Patching Neural Barrier Functions Using Hamilton-Jacobi Reachability

Sander Tonkens, Alex Toofanian, Zhizhen Qin, Sicun Gao, and Sylvia Herbert

***Abstract*— Learning-based control algorithms have led to major advances in robotics at the cost of decreased safety guarantees. Recently, neural networks have also been used to characterize safety through the use of barrier functions for complex nonlinear systems. Learned barrier functions approximately encode and enforce a desired safety constraint through a value function, but do not provide any formal guarantees. In this paper, we propose a local dynamic programming (DP) based approach to "patch" an almost-safe learned barrier at potentially unsafe points in the state space. This algorithm, HJ-PATCH, obtains a novel barrier that provides formal safety guarantees, yet retains the global structure of the learned barrier. Our local DP based reachability algorithm, HJ-PATCH, updates the barrier function "minimally" at points that both (a) neighbor the barrier safety boundary and (b) do not satisfy the safety condition. We view this as a key step to bridging the gap between learning-based barrier functions and Hamilton-Jacobi reachability analysis, providing a framework for further integration of these approaches. We demonstrate that for well-trained barriers we reduce the computational load by 2 orders of magnitude with respect to standard DP-based reachability, and demonstrate scalability to a 6-dimensional system, which is at the limit of standard DP-based reachability.**

I. INTRODUCTION

Robotics has seen major advances in the past decade in several areas, including manipulation [1] and locomotion [2] in unstructured environments. Machine learning and improvements in simulation engines have been key enablers in this progress [3]. Namely, machine learning has enabled learning and representation of the complex interaction between robots and their environment implicitly. However, progress in domains where failures have major consequences has been hampered by a need for quantitative safety assurances [4]. Obtaining such strict assurances is very challenging when including learning-based components in the autonomy stack [5]. A popular method to guarantee safety for a task is through the use of value functions that encode and enforce safety, for example (control) barrier functions (CBFs) [6]. Critically, this enables enforcing safety through an independent module in the autonomy stack. Concretely, fixed-policy barrier functions (BFs) can be used to provide guaranteed maximum error bounds around a planned path to validate whether a planned path is safe *post hoc*. CBFs can be used to minimally modify a safety-agnostic or otherwise unsafe control policy, using an optimization-based safety filter, to ensure a trajectory remains safe [7].

For many realistic autonomy settings, suitable (safe, yet not too conservative) value functions cannot be derived analytically. In the following, we use value functions to denote the wide class of safety functions that encode and enforce set invariance, including BFs and CBFs. Numerical methods for synthesizing barrier functions include solving a partial differential equation (PDE) through spatial discretization [8] and sum-of-squares (SoS) techniques [9]; These both scale poorly to higher dimensional systems. A popular recent line of work proposes leveraging neural networks to learn approximate value functions [10], [11], [12]. However, these do not readily come with any formal guarantees.

The authors are with University of California, San Diego {sander, atoofanian, zhq005, sicung, sherbert}@ucsd.edu.

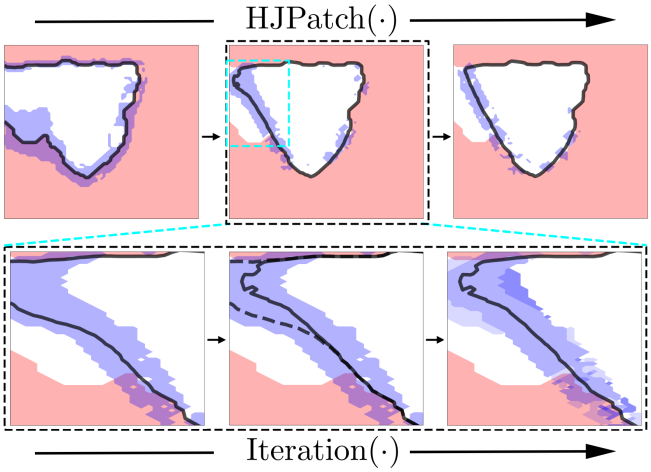

Fig. 1: HJ-PATCH iteratively (above) modifies an almost-barrier (unsafe region in red), e.g., a learned barrier, at states near the 0-level set (active set, blue) until convergence. This set represents the potentially unsafe boundary states at that iteration. For each iteration (below, zoomed), we compute the updated value (center) for the states in the active set. Next, we identify the new active set (right) by adding states that neighbor (dark blue) the value decreasing states and discard states far from the boundary (light blue). See Algorithm 1 for details.

Independently, neural network verification techniques have been developed to check whether a network satisfies certain input-output properties. We refer to [13] for an overview of verification methods. In the context of verifying safety (for the learned value functions) these methods typically employ robustness analysis [14] to provide point-wise, i.e. single time-step, guarantees [15]. However, as they rely on over-approximations or sampling, they can generally not be used to satisfy safety properties, such as forward invariance, for the entire system.

In this work, we propose taking advantage of the scalability of neural value function techniques to generate an initial high-dimensional approximate value function. We then selectively and locally refine this value function using a formal safety analysis technique that leverages spatially discretized local dynamic programming. Critically, we update only potentially unsafe regions around the value function's safety boundary. Upon convergence of our proposed iterative algorithm, we obtain a set that is guaranteed to be forward invariant and a value function that encodes and enforces safety.

Related work: Hamilton-Jacobi (HJ) reachability analysis, a formal method, can be used to compute the viability kernel, i.e. the largest feasible control invariant set for a given initial constraint set. For the infinite-horizon safety problems considered in this paper, HJ reachability uses dynamic programming (DP) to solve a value function whose 0-superlevel set denotes the viability kernel, and whose gradients prescribe the direction of safe control actions [16]. While constructive, DP methods rely on spatial discretization of the state space, and hence scale exponentially with the dimensionality of the system. In practice, direct use of this method is tractable only for systems up to 5D offline, and 2D-4D online (depending on the application) [17]. To circumvent the computational complexity (both memory and compute) of DP, learning-inspired methods have been introduced to approximately characterize the DP solution. However, directly learning HJ value functions by leveraging neural networks for supervised approximate DP does not provide hard guarantees [18]. Another approach, which leverages the similarity of HJ reachability and reinforcement learning does not necessarily encode safety [19].

CBFs are another popular approach for safe control of autonomous systems. If a valid CBF for the system and environment can be found, it can be used as a safety filter to enforce safety online via a point-wise inequality constraint on the input through the derivative of the CBF [7]. Crucially, for control-affine systems, this inequality constraint is linear in the control input, turning the safety filter optimization problem into a quadratic program (QP); this can be solved efficiently, enabling online implementation. However, for general complex systems with control bounds and disturbances, analytical closed-form functions that define a CBF do not exist or are highly non-trivial to synthesize. As such, recent work has leveraged learning-based function approximations to characterize CBFs, e.g, using safe expert trajectories [11] or labeled state-action pair samples [20], [21]. Imitation learning-inspired approaches can lead to undesired and/or overly conservative behavior in practice, whereas sampling-based approaches do not provide strict safety assurances.

For dynamics with a fixed policy, recent work has focused on learning certificate functions [22]; these are barrier functions (BFs) in the context of set invariance. Notably, recent work has paired learned BFs with a neural network verification technique to provide point-wise safety guarantees for states on the boundary of the safe set [15]. However, as they rely on sampling and do not guarantee safety of every boundary state, the BF is not guaranteed to be safe and we cannot ensure that trajectories will remain safe indefinitely.

Recently, Tonkens and Herbert [23] showed that warm-starting (i.e. initializing) DP with an approximate value function will lead to a safer value function at each iteration and converge to a guaranteed safe value function. However, this work updates the entire state-space to converge to the locally (i.e. initialization dependent) optimal value function with respect to an *a priori* known constraint function ℓ. Its 0-superlevel set corresponds to the locally largest invariant set of the constraint set $\mathcal{C}_\ell$. As it converges to the optimal value function, a good initialization only partially aids in convergence. Methods that recover the optimal value function greatly inhibit scalability.

Locally updating a DP-based value function has been proposed by Bajcsy et al. [24] in the context of updating the safe region when new sensor measurements become available. However, the method is applicable solely to expanding the safe set, and it similarly finds the locally largest invariant set (and optimal value function) for a known constraint set, again limiting scalability. In summary, while several methods have proposed techniques that reduce the computational effort to find a valid value function using an appropriate initialization, such as a learned barrier, these methods only provide limited scalability advantages and do not leverage the quality of the approximately learned value function (which often roughly characterizes an invariant set) to its full extent.

Contributions: This work proposes locally modifying an approximate learned value function to obtain a valid value function. We present an algorithm, HJ-PATCH, that is guaranteed to find the largest containing invariant set with a very limited (comparatively) number of total queries of the recursive DP-based reachability computation. Our experiments demonstrate (i) the necessity of refining learned value functions and (ii) the computational benefit (yielding $10 - 100$x speedup) of only selectively updating potentially unsafe states around the boundary of the learned barrier. Specifically, we deploy our algorithm on a 4 and 6-dimensional quadrotor model, warmstarting HJ-PATCH with the learned barrier and deploying these as a CBF within a safety filter.

Organization: We proceed with a brief overview of (control) barrier functions and learning almost-barrier functions in Section II. Next, we present HJ-PATCH, an algorithm for minimally updating approximate value functions and its theoretical guarantees in Section III. We present case studies in simulation in Section IV and conclude with the merits of our approach and avenues of future work in Section V.

II. PRELIMINARIES

We consider a dynamical system $\dot{x} = f(x, u)$, with state $x \in \mathcal{X} \subseteq \mathbb{R}^n$, input $u \in \mathcal{U} \subseteq \mathbb{R}^m$, and function $f : \mathcal{X} \times \mathcal{U} \to \mathbb{R}^n$ assumed to be Lipschitz continuous on its domain. Given a fixed continuous policy $u = \pi(x)$ that is locally Lipschitz in x, we consider the nonlinear closed-loop system $\dot{x} = f_\pi(x)$. We define system trajectories at time t starting at x_0, t_0 under control signal $u(\cdot)$ by $\xi^{u(\cdot)}_{x_0,t_0}(t)$. The Lipschitz continuity of f (and π) provides a sufficient condition on the existence and uniqueness of solutions to the system.

A set $\mathcal{C}$ with closed-loop dynamics is considered forward invariant if for any trajectory starting at $x_0 \in \mathcal{C}$, $\xi^{\pi}_{x_0,t_0}(t) \in \mathcal{C}$ indefinitely. By extension, a set $\mathcal{C}$ is considered control invariant if for any trajectory starting at x_0 there exists an admissible control signal that the keeps the system inside $\mathcal{C}$, i.e. $\exists u(\cdot) \in \mathcal{U}$ s.t. $\xi^{u(\cdot)}_{x_0,t_0}(t) \in \mathcal{C}$. The largest (control) invariant subset of a given set $\mathcal{C}$ is referred to as the viability kernel, denoted by $\bar{\mathcal{C}}$.

Additionally, we define safety using a real-valued bounded Lipschitz continuous value function $h : \mathcal{X} \to \mathbb{R}$. Common value functions include HJ reachability value functions, barrier functions, and control barrier functions. We define the value function in this work such that its 0-superlevel set

$\mathcal{C}_h = \{x \mid h(x) \geq 0\} \subseteq \mathcal{X}$ is the set of states in which we wish to stay, i.e. the safe set. For example, for avoiding a set of failure states $\mathcal{L}$, we aim to synthesize h such that (1) $\mathcal{C}_h \cap \mathcal{L} = \emptyset$ and (2) $\mathcal{C}_h$ is invariant.

A. Control Barrier Functions

In the following, we define barrier functions for fixed policy systems and control barrier functions for controlled systems.

Definition 1 (Barrier Function)**.** *Let $\mathcal{C}_h$ denote a 0-superlevel set of a continuously differentiable value function $h : \mathcal{X} \to \mathbb{R}$, then h is a barrier function for $f_\pi(x)$ if there exists an extended class $\mathcal{K}$ function α such that for all $x \in \mathcal{C}_h$:*

$$\dot{h}(x) := L_{f_\pi} h(x) \geq -\alpha(h(x)), \tag{1}$$

where L is the lie derivative. A barrier function defines a forward invariant set $\mathcal{C}_h$. It can be used to certify safety.

Definition 2 (Control Barrier Function)**.** *Let $\mathcal{C}_h$ denote a 0-superlevel set of a continuously differentiable value function $h : \mathcal{X} \to \mathbb{R}$, then h is a control barrier function for $f(x, u)$ if there exists an extended class $\mathcal{K}$ function α such that:*

$$\sup_{u \in \mathcal{U}} \dot{h}(x) := \sup_{u \in \mathcal{U}} L_f h(x) \geq -\alpha(h(x)). \tag{2}$$

A control barrier function defines a control invariant set $\mathcal{C}_h$. Specifically, any control that satisfies the inequality in (2) will result in the system remaining in the set $\mathcal{C}_h$ (indefinitely). CBFs, in contrast to BFs, are typically defined with respect to a given environment, e.g., a set of obstacles. Online, at each time step, a nominal policy is then passed through a filter that solves the following optimization problem:

$$\begin{aligned} u^*(x) = \ \arg\min_{u} \quad & \|u - \hat{\pi}(x)\|_R^2 \\ \text{subject to} \quad & \dot{h}(x) + \alpha(h(x)) \ \geq 0 \\ & u \in \mathcal{U}, \end{aligned} \tag{3}$$

where $\hat{\pi}$ is the nominal, safety-agnostic policy. The obtained control u^* is guaranteed to retain safety while minimally modifying the nominal policy.

B. Learned Almost-Barrier Functions

Learning-based methods have recently been introduced for using expressive function approximators such as neural networks to represent barrier functions to handle systems with nonlinearity and uncertainty [10], [11]. Because sampling-based learning methods are inherently approximate, the notion of almost-barrier functions have been defined. It is a relaxation of the standard notion of barrier functions by allowing small regions on the safety barrier to be uncertified. Almost-barrier functions can be parameterized by an explicit numerical tolerance bound $\varepsilon > 0$, referred to as ε-barrier functions, defined as:

Definition 1 (ε-Barrier Functions [15])**.** *Let h be a value function over the state space $\mathcal{X}$ of the system $f_\pi(x)$, and denote it's 0-level set as $\partial\mathcal{C}_h = \{x \in \mathcal{X} : h(x) = 0\}$, and let $\varepsilon \in [0, 1)$ be a parameter. Suppose there exists a measurable $\mathcal{S} \subseteq \mathcal{X}$ and the ratio of the areas between $\mathcal{S} \cap \partial\mathcal{C}_h$ and $\partial\mathcal{C}_h$ is less than ε. Namely, $A(\mathcal{S} \cap \partial\mathcal{C}_h)/A(\partial\mathcal{C}_h) \leq \varepsilon$, where $A(B) = \iint_B \mathrm{d}A$ indicates surface areas. We say h is an almost ε-barrier function with respect to safe state set $X_s \subseteq \mathcal{X}$ and unsafe state set $X_u \subseteq \mathcal{X}$, if $h(x) \geq 0$ for all $x_s \in X_s$, $h(x) < 0$ for all $x_u \in X_u$, and $L_f h(x) > 0$ for all $x \in \mathcal{S}^c \cap \partial\mathcal{C}_h$, where $\mathcal{S}^c = X \setminus \mathcal{S}$.*

In other words, an almost-barrier function satisfies the standard barrier function conditions Eq.(1) or (2), almost everywhere other than in the region of $\mathcal{S} \cap \partial\mathcal{C}_h$. Clearly, this relaxed notion of barrier functions no longer implies safety in the standard sense. There is now a "leaking" area of the 0-level set (i.e. $\mathcal{S} \cap \partial\mathcal{C}_h$) from which the system may cross over to the unsafe area X_u.

This relaxation of the original definition enables us to use sampling-based methods train a neural network representation of almost-barrier functions. Some works [25] additionally propose a falsifier accompanying the learning to find counterfactuals. More generally, sampling-based methods rely on expert policies [11], learn a policy independently [15], or jointly learn a policy with the barrier using actor-critic methods [20]. In this work, we independently learn a policy and find a neural almost-barrier and its potentially unsafe region with the 3 following steps:

1) Training control policy and sampling safe and unsafe states: We can use standard policy optimization algorithms such as soft actor-critic (SAC) [26] to learn a control policy $\pi : \mathcal{X} \to \mathcal{U}$ using a reward function that encourages goal reaching and penalizes constraint violations. After training concludes, we collect safe and unsafe samples based on whether sampled trajectories reach the goal or violate the constraints.

2) Training neural barriers: After labeling the safe and unsafe states, we train the barrier function using the following loss function (see also [15]):

$$\begin{aligned} \ell(\theta) = w_s \frac{1}{N_s} \sum_{i=0}^{N_s} \phi(-h(x_s^{(i)})) + w_u \frac{1}{N_u} \sum_{i=0}^{N_u} \phi(h(x_u^{(i)})) \\ + w_l \frac{1}{N_s} \sum_{i=0}^{N_s} \phi(-L_f h(x_s^{(i)}) - \gamma h(x_s^{(i)}))) \end{aligned} \tag{4}$$

where $\phi(x) = \max(x, 0)$ and $N_{(s,u)} = |X_{(s,u)}|$. Positive parameters w_s, w_u, w_l, γ balance the weights of the different components of the loss. $\ell(\theta)$ reaches a global minimum $\ell(\theta) = 0$ if and only if the barrier conditions in 1 are completely satisfied at each training sample.

3) Evaluating neural barrier: The evaluation process identifies "leaky" areas on the boundary through a certification process to bound the Lie derivative $L_f h(x) = \sum_{i=1}^{n} \frac{\partial h}{\partial x_i} f_i(x, u)$. For a small local neighborhood around state x, the term $\frac{\partial h}{\partial x_i}$ is bounded by using neural network analysis tools, and the term $f_i(x, u)$ is bounded by estimating the Jacobian of the dynamics through sampling the dynamics of states that are close to x in each dimension. This procedure enables us to have certifiable upper and lower bounds in the local neighborhood of x. In particular, if the lower bound is greater than 0, this neighborhood region is certifiably safe for the current value function. We denote the set of such safe neighborhoods by C.

In many cases, the control policy and neural value function may be suboptimal (i.e. potentially unsafe) in regions of the state space. Obtaining a neural barrier with no leaky areas is a challenging task. Moreover, even if we retrain the barrier with additional samples in leaky areas, the updated neural network may produce new leaky areas. Therefore, it is crucial to be able to quantify where the potential "leaky" areas are. A readily available almost-barrier with potentially unsafe regions identified motivates the need to locally patch these "leaky" areas without modifying the global structure of the function. As detailed in Section III, we propose to do this using local DP-based reachability analysis.

C. Hamilton-Jacobi Reachability

Assume we are given a set $\mathcal{C}_h \subseteq \mathcal{X}$ characterized as the 0-superlevel set of a cost function h. We define the associated HJ reachability value function:

$$V_h(x,t) = \max_{u \in \mathcal{U}_{[t,0]}} \min_{s \in [t,0]} h(\xi^u_{x,t}(s)), \tag{5}$$

which finds a control sequence $u(\cdot)$ that maximizes the minimum value of h obtained over its trajectory ξ. By definition, (as $t \to -\infty$), the 0-superlevel set of $V^*_h(x) = \lim_{t\to-\infty} V_h(x,t)$, $\mathcal{C}^*_h$, characterizes the infinite-time viability kernel of $\mathcal{C}_h$, i.e. $\bar{\mathcal{C}}_h = \mathcal{C}^*_h$ [27]. For the purposes of this work, we are only interested in persistent safety and hence drop the explicit time dependence.

In practice, we spatially discretize the state space and solve for the value function V_h recursively using dynamic programming on this grid [28]:

$$V^{(k+1)}(x) = V^{(k)}(x) + \Delta^{(k)} \min\{0, \max_{u\in\mathcal{U}} L_f V^{(k)}(x)\}, \tag{6}$$

where we use superscript (k) to denote the iteration of the algorithm. In practice, $\Delta^{(k)}$ is dynamically updated based on the size of the Hamiltonian, $\mathcal{H} = \max_{u\in\mathcal{U}} L_f V^{(k)}(x)$. By extension, for closed-loop dynamics $f_\pi(x)$, we consider $\mathcal{U} = \pi(x)$ to be a point-value.

HJ reachability can be used to generate the largest containing (control) invariant set of a set defined by some cost function. Hence, given an almost-barrier function, HJ reachability can be initialized with this cost function such that the resulting function's 0-superlevel set is invariant. In previous work [23], we presented a formulation to refine CBFs, but we considered the general case of both conservative or unsafe CBFs. In this work, we solely consider (potentially) unsafe barrier functions, as we limit ourselves to finding the largest control invariant subset for the value function. As we will present in Section III, this assumption allows for significant improvement in computational speed by leveraging local updating, which we exploit in our algorithm introduced below.

III. Locally Updating the Barrier Boundary

In this section, we present HJ-PATCH (Algorithm 1), an efficient algorithm to obtain the viability kernel of a given set by minimally and locally updating the corresponding spatially-discretized value function. Specifically, our algorithm obtains the viability kernel of the almost-barrier and associated value function by updating a very limited cumulative number of grid cells. As highlighted in Section IV, for reasonably accurate almost-barrier functions, our proposed algorithm provides significant speedups, allowing to scale formal safety analysis by up to 2 orders of magnitude. In this section we denote x^d as the discretized grid states. Additionally, we refer to the value function obtained when updating over the entire state space with Eq. 6 as the global HJ reachability value function.

Algorithm 1 HJ-PATCH

Require: $h(x)$: Initial value function (almost-barrier)
C : Oracle-certified safe cells (Optional)

// Active set is set of potentially unsafe states
$Q^{(0)} \leftarrow \{x^d \mid |h(x^d)| \le \zeta\}$ ◁ *Initialize active set*
$Q^{(0)} \leftarrow Q^{(0)} \setminus C$ ◁ *Remove oracle-certified cells*
$Q^{(k)} \leftarrow Q^{(0)}, \quad \hat{V}^{(k)} \leftarrow h$
// Boundary is certified once $Q^{(k)}$ is empty
while $Q^{(k)}$ is not empty **do**
  $\hat{V}^{(k+1)}(x^d) \leftarrow$ Eq. (6) $\forall\ x^d \in Q^{(k)}$
  // Value-decreasing states form updated active set
  $Q^{(k+1)} \leftarrow \{x^d \mid \hat{V}^{(k+1)} < \hat{V}^{(k)},\ \forall\ x^d \in Q^{(k)}\ \}$
  // Add neighboring cells to active set and exclude states far from $\hat{V}^{(k+1)}$'s 0-level set
  $Q^{(k+1)} \leftarrow \mathbf{pad}(Q^{(k+1)}) \cap \{x^d \mid |\hat{V}^{(k+1)}(x^d)| \le \zeta\}$
  $Q^{(k)} \leftarrow Q^{(k+1)}, \quad \hat{V}^{(k)} \leftarrow \hat{V}^{(k+1)}$
end while
return $\hat{V}^{(k)}$ ◁ HJ-PATCH*'s converged barrier function*

A. Algorithm overview

Leveraging the structure of infinite-time HJ reachability problems, the 0-level set of the value function obtained with our proposed approximate dynamic programming algorithm will correspond to the global HJ reachability's value function's 0-level set, see Theorem 1 for details. A visual depiction of the steps of Alg. 1 is provided in Figure 1, which we refer to in the step-by-step description below. Our algorithm takes in an almost-barrier function $h(x)$ and optionally a set of certified cells C, from e.g., neural network verification methods, see Section II-B.3. Then (1-3), we initialize the to-be-updated grid cells (active cells) as the cells around the 0-level set of the initial value function which are not certified to $Q^{(0)}$ (top left). ζ is a user-specified threshold value for the size of the boundary of interest around the 0-level set. At each iteration (5, bottom left), we compute the lie derivative (for closed-loop systems) or the Hamiltonian (for controlled systems) at these active cells $Q^{(k)}$ and update the value function using the discrete-time HJ update equation (6) (bottom center). We form the new temporary active set (6) as the states at which the values decreased. Next (7, bottom right), we pad this active set with its neighbors, and lastly discard any states far from $\hat{V}^{(k+1)}$'s 0-level set. We repeat this procedure (8) until convergence. Upon termination (10, top right), the 0-superlevel set of $\hat{V}^*$, $\hat{\mathcal{C}}^*_h$, is invariant (or empty).

B. Theoretical guarantees

The theoretical guarantees underpinning our proposed algorithm, Alg. 1, hold solely for the discretized states on the

grid and an appropriate choice of numerical solver parameters (e.g., terminal convergence threshold, finite difference, and forward integration scheme). Standard HJ reachability [17] and any other space-discretized schemes have the same properties. To highlight this, we empirically demonstrate that the obtained (control) invariant set for both standard HJ reachability and HJ-PATCH recover an almost (control) invariant set. The theory below generally holds solely at the discretized states on the grid, x^d. Solving over a discretized grid is the standard approach for HJ reachability [12], for details on discretization see [29].

We first demonstrate that Alg. 1 recovers an invariant set, if such a set exists:

Lemma 1 (Algorithm 1 converges to a control-invariant set)**.** *The 0-superlevel set of the value function $\hat{V}^*$, $\hat{\mathcal{C}}_h^*$, obtained upon termination of Alg. 1, is either (1) (control) invariant for the discretized states x^d or (2) is empty, i.e. $\hat{\mathcal{C}}_h^* = \emptyset$, implying that either (a) $\mathcal{C}_h$ has no invariant subset or (b) the current resolution of the grid does not admit an invariant subset.*

Proof. Without loss of generality, we consider $C = \emptyset$, i.e. no cells are oracle-certified. In addition to Eq. (6) we have the following equation that relates the current value function to the Lie derivative:

$$L_f \hat{V}(x) = \langle \nabla \hat{V}, f \rangle \approx \langle \delta_\Delta^p[\hat{V}], f \rangle, \tag{7}$$

with $\delta_\Delta^p[\hat{V}]$ a finite difference operator of order p, which is a function of $\hat{V}(x^d)$ and the values $\hat{V}$ of p neighbors of x^d in every dimension. We denote this set of grid states as $x^{d,p}$. Hence, for a boundary cell at iteration k, $L_f \hat{V}^{(k)}(x^d) \neq L_f \hat{V}^{(k-1)}(x^d)$ if and only if $\hat{V}^{(k-1)}(x^{d,p}) \neq \hat{V}^{(k-2)}(x^{d,p})$. Therefore, when including all p neighbors in the march algorithm, we have $L_f \hat{V}^{(k)}(x^d) \neq L_f \hat{V}^{(k-1)}(x^d)$ if and only if $x^d \in Q^{(k)}$. Hence, a boundary cell $x^d \notin Q^{(k)}$, if and only if $L_f \hat{V}^{(k)}(x^d) \geq 0$. By extension, $Q^{(k)} = \emptyset$ implies $L_{f(x,u)} \hat{V}(x^d) \geq 0 \ \forall x^d \in \partial \hat{\mathcal{C}} = \{x^d \mid \hat{V}(x^d) = 0\}$ □

Essentially, we show that when the active set is empty, its associated 0-superlevel set is invariant. In particular, we show that when including all neighbor cells that are used for calculating the spatial derivative of $V(x)$ in the march, i.e. the neighbors of active cells, the lie derivative of boundary cells $V^{(k)}(x^d)$ at iteration k can be negative if and only if x^d is part of the active set at iteration k, $x \in Q^{(k)}$.

In addition to being an invariant set, we assert that $\hat{\mathcal{C}}_h^*$ is the viability kernel $\bar{\mathcal{C}}_h$ of the candidate set $\mathcal{C}_h$. To show this, we first recall the following lemma from global warmstarting of HJ reachability [30]. We drop the argument (x) from the value functions for readability.

Lemma 2 (Optimistic global warm-start HJ reachability recovers the viability kernel [30])**.** *If we warmstart Eq.* (6) *with $V^{(0)} \geq V^* \ \forall x$, then $V^{(\infty)} = V^*$.*

In words, if HJ reachability is initialized optimistically (i.e. over-approximates the safe set), the global analysis will converge to the value function $V^*(x)$ associated with the viability kernel.

To extend this result to the local method described in this paper, we prove the theorem below.

Theorem 1 (Alg. 1 recovers the viability kernel $\bar{\mathcal{C}}_h$ of the initial set $\mathcal{C}_h$)**.** *Assume an optimistic intialization, i.e. $\hat{V}^{(0)}(x) \geq V^*(x)$. Upon convergence, the computed set $\hat{\mathcal{C}}_h^*$ from Alg. 1 recovers the viability kernel $\bar{\mathcal{C}}_h$ of the initial set $\mathcal{C}_h$ defined by the almost-barrier function h.*

Proof. Alg 1 updates the value function at only a subset of the state compared to global reachability, hence for $\hat{V}^{(0)} = V^{(0)}$, $\hat{V}^{(1)} \geq \hat{V}^{(0)}$. In particular, given an active set $Q^{(0)}$, we have $\hat{V}^{(1)}(x^d) = V^{(1)}(x^d) \ \forall \ x^d \in Q^{(0)}$ and $\hat{V}^{(1)}(x^d) \geq V^{(1)}(x^d) \ \forall \ x^d \notin Q^{(0)}$. This holds for subsequent iterations, as a single global update starting from $\hat{V}^{(1)}$ will be an upper-bound to global updating after 2 iterations, $V^{(2)}$, and henceforth. Upon convergence, $\hat{V}^* \geq V^*$, with $V^* = V^*$, by Lemma 2, so $\hat{V}^* \geq V^*$. Hence, the obtained 0-superlevel set $\hat{\mathcal{C}}_h^* \supseteq \bar{\mathcal{C}}_h$. From Lemma 1 we know that $\hat{\mathcal{C}}_h^*$ is invariant. By definition, the viability kernel $\bar{\mathcal{C}}_h$ is the largest invariant set, therefore $\hat{\mathcal{C}}_h^* = \bar{\mathcal{C}}_h$. □

In summary, the proof utilizes that we update solely on a subset of the states x^d at every iteration. We then use this to demonstrate that upon convergence the obtained value function from Alg. 1 is an upper bound to global HJ reachability, i.e. $\hat{V}^* \geq V^*$. Invoking Lemma 1, we additionally know that $\hat{\mathcal{C}}_h^*$ is invariant, hence we recover the viability kernel.

For control invariant sets, invariance by itself is not a sufficient condition to use $\hat{V}^*$ in a safety filter (Eq. 3). However, we remark that within $\hat{\mathcal{C}}^*$, $\hat{V}^*$ can be used as a safety filter.

Remark 1. *The obtained value function $\hat{V}^*(x)$, is a (potentially non-smooth) CBF for an appropriately chosen class $\mathcal{K}$ function α inside its 0-superlevel set, $\hat{\mathcal{C}}^*$. In practice, the obtained value function has relatively large gradients in the interior of the safe set. If these are problematic, we can reconstruct a signed distance function to the obtained viability kernel $\hat{\mathcal{C}}$ or consider a large threshold ζ in Alg. 1 for the size of the boundary considered for updating at each iteration.*

IV. RESULTS

We evaluate the performance of our algorithm on 2 different systems, a 4-dimensional quadrotor that ignores lateral motion and a 6-dimensional planar quadrotor. For both environments we train a neural barrier to self-right from a random initial state and consider vertical constraints, the floor and ceiling (for 4D and 6D), and additionally horizontal constraints, 2 walls (for 6D). The dynamics for the planar quadrotor models are adopted from [31]. The results include a comparison to standard HJ reachability (here considered to be ground truth), both in terms of the viability kernel obtained and the speed-up obtained from using our patching algorithm. Additionally, for both the 4D and 6D environments we provide an empirical validation of the obtained CBF on a large number of rollouts compared to the almost-neural barrier.

TABLE I: Quantitative comparison of HJ-PATCH compared to learned barrier and global reachability for the 4D quadrotor. We provide a quantitative comparison of the performance of the safety filter through random rollouts and quantify the speedup of HJ-PATCH by comparing the total number of cells that have been updated to obtain the value function. We observe that HJ-PATCH drastically reduces the rate of unsafe trajectories compared to the learned almost-CBF while being moderately more unsafe than when applying reachability on the full grid. However, applying reachability globally comes with a 13x increase in computational cost.

	Empirical safety when deployed within safety filter				Computational cost	
	Neural nominal policy		LQR nominal policy			
	Share of states visited that are unsafe [%]	Share of rollouts that are unsafe [%]	Share of states visited that are unsafe [%]	Share of rollouts that are unsafe [%]	Hamiltonians computed [#]	Share of grid labeled safe [%]
Learned almost-CBF	39.8	66.1	36.2	52.7		31.5
Warm-started global HJR	0.2	2.0	0.3	2.4	9.5e7	21.7
HJ-PATCH (Ours)	1.3	4.4	1.3	4.2	7.3e6	22.7

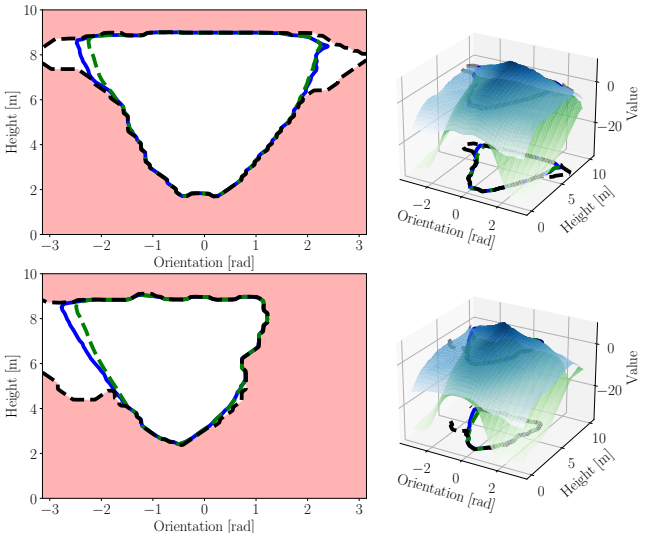

Fig. 2: 0-superlevel set (left) and value function (right) on different 2D projections, 0 velocities (top) and negative velocities (bottom). The patched value function is in blue, the global HJR value function in green, and the set associated with the original neural almost-CBF in black. The patched CBF is a good approximation of the global solution near the boundary, but has a flattened value function, corresponding to small gradients, outside the safe set.

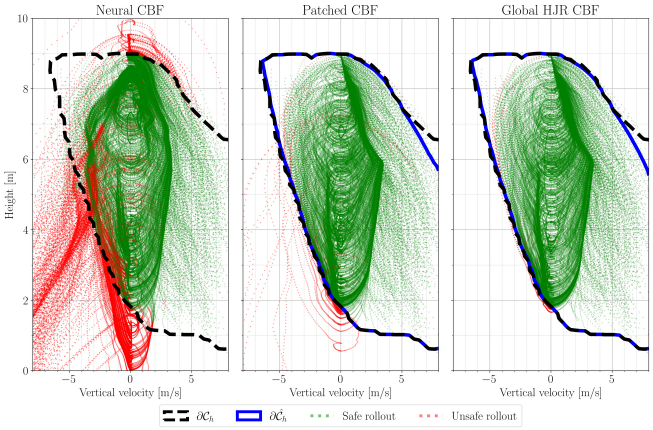

Fig. 3: Rollouts of 1000 trajectories over $10s$. The learned policy acts as the nominal controller, regulated with a CBF-based safety filter using the neural almost-CBF (left), the HJ-PATCH value function (center), and the warm-started global HJ value function (right). The large number of unsafe trajectories (red; green is safe) for the almost-CBF highlights the need for patching neural barrier functions. As detailed in Section III-B, invariance is only guaranteed at the discretized states, hence resulting in non-zero unsafe trajectories for both HJ-PATCH and the global HJ value function. Additionally, we observe that once a trajectory becomes unsafe, HJ-PATCH's value function, unlike the warm-started global HJ value function, does not provide an exponential return to safety.

We provide an open-source implementation of our approach at `https://github.com/UCSD-SASLab/HJ-Patch`, which builds wrappers around the OPTIMIZEDDP toolbox [32] and the jax-based HJREACHABILITY toolbox.

A. 4D Quadrotor: Ground truth comparison

The 4D dynamics model enables us to compare against standard HJ reachability, where all cells are updated at every iteration. As seen from Fig. 2, left, we see that for different 2D projections of the 4D viability kernel, the solution from our algorithm and global reachability match closely (although we obtain an overapproximation of the global viability kernel). As expected the value function (see Fig. 2, right) matches closely to the global solution far from the boundary and at the boundary, whereas the local solution has steeper (flatter) gradients near the boundary within (outside) the viability kernel.

In addition, we demonstrate the need for patching neural almost-barriers by rolling out trajectories ($n = 1000$, $t = 10s$) that start within the safe set and employ said almost-barrier as a CBF in a safety filter. This equally enables us to quantify the performance of the obtained value functions, both global and local, when used as CBFs in a safety filter with a nominal policy that is potentially unsafe, see Table I for a quantitative comparison. We observe that the share of unsafe trajectories of HJ-PATCH and global HJR are on the same order of magnitude, while HJ-PATCH provides a 14x computational speedup. The large amount of unsafe trajectories for the neural almost-barrier highlight the need for formal safety methods. We also visualize the trajectories in Fig. 3 in the vertical position-vertical velocity plane. We additionally observe that although the share of unsafe trajectories of HJ-PATCH and global HJR are on the same order of magnitude, HJ-PATCH's value function does not provide an exponential return to safety. This is likely due to the flattened gradients outside the safe set. Fitting a signed-distance function to the obtained 0-superlevel set can mitigate this behavior.

TABLE II: Quantitative comparison of HJ-PATCH compared to the learned barrier and global reachability for 6D planar quadrotor. We provide a quantitative comparison of the performance of the safety filter through random rollouts and quantify the speedup of HJ-PATCH by comparing the total number of cells that have been updated to obtain the value function. We observe that HJ-PATCH drastically reduces the rate of unsafe trajectories compared to the learned almost-CBF and has a lower unsafe rate than global reachability. Additionally, HJ-PATCH has an 100x lower computational cost.

	Empirical safety when deployed within safety filter				Computational cost	
	Neural nominal policy		LQR nominal policy			
	Share of states visited that are unsafe [%]	Share of rollouts that are unsafe [%]	Share of states visited that are unsafe [%]	Share of rollouts that are unsafe [%]	Hamiltonians computed [#]	Share of grid labeled safe [%]
Learned almost-CBF	0.4	2.3	38.6	57.3		4.2
Warm-started global HJR	0.7	6.1	7.0	16.2	1.6e9	2.8
HJ-PATCH (Ours)	0.0	0.5	0.1	0.7	1.6e7	3.7

B. 6D Quadrotor: Scaling reachability

To provide a valid comparison to the global solution, we consider a 6D grid with relatively large spacing between grid cells to retain computational tractability. Again, like for the 4D vertical quadrotor, we quantitatively compare HJ-PATCH to the global HJR solution and the learned almost-CBF. We observe that the learned almost-CBF requires only limited "patching" (see Table II, the share of cells labeled as safe upon convergence of HJ-PATCH overlaps with the learned barrier for $> 90\%$). Additionally, using the same convergence criteria, we see that HJ-PATCH obtains a speedup of 2 orders of magnitude. In line with our expectations, the speedup of HJ-PATCH becomes larger for higher dimensional systems, although the learned barrier for the 6-dimensional quadrotor is significantly better (i.e. almost-barrier with lower ε) than the 4-dimensional quadrotor. For the empirical safety quantification of the CBF, we observe that both global HJR and HJ-PATCH improve safety, but that HJ-PATCH has a lower failure rate (see Table II). We attribute the higher failure rate for global HJR to the limited resolution of the grid (although we use the same grid as for HJ-PATCH).

V. CONCLUSION

In this work, we propose HJ-PATCH, a constructive approach to patching neural almost-barriers using local dynamic programming. Our method guarantees convergence to a control invariant subset of the neural almost-barrier's safe set, and the resulting value function can be used within to enforce safety using, e.g., a safety filter. As highlighted in the results, HJ-PATCH enables scaling HJ reachability by 2 orders of magnitude, enabling scaling HJ reachability by $1-2$ dimensions. Additionally, we observe that we converge to a close over-approximation of the viability kernel that is obtained with global HJ reachability. Nonetheless, due to the local nature of the updating, once a trajectory leaves the safe set, our method does not provide an exponential return to safety, unlike global reachability methods.

Future work: We are interested in further speedups for locally updating value functions using more intelligent updating strategies and leveraging techniques from computational physics such as adaptive grids. Additionally, for barrier functions, the "nearest" invariant set will often be a superset of the learned barrier. We plan to extend our algorithm to converge to the smallest invariant superset of a given initial learned almost-barrier.

ACKNOWLEDGEMENTS

We would like to thank Minh Bui for assisting in the implementation of our work.

REFERENCES

[1] O. Kroemer, S. Niekum, and G. Konidaris, "A review of robot learning for manipulation: Challenges, representations, and algorithms," *Journal of Machine Learning Research*.

[2] 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*.

[3] C. K. Liu and D. Negrut, "The role of physics-based simulators in robotics," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 4, no. 1, pp. 35–58, 2021.

[4] J. García and F. Fernández, "A comprehensive survey on safe reinforcement learning," *Journal of Machine Learning Research*, vol. 16, no. 1, pp. 1437–1480, 2015.

[5] L. Brunke, M. Greeff, A. W. Hall, Y. Zhaocong, S. Zhou, J. Panerati, and A. P. Schoellig. (2021) Safe learning in robotics: From learning-based control to safe reinforcement learning. Available at https://arxiv.org/abs/2108.06266.

[6] 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.

[7] 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.

[8] 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.

[9] A. A. Ahmadi and A. Majumdar, "Some applications of polynomial optimization in operations research and real-time decision making," *Optimization Letters*, vol. 20, no. 4, pp. 709–729, 2016.

[10] C. Dawson, S. Gao, and C. Fan, "Safe control with learned certificates: A survey of neural lyapunov, barrier, and contraction methods for robotics and control," pp. 1–19, 2023.

[11] L. Lindemann, A. Robey, L. Jiang, S. Tu, and N. Matni, "Learning robust output control barrier functions from safe expert demonstrations," 2021, available at https://arxiv.org/abs/2111.09971.

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

[13] 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.

[14] H. Zhang, T.-W. Weng, P.-Y. Chen, C.-J. Hsieh, and L. Daniel, "Efficient neural network robustness certification with general activation functions," in *Conf. on Neural Information Processing Systems*, 2018.

[15] Z. Qin, T.-W. Weng, and S. Gao, "Quantifying safety of learning-based self-driving control using almost-barrier functions," in *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, 2022.

[16] 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.

[17] 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.

[18] A. Lin and S. Bansal. (2022) Generating formal safety assurances for high-dimensional reachability. Available at https://arxiv.org/abs/2209.12336.

[19] K.-C. Hsu, V. Rubies-Royo, C. J. Tomlin, and J. F. Fisac, "Safety and liveness guarantees through reach-avoid reinforcement learning," in *Robotics: Science and Systems*, 2021.

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

[21] S. Liu, C. Liu, and J. Dolan, "Safe control under input limits with neural control barrier functions," in *Conf. on Robot Learning*, 2022.

[22] S. M. Richards, F. Berkenkamp, and A. Krause, "The Lyapunov neural network: Adaptive stability certification for safe learning of dynamical systems," in *Conf. on Robot Learning*, 2018.

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

[24] 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.

[25] Y.-C. Chang, N. Roohi, and S. Gao, "Neural Lyapunov control," in *Conf. on Neural Information Processing Systems*, 2019.

[26] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Int. Conf. on Machine Learning*, 2018.

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

[28] S. Herbert, "Safe real-world autonomy in uncertain and unstructured environments," Ph.D. dissertation, Dept. of EECS, Univ. of California, Berkeley, 2020.

[29] J. C. Strikwerda, *Finite Difference schemes and partial differential equations*. SIAM, 2004.

[30] 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.

[31] S. Singh, S. M. Richards, V. Sindhwani, J.-J. E. Slotine, and M. Pavone, "Learning stabilizable nonlinear dynamics with contraction-based regularization," *Int. Journal of Robotics Research*, 2020.

[32] M. Bui, G. Giovanis, M. Chen, and A. Shriraman. (2022) Optimizeddp: An efficient, user-friendly library for optimal control and dynamic programming. Available at https://arxiv.org/abs/2204.05520.