On Safety and Liveness Filtering Using Hamilton-Jacobi Reachability Analysis

Javier Borquez, Kaustav Chakraborty, Hao Wang, Somil Bansal

***Abstract*—Hamilton-Jacobi (HJ) reachability-based filtering provides a powerful framework to co-optimize performance and safety (or liveness) for autonomous systems. Under this filtering scheme, a nominal controller is minimally modified to ensure system safety or liveness. However, the resulting controllers can exhibit abrupt switching and bang-bang behavior, which is not suitable for applications of autonomous systems in the real world. This work presents a novel, unifying framework to design safety and liveness filters through reachability analysis. We explicitly characterize the maximal set of control inputs that ensures safety (or liveness) at a given state. Different safety filters can then be constructed using different subsets of this maximal set along with a projection operator to modify the nominal controller. We use the proposed framework to design three safety filters, each balancing performance, computation time, and smoothness differently. We highlight their relative strengths and limitations by applying these filters to autonomous navigation and rocket landing scenarios and on a physical robot testbed. We also discuss practical aspects associated with implementing these filters on real-world autonomous systems. Our research advances the understanding and potential application of reachability-based controllers on real-world autonomous systems.**

***Index Terms*—Robot Safety, Reachability Analysis, Safety Filtering.**

I. Introduction

As autonomous systems become integral to our daily lives, it is crucial to design controllers that are both safe and performant. Typically, controller design is framed as an optimal control problem, balancing performance criteria (e.g., minimizing control energy), with safety constraints, such as obstacle avoidance. The resultant constrained optimal control problem is solved using a variety of methods ranging from Model Predictive Control (MPC) [26], [12], [7] and (approximate) dynamic programming [5], [6] to data-driven, reinforcement learning approaches [15], [24]. Despite impressive performance of these methods, ensuring hard safety constraints for general, nonlinear systems remains a challenge, especially if the optimal control problem needs to be solved online.

An alternative approach is *safety filtering*, where a nominal control (often performance-oriented) is minimally adjusted (i.e., "filtered") to satisfy the safety constraint. If the input is safe, it is used directly; otherwise, it is projected to a set of safe control inputs. This guarantees safety while optimizing performance whenever the system safety is not at risk. Akin to safety filters, one can also design *liveness filters*, which ensure that the system reaches a desired set of states (e.g., the landing pad for a rocket) within a set timeframe while optimizing the performance (e.g., minimizing the control energy).

*Authors are with the ECE department at the University of Southern California. This research is supported in part by the DARPA ANSR program, the NSF CAREER program (award number 2240163), and Universidad de Santiago de Chile.

A number of approaches have been proposed in literature to construct safety and liveness filters for dynamical systems, such as Control Barrier and Control Lyapunov functions, Hamilton-Jacobi reachability, and MPC. We discuss some of the relevant approaches here and refer the interested readers to [14] and [25] for a detailed survey on filtering methods.

Hamilton-Jacobi (HJ) reachability analysis [18], [20] is a popular mechanism to design these filters due to its ability to ensure safety and liveness for dynamical systems with general nonlinear dynamics, control bounds, and disturbances [19], [3], [4]. In reachability analysis, one is interested in computing the *backward reachable tube (BRT)* of a dynamical system, i.e. the set of all initial states from which the system will eventually reach a target set, despite the worst case disturbance. Thus, if the target set represents the set of desirable states, the BRT represents the set of states from which liveness can be guaranteed. Conversely, if the target set represents the set of undesirable states, the BRT contains states which are potentially unsafe for the system and should be avoided. Alongside the BRT, reachability analysis provides a safety controller (respectively liveness controller) that will provably steer the system away from the target set (respectively towards the target set). A simple safety filter uses the BRT and safety controller: the system employs the nominal controller outside the BRT and switches to the reachability safety controller at the BRT boundary, ensuring constant safety. However, controllers from HJ reachability are often extremum seeking, resulting in a bang-bang behavior. This behavior along with a sudden switching between the nominal and reachability controllers leads to jittery state and control trajectories, which is often undesirable for real-world autonomous systems [25].

An alternative approach to filtering uses Control Barrier Functions (CBF) for safety and Control Lyapunov Functions (CLF) for liveness filtering. Both utilize Lyapunov-like conditions to ensure the forward invariance of a set [1], which then aids in creating a quadratic program (QP) for a smooth blending of nominal and safety (or liveness) controllers. However, despite recent progress [10], [23], [22], [28], constructing a valid Lyapunov or barrier function for general nonlinear systems with control bounds and disturbances remains a challenge. Some recent studies have addressed this by framing CBF synthesis as an HJ reachability problem, capitalizing on the constructive attributes of reachability methods for obtaining a valid CBF [10]. The resultant CBF can be used for smooth, QP-based controller synthesis. However, a few questions remain: can we derive smooth control laws directly from reachability analysis without constructing a CBF first? How do these safety filters compare in performance and computation? And can we design such control laws for liveness filtering?

In this work, we answer the aforementioned questions in affirmative and introduce a general framework to design safety and liveness filters using reachability analysis. Under the proposed framework, any safety and liveness filter can be thought of as a combination of two components: (a) a *projection set* – a set of safe or live control inputs that the nominal controller will be projected to; and (b) a *projection operator* to align a nominal control input with this set. Drawing on this insight, our key idea is to use HJ reachability analysis to explicitly characterize the set of *all* possible control inputs at a state that guarantees system safety or liveness. Various safety filters can then be obtained by using different subsets of this maximal safe control set as the projection set, along with a projection operator, to mathematically characterize the proposed framework we will use the notation presented in Table I. Building upon this approach, we propose three different safety and liveness filters based on reachability analysis: a least restrictive safety filter, a smooth least restrictive filter, and a smooth blending filter. The proposed filters are versatile, applicable to both liveness and safety, accommodating control bounds, adversarial disturbances, model uncertainties, and time-specific liveness and safety properties (e.g., reach the target set within 5 seconds). We further compare these filters on various metrics including performance, computation time, controller tuning, and control profile smoothness, as well as a comparison to CBF methods for the safety filter. Beyond these specific filters, our approach provides a unifying framework for existing reachability-based filters and designing new ones using diverse subsets of the maximal safe set and projection operators. We showcase our approach in two different applications inspired by rocket landing within a given landing window and safe autonomous blimp navigation, as well as on a physical robot testbed involving a wheeled robot navigating through a cluttered environment.

Symbol	Definition
$n \in \mathbb{Z}_+$	Number of states
$\mathcal{X} \subseteq \mathbb{R}^n$	State space
$\mathbf{x} \in \mathcal{X}$	System state
$\dot{\mathbf{x}}$	Time derivative of state $\mathbf{x}$
$\mathcal{U}$	Control space
$u \in \mathcal{U}$	Control input
$\mathcal{D}$	Disturbance space
$d \in \mathcal{D}$	Disturbance input
f	System dynamics
$\mathbf{u}(\cdot)$	Control signal (over time)
$\mathbf{d}(\cdot)$	Disturbance signal (over time)
$\xi^{\mathbf{u},\mathbf{d}}_{\mathbf{x},t}(\tau)$	State at time τ starting from state $\mathbf{x}$ at time t under control and disturbance signals $\mathbf{u}(\cdot)$ and $\mathbf{d}(\cdot)$
$\mathcal{L} \subseteq \mathcal{X}$	Target set. Could represent goal or failure states.
$l : \mathcal{X} \to \mathbb{R}$	Target function. Signed distance to the target set.
T	Time horizon.
π_{nom}	Nominal controller
$V(\mathbf{x}, t)$	Value function at state $\mathbf{x}$ and time t
$D_t V(\mathbf{x}, t)$	Time derivative of the value function
$\nabla V(\mathbf{x}, t)$	Spatial derivative of the value function
$\mathcal{G}_{\text{live}}(t)$	BRT of the target set in the liveness problem.
$\pi^*_{\text{live}}(\mathbf{x}, t)$	Default liveness controller from the reachability analysis
$\mathcal{U}_{\text{live}}(\mathbf{x}, t)$	Set of liveness preserving controls at state $\mathbf{x}$ and time t.
h	Projection operator for filtering
$\widetilde{\mathcal{U}}$	Projection set for filtering
γ	Constant used in the smooth blending filter.
$\mathcal{G}_{\text{unsafe}}$	BRT of the target set in the safety problem.
$\pi^*_{\text{safe}}(\mathbf{x})$	Default safety controller from the reachability analysis.
$\mathcal{U}_{\text{safe}}(\mathbf{x})$	Set of safety preserving controls.

TABLE I. A summary of all the symbols used in the paper.

II. Problem Statement

Consider an autonomous system with state $\mathbf{x} \in \mathcal{X} \subseteq \mathbb{R}^n$ that evolves according to dynamics

$$\dot{\mathbf{x}} = f(\mathbf{x}, u, d) \tag{1}$$

where $u \in \mathcal{U}$ and $d \in \mathcal{D}$ are the control and disturbance of the system, respectively. d can represent potential model uncertainties or an actual, adversarial exogenous input to the system. We assume the dynamics are uniformly continuous in u and d, bounded, and Lipschitz continuous in $\mathbf{x}$ for fixed u and d. Finally, let $\xi^{\mathbf{u},\mathbf{d}}_{\mathbf{x},t}(\tau)$ denote the system state at time τ, starting from the state $\mathbf{x}$ at time t under control signal $\mathbf{u}(\cdot)$ and disturbance signal $\mathbf{d}(\cdot)$ while following the dynamics (1). A control signal $\mathbf{u}(\cdot)$ is defined as a measurable function mapping from the time horizon to the set of admissible controls $\mathcal{U}$, and a disturbance signal is similarly defined. We additionally assume that the control and disturbance signals $\mathbf{u}(\cdot)$ and $\mathbf{d}(\cdot)$ are piecewise continuous in t. This assumption ensures that the system trajectory $\xi^{\mathbf{u},\mathbf{d}}_{\mathbf{x},t}$ exists and is unique and continuous for all initial states [11], [8].

In this work, we are interested in synthesizing potentially time-varying controllers $\pi : [0, T] \times \mathcal{X} \to \mathcal{U}$ that steer the system to reach (liveness) or avoid (safety) a given target set $\mathcal{L} \subseteq \mathcal{X}$, within the time horizon $[0, T]$. $\mathcal{L}$ can represent the goal region in the case of liveness or an unsafe region of the state space in the case of safety, e.g., obstacles for a navigation robot. Furthermore, we would like our controllers to consider other performance objectives while guaranteeing liveness/safety. We assume the performance objectives are encoded and optimized by an user-defined nominal controller π_{nom}. However, π_{nom} may not necessarily ensure liveness (or safety). *Our goal is to design π to follow π_{nom} to the extent possible, while guaranteeing liveness/safety requirement.*

One popular mechanism to achieve this goal is *liveness or safety filtering*, in which π_{nom} is modified minimally to ensure system liveness or safety [2], [3]. In this paper, we will leverage reachability analysis to synthesize such filters.

Running Example (Liveness). We use a rocket landing system as a running example throughout this paper to demonstrate the liveness properties of our controllers. The rocket is modeled as a 6D system with dynamics,

$$f_{\text{rocket}}(\mathbf{x}, u) = \frac{d}{dt}\begin{bmatrix} y \\ z \\ \theta \\ \dot{y} \\ \dot{z} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \dot{y} \\ \dot{z} \\ \dot{\theta} \\ \cos(\theta)u_y - \sin(\theta)u_z \\ \sin(\theta)u_y + \cos(\theta)u_z - g \\ \alpha u_1 \end{bmatrix} \tag{2}$$

Here y, z denotes the horizontal and vertical positions of the center of mass of the rocket, θ denotes the rocket's heading, u_y and u_z denote the thrust in the y and z direction, respectively, and g is the acceleration due to the gravity of Earth. The task of the liveness controller is to generate thrust inputs $u = [|u_y| < 250, |u_z| < 250]$ such that

the system can reach a landing pad (target set) within one second. The target set is defined as the rectangular area $\mathcal{L} = \{(y, z) : |y| < 20, 0 < z < 20\}$. The corresponding target function is given by $l(\mathbf{x}) = \max\{|y| - 20, z - 20, -z\}$, which approximates the signed distance to the target set. For this example, we assume no external disturbances are acting on the system.

We first synthesize a nominal controller that attempts to take the system to $\mathcal{L}$. The nominal controller is a sampling-based model predictive controller (MPC), whose objective is to minimize the distance between center of mass position and the boundary of $\mathcal{L}$, and the control energy over the entire trajectory. Intuitively, the nominal controller can be considered as a performance control that tends to drive the system state to $\mathcal{L}$ while optimizing the amount of control effort. The MPC cost to be minimized over the entire trajectory is given as,

$$\min_{u} \quad \sum_{i=1}^{t} || \, [u_{y_i}, u_{z_i}] \, ||_2 + \sqrt{(|y_i| - 20)^2 + (|z_i| - 20)^2} \tag{3}$$

Here, the first term penalizes the control cost while the second term penalizes the distance from the target set. i is the discrete timestep and $||.||_2$ is the L_2 norm. The MPC controller is implemented in a receding horizon fashion with a horizon of 1000 steps. We use a first-order Euler discretization of the continuous dynamics in (2) with timestep, δ = 0.0005s. In our simulation studies, we will show that, in this case, such a nominal controller is insufficient to assure the liveness of our system. Liveness might be achievable using MPC, but designing such a controller would require more care and time. In this work, we will use a liveness filter to refine this nominal controller and produce a controller with the desired liveness properties.

III. Background: Hamilton-Jacobi Reachability

Hamilton-Jacobi (HJ) reachability analysis is a formal verification technique that characterizes the set of states from which the liveness (or safety) constraints can be satisfied with some control under the worst case disturbance. In our work, we will leverage this technique to synthesize liveness (and safety) filters for the nominal controller. In this section, we provide a brief overview of HJ reachability analysis.

A. Liveness and Safety Problems in HJ Reachability

In HJ reachability analysis, the liveness and safety problems are posed as optimal control problems; informally, both problems intend to find a control signal $\mathbf{u}(\cdot)$ that steers the system as "deep" into or as "far away" from the target set as possible. To capture this semantic, the minimum distance between the system and the target set $\mathcal{L}$ over the time horizon is defined as the objective of the optimal control problems. Let $l : \mathcal{X} \to \mathbb{R}$ be some bounded and Lipschitz continuous function whose sub-zero level is given by the target set: $\mathcal{L} = \{\mathbf{x} : l(\mathbf{x}) \leq 0\}$. Here, we present the reachability analysis for the liveness case and then comment on the safety case. Given l, the liveness problem is defined in (4):

$$\begin{aligned} \max_{\mathbf{d}(\cdot)} \min_{\mathbf{u}(\cdot)} \min_{\tau \in [t,T]} & \; l(\xi^{\mathbf{u},\mathbf{d}}_{\mathbf{x},t}(\tau)) \\ s.t. \quad & \dot{\mathbf{x}} = f(\mathbf{x}, u, d) \end{aligned} \tag{4}$$

where the minimum cost over time for the trajectory is defined as the cost function $J(\mathbf{x}, t, \mathbf{u}(\cdot), \mathbf{d}(\cdot)) = \min_{\tau \in [t,T]} l(\xi^{\mathbf{u},\mathbf{d}}_{\mathbf{x},t}(\tau))$. The above optimization problem (4) finds the minimum distance to the target set over the system trajectory, under the optimal control and the worst case disturbance. Thus, the system can reach the target set if and only if the minimum distance is less than 0, and this minimum distance is captured by the value function:

$$V(\mathbf{x}, t) = \max_{\mathbf{d}(\cdot)} \min_{\mathbf{u}(\cdot)} J(\mathbf{x}, t, \mathbf{u}(\cdot), \mathbf{d}(\cdot)) \tag{5}$$

The value function (5) can be computed using dynamic programming, and it satisfies the following final value Hamilton-Jacobi-Isaacs Variational Inequality (HJI-VI) [20], [17]:

$$\begin{aligned} \min\{D_t V(\mathbf{x}, t) + H(\mathbf{x}, t, \nabla V(\mathbf{x}, t)), l(\mathbf{x}) - V(\mathbf{x}, t)\} = 0 \\ \text{for } t \in [0, T] \text{ and } V(\mathbf{x}, T) = l(\mathbf{x}) \end{aligned} \tag{6}$$

Here, $D_t V(\mathbf{x}, t)$ and $\nabla V(\mathbf{x}, t)$ denote the temporal derivative and the spatial gradients of the value function $V(\mathbf{x}, t)$, respectively. The Hamiltonian encodes how the control and disturbance interact with the system dynamics and is given as:

$$H(\mathbf{x}, t, \nabla V(\mathbf{x}, t)) = \min_{u \in \mathcal{U}} \max_{d \in \mathcal{D}} \nabla V(\mathbf{x}, t) \cdot f(\mathbf{x}, u, d) \tag{7}$$

Given the value function, one can also obtain the Backward Reachable Tube $\mathcal{G}_{\text{live}}(t)$ of the system. Since $\mathcal{G}_{\text{live}}(t)$ is defined to be the set of initial states from which the system can reach the target set within the time horizon $(T - t)$ despite worst-case disturbance, $\mathcal{G}_{\text{live}}(t)$ is the sub-zero level set of the value function:

$$\mathcal{G}_{\text{live}}(t) \coloneqq \{\mathbf{x} : V(\mathbf{x}, t) \leq 0\} \tag{8}$$

Consequently, liveness is guaranteed as long as the system state is inside $\mathcal{G}_{\text{live}}(t)$. Conversely, if the system state is outside $\mathcal{G}_{\text{live}}(t)$, the liveness can't be ensured despite the best control effort. Thus, any live controller must maintain the system state within $\mathcal{G}_{\text{live}}(t)$ at all time t.

The HJ reachability analysis for the safety problem is similar to that for the liveness problem, except that the control tries to avoid entering $\mathcal{L}$. Thus, the role of control and disturbance is switched in (4) and (7). The sub-zero level of the value function gives us the BRT $\mathcal{G}_{\text{unsafe}}$, which represents the set of all states from which the system is guaranteed to enter the target set (the unsafe region in this case), despite the best control effort. Conversely, if the system state is outside the $\mathcal{G}_{\text{unsafe}}$, there exists a controller that will keep the system safe, despite worst-case disturbance.

B. Default Liveness (Safety) Controller from HJ Reachability

Along with the value function and the BRTs, reachability analysis also provides a liveness/safety controller for the system (referred to as the *default liveness/safety reachability*

controller from here on). At state $\mathbf{x}$ and time t, the default liveness reachability controller is given as:

$$\pi^*_{\text{live}}(\mathbf{x}, t) = \arg\min_{u\in\mathcal{U}} \max_{d\in\mathcal{D}} \nabla V(\mathbf{x}, t) \cdot f(\mathbf{x}, u, d) \tag{9}$$

Similarly, the optimal disturbance is given as:

$$d^*(\mathbf{x}, t) = \min_{u\in\mathcal{U}} \arg\max_{d\in\mathcal{D}} \nabla V(\mathbf{x}, t) \cdot f(\mathbf{x}, u, d) \tag{10}$$

Intuitively, at any time step, $\pi^*_{\text{live}}(\mathbf{x}, t)$ tries to *maximally* steer the system state towards the target set (we omit the state and time dependencies of the optimal controller and disturbance onwards for compactness). To understand this, recall that the value function for the liveness problem (4), $V(\mathbf{x}, t)$ represents the minimum cost the system can achieve over the time horizon $[t, T]$, starting from state $\mathbf{x}$ and time t under worst case disturbance. In the liveness problem, we would like the system to reach a state of lower value. π^*_{live} seeks to accomplish this objective by finding a control which steers the system in the direction of greatest decent on the value function (i.e. minimizing the dot product between $\dot{\mathbf{x}}$ and the spatial gradient of the value function ∇V). Moreover, it can be shown that as long as the system starts inside $\mathcal{G}_{\text{live}}(t)$, it is guaranteed to eventually reach the target set under π^*_{live} despite the worst-case disturbance [3]. This is analogous to the safety case, with the difference that the controller would try to steer the system away from the target set.

Even though the default reachability controllers respect the liveness/safety constraints, they do not consider other objectives of interests, such as minimizing control energy or tracking some nominal controller. Thus, a natural solution is to blend the default reachability controller with a nominal controller that takes these performance criterion into consideration while respecting safety/liveness constraints, which is the focus of this paper.

Running Example (Liveness). As discussed earlier, in the liveness problem, we are interested in computing $\mathcal{G}_{\text{live}}(t)$, along with associated value function. This requires solving the HJI-VI (6). Traditional methods compute a numerical solution of the HJI-VI over a state space grid [19]; however, these methods suffer from the curse of dimensionality and are not suitable for a 6D system. Instead, we use DeepReach [4] to compute the value function. Rather than solving the HJI-VI over a grid, DeepReach represents the value function as a sinusoidal neural network and learn a parameterized approximation of the value function. Thus, memory and complexity requirements for training scale with the value function complexity rather than the grid resolution. To train the neural network, DeepReach uses self-supervision on the HJI-VI itself. Ultimately, it takes as input a state $\mathbf{x}$ and time t, and it outputs the value $V_\beta(\mathbf{x}, t)$, where β are the parameters of the NN. We refer interested readers to [4] for further details.

For this system, we use a sinusoidal neural network with three hidden layers, and 512 neurons per layer. The training took approximately two hours for 20000 epochs on an NVIDIA RTX 4090 GPU. Using scenario optimization [16], we verify the trained neural network by computing a high confidence error bound over the accuracy of the learned value function. A slice of the obtained BRT projected over the YZ-plane is shown in Fig. 1. As long as the system starts inside the teal region, it is guaranteed to reach the pink region (landing pad) under π^*_{live}. The system trajectory under π^*_{live} for one such starting state is shown in dark blue in Fig. 1(a) and the corresponding control profiles are shown in Fig. 1(b), (c).

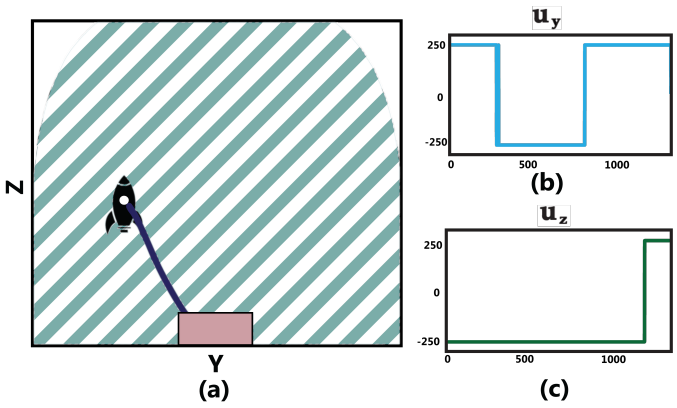


Fig. 1. (a) BRT slice for a rocket landing problem in the (y, z) states (all other states = 0) shown in teal. The target set $\mathcal{L}$ is shown in light pink rectangle. The trajectory followed by the rocket from its initial position (shown with the black rocket icon) under the default controller is shown in dark blue. (b) u_y and (c) u_z controller profiles generated by the default reachability controller displaying bang-bang behavior.

As expected, the rocket eventually reaches the landing pad. However, the default liveness controller only chooses the maximal control authority to steer the system (in this case, $|u_y| = |u_z| = 250$). This is a typical bang-bang nature of the reachability controllers for systems with control-affine dynamics and independently bounded control inputs, that ensures that the value function gradient is always pointing towards the direction of maximum decent.

However, as we will show in later sections, this approach results in jittery, high-energy control profiles that do not take into account any performance criteria, which results in a very high total cost for this control.

IV. Liveness Filtering Using HJ Reachability

The default controller obtained from the reachability analysis ensures liveness, but it does not consider other performance objectives. In this section, we first use reachability analysis to characterize the set of *all* liveness-ensuring control inputs at a particular state. We then use this set to design different liveness filters.

A. Characterizing the set of live controls

At state $\mathbf{x}$ and time t, the set of all liveness-ensuring control inputs (abbreviated as *live controls* for the remainder of the paper) is given as:

$$\mathcal{U}_{\text{live}}(\mathbf{x}, t) = \{u : \exists \epsilon > 0, \\ V\left(\xi^{u,d^*}_{\mathbf{x},t}(t+\delta), t+\delta\right) \leq 0 \quad \forall \delta \in [0, \epsilon]\} \tag{11}$$

Here, V represents the value function obtained using the HJI-VI in (6). d^* is the optimal disturbance given by (10). Intuitively, $\mathcal{U}_{\text{live}}(\mathbf{x}, t)$ (11) is the set of all controls that instantaneously keep the next state of the system inside (or

at the boundary of) $\mathcal{G}_{\text{live}}$, despite the worst case disturbance d^*. This ensures that the system can still reach the target set from the next state, thereby ensuring liveness recursively.

We will next use (11) to characterize $\mathcal{U}_{\text{live}}(\mathbf{x}, t)$. We assume that $\mathbf{x} \notin \mathcal{L}$; otherwise, the system state is already within the target set and no further control action is needed.

Lemma 1: Assume that the value function $V(\mathbf{x}, t)$ is differentiable for all $\mathbf{x}$ and t. Let $\mathbf{x}$ be any state such that $\mathbf{x} \notin \mathcal{L}$ and $d^* \in \mathcal{D}$ the optimal disturbance that maximally decreases liveness. Then the set of live controls at $\mathbf{x}$ is given as:

$$\mathcal{U}_{\text{live}}(\mathbf{x}, t) = \begin{cases} \mathcal{U} & \text{if } V(\mathbf{x}, t) < 0 \\ \{u \in \mathcal{U} : \\ D_t V(\mathbf{x}, t) + \nabla V(\mathbf{x}, t) \cdot f(\mathbf{x}, u, d^*) = 0\} \\ \qquad\qquad \text{if } V(\mathbf{x}, t) = 0 \\ \emptyset & \text{if } V(\mathbf{x}, t) > 0 \end{cases} \tag{12}$$

The proof of Lemma 1 is in the Appendix. Intuitively, (12) states that $\mathcal{U}_{\text{live}}(\mathbf{x}, t)$ is the set of all admissible controls $\mathcal{U}$ when the system state is in the interior of $\mathcal{G}_{\text{live}}$. This makes sense as the system can apply any control and instantaneously remain inside $\mathcal{G}_{\text{live}}$ for a sufficiently small time step. On the other hand, when the system state is outside $\mathcal{G}_{\text{live}}$, liveness can't be ensured despite the best control effort, resulting in an empty set. Finally, when the system state is at the boundary of $\mathcal{G}_{\text{live}}$, $\mathcal{U}_{\text{live}}(\mathbf{x}, t)$ is given by the controls that ensure the *total* derivative of the value function, $\frac{d}{dt}V(\mathbf{x}, t)$, to be 0 (i.e., the control inputs that keep the system state instantaneously at the boundary of $\mathcal{G}_{\text{live}}$).

There are a few interesting observations to be made about the set of live controls in (12):

1) Lemma 1 provides a time-dependent characterization of the live control inputs, allowing us to capture time-constrained liveness properties (e.g., reach the target set within T seconds), as opposed to just asymptotic liveness.
2) For control and disturbance-affine system dynamics, the condition of $\frac{d}{dt}V(\mathbf{x}, t) = 0$ is linear in u. To see this, suppose the system dynamics are given by $f(\mathbf{x}, u, d) = f_1(\mathbf{x}) + f_2(\mathbf{x})u + f_3(\mathbf{x})d$. Its total derivative $\frac{d}{dt}V(\mathbf{x}, t)$, under the optimal disturbance d^*, is given by:

   $$\begin{aligned} &D_t V(\mathbf{x}, t) + \nabla V(\mathbf{x}, t) \cdot f(\mathbf{x}, u, d^*) \\ &= D_t V(\mathbf{x}, t) + \nabla V(\mathbf{x}, t) \cdot (f_1(\mathbf{x}) + f_2(\mathbf{x})u + f_3(\mathbf{x})d^*) \\ &= \alpha + \beta u \end{aligned}$$

   We can observe that $\frac{d}{dt}V(\mathbf{x}, t)$ is now a linear equation of u, where $\alpha = D_t V(\mathbf{x}, t) + \nabla V(\mathbf{x}, t) \cdot (f_1(\mathbf{x}) + f_3(\mathbf{x})d^*)$ and $\beta = \nabla V(\mathbf{x}, t) \cdot f_2(\mathbf{x})$. Thus, $\mathcal{U}_{\text{live}}(\mathbf{x}, t)$ is given by a hyperplane in $\mathcal{U}$ (a point if the control space is one-dimensional), whenever the system state is at the boundary of $\mathcal{G}_{\text{live}}(t)$. As we will see later, this will make the liveness filter design computationally efficient for control-affine systems.
3) Control inputs given by the default liveness controller (9) is always contained within $\mathcal{U}_{\text{live}}(\mathbf{x}, t)$. The following corollary formalizes this result.

   Corollary 1: $\pi^*_{\text{live}}(\mathbf{x}, t) \in \mathcal{U}_{\text{live}}(\mathbf{x}, t)\ \forall \mathbf{x} \in \mathcal{G}_{\text{live}}$.
   Thus, the default controller can be used to steer the system to the target set, consistent with our expectation.

The real utility of (12) is that it can potentially provide more than one liveness-ensuring control inputs, whereas the default liveness controller can only provide one at a given state $\mathbf{x}$ and time t. We now formally establish that applying any control inputs from $\mathcal{U}_{\text{live}}(\mathbf{x}, t)$ is sufficient to ensure that the system will eventually reach the target set within the established time horizon $[t, T]$.

Lemma 2: Suppose the system starts inside $\mathcal{G}_{\text{live}}$ at time t. If the system applies control $u(\tau) \in \mathcal{U}_{\text{live}}(\mathbf{x}(\tau), \tau)\ \forall \tau \in [t, T]$, then $\exists s \in [t, T]$ such that $\mathbf{x}(s) \in \mathcal{L}$.

Lemma 2 effectively allows us to use any subset of $\mathcal{U}_{\text{live}}(\mathbf{x}, t)$ for the projection of the nominal controller, while still maintaining liveness. This suggests the following general structure for liveness filters:

General Liveness Filter. Given a nominal controller, π_{nom}, a general liveness filter can be formulated as:

$$\begin{aligned} \pi(\mathbf{x}, t) = \arg\min_{u} \quad & h(u, \pi_{\text{nom}}(\mathbf{x}, t)) \\ & s.t.\ u \in \widetilde{\mathcal{U}}(\mathbf{x}, t), \text{ with } \widetilde{\mathcal{U}}(\mathbf{x}, t) \subseteq \mathcal{U}_{\text{live}}(\mathbf{x}, t) \end{aligned} \tag{13}$$

In (13), we refer to h as the *projection operator* and $\widetilde{\mathcal{U}}$ as the *projection set*. Since $\widetilde{\mathcal{U}} \subseteq \mathcal{U}_{\text{live}}$, the above filter makes sure that $\pi(\mathbf{x}, t) \in \mathcal{U}_{\text{live}}(\mathbf{x}, t)$, thereby ensuring system liveness at all times (by Lemma 2). Nevertheless, different choices of h and $\widetilde{\mathcal{U}}$ will lead to different trade-offs between performance and liveness.

To make sure that the optimization problem (13) can be solved in a computationally efficient manner, it is often desirable to use a projection operator and set that are convex in u for a given $\mathbf{x}$ and t. A particularly popular choice in the literature for h is the l_2 distance from the nominal controller, i.e., $h := ||u - \pi_{\text{nom}}(\mathbf{x}, t)||_2^2$. For the remainder of this section, we use this l_2 projection operator and choose three different $\widetilde{\mathcal{U}}$ that result in particularly interesting liveness filters. We will conclude the section by comparing the resultant filters.

B. Least Restrictive Filter

To blend performance with liveness, the reachability analysis is typically applied in a least-restrictive fashion [3]:

$$\pi(\mathbf{x}, t) = \begin{cases} \pi_{\text{nom}}(\mathbf{x}, t) & \text{if } V(\mathbf{x}, t) < 0 \\ \pi^*_{\text{live}}(\mathbf{x}, t) & \text{if } V(\mathbf{x}) = 0 \end{cases} \tag{14}$$

The *least restrictive (LR) filter* follows the nominal controller when the system state $\mathbf{x}$ is in the interior of $\mathcal{G}_{\text{live}}(t)$ (i.e. $V(\mathbf{x}, t) < 0$), and it takes corrective actions given by $\pi^*_{\text{live}}(\mathbf{x}, t)$ when the system is on the boundary of or at the risk of exiting $\mathcal{G}_{\text{live}}(t)$. This ensures that the system never exits $\mathcal{G}_{\text{live}}(t)$. The controller in (14) is least restrictive in the sense that it follows the given nominal controller that can optimize criteria besides liveness, and only interferes when the system is at the risk of breaching liveness. Consequently, the system can optimize other objectives while progressing towards the goal, but such

freedom is not afforded by $\pi^*_{\text{live}}(\mathbf{x}, t)$ as its sole objective is to ensure system's liveness.

The LR filter (14) can be obtained using our general framework (13) by using the following $\widetilde{\mathcal{U}}$:

$$\widetilde{\mathcal{U}}(\mathbf{x}, t) = \begin{cases} \mathcal{U} & \text{if } V(\mathbf{x}, t) < 0 \\ \{\pi^*_{\text{live}}(\mathbf{x}, t)\} & \text{if } V(\mathbf{x}, t) = 0 \end{cases} \tag{15}$$

Thus, $\widetilde{\mathcal{U}}$ is the set of all permissible controls whenever the system state is inside $\mathcal{G}_{\text{live}}(t)$ and is given by a singleton when it is at the BRT boundary. It is easy to verify that $\widetilde{\mathcal{U}}(\mathbf{x}, t) \subset \mathcal{U}_{\text{live}}(\mathbf{x}, t)$ for all $\mathbf{x}$ and t. Thus, the system will always remain live under the LR filter.

Running Example (Liveness). We now demonstrate the LR filter (14) on the running example. The corresponding system trajectory and the control profile (for u_1) are shown in purple in Fig. 2 and Fig. 3, respectively. As expected, the LR filter chooses between π^*_{live} and π_{nom} to steer the system towards the target set. However, as discussed earlier, using π^*_{live} results in bang-bang behavior (as evident from the spikes in the control profile for the LR filter in Fig. 3) and high control energy. In the next subsection, we propose a smoother version of the LR filter to overcome this challenge.

Remark 1: Upon careful observation, one might note that the LR filter does not take the system fully inside the target set within the time horizon T (the purple trajectory is slightly outside the target set in Fig. 2). This discrepancy between theory and practice is due to the effect of using a discrete-time simulation of a continuous-time system, which sometimes causes a slight delay in the switching between the nominal and the default controller. We will discuss this aspect more as well as a few potential solutions in Sec. VII.

C. Smooth Least Restrictive Filter

While the LR filter (14) accounts for performance and liveness, it still switches to a high energy, bang-bang policy π^*_{live} at the boundary of $\mathcal{G}_{\text{live}}$. To overcome this challenge, we propose an alternative liveness filter that uses the full control authority available to the system as the projection set, rather

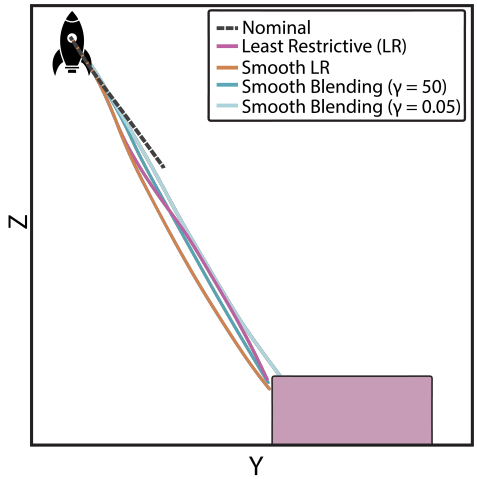


Fig. 2. System trajectories under the nominal and filtered liveness controllers for the rocket landing system. Pink region represents the target set.

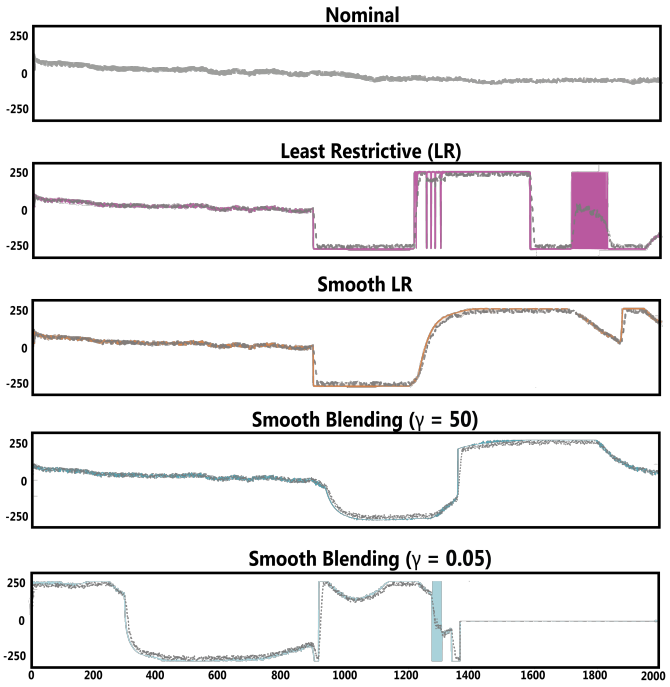


Fig. 3. Control profiles for u_1 for different liveness filters. The nominal controller is shown in dashed gray.

than just $\{\pi^*_{\text{live}}\}$:

$$\begin{aligned} \pi(\mathbf{x}, t) = \arg\min_{u \in \mathcal{U}} \quad & ||u - \pi_{\text{nom}}(\mathbf{x}, t)||_2^2 \\ s.t. \quad & u \in \mathcal{U}_{\text{live}}(\mathbf{x}, t) \end{aligned} \tag{16}$$

i.e., $\widetilde{\mathcal{U}}(\mathbf{x}, t) = \mathcal{U}_{\text{live}}(\mathbf{x}, t)$ for all $(\mathbf{x}, t)$. Intuitively, the above optimization problem computes a control input at each time instant that is "closest" to the nominal controller, yet is within the set of live controls. Furthermore, since $\mathcal{U}_{live} = \mathcal{U}$ whenever $V(\mathbf{x}, t) < 0$, we can simplify the filter in (16) as:

$$\pi(\mathbf{x}, t) = \begin{cases} \pi_{\text{nom}}(\mathbf{x}, t) & V(\mathbf{x}, t) < 0 \\ \pi^+(\mathbf{x}, t) & V(\mathbf{x}, t) = 0 \end{cases} \tag{17}$$

where $\pi^+(\mathbf{x}, t)$ is obtained by solving the following optimization problem:

$$\begin{aligned} \min_{u \in \mathcal{U}} \quad & ||u - \pi_{\text{nom}}(\mathbf{x}, t)||_2^2 \\ s.t. \quad & D_t V(\mathbf{x}, t) + \nabla V(\mathbf{x}, t) \cdot f(\mathbf{x}, u, d^*) = 0 \end{aligned} \tag{18}$$

where d^* is the optimal disturbance given by (10). We refer to the controller in (17) as the *smooth least restrictive (smooth LR) filter*. Similar to the LR filter (14), the smooth LR filter (17) follows π_{nom} when the system state is in the interior of $\mathcal{G}_{\text{live}}(t)$. However, when the system state is at the boundary of $\mathcal{G}_{\text{live}}(t)$, rather than using $\pi^*_{\text{live}}(\mathbf{x}, t)$, it chooses a live control that is *closest* to the nominal control by solving a Quadratic Program (QP), thus allowing for a smoother control profile.

Remark 2: The objective of the optimization problem (18) need not to be quadratic. As long as the objective is convex in u, the optimization problem in (18) remains convex in u for control-affine systems and can be solved efficiently online.

Running Example (Liveness). The trajectory and control profile corresponding to the smooth LR filter are shown in orange in Fig. 2 and 3. We observe that the smooth LR filter

effectively "smoothes" out the bang-bang behaviors that the LR filter displayed; control profile now follows a gradual curve dictated by the QP formulation. This results in lower control energy and less jittery behavior of the resulting system when compared to the original LR controller.

D. Smooth Blending of Performance and Liveness

The LR and smooth LR filters employ live controls as a last-minute resort when the system is at the risk of breaching liveness. Despite its simplicity, this sudden and inconsistent switching to a liveness controller has a few drawbacks: (a) switching at the BRT boundary can still result in a jittery control profile if the nominal control is far from the set of live controls; (b) a last-minute switching can lead to liveness violation upon the slightest delay in switching. To remedy these drawbacks, we take inspiration from the CBF literature. Specifically, we introduce a "CBF-like" constraint to define the projection set that encourages the filtered control to be "more" liveness-ensuring as the system approaches the boundary of $\mathcal{G}_{\text{live}}(t)$:

$$\begin{aligned}\widetilde{\mathcal{U}}(\mathbf{x},t) =&\{u \in \mathcal{U} : \\ &D_t V(\mathbf{x},t) + \nabla V(\mathbf{x},t)\cdot f(\mathbf{x},u,d^*) \leq -\gamma V(\mathbf{x},t)\}\end{aligned} \tag{19}$$

where γ is a tunable parameter that determines how quickly a control is permitted to drive the systems towards the boundary of $\mathcal{G}_{\text{live}}(t)$. The left hand side of the constraint in (19) is the total derivative of $V(\mathbf{x},t)$ evaluated at state $\mathbf{x}$ and time t. Thus, intuitively, the above constraint limits the rate at which the value function can increase, i.e., how quickly is the system allowed to approach the boundary of $\mathcal{G}_{\text{live}}(t)$, at any given state $\mathbf{x}$. When the system is on the boundary of $\mathcal{G}_{\text{live}}(t)$, we have $-\gamma V(\mathbf{x},t) = 0$, and the controller must output a control that would drive the system to a state that has same value. Specifically comparing with the LR and smooth LR filters in this section, the projection set in (19) blends π_{nom} with π^*_{live} in such a way that the system can take gradually more stringent corrective actions as the system becomes more at risk of breaching the liveness, resulting in less jerky controls.

The behavior of the control constraint in (19) is quite similar to that of the CBF constraint in CBF-QP [2], as γ determines how quickly the value function can change. However, it can be difficult to find a valid CBF function in the presence of control bounds and disturbances, requiring online tuning of γ to make sure that the constraint remains feasible at all times, which can be quite challenging in practice. On the other hand, the reachability-based controller in (19) is always *live* and *feasible*, as we prove next, bypassing the need of tuning γ to ensure liveness.

Lemma 3: Let $\mathbf{x}$ be any state such that $V(\mathbf{x},t) \leq 0$ and $\mathbf{x} \notin \mathcal{L}$. Define $\widetilde{\mathcal{U}}(\mathbf{x},t) = \{u \in \mathcal{U} : D_t V(\mathbf{x},t) + \nabla V(\mathbf{x},t) \cdot f(\mathbf{x},u,d^*) \leq -\gamma V(\mathbf{x},t)\}$. Then (a) $\widetilde{\mathcal{U}}(\mathbf{x},t)$ is non-empty, and (b) $\widetilde{\mathcal{U}}(\mathbf{x},t) \subseteq \mathcal{U}_{\text{live}}(\mathbf{x},t)$ for all $\gamma \geq 0$.

Intuitively, Lemma 3 states that (a) using the projection set in (19) always results in a feasible filtering problem, and (b) liveness is always guaranteed under the resultant control input, regardless of the value of γ. An alternative interpretation of Lemma 3 is that, as long as the value function $V(\mathbf{x},t)$ is differentiable, it acts as a robust CBF for any value of γ and can be used to synthesize CBF-like controllers. Even though a number of works have synthesized CBFs using the reachability analysis (e.g., [10]), these methods need to explicitly select a value of γ during the value function computation, making it challenging to tune γ online. In contrast, Lemma 3 indicates that γ can in fact be chosen online, independent of the value function computation.

Nevertheless, the value of γ still affects the blending of π_{nom} and π^*_{live}. Specifically, as $\gamma \to \infty$, the constraint in (19) no longer limits the rate of change in the value function and the blending controller starts behaving like the smooth LR filter (17). Thus, the system prioritizes the performance until the liveness is at risk. On the other hand, as $\gamma \to 0$, the constraint in (19) becomes very stringent and does not allow value function to increase. Consequently, the blending controller starts behaving like the default liveness controller in (9), prioritizing the system liveness over performance. Thus, by choosing γ in between the two extremes, we are effectively blending these two controllers, creating different trade-offs between liveness and performance.

Running Example (Liveness). We show the smooth blending filter for different values of γ. The trajectories are shown in different shades of cyan in Fig. 2. In the first case with $\gamma = 0.05$, the system's behavior is very similar to the default reachability controller, where the system is very cautious and avoid moving towards the BRT boundary. However, from the corresponding control profile in Fig. 3, we notice that the controller follows a slightly smoother version of the bang-bang behavior that we saw in Fig. 1 and the control applied is not always the maximal control, especially towards the later times. On the other hand, using $\gamma = 50$, the system goes very close to the boundary of the BRT, behaving similarly to the smooth LR filter, as expected.

Effect of γ. To illustrate the effect of γ, we plot the overall trajectory cost (given by (3)), normalized by the cost of the default controller for different values of γ in Fig. 4. As evident from the figure, the choice of γ can significantly affect the performance of the smooth blending filter. Specifically, as γ approaches 0, the system behaves more and more like π^*_{live}, keeping stringent liveness characteristics. This liveness, however, comes at the cost of a drop in performance (the controller cost is very high for small γ). On the other hand, as we increase γ (till around $\gamma = 1$), we see a proclivity towards using a controller that is close to the nominal controller while encouraging liveness. This results in a significant decrease in the trajectory cost. However, beyond a certain γ ($\gamma = 1$ in this case), the cost is again seen to rise slightly. This is because, for such a high γ, the controller is allowed to behave more and more like the smooth LR filter. The blending filter chooses the nominal controller as long as the system remains live; however, this choice leads the system closer to the BRT boundary, and the controller has to revert to a high-cost control in order to keep the system live, ultimately raising the overall controller cost. Hence, for high γ, the smooth-blending controller has a similar cost as the smooth LR controller.

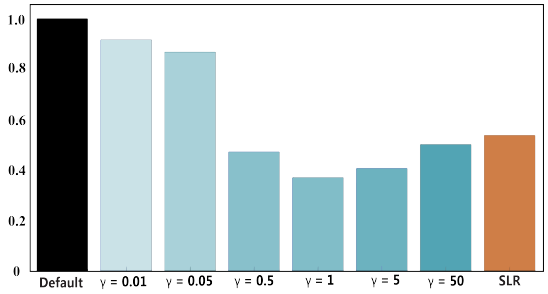


Fig. 4. Overall (normalized) trajectory cost for variations in γ.

E. Comparison of Different Liveness Filters

We now compare the performance of different liveness filters across five metrics:

1) *minimum distance to the target set*, measuring the liveness of each filter;
2) *total cost* accumulated over the trajectory, measuring the performance of each filter;
3) *computation time* per step, measuring the filter latency;
4) *control energy*, measuring control authority used;
5) *jerk energy*, measuring the jerk in the control profile.

The obtained results are summarized in Fig. 5.

First, we note that despite a poor nominal controller, all liveness filters steer the system very close to the target set (minimum distance to the target close to zero). However, most filters have a small non-zero distance to the target set. This is because in our simulation we use a small but non-zero timestep, which introduces a small delay in the application of the filtered policy, resulting in the system state being outside of the BRT momentarily. We discuss this aspect further, as well as potential remedies in Sec. VII-A. The exception is the smooth blending filter with small γ, as it very closely behaves like the default controller sacrificing performance almost entirely to reach the target. In terms of performance, we note that π^*_{live} results in a very high trajectory cost, mostly due a high control energy cost expected of its bang-bang nature. In contrast, liveness filters balance liveness with performance, resulting in a lower trajectory cost. In this case, we observe that smoother control profiles typically equate to lower trajectory costs, resulting in a better performance by smooth blending filter followed by smooth LR filter followed by LR filter. We further note that for $\gamma = 50$, the performance of the smooth blending and smooth LR filters is very close, as expected. However, as γ decreases, the smooth blending filter prioritizes liveness over performance, resulting in an increased trajectory cost. We see a very similar pattern in the control energy plots.

Next, we compare the computation time of different controllers. Here, the default controller emerges as the fastest option, as it only requires computing π^*_{live}, which typically is very fast, especially when the value function is represented as a neural network. In fact, in this case, computing the nominal controller is more expensive than computing π^*_{live}. For the same reason, the LR filter is faster than the MPC method, as it resorts to π^*_{live} when the system state is at the BRT boundary. Compared to the aforementioned controllers, the smooth LR and smooth blending filters leads to a higher computation time, as they require solving an optimization problem additionally

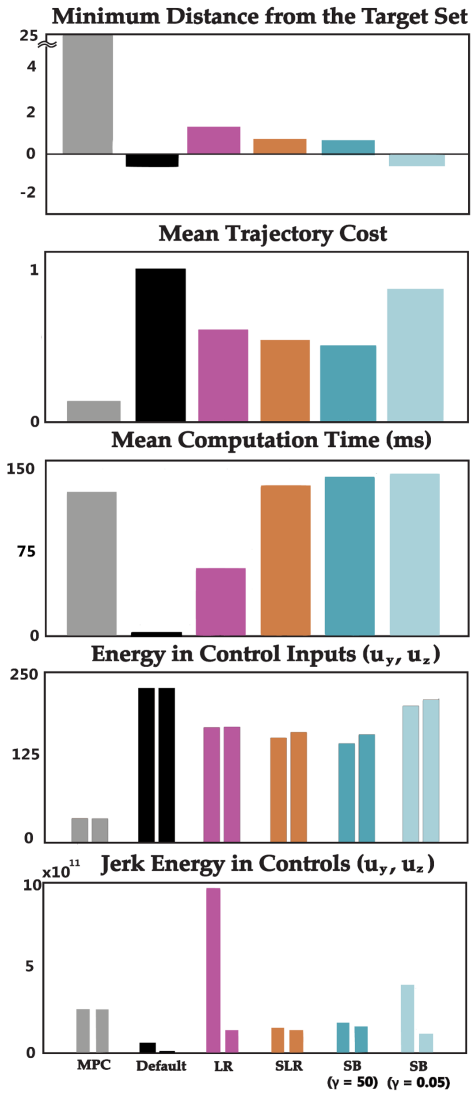


Fig. 5. Comparison of different liveness filters across performance, computation time, and smoothness of the control profile. MPC is the nominal controller. LR and SLR stand for the least-restrictive and the smooth least-restrictive filters. SB stands for smooth blending.

to computing the nominal MPC controller. Moreover, the smooth LR filter has a lower computation cost among the two because it only solves an optimization problem when the system state is at the BRT boundary. In general, the better the nominal controller is at maintaining liveness, the smaller the computation time for the smooth LR filter.

Finally, we measure the average jerk in the control trajectories generated by different filters. Despite its extremum seeking behavior, the default controller in this case has a low jerk, as the controller does not switch between the two extremes often. In contrast, the LR filter leads to a sudden switch from the nominal controller to π^*_{live} at the BRT boundary, leading to a chattering and high jerk. The smooth LR and smooth blending filters smooth out these transitions between the two controllers, leading to lower jerks.

Overall recommendation. Upon comparing different liveness filters, we recommend to use the smooth blending controller with a high value of γ, as it nicely trades off performance and liveness trough the tuning of γ, while maintaining a low

jerk. At the same time, it allows to pick a large γ to spare the user from tuning while keeping a considerable performance increase as seen in Fig. 4. A close second choice is the smooth least restrictive filter, which also shares the above advantages. In addition, it has a lower computation latency, making it particularly suitable for real-world robotic systems.

V. Safety Filtering Using Reachability Analysis

We now turn our attention to safety filtering. One important difference to note is that the value function often converges for the safety problem, i.e. $\mathcal{G}_{\text{unsafe}}$ stops growing after some amount of time, beyond which the system has enough time to avoid the target set despite the worst case disturbance. Consequently, we can use the converged value function $V(\mathbf{x})$, which is no longer a function of time t, to synthesize safety controllers, and the resulting controllers are time-invariant and can be expressed as:

$$\pi^*_{\text{safe}}(\mathbf{x}) = \arg\max_{u\in\mathcal{U}} \min_{d\in\mathcal{D}} \nabla V(\mathbf{x}) \cdot f(\mathbf{x}, u, d) \tag{20}$$

Nevertheless, the proposed filters can easily be extended to incorporate time, similar to liveness filters. To demonstrate its effectiveness, we introduce the following example case.

Running Example (Safety). Consider a 4D system model of a blimp with state $\mathbf{x} = (x, y, z, \theta)$, with control over the rate of change of altitude and yaw angle, as described by the following dynamics:

$$\begin{aligned} \dot{x} &= v\cos(\theta) + d_1 \\ \dot{y} &= v\sin(\theta) + d_2 \\ \dot{z} &= u_1 \\ \dot{\theta} &= u_2 \end{aligned} \tag{21}$$

where x, y and z denote the X, Y and Z positions of the center of mass of the system, respectively, and v is the speed of the system on the XY plane. The control inputs are the vertical velocity in the Z direction and the rate of change of heading. In addition, there are velocity disturbances in the X and Y directions. We consider a $10m$ x $10m$ x $5m$ position state space (Fig. 6) where the objective is to reach a target area of radius $R = 0.5m$ located at $(x, y, z) = (7, 7, 3)m$ (pink sphere) while avoiding collision with the obstacle set $\mathcal{L}$ that corresponds to a collection of spheres of various sizes spread across the position state space (gray spheres). In this case, the implicit obstacle function is defined as the signed distance to the closest gray sphere, $l(\mathbf{x}) = \min_i d(\mathbf{x}, S_i)$ where $d(\mathbf{x}, S_i) = \sqrt{(x_i - x)^2 + (y_i - y)^2 + (z_i - z)^2} - R_i$ is the distance from state $\mathbf{x}$ to the sphere S_i of radius R_i centered at (x_i, y_i, z_i). In (21) $v = 2m/s$ is the velocity in the XY plane, the control and disturbance bounds are given as $u_1 \in [-1, 1]m/s$, $u_2 \in [-\pi, \pi]\text{rad}/s$ and $|| \begin{bmatrix} d_1 & d_2 \end{bmatrix}^\top ||_2 \leq 0.5m/s$.

For this example, to illustrate that the proposed method is agnostic to how the value function is obtained, we leverage the Level Set Toolbox [21] and its reachability helper library HelperOC [9] to compute a numerical approximation of the BRT. Level-Set methods solve the HJB-VI numerically over a uniformly discretized state space grid. We use a 4D grid of $81 \times 81 \times 41 \times 21$ points for our computations. The solution of the HJB-VI provides us with the value function, $V(\mathbf{x}, \tau)$, which can be used to extract the $\mathcal{G}_{\text{unsafe}}$ similar to (8). In this case, the value function converges after 0.5 seconds; thus, we consider the time converged $\mathcal{G}_{\text{unsafe}}$. A 3D slice of $\mathcal{G}_{\text{unsafe}}$ for a fixed value of $\theta = 0$ is presented in Fig. 6. The expansion of the BRT can be seen in the opposite direction, as states oriented this way will inevitably collide with obstacles if they start too close. By definition of the BRT, as long as the system starts outside the blue region, it is guaranteed to remain safe under the default safety controller π^*_{safe} (20). The system trajectory under π^*_{safe} starting at a state outside the BRT and the corresponding control profiles are shown in Fig. 6. The system remains safe under the default safety controller as expected; however, the default safety controller steers the system as far away from the obstacles as possible, disregarding any other performance criterion the system might have (such as reaching the pink goal region). Moreover, the control profile is jittery, which is typically undesirable for real-world applications.

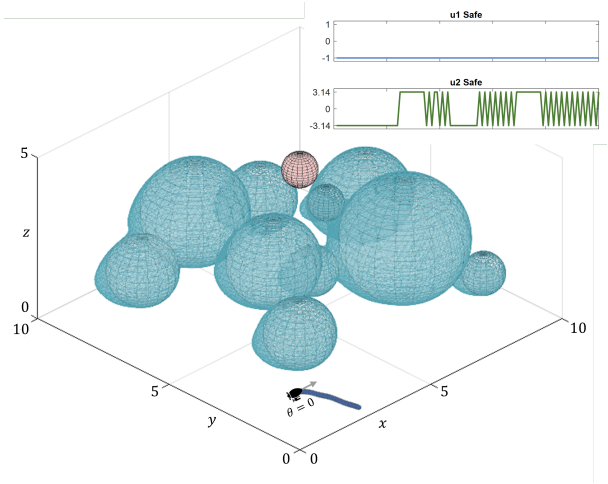


Fig. 6. A xyz-slice (for $\theta = 0$) of the BRT, $\mathcal{G}_{\text{unsafe}}$, for the safety example is shown in blue. The gray spheres inside the BRT correspond to the obstacles (the target set in this case) and the pink sphere represents the set of goal states. The system trajectory under the default safety controller is shown in dark blue and the corresponding controller profiles are shown in the inlets.

A. Characterizing the set of safe controls

Similar to liveness filtering, we will first characterize the set of *all* safe control inputs for the system, and then use it to design various safety filters. To ensure safety, we would like the system to stay out of the $\mathcal{G}_{\text{unsafe}}$ at all times. Thus, intuitively, the set of safe control $\mathcal{U}_{\text{safe}}(\mathbf{x})$ at a state $\mathbf{x}$ are the control inputs that instantaneously keep the system state outside $\mathcal{G}_{\text{unsafe}}$, thereby ensuring recursive safety. Mathematically,

$$\mathcal{U}_{\text{safe}}(\mathbf{x}) = \begin{cases} \mathcal{U} & \text{if } V(\mathbf{x}) > 0 \\ \{u \in \mathcal{U} : \nabla V(\mathbf{x}) \cdot f(\mathbf{x}, u, d^*) = 0\} & \text{if } V(\mathbf{x}) = 0 \\ \emptyset & \text{if } V(\mathbf{x}) < 0 \end{cases} \tag{22}$$

The proof of (22) follows similarly to that for the liveness case, except that we do not need to account for the time-derivative of the value function in the controller expression, since the

value function has converged. The set of safe controls is $\mathcal{U}$, when the system state is outside $\mathcal{G}_{\text{unsafe}}$. When the system state is at the boundary of $\mathcal{G}_{\text{unsafe}}$, the set of safe controls are those that keep the state instantaneously at the boundary of $\mathcal{G}_{\text{unsafe}}$. Finally, safety cannot be ensured under any control input if the system state is inside $\mathcal{G}_{\text{unsafe}}$. An explicit characterization of $\mathcal{U}_{\text{safe}}$ suggests the following general structure of a safety filter:

General Safety Filter. Given a nominal controller, π_{nom}, a safety filter can be constructed as:

$$\begin{aligned} \pi(\mathbf{x},t) = \arg\min_{u} \quad & h(u, \pi_{\text{nom}}(\mathbf{x},t)) \\ s.t.\ u \in \widetilde{\mathcal{U}}(\mathbf{x},t), \ & \text{with } \widetilde{\mathcal{U}}(\mathbf{x},t) \subseteq \mathcal{U}_{\text{safe}}(\mathbf{x}) \end{aligned} \tag{23}$$

Similar to liveness filters, we refer to h as the *projection operator* and $\widetilde{\mathcal{U}}$ as the *projection set*. Since $\widetilde{\mathcal{U}} \subseteq \mathcal{U}_{\text{safe}}$, the above filter makes sure that the system remains safe at all times.

B. Least Restrictive Safety Filter

The set of safe controls can be used to design a least restrictive safety controller for the system:

$$\pi(\mathbf{x},t) = \begin{cases} \pi_{\text{nom}}(\mathbf{x},t) & V(\mathbf{x}) > 0 \\ \pi^*_{\text{safe}}(\mathbf{x}) & V(\mathbf{x}) = 0 \end{cases} \tag{24}$$

Similar to the liveness case, the least restrictive safety controller follows the nominal controller when the system is not risk of breaching safety, and it switches to $\pi^*_{\text{safe}}(\mathbf{x})$ (20) when the system is on the boundary of the BRT. The LR safety filter in (24) can be obtained using our general framework in (23) by using the following $\widetilde{\mathcal{U}}$:

$$\widetilde{\mathcal{U}}(\mathbf{x},t) = \begin{cases} \mathcal{U} & \text{if } V(\mathbf{x}) > 0 \\ \{\pi^*_{\text{safe}}(\mathbf{x})\} & \text{if } V(\mathbf{x}) = 0 \end{cases} \tag{25}$$

Running Example (Safety). For this running example we use an MPC-based nominal controller, whose objective is to minimize distance between the system and the target area, subjected to the system dynamics and the control bounds. To highlight the impacts of safety filtering, the nominal controller is built such that it does not consider obstacle avoidance in its objective. The system state trajectories and control profiles under the nominal controller are shown in grey in Fig. 7 and Fig. 8 respectively. As expected, the nominal controller fails to satisfy the safety constraint (i.e. avoiding the obstacles).

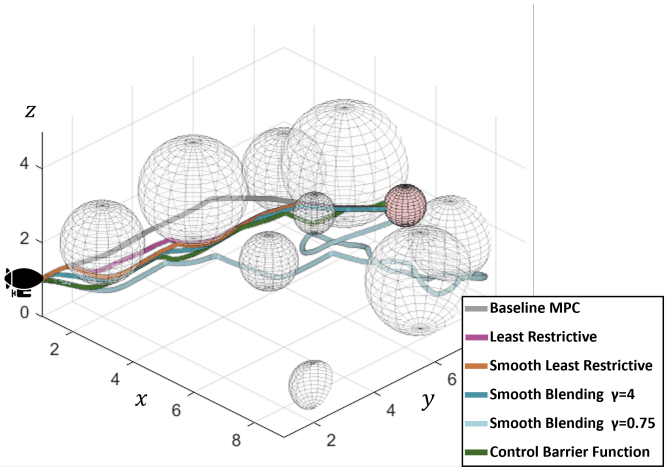


Fig. 7. Trajectories for the nominal control, the proposed safety filters, and a CBF-based filter.

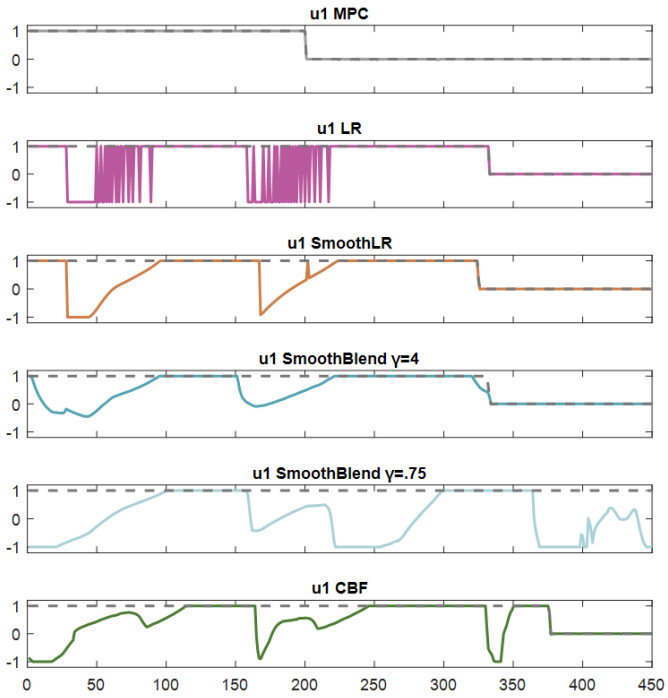


Fig. 8. Control profiles for u_1 (z-velocity) for the nominal controller (gray dashed line), the proposed safety filters, and a CBF-based filter.

We next demonstrate the least restrictive safety filter acting over the nominal controller. The corresponding system trajectory and control profiles (for u_1) are shown in Fig. 7 (purple) and Fig. 8 respectively. This filtering technique allows the system to avoid collision with obstacles, but it comes with the shortcoming of an overly aggressive control profile at the boundary of the unsafe set near the obstacles, as the controller switching and the bang-bang nature of π^*_{safe} results in a jittery control profile that jumps to the limits of the control authority.

C. Smooth Least Restrictive Safety Filter

Some of the shortcomings of the least restrictive filter can be addressed by the smooth least restrictive filter:

$$\pi(\mathbf{x},t) = \begin{cases} \pi_{\text{nom}}(\mathbf{x},t) & V(\mathbf{x}) > 0 \\ \pi^+(\mathbf{x},t) & V(\mathbf{x}) = 0 \end{cases} \tag{26}$$

where $\pi^+(\mathbf{x})$ is obtained by solving the following optimization problem:

$$\begin{aligned} \min_{u \in \mathcal{U}} \quad & ||u - \pi_{\text{nom}}(\mathbf{x},t)||_2^2 \\ s.t. \quad & \nabla V(\mathbf{x}) \cdot f(\mathbf{x}, u, d^*) = 0 \end{aligned} \tag{27}$$

The smooth least restrictive filter can readily be obtained using our general filtering framework by selecting $\widetilde{\mathcal{U}} = \mathcal{U}_{\text{safe}}$, i.e., using the maximal safe control set at all times.

Compared to the least restrictive filter, the smooth least restrictive filter allows the system to take a control action that is "closest" to the nominal control, yet keeping it safe at all times, leading to a smoother control profile. Note that we have

dropped the D_t term from the constraint in (27) because we assume the usage of the converged value function.

Running Example (Safety). The results of applying a smooth least restrictive filter to the running example is shown in orange in Fig. 7 and its control profiles in Fig. 8. Wherever possible, the controller avoids the bang-bang nature of π^*_{safe} by allowing the selection of controls that are close to the nominal controller, yet guaranteed to be safe. This smooths out an overly aggressive control profile generated by the least restrictive filter at the boundary of $\mathcal{G}_{\text{unsafe}}$.

D. Smooth Blending of Performance and Safety

Again quite similar to its liveness counterpart, the smooth blending safety filter utilizes the CBF-like constraint to blend the safety and performance objectives.

$$\begin{aligned} \min_{u\in\mathcal{U}} \quad & h(u, \pi_{\text{nom}}(\mathbf{x},t)) = ||u - \pi_{\text{nom}}(\mathbf{x},t)||_2^2 \\ s.t. \quad & \nabla V(\mathbf{x}) \cdot f(\mathbf{x}, u, d^*) \geq -\gamma V(\mathbf{x}) \end{aligned} \tag{28}$$

Similar to the least restrictive and smooth least restrictive filters, the smooth blending filter allows the system state to move towards the boundary of $\mathcal{G}_{\text{unsafe}}$ (i.e., the value function can decrease), but unlike the aforementioned controllers, it limits the rate (determined by the user-defined coefficient γ and the value of the current state $\mathbf{x}$) at which the value function is allowed to decrease. This phenomenon (a) avoids a sudden switching to $\pi^*_{\text{safe}}(\mathbf{x})$, leading to a smoother profile; and, (b) encourages the system to maintain a non-zero distance from the BRT boundary at all times.

The smooth blending filter can be obtained by using $\widetilde{\mathcal{U}}(\mathbf{x},t) = \{u \in \mathcal{U} : \nabla V(\mathbf{x}) \cdot f(\mathbf{x},u,d^*) \geq -\gamma V(\mathbf{x})\}$ within the proposed general safety filter framework. Similar to Lemma 3, it can be shown that $\widetilde{\mathcal{U}}(\mathbf{x},t) \subseteq \mathcal{U}_{\text{safe}}(\mathbf{x})$ for all $\gamma \geq 0$.

Running Example (Safety). Resulting state trajectories of applying the smooth blending filter for two different values of γ are shown in blue and light blue in Fig. 7, and their control profiles are shown in Fig. 8. As evident from the control profiles, the smooth blending controllers do not produce any sudden switching since they are free to select controls closer to the nominal controls as long as the decrease in value is bounded by the prescribed rate. This rate is adjusted through the parameter γ, which determines how "confident" the system feels moving toward the unsafe states at a given value level. For this example we used values of $\gamma = 4$ and $\gamma = 0.75$. A larger value of γ allows the system trajectory to move closer to the obstacles (increasingly behaving like a smooth LR filter), while a smaller value results in a more conservative behavior (behaving similar to the default safety controller). This can also be seen from the system trajectory corresponding to $\gamma = 0.75$, wherein the safety filter is pushing the system away from the obstacles to ensure that the value function doesn't decrease too much (similar to the default safety controller), resulting in a very curvy and inefficient trajectory to the goal. In contrast, the trajectory corresponding to $\gamma = 4$ ventures fairly close to the BRT boundary. These trajectories highlight a critical characteristic of smooth blending filter – the system performance can heavily depend on γ. Thus, a proper tuning of γ is needed to obtain a satisfactory behavior.

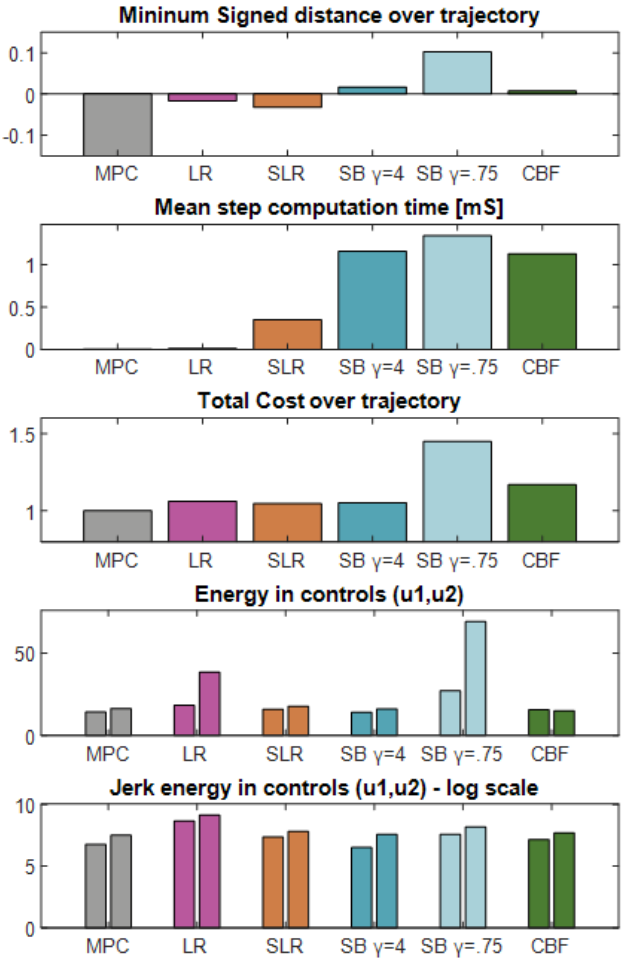


Fig. 9. Comparison between the baseline MPC controller, proposed safety filters and a CBF-based filter.

Even though the proposed filter ensures safety for any value of $\gamma \geq 0$, we do not study mechanisms to compute an appropriate γ in this work. An appropriate value of γ will depend on how the user characterizes performance, safety conservativeness, and the compromise between them. The choice of gamma further depends on the system dynamics, failure set, control and disturbance bounds, and the nominal controller. For our running example, we simulated various values of γ and picked two values to highlight the extreme behaviors that γ can cause in the filtered trajectory.

E. Comparison of Different Controllers

In this section, we present a comparison between the proposed filtering schemes across five metrics: *total cost* accumulated over the trajectory, *minimum distance to the obstacle set*, the controller *computation time* per step, *control energy*, and control *jerk energy*. The control profiles for u_1 are shown in Fig. 8 and the metrics for both u_1 and u_2 are presented in Fig. 9.

The baseline MPC controller does not consider any obstacles, thus getting the most negative signed distance as it penetrates through them; for all other metrics, it will be considered as the base case we will use for comparison.

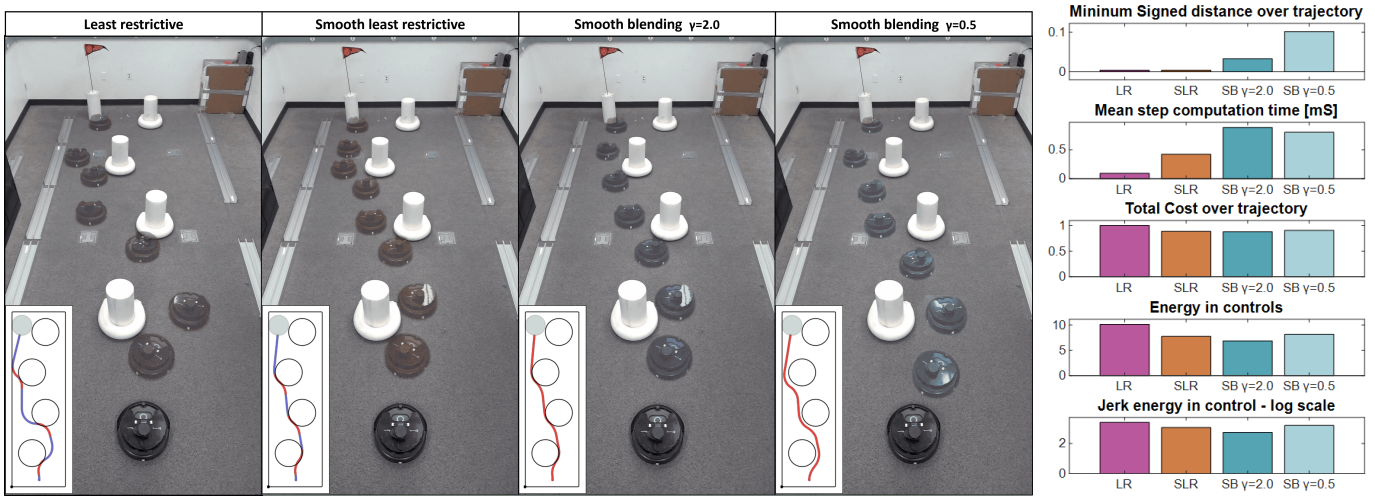


Fig. 10. (Left) Trajectories for the Turtlebot safely navigating through the obstacle field to reach the goal area under different safety filters. A sampling-based MPC method is used as a nominal controller. Inlet plots consider the robot as a point entity and inflated obstacles, blue trajectories indicate the nominal controller is in operation, while red indicates that the safety filtered control is being applied to the robot. (Right) Comparison of different safety filters across performance, controller smoothness, computation time, and safety metrics.

Considering the smooth blending filter's CBF-inspired nature, we also implement a CBF-based safety filter applied to the same baseline MPC controller. Since the dynamical system defined by (21) is control-affine and the relative degree of $l(\mathbf{x})$ to the dynamics is one, we can use the approach presented in [27] to use it as a High Order CBF. We choose the hyperparameters for this approach such that the CBF covers $\mathcal{G}_{\text{unsafe}}$, and the hyperparameter that corresponds to our parameter γ is set to 4 for a fair comparison.

The least restrictive filter maintains safety and only gets minimum penetration over obstacles; this slight penetration is due to the switching happening not strictly at $V(\mathbf{x}) = 0$ but when the value function is negative due to finite grid and time step resolution. The safety enforcement increases the total cost over the trajectory compared to the base MPC controller, because when safety is at risk, it steers the system away from unsafe states which conflicts with the baseline controller objective. The least restrictive filter is the fastest to implement among all safety filters as the default safety controller can be computed quickly using the BRT, without requiring to solve any optimization problem, unlike other safety filters. The downside lies in high energy and high jerk in the controls due to the bang-bang nature of the default safety controller.

The smooth least restrictive filter also suffers from some penetration over obstacles because it uses the same switching condition as the regular least restrictive filter, but presents improvements both in the energy and jerk of the controls as it no longer relies on purely bang-bang safety controller, with the drawback of increased computation time per step as (27) needs to be solved each time that safety is at risk.

The final three controllers correspond to smooth blending of performance and safety for two γ values and the CBF-based filter. Inclusion of the $-\gamma V(\mathbf{x})$ term in the safety filter allows the system to reason about safety before reaching the boundary of the unsafe set, allowing the system to maintain a positive signed distance to obstacles over the whole trajectory. This comes at the expense of an increased mean calculation time, especially for the case with smaller γ as the more conservative constraint in (28) is slightly harder to solve. The characteristics mentioned so far are shared with the CBF-based filter, where we observe very similar results to that of a smooth blending filter with $\gamma = 4$. This is expected since we are essentially solving an equivalent optimization problem. We also observe some increased cost for the CBF-based filter, as it leads to a more conservative unsafe set compared to the BRT, which corresponds to the minimal unsafe set. The behavior of the CBF-based safety filter becomes even more conservative or overly optimistic if the hyperparameter during the CBF synthesis is not properly tuned. Thus, the safety and conservativeness of a CBF-based filter are directly affected by its synthesis, which can be challenging in practice for general nonlinear systems. It is also worth noting that a high γ behaves quite similarly to the smooth LR filter, without its downside of accidentally penetrating the obstacle due to a last-minute switching.

Overall recommendation. Similar to the liveness case, we recommend to use the smooth blending safety filter with a high value of γ, as it nicely tradeoff performance and safety, while maintaining a low jerk. At the same time, using a very high γ can bypass the need for tuning it.

VI. Hardware Experiments

We next apply the proposed safety filters on a Turtlebot4, a Dubins-like robotic platform (see Fig. 10). The robot needs to reach a goal position in an area situated with multiple obstacles. To complete this task, we use a shooting-based MPC nominal controller that does not consider obstacle avoidance in its objective. We choose this nominal controller on purpose to highlight the effect of safety filtering. The nominal controller will be filtered using the techniques presented in Section V to guarantee safety while navigating towards the goal.

We model the dynamics of the system as a Dubins car with $V = 0.3\ m/s$ and $u \in [-0.75, 0.75]\ rad/s$ where the state $\mathbf{x} = [x, y, \varphi]^T$ represents the x and y positions and the heading of the robot:

$$\dot{x} = V \sin\varphi; \quad \dot{y} = V \cos\varphi; \quad \dot{\varphi} = u \tag{29}$$

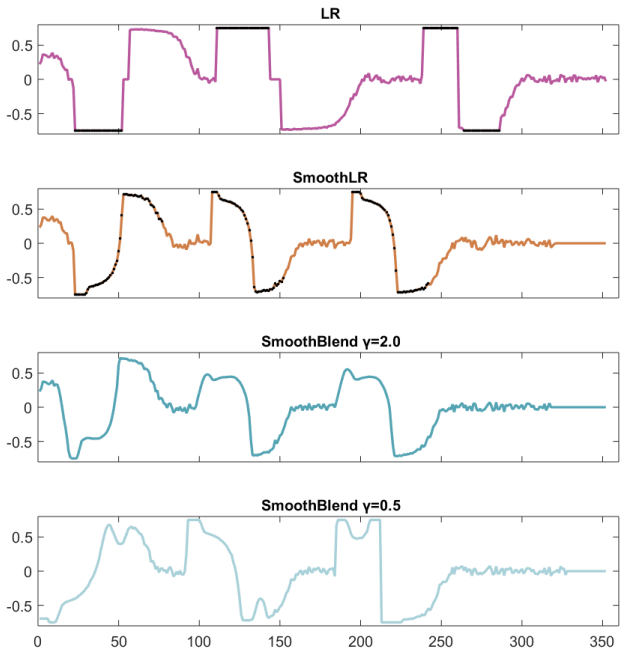


Fig. 11. Control profiles for different safety filters obtained during our Turtlebot experiments. Black points in the Least Restrictive and Smooth Least Restrictive filters indicate that the filter is activated. They are omitted for the Smooth Blending filter as the filter is always active.

The BRT for this experiment is calculated on a $[x, y, \varphi]$ grid of $[101; 281; 181]$ points, which gives a resolution of $2cm$ in position over the experiment space and 2° in orientation. The BRT was computed until convergence (in this case, the BRT converges after a horizon of $1s$). The converged value function is then used to synthesize various safety filters.

We assume perfect state measurement during our experiments. The system state is obtained through the robot's internal pose estimation capability given by a combination of measurements from an IMU, optical floor tracking sensor, and wheel encoders. To avoid accidental breach of safety in least restrictive and smooth least restrictive filters due to limited grid resolution and discrete timestep (see Sec. V-B), we use a non-zero threshold to activate the safety controller. Specifically, we use the condition $V(\mathbf{x}) = \epsilon$ (instead of $V(\mathbf{x}) = 0$) to trigger switching to π^*_{safe}. A value of $\epsilon = 0.05$ was used in the experimental results shown.

The robot trajectories corresponding to different safety filters are shown in Fig. 10. The corresponding control trajectories are shown in Fig. 11. We also compare different safety filters across various metrics. These results are shown in Fig. 10 (right). With this information, we highlight the core characteristics of each safety filter:

Least Restrictive Filter: Even though the least restrictive filter keeps the system safe with fast queries of the default safety controller, the bang-bang nature of the safety controller forces the system to take the sharpest turn possible as it only reasons about maximally increasing safety. Additionally, it results in high energy control inputs. Together, these results in a high accumulated cost over the robot trajectory.

Smooth Least Restrictive Filter: Similarly to the previously presented simulation results, this method shows improvements compared to the least restrictive filter, both in the total energy and jerk of the control profile. It also keeps lower accumulated cost as the safe control is trying to align with the underlying nominal controller. The drawback comes in the form of an increased computation time per step, as (27) must be solved each time the system safety is at risk. It is also worth noting that using a non-zero threshold to activate the QP-based safety controller in (26) avoids any accidental collision with the obstacles, unlike our simulation results.

Smooth Blending of Performance and Safety: The final two trajectories show filtering with the smooth blending filter with $\gamma = 2$ and $\gamma = 0.5$. The use of the filter on every time step of the trajectory (rather than just at the BRT boundary) allows the system to stay further away from the obstacles. The drawback is the requirement to solve (28) on each step, which results in an increase in the mean step computation time. The computation time, however, is still low enough to have any significant impact on robot operation. We also observe that a smaller value of γ leads to a more cautious robot behavior, reflected in the largest separation to obstacles among all safety filters (see Fig. 10). This comes at the cost of an increased control energy, control jerk, and an overall higher accumulated cost, as compared to $\gamma = 2$.

VII. Discussion of the Theory-Practice Gap

A. Effect of a finite simulation timestep

The filters designed in this paper are based on the assumption that the control input can be applied to the system in a continuous time fashion. However, controller implementation on a real system often involves a zero-order hold, leading to a non-zero simulation timestep δ. If δ is too large, the proposed controllers may no longer be able to ensure liveness/safety for the underlying continuous time system. For example, a least restrictive filter ensures liveness by switching to π^*_{live} at the boundary of the BRT. However, under a finite δ, π^*_{live} may no longer be able to keep the system state within the BRT, as the liveness is only guaranteed under a *continuous time*, state feedback reachability controller. This discrepancy between the continuous-discrete structure is demonstrated in Fig. 12, where we show the trajectory variations generated by the least restrictive filter for different values of δ.

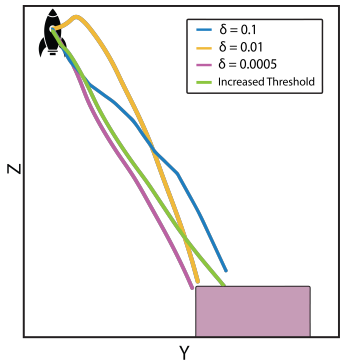


Fig. 12. Effect of discretization timestep (δ). With all other parameters kept the same, the trajectories followed by the system under the same LR controller differ with different δ (blue $\delta = 0.1$, yellow $\delta = 0.10$, and pink $\delta = 0.0005$ trajectories). Using a modified version of the LR controller ($\delta = 0.0005, \epsilon = -0.1$), the system reaches the goal (green trajectory).

Such effects are more pronounced in the least restrictive filters since there, the system evolves along the BRT boundary. If our simulation timestep is too high, it is highly possible that our system temporarily escape the BRT and enter the unsafe region. One possible solution could be a faster feedback with gradually decreasing δ as the system approaches the BRT or adaptive δ depending on the value function. One can also use a discrete-time variant of the BRT computation itself. Alternatively, instead of the switching at the BRT boundary ($V(\mathbf{x}) = 0$) as seen in Eqn. (24), we can choose to switch with some ϵ margin from the boundary. This slightly modifies the least restrictive controller as follows,

$$\pi(\mathbf{x}, t) = \begin{cases} \pi_{\text{nom}}(\mathbf{x}, t) & \text{if } V(\mathbf{x}, t) < \epsilon \\ \pi^*_{\text{live}}(\mathbf{x}, t) & \text{if } V(\mathbf{x}) \geq \epsilon \end{cases} \tag{30}$$

An example corresponding to $\epsilon = -0.1, \delta = 0.0005$ is shown with the green line in Fig. 12. As evident from the figure, this strategy is able to maintain liveness at all times and eventually reaches the target set (unlike the pink trajectory using the same $\delta = 0.0005$, but $\epsilon = 0$). This strategy was also employed during our hardware experiments to avoid accidental collisions with the obstacles. Nevertheless, a thorough analysis of the effect of a finite simulation timestep on system liveness/safety would be a promising future research direction.

B. Non-differentiable value functions

The proposed controllers in this paper rely on differentiability of the value function, which may not always be the case [13], [20]. To discuss the impact of non-differentiability of the value function on the proposed controllers, consider the Dubins car dynamics in (29). Moreover, we consider the smooth least restrictive safety filter for our illustration. This requires solving an optimization problem where we find the closest control to the nominal control, such that the condition $\langle \nabla V(\mathbf{x}), f(\mathbf{x}, u, d^*) \rangle = 0$ with $u \in \mathcal{U}$ is met.

First we consider the scenario where the value function is being numerically calculated using a grid based approach [9],[21]. The goal region and the obstacle are shown in light pink and gray respectively in Fig. 13(left). The nominal planner, unaware of the environment obstacle, plans a straight path to the goal region (blue trajectory).

Correspondingly, the safety controller engages when the system state reaches the BRT boundary (purple point). The value function has a kink at this state, wherein the system has two equally viable option (hard left-turn or hard right-turn) to barely avoid the obstacle. However, at this state, the value function gradient in the theta direction does not exist, as is evident from the inlet in Fig. 13(left) showing the variations on value around this point as a function of θ . Thus, one can no longer use the safety filters that rely on the value function gradient (such as a smooth least restrictive filter or a smooth blending filter). One approach to avoid this issue could be to use one-sided gradient of the value function to construct the safety filter. The result of using a left-sided or a right-sided approximation of the gradient is shown in green trajectories in Fig.13(left). Both approximations are able to keep the system safe while steering it to its goal location.

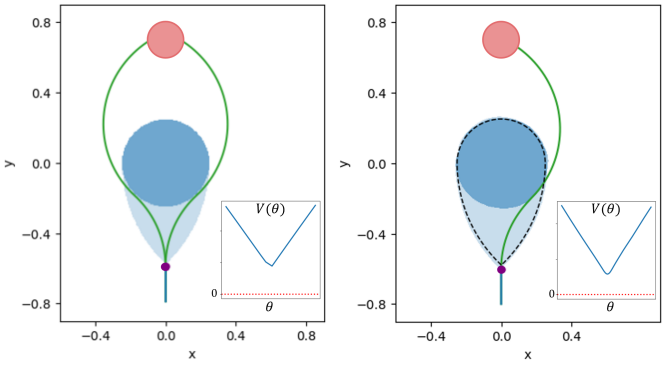


Fig. 13. (Left) BRT slice for $\theta = \pi/2$. The value function computed using a grid-based approach exhibits non-differentiability at the purple point. Inlet: the value function plot at the purple point as a function of theta. The value function exhibits a kink at $\theta = \pi/2$. The green trajectories are computed using one-sided gradients. (Right) A smooth approximation of the value function computed using a learning-based method (the black dashed line is the BRT corresponding to the grid-based approach). The value function now changes smoothly as a function of θ at the purple point.

A second approach is to obtain a conservative approximation of the value function that is differentiable and then construct safety/liveness filters using this approximation. For illustration, here we use a learning-based method, DeepReach, to approximate the value function using neural network representations that are differentiable by construction [4], [16]. In Fig.13(right), we show a comparison of the BRT obtained through DeepReach and the previously discussed grid-based method. The inlet in Fig. 13(right) shows how the learned value function is differentiable at the tip of the BRT, unlike the grid-based value function. Moreover, the BRT obtained through the learning method encompasses the numerically approximated value function, which keeps all the safety guarantees intact at the expense of a slightly more conservative BRT, for completeness the smooth least restrictive filtered trajectory using this differentiable representation is shown in green.

VIII. Conclusion And Future Work

In this paper, we explicitly characterize the set of all live and safe controls for a dynamical system using Hamilton-Jacobi reachability analysis. Leveraging these maximal sets, we introduce a general framework for designing safety and liveness filters for the system. We propose three distinct safety/liveness filters, balancing performance, control smoothness, and latency, allowing to select based on system needs.

While these filters integrate safety/liveness and performance, they modify the nominal controller only at the current timestep, unaware of the long-term effects of its action. This might result in myopic controllers. Second, the discrete time nature of actual hardware implementations of these controllers might pose a challenge to the guarantees proposed. Third, we currently assume that the target set is known and does not change online. However, that may not be the case for many robotics applications and an online adaptation of the reachability analysis and safety filters might be required. We aim to tackle these challenges in future works. Finally, we currently rely on the differentiability of the value function to construct safety and liveness filters. A theoretical analysis of filter design using subgradients or one-sided gradients of the value function is another important future research direction.

REFERENCES

[1] A. D. Ames, X. Xu, J. W. Grizzle, and P. Tabuada. Control barrier function based quadratic programs for safety critical systems. *IEEE Transactions on Automatic Control*, 62(8):3861–3876, 2017.

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

[3] S. Bansal, M. Chen, S. Herbert, and C. J. Tomlin. Hamilton-Jacobi reachability: A brief overview and recent advances. In *CDC*, 2017.

[4] Somil Bansal and Claire J. Tomlin. Deepreach: A deep learning approach to high-dimensional reachability. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1817–1824, 2021.

[5] Dimitri Bertsekas. *Dynamic programming and optimal control: Volume I*, volume 4. Athena scientific, 2012.

[6] Dimitri Bertsekas. *Reinforcement learning and optimal control*. Athena Scientific, 2019.

[7] Francesco Borrelli, Alberto Bemporad, and Manfred Morari. *Predictive control for linear and hybrid systems*. Cambridge University Press, 2017.

[8] Frank M Callier and Charles A Desoer. *Linear system theory*. Springer Science & Business Media, 2012.

[9] Mo Chen and Sylvia Herbert. helperOC Library, 2019. https://github.com/HJReachability/helperOC.

[10] Jason J. Choi, Donggun Lee, Koushil Sreenath, Claire J. Tomlin, and Sylvia L. Herbert. Robust control barrier–value functions for safety-critical control. In *2021 60th IEEE Conference on Decision and Control (CDC)*, pages 6814–6821, 2021.

[11] E. A. Coddington and N. Levinson. *Theory of ordinary differential equations*. Tata McGraw-Hill Education, 1955.

[12] Mark L Darby and Michael Nikolaou. Mpc: Current practice and challenges. *Control Engineering Practice*, 20(4):328–342, 2012.

[13] Lawrence C Evans and Panagiotis E Souganidis. Differential games and representation formulas for solutions of hamilton-jacobi-isaacs equations. *Indiana University mathematics journal*, 33(5):773–797, 1984.

[14] Kai-Chieh Hsu, Haimin Hu, and Jaime Fernández Fisac. The safety filter: A unified view of safety-critical control in autonomous systems. *arXiv preprint arXiv:2309.05837*, 2023.

[15] Yuxi Li. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*, 2017.

[16] Albert Lin and Somil Bansal. Generating formal safety assurances for high-dimensional reachability. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10525–10531, 2023.

[17] John Lygeros. On reachability and minimum cost optimal control. *Automatica*, 40(6):917–927, 2004.

[18] K. Margellos and J. Lygeros. Hamilton-Jacobi Formulation for Reach–Avoid Differential Games. *IEEE Trans. on Automatic Control*, 56(8):1849–1861, 2011.

[19] I. Mitchell. A toolbox of level set methods. *http://www. cs. ubc. ca/mitchell/ToolboxLS/toolboxLS. pdf, Tech. Rep. TR-2004-09*, 2004.

[20] Ian Mitchell, Alex Bayen, and Claire J. Tomlin. A time-dependent Hamilton-Jacobi formulation of reachable sets for continuous dynamic games. *IEEE Transactions on Automatic Control*, 2005.

[21] Ian M Mitchell et al. A toolbox of level set methods. *UBC Department of Computer Science Technical Report TR-2007-11*, page 31, 2007.

[22] Alexander Robey, Haimin Hu, Lars Lindemann, Hanwen Zhang, Dimos V. Dimarogonas, Stephen Tu, and Nikolai Matni. Learning control barrier functions from expert demonstrations. In *2020 59th IEEE Conference on Decision and Control (CDC)*, pages 3717–3724, 2020.

[23] Mohit Srinivasan, Amogh Dabholkar, Samuel Coogan, and Patricio A. Vela. Synthesis of control barrier functions using a supervised machine learning approach. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7139–7145, 2020.

[24] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[25] Kim P. Wabersich, Andrew J. Taylor, Jason J. Choi, Koushil Sreenath, Claire J. Tomlin, Aaron D. Ames, and Melanie N. Zeilinger. Data-driven safety filters: Hamilton-Jacobi reachability, control barrier functions, and predictive methods for uncertain systems. *Preprint*, 2023.

[26] Grady Williams, Paul Drews, Brian Goldfain, James M Rehg, and Evangelos A Theodorou. Aggressive driving with model predictive path integral control. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1433–1440. IEEE, 2016.

[27] Wei Xiao and Calin Belta. Control barrier functions for systems with high relative degree. In *2019 IEEE 58th Conference on Decision and Control (CDC)*, pages 474–479, 2019.

[28] Wei Xiao, Ramin Hasani, Xiao Li, and Daniela Rus. Barriernet: A safety-guaranteed layer for neural networks, 2021.

APPENDIX

A. Proof of Lemma 1

Case 1: $V(\mathbf{x},t) < 0$. Since the value function is continuous in state and time, and the system trajectory itself is continuous in time, the value function is continuous along the system trajectory. Thus, we can always pick a small enough δ such that $V\left(\xi_{\mathbf{x},t}^{u,d^*}(t+\delta), t+\delta\right) < 0$. That is, we can keep the system momentarily inside the BRT for any u. Thus, $\mathcal{U}_{\text{live}}(\mathbf{x},t) \equiv \mathcal{U}$.

Case-2: $V(\mathbf{x},t) = 0$. Since by assumption the value function is differentiable for all $\mathbf{x}$ and t, for sufficiently small $\delta > 0$, we can rewrite the value function using Taylor expansion:

$$\begin{aligned} &V\left(\xi_{\mathbf{x},t}^{u,d^*}(t+\delta), t+\delta\right) \\ &\approx V(\mathbf{x},t) + \frac{\partial V}{\partial t}\delta + \frac{\partial V}{\partial \mathbf{x}} \cdot f(\mathbf{x},u,d^*)\,\delta \\ &= \delta\left[\frac{\partial V}{\partial t} + \frac{\partial V}{\partial \mathbf{x}} \cdot f(\mathbf{x},u,d^*)\right] \end{aligned}$$

where the equality follows because $V(\mathbf{x},t) = 0$. Thus, to ensure that $V\left(\xi_{\mathbf{x},t}^{u,d^*}(t+\delta), t+\delta\right) \leq 0$, we must have

$$\left[\frac{\partial V}{\partial t} + \frac{\partial V}{\partial \mathbf{x}} \cdot f(\mathbf{x},u,d^*)\right] \leq 0$$

Since $\mathbf{x} \notin \mathcal{L}$, $l(\mathbf{x}) > 0$. It follows immediately that $l(\mathbf{x}) - V(\mathbf{x},t) = l(\mathbf{x}) - 0 > 0$. Thus, according to the HJI-VI in (6), we must have:

$$\begin{aligned} &\frac{\partial V}{\partial t} + \min_u \frac{\partial V}{\partial \mathbf{x}} \cdot f(\mathbf{x},u,d^*) = 0 \\ &\equiv \frac{\partial V}{\partial t} + \frac{\partial V}{\partial \mathbf{x}} \cdot f(\mathbf{x},u,d^*) \geqslant 0 \quad \forall u \in \mathcal{U} \end{aligned}$$

Thus, the only feasibility for ensuring liveness is

$$\left[\frac{\partial V}{\partial t} + \frac{\partial V}{\partial \mathbf{x}} \cdot f(\mathbf{x},u,d^*)\right] = 0$$

which corresponds to $V\left(\xi_{\mathbf{x},t}^{u,d^*}(t+\delta), t+\delta\right) = 0$

Case-3: $V(\mathbf{x},t) > 0$. In this case, the set of live controls is trivially an empty set, as per the definition of a BRT.

B. Proof of Corollary 1

Suppose the value function $V(\mathbf{x},t)$ is differentiable at all $\mathbf{x}$ and t. Take $\mathbf{x} \in \mathcal{G}_{\text{live}}$. We split into the following two cases.

Case 1: $V(\mathbf{x},t) < 0$. Since $\pi^*_{\text{live}}(\mathbf{x},t) \in \mathcal{U}$, we also have $\pi^*_{\text{live}}(\mathbf{x},t) \in \mathcal{U}_{\text{live}}(\mathbf{x},t)$ as $\mathcal{U} \equiv \mathcal{U}_{\text{live}}(\mathbf{x},t)$ in this case.

Case 2: $V(\mathbf{x},t) = 0$. According to the HJI-VI in (6), we must have:

$$\frac{\partial V}{\partial t} + \min_u \frac{\partial V}{\partial \mathbf{x}} \cdot f(\mathbf{x},u,d^*) = 0$$

By the definition of the default controller, it achieves the minimum in the above equation, i.e.,

$$\frac{\partial V}{\partial t} + \frac{\partial V}{\partial \mathbf{x}} \cdot f(\mathbf{x},\pi^*_{\text{live}}(\mathbf{x},t),d^*) = 0$$

Thus, $\pi^*_{\text{live}}(\mathbf{x},t) \in \mathcal{U}_{\text{live}}(\mathbf{x},t)$ by the definition of the set of live controls.

C. Proof of Lemma 2

We will prove this result by contradiction. Suppose the system never reaches the target set in the time interval $[t, T]$. Then we must have $V(\mathbf{x}(T), T) > 0$ (if not, then $V(\mathbf{x}(T), T) = l(\mathbf{x}(T)) < 0$ and we are done).

Since $V(\mathbf{x}(t), t) < 0$ and the value function is continuous, we must have that $V(\mathbf{x}(\tilde{t}), \tilde{t}) = 0$ for some $\tilde{t} \in (t, T)$. Moreover, for state $\mathbf{x}(\tilde{t})$, the set of live controls is given by (Eq. (12)

$$\frac{\partial V}{\partial t} + \frac{\partial V}{\partial \mathbf{x}} \cdot f(\mathbf{x}, u, d^*) = 0$$

It follows immediately that $\frac{dV}{dt} = \frac{\partial V}{\partial t} + \frac{\partial V}{\partial \mathbf{x}} \cdot f(\mathbf{x}, u, d^*) = 0$. Thus, we must have: $V(\mathbf{x}(\tau), \tau) = 0 \quad \forall \tau \geqslant \tilde{t}$ This contradicts our initial hypothesis that $V(\mathbf{x}(T), T) > 0$. Hence, the system must reach the target set at some point in the time interval $[t, T]$.

D. Proof of Lemma 3

Case 1: $V(\mathbf{x}, t) = 0$. In this case,

$$\widetilde{\mathcal{U}}(\mathbf{x}, t) = \{u \in \mathcal{U} : D_t V(\mathbf{x}, t) + \nabla V(\mathbf{x}, t) \cdot f(\mathbf{x}, u, d^*) \leq 0\}$$

However, as per the HJI-VI in (6), we have

$$D_t V(\mathbf{x}, t) + \nabla V(\mathbf{x}, t) \cdot f(\mathbf{x}, u, d^*) \geq 0 \ \forall u \in \mathcal{U}$$

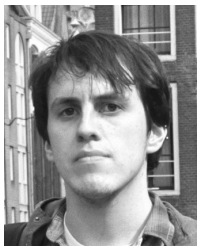
Thus, $\widetilde{\mathcal{U}}(\mathbf{x}, t)$ can be equivalently written as

$$\begin{aligned}\widetilde{\mathcal{U}}(\mathbf{x}, t) =& \{u \in \mathcal{U} : D_t V(\mathbf{x}, t) + \nabla V(\mathbf{x}, t) \cdot f(\mathbf{x}, u, d^*) = 0\} \\ =& \mathcal{U}_{\text{live}}(\mathbf{x}, t)\end{aligned}$$

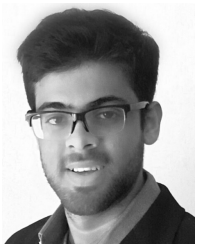
Moreover, since $\pi^*_{\text{live}}(\mathbf{x}, t) \in \mathcal{U}_{\text{live}}(\mathbf{x}, t)$ by Corollary 1, we have that $\pi^*_{\text{live}}(\mathbf{x}, t) \in \widetilde{\mathcal{U}}(\mathbf{x}, t)$. Thus, $\widetilde{\mathcal{U}}(\mathbf{x}, t)$ is non-empty.

Case-2: $V(\mathbf{x}, t) < 0$. In this case, $\mathcal{U}_{\text{live}}(\mathbf{x}, t) = \mathcal{U}$ so $\widetilde{\mathcal{U}}(\mathbf{x}, t) \subseteq \mathcal{U}_{\text{live}}(\mathbf{x}, t)$ trivially.

To prove that $\widetilde{\mathcal{U}}(\mathbf{x}, t)$ is non-empty, let $\alpha = -\gamma V(\mathbf{x}, t)$. Thus, $\widetilde{\mathcal{U}}(\mathbf{x}, t) = \{u : D_t V(\mathbf{x}, t) + \nabla V(\mathbf{x}, t) \cdot f(\mathbf{x}, u, d^*) \leq \alpha\}$. Note that since $V(\mathbf{x}, t) < 0, \alpha > 0$. Moreover, since for the default liveness controller, we have that $D_t V(\mathbf{x}, t) + \nabla V(\mathbf{x}, t) \cdot f(\mathbf{x}, \pi^*_{\text{live}}(\mathbf{x}, t), d^*) = 0 < \alpha$, $\pi^*_{\text{live}}(\mathbf{x}, t) \in \widetilde{\mathcal{U}}(\mathbf{x}, t)$. Thus, $\widetilde{\mathcal{U}}(\mathbf{x}, t)$ is non-empty.



Javier Borquez Currently pursuing a Ph.D. in Electrical and Computer Engineering at the University of Southern California, Los Angeles, as a member of the Safe and Intelligent Autonomy Lab. He received his M.S. and B.S. in Electrical Engineering from the University of Santiago de Chile in 2017 and 2015, respectively. His research interests lie in using optimal control theory, numerical methods, and learning-enabled approaches to develop safety-guaranteeing frameworks for the real-world deployment of robots.



Kaustav Chakraborty Ph.D. student in the ECE department at the University of Southern California, Los Angeles, where he is advised by Prof. Somil Bansal and is a member of the Safe and Intelligent Autonomy Lab. Before USC, he received his M.S. in Robotics from the University of Michigan, Ann Arbor, and his B.Tech in Mechanical Engineering from Vellore Institute of Technology, India. His research centers on designing safety frameworks for autonomous systems using sensory feedback.

Hao Wang Currently pursuing the Ph.D. degree in Electrical and Computer Engineering at the University of Southern California, Los Angeles. He received the B.S. degrees in computer science and mechanical engineering from the University of Michigan, Ann Arbor, in 2022. His research interests include optimal control theory and safety in autonomous systems. He is interested in utilizing tools from control theory, machine learning, and optimization to synthesize safe and performant controllers.

Somil Bansal Assistant professor in the ECE department at the University of Southern California, where he leads the Safe and Intelligent Autonomy lab. He received a Ph.D. in Electrical Engineering and Computer Sciences from the University of California at Berkeley in 2020. Before that, he obtained a B.Tech. in Electrical Engineering from the Indian Institute of Technology, Kanpur, and an M.S. in Electrical Engineering and Computer Sciences from UC Berkeley in 2012 and 2014, respectively. His research focuses on understanding how machine learning methods can be combined with classical, model-based planning and control methods to enable intelligent and safe decision-making.