Space to Time: Out-of-Distribution Generalization of Safety Filters via Temporal Disturbance Encoding

Sander Tonkens*, Nikhil Uday Shinde*, Azra Begzadić*, Michael C. Yip, Jorge Cortés, Sylvia Herbert

***Abstract*—Safe operation is essential for autonomous systems in safety-critical environments such as urban air mobility. Value function-based safety filters provide formal guarantees on safety, wrapping an autonomy stack with a layer of protection on its outputted action. Recent approaches leverage offline learned value functions to scale these safety filters to high-dimensional systems. Yet these methods assume knowing the full operational domain a priori - information that is typically unavailable in real world settings. For example, detailed prior knowledge of all possible sources of model mismatch, in the form of disturbances, in the environment is highly unrealistic. Even in well-mapped environments like urban canyons or industrial sites, drones encounter complex, spatially-varying disturbances arising from payload-drone interaction, turbulent airflow, and other environmental factors. We introduce SPACE2TIME, which enables safe and adaptive deployment of offline-learned safety filters, by generalizing across unknown, spatially-varying disturbances. The key idea is to reparameterize spatial disturbances as a time-varying formulation, allowing the use of temporally varying precomputed value functions during online operation. We validate SPACE2TIME through extensive simulations on diverse quadcopter models and real-world hardware experiments, demonstrating significantly improved safety performance over worst-case and naive baselines.**

I. Introduction

Autonomous systems are often deployed in safety-critical environments, where ensuring reliable and safe operation is of paramount importance. For instance, drones operating in mapped environments such as urban canyons or shipyards must remain within known safe regions while subject to complex, spatially-varying wind disturbances. Rather than synthesizing or learning a new safe controller for every task, a more effective approach is to use safety filters that monitor the control at runtime and intervene only when necessary, ensuring safety while not impeding an autonomous system from achieving its non-safety related objectives [1]. A popular approach for constructing safety filters relies on the concepts of Control Barrier Functions (CBFs) [2] and Hamilton-Jacobi reachability (HJR) analysis [3]. Recent efforts have sought to blend these two approaches and derive value function-based safety filters, which use the reachability-based value function as a barrier certificate to ensure formal safety guarantees [4, 5, 6].

However, both of these methods rely on an accurate model of the system and its operational domain. The operational domain characterizes the range of conditions we design the system to operate in, including dynamical changes and environmental variations. Beyond this challenge, constructing a suitable CBF remains an open problem. On the other hand, while HJR offers formal safety guarantees, it suffers from poor scalability due to the curse of dimensionality, limiting its use in high-dimensional systems. As such, learning-based safety value functions have risen in prominence to combat the aforementioned scalability issues [7, 8, 9]. However, these methods also typically assume a known model and fixed operational domain or attempt to be robust across the full operational domain [10]. Such assumptions limit their ability to generalize under distributional shift, resulting in either unsafe or overly conservative behavior.

This work tackles the challenge of learning safe value functions for autonomous systems under varying disturbances in safety-critical environments, such as urban air mobility. This work provides a first step in bridging the gap between offline-learned value functions, which address the scalability challenges of HJR, and the need to handle an evolving or adapting operational domain online. Specifically, we focus on unknown spatially-varying disturbances, a common challenge in urban air mobility, where safe adaptation to such uncertainties is paramount for successful deployment.

To this end, we introduce SPACE2TIME as outlined in Fig. 1, a framework that enables the safe deployment of offline-learned safety filters by adapting to unknown and spatially-varying disturbances encountered during operation. Our key insight is to convert the disturbance's *spatial* rate of change into a *temporal* rate of change, allowing for easy parameterization for offline learning. Paired with an efficient online algorithm, this allows for safe navigation that is less conservative than assuming worst conditions everywhere while being safe, unlike existing approaches. Such an approach tackles both real-time and episodic out-of-distribution (OOD) problems, by reformulating any spatial and temporal disturbances as temporal distribution shifts [11]. The main contributions of our work are as follows:

- We introduce a time-varying disturbance-conditioned safety value function formulation, which is a function of state, a constant disturbance bound increase rate, and time. Effectively, we learn a value function that incorporates disturbances growing over time.
- We propose SPACE2TIME as shown in Fig. 1, a framework that ensures safety in the presence of unknown, spatially-varying disturbances through the use of a time-varying safety filter leveraging the aforementioned offline-learned value functions.
- We validate SPACE2TIME through extensive simulations for a planar quadcopter model and real-world hardware experiments for a planar environment, demonstrating substantial improvements in safety compared to existing approaches,

*Equal contribution. The authors are with the University of California San Diego (E-mails: {stonkens, nshinde, abegzadic, yip, cortes, sherbert}@ucsd.edu)

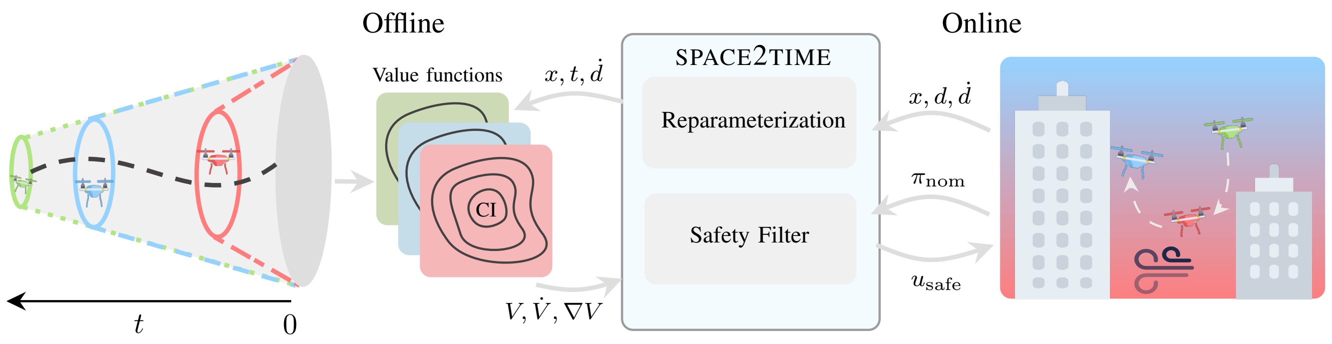


Fig. 1: Conceptual overview of the SPACE2TIME framework. SPACE2TIME characterizes safety by re-parameterizing the spatial variation of the disturbance as a temporal change within the system dynamics. Value functions for this transformed system are then learned offline. This re-parameterization allows the system to maintain safety while leveraging learned value function based safety filters to generate safe, disturbance-aware control in real time.

without significantly sacrificing performance.

II. Related Work

The past decade has seen major progress in safety filtering for robotic systems, particularly through approaches grounded in HJR and CBFs. We highlight relevant approaches to our setting and point readers to recent surveys [12, 13].

Learning-Based Approaches for Hamilton Jacobi reachability analysis. HJR produces a value function whose zero level set encodes the set of initial conditions from which a system may reach a goal and/or avoid an obstacle despite worst-case disturbance [3]. Traditional approaches to solve HJR rely on dynamic programming and therefore scale exponentially with the dimensionality of the system. In recent years, learning-based approaches have vastly improved the scalability of HJR. Reinforcement learning-based methods have been developed to estimate HJR value function based on a Bellman recursive framework that learns from samples collected, with success in many applications [8, 9]. Physics-informed neural networks (PINNs) have also been employed in a self-supervised manner to approximate the value function [7]; an extension of this work learns a parameterized version of the reachability value function based on different disturbance bounds [14]. This approach inspires our work, as it can provide safety across a range of operational domains, each with a different maximum disturbance level. Assuming, however, a set of operational domains fails to address the shift between operational domains. Accounting for this shift is crucial for safety in OOD applications.

Learning-Based Approaches for Control Barrier Functions. Safety is also frequently ensured through the use of CBFs [2, 15, 16, 17]. The construction of a valid CBF and feasibility of the safety filter remains a challenge [12]. Recently, learning-based approaches have been introduced to obtain CBFs, but they often lack formal guarantees or rely on restrictive assumptions [18, 9, 19, 20]. As an alternative, recent work has shown that CBFs can be constructed directly using techniques from HJR [4].

III. Background

Consider a system of the form $\dot{x} = \tilde{f}(x, u, d) = f(x) + g(x)u + d(x)$, where $x \in \mathbb{R}^n$ is the state, $u \in \mathcal{U} \subseteq \mathbb{R}^p$ is the control input, $d \in \mathcal{D} \subseteq \mathbb{R}^q$ is the disturbance, and $\mathcal{U}$ and $\mathcal{D}$ are convex and compact sets. We consider a time horizon $[t, 0]$, where the initial time $t \leq 0$. For each initial time $t \leq 0$, we denote the sets of admissible control and disturbance signals by $\mathbb{U}(t) := \{\boldsymbol{u} : [t, 0] \to \mathcal{U} \mid \boldsymbol{u} \text{ is measurable}\}$ and $\mathbb{D}(t) := \{\boldsymbol{d} : [t, 0] \to \mathcal{D} \mid \boldsymbol{d} \text{ is measurable}\}$. Under Assumption 1 in Appendix A, we denote by $\xi_{x,t}^{\boldsymbol{u},\boldsymbol{d}}(\tau)$ a forward trajectory starting at state x and time t under control and disturbance signals $\boldsymbol{u}$, $\boldsymbol{d}$. The output of this trajectory is the state queried at time τ.

Hamilton-Jacobi Reachability is a model-based optimal control framework that characterizes the set of initial states from which a system can reach a target set and/or avoid a failure set. Here, we introduce both the avoid problem and the reach-avoid problem. For both problems, we consider a constraint function $g : \mathbb{R}^n \to \mathbb{R}$ describing the failure set as $\mathcal{F} := \{x \in \mathbb{R}^n \mid g(x) \leq 0\}$. Then, the reward function associated with the avoid problem is:

$$r_{\text{A}}(x, t, \boldsymbol{u}, \boldsymbol{d}) = \min_{\tau \in [t,0]} g(\xi_{x,t}^{\boldsymbol{u},\boldsymbol{d}}(s)). \tag{1}$$

This encodes the minimum value of g attained by the trajectory over the time horizon. If this minimum is smaller than 0, the trajectory entered the failure set. For the reach-avoid problem, additionally consider the target function $l : \mathbb{R}^n \to \mathbb{R}$ describing the target set $\mathcal{T} := \{x \in \mathbb{R}^n \mid l(x) \geq 0\}$. Then, the reward function associated with the reach-avoid problem is:

$$r_{\text{RA}}(x,t,\boldsymbol{u},\boldsymbol{d}) = \max_{\tau \in [t,0]} \min\{l(\xi_{x,t}^{\boldsymbol{u},\boldsymbol{d}}(\tau)), \min_{s \in [t,\tau]} g(\xi_{x,t}^{\boldsymbol{u}\boldsymbol{d}}(s))\}. \tag{2}$$

A trajectory has a positive reward if it reaches the target $\mathcal{T}$ at some time τ_1, i.e., $l(\xi_{x,t}^{\boldsymbol{u},\boldsymbol{d}}(\tau_1)) > 0$ while having stayed clear of the failure set $\mathcal{F}$, i.e., $g(\xi_{x,t}^{\boldsymbol{u},\boldsymbol{d}}(s)) \geq 0$ for all $s \in [t, \tau_1]$.

HJR considers a game in which one player, here the control u, tries to maximize the reward, whereas a second player, here the disturbance d, acts antagonistically and attempts to minimize the reward. The value function of this game is defined as $V(x, t) = \min_{d \in \mathcal{D}} \max_{u \in \mathcal{U}} c(x, t, \boldsymbol{u}, \boldsymbol{d})$. For the avoid problem, the value function encodes the avoid tube $\mathcal{A}(\mathcal{F}, t) := \{x \in \mathbb{R}^n \mid V(x, t) \geq 0\}$, which is the set of states that can avoid the failure set for time t. Alternatively, the reach-avoid tube $\mathcal{RA}(\mathcal{T}, \mathcal{F}, t) := \{x \in \mathbb{R}^n \mid V(x, t) \geq 0\}$ represents the set of states from which the system is guaranteed to safely reach the target while avoiding the failure set within time t. In Appendix A and B we show how the two formulations can be solved directly or approximately by traditional dynamic programming [3] and self-supervised learning [7].

Control Barrier Functions are a popular technique for enforcing safety. Typically, CBFs consider control-affine systems without disturbances, i.e., $\dot{x} = f(x) + g(x)u$ [2]. A continuously differentiable function h is a CBF if 1) we can represent a given safe set $\mathcal{C}$ as the 0-superlevel set of h, and 2) there exists an extended class $\mathcal{K}$ function α such that, for each $x \in \mathcal{C}$, there exists $u \in \mathbb{R}^m$ satisfying $\nabla h(x)^\top (f(x) + g(x)u) + \alpha(h(x)) \geq 0$. For details on this formulation, see Appendix A.

If a Lipschitz continuous controller $k : \mathbb{R}^n \to \mathbb{R}^m$, defined as $u = k(x)$, satisfies the CBF constraint for all $x \in \mathcal{C}$, then a CBF can be used to ensure safety. Given any nominal control law $u_{\text{nom}} : \mathbb{R}^n \to \mathbb{R}^p$ that may violate safety, CBFs allow for minimal correction using the following optimization problem:

$$\begin{aligned} u^*(x) = \arg\min_u & \|u - u_{\text{nom}}(x)\|_2^2 \\ \text{subject to } & \nabla_x h(x)^\top (f(x) + g(x)u) + \alpha(h(x)) \geq 0. \end{aligned} \tag{3}$$

For control-affine dynamics, this is a quadratic program, resulting in a computationally efficient safety filter. The main challenge with CBFs is in finding a valid CBF. To address this, CBFs can be constructed using HJR [4, 6], which incorporates finite time horizons, control bounds, and disturbance bounds.

IV. Problem Statement

We consider safety-filter synthesis for control-and disturbance-affine systems operating under spatially varying disturbances. The disturbance $d : \mathcal{X} \to \mathbb{R}^q$ is unknown, but its magnitude is bounded by $d_{\max} \in \mathbb{R}^q$ and it is Lipschitz continuous with a known constant L_d. Given a prescribed failure set $\mathcal{F} \subset \mathcal{X}$, our objective is to design a safety filter that adapts to the spatially-varying disturbance, guarantees avoidance of $\mathcal{F}$, and admits an offline-learnable value function for formal safety certification. We consider the following existing approaches from literature for this problem.

Constant Maximum Disturbance. Classical reachability-inspired formulations typically assume a fixed control set $\mathcal{U}$ and a fixed disturbance set $\mathcal{D}$ [3, 10]. While control inputs are often (in practice) spatially invariant across the environment[1], disturbances such as wind may vary more smoothly over space. Based on the full operational design domain, a maximum allowable disturbance set $\mathcal{D}_{\max}$ is typically considered to account for worst-case conditions. However, naively assuming that the disturbance can take any value within $\mathcal{D}_{\max}$ at all states might result in a very conservative value function $V(x, t, \mathcal{U}, \mathcal{D}_{\max})$, particularly if this maximum disturbance is localized in the state space.

Perfect Information (Oracle) . An oracle value function would have perfect access to the complete deterministic state-dependent disturbance function $d(x) \in \mathcal{D}_{\max}$, and compute the optimal value function with $\mathcal{D}_{\text{oracle}}(x,t) = \{d(x)\}$, to find $V_{\text{oracle}} = V(x, t, \mathcal{D}_{\text{oracle}})$. In practice, however, the disturbance function d is unknown prior to deployment (making it unsuitable for offline learning), and pretraining for every possible spatially-varying disturbance landscape is intractable.

[1]Apart from sudden changes due to e.g., actuator failures or restricting the acceleration limits of a car on a slippery surface such as ice.

Parameterizing the Pretrained Value Function by Constant Disturbance Sets. An alternative is to pretrain a value function for a range of potential operational domains, each with a maximum disturbance that is constant across space, i.e., an ensemble of K value functions, where each iteration k corresponds to a constant disturbance bound set $\mathcal{D}_k$ satisfying $\mathcal{D}_1 \subseteq \cdots \subseteq \mathcal{D}_K \subseteq \mathcal{D}_{\max}$. This yields a family of value functions $V_1(x,t), \ldots, V_K(x,t)$ that can be used to approximate the true disturbance-dependent behavior [14, 21]. However, this approach adapts naively to newly observed measurements online, assuming that the currently observed disturbance bound will remain constant over space and time. Thus, this offers no formal safety guarantees in environments with varying disturbances.

Given the limitations of each approach described above, there is a need to formulate a value function that is amenable to offline learning, but can be used in an adaptive manner online in the face of varying disturbances while maintaining safety. This is the problem tackled in this paper.

V. space2time: Time-Varying Disturbances for Adaptive Safety

A. Offline Safety Value Function Learning

Our key insight is that changes in disturbance bounds over space can be encoded as changes in disturbance bounds over time, as the system moves through the environment, Fig. 1. We can use this temporal reparameterization with a pre-defined control invariant safe target set to compute a reach-avoid value function offline. This computed value function is used as a minimally invasive safety filter online to ensure we can always return to this safe set under our assumed temporal disturbance increase and enables us to safely navigate through OOD environments with unknown spatially varying disturbances. For offline learning, we assume that as the system moves, the disturbance grows linearly over time. Specifically, the disturbance grows at a rate bounded by $|\dot{d}(x)| \leq L_d M_{\tilde{f}} = \dot{d}_{\max}$, where $M_{\tilde{f}}$ bounds the system dynamics $\tilde{f}$. For notational simplicity, we first consider a one-dimensional disturbance, i.e., $d : \mathcal{X} \to \mathbb{R}$. Given a maximum rate of change of a disturbance $\dot{d}_{\max}$, we define the worst-case time-varying disturbance set as

$$\mathcal{D}_{\text{tv}}(x,t) = \left\{ d \in \mathbb{R} \mid |d| \leq \max\{0, d_{\max} - |t|\dot{d}_{\max}\} \right\}. \tag{4}$$

In practice, we construct an ensemble of K value functions, each associated with a constant disturbance rate $\dot{d}_1 \leq \cdots \leq \dot{d}_K \leq \dot{d}_{\max}$. Intuitively, as the system moves forward in time, the disturbance set $\mathcal{D}_{\text{tv}}$ gets progressively larger. To capture this evolution, we augment the dynamics to explicitly model the disturbance rate, resulting in (5) where $z = [x, \dot{d}]$ is the joint state and $w \in [-1, 1]$ represents the normalized range of the disturbance. This formulation is particularly relevant for robotic systems operating in environments where we can model disturbances to vary over the course of a deployment, as is typically considered for OOD problems.

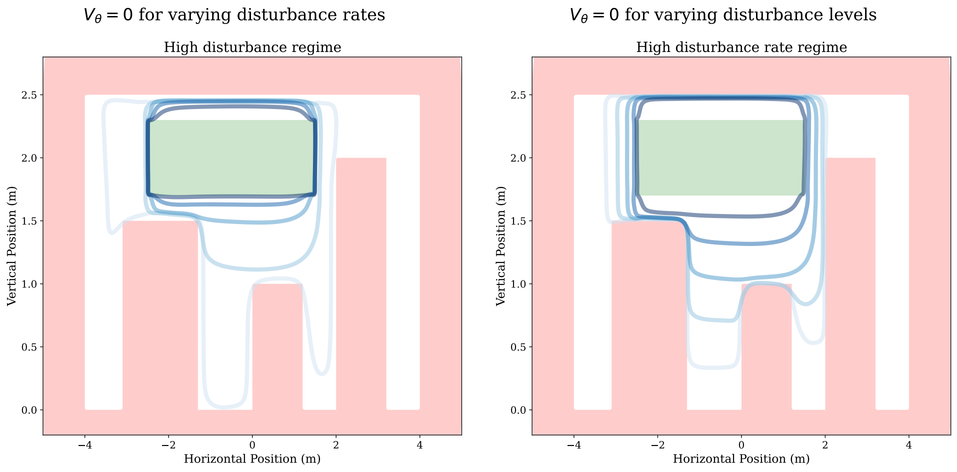


Fig. 2: Safe Sets: The figure on the left shows the 0-level set of the learned value function for a fixed high disturbance level d and increasing disturbance rates, $\dot{d}$,(light to dark blue). Notice that even with higher fixed disturbance, the safe set can be relatively large if the disturbance rate remains low. The figure on the right shows the 0-level set of the learned value function for a fixed high disturbance rate, $\dot{d}$, for increasing levels of disturbance, d (light to dark blue). As the disturbance magnitude increases, the time to get back and thus the safe set contracts.

$$\dot{z} = \hat{f}(z,u,\omega) = \begin{bmatrix} \dot{x} \\ \ddot{d} \end{bmatrix} = \begin{bmatrix} f(x) + g(x)u + \max\{0, d_{\max} - |t|\dot{d}\} \cdot w \\ 0 \end{bmatrix}, \tag{5}$$

Next, we specify a failure set $\mathcal{F}$ (e.g., static obstacles such as buildings) and a fixed static control set $\mathcal{U}$. Additionally, given the assumption that the disturbance increases over time, our proposed approach hinges on an a priori learned or expertly-chosen target set $\mathcal{T}$, which is control-invariant under worst-case disturbance, i.e., for all $d \in \mathcal{D}_{\max}$. This set can be interpreted as a fallback zone, such as a docking location or an open-sky area above a city[2]. We then construct the associated value function and corresponding reach-avoid set under dynamics (5), expressed as $V(z,t) = V(x,t,\dot{d})$. Given a disturbance magnitude $d \in \mathbb{R}^q$ and its rate of change $\dot{d} \in \mathbb{R}^q$, we compute the time required to safely return to the control-invariant set as

$$t_{\text{return}} = \min\left(\frac{d_{\max} - d}{\dot{d}}\right), \tag{6}$$

where the minimum is over the q dimensions of d. This denotes the time at which the disturbance attains its maximum magnitude under our temporally increasing assumption, and thus defines the latest time by which the system must re-enter the prescribed safe set. It is used to query the value function for the safety filter. To handle multi-dimensional disturbances, each disturbance dimension is treated independently in time when computing the value function. The time component in the value function remains one-dimensional and corresponds to the minimum time over all disturbance dimensions.

Parameterized for $\dot{d}$, the value function for the dynamics in (5) can be learned offline using the methods in Appendix B, and provides a more realistic approximation of a spatial variational environment than existing alternatives. In the next section, we discuss how to use this value function online.

[2]We make no prior assumptions on the magnitude of disturbances anywhere in the state space. This in contrast to methods which assume an initial set with a fixed disturbance magnitude (typically 0). Our method also applies to such settings, and solely requires modifying the initial conditions and dynamics.

B. Online Deployment of Temporally-Varying Value Functions

An overview of our framework can be seen in Fig. 1 and Algorithm 1. We consider the setting where $\dot{d}$ can be approximated online. While it is possible to assume the worst-case scenario at all times by setting the disturbance rate $\dot{d}$ equal to its maximum possible value, i.e., $\dot{d} = \dot{d}_{\max}$ for all states x, this approach can be very conservative. Instead, we propose an adaptive strategy that leverages recent observations of the disturbance d and the disturbance rate $\dot{d}$. We store the past H observed values, and select the highest value as our disturbance sample. This enables adaptation to local disturbance dynamics without always assuming the worst case, reducing conservativeness while maintaining a margin of safety.

Algorithm. While the training of our value function is performed offline, our objective is to learn a representation that captures variations induced by different disturbance realizations to use online. Given the current state x, we obtain $d(x)$ and $\dot{d}(x)$. Given a nominal control π_{nom} and value function V, we synthesize a control input u^* using the VB-CBF-based safety filter [6] through (7) where the value function V, its gradient $\nabla_x V$, and its time derivative $\frac{\partial}{\partial t} V$ can be efficiently computed through the forward and backward pass $\mathcal{V}$ of the trained neural network (NN in Alg. 1. The dynamics $\hat{f}(z,u,\omega)$ are provided in (5). The control law is evaluated at each decision step, allowing the system to react online to changes in the estimated disturbance without retraining the value function. Additional implementation details are provided in Appendix F.

SPACE2TIME guarantees safety as long as the spatial variation in the disturbance satisfies the Lipschitz assumptions on the disturbance bounds, ensuring consistency between the true dynamics and the assumptions underlying the value function. For HJR, the value function itself is valid assuming a sufficiently dense grid; for Deepreach, conformal prediction [22] can be used to obtain a valid value function with high probability.

Algorithm 1 SPACE2TIME

```
Require: $V(z,t)$, $\pi_{\text{nom}}(x)$
1: Measure state $x$
2: Determine $d, \dot{d}$ based on approximation scheme        ▷ Conservative or recency-based
3: $t_{\text{return}} \leftarrow \min\left\{\frac{d_{\max}-d}{\dot{d}}\right\}$        ▷ Time to get back
4: $\dot{V}, \nabla_z V, V \leftarrow$ NN($V(z, t_{\text{return}}, \dot{d})$)
5: $u^* \leftarrow$ CBF($z, V, \frac{\partial}{\partial t}V, \nabla_z V, \pi_{\text{nom}}(x)$)        ▷ see (7)
6: Apply $u^*$ to system.
```

TABLE I: Comparison of Our approach against baselines using HJR and Deepreach based value functions. Metrics are generated over 50 trajectories, with 10 goals and 1000 control steps each. % Safety Violations indicates the percentage of failed trajectories, Mean Goal Dist reflects the average minimum distance to each goal with lower values indicating better goal completion, and Mean Traj Len refers to the mean trajectory length before failure with a max of 1000. SPACE2TIME provides the greatest balance between safety and performance.

Approach	% Safety Violations ↓	Mean Goal Dist ↓	Mean Traj Len ↑
HJR Naive	84%	0.73	529.06
HJR Naive Worstcase	0%	1.90	1000
DeepReach Naive	78%	0.79	510.26
DeepReach Ours	30%	0.78	800.14
HJR Ours	30%	0.66	810.46

VI. Experimental Results

We evaluate the performance of SPACE2TIME extensively in simulation and showcase how our method can be deployed directly on hardware. We provide comparisons against baselines that do not explicitly model for disturbance variations.

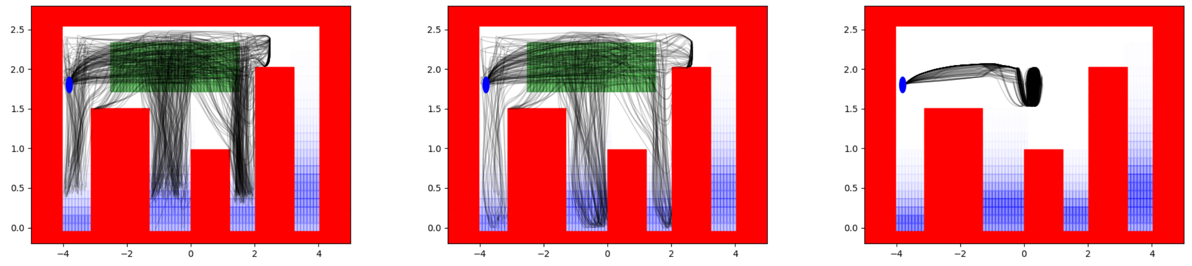
Learning Value Functions. We learn value functions using Deepreach [7], a self-supervised PINN, and parameterize the dynamics across a range of disturbance rates for our method (and disturbance values for the baselines). Unlike the baseline methods, SPACE2TIME requires solving a reach-avoid problem.

Simulation Experiments. We simulate a planar drone environment with state $[x, z, \dot{x}, \dot{z}]$, where the drone flies in the $x-z$ plane. We specifically choose this planar setting to enable validation of our proposed reparameterization against ground truth computed via a dynamic programming-based HJR method. This confirms the validity of our formulation and that it can be effectively approximated through learning. The environment is shown in Fig. 3. The simulated setting represents a cityscape with known building configurations. Additional modeling and implementation details are provided in Appendix D.

The experiments replicate urban inspection or drone delivery scenarios, where the drone must navigate through and above a cityscape, including high-disturbance areas near buildings, to reach multiple sequential goals. We compare HJR and Deepreach variants of our method with naive baseline controllers: i) **HJR Ours:** SPACE2TIME using an ensemble of HJR value functions with fixed disturbance rates ii) **Deepreach (DR) Ours:** using learned disturbance rate parameterized value functions iii) **HJR Naive:** Considers an ensemble of HJR value functions for different disturbance bounds iv) **HJR Naive Worst-Case:** Considers HJR value function for the worst case disturbance bounds $\mathcal{D}_{\max}$ v) **DR Naive:** Using disturbance parameterized learned value function. Each method incorporates a recency-based disturbance estimation strategy, in which the drone uses the most recent disturbance sample from the environment, updated at 4 Hz. All CBF filters are implemented consistently across methods. As summarized in Table I, our approach achieves lower safety violation rates while demonstrating improved goal-reaching performance. Fig. 3 also shows representative trajectories comparing DR ours to DR Naive and HJR Worst Case baseline. The HJR Worst case approach is too conservative, failing to venture out from the safe target set and DR Naive violates safety and crashes as it fails to adequately consider the changing disturbance. DR Ours allows the drone to safely approach the goal.

Hardware Experiments. Like for the simulation experiments, we consider an x, z plane, but to capture realistic drone behavior we rely on a 6-dimensional drone model that incorporates delays in setting desired attitudes. The state includes $[x, v_x, \theta_x, \omega_x, z, v_z]$ where θ_x is the pitch and ω_x is the angular velocity. The control inputs are the desired pitch $\theta_{x,\text{des}}$ and the whole body thrust T. We replicate the urban environment in a hardware experiment using Crazyflie drones with an OptiTrack motion capture setup for state estimation. The cityscape is recreated with stacked boxes, Fig. 4 and spatially varying disturbances are introduced as described in Appendix E. The drone's control rate runs at 50Hz while disturbance sampling is run at 5 Hz (as before, ensuring robustness with $H = 10$). The higher dimensionality of the dynamics model (8 dimensional with the parameteric dimensions) prohibits the use of traditional HJR, thus motivating the use of learning-based value functions, e.g., Deepreach. We compare SPACE2TIME with a Deepreach-based baseline that naively switches between different disturbance bounds [14].

$$u^*(z,t) = \arg\min_{u\in\mathcal{U}} \|u - \pi_{\text{nom}}(x)\|_2^2 \qquad \text{subject to } \frac{\partial}{\partial t}V(z,t) + \min_{\omega\in[-1,1]} \nabla_z V(z,t)^\top \hat{f}(z,u,\omega) \geq -\alpha(V(z,t)), \tag{7}$$



(a) DR Ours **(b)** DR Naive **(c)** HJR Worst Case

Fig. 3: Trajectory Comparison: We visualize the trajectories of Table I of the drone moving through random goals in the city environment. Fig. 3a (on the left) shows trajectories from SPACE2TIME with a value function generated using Deepreach (DR). Fig. 3b (in the center) shows trajectories from the naive parameterized baseline where the value function is also generated with DR. Notice that our safety filter prevents a majority of trajectories from descending too far down the urban canyon and crashing. The baseline trajectories in Fig. 3b are less dense, as a large percentage of the trajectories are cut short due to collisions. Fig. 3c shows the trajectories when using the safety value function for the worst case disturbance bounds. Solely considering worst case bounds results in conservative, non-performant trajectories.

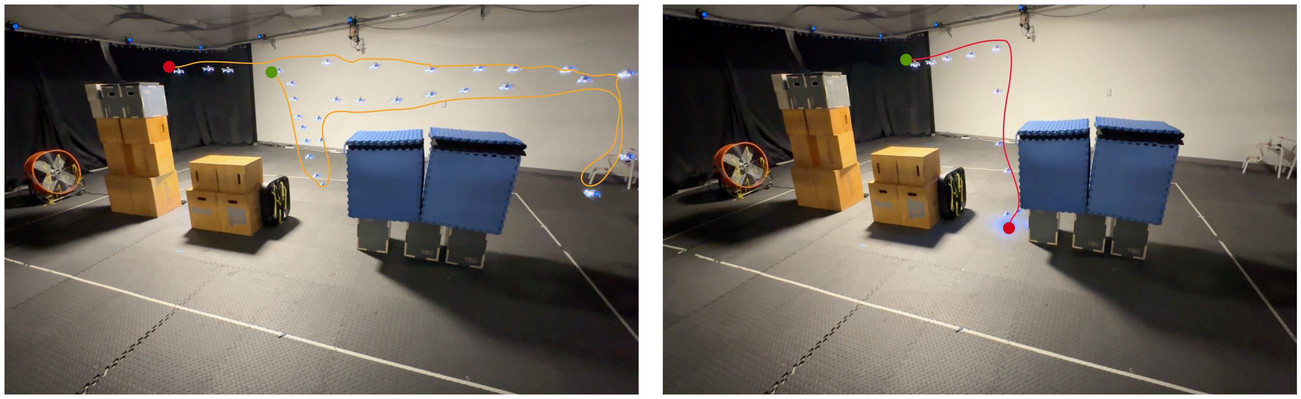


Fig. 4: Real-world drone experiment, comparing **Ours** (left), successfully accounts for the disturbance increase as we descend between the obstacles. In contrast, a **Naive** comparison (right) fails to adequately adapt, leading to a crash.

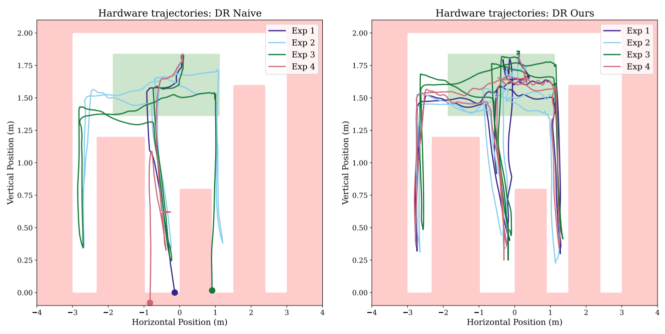


Fig. 5: This visualizes the remaining 4 hardware trajectories, with DR Naive (left) and DR Ours (right). In comparison to our method, DR Naive fails to account for the increase in disturbances and crashes after descending too far down the canyon. These collisions are shown as colored circles (3 out of 4 crashes).

We show that SPACE2TIME preserves safety, whereas naively switching between fixed disturbance bounds leads to crashes (as the drone fails to adequately adapt to the disturbances), Over 5 trials our method achieves a 100% success rate, with the baseline succeeding only 20% of the time. Our hardware experiments are visualized in Fig. 4 and 5. Specifically, by adjusting naively to new disturbance bounds (Fig 5, left), the drone flies into a high wind region (near the bottom of the canyon) and fails to recover, while accounting for these variations through the temporal rate of change (Fig 5, right) results in disturbance-aware trajectories that stay out of the high wind regions. The experimental setup is detailed in Appendix E.

VII. Conclusion

This paper introduced SPACE2TIME, a novel framework for enabling learning value functions offline for deployment in environments with unknown, spatially-varying disturbances. By recasting spatial disturbances into a time-varying formulation, our method leverages the scalability of offline learning, while providing the adaptability required for real-world operation. We validated our approach through extensive simulations and real-world experiments, demonstrating improved safety performance compared to worst-case and naive baselines. As part of our future work, we aim to propose more intelligent safety filters that leverage observed disturbances in known areas of the state space and provide more principled techniques for conservatively estimating the disturbance (and its rate of change). Lastly, we plan to consider a broader class of problems within the OOD domain, focused on settings where it is unreasonable to assume a maximal disturbance bound and explicitly decouple variations across different timescales of operation.

References

[1] K.-C. Hsu, H. Hu, and J. F. Fisac, “The Safety Filter: A Unified View of Safety-Critical Control in Autonomous Systems,” *Annu. Rev. Control. Robotics Auton. Syst.*, vol. 7, 2023.

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

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

[4] J. J. Choi, D. Lee, K. Sreenath, C. J. Tomlin, and S. L. Herbert, “Robust Control Barrier-Value Functions for safety-critical control,” in *Proc. IEEE Conf. on Decision and Control*, 2021.

[5] S. Tonkens and S. Herbert, “Refining Control Barrier Functions through Hamilton-Jacobi reachability,” in *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, 2022.

[6] A. Begzadić, N. Shinde, S. Tonkens, D. Hirsch, K. Ugalde, M. C. Yip, J. Cortés, and S. Herbert, “Back to Base: Towards Hands-Off Learning via Safe Resets with Reach-Avoid Safety Filters,” *ArXiv*, vol. abs/2501.02620, 2025.

[7] S. Bansal and C. J. Tomlin, “DeepReach: A Deep Learning Approach to High-Dimensional Reachability,” in *Proc. IEEE Conf. on Robotics and Automation*, 2021.

[8] K.-C. Hsu, V. Rubies-Royo, C. J. Tomlin, and J. F. Fisac, “Safety and Liveness Guarantees through Reach-Avoid Reinforcement Learning,” in *Robotics: Science and Systems*, 2021.

[9] O. So, Z. Serlin, M. Mann, J. Gonzales, K. Rutledge, N. Roy, and C. Fan, “How to Train Your Neural Control Barrier Function: Learning Safety Filters for Complex Input-Constrained Systems,” in *Proc. IEEE Conf. on Robotics and Automation*, 2024.

[10] D. P. Nguyen, K.-C. Hsu, W. Yu, J. Tan, and J. F. Fisac, “Gameplay Filters: Robust Zero-Shot Safety through Adversarial Imagination,” in *Conf. on Robot Learning*, 2024.

[11] R. Sinha, A. Sharma, S. Banerjee, T. Lew, R. Luo, S. M. Richards, Y. Sun, E. Schmerling, and M. Pavone, “A system-level view on out-of-distribution data in robotics,” 2022.

[12] L. Brunke, M. Greeff, A. W. Hall, Z. Yuan, S. Zhou, J. Panerati, and A. P. Schoellig, “Safe Learning in Robotics: From Learning-Based Control to Safe Reinforcement Learning,” *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 5, pp. 411–444, 2022.

[13] K. P. Wabersich, A. J. Taylor, J. J. Choi, K. Sreenath, C. J. Tomlin, A. Ames, and M. N. Zeilinger, “Data-Driven Safety Filters: Hamilton-Jacobi Reachability, Control Barrier Functions, and Predictive Methods for Uncertain Systems,” *IEEE Control Systems*, vol. 43, pp. 137–177, 2023.

[14] J. Borquez, K. Nakamura, and S. Bansal, “Parameter-Conditioned Reachable Sets for Updating Safety Assurances Online,” in *Proc. IEEE Conf. on Robotics and Automation*, 2022.

[15] M. Tayal and S. N. Y. Kolathaya, “Control Barrier Functions in Dynamic UAVs for Kinematic Obstacle Avoidance: A Collision Cone Approach,” in *American Control Conference*, 2023.

[16] M. Yu, C. Yu, M.-M. Naddaf-Sh, D. Upadhyay, S. Gao, and C. Fan, “Efficient Motion Planning for Manipulators with Control Barrier Function-Induced Neural Controller,” in *Proc. IEEE Conf. on Robotics and Automation*, 2024.

[17] R. Grandia, A. J. Taylor, A. Ames, and M. Hutter, “Multi-Layered Safety for Legged Robots via Control Barrier Functions and Model Predictive Control,” in *Proc. IEEE Conf. on Robotics and Automation*, 2020.

[18] A. Robey, H. Hu, L. Lindemann, H. Zhang, D. V. Dimarogonas, S. Tu, and N. Matni, “Learning Control Barrier Functions from Expert Demonstrations,” in *Proc. IEEE Conf. on Decision and Control*, 2020.

[19] M. Tayal, H. Zhang, P. Jagtap, A. Clark, and S. N. Y. Kolathaya, “Learning a Formally Verified Control Barrier Function in Stochastic Environment,” in *Proc. IEEE Conf. on Decision and Control*, 2024.

[20] L. Manda, S. Chen, and M. Fazlyab, “Learning Performance-oriented Control Barrier Functions Under Complex Safety Constraints and Limited Actuation,” in *Conf. on Robot Learning*, 2025.

[21] H. J. Jeong, Z. Gong, S. Bansal, and S. L. Herbert, “Parameterized Fast and Safe Tracking (FaSTrack) using Deepreach,” in *Learning for Dynamics & Control*, 2024.

[22] A. Lin and S. Bansal, “Verification of neural reachable tubes via scenario optimization and conformal prediction,” in *Learning for Dynamics & Control*, 2024.

[23] S. L. Herbert, J. J. Choi, S. Qazi, M. T. Gibson, K. Sreenath, and C. J. Tomlin, “Scalable Learning of Safety Guarantees for Autonomous Systems using Hamilton-Jacobi Reachability,” in *Proc. IEEE Conf. on Robotics and Automation*, 2021.

APPENDIX

A. Theoretical Background

Consider a system of the form

$$\dot{x} = \tilde{f}(x,u,d) = f(x) + g(x)u + d(x), \tag{8}$$

where $x \in \mathbb{R}^n$ is the state, $u \in \mathcal{U} \subseteq \mathbb{R}^p$ is the control input, $d \in \mathcal{D} \subseteq \mathbb{R}^q$ is the disturbance, and $\mathcal{U}$ and $\mathcal{D}$ are convex and compact sets. Throughout this work, we make the following assumption about the dynamics $\tilde{f}$.

Assumption 1: The function $\tilde{f} : \mathbb{R}^n \times \mathcal{U} \times \mathcal{D} \to \mathbb{R}^n$ is above bounded by $M_{\tilde{f}}$ and globally Lipschitz.

This assumption ensures the existence of a unique solution $\boldsymbol{x} : [t, 0] \to \mathbb{R}^n$ to (8) with $\boldsymbol{x}(t) = x$, which we denote by $\xi_{x,t}^{\boldsymbol{u},\boldsymbol{d}}$.

Recall that Hamilton–Jacobi reachability (HJR) formulates the problem as a differential game between two players: the control input u, which seeks to maximize the reward, and the disturbance d, which acts adversarially to minimize it. Then, the value function of this game is defined by (9) In general, solving the value function optimization problem in (9) is non-convex and therefore challenging. However, using dynamic programming (backwards in time), the value function V is the unique viscosity solution of the following Hamilton-Jacobi-Isaacs Variational Inequality (HJI VI)

$$\begin{aligned} 0 = \min\{g(x) - V(x,t), \max\{l(x) - V(x,t), \\ \frac{\partial}{\partial t}V(x,t) + H(\nabla V(x,t), x)\}\}, \end{aligned} \tag{10}$$

with terminal cost $V(x, 0) = \min\{l(x), g(x)\}$ and Hamiltonian defined by $H(\lambda, x) = \max_{u\in\mathcal{U}} \min_{d\in\mathcal{D}} \lambda^\top f(x,u,d)$. The gradients of this function enable the computation of the optimal safety control $u^*(x,t)$ such that

$$u^*(x,t) = \arg\max_{u\in\mathcal{U}} \min_{d\in\mathcal{D}} \nabla V(x,t)^\top \tilde{f}(x,u,d). \tag{11}$$

Value functions can be computed effectively via dynamic programming in low-dimensional systems by discretizing the state space to form a high-resolution grid. However, the reliance on grid-based discretization makes such methods scale exponentially with the system's state dimension, making these methods unsuitable for systems with more than $5 - 6$ state variables.

Constructing valid CBFs for complex systems with input bounds and disturbances is often challenging, especially when the safe set is difficult to characterize analytically. By leveraging reachability analysis, one can systematically synthesize CBF-like safety filters to ensure safety. For instance, HJR reach-avoid problems, with value function h_v, can be integrated with CBFs, leading to the following definition.

Definition A.1: (**Viscosity-Based Control Barrier Function [6]**) Consider a continuous function $h_v : \mathbb{R}^n \times (-\infty, 0] \to \mathbb{R}$, and for each $t \le 0$, let $\mathcal{C}_v(t) = \{x \in \mathbb{R}^n \mid h_v(x,t) \ge 0\}$. Then h_v is a viscosity-based control barrier function (VB-CBF) for system (8) on $\mathcal{C}_v(\cdot)$ if there exists an extended class $\mathcal{K}$ function α such that for all $t < 0$ and all $x \in \mathcal{C}_v(t)$, the inequality $\frac{\partial}{\partial t}h_v(x,t) + \max_{u\in\mathcal{U}} \min_{d\in\mathcal{D}} \nabla_x h_v(x,t)^\top \tilde{f}(x,u,d) \ge -\alpha(h_v(x,t))$ holds in a viscosity sense.

Given a nominal control policy u_{nom} that may violate input or safety constraints, viscosity-based control barrier functions enable minimal modification of the nominal input via a quadratic program. The resulting minimally invasive safety filter not only ensures constraint satisfaction but also guides the system away from unsafe regions and back toward the desired target set.

B. Learning-Based Reachability Analysis

Several learning-based approaches have been developed to approximate the reach-avoid reachability value function. Although our framework is compatible with a broad range of methods (including self-supervised learning [7] and reinforcement learning [8]), we implement our method using Deepreach:

Self-Supervised Learning of Reachability Value Functions To avoid solving HJI-VI with grid-based discretization, we employ a Physics-Informed Neural Network (PINN) to learn the value function used in minimally invasive safety filters. In particular, we leverage the DeepReach framework to approximate the safety value function by employing a sinusoidal deep neural network architecture [7]. Consequently, the computational and memory demands of training depend on the intrinsic complexity of the value-function approximation rather than on the grid resolution. For a reach-avoid problem, the loss function $\mathcal{L}$ used for training DeepReach is given by

$$\begin{aligned} \mathcal{L}(\theta) = \mathbb{E}_{z,t}\Bigg[\Bigg\| \min\Bigg\{ g(x) - V_\theta(z,t), \\ \max\Bigg\{ l(x) - V_\theta(z,t),\ \frac{\partial V_\theta}{\partial t} + H(\nabla_z V_\theta, z)\Bigg\}\Bigg\}\Bigg\|\Bigg], \end{aligned} \tag{12}$$

where $V_\theta(z,t) = \min\{l(x), g(x)\} - t \cdot \text{NN}_\theta(z,t)$, with $z = [x,p]$ the joint state, x the model state and p the parameterized state, which are discussed in more detail in Appendix D.

The loss function (12) involves nested $\min$ and $\max$ operations; These operations induce non-smooth behavior in the loss function This poses a challenge for neural networks, as learning relies on backpropagation and smooth gradient flow to update the model parameters effectively. Moreover, the smoothness of the learned value function is critical, as our safety filter relies directly on the gradient of the value function to ensure safe control inputs (motivating [7]'s use of sinusoidal activation functions, which we also employ).

To improve the learning process, we leverage a key observation relevant to our application; The target set characterizing the reach-avoid tube problem is control invariant. That is, once the drone reaches the target set, it can remain within this set and maintain safety under any level of disturbances. This property allows reformulating the learning objective by

$$V(x,t) = \min_{\mathbf{d}\in\mathbb{D}(t)} \max_{\mathbf{u}\in\mathbb{U}(t)} r_{\text{RA}}(x,t,\boldsymbol{u},\boldsymbol{d}) = \min_{\mathbf{d}\in\mathbb{D}(t)} \max_{\mathbf{u}\in\mathbb{U}(t)} \max_{\tau\in[t,0]} \min\{l(\xi_{x,t}^{\boldsymbol{u},\boldsymbol{d}}(\tau)), \min_{s\in[t,\tau]} g(\xi_{x,t}^{\boldsymbol{u},\boldsymbol{d}}(s))\}. \tag{9}$$

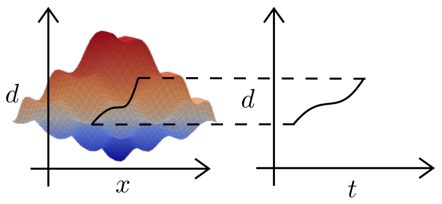


Fig. 6: Change in disturbance bounds over state x is encoded as change in bounds over time t. The disturbance bounds increase towards the red region in the left image. This is encoded as a temporal disturbance increase shown in the plot on the right.

shifting the focus from learning a value function to reach a target while avoiding unsafe regions over time, to instead learning or defining a control-invariant set and learning a value function to avoid unsafe regions over time, while only implictly encoding the reaching of the target in the boundary condition. Hence, the modified reach-avoid control-invariant loss function for DeepReach is given by

$$\mathcal{L}(\theta) = \mathbb{E}_{z,t}\left[\left\| \min\left\{ g(x) - V_\theta(z,t), \frac{\partial V_\theta}{\partial t} + H(\nabla_z V_\theta, z) \right\} \right\|\right], \tag{13}$$

where $V_\theta(z,t) = \min\{l(x), g(x)\} - t \cdot \text{NN}_\theta(z,t)$, like before. We empirically observe smoother gradients and a better-learned solution employing (13).

C. DeepReach Training Details

To evaluate our method and relevant baselines, we adopt a modified, parameterized version of DeepReach (DR) to learn the value function under different environmental conditions.

For the **DR Naive** baseline, the parameterized inputs correspond to the disturbance magnitude (applied to position and velocity in x and z directions in the 4D model and only to the velocity components in the x and z directions in the 6D model).

For the **DR ours**, the parameterized inputs correspond to the disturbance slope, which defines how the disturbance magnitude varies with the environment. This includes slopes over position and velocity disturbance magnitudes in the x and z directions in the 4D model and slopes for the velocity disturbance magnitude in the x and z directions in the 6D model.

Parameterized Value Function Using DeepReach. To incorporate these parameters, we modify DR following the approach in [14] to account for the environmental conditions, such as disturbance bounds or slopes, as part of the input space. We augment the system with the disturbance rate $\dot{d}$ as in (5) and compute the backward reachable tube for the resulting parameterized system using DR. Accordingly, the neural network takes as input the state $z = [x, \dot{d}]$ and time t, and outputs the corresponding value function $V_\theta(z,t)$, where θ denotes the network parameters. For the baselines we consider the joint state $z = [x, d]$, with d the disturbance magnitude. We generate training inputs using uniform sampling over both the state and parameter dimensions, covering the desired range of environmental conditions. The model is trained with default Deepreach settings, most importantly a batch size of 65k states, over 100k steps with a learning rate $\eta = 2e^{-5}$. It uses the default parameters of the Deepreach repository https://github.com/smlbansal/deepreach/tree/public_release. For the 4D simulation model, we use (12) for the loss, which had adequate performance for this setting. However, for the 6D hardware experiments, we adopt the reach-avoid control-invariant loss (13) which leads to a better solution.

Training Challenges. For the DR baselines, we initially aimed to learn a time-invariant always-avoid value function for the environment with varying maximum disturbance bounds. However, due to convergence difficulties, this approach did not yield a sufficiently performant solution. To ensure a fair comparison, we instead trained DR on a reach-avoid formulation toward the same target set as our method, while instead parameterizing over the disturbance bounds. This formulation enabled successful training and produced reliable value functions. During deployment, we evaluated the value function solely at the final time point, effectively considering an avoid-only value function, which allowed us to construct a time-invariant, minimally invasive safety filter.

Interpreting learned value functions Figures 7 and 8 visualize the 0-level sets of the learned value functions in the considered environment. Specifically, the left, center, and right figures in Fig. 7 showcase a varying disturbance magnitude level (from light blue to dark blue) for fixed low, medium and high disturbance rates. This showcases that even for high disturbance magnitudes as long as the disturbance rate is small (left) the safe region is relatively large, while for a high disturbance rates the safe region is much smaller (right).

Next, the left, center, and right figures in Fig. 8 showcase a varying disturbance rate (from light blue to dark blue) for fixed low, medium, and high disturbance magnitudes. This figures showcases that as long as the current disturbance magnitude is small (left) the safe region is relatively large even for high disturbance rates, while for high current disturbance magnitudes (right) the safe region is only slightly larger than the control invariant set (right).

D. Simulation Experiments

We consider a 4D drone dynamics model given by

$$\begin{bmatrix} \dot{p}_x \\ \dot{p}_z \\ \dot{v}_x \\ \dot{v}_z \end{bmatrix} = \begin{bmatrix} v_x + d_1 \\ v_z + d_2 \\ g u_1 + d_3 \\ u_2 - g + d_4 \end{bmatrix}, \tag{14}$$

where the control input is denoted by $[u_1,\ u_2]^\top$ and the state vector is given by $[p_x,\ p_z,\ v_y,\ v_z]^\top$, with p_y and p_z denoting the position of drone, and v_x and v_z representing the corresponding velocity components[3]. The disturbances d_1, d_2, d_3 and d_4 represent wind, and g is gravity. We choose this low-dimensional model to facilitate comparison with classical methods, which are less prone to approximation errors inherent in learned approaches. Moreover, it allows us to demonstrate that our method generalizes across various strategies for computing the value function. While wind

[3]In the main body of the work we consider the state $[x, z, \dot{x}, \dot{z}]$ which overloads the notation on $x$therefore consider position p and velocity v with subscripts for its components

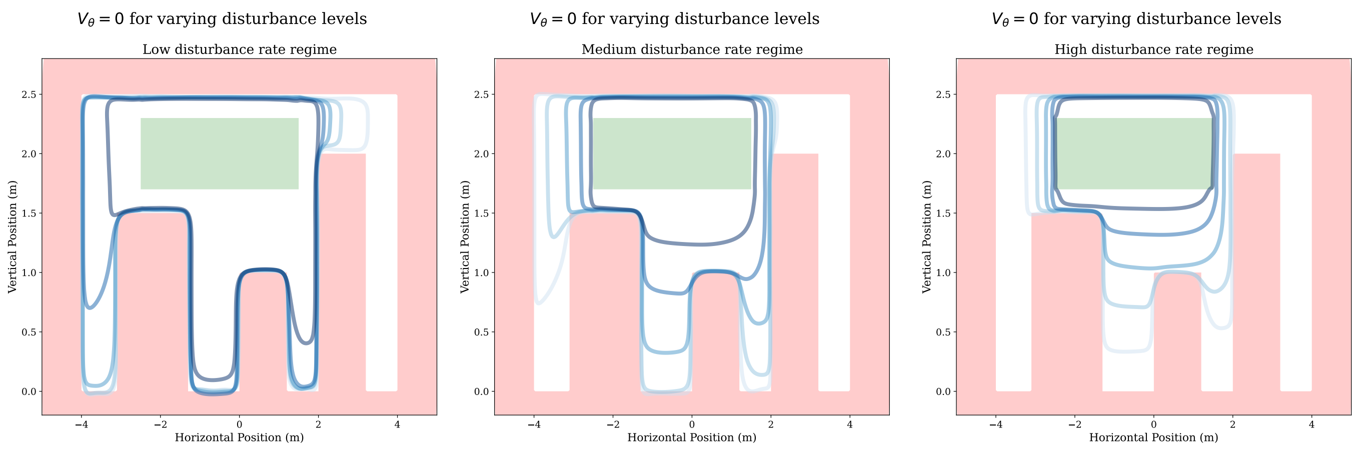


Fig. 7: The 0-level set of the learned value function for a fixed disturbance rate $\dot{d}$ for increasing levels (light to dark blue) of disturbance d. Left-to-right visualizes a low fixed disturbance rate, a medium disturbance rate, and a high disturbance rate respectively. This is evaluated for the $v_x = 0, v_z = 0$ slice. $\dot{d}$ is encodes through the parameterized state, whereas d is encoded through the time slice of the value function with $t = (d_{\max} - d)/\dot{d}$

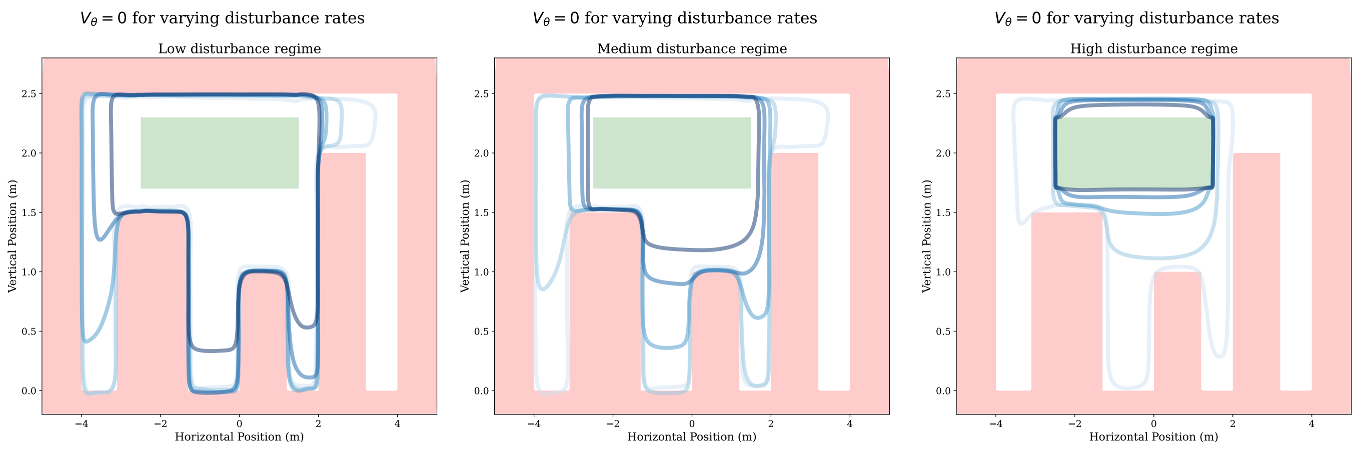


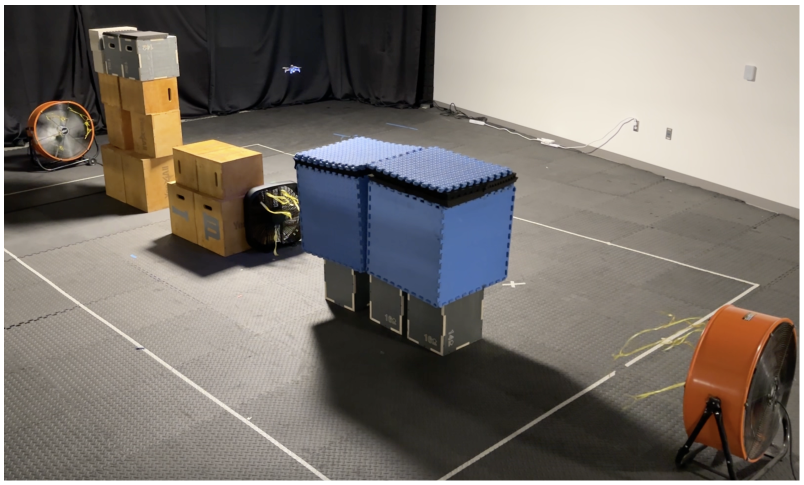
Fig. 8: The 0-level set of the learned value function for a fixed disturbance level d for increasing (light to dark blue) disturbance rates $\dot{d}$. Left-to-right visualizes a low fixed disturbance level, a medium disturbance level, and a high disturbance level respectively. This is evaluated for the $v_x = 0, v_z = 0$ slice. $\dot{d}$ is encodes through the parameterized state, whereas d is encoded through the time slice of the value function with $t = (d_{\max} - d)/\dot{d}$.

typically only affects the velocity components of the dynamics, we include the positional disturbances to circumvent the CBF (and value function) from learning to cancel out the disturbances directly in the control input (and thus effectively just reducing the control input bounds). Instead, by considering positional disturbances, the problem is interesting (while remaining tractable to solve with HJR methods).

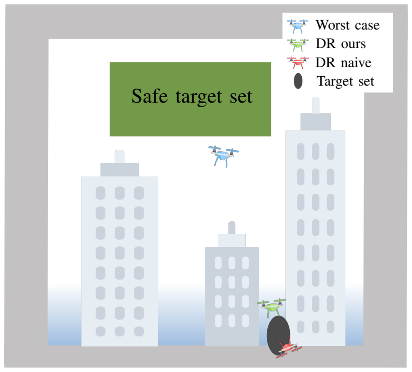
Environment Setup. We design a city-like environment with multiple tall buildings that the drone must avoid during its operation. The environment is defined in between $p_x \in [-5, 5]$ and $p_z \in [-0.2, 2.8]$, and includes three rectangular buildings located at: $(p_x, p_z) \in [-3.1, -1.3] \times [0, 1.5]$, $(p_x, p_z) \in [0.0, 1.2] \times [0.0, 1.0]$, and $(p_x, p_z) \in [2.0, 3.2] \times [0.0, 2.0]$. A spatial boundary restricts the drone to remain within $p_x \in [-4.0, 4.0]$ and $p_z \in [0.0, 2.5]$, and the velocity of the drone is constrained within $[-1.9, 1.9]$ for both v_x and v_y. A safe target region, representing a designated flyover corridor above the cityscape, is defined as a rectangular set with $(p_x, p_z) \in [-2.5, 1.5] \times [1.7, 2.3]$. To render the set approximately control invariant, both v_x and v_z are restricted to the range $[-1.0, 1.0]$ within the target region.

The wind intensity is modeled as an exponential function of the drone height, z, with wind intensifying near the ground in an urban canyon. These wind effects are represented as direct disturbances on position and velocity in system dynamics f on both x and z.

The dynamics are simulated at 40 Hz, while environmental disturbances are updated at 4 Hz using our proposed adaptive strategy for $H = 10$ to ensure robustness for real-world disturbance estimation. All obstacles and boundaries are indicated in red while the target region is depicted in green in Figure 10. Wind is modeled as a deterministic disturbance acting within the urban canyons formed between buildings, with a fixed direction given by the vector $[1, -1]$ in directions x and



(a) Real-world setup with obstacles and unknown disturbances



(b) Simulation environment

Fig. 9: Fig. 9a illustrates our real world experiment setup using crazyflies in an optitrack arena made to mimic the cityscape visualized in our simulation environment shown in Fig. 9b. Fig. 9b illustrates the cityscape simulation environment along with simulation results. The blue color gradient indicates increasing wind disturbance in the urban canyons. The drones in the figure show a comparison of our method vs. baselines when attempting to go to the goal region denoted by the dark gray oval.

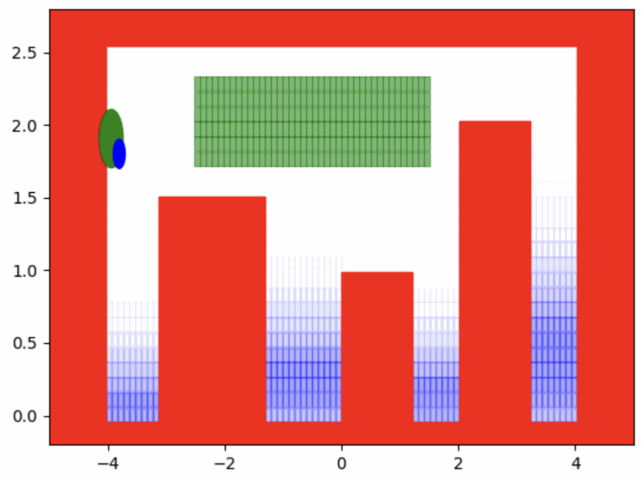


Fig. 10: Environment setup used in our simulations. This environment shows a quadcopter flying in a 2D p_x, p_z slice where p_x is the horizontal axis and p_z is the vertical axis. The quadcopter is denoted using a blue oval, while a current goal is shown using a green oval. All the obstacles in the environment, that denote unsafe regions for the quadcopter to enter, are shown in red. The safe control invariant target set that is used for our method is the translucent rectangle in free space shown in green. The wind between the urban canyon is shown in blue, where the darkness of the color corresponds to increased wind magnitude. For our experiments all wind disturbance is in the positive x direction and negative z direction.

z. The magnitude of the wind is bounded and increases toward the bottom of the canyons, capturing the channeling effects of urban geometry. The wind is shown in blue in Figure 10. The darkness of the blue corresponds to the wind magnitude which increases as we descend (lower p_z). We model the increase in wind magnitude using an exponential function defined as

$$f(W_L) = D \cdot \mathbf{W} \cdot \left(1 - \exp\left(-\frac{r \cdot (W_L - W_{\text{min}})}{(W_{\text{max\_pos}} - W_{\text{min}})}\right)\right) \quad (15)$$

where D denotes the disturbance factor. $\mathbf{W}$ is the maximum wind value magnitude. r is the exponential ramp rate. W_L is wind location which denotes the state of the robot where the wind is to be evaluated.W_{min} is the lower bound of the state range where the wind is defined. $W_{\text{max\_pos}}$ is the position of the maximum wind value. This wind function defines the magnitude of the wind between $W_{\text{min}}, W_{\text{max}}$ and the wind magnitude is 0 outside these bounds. These wind functions are defined for every state dimension and composed.

This function increases the wind magnitude to the pre-known maximum at a set height. For the experiments shown in Table I, in the main body of the paper, this height is fixed at $W_{\text{max\_pos}} = 0.1$, and the exponential growth rate of each wind field is set to $r = 5$. The resulting maximum wind magnitude is $[0.75, 1.5]$ for the position and velocity disturbances in the HJR experiments, and $[0.75, 1.0]$ in the DR experiments.

Value Function Learning. In simulation, we compare both the HJR and DR variants of our method against corresponding HJR and DR implementations of a baseline approach. Below, we outline the specific setup used for each method.

DR Ours. The DeepReach network is trained to approximate the reach-avoid value function over a time horizon from 0.0 to 5.0 seconds. The training time is approx. 2 hours.

DR Naive. As discussed in Section B, due to challenges in obtaining a good performing value function for the avoid-only problem, we train DR to solve a reach-avoid problem, over a time horizon from 0.0 to 5.0 seconds. The training time is approx. 2 hours.

HJR Ours We compute the value function using dynamic programming via the JAX-based HJ Reachability Toolbox, https://github.com/StanfordASL/hj_reachability. The

TABLE II: Comparison of Our approach against baselines using HJR and Deepreach based value functions. Metrics are generated over 100 trajectories, with 15 goals and 1500 control steps each. The trajectories are run over randomly generated environment where the exponential rate for the wind as well as the position of the max wind is varied for each wind field. Each method is evaluated across the same 100 random environments for a fair comparison. % Safety Violations indicates the percentage of failed trajectories, Mean Goal Dist reflects the average minimum distance to each goal with lower values indicating better goal completion, and Mean Traj Len refers to the mean trajectory length before failure with a max of 1500. The table highlights how SPACE2TIME provides the greatest balance between safety and performance of the compared methods.

Approach	% Safety Violations $\downarrow$	Mean Goal Dist $\downarrow$	Mean Traj Len $\uparrow$
HJR Naive	100%	1.13	393.03
HJR Naive Worstcase	0%	2.09	1500
DeepReach Naive	93%	1.15	533.70
DeepReach Ours	60%	1.00	961.31
HJR Ours	58%	0.96	985.21

extended dynamics system, including disturbances, is of dimension 8 and therefore cannot be computed directly using grid-based dynamic programming (limited to $4-5$ dimensions on a GPU, 6 dimensions on a CPU, albeit taking hours to solve). Therefore, to account for varying environmental conditions, we precompute 5 value functions corresponding to disturbance slope magnitudes evenly spaced between zero and the maximum disturbance magnitude (with equivalent relative size for each disturbance dimension). During runtime, the safety filter associated with the smallest precomputed slope bound that exceeds the detected slope magnitude is selected and applied.

HJR Naive This approach uses the same dynamic programming framework as **HJR Ours**. In this case, the five precomputed value functions correspond to different maximum disturbance magnitudes, evenly spaced between zero and the upper bound. At runtime, the safety filter associated with the smallest disturbance bound that exceeds the detected disturbance magnitude is selected for execution.

HJR Worst Case This method computes a single safety value function using dynamic programming, based on the worst-case disturbance bounds observed in the environment.

Experimental Results. We evaluate performance across 50 trajectories, each consisting of 10 random goals, with the drone navigating through the city environment under the wind, as summarized in Table I in the main paper. The drone is controlled using a naive nominal controller based on Linear Quadratic Regulation (LQR), which does not account for obstacles in the environment. Goals are generated in a cyclical manner every 100 time steps to ensure diversity in target locations while remaining reachable by the nominal controller. Specifically, the first goal is placed above a randomly selected urban canyon, the second is set near the bottom of that same canyon, and the third is placed near the top. This three-step cycle is repeated to yield 10 goals per trajectory.

Here, we provide the additional experimental results. In this setting, each trajectory consists of 15 goals generated using the same cyclical procedure described previously. To further evaluate generalization, the environment is randomized across trajectories. For each wind field, the exponential rate governing the wind magnitude profile is sampled uniformly between 3 and 7, while the height at which the wind reaches its maximum is selected randomly between 0.1 and one-third of the wind field height. Results averaged over 100 randomized trajectories are shown in Table II. We visualize all of the trajectories pertaining to our method in Figure 11 and all the baseline trajectories in Figure 12. From these figures we see the major impact of our method, where by properly reasoning about the potential increase in spatial disturbances our method does not descend far in the urban canyons and thus remains safe. Meanwhile, the naive baselines fail to reason about the spatially changing disturbances and ends up crashing (therefore cutting their trajectories short). This is also visible by looking at the density of the trajectory traces, where our ability to fly safely for longer is visualized through denser trajectory traces compared to the baselines that fail. The HJR Worst Case baseline, while safe, ends up being too conservative and is confined to a very small region of the environment.

E. Hardware experiments

This hardware experiment setup discussion supplements the discussion in the main paper, Sec. VI. The hardware experiments are conducted in a planar $x-z$ plane, with the following state range $p_x \in [-4.0, 4.0]$ and $p_z \in [0.0, 2.0]$. On a quadcopter, unlike in simulation, we cannot set the pitch rate θ_x directly. Instead, we consider a cascaded model, inspired by [23], with as inputs the desired pitch rate S_x and the combined thrust T. The control inputs for the full system are S_x, S_y, $\dot{\psi}$, and T, with S_y the desired roll rate and $\dot{\psi}$ the yaw rate. The nominal controller is an LQR-based controller for $x_{\text{LQR}} = [p_x, v_x, p_y, v_y, p_z, v_z, \psi]$. Then, $S_{x,\text{nom}} = K_{p_x}(p_x - p_{x,\text{goal}}) + K_{v_x} v_x$, $S_{y,\text{nom}} = K_{p_y}(p_y - p_{y,\text{goal}}) + K_{v_y} v_y$, $\dot{\psi}_{\text{nom}} = K_\psi(\psi - 0)$, and $T_{\text{nom}} = K_{p_z}(p_z - p_{z,\text{goal}}) + K_{v_z} v_z$. Specifically, we set $K_{p_x} = K_{p_y} = K_{v_x} = K_{v_y} = -0.2$, $K_\psi = -20.0$ and $K_{p_z} = K_{v_z} = -10.0$ As our experiments are planar the nominal model considered for our CBF is $u_{\text{nom}} = [S_{x,\text{nom}}, T]$. The nominal desired roll attitude and yaw rate are directly fed into the system, with the objective of keeping $p_y = p_{y,\text{goal}} = 0.0$ throughout the trajectory and keeping $\psi = 0$. We consider a 6D drone dynamics model given by

$$\begin{bmatrix} \dot{p}_x \\ \dot{v}_x \\ \dot{\theta}_x \\ \dot{\omega}_x \\ \dot{p}_z \\ \dot{v}_z \end{bmatrix} = \begin{bmatrix} v_x \\ g\tan(\theta_x) + d_x - c_x v_x \\ -d_1\theta_x + \omega_x \\ -d_0\theta_x + n_0 S_x \\ v_z \\ k_T T_z - g + d_z \end{bmatrix} \tag{16}$$

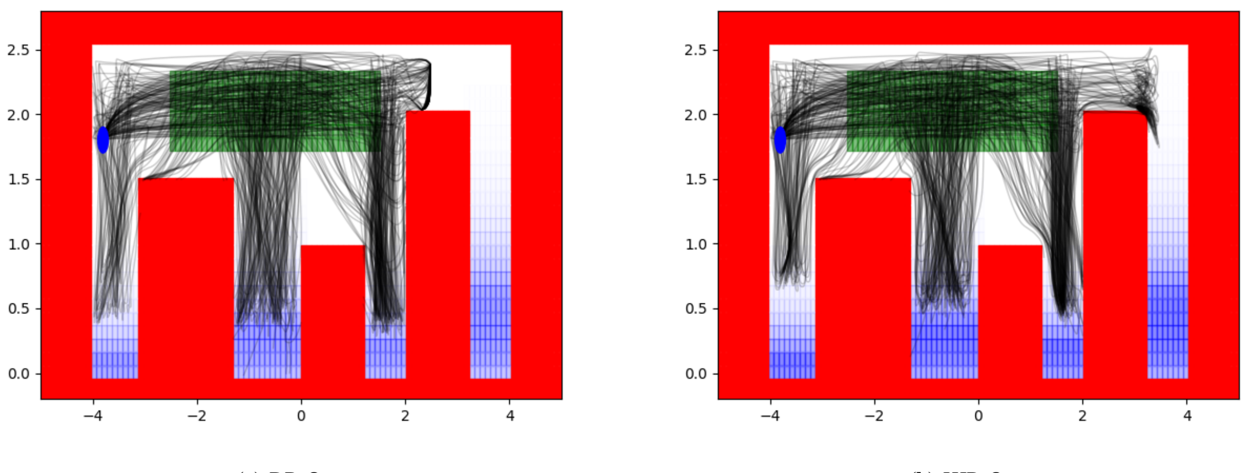

(a) DR Ours **(b)** HJR Ours

Fig. 11: Our Method Trajectories: Figures showing 100 of our trajectories moving between 15 random goals through the city environment with randomized wind fields. Notice that our safety filter prevents a majority of our trajectories from going too far down the urban canyon in response to increasing disturbance slope estimates. This leads to safer trajectories that fail less often. Fig. 11a (on the left) shows these trajectories using our method with a learned value function using Deepreach with the 4 dimensional quadcopter model. Fig. 11b shows these trajectories using our method with value functions computed using HJR with the 4D quadcopter model.

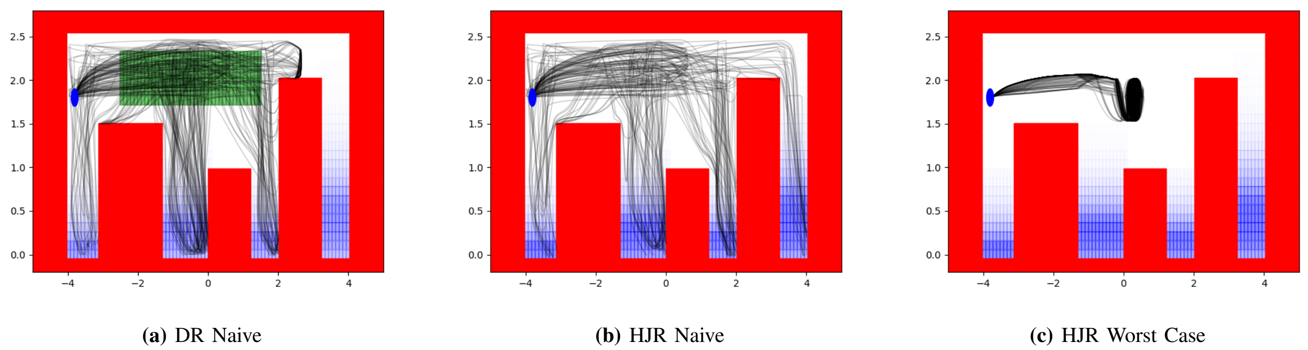

(a) DR Naive **(b)** HJR Naive **(c)** HJR Worst Case

Fig. 12: Baseline Trajectories: Figures showing 100 of our trajectories moving between 15 random goals through the city environment with randomized wind fields. Fig. 12a (on the left) shows trajectories from the naive parameterized baseline using a value function learned with deepreach. Fig. 12b (in the center) shows trajectories from the naive parameterized baseline using a value function computed using HJR. Notice that these trajectories fail to properly consider the unknown spatially varying disturbance and as a result go much further down the canyon and end up crashing. Notice that compared to our method showin in Figures 11a and 11b the trajectory tracks are less dense. This is as a majority of the naive trajectories are cut short by the drones crashing whereas our method is able to continue for much longer. Finally Fig. 12c shows the trajectories when using the HJR computed safety value function for the worst case disturbance bounds. Only considering the worst case results in extremely conservative, non-performant trajectories that fail to move beyond a small set away from the obstacles.

where the desired pitch is given by S_x and the state vector is given by $[p_x,\ v_x,\ \theta_x,\ \omega_x,\ p_z,\ v_z]^\top$, with p_x and p_z denoting the position of drone, v_x and v_z representing the corresponding velocity components, θ_x denoting pitch, and ω_x denoting pitch rate. The disturbances d_x and d_z represent wind, and g is gravity. Extending upon [23], we consider a drag term which we found to be necessary to achieve a good model fit. The parameters c_x, d_1, d_0, n_0, k_T that we used were fit using system-identification on a 1 minute trajectory with random setpoints in $[p_x, p_y, p_z]$ updated every 6 seconds, and are $c_x = 0.3$, $d_1 = 4.5$, $d_0 = 20.0$, $n_0 = 18.0$, and $k_T = 0.83$.

Environment Setup. Our hardware experiments are conducted in an OptiTrack motion capture arena for precise state estimation. To emulate the simulation conditions, we construct a mock urban environment consisting of three tower-like obstacles built from stacked boxes. However, due to the space constraints of the flight arena the boundary's and each obstacle's x positions are scaled by a factor of 0.8 and z positions are scaled by a factor of 0.75. For the results shown in Figure 4, we introduce artificial disturbances into the state measurements to mirror the simulated dynamics. Specifically, we use the OptiTrack system to measure the drone's state in real time, evaluate the corresponding disturbance (using the same wind function employed in our simulation experiments), and add the resulting values into the velocity components along the x and z axes, v_x and v_z. This setup effectively spoofs the model to think it is perturbed by actual wind, thus allowing for consistent (non-turbulent) airflow, while mimicking the wind profile of urban canyons. This setup enables consistent comparison between our proposed method and baselines under equivalent disturbance profiles. The fans are placed in the environment for conceptual visualization only. The reasons for not using fans are two-fold: 1) The airflow profile of radial fans causes a very rapid increase in wind disturbance at the edge of the fans, destabilizing the drone. 2) Measuring the wind disturbance without an airflow sensor is difficult and relying on single-step error measurements from the Optitrack system has too much variance to provide useful estimates. The drone platform itself has very limited compute and is purely used for state estimation using its IMU (in combination with the external Optitrack system) and sending input commands.

Value Function Training. We evaluate different strategies for learning value functions under varying environmental conditions and disturbance models:

DR Ours. We train a reach-avoid value function for the 6D quadrotor model in the urban environment, parameterized by the disturbance slope affecting the x and z velocity components, v_x and v_z. The environment includes time-varying dynamics, with maximum disturbance magnitudes of 0.75 in both velocity directions, and maximum disturbance rate of 1.5 in both velocity directions. The time horizon is from 0.0 to 5.0 seconds and the value function is learned using the reach-avoid control-invariant loss (13).

DR Naive. As in the simulation experiments, directly learning an avoid-only safety value function leads to poor convergence and suboptimal performance. Instead, we train a reach-avoid value function under time-invariant dynamics with maximum disturbance magnitudes of 0.75 in both velocity directions. The time horizon is from 0.0 to 5.0 seconds and the value function is learned using the reach-avoid control-invariant loss (13).

F. Algorithmic design details

Here, we provide implementation details of Algorithm 1. Specifically, L2 computes $d, \dot{d}$. We ensure that our environments do not exceed $d_{\max}$ and $\dot{d}_{\max}$, however, we clamp the values to $[0, d_{\max}]$ and $[0, \dot{d}_{\max}]$ respectively to ensure we do not extrapolate beyond data observed in training. Then, given $d, \dot{d}$ we compute t_{return} in L3, which again is clamped to $[0, t_{\max}]$ to ensure the input to the neural network is in distribution. Lastly, if $\dot{d} = 0$ we instead set $\dot{d} = (d_{\max} - d)/t_{\max}$, as setting $\dot{d} = 0$ effectively corresponds to considering $d = d_{\max}$ for all time (thus recovering the most-conservative value function).