left(\mathbf{x}^{\mathbf{u},\mathfrak{d}}_{x,t}(s)\right). \quad (9)$$

We distinguish two popular formulations of HJ reachability to solve the safety scenario described in this paper. The first formulation, a single-boundary partial differential equation (PDE), forces contraction of the value function, and recovers the largest control invariant subset of the initial value function's 0-level set. The second, a variational inequality (VI), recovers a control invariant set which is a subset of $\mathcal{L}$. In the classic HJR setting in which the terminal boundary condition is the constraint function, $h_b(x) = \ell(x)$, these two formulations are equivalent.

1) Single-Boundary PDE Formulation [35]: The value function $h(x,t)$ is the viscosity solution to the following Hamilton-Jacobi-Isaacs PDE (HJI-PDE):

$$D_t h(x,t) + \min\{0, \max_{u\in\mathcal{U}} \min_{d\in\mathcal{D}} D_x h(x,u,d,\, t)\cdot F(x,u,d)\} = 0$$
$$h(x,0) = h_b(x). \quad (10)$$

The discrete time equivalent of (10) uses the following dynamic programming update rule:

$$h^{(k+1)}(x) = h^{(k)}(x) + \Delta^{(k)} \min\{0, L^*_F h^{(k)}(x)\}, \quad (11)$$

with $\Delta^{(k)}$ denoting the time-step which is dynamically updated based on the magnitude of the Hamiltonians $L^*_F h^{(k)}(x)$, see [36] and (14) for details. Eqs. (10) and (11) define contractive mappings over time through the min-with-zero operation.

2) Variational Inequality Formulation [34]: The value function $h(x,t)$ is the viscosity solution to the following HJI-VI:

$$0 = \min\{\ell(x) - h(x,t),$$
$$D_t h(x,t) + \max_{u\in\mathcal{U}} \min_{d\in\mathcal{D}} D_x h(x,u,d,\, t)\cdot F(x,u,d)\}$$
$$h(x,0) = h_b(x). \quad (12)$$

The discrete time equivalent of (12) uses the following dynamic programming update rule:

$$h^{(k+1)}(x) = \min\left\{\ell(x), h^{(k)}(x) + \Delta^{(k)} L^*_F h^{(k)}(x)\right\}. \quad (13)$$

In practice, we solve (11) and (13) through spatial discretization on a pre-defined set of grid points as no closed-form solution exists. The Hamiltonian $L^*_F h^{(k)}(x)$ is typically approximated using finite difference methods, namely (weighted) essentially non-oscillatory ((W)ENO) schemes [37] as follows:

$$L_F h(x,u,d) := \langle \nabla h(x), F(x,u,d)\rangle \approx \langle \delta^p_\Delta(h), F(x,u,d)\rangle, \quad (14)$$

with δ^p_Δ a finite difference operator of order p, i.e. the operator includes p neighbors in every dimension. In addition, the Hamiltonian $L^*_F h(x)$ is computed through an exhaustive search of the vertices of $\mathcal{U}$ and $\mathcal{D}$.

Remark 2 (Convergence of CBVF)**.** *A stationary solution to* (10) *and* (12)*, or equivalently, if* (11) *and* (13) *converge ($k \to \infty$), the result characterizes an infinite-time CBVF, which encodes safety for an infinite duration. We drop the time-dependence for such a value function $h(x)$.*

III. Refining CBFs

This section details our primary contribution: a framework for refining a candidate value function h^0, using the HJI-VI formulation (12). This approach leverages an initial guess—such as a learned neural network or a hand-designed function—to "warm-start" the computation, enabling faster convergence and online adaptation. We first establish the theoretical guarantees for our method in a static environment. We then present two algorithms: RefineCBF, a direct implementation of the refinement process, and SafeAdaptRefineCBF, a variant with stronger intermediate safety guarantees, at the cost of added conservativeness. Finally, we discuss the practical implications of using these algorithms for in-the-loop adaptation in dynamic environments.

A. Guarantees underlying refining CBFs

We begin by establishing the theoretical guarantees of our approach in a static environment. In this section, we first prove our main result: that the refinement process, when warm-started with a candidate function h^0 is guaranteed to converge to a valid and safe CBVF (Theorem 1). We then analyze the properties of the value function during convergence, establishing conditions under which safety is preserved throughout the refinement process (Lemma 4).

We define the boundary function of the HJI-VI (12) as the candidate value function, $h_b = h^0$, shifting the objective from a standard safety evaluation in (9) to include a recursive refinement of the candidate value function. This defines the HJI-VI whose unique viscosity solution is the following trajectory-based value function:

$$h(x,t) = \min_{\mathfrak{d}\in\Xi}\max_{\mathbf{u}\in\mathbb{U}}\min\left\{\min_{s\in[t,0)}\ell\left(\mathbf{x}_{x,t}^{\mathbf{u},\mathfrak{d}}(s)\right), h^0\left(\mathbf{x}_{x,t}^{\mathbf{u},\mathfrak{d}}(0)\right)\right\}. \tag{15}$$

To facilitate the proofs, we assume the following:

Assumption 1 (The candidate CBF is pointwise conservative with respect to the constraint function)**.** *For all x, we assume the candidate CBF $h^0(x)$ satisfies $h^0(x) \leq \ell(x)$.*

As outlined in [27], Assumption 1 is not strictly required, but facilitates the proofs below. It can be trivially satisfied by defining a modified candidate CBF $h^0(x) = \min\{h^0(x), \ell(x)\}$.

Theorem 1 below is the key theoretical underpinning of REFINECBF.

Theorem 1 (Convergence to a safe CBVF)**.** *If* (15) *converges to $h^*(x)$ as $t \to -\infty$, then $h^*(x)$ is a safe CBVF for all $x \in \mathcal{H}^*$, its 0-superlevel set.*

The proof of Theorem 1 relies on the following three foundational lemmas, which establish that the converged value function's 0-superlevel set is control invariant (Lemma 1), a subset of the viability kernel (Lemma 2), and that the function itself satisfies the CBF inequality (Lemma 3).

Lemma 1 (Convergence to a control invariant set)**.** *If* (15) *converges to $h^*(x)$ as $t \to -\infty$, i.e. $h^*(x)$ characterizes a stationary solution of* (12)*, then $\mathcal{H}^*$ is control invariant, i.e. its Hamiltonian $L_F^* h^*(x) \geq 0$ for all $x \in \partial\mathcal{H}^*$.*

Proof. The stationary version of the HJI-VI, is as follows, see [38]:

$$\min\left\{\ell(x) - h(x), \max_{u\in\mathcal{U}}\min_{d\in\mathcal{D}} D_x h(x,u,d)\cdot F(x,u,d)\right\} = 0. \tag{16}$$

Upon inspection of the second term of the minimum operator, any $h^*(x)$ satisfying (16) satisfies $L_F^* h^*(x) \geq 0$ for all x. Particularly, this holds for all $x \in \partial\mathcal{H}^*$, thus $\mathcal{H}^*$ is a control invariant set by Nagumo's theorem, (2), see [32]. □

Lemma 2 (Convergence to a subset of the viability kernel)**.** *If* (15) *converges to $h^*(x)$ as $t \to -\infty$, its associated 0-superlevel set $\mathcal{H}^*$ is a subset of the viability kernel of the constraint set $\mathcal{S}(\mathcal{L})$, i.e. $\mathcal{H}^* \subseteq \mathcal{S}(\mathcal{L})$.*

Proof. This proof builds on the proof of Theorem 1 in [14]. We (a) leverage that standard HJ Reachability recovers a value function whose 0-superlevel set corresponds to the viability kernel [39], and then (b) prove that for any time $t \in (-\infty, 0]$ the warmstarted value function is pointwise upper bounded by standard HJ reachability's value function. For the purposes of this proof we define the CBVF obtained when refining a CBF h^0, i.e. through (15), as $h(x,t;h^0)$. In contrast, we define the CBVF obtained using standard HJ reachability, i.e. through (9), as $h(x,t;\ell)$. Then, we aim to show that $h(x,t;h^0) \leq h(x,t;\ell)$ for all x,t. We observe:

$$\begin{aligned}
h(x,t;h^0) &= \min_{\mathfrak{d}\in\Xi}\max_{\mathbf{u}\in\mathbb{U}}\min\left\{\min_{s\in[t,0)}\ell\left(\mathbf{x}(s)\right), h^0\left(\mathbf{x}(0)\right)\right\} \\
&\leq \min_{\mathfrak{d}\in\Xi_{[t,0]}}\max_{\mathbf{u}\in\mathcal{U}_{[t,0]}}\min\left\{\min_{s\in[t,0)}\ell\left(\mathbf{x}(s)\right), \ell\left(\mathbf{x}(0)\right)\right\} \\
&= \min_{\mathfrak{d}\in\Xi_{[t,0]}}\max_{\mathbf{u}\in\mathcal{U}_{[t,0]}}\min_{s\in[t,0]}\ell\left(\mathbf{x}(s)\right) \\
&= h(x,t;\ell),
\end{aligned}$$

where we drop $\substack{\mathbf{u},\mathfrak{d}\\x,t}$ from the trajectory $\mathbf{x}(s)$ for readability. The inequality follows directly from Assumption 1, i.e. $h^0(x) \leq \ell(x)$ for all x. Therefore, as $t \to -\infty$, we have $h^*(x;h^0) \leq h^*(x;\ell)$. Combining (a) and (b) we have $\mathcal{H}^* = \{x \mid h^*(x;h^0) \geq 0\} \subseteq \{x \mid h^*(x;\ell) \geq 0\} = \mathcal{S}(\mathcal{L})$, with the last equality by Definition 2, concluding the proof. □

Lemma 3 (Convergence to a CBVF on a subset of the state space)**.** *If* (15) *converges to $h^*(x)$ as $t \to -\infty$, i.e. $h^*(x)$ characterizes a stationary solution of* (12)*, then $h^*(x)$ is a valid CBVF for all $x \in \mathcal{H}^*$.*

Proof. Similarly to Lemma 1, we begin by inspecting the stationary version of (12), (16). The second term of the minimum operator implies that any $h^*(x)$ satisfying (16) satisfies $L_F^* h^*(x) \geq 0$ for all x.

The value function h^*, for any choice of $\gamma \geq 0$ in (3), with $\alpha(h^*(x)) = \gamma h^*(x)$ satisfies:

$$L_F^* h^*(x) \geq 0 \geq -\gamma h^*(x),$$

for all $x \in \mathcal{H}^*$, with the first inequality by satisfaction of (16) and the second inequality from $h^*(x) \geq 0$ for all $x \in \mathcal{H}^*$. □

We are now ready to prove Theorem 1:

Proof of Thm 1. Safety is guaranteed by $h^*(x)$'s 0-superlevel set, $\mathcal{H}^*$ being (a) control invariant, Lem. 1, and (b) being a subset of the viability kernel, Lem. 2. Combined with Lem. 3, we obtain a safe CBVF, concluding the proof. □

The convergence guarantees match those of standard HJR, and similarly do not provide guarantees on the rate of convergence to a safe CBVF, but progressively improve the CBVF's minimum finite-time safety as t increases (Remark 3). Additionally, the key concern in the online setting is not the final converged value, but the properties of the value function

during refinement. A crucial property is that once the value function becomes control invariant at any iteration, it remains so thereafter (Lemma 4).

Remark 3 (Duration of safety)**.** *The temporal input of the value function $h(x,t)$ implicitly provides the duration of safety. This follows directly from* (15)*, which implies that for any trajectory starting at x such that $h(x,t) \geq 0$ there exists a control signal $\mathbf{u} \in \mathbb{U}$ such that for any non-anticipative disturbance strategy $\mathfrak{d} \in \Xi$ the trajectory $\mathbf{x}$ remains in $\mathcal{L}$ for at least time t, i.e. $\ell(\mathbf{x}(s)) \geq 0$ for all $s \in [t,0]$. Any control signal satisfying the time-varying control set defined in* (7) *retains safety for at least time t, thus acting as a lower bound on the duration of safety.*

We further note that if the initial approximate CBF $h^0(x)$ is control invariant or if, by coincidence or by deliberate action, there is a t_1 such that $h(x,t_1)$ is control invariant, we provide a guarantee on preserving safety while converging:

Lemma 4 (Refining a safe CBVF will preserve safety)**.** *If there exists $t_1 \leq 0$ such that $h(x,t_1)$ from* (15) *is control invariant, $h(x,t)$ is control invariant for all $t < t_1 \leq 0$.*

Proof. For any $t \leq t_1 \leq 0$, we can write the following:

$$h(x,t) = \min_{\mathfrak{d}\in\Xi} \max_{\mathbf{u}\in\mathbb{U}} \min \left\{ \min_{s\in[t,0)} \ell\left(\mathbf{x}(s)\right), h^{(0)}\left(\mathbf{x}(0)\right) \right\}$$
$$= \min_{\mathfrak{d}\in\Xi} \max_{\mathbf{u}\in\mathbb{U}} \min \left\{ \min_{s\in[t,t_1)} \ell\left(\mathbf{x}(s)\right), h\left(\mathbf{x}(t_1), t_1\right) \right\}.$$

Then, any $x \in \mathcal{H}(t)$ satisfies (a) $\min_{s\in[t,t_1)} \ell\left(\mathbf{x}(s)\right) \geq 0$ and (b) $h\left(\mathbf{x}(t_1), t_1\right) \geq 0$. Hence, we can find a trajectory starting at x that (a) stays safe for at least $t - t_1$ time and (b) enters a control invariant set $\mathcal{H}(t_1)$ within $t - t_1$ time, in which it can stay indefinitely. By construction, $\mathcal{H}(t) \subseteq \mathcal{L}$ for all $t_1 \leq 0$, hence $\mathcal{H}^*$ also characterizes safety. □

We leverage Lemma 4 to present a conservative version of the REFINECBF algorithm, SAFEADAPT-REFINECBF , Alg. 3, which first forces contraction to a safe (yet not necessarily performant) CBVF, followed by a possible set expansion.

B. REFINECBF *algorithm overview*

Algorithm 1 presents our numerical implementation for refining a candidate CBF (the in-the-loop extension is highlighted in teal). The core of the algorithm is the iterative application of the HJI-VI update rule from 13 (line 4), initialized with h^0. This process can be run offline until convergence or, more importantly for our work, executed continuously in-the-loop. In the online setting, the algorithm observes real-time changes to the environment-new obstacles (through $\mathcal{L}$), modified actuation limits of the robot (through $\mathcal{U}$), or changed disturbances to the system (through $\mathcal{D}$) (lines 5-6)-into the refinement process. The most current value function is continuously published (line 7) for use by a safety filter. As is standard for HJR-based methods, our implementation solves these updates over a discretized state-space grid.

The accompanying safety filter is detailed in Alg. 2, with the in-the-loop extension in teal. For each nominal control, the filter queries the latest value function (and its gradient) to solve the QP (5). This QP finds the minimum-norm control input that satisfies the safety constraint, and this safe action is then applied to the robot (lines 5-7). As we only keep track of the most recent computed $h(x)$, we assume $D_t h = 0$, i.e. the safe control set is characterized by (8).

Algorithm 1 REFINECBF (In-the-loop)

Require: $h^0(x) : \mathcal{X} \mapsto \mathbb{R}$: Initial value function
$\ell(x) : \mathcal{X} \mapsto \mathbb{R}$: Current constraint function

1: $k \leftarrow 0$
2: $h^{(0)}(x) \leftarrow \min\{h^0(x), \ell(x)\}$
3: **while** $|h^{(k)}(x) - h^{(k-1)}(x)| > \epsilon$ (or Robot operating) **do**
4: $\quad h^{(k+1)}(x) \leftarrow \min\left\{\ell(x), h^{(k)}(x) + \Delta^{(k)} L_F^* h^{(k)}(x)\right\}$
$\quad$ *// Environment updates can occur at a different rate*
5: $\quad$ Observe environment
6: $\quad \mathcal{U}^{(k+1)}, \mathcal{D}^{(k+1)}, \ell^{(k+1)} \leftarrow$ Update $(\mathcal{U}^{(k)}, \mathcal{D}^{(k)}, \ell^{(k)})$
7: $\quad$ **publishCurrentCBF**$(h^{(k+1)})$
8: $\quad k \leftarrow k + 1$
9: **end while**
10: **return** $h^*(x) = h^{(k)}(x)$ $\quad \triangleleft$ *Converged value function*

Algorithm 2 REFINECBF safety filter (In-the-loop)

1: $j \leftarrow 0$
2: **while** Robot operating **do**
3: $\quad x^{(j)} \leftarrow$ **estimateState**()
4: $\quad \hat{u} \leftarrow$ **nominalController**()
5: $\quad h^{x^{(j)}}, \frac{\partial h}{\partial x}^{x^{(j)}} \leftarrow$ **getAndQueryCurrentCBF**$(x^{(j)})$
6: $\quad u^{(j)} \leftarrow$ **solve** (5)$(h[x^{(j)}], \frac{\partial h}{\partial x}[x^{(j)}], \mathcal{U}^{(j)}, \mathcal{D}^{(j)}, \hat{u})$
7: $\quad$ Apply $u^{(j)}$ to robot
$\quad$ *// Environment updates can occur at a different rate*
8: $\quad$ Observe environment
9: $\quad \mathcal{U}^{(j+1)}, \mathcal{D}^{(j+1)} \leftarrow$ Update $(\mathcal{U}^{(j)}, \mathcal{D}^{(j)})$
10: $\quad j \leftarrow j + 1$
11: **end while**

C. *Stronger guarantees at increased conservativeness*

While REFINECBF guarantees safety upon convergence (Theorem 1), some applications require stricter assurances during the refinement process. Therefore, we present a modification to Alg. 1 which guarantees a reduction of false positive states with every iteration:

Definition 4 (False positive states)**.** *A false positive state is a state x that a value function h characterizes as safe which is not part of the viability kernel for a given dynamical system and a constraint set $\mathcal{L}$, i.e. any x such that $x \in \mathcal{H} \cap \mathcal{S}(\mathcal{L})^C$.*

The standard REFINECBF algorithm does not provide this guarantee, as warm-starting with an arbitrary candidate function implies the HJI-VI update (13) is not a contraction mapping, allowing $\mathcal{H}$ to expand to include unsafe states.

To address this, we propose SAFEADAPT-REFINECBF (Alg. 3), a two-phase algorithm that ensures a monotonic reduction in false positive states:

1) **Retraction phase**: Initially the algorithm uses the contractive HJI-PDE update rule (line 9). This forces the value function's safe set to shrink until it represents a

Algorithm 3 SAFEADAPT-REFINECBF (In-the-loop)

Require: $h^0(x) : \mathcal{X} \mapsto \mathbb{R}$: Initial value function
$\ell(x) : \mathcal{X} \mapsto \mathbb{R}$: Current constraint function

1: $k \leftarrow 0$
2: retracting $\leftarrow$ **true**
3: $h^0(x) \leftarrow \min\{\ell(x), h^0(x)\}$
4: **while** $|h^{(k)}(x) - h^{(k-1)}(x)| > \epsilon$ **or** retracting (**or** Robot operating) **do**
5: **if** retracting AND $|h^{(k)}(x) - h^{(k-1)}(x)| < \epsilon$ **then**
6: retracting $\leftarrow$ **false**
7: **end if**
8: **if** retracting **then**
9: $h^{(k+1)}(x) \leftarrow h^{(k)}(x) + \Delta^{(k)} \min\{0, L_F^* h^{(k)}(x)\}$
10: **else**
11: $h^{(k+1)}(x) \leftarrow \min\left\{\ell(x), h^{(k)}(x) + \Delta^{(k)} L_F^* h^{(k)}(x)\right\}$
12: **end if**
// Environment updates can occur at a different rate
13: Observe environment
14: $\mathcal{U}^{(k+1)}, \mathcal{D}^{(k+1)}, \ell^{(k+1)} \leftarrow$ Update $(\mathcal{U}^{(k)}, \mathcal{D}^{(k)}, \ell^{(k)})$
15: **publishCurrentCBF**$(h^{(k+1)})$
16: **if** $\mathcal{U}^{(k+1)} \subseteq \mathcal{U}^{(k)}$ or $\mathcal{D}^{(k+1)} \supseteq \mathcal{D}^{(k)}$ or $\exists\, x$ such that $h^{(k+1)}(x) > \ell^{(k+1)}(x)$ **then**
17: retracting $\leftarrow$ **true**
18: $h^{(k+1)}(x) \leftarrow \min\{\ell^{(k+1)}(x), h^{(k+1)}(x)\}$
19: **end if**
20: $k \leftarrow k + 1$
21: **end while**
22: **return** $h^*(x) = h^{(k)}(x)$ $\triangleleft$ *Converged value function*

provably safe, control-invariant subset of the viability kernel. At this point, the set of false positives is empty.

2) **Refining phase**: Once a safe set has been established, the algorithm switches to the HJI-VI update rule (line 11), which is not contractive. This allows the safe set to expand to improve performance, while Lemma 4 guarantees that safety, once established, is never lost.

The two-phase implementation in Alg. 3, , with the in-the-loop extension in teal, guarantees that safety improves with every iteration, as formalized hereafter in Theorem 2.

Theorem 2 (Safer with every iteration)**.** *Alg. 3 monotonically decreases the rate of false positive states with every iteration.*

The proof of Theorem 2 relies on previously introduced Lemmas 2 and 4. In addition, we introduce Lemma 5 which guarantees the contraction phase converges to a control invariant set to finalize the proof.

Lemma 5 (Convergence to a control invariant set)**.** *If* (9) *converges to* $h^*(x)$ *as* $t \to -\infty$ *for* $\ell(x) = h^0(x)$*, i.e.* $h^*(x)$ *characterizes a stationary solution of* (10) *initialized with* $h(x, 0) = h^0(x)$*, then its associated 0-superlevel set* $\mathcal{H}^*$ *is control invariant, i.e.* $L_F^* h^*(x) \geq 0$ *for all* $x \in \partial\mathcal{H}^*$*.*

Proof. The stationary version of the continuous PDE CBVF, (10), is as follows:

$$\min\left\{0, \max_{u \in \mathcal{U}} \min_{d \in \mathcal{D}} D_x h(x, t) \cdot F(x, u, d)\right\} = 0. \tag{17}$$

Upon inspection of the second term of the minimum operator, we have the same inequality as in Lemma 1, and hence guarantee control invariance upon convergence. □

We are now ready to prove Theorem 2.

Proof of Thm. 2. Alg. 3 is split into 2 stages, (a) *retracting* and (b) *refining*. We will show that (a) iteratively decreases the number of false positive states to 0, and that subsequently (b) maintains 0 false positive states with every iteration.

For (a), in the *retracting* phase, we use the PDE-based DP update, (11), in line 9. By inspection, $h(x, t_1) \leq h(x, t_2)$ for all x for any $t_1 \leq t_2 \leq 0$ by (11), thus $\mathcal{H}(t_1) \subseteq \mathcal{H}(t_2)$. This implies, by Def. 4, that the set of false positive states monotonically decreases with the iterations.

Next, by Lemma 5, if (9) converges to $h_a^*(x)$, $\mathcal{H}_a^*$ is a control invariant set[1]. Additionally, by Lemma 2, $\mathcal{H}_a^* \subseteq \mathcal{S}(\mathcal{L})$, so by Definition 4, post stage (a), $\mathcal{H}_a^*(x)$ has 0 false positive states.

For stage (b), refining a safe CBVF (here h_a^*) preserves safety (Lemma 4). In other words, $\mathcal{H}_b^{(k)}$ has 0 false positive states for every k, concluding the proof. □

D. Implications for online adaption in evolving environments

The guarantees established previously assume a static environment, as is standard in HJR literature [6]. We now analyze the practical scenario where either a converged or converging value function faces a sudden environmental change. While the refinement process will eventually converge to a new safe value function, the safety filter's transient behavior during this adaptation is critical for safety. We categorize the nature of these environmental changes as follows:

- **Safety enhancing updates** $(\mathcal{U} \uparrow, \mathcal{D} \downarrow, \mathcal{L} \uparrow)$: These changes enlarge the true viability kernel, such as increased actuation limits, reduced disturbance bounds, or a decrease in failure states. These updates simply render the current value function to be "safer", thus maintaining or improving the degree of safety of the associated filter.
- **Safety decreasing updates** $(\mathcal{U} \downarrow, \mathcal{D} \uparrow, \mathcal{L} \downarrow)$: These changes shrink the true viability kernel, such as reduced actuation limits, larger disturbances, or new obstacles. In this scenario, the existing value function may now falsely label unsafe states in the new environmental condition as safe. For this scenario, to quantify the degree of safety (through Def. 4), we compare the first observation after the update $(k + 1)$ with the prior value function (k), with both considering safety with respect to the novel environment.

The two proposed algorithms handle safety-decreasing updates differently. REFINECBF has no special mechanism for updates; it continues iterating and eventually (re-)converges to a safe value function for the new environment, but offers no guarantees during this transient period. In contrast, SAFEADAPT-REFINECBF is explicitly designed to handle this scenario robustly. As detailed in Alg. 3, at risk safety-decreasing updates force the algorithm back to its retraction phase (line 17). This ensures the system contracts (if needed) to a control invariant set under the new environment before

[1]The subscript a denotes stage a of SAFEADAPT-REFINECBF.

attempting to expand again. This provides a principled way to ensure that the safety improves with each iteration, allowing Thm. 2 to extend to in-the-loop operation.

Note that this algorithm is primarily intended for piecewise stationary environments, rather than those with continuously time-varying dynamics. In addition, the safety of the filter is not assured during safety decreasing updates, see Remark 4.

Remark 4 (Safety of the filter)**.** *The safety increasing result of Alg. 3 between iterations is defined as a decrease in false positive states. This does not imply that safety of the system is guaranteed when faced with an unexpected safety decreasing update, as the state of the system can be within the newly false positive set. During the retraction phase, to encourage returning to safety, it is possible to either (i) modify the nominal controller's goal (if goal position-conditioned) to the closest state inside the current safe set or (ii) include a* $D_t h^{(k)}(x) \approx \frac{h^{(k)} - h^{(k-1)}}{\Delta^{(k)}}$ *term to tighten the linear inequality of* (5)*, although this is now not guaranteed to be feasible with the input constraints.*

E. Use-cases and limitations of REFINECBF

The dense and uniform nature of refineCBF's global computation is highly amenable to massive parallelization on modern GPUs, offering the potential for significant computational speedups. Nonetheless, a key property of the value function $h^*(x)$ obtained by REFINECBF is that it satisfies a stronger condition than is strictly necessary. While a standard CBF only requires its 0-superlevel set to be control invariant, the converged HJI-based value function has the property that all of its superlevel sets $\{x \mid h^*(x) \geq \beta\}$ for all $\beta \in \mathbb{R}$ are control invariant. Enforcing this unnecessarily strong condition can increase the computational burden and number of iterations required for convergence. In addition, while converging, this method requires updating the entire grid at each iteration. This is particularly inefficient in common scenarios where the initial candidate CBF is already "mostly safe" and requires only minor, localized corrections rather than a full global refinement. This limitation directly motivates HJ-PATCH, which is designed for precisely this use case.

IV. PATCHING CBFS

As a computationally efficient alternative to REFINECBF, we introduce HJ-PATCH, an algorithm designed to "patch" an existing candidate value function, h^0, rather than performing a full global refinement. The core trade-off is this: instead of converging to a control invariant set within the constraint set $\mathcal{L}$ (as REFINECBF does with the VI formulation), HJ-PATCH finds the largest control invariant subset within the 0-superlevel set of the initial guess, h^0. This is achieved by using the contractive HJI-PDE formulation (10).

The primary source of computational speedup comes from a "local" dynamic programming approach. We introduce the following notation to denote near-boundary states of the value function h and its 0-superlevel set $\mathcal{H}$:

$$\partial^\zeta \mathcal{H} = \{x \mid |h(x)| \leq \zeta\}$$

Our key insight is that for the HJI-PDE formulation, we only need to update states that directly impact the invariance of the approximate safe set, i.e. value decreasing states near the boundary. Therefore, HJ-PATCH maintains an active set of states, which we denote R. At each iteration, only the states in this active set are updated, drastically reducing the computational cost compared to a global update. This active set is intelligently truncated to exclude states that are non-contractive (and thus locally safe), and expanded to include neighbors of states whose values decrease; this ensures that the "patching" correctly propagates while maintaining computational efficiency. The full method is detailed in Algorithm 4.

Upon convergence, HJ-PATCH provably recovers the same 0-superlevel set as a global contraction - the viability kernel of the initial set, $\mathcal{S}(\mathcal{H}^{(0)})$, see Theorem 3. Additionally, we recover a safe CBVF upon convergence for an appropriately chosen γ in (5). The quality of the converged value function is directly correlated to the quality and conservativeness of the initial candidate value function h^0.

Algorithm 4 HJ-PATCH (In-the-loop)

Require: $h^0(x) : \mathcal{X} \mapsto \mathbb{R}$: Initial value function
$\ell(x) : \mathcal{X} \mapsto \mathbb{R}$: Constraint function (Optional)
$C \subseteq \mathcal{X}$: Oracle-certified safe cells (Optional)

// Active set is set of potentially unsafe states
$h^0(x) \leftarrow \min\{\ell(x), h^0(x)\}$
$R^{(0)} \leftarrow \partial^\zeta \mathcal{H}^{(0)}$ ◁ *Initialize active set*
$R^{(0)} \leftarrow R^{(0)} \setminus C$ ◁ *Remove oracle-certified cells*
$k \leftarrow 0$
// Boundary is certified once $R^{(k)}$ *is empty*
while $R^{(k)}$ is not empty (or Robot operating) **do**
  for $x \in R^{(k)}$ **do**
    $h^{(k+1)}(x) \leftarrow h^{(k)}(x) + \Delta^{(k)} \min\{0, L_F^* h^{(k)}(x)\}$
  end for
  $\mathcal{H}^{(k+1)} \leftarrow \{x \mid h^{(k+1)}(x) \geq 0\}$
  // Value-decreasing states form set of interest
  $Q^{(k+1)} \leftarrow \{x \mid h^{(k+1)}(x) \neq h^{(k)}(x) \wedge x \in R^{(k)}\}$
  // Pad set with neighbors if near safe set boundary
  $R^{(k+1)} \leftarrow \mathbf{pad}^p(Q^{(k+1)}) \cap \partial^\zeta \mathcal{H}^{(k+1)}$
  Observe environment
  $\mathcal{U}^{(k+1)}, \mathcal{D}^{(k+1)}, \ell^{(k+1)} \leftarrow$ Update $(\mathcal{U}^{(k)}, \mathcal{D}^{(k)}, \ell^{(k)})$
  // Less actuation or more disturbance can be unsafe
  if $\mathcal{U}^{(k+1)} \subseteq \mathcal{U}^{(k)}$ or $\mathcal{D}^{(k+1)} \supseteq \mathcal{D}^{(k)}$ **then**
    $R^{(k+1)} \leftarrow \partial^\zeta \mathcal{H}^{(k+1)}$ ◁ *All boundary states active*
  end if
  // New obstacles can be unsafe
  if $\exists\, x$ such that $h^{(k+1)}(x) > \ell^{(k+1)}(x)$ **then**
    $T \leftarrow \{x \mid \ell^{(k+1)}(x) < h^{(k+1)}\}$ ◁ *New active states*
    $h^{(k+1)}(x) \leftarrow \min\{\ell^{(k+1)}(x), h^{(k+1)}(x)\}$
    $\mathcal{H}^{(k+1)} \leftarrow \{x \mid h^{(k+1)}(x) \geq 0\}$
    // New active set includes prior and obstacle-updated active states around new boundary
    $R^{(k+1)} \leftarrow (R^{(k+1)} \cup T) \cap \partial^\zeta \mathcal{H}^{(k+1)}$
  end if
  publishCurrentCBF$(h^{(k+1)})$
  $k \leftarrow k + 1$
end while
return $h^*(x) = h^{(k)}(x)$ ◁ *Converged value function*

A. Algorithm details

We discuss HJ-PATCH for offline convergence and online adaptation below.

a) Offline patching: Alg. 4 takes an approximate CBF $h^0(x)$, optionally a constraint function $\ell(x)$, and optionally a set of oracle-certified safe states as its input. As detailed in Algorithm 4, the iterative process is initialized with a candidate value function $h^0(x)$. The initial active set, $R^{(0)}$, comprises the states near the boundary of the candidate safe set (line 2). If an external oracle can certify that a subset of states C is already safe (e.g., via neural network verification techniques [40], [41]), these states can be removed from the active set for further computational efficiency (line 3). The iterative loop applies the contractive HJI-PDE update (11) to states in the active set (lines 6- 7). The active set is then updated to filter out locally safe states, i.e. non-decreasing values (line 10) after which it is expanded to include its neighbors (line 11). $\mathbf{pad}^p$ denotes padding a space-discretized set with its p neighbors in every dimension (equivalent to the Minkowski sum with a ball of small radius in continuous space):

$$\mathbf{pad}^p(R) = \bigcup_{r \in R} X^p(r). \tag{18}$$

At every iteration k, $R^{(k)} \subseteq \partial^\zeta \mathcal{H}^{(k)}$ by line 11, ensuring only a subset of the boundary cells are "active". This procedure repeats until the active set is empty (line 26) indicating convergence.

b) Online patching: When used in-the-loop, the algorithm includes additional logic to handle environmental changes. Safety-decreasing updates require special care to ensure that states previously assumed safe are re-evaluated. If actuation limits decrease or disturbance bounds increase, the algorithm conservatively resets the active set to the current boundary states to force a full re-evaluation (line 15). If new obstacles appear, the value function is first clipped to respect the new constraints, and the active set is expanded to include any boundary states that coincide with the new obstacles' boundaries (lines 17- 21). Safety-enhancing updates do not require changing the active set.

B. Theoretical guarantees for HJ-PATCH

Because HJ-PATCH updates only a subset of states at each iteration, our theoretical analysis must explicitly consider the discretized state-space on which the algorithm operates. These guarantees assume a static environment. Theorem 3 provides the key theoretical underpinning of HJ-PATCH.

Theorem 3 (Algorithm 4 recovers the viability kernel of the initial candidate safe set)**.** *If Alg. 4 converges, then, initializing with $h^{(0)}(x) = h^0(x)$, we have $\mathcal{H}^* = \mathcal{S}(\mathcal{H}^0)$.*

The proof of Theorem 3 relies on three supporting lemmas. We first define a spatially discretized equivalent of control invariance, which we term discrete-approximatedcontrol invariance. We then prove that HJ-PATCH converges to a discrete-approximatedcontrol invariant set (Lemma 6), and that, leveraging Lemma 7, this set is a superset of the viability kernel (Lemma 8). Together, these properties are sufficient to prove the main theorem.

Inspired by Nagumo's Theorem (2), we first define a discrete-approximatedcontrol invariant set for a discretized state space.

Definition 5 (Discrete-approximatedcontrol invariant set $\mathcal{H}$)**.** *Let $h(x)$ be defined on a discretized state-space and $\mathcal{H}$ its 0-superlevel set. Assuming $\frac{\partial h}{\partial x} \neq 0$ for all $x \in \partial^\zeta \mathcal{H}$, then $\mathcal{H}$ is discrete-approximatedcontrol invariant if and only if*

$$L_F^* h(x) \geq 0 \text{ for all } x \in \partial^\zeta \mathcal{H}. \tag{19}$$

While (19) is spatially more restrictive than (2), it acts as a conservative buffer against interpolation errors that otherwise preclude a strict guarantee on coarse grids or for value functions with high Lipschitz constants.

Lemma 6 (Algorithm 4 converges to a control-invariant set)**.** *If Alg. 4 converges, then the safe set $\mathcal{H}^*$ obtained upon termination of the algorithm is discrete-approximatedcontrol invariant.*

Proof. Without loss of generality, we consider $C = \emptyset$, i.e. no cells are oracle-certified upon initiation. Recall by Def. 5 that $\mathcal{H}$ is discrete-approximatedcontrol invariant if and only if $L_F^* h(x) \geq 0$ for all $x \in \partial^\zeta \mathcal{H}$. Specifically, we seek to show that for every iteration k, every boundary cell not in the current active set has a positive Lie derivative, i.e. if $x \in \partial^\zeta \mathcal{H}^{(k)} \setminus R^{(k)}$, then $L_F^* h^{(k)}(x) \geq 0$. Then, if Alg. 4 converges, i.e. $R = \emptyset$, we either have (i) $\partial^\zeta \mathcal{H} = \emptyset$ implying $\mathcal{H} = \emptyset$ or (ii) $L_F^* h(x) \geq 0$ for all $x \in \partial^\zeta \mathcal{H}$.

We proceed by induction. Recall that $R^{(0)} = \partial^\zeta \mathcal{H}^{(0)} \setminus C$. We have to certify that if $x \in \partial^\zeta \mathcal{H}^{(0)} \setminus R^{(0)} = \partial^\zeta \mathcal{H}^{(0)} \cap C$, then $L_F^* h^{(0)}(x) \geq 0$, which is guaranteed for all $x \in C$.

Next, for iteration k we assume that if $x \in \partial^\zeta \mathcal{H}^{(k)} \setminus R^{(k)}$, then $L_F^* h^{(k)}(x) \geq 0$. We hence need to show that $x \in \partial^\zeta \mathcal{H}^{(k+1)} \setminus R^{(k+1)}$ implies $L_F^* h^{(k+1)}(x) \geq 0$, or its contrapositive; if $L_F^* h^{(k+1)}(x) < 0$, then $x \in R^{(k+1)}$ for all $x \in \partial^\zeta \mathcal{H}^{(k+1)}$. For all $x \in \partial^\zeta \mathcal{H}^{(k+1)}$ it is then sufficient to show $x \in R^{(k+1)}$ for the following scenarios: (i) $L_F^* h^{(k+1)}(x) = L_F^* h^{(k)}(x) < 0$ and (ii) $L_F^* h^{(k+1)}(x) \neq L_F^* h^{(k)}(x)$.

For (i), by assumption, if $L_F^* h^{(k)}(x) < 0$, then $x \in R^{(k)}$. As $L_F^* h^{(k)}(x) < 0$, by (11), $h^{(k+1)}(x) \neq h^{(k)}(x)$, hence $x \in Q^{(k+1)}$, and by extension $x \in R^{(k+1)}$ if $x \in \partial^\zeta \mathcal{H}^{(k+1)}$.

For (ii), by (14), $L_F^* h^{(k+1)}(x) \neq L_F^* h^{(k)}(x)$ implies that there exists a state $y \in X^p$ such that $h^{(k+1)}(y) \neq h^{(k)}(y)$. Hence, by Alg. 4 (L10), $y \in Q^{(k+1)}$. By definition, x is a neighbor of y (bijective mapping), hence $x \in \mathbf{pad}^p(Q^{(k+1)})$. Then, $x \in R^{(k+1)}$ if $x \in \partial^\zeta \mathcal{H}^{(k+1)}$ (Alg. 4 (L11)). □

We also point out the following:

Remark 5. *It is possible that positive-valued states at an iteration k (or oracle-safe cells at iteration $k = 0$) become part of the active set at a later iteration through padding.*

Lemma 7 (Optimistic global warm-start HJ reachability recovers the viability kernel [14, Thm. 2])**.** *Let $h^0(x)$ be the initial function and $h^*(x)$ be the value obtained upon convergence of* (10)*. Then, there exists a continuous function $m : \mathcal{X} \mapsto \mathbb{R}$ such that $\mathcal{S}(\mathcal{H}^0) = \{x \mid m(x) \geq 0\}$, $h^0(x) \geq m(x)$ for all $x \in \mathcal{X}$, and $h^*(x) = m(x)$ for all $x \in \mathcal{X}$.*

Lemma 8. *[Algorithm 4's safe set upon convergence is a superset of the viability kernel] If Alg. 4 converges, then, initializing with $h^{(0)}(x) = h^0(x)$, we have $\mathcal{H}^* \supseteq \mathcal{S}(\mathcal{H}^0)$ upon convergence.*

Proof. We denote Λ as the single-step (i.e. within the while loop) operator for Alg. 4 and Γ as the single-step operator for standard reachability, i.e. (11). It applies to both $h(x)$ and $\mathcal{H}$. Hence, $\mathcal{H}^{(k+1)} = \Lambda(\mathcal{H}^{(k)})$. Lastly, we note that $h(x) \geq g(x)$ for all x if and only if $\mathcal{H} \supseteq \mathcal{G}$, with $\mathcal{H}$ and $\mathcal{G}$ the 0-superlevel sets of $h(x)$ and $g(x)$.

Then, if for every iteration k, $\mathcal{H}^{(k)} \supseteq \mathcal{S}(\mathcal{H}^0)$, we converge to a superset of $\mathcal{S}(\mathcal{H}^0)$. We proceed by induction. By definition of the viability kernel (Def. 2), we have $\mathcal{H}^{(0)} \supseteq \mathcal{S}(\mathcal{H}^0)$.

Next, for iteration k we assume $h^{(k)}(x)$ is such that $\mathcal{H}^{(k)} \supseteq \mathcal{S}(\mathcal{H}^0)$. We hence need to show that $\mathcal{H}^{(k+1)} \supseteq \mathcal{S}(\mathcal{H}^0)$. Particularly, we will show that $\mathcal{H}^{(k+1)} \supseteq \mathcal{G}$, and $\mathcal{G} \supseteq \mathcal{S}(\mathcal{H}^0)$, for $\mathcal{G} = \Gamma(\mathcal{H}^{(k)})$.

By construction, for any value function $h(x)$, by Alg. 4, we have $\Lambda(h(x)) \geq \Gamma(h(x))$, hence $\Lambda(\mathcal{H}) \supseteq \Gamma(\mathcal{H})$. Thus, $\mathcal{H}^{(k+1)} = \Lambda(\mathcal{H}^{(k)}) \supseteq \Gamma(\mathcal{H}^{(k)}) = \mathcal{G}$.

It remains to show that $\mathcal{G}$ is such that $\mathcal{G} \supseteq \mathcal{S}(\mathcal{H}^0)$. By Lemma 7 and given $\mathcal{H}^{(k)} \supseteq \mathcal{S}(\mathcal{H}^0)$, applying standard reachability (11) recursively to $h^{(k)}(x)$ recovers a value function $\hat{h}^*(x) = \Gamma \circ \cdots \circ \Gamma(h^{(k)}(x))$, such that $\hat{\mathcal{H}}^* = \mathcal{S}(\mathcal{H}^0)$. Noting that Γ is a contraction mapping, we have $\mathcal{S}(\mathcal{H}^0) = \Gamma \circ \cdots \circ \Gamma(\mathcal{H}^{(k)}) \subseteq \Gamma(\mathcal{H}^{(k)}) = \mathcal{G}$.

Combining, we have $\mathcal{H}^{(k+1)} \supseteq \mathcal{S}(\mathcal{H}^0)$, concluding the proof. □

We are now ready to prove Theorem 3

Proof of Thm. 3. Combining Lemma 6 and Lemma 8 and noting that the viability kernel is defined as a set's largest control invariant subset (Def. 2), we directly obtain $\mathcal{H}^* = \mathcal{S}(\mathcal{H}^0)$. □

Furthermore, because the algorithm relies on a contractive update, it provides the same strong increasing safety guarantee as SAFEADAPT-REFINECBF.

Theorem 4 (Safer with every iteration)**.** *Alg. 4 montonically decreases the rate of false positive states with every iteration*

Proof. HJ-PATCH guarantees contraction of the value function h with every iteration, thus guaranteeing a monotonic decrease of the number of false positive states (Def. 4) □

C. Implications and use-cases for HJ-PATCH

The theoretical results highlight the fundamental difference between our two proposed algorithms. REFINECBF can expand the safe set beyond the initial guess, making it ideal for improving overly conservative initializations, including analytical and backup CBFs. In contrast, HJ-PATCH is designed to efficiently patch an initial guess by finding its largest control invariant subset. This makes it particularly well-suited for patching "almost-safe" learned CBFs, which may be performant but contain small, critical failure regions. One core difference between the theory outlined for REFINECBF and HJ-PATCH is the difference in the obtained safe set. Additionally, unlike HJ-PATCH, REFINECBF can reduce conservatism after safety-increasing updates by expanding the safe set.

The primary benefit of HJ-PATCH is its computational performance. However, the speedup is most significant when the "patch" needed is very localized (relative to the entire state space). The algorithm's local update structure is inherently well-suited for modern multi-core CPUs, which excel at the logic required to process a small, irregular active set. Our CPU-based implementation leverages this for significant performance gains, especially for localized active sets. A GPU implementation presents a more nuanced trade-off: while a naive approach would suffer from poor hardware utilization, high performance is theoretically achievable using advanced patterns like stream compaction to first gather the active states. This, however, adds a non-trivial pre-processing overhead. Given the demonstrated effectiveness of the CPU for patching localized errors, we leave a detailed GPU implementation and performance comparison as an avenue for future work.

V. EXPERIMENTS (SIMULATION)

Previous work established the efficacy of REFINECBF [27] and its computational scalable successor, HJ-PATCH [28], for offline value function refinement. We demonstrated that these methods can improve conservative, analytically-derived CBFs for adaptive cruise control, correct unsafe and relax conservative CBFs for high-dimensional quadcopter dynamics, and scale to patch neural network-based CBFs. However, these validations were conducted offline on static, pre-defined environments.

The central contribution of this paper is to evaluate REFINECBF and HJ-PATCH for in-the-loop adaptation to non-stationary environments. Here, the algorithms must update the safety-critical value function in real-time in response to environmental changes, such as the appearance of new obstacles or shifts in system dynamics.

A key challenge for in-the-loop deployment is computational cost. However, a principal advantage of our methods is their ability to warm-start the refinement process from a previously computed value function, a prior CBF from a similar environment, or even a simple signed-distance function. This warm-starting capability is crucial for rapid adaptation. While formal safety guarantees are contingent upon the convergence of the algorithms, SAFEADAPT-REFINECBF and HJ-PATCH theoretically ensure that the value function's safety improves with every iteration. In practice, REFINECBF has almost monotonic improvement in our experiments. Many realistic environmental pertubations, such as those in our experiments, are "localized" in their impact. This allows the algorithms to rapidly refine or patch the value function and consistently converge within a few iterations.

To validate this online adaptation capability, we conduct a series of experiments in realistic simulators, comparing our methods against relevant baselines. Each experiment is repeated over 10 runs (5 if all failures) to ensure statistical significance. All simulations were performed on a desktop computer equipped with an NVIDIA RTX 3090 GPU and 32GB of RAM. All controllers run at 50Hz with the value function updates running in parallel with $\Delta^{(k)} = 0.1$s for all iterations k at a typical rate of $1 - 3$Hz.

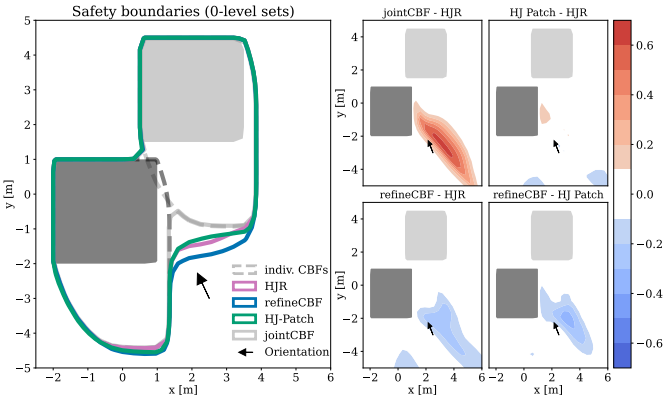


Fig. 1: Comparison of converged value functions for REFINECBF, HJ-PATCH, HJR, and jointCBF at a fixed robot orientation (see arrow) and velocity ($v = 1.0$m/s) slice. (Left) The 0-level sets show that naively combining constraints (jointCBF) incorrectly merges the safe sets, creating an unsafe region between the obstacles. HJ-PATCH and REFINECBF both produce safe, comparable boundaries compared to the HJR baseline. (Right) The difference plots highlight that HJ-PATCH's value function is a close match to the HJR solution, while REFINECBF is more conservative (negative differences) and jointCBF is dangerously optimistic (large positive differences).

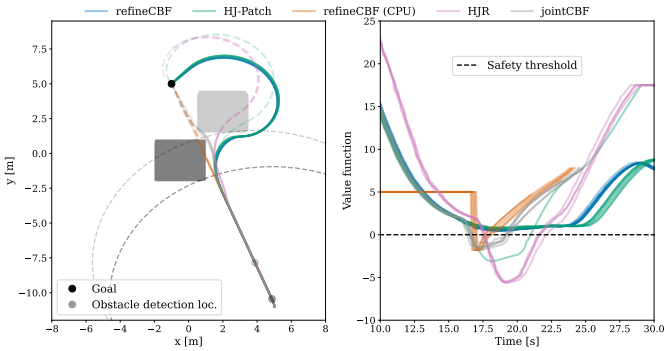


Fig. 2: Online performance with limited-range obstacle detection. These plots show system behavior across 10 simulation rollouts where obstacles are discovered online. (Left) shows the trajectories for all methods (with dotted lines for trajectories post collision) for multiple rollouts. We additionally visualize the detection points based on the limited range of the sensor. (Right) visualizes the value function (with the safety boundary). These rollouts demonstrate the need for refinement, as HJR and jointCBF lead to collision. Additionally, when a GPU is not available, HJ-PATCH's computational efficiency enables patching rapidly to provide safety, whereas REFINECBF (CPU) fails.

A. *Ground robot: Combining obstacle-dependent CBFs*

For systems like ground vehicles, CBFs for individual obstacles can be derived analytically or via Hamilton-Jacobi (HJ) reachability. A common practice is to enforce these individual CBFs as independent constraints in an optimization problem. However, these constraints are not necessarily jointly feasible, which can compromise safety.

Our experiment investigates this problem in a Gazebo simulation where a Jackal robot, modeled as a dynamically extended unicycle model with a limited-range LiDAR navigates to a goal. We model the jackal robot as a dynamically extended bicycle model:

$$\dot{p}_x = v\cos(\theta), \quad \dot{p}_y = v\sin(\theta), \quad \dot{v} = a, \quad \dot{\theta} = \omega, \qquad (20)$$

with inputs $[a, \omega]$. Upon detecting a new obstacle, we must update the safety controller online. We compare our warm-started methods—REFINECBF (on both CPU and GPU) and HJ-PATCH (CPU)—against two baselines: naively combining constraints (jointCBFs) and re-solving the problem from scratch using HJ reachability (HJR). All methods operate under a minimum forward velocity constraint of 0.1 m/s to prevent deadlock and to showcase how our methods systematically incorporates bounded control inputs.

First, we analyze the quality of the converged value functions (if given enough time and perfect knowledge of the environment) in a static two-obstacle environment (Fig. 1), where REFINECBF and HJ-PATCH are initialized from the individual CBF of each obstacle. The individual CBFs, h_i's, are computed by running (11) until convergence for the rectangular single obstacle constraint function $\ell(x) = \ell(p_x, p_y)$, thus $h_i(x) = h_i(p_x, p_y, v, \omega)$. Naively combining individual CBFs $h(x) = min_i\{h_i(x)\}$ (jointCBF) produces an overly optimistic and unsafe value function. In contrast, HJ-PATCH successfully computes a safe value function that closely approximates the ground truth from HJR, while REFINECBF is safe but more conservative.

TABLE II: Quantiative measure (↓ better) of number of **collisions** of jackal simulation experiments for online adaptation. Experiments that constantly lead to collisions are only executed 5 times. These measures highlight the benefits of REFINECBF and HJ-PATCH.

REFINECBF	REFINECBF (CPU)	HJ-PATCH (CPU)	jointCBFs	HJR
0/10	5/5	1/10	5/5	5/5

Next, we evaluate the critical challenge of online adaptation as the robot discovers obstacles in real time (Fig. 2). The results, quantified in Table II, demonstrate that the baseline approaches lead to collisions for distinct reasons. The jointCBF method is inherently unsafe, as it fails to account for joint feasibility between obstacle constraints. In parallel, re-computing the ground-truth safe set from scratch (HJR) is computationally intractable for real-time updates. In contrast, while REFINECBF requires a GPU to be fast enough for safe online updates, HJ-PATCH provides robust safety performance on a CPU in 9/10 rollouts. A single outlier is visible in Fig. 2 (right). The logs show a communication delay between computation of $h^{(k)}$ and its transmission to the safety filter at 2 subsequent iterations, which could explain the failure. This highlights HJ-PATCH's computational efficiency, making it a viable solution for resource-constrained platforms where GPU acceleration is unavailable.

B. *Quadcopter: Backup CBF comparison*

Backup Control Barrier Functions (Backup CBFs) are a method for ensuring safety by relying on a pre-defined, expert-chosen backup maneuver. A control action is deemed safe if the system can always execute this fixed backup policy (e.g., "brake hard") for a set time without violating safety constraints [42]. This approach avoids complex online optimization but is inherently limited by the quality and applicability of the single, fixed backup policy. We compare against a variant that mixes nominal and backup controllers, as it is more computationally tractable.

We demonstrate this limitation in a planar quadcopter simulation where the goal is occluded by an obstacle (Fig. 3), with

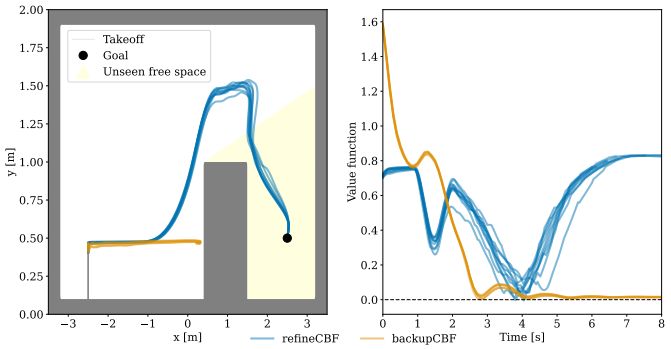


Fig. 3: Online adaptation and deadlock avoidance in a quadcopter navigation task. The quadcopter must navigate to a goal in a region initially occluded by an obstacle. (Left) Trajectories show REFINECBF successfully discovering a path over the obstacle, progressively incorporating the newly seen free space (yellow) into the safe set, then iteratively incorporating this new information into its safe set. In contrast, the backupCBF method gets stuck in a deadlock, unable to find a path to the goal with its fixed backup policy. (Right) The value function confirms this behavior. REFINECBF's value dips as it navigates the obstacle and unseen space boundary, and then recovers as the safe set expands. The backupCBF value drops and oscillates near the safety boundary (value=0), indicating a persistent deadlock. Further analysis on why REFINECBF avoids deadlock is provided in Fig. 10a.

dynamics:

$$\dot{p}_y = v_y, \quad \dot{p}_z = v_z, \quad \dot{v}_y = g\tan(\phi), \quad \dot{v}_z = T - g, \quad (21)$$

with inputs $u = [\phi, T]$. The drone uses a simulated front-facing camera, meaning the safety constraint requires it to remain within the currently visible, obstacle-free space. The nominal controller is a simple LQR designed to hover at the goal, which has no inherent obstacle avoidance capabilities. Our method, REFINECBF, is initialized with the signed-distance function $\ell(x) = \ell(p_y, p_z)$ of the initially known environment, which can be computed efficiently analytically (for polytopes) and through discretization using Fast Marching Methods [43] (any environment).

The experiment highlights a critical failure mode for methods relying on fixed expert policies. The backupCBF method successfully maintains safety but its simple backup maneuver (here: stabilize to a hovering position) is insufficient to navigate the obstacle, resulting in deadlock (Fig. 3). In contrast, REFINECBF characterizes a safety filter over the full dynamics of the system. This obtains more informative gradients that are fully state-dependent, enabling the drone to navigate around the obstacle, as the safe set at a high velocity "nudges" the drone upwards, see Fig. 10b (left), for a visualization.

As new free space is revealed, REFINECBF seamlessly expands the safe set online (Fig. 10b, center and right). We do not compare against HJ-PATCH in this scenario, as its formulation does not support the expansion of the safe set into previously unknown areas.

VI. EXPERIMENTS (HARDWARE)

To validate our approach on a physical platform, these hardware experiments are designed to demonstrate the real-time adaptation capabilities of REFINECBF.

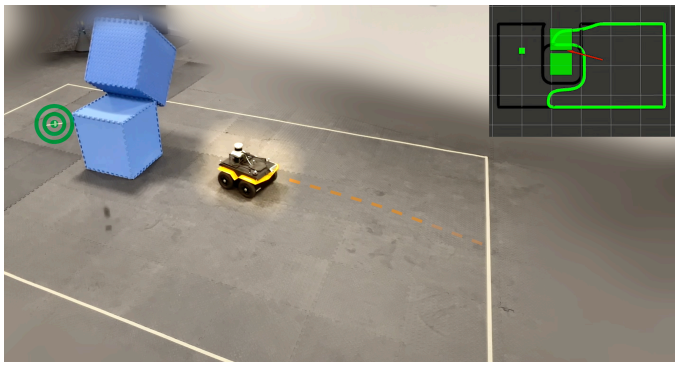


Fig. 4: A snapshot of the Jackal hardware experiment. The robot detects the suddenly fallen obstacle blocking its path. The top-right inset shows the robot's internal map at the moment of detection; the safe set (green) has not yet been updated to incorporate the new SDF map (black), highlighting the need for rapid, in-the-loop replanning to ensure safety.

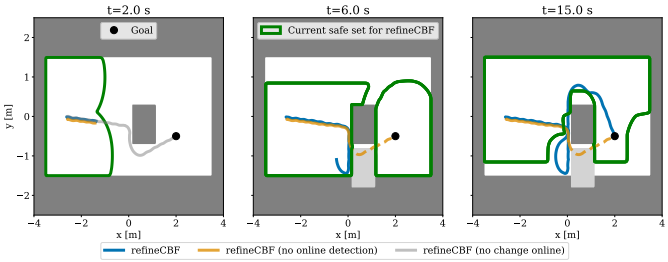


Fig. 5: Trajectory comparison for the Jackal hardware experiment. These plots show time snapshots of the system's trajectory after the environment is altered at $t = 3.0s$. The fully adaptive REFINECBF (blue) successfully backtracks to find a new, safe path. The non-adaptive baseline (refineCBF (no online detection), orange) operates on a stale map, leading directly to a collision with the new obstacle.

- Fully Adaptive REFINECBF (Our Method): Continuously refines its value function in response to environmental changes perceived in real time by the robot's sensors (simulated sensing).
- Static REFINECBF (Baseline): Operates REFINECBF on a static representation of the environment captured only at initialization. This controller does not incorporate subsequent perceptual updates, highlighting the necessity of online adaptation for safety.

Both still iteratively update the value function online, but the baseline does not observe environmental changes in the loop. All experiments were conducted on a desktop computer with an NVIDIA RTX-4090 and 64GB of RAM. All controllers run at 50Hz with the value function updates running in parallel with $\Delta^{(k)} = 0.1$s for all iterations k at a typical rate of $2-5$Hz.

A. Mobile robots: Obstacle appears

We validate our approach in a dynamic hardware experiment that mimics a challenging search-and-rescue scenario where the environment changes unexpectedly. We initialize the safe set using a CBF centered at the starting position p_0 with a small initially safe region, with a learned $h^0(x)$ using [10] with $\ell(x) = \ell(p_x, p_y)$ a circle around p_0. A Clearpath Jackal

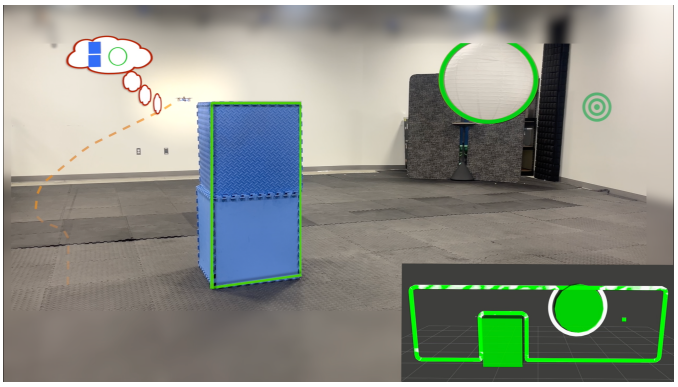


Fig. 6: Quadcopter hardware experiment with a previously occluded obstacle. Snapshot of the physical setup where a quadcopter navigates towards its goal (green circle). The circular obstacle is initially hidden from view by the blue block. The bottom-right inset shows the constraint and safe set at the moment the new obstacle is first detected; the safe set (green) has not yet been updated, highlighting the challenge for the real-time safety filter.

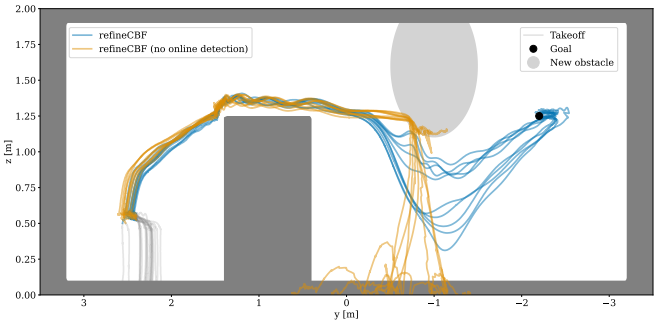


Fig. 7: Trajectories from 10 hardware trials comparing adaptive and non-adaptive REFINECBF. The quadcopter must navigate an environment with an unexpected obstacle (light gray circle) that only becomes visible after clearing the first obstacle. The fully adaptive REFINECBF (blue) successfully avoids the new obstacle in all 10 trials. The non-adaptive baseline (refineCBF (no online detection), orange), operating under the optimistic assumption that unseen space is free, collides in every trial.

UGV, modeled using (20), is tasked with reaching a goal placed behind a two-tiered obstacle structure (Fig. 4). During operation, the top block is deliberately knocked over, blocking the robot's initial path and forcing an online adaptation.

As shown in Fig. 5, the fully adaptive REFINECBF immediately perceives the new obstacle, backtracks, and the safety filter enables safely getting to the goal. In contrast, the non-adaptive baseline, which operates on a stale map of the initial environment, fails to see the change and collides with the fallen block. This result demonstrates that with a simple nominal controller, our online adaptation is critical for maintaining safety in dynamic environments. We present a single, representative run for each case, as this outcome was highly consistent across 5 trials.

B. Drone: Sensing limitations

Robots with forward-facing sensors like cameras or LiDAR have a limited field of view, forcing them to make assumptions about unobserved areas. A common and efficient strategy is to plan optimistically, assuming unobserved space is

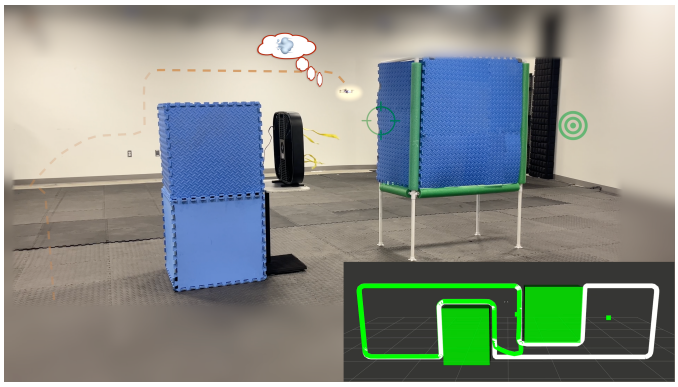


Fig. 8: Hardware experiment with unmodeled aerodynamic disturbances. A snapshot of the quadcopter navigating through a narrow passage containing a fan (wind visualized with yellow streamlines). While the geometry of the passage is known, the aerodynamic disturbance produced by the fan's and its interaction with the environment is not and must be detected and compensated for in real time.

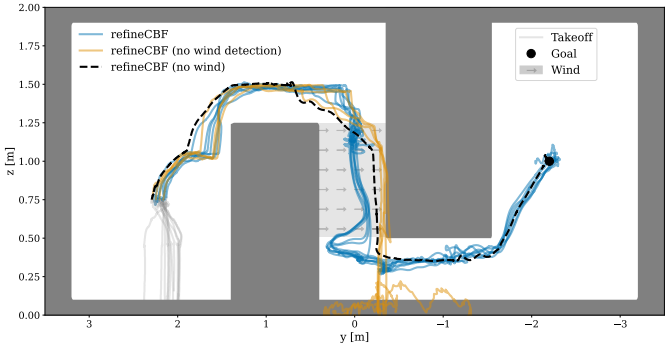


Fig. 9: Trajectories from 10 hardware trials with a dynamic disturbance. The quadcopter encounters a strong wind disturbance while flying between the two known obstacles. The safety value function is initialized as a CBF centered at the starting position with a small initially safe region. The adaptive REFINECBF, which incorporates real-time disturbance changes, successfully compensates and navigates the passage in 9/10 trials. The baseline controller (orange), which assumes nominal dynamics, fails in every trial, consistently being pushed into the wall by the unexpected disturbance.

obstacle-free until proven otherwise. We designed a hardware experiment to demonstrate that REFINECBF's rapid online adaptation makes this optimistic approach both viable and safe.

In our scenario, a quadcopter, modeled using (21), must navigate to a goal that is initially occluded by a large rectangular obstacle (Fig. 6). Hanging from the ceiling behind this obstacle is a second, circular obstacle, completely hidden from the robot's initial view. The initial value function $h^0(x) = \ell(x)$ is the SDF (computed as described in Section V). Across 10 consecutive trials, the fully adaptive REFINECBF successfully perceived the new obstacle (recomputing the constraint function), updated its value function in real time (re-converging in under 2 seconds), and adjusted its trajectory to safely reach the goal. In contrast, the non-adaptive baseline, acting on its initial optimistic map, consistently collided with the unexpected obstacle, as shown in the trajectories in Fig. 7.

Defining the safety value function over all states enables navigating between obstacles without a sophisticated nominal policy, see Fig. 10b for details.

C. Quadcopter: Varying disturbances

Our final hardware experiment tests REFINECBF's ability to adapt not just to geometric changes, but also to unmodeled dynamic disturbances. In this scenario, a quadcopter navigates a fully known environment, but a fan creates a strong, localized wind disturbance within a narrow passageway (Fig. 8). We consider the following dynamics:

$$\dot{p}_y = v_y + d_0, \quad \dot{p}_z = v_z + d_1, \quad \dot{v}_y = g\tan(\phi), \quad \dot{v}_z = T - g,$$

with inputs $u = [\phi, T]$ and disturbances $d = [d_0, d_1]$. We simulate a drone equipped with a flow sensor that allows it to detect and incorporate this disturbance in the safety analysis in real time, with a default value of $d_0, d_1 \in \mathcal{D}_0 = 0$. The bounds $\mathcal{D}_{\text{fan}}$ are fitted a priori for different fan speeds. The initial value function h^0 is a learned CBF, using [10], centered around the initial hover point p_0.

Upon entering the passage and detecting the airflow, the fully adaptive REFINECBF rapidly updates its safety value function to account for these new dynamics. As the results from 10 trials show (Fig. 9), this allows the drone to successfully counteract the disturbance and traverse the passage in 9 out of 10 attempts. We hypothesize that the single failure occured due to the non-uniformity of the "wind" provided by the radial fan, which can destabilize the quadcopter. The non-adaptive baseline, operating on the assumption of nominal dynamics, is unable to compensate for the wind and consistently pushed into the obstacle.

For this experiment, we intialized the CBVF conservatively around its initial hover point, see Fig. 10c (left). This experiment qualitatively demonstrates that the CBVF is able to expand from an initial safe but conservative safe set (Fig. 10c, center and right). It leverages SAFEADAPT-REFINECBF to ensure contraction upon detection of the wind disturbance. The nominal policy is a proportional controller with sub-goals dynamically updated to be the closest safe state (at 0 velocity) to the final goal.

VII. Practical Use-Cases, Limitations & Future Directions

This work provides a step towards scalable, real-time safety adaptation for robotic systems. Our experiments demonstrate that by warm-starting from a prior value function, methods like REFINECBF and HJ-PATCH can successfully adapt to sudden environmental changes, such as new obstacles or unmodeled dynamic disturbances (e.g., wind). This capability is critical for practical applications in evolving environments, including search-and-rescue, autonomous logistics, and long-term mobile robot deployment. However, several limitations highlight important avenues for future research.

While promising, our approach has three principal limitations: idealized sensors, model dependency, and computational scalability.

1) **Idealized Sensors**: In our hardware experiments, we focus on the controls and safety pipeline, and therefore simulated the sensing pipeline and used high-fidelity motion capture for state estimation. This setup bypasses some significant challenges of a full-stack implementation, such as incorporating uncertainty inherent in state estimation and perception.
2) **Known dynamics model:** Our methods fundamentally rely on an accurate known dynamics model with bounded uncertainties or disturbances, i.e. $F(x, u, d)$, limiting adoption to a subset of robotic domains.
3) **Scalability of dynamic programming:** The core of both REFINECBF and HJ-PATCH is based on dynamic programming (DP), operating on a discretized state-space grid. While effective for the systems we studied (up to 4D for on-the-fly refinement), relying on DP inherently limits scalability.

These limitations highlight several key avenues for future work. The most immediate practical step is to integrate realistic sensor modalities, developing methods that update the value function directly from noisy sensor data and explicitly account for perceptual uncertainty.

To address scalability, a promising direction is to merge our framework with modern learning-based approaches. While fully learned value functions have shown promise for high-dimensional systems, they are typically static and cannot adapt online. Some recent approaches have started to enable online variation by parameterizing the value function based on disturbances or obstacles [44]. Building on this, a compelling research path is to synthesize our more general real-time adaptation techniques with these expressive, high-dimensional function approximators. Such a merger could combine the scalability of deep learning with the dynamic responsiveness demonstrated in our work, enabling safe, real-time adaptation for complex robots like manipulators and humanoids.

VIII. Conclusions

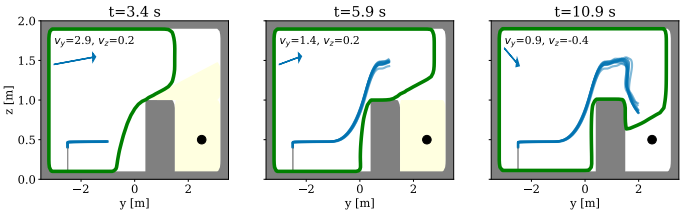
This work demonstrates that HJ reachability, a cornerstone of formal safety analysis, can be made practical for real-time robotic applications when deployed with a CBF-like safety filter. We presented two algorithms, REFINECBF and its successor HJ-PATCH, that effectively adapt safety-critical value functions to in-the-loop changes in a robot's environment or dynamics. By leveraging warm-starting, our methods rapidly re-converge to a safe policy following unexpected events, a capability we validated against multiple baselines in realistic simulations and on physical hardware. Our results highlight a key finding: warm-started HJ reachability provides a principled and effective framework for ensuring safety in dynamic, real-world settings, bridging the gap between theoretical guarantees and practical deployment.

References

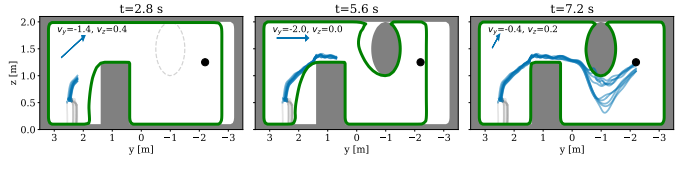
[1] T. Miki, J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning robust perceptive locomotion for quadrupedal robots in the wild," *Science Robotics*, vol. 7, no. 62, 2022.

[2] D. Shah, A. Sridhar, A. Bhorkar, N. Hirose, and S. Levine, "GNM: A General Navigation Model to Drive Any Robot," in *Proc. IEEE Conf. on Robotics and Automation*, 2023.

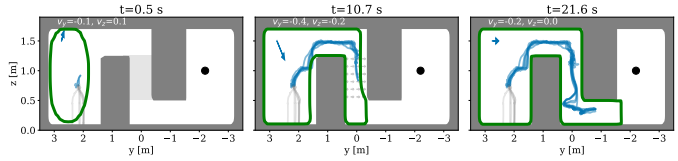
[3] L. Brunke, M. Greeff, A. W. Hall, Z. Yuan, S. Zhou, J. Panerati, and A. P. Schoellig, "Safe learning in robotics: From learning-based control to safe reinforcement learning," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 5, no. 1, pp. 411–444, 2022.



(a) Visualized is the CBF expansion over time for the simulated Crazyflie UAV experiment. The CBVF is initialized as the SDF for the visible free space, and its corresponding safe set is shown in green. Note that the CBVF is in 4D; we therefore show the $y - z$ slice at the current v_y, v_z values (indicated by the blue arrows) along the trajectory. The unseen region is marked in yellow. This experiment demonstrates the benefit of computing CBFs in higher dimensions: at higher speeds (left), the UAV must deviate from the square obstacle more quickly; this results in a CBVF whose gradients force the UAV upwards, avoiding the deadlock that occurs with position-based CBFs as in Fig 3.



(b) Visualized is the 2D slice of the safe set over time for the hardware Crazyflie UAV experiment. The CBVF is initialized as the SDF (without the unobserved circular obstacle). The safe sets in (left, center) are at high horizontal velocity, which results in the CBVF guiding the drone up and down, respectively, proactively providing safety by incorporating the full dynamics and state space information. The comparison to a static REFINECBF is given in Fig. 7. This once again highlights the importance of velocity information in the safety filter to avoid deadlock.



(c) Visualized is the 2D slice of the safe set over time for the wind-affected hardware Crazyflie UAV experiment. The CBVF is initialized within a known conservative safe set (left). The safe set grows initially, then shrinks slightly upon observing the wind (center) leveraging SAFEADAPT-REFINECBF. The safe set continues to expand while providing control invariance after the wind corridor (right), before proceeding to its goal, see also Fig. 9.

Fig. 10: Safe set across the position space evaluated at the current velocities (arrow and slice highlight the velocities). A broad range of value function initializations are presented (conservative SDF (top), optimistic SDF (center), and conservative CBF guess (bottom)), in addition to a broad range of online changes. REFINECBF (or SAFEADAPT-REFINECBF) succeeds in all scenarios, highlighting the diversity of scenarios where it can be deployed. In addition, all experiments highlight that encoding safety through a value function over all states, not just positional states, is required in dynamic scenarios to preserve safety and maintain performance.

[4] K. P. Wabersich, A. J. Taylor, J. J. Choi, K. Sreenath, C. J. Tomlin, A. Ames, and M. N. Zeilinger, "Data-driven safety filters: Hamilton-Jacobi reachability, Control Barrier Functions, and predictive methods for uncertain systems," *IEEE Control Systems Magazine*, vol. 43, pp. 137–177, 2023.

[5] A. D. Ames, J. W. Grizzle, and P. Tabuada, "Control barrier function based quadratic programs with application to adaptive cruise control," in *Proc. IEEE Conf. on Decision and Control*, 2014.

[6] S. Bansal, M. Chen, S. Herbert, and C. J. Tomlin, "Hamilton-Jacobi reachability: A brief overview and recent advances," in *Proc. IEEE Conf. on Decision and Control*, 2017.

[7] U. Rosolia and F. Borrelli, "Learning model predictive control for iterative tasks. a data-driven control framework," *IEEE Transactions on Automatic Control*, vol. 63, no. 7, pp. 1883–1896, 2018.

[8] A. Abate, D. Ahmed, M. Giacobbe, and A. Peruffo, "Formal synthesis of lyapunov neural networks," *IEEE Control Systems Letters*, vol. 5, no. 3, pp. 773–778, 2021.

[9] A. D. Ames, X. Xu, J. W. Grizzle, and P. Tabuada, "Control Barrier Function based quadratic programs for safety critical systems," in *IEEE Transactions on Automatic Control*, vol. 62, 2017, pp. 3861–3876.

[10] C. Dawson, Z. Qin, S. Gao, and C. Fan, "Safe nonlinear control using robust neural lyapunov-barrier functions," in *Conf. on Robot Learning*, 2021.

[11] A. Robey, H. Hu, L. Lindemann, H. Zhang, D. V. Dimarogonas, S. Tu, and N. Matni, "Learning control barrier functions from expert demonstrations," in *Proc. IEEE Conf. on Decision and Control*, 2020.

[12] T. Gurriet, M. Mote, A. Singletary, P. Nilsson, E. Feron, and A. D. Ames, "A scalable safety critical control framework for nonlinear systems," *IEEE Access*, vol. 8, pp. 187 249–187 275, 2020.

[13] 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*, vol. 50, pp. 947–957, 2005.

[14] S. L. Herbert, S. Bansal, S. Ghosh, and C. J. Tomlin, "Reachability-based safety guarantees using efficient initializations," in *Proc. IEEE Conf. on Decision and Control*, 2019.

[15] M. Chen, S. L. Herbert, M. S. Vashishtha, S. Bansal, and C. J. Tomlin, "Decomposition of reachable sets and tubes for a class of nonlinear systems," *IEEE Transactions on Automatic Control*, vol. 63, no. 11, pp. 3675–3688, 2018.

[16] S. Bansal and C. J. Tomlin, "Deepreach: A deep learning approach to high-dimensional reachability," in *Proc. IEEE Conf. on Robotics and Automation*, 2021.

[17] J. F. Fisac, N. F. Lugovoy, V. Rubies-Royo, S. Ghosh, and C. J. Tomlin, "Bridging hamilton-jacobi safety analysis and reinforcement learning," in *Proc. IEEE Conf. on Robotics and Automation*, 2019.

[18] A. Bajcsy, S. Bansal, E. Bronstein, V. Tolani, and C. J. Tomlin, "An efficient reachability-based framework for provably safe autonomous navigation in unknown environments," in *Proc. IEEE Conf. on Decision and Control*, 2019.

[19] R. Tedrake, I. R. Manchester, M. Tobenkin, and J. W. Roberts, "LQR-trees: Feedback motion planning via sums-of-squares verification," *Int. Journal of Robotics Research*, vol. 29, no. 8, pp. 1038–1052, 2010.

[20] A. Majumdar and R. Tedrake, "Funnel libraries for real-time robust feedback motion planning," *Int. Journal of Robotics Research*, vol. 36, no. 8, pp. 947–982, 2017.

[21] M. Althoff and B. H. Krogh, "Zonotope bundles for the efficient computation of reachable sets," in *Proc. IEEE Conf. on Decision and Control*, 2011.

[22] A. Majumdar, A. A. Ahmadi, and R. Tedrake, "Control design along trajectories with sums of squares programming," in *Proc. IEEE Conf. on Robotics and Automation*, 2013.

[23] S. Kousik, S. Vaskov, M. Johnson-Roberson, and R. Vasudevan, "Safe trajectory synthesis for autonomous driving in unforeseen environments," in *Proc. ASME Dynamic Systems and Control Conference*, 2017.

[24] S. Kousik, S. Vaskov, F. Bu, M. Johnson-Roberson, and R. Vasudevan, "Bridging the gap between safety and real-time performance in receding-horizon trajectory design for mobile robots," pp. 1419–1469, 2020.

[25] O. So, Z. Serlin, M. Mann, J. Gonzales, K. Rutledge, N. Roy, and C. Fan, "How to train your neural Control Barrier Function: Learning safety filters for complex input-constrained systems," in *Proc. IEEE Conf. on Robotics and Automation*, 2024.

[26] J. J. Choi, D. Lee, K. Sreenath, C. J. Tomlin, and S. L. Herbert, "Robust control barrier-value functions for safety-critical control," in *Proc. IEEE Conf. on Decision and Control*, 2021.

[27] S. Tonkens and S. Herbert, "Refining control barrier functions through hamilton-jacobi reachability," in *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, 2022.

[28] S. Tonkens, A. Toofanian, Q. Zhizhen, S. Gao, and S. Herbert, "Patching approximately safe value functions leveraging local hamilton-jacobi reachability analysis," in *Proc. IEEE Conf. on Decision and Control*, 2024.

[29] L. C. Evans and P. E. Souganidis, "Differential games and representation formulas for solutions of hamilton-jacobi-isaacs equations," *Indiana Univ. Mathematics Journal*, vol. 33, no. 5, pp. 773–797, 1984.

[30] J.-P. Aubin, A. M. Bayen, and P. Saint-Pierre, *Viability theory: new directions*. Springer Science & Business Media, 2011.

[31] F. H. Clarke, Y. S. Ledyaev, R. J. Stern, and R. Wolenski, *Nonsmooth analysis and control theory*. Springer, 1998.

[32] F. Blanchini, "Set invariance in control," *Automatica*, vol. 35, pp. 1747–1767, 1999.

[33] A. D. Ames, S. Coogan, M. Egerstedt, G. Notomista, K. Sreenath, and P. Tabuada, "Control barrier functions: Theory and applications," in *European Control Conference*, 2019.
[34] J. F. Fisac, M. Chen, C. J. Tomlin, and S. S. Sastry, "Reach-avoid problems with time-varying dynamics, targets and constraints," in *Hybrid Systems: Computation and Control*, 2015.
[35] 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*, vol. 50, no. 7, pp. 947–957, 2005.
[36] I. M. Mitchell, "The flexible, extensible and efficient toolbox of level set methods," *SIAM Journal on Scientific Computing*, vol. 35, no. 2-3, pp. 300–329, 2008.
[37] C.-W. Shu, "Essentially non-oscillatory and weighted essentially non-oscillatory schemes for hyperbolic conservation laws," in *Advanced Numerical Approximation of Nonlinear Hyperbolic Equations*. Springer, 2016, pp. 325–432.
[38] I. Fialho and T. Georgiou, "Worst case analysis of nonlinear systems," *IEEE Transactions on Automatic Control*, vol. 44, no. 6, pp. 1180–1196, 1999.
[39] J.-P. Aubin, *Viability Theory*. Birkhäuser, 1991.
[40] C. Liu, T. Arnon, C. Lazarus, C. Strong, C. Barrett, and M. J. Kochenderfer, "Algorithms for verifying deep neural networks," *Foundations and Trends in Optimization*, vol. 4, no. 3–4, pp. 244–404, 2021.
[41] M. Fazlyab, A. Robey, H. Hassani, M. Morari, and G. Pappas, "Efficient and accurate estimation of lipschitz constants for deep neural networks," in *Conf. on Neural Information Processing Systems*, 2019.
[42] A. Singletary, A. Swann, Y. Chen, and A. D. Ames, "Onboard safety guarantees for racing drones: High-speed geofencing with control barrier functions," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 2897–2904, 2022.
[43] J. A. Sethian, "Fast marching methods," *SIAM Review*, vol. 41, no. 2, pp. 199–235, 1999.
[44] Y. Li and M. Chen. (2025) Hjrno: Hamilton-jacobi reachability with neural operators. Available at https://arxiv.org/abs/2504.19989.