2026 IEEE International Conference on Robotics and Automation (ICRA 2026)
June 1-5, 2026. Vienna, Austria

Performance-Guided Refinement for Visual Aerial Navigation using Editable Gaussian Splatting in FalconGym 2.0

Yan Miao[1], Ege Yuceel[1], Georgios Fainekos[2], Bardh Hoxha[2], Hideki Okamoto[2] and Sayan Mitra[1]

***Abstract*— Visual policy design is crucial for aerial navigation. However, state-of-the-art visual policies often overfit to a single track and their performance degrades when track geometry changes. We develop *FalconGym 2.0*, a photorealistic simulation framework built on Gaussian Splatting (GSplat) with an *Edit API* that programmatically generates diverse static and dynamic tracks in milliseconds. Leveraging FalconGym 2.0's editability, we propose a *Performance-Guided Refinement (PGR)* algorithm, which concentrates visual-policy training on challenging tracks while iteratively improving performance. Across two case studies (fixed-wing UAVs and quadrotors) with distinct dynamics and environments, we show that a single visual policy trained with PGR in FalconGym 2.0 outperforms state-of-the-art baselines in generalization and robustness: it generalizes to three unseen tracks with 100% success *without* per-track retraining and maintains higher success rates under gate-pose perturbations. Finally, we demonstrate zero-shot sim-to-real transfer of the PGR-trained visual policy to quadrotor hardware, achieving a 98.6% success rate (69/70 gates) over 30 trials across two three-gate tracks and one moving-gate track.**

I. Introduction

Visual aerial navigation is critical for applications such as mapping, search-and-rescue, environmental monitoring, and racing. Recent progress in photorealistic simulation environments has fueled zero-shot sim-to-real success for visual aerial navigation. Notably, SOUS VIDE [1] used Gaussian Splatting (GSplat) [2] to reconstruct an indoor lab and achieved zero-shot sim-to-real navigation; FalconGym [3] used NeRF [4] to build digital twins of three racing tracks and demonstrated zero-shot sim-to-real transfer of quadrotor gate crossing via imitation learning; Geles *et al.* [5] reported strong sim-to-real performance with a vision-only asymmetric actor-critic on three quadrotor racing tracks. While effective, these benchmarks face generalization limits: all three achieve high performance on their training tracks but do not generalize to unseen tracks (as we also confirm in Section IV), restricting broader applicability. GRaD-Nav [6] improves robustness to different gate positions and background distractors with a single policy, but still relies on constructing multiple GSplat tracks for training.

In this paper, we focus on developing a single visual policy that can traverse a family of tracks, where each track consists of an ordered sequence of tight racing gates, as shown in Figure 1 and Figure 4. To enable such cross-track generalization from limited real data, we develop *FalconGym 2.0*. From minutes of real-world video, FalconGym 2.0 generates arbitrarily many synthetic yet photorealistic tracks without additional data collection. As shown in Figure 3, our *Edit API* programmatically edits GSplat gate placements in milliseconds and also supports 4D time-varying simulation with moving gates (Section IV). Although we focus on aerial gate navigation, the same API extends to broader robotics tasks (e.g., obstacle placement for ground robots).

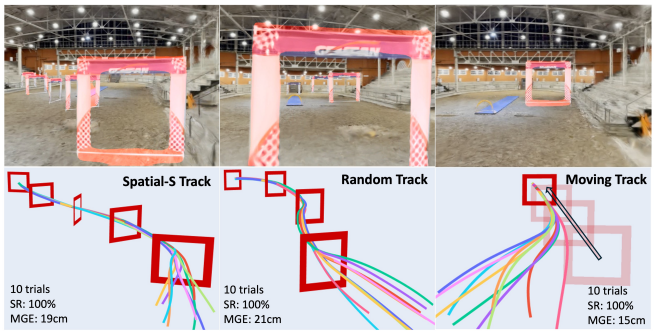

Fig. 1: **Trajectories for UAV case study in FalconGym 2.0 across three unseen tracks (Spatial-S, Random and Moving).** Top row: red overlays visualize predicted gate masks (Section III-B). Bottom row: 10 trials per track from different initial states; the translucent gates in the Moving track show the gate's past positions.

FalconGym 2.0's ability to generate arbitrarily many photorealistic tracks enables a range of visual-policy training strategies, from active learning to curriculum learning [7]. We develop a *Performance-Guided Refinement (PGR)* algorithm that: (i) identifies challenging tracks where the visual policy underperforms; (ii) synthesizes similar yet slightly different challenging tracks with the Edit API to augment the training dataset; and (iii) iteratively refines the visual policy via imitation learning from a state-based expert.

Beyond the Edit API and the PGR algorithm it enables, FalconGym 2.0 improves over FalconGym [3] by (i) replacing NeRF rendering with fast GSplat to accelerate training and (ii) eliminating expensive motion capture with an accessible ArUco marker for world-frame simulation reconstruction.

Through two case studies (fixed-wing UAVs and quadrotors) with different dynamics, environments and gate geometries, we show that our visual policy trained with PGR can generalize to three unseen tracks in FalconGym 2.0 with 100% success. Moreover, we outperform state-of-the-art visual baselines [3], [5] in both generalization and robustness: our single visual policy operates across three

This research was carried out at Toyota Motor North America R&D during Yan Mao's internship, while experiments were conducted at UIUC.

[1] Yan Miao, Ege Yuceel and Sayan Mitra are with the Department of Electrical and Computer Engineering, University of Illinois at Urbana Champaign {yanmiao2, eyceel2, mitras}@illinois.edu

[2] Georgios Fainekos, Bardh Hoxha and Hideki Okamoto are with Toyota Motor North America R&D {firstname.lastname}@toyota.com

unseen tracks, where baselines require separate per-track policies, and we maintain higher performance under gate-pose perturbations. Finally, the visual policy trained with PGR in FalconGym 2.0 can zero-shot transfer to quadrotor hardware, achieving a 98.6% success rate (69/70 gates) in 30 hardware trials spanning two three-gate tracks and one moving-gate track, as shown in Figure 6.

In summary, our contributions are: (1) *FalconGym 2.0*: an editable GSplat-based photorealistic simulator for fast world-frame environment modification. (2) *Performance-Guided Refinement (PGR)*: an algorithm that leverages editability to target challenging tracks and iteratively improve the visual policy. (3) *Zero-shot sim-to-real transfer*: a visual policy trained with PGR that zero-shot transfers to real quadrotor hardware and traverses three unseen tracks.

II. Related Work

a) Sim-to-Real in Robotics: Sim-to-real transfer is a longstanding goal in robotics, due to its efficient and safe training before deployment. With advances in photorealistic scene reconstruction like NeRF [4] and 3D Gaussian Splatting (GSplat) [2], recent robotics work increasingly trains in high-fidelity simulators. NeRF2Real [8] and RialTo [9] demonstrate NeRF-to-real transfer for humanoid navigation and robot manipulation, respectively. Vid2Sim [10] converts real-world videos into interactive simulators for urban navigation using GSplat reconstruction, while RoboSplat [11] and Splat-MOVER [12] leverage GSplat for domain randomization to improve robot manipulation performance.

In aerial navigation, FalconGym [3] constructs NeRF-based digital twins of racing tracks and achieves zero-shot sim-to-real quadrotor gate crossing via imitation learning, while SOUS VIDE [1] uses GSplat to build an indoor digital twin and also reports zero-shot transfer. Geles *et al.* [5] train a vision-only policy using async actor-critic and demonstrate high-speed racing ($40\,\mathrm{km/h}$) across three tracks. Despite strong sim-to-real results, these benchmarks generally require per-track training and struggle to generalize to unseen tracks.

b) Policy Refinement by Environment Shaping: Curriculum-based environment shaping [7] has been used to improve control policies in robotics by adaptively shaping training environments. It has been successful in simulation for RL agents in bipedal walkers [13]. [14] uses a similar "environment policy" idea for quadrotor navigation. However, these works use state-based control and do not involve rendering or visual policies, whereas our PGR algorithm adapts an editable photorealistic GSplat environment to improve visual-policy performance for aerial navigation.

c) Editable Gaussian Splatting: Recent work uses editable GSplat to generate synthetic yet photorealistic environments. GaussianEditor [15] enables text-guided appearance and geometry edits of objects in GSplat. VCEDIT [16] ensures multi-view consistency when using diffusion-guided GSplat edits. Instruct-GS2GS [17] provides instruction-based editing of GSplat scenes via 2D diffusion. While these works advance GSplat scene editing in computer vision, they focus on photorealism and rendering quality rather than downstream closed-loop robotics. In contrast, FalconGym 2.0 provides open-source Edit APIs for object editing in GSplat and couples this editability with performance-guided refinement to iteratively improve closed-loop visual-policy performance.

III. Methods

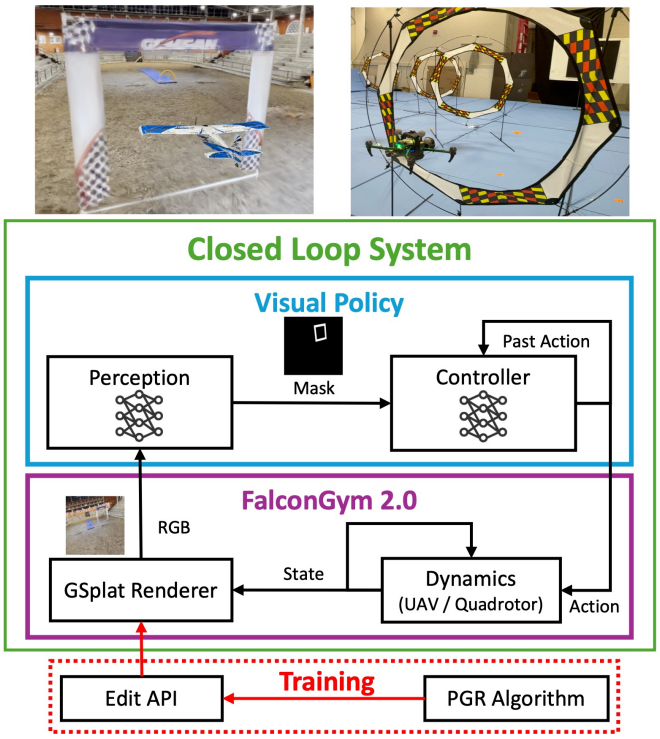

Fig. 2: **Closed-loop system in FalconGym 2.0:** we provide dynamics for a fixed-wing UAV and a quadrotor, and a GSplat renderer that produces photorealistic RGB from arbitrary camera poses in either scene. At each timestep, the dynamics propagate the state, the renderer generates an RGB image, a perception module predicts a gate mask, and a controller consumes the mask plus past actions to predict the next action. During training, a *Performance-Guided Refinement (PGR)* algorithm (Section III-C) focuses training on challenging tracks generated using *Edit API* (Section III-A)

In this paper, we consider visual aerial navigation in the gate-based track setting, where each track is an ordered sequence of tight racing gates and the aerial vehicle must safely traverse a set of tracks using only visual feedback. Successful trajectories are shown in FalconGym 2.0 for a fixed-wing UAV (Figure 1) and a quadrotor (Figure 4), and on hardware for the quadrotor (Figure 6).

Prior work [3], [5] tackles this same aerial navigation problem and reports strong gate-crossing performance; however, these methods require *per-track* training and a visual policy trained on one track does not generalize reliably to unseen tracks. This overfitting could be costly in practice, since recollecting data and retraining for every new track is time-consuming. Therefore, our goal is to develop a single visual policy that operates zero-shot on unseen tracks, eliminating per-track retraining while maintaining robustness to track-configuration changes. In this paper, we restrict the tracks

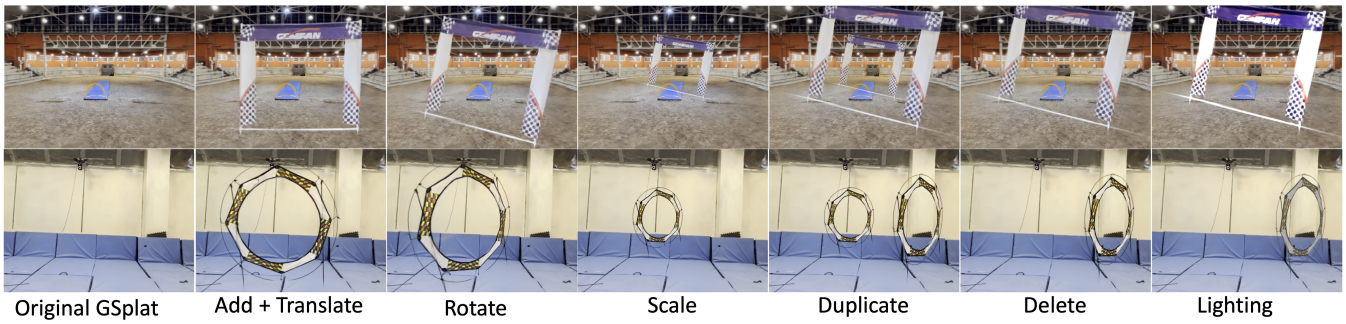

Fig. 3: **Edit API in FalconGym 2.0.** Our Edit API (Section III-A) provides world-frame programmatic placement of objects while the backend handles all coordinates and camera-to-world transforms. The seven APIs, `add`, `translate`, `rotate`, `scale`, `duplicate`, `delete`, and `lighting`, allow users to modify object pose, size, and appearance to generate a photorealistic 4D simulation environment. Shown are gate edits across two environments. This editability enables the PGR algorithm to improve visual-policy performance (Section III-C).

to be dynamically feasible and observable (when the aerial vehicle crosses current gate center orthogonally, the next gate must be within the camera's field of view). This observability requirement that keeps the next gate in sight is reasonable in a visual-only setting without knowledge of the map.

Our method comprises three components. First, we present *FalconGym 2.0* (Sec. III-A), a GSplat-based photorealistic simulation framework coupled with an *Edit API* that enables dynamic gate placement, extending prior static simulators [1], [3]. Second, we design a visual policy (Section III-B) in the closed loop in FalconGym 2.0 that mitigates the overfitting observed in earlier work [3] through modular architecture that separates perception and control. Third, we introduce a novel *Performance-Guided Refinement (PGR)* algorithm (Section III-C) that uses FalconGym 2.0's editability to expose visual policy to challenging tracks and iteratively refine its performance.

A. FalconGym 2.0: Editable GSplat

We propose a photorealistic simulation framework, *FalconGym 2.0*, that enables developing and testing different visual policies while improving on FalconGym [3] by replacing NeRF with GSplat for faster rendering and substituting motion capture with an ArUco marker for world-frame alignment. [1] More importantly, FalconGym 2.0 introduces an *Edit API* capable of generating arbitrarily many training tracks.

a) GSplat Scene representation: A trained GSplat scene consists of a set of N anisotropic Gaussians, i.e. $\mathcal{S} = \left\{ (\mu_j, \Sigma_j, c_j, \alpha_j) \right\}_{j=1}^{N}$, where $\mu_j \in \mathbb{R}^3$ is the mean; $\Sigma_j \in \mathbb{R}^{3\times3}$ is the covariance, parameterized via a rotation $R_j \in \mathrm{SO}(3)$ and per-axis scales $s_j \in \mathbb{R}^3_{>0}$ (i.e., $\Sigma_j = R_j \,\mathrm{diag}(s_j^2) R_j^\top$); $c_j \in \mathbb{R}^3$ is the color; and $\alpha_j \in [0,1]$ is the opacity. For rendering an image as seen by a camera in this scene $\mathcal{S}$, the N 3D Gaussians are projected onto the camera's image plane and then their colors are blended according to their depths relative to the camera.

b) Edit API: Because NeRFStudio [20] (the library we used to train GSplat) optimizes scenes in an internal coordinate frame, programmatic editing is inconvenient for users who reason in a world frame. FalconGym 2.0 instead exposes world-frame edits and handles all coordinate conversions and camera-to-world transformations in the backend. To edit a scene in FalconGym 2.0, users can first select an object either via predefined Gaussian IDs (e.g., gate primitives from another GSplat scene provided by us) or via user-defined world-frame bounding boxes, then apply pose, color and scale edits via our API. Concretely, we provide seven composable operations, as shown in Figure 3: `add()` inserts objects from another GSplat scene; `translate()` moves selected objects to user-defined poses; `rotate()` rotates the selected objects around their centers with user-defined rotations; `scale()` scales the selected objects' sizes; `duplicate()` clones the selected object; `delete()` removes selected objects; `lighting()` adjusts Gaussian-level colors for selected objects. Each operation updates the selected objects' corresponding Gaussians' μ_j, Σ_j (via R_j, s_j), c_j, and α_j on the backend. Since each API is a parallelizable tensor operation, each edit operation takes ~ 0.004 s on average with an RTX 4090 GPU. Users can freely combine any of the seven APIs to accomplish complex edits and build customized 4D simulations (e.g., moving gates in Section IV). Although our experiments focus on aerial navigation with gates, we expect the same Edit API to work on broad robotics tasks that would benefit from photorealistic editable GSplat scenes.

B. Visual Policy in the Closed Loop

In this subsection, we introduce our closed-loop system architecture and visual policy design, as shown in Figure 2. The previous subsection has introduced the Edit API for

[1] More specifically, to construct FalconGym 2.0, a human operator uses the onboard camera to capture a 3-minute video across the flying arena from diverse viewpoints. We recover camera intrinsics and initial poses with COLMAP [18]. Because COLMAP's frame is arbitrary, we align it to a physically meaningful world frame by placing an easily accessible ArUco marker in the scene, thereby eliminating the need for expensive motion capture. Using OpenCV's ArUco detector, we locate the marker center in a subset of images where the marker is visible and treat it as the global origin. We then compute the rigid transform between the COLMAP and world frames via the Kabsch-Umeyama method [19]. Finally, the images and poses are fed to the NeRFStudio `Splatfacto` pipeline [20] to train a photorealistic GSplat scene in world coordinates.

moving objects (i.e. gates) within GSplat scenes; here we focus on moving the aerial vehicle (i.e., the onboard camera). Although the aerial vehicle could be placed in arbitrary poses in FalconGym 2.0, closed-loop control requires its motion obey physically meaningful dynamics. FalconGym 2.0 provides two aerial dynamics models: a Dubins airplane dynamics model for fixed-wing UAVs [21] and a quadrotor dynamics model [22]. The dynamics model is implemented in a plug-and-play fashion so users can substitute it with alternative platforms' dynamics (e.g., VTOL or ground robots).

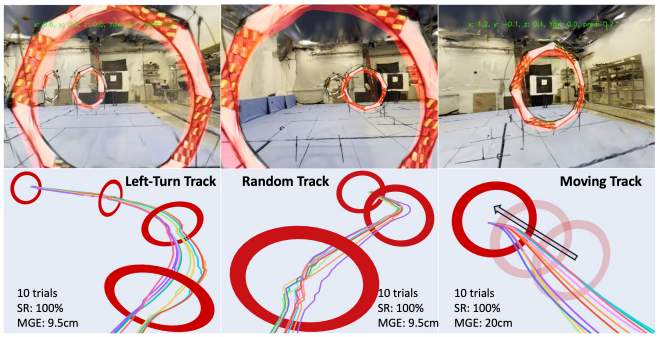

Fig. 4: Trajectories for quadrotor case study in FalconGym 2.0 across three unseen tracks (Left-Turn, Random and Moving).

With renderer and dynamics in place, next we focus on the visual policy design. Miao et al. [3] report strong zero-shot sim-to-real quadrotor navigation with an end-to-end ViT policy. However, as also acknowledged by their authors, their visual policy trained on one track tends to specialize to that track. Our further experiments confirm [3]'s overfitting: even after removing the gates entirely using our Edit API, [3]'s visual policy continued to fly around the memorized route, suggesting reliance on background appearance rather than gate-relevant features. Moreover, Miao et al. [3] run hardware experiments offboard because the dual-Vision-Transformer (ViT) policy is computationally heavy.

Therefore, to mitigate overfitting and reduce model complexity for onboard deployment, we design a modular architecture that decouples perception from control, inspired by [5] which trains a RL algorithm based on gate-perception masks to fly through racing gates, and [23] which trains visual policy using masks to follow leader plane. First, we design a perception module that predicts a gate mask. Next, a lightweight controller consumes this mask with past controls to output the next action, as shown in Figure 2. As validated in Section IV, this modular pipeline not only improves generalization, but also supports on-board execution due to the smaller model size compared to [3]'s dual-ViT setup.

a) Gate-Detection Perception Module: We train the perception module completely in FalconGym 2.0. As shown in the blue box in Figure 2, the perception module takes an onboard RGB image and predicts a binary mask where white pixels indicate gates and black pixels indicate background. We adopt a U-Net [24] backbone for gate segmentation for its strong performance on dense prediction. To obtain ground-truth masks, we automatically generate them analytically via standard 3D-to-2D projection techniques in computer vision. Because gate positions (placed in FalconGym 2.0 through our Edit API) are known and the geometry (e.g., diameter) is measured, we can project 3D gates to a 2D image plane using known camera matrices. Pixels whose coordinates lie within the gate's projected ring (i.e., between inner and outer boundaries) are labeled white, and all others black.

To collect training data for the U-Net, we: (i) leverage the Edit API to place a two-gate track in the workspace; (ii) sample a feasible camera pose at an appropriate distance with yaw roughly oriented toward the track; and (iii) render the RGB image and compute its ground truth 2D mask as above. We sample two-gate tracks because we want the U-Net to learn about scenarios where one gate might be occluded by the other, and experiments show our U-Net could generalize to multi-gate detection. We gather RGB images (square-gate UMX for the UAV case study and circular-gate for the quadrotor case study) and train separate U-Nets for each case study with supervised learning on these image-mask pairs. Qualitative results are shown in Figures 1 and 4 in FalconGym 2.0, while Figure 6 visualizes performance when zero-shot deployed on hardware.

b) Controller: With a predicted mask, the controller receives an explicit geometry-focused signal of gate locations, potentially reducing reliance on background cues and mitigating overfitting. Next, we further improve the architecture of [3] by removing IMU inputs and instead feeding a short history of past controls to the controller (blue box in Figure 2), which provides implicit temporal context. Next, controller training follows a similar imitation-learning procedure to [3]: we first implement a state-based expert with the ground-truth locations of the vehicle and gates at all times to fly through different tracks in simulation; at each timestep, we render the onboard RGB image and record the state-based controller's expert action. During the data collection, we also add mild noise to the expert action, which is typical in imitation learning, to encourage exploration while making sure all gates can still be passed successfully. For more details, we refer readers to [3]. The RGB image is passed through the trained U-Net to obtain a binary mask, and we form supervised pairs where the masked image coupled with the past control actions are used to predict the current action to train the controller.

Thanks to the Edit API, we can synthesize essentially arbitrarily many tracks in FalconGym 2.0 to train both perception and controller without additional per-track real-world effort required by [1], [3], [5]. To sample efficiently, our unique design choice is to train on *two-gate tracks*. Intuitively, the initial state together with two successive gates spans the local geometric variability of longer courses; a controller that performs well on such segments could generalize well to multi-gate tracks by invariance and composition, as is empirically confirmed in Section IV.

C. Performance-Guided Refinement Training

A straightforward method to collect training data for the visual policy would be to uniformly sample the two-gate

track space that is dynamically feasible and observable (as defined at the start of this section). However, uniform sampling can be sample-inefficient in a large high-dimensional workspace. With our Edit API, we can steer training data collection toward the visual policy's weak spots and iteratively refine to improve the visual policy.

Inspired by adversarial training [25], [26] and min-max optimization [27], which minimizes loss under worst-case conditions, we cast aerial navigation as a min-max problem:

$$\min_{\theta} \max_{g \in G} \mathbb{E}_{\tau \sim \pi_\theta} \big[\mathcal{L}(\tau; g) \big], \tag{1}$$

where g parameterizes a two-gate track, G is the dynamically feasible and observable two-gate space, and $\mathcal{L}$ measures task performance on a trajectory rollout τ achieved by the visual policy (perception and controller) π parameterized by θ. We define the task performance function as $\mathcal{L}(\tau; g) = \mathbf{1}\{\text{collision or timeout}\} + \lambda_{\text{pos}} \left\| p' - c(g) \right\|_2$, where $c(g) \in \mathbb{R}^3$ is the gate center, and $p' \in \mathbb{R}^3$ is the aerial vehicle's position when crossing the target gate plane. The second term is only evaluated on successful crossings. Intuitively, the "maximizer" proposes challenging gate placements to induce poor visual-policy performance of π_θ, while the "minimizer" updates π_θ to reduce loss. This focuses training on regions where the visual policy underperforms.

Algorithm 1 Performance-Guided Refinement of Visual Policy

Input: Gate space G partitioned into M grids, state-based expert policy π_s, iterations T, validation gate set G_{val}
Output: Trained visual policy π_θ

$\mathcal{D} \leftarrow \varnothing$ ▷ Training dataset
Sample initial tracks $G_1 \subset G$ uniformly at random
for $t = 1$ to T **do**
  for all two-gate track $g \in G_t$ **do**
    Run closed-loop trajectory rollout with π_s and g
    Collect trajectory (action, image) and add to $\mathcal{D}$
  end for
  Train visual policy π_θ on $\mathcal{D}$
  for $i = 1$ **to** M **do** ▷ evaluate grid-wise performance
    $\ell_i \leftarrow \frac{1}{|G_{\text{val}} \cap M_i|} \sum_{g \in G_{\text{val}} \cap M_i} \mathcal{L}(\pi_\theta; g)$
  end for
  Compute normalized grid weights:
  $w_i \leftarrow \frac{\ell_i}{\sum_{i=1}^{M} \ell_i} \quad \forall i = 1, \ldots, M$
  $w_i \leftarrow (1 - \beta)\, w_i + \frac{\beta}{M}$ ▷ Avoid Mode Collapse
  Generate next set G_{t+1} by first choosing M_i with probability w_i and then sampling $g \sim M_i$ uniformly
end for

Yet directly solving Equation (1) is infeasible, so we implement grid-based *Performance-Guided Refinement (PGR)*, as shown in Algorithm 1. We partition the two-gate space G into M grids $\{M_i\}_{i=1}^{M}$. During the first iteration, we synthesize G_1 by first uniformly sampling the grids and then drawing two-gate layouts uniformly within selected grids. We delete the dynamically infeasible and unobservable tracks automatically using the expert state-based controller and geometric calculation. Next, for each iteration t: we (i) collect trajectory rollouts on a batch of two-gate layouts G_t using the expert state-based controller to augment the perception and imitation dataset $\mathcal{D}$, (ii) train the visual policy π_θ on $\mathcal{D}$, (iii) evaluate per-grid validation performance ℓ_i using a held-out set G_{val}, and (iv) *resample* the next batch G_{t+1} by first drawing grids with probability proportional to ℓ_i and then sampling tracks uniformly within each selected grid. We apply this PGR to both perception and controller training since poor performance could come from both modules. To avoid mode collapse (as observed in [25]), we mix a small fraction of uniform sampling $w_i \leftarrow (1 - \beta)\, w_i + \beta/M$, and reuse a fixed G_{val} for stable scoring. "Ours (w/ PGR)" in Table I and Table II denotes performance-guided refinement, while "Ours (w/o PGR)" denotes uniform sampling of two-gate tracks under the same computational budget.

IV. Experiments

We evaluate our approach in two case studies: fixed-wing UAVs and quadrotors. For each case study, we first describe the experimental setup and then quantitatively evaluate our visual policy against baselines in FalconGym 2.0. We further demonstrate zero-shot sim-to-real transfer on the quadrotor case study in Section IV-B.2.

A. Case Study 1: fixed-wing UAV

We model the fixed-wing UAV with Dubins Airplane dynamics [21]. The state is (x, y, z, ψ, θ) (3D position, yaw and pitch). The control action is bank rate and pitch rate. Forward speed is fixed at $7\,\mathrm{m/s}$. We also implement the state-based controller from [21] as both a baseline and the expert for imitation learning (Section III-B).

Our UAV flying arena measures $40 \times 20 \times 4\,\mathrm{m}$. The square gates used in the UAV case study have an inner side length of $200\,\mathrm{cm}$. The UAV has a width of $40\,\mathrm{cm}$. A gate crossing is deemed *success* if, at the instant the aerial vehicle passes the gate plane, the distance from the UAV to the gate center is less than $80\,\mathrm{cm}$. We evaluate gate-crossing performance using two metrics: *Success Rate (SR)*, the percentage of gates the visual policy successfully crosses, and *Mean Gate Error (MGE)*, the average distance between the UAV and the gate center at crossing time.

Using FalconGym 2.0's Edit API, we design three dynamically feasible, observable tracks (Figure 1): (i) *Spatial-S*, requiring consecutive sharp turns with altitude changes; (ii) *Random*, where the UAV largely maintains heading while alternating left/right gates and adjusting height; (iii) *Moving*, where one gate translates from right to upper-left at $2\,\mathrm{m/s}$. For each track, we evaluate 10 slightly different initial poses and report average SR and MGE in Table I.

We evaluate five policies by SR, MGE, and generalization. The state-based expert [21], which has full state and gate poses, serves as oracle and baseline and achieves the best SR/MGE. We also implement two visual baselines [3], [5]. For Miao et al. [3], we retrain their architecture in FalconGym 2.0 and replace IMU input with past control (IMU is inapplicable to Dubins dynamics). For Geles et al. [5], whose code is unavailable, we keep their controller design

but replace their detector with our mask detector and retune reward hyperparameters. Our reproduced trends (e.g., Figure 5) match reported behavior. Because both baselines require matched train/test tracks, we train on Spatial-S and evaluate on all three tracks. They perform well on Spatial-S but degrade on unseen Random and Moving (Table I); when retrained and tested on the exact Random track, both return to 100% SR, confirming track overfitting.

TABLE I: Evaluation of five policies by Success Rate (SR) and Mean Gate Error (MGE) in the UAV case study. Success requires gate-plane crossing error (MGE) ≤ 80 cm.

		FalconGym 2.0			
Track	Method	Vision?	Generalize?	SR ↑	MGE (cm) ↓
Spatial-S (5 gates)	State-based	✗	✓	100%	15
	Miao et al. [3]	✓	✗	100%	24
	Geles et al. [5]	✓	✗	100%	20
	Ours (w/o PGR)	✓	✓	100%	47
	Ours (w/ PGR)	✓	✓	**100%**	**19**
Random (4 gates)	State-based	✗	✓	100%	8
	Miao et al. [3]	✓	✗	0%	N/A
	Geles et al. [5]	✓	✗	50%	47
	Ours (w/o PGR)	✓	✓	100%	33
	Ours (w/ PGR)	✓	✓	**100%**	**21**
Moving (1 gate)	State-based	✗	✓	100%	14
	Miao et al. [3]	✓	✗	0%	N/A
	Geles et al. [5]	✓	✗	0%	N/A
	Ours (w/o PGR)	✓	✓	100%	47
	Ours (w/ PGR)	✓	✓	**100%**	**15**

In contrast, both of our methods, uniform sampling and performance-guided refinement, generalize to unseen tracks with a *single* visual policy. All three evaluation tracks are unseen during training, which uses only two-gate layouts (Section III-B). "Ours (w/o PGR)" uniformly samples the gate space to generate 300k dynamically feasible, observable two-gate tracks and trains the visual policy by imitation. "Ours (w/ PGR)" uses the performance-guided refinement in Section III-C. Specifically, we partition $G \subset \mathbb{R}^8$ (each gate has (x, y, z, yaw)) into $(4 \times 4 \times 3 \times 3)^2 \approx 20$k grids. In the first iteration we draw 5 samples per grid to form 100k training tracks; we then run PGR for $T{=}3$ iterations with $\beta{=}0.05$ in Algorithm 1. Both variants achieve high SR and generalize across all tracks, while PGR further reduces MGE, indicating safer crossings.

We also assess robustness to gate-position perturbations on Spatial-S using FalconGym 2.0's domain-randomization Edit API (Figure 5). For fair comparison, we conduct this ablation study on the Spatial-S track on which both baselines are trained, and evaluate perturbed variants of that track. At perturbation level a (cm), every gate is independently shifted by a 3D offset $\delta \in [-a, a]^3$ along x, y, and z with random signs. We generate 10 perturbed tracks (50 gates total) for each perturbation level and run all five policies, reporting Success Rate (SR). Because Spatial-S already requires sharp turns and altitude changes, larger a demands more aggressive maneuvers, so SR decreases for all methods. At high perturbation levels, even the state-based expert fails due to dynamic infeasibility. Figure 5 shows that PGR is more robust to gate-pose perturbations than the visual baselines and uniform sampling. PGR closely tracks the expert trend with only a small SR gap, consistent with imitation from the state-based expert.

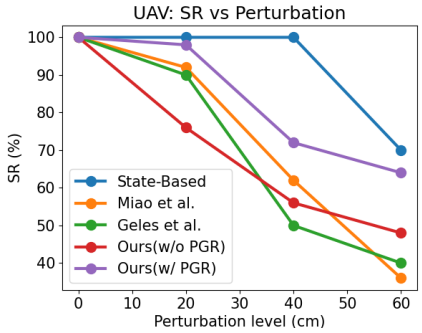

Fig. 5: **Robustness to gate-pose perturbations on *Spatial-S* track in FalconGym 2.0.** For a perturbation level a cm, each gate is independently shifted by a random 3D offset $\delta \in [-a, a]^3$. For each perturbation level, we run all five policies on 10 randomized tracks (50 gates total) and report the Success Rate (SR).

B. Case Study 2: Quadrotor

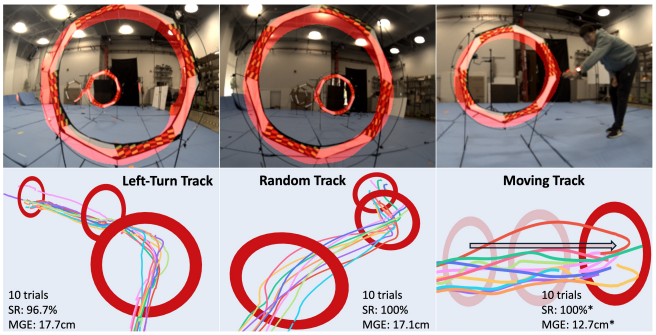

Fig. 6: **Trajectory plots for the quadrotor case study on real hardware.** The visual policy learned in FalconGym 2.0 transfers zero-shot to real hardware. For safety, we disable forward velocity on the Moving track and evaluate lateral tracking only.

1) Performance in FalconGym 2.0: We model the quadrotor with the standard 12-state dynamics of [22]. The state comprises position, linear velocity, attitude, and angular velocity. The state-based expert controller is also from [22], with constant forward speed $1\,\mathrm{m/s}$. Among several control modalities, we choose body-frame linear velocities and yaw rate to match the hardware interface used for zero-shot sim-to-real transfer in Section IV-B.2. Our quadrotor flight arena measures $6 \times 6 \times 3\,\mathrm{m}$, as shown in Figure 6. The circular gates have a $78\,\mathrm{cm}$ inner diameter, and the quadrotor width is $18\,\mathrm{cm}$. Therefore, a crossing is considered *successful* if, at the instant the quadrotor intersects the gate plane, its distance to the gate center (MGE) is $\leq 30\,\mathrm{cm}$. We adapt the same metrics as in the fixed-wing study: *Success Rate (SR)* and *Mean Gate Error (MGE)*.

We again design three dynamically feasible, observable tracks (Figure 4) using FalconGym 2.0's Edit API: (i) *Left-Turn*, which involves sustained turning; (ii) *Random*, where the quadrotor maintains a general heading while alternating

left/right gates; (iii) *Moving*, where one gate translates right at $0.25\,\mathrm{m/s}$.

TABLE II: Evaluation of five policies by SR and MGE in FalconGym 2.0 and the real world for the quadrotor case study. * indicates lateral-tracking-only evaluation for the Moving track (safety). Success requires MGE ≤ 30 cm.

		FalconGym 2.0		Real World	
Track	Method	SR ↑	MGE (cm) ↓	SR ↑	MGE (cm) ↓
Left-Turn (3 gates)	State-based	100%	5.8	100%	11.6
	Miao et al. [3]	100%	14.3	/	/
	Geles et al. [5]	100%	17.0	/	/
	Ours (w/o PGR)	100%	11.3	93.3%	21.4
	Ours (w/ PGR)	**100%**	**9.5**	**96.7%**	**17.7**
Random (3 gates)	State-based	100%	5.7	100%	5.9
	Miao et al. [3]	67.7%	32.3	/	/
	Geles et al. [5]	25%	40.0	/	/
	Ours (w/o PGR)	100%	11.4	96.7%	16.2
	Ours (w/ PGR)	**100%**	**9.5**	**100%**	**17.1**
Moving (1 gate)	State-based	100%	2.0	100%*	7.4*
	Miao et al. [3]	0%	N/A	/	/
	Geles et al. [5]	0%	N/A	/	/
	Ours (w/o PGR)	100%	24.1	100%*	16.8*
	Ours (w/ PGR)	**100%**	**20.7**	**100%***	**12.7***

Baselines are implemented as in the fixed-wing study but trained with the quadrotor GSplat data, with the modifications described in Section IV-A to fit our pipeline. While our full visual policy (perception and controller) also works for quadrotors, we use a modified version that replaces the imitation-learned controller with a classical controller that (i) filters perception noise and keeps only the largest connected component (closest gate) in the predicted mask and (ii) commands the vehicle to follow that component's centroid. This engineering trade-off is necessary because the hardware quadrotor (Section IV-B.2) cannot execute two neural networks sequentially at the required closed-loop rate. Therefore, we keep the U-Net for perception and replace the second network with a classical controller. As a result, instead of applying PGR to the entire visual policy (perception and controller) as in the fixed-wing UAV case study, we apply it only to the perception module.

As in the fixed-wing UAV experiments, we evaluate all five policies on three tracks with ten different initial conditions. Again, both baselines perform well on the training *Left-Turn* track but overfit and degrade on the unseen *Random* and *Moving* tracks, as shown in Table II. By contrast, both our uniform sampling (Ours w/o PGR) and performance-guided refinement (Ours w/ PGR) maintain 100% SR across all tracks. While PGR further improves MGE, the margin over uniform sampling is smaller than in the fixed-wing case. We believe this is due to: (i) PGR is only applied to the perception module (rather than the full perception-control stack) and (ii) the quadrotor's smaller workspace with coarser grid discretization, i.e., $\left(3 \times 3 \times 2 \times 2\right)^2 = 1{,}296$ grids.

We also conduct a gate-pose perturbation study similar to the fixed-wing ablation study on the Left-Turn track in FalconGym 2.0. Figure 7 shows that baselines' SR drops rapidly as perturbation level increases, whereas our policies remain substantially more robust and closely track the expert's. The incremental gain from PGR over uniform sampling is

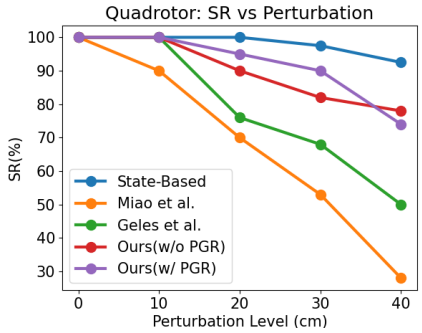

Fig. 7: **Gate perturbations on Left-Turn track in FalconGym 2.0.** For each perturbation level, we run all five policies on 10 randomized tracks (30 gates total) and report the Success Rate (SR).

modest, again likely due to the partial refinement on the perception only and the smaller space and number of grids.

2) Zero-shot Sim-to-Real Transfer: We zero-shot deployed the trained perception module and the same classical controller used in FalconGym 2.0 on a 280 g ModalAI® Starling 2 quadrotor (upper-right of Figure 2). The quadrotor is equipped with a PX4 flight controller, a VOXL 2® onboard computer with an Adreno 650 GPU for neural inference, and a 12 MP RGB camera. The onboard closed-loop frequency is 8 Hz when running U-Net perception and classical controller.

Although our visual policy does not require the tracks to be the same as in training or in FalconGym 2.0, for a fair sim-to-real comparison, we arranged the hardware courses to mirror the simulation layouts: *Left-Turn*, *Random*, and *Moving*. Due to the small indoor arena and observability assumptions, for *Left-Turn* and *Random* we cross only three gates and reserve the fourth as a navigational waypoint to satisfy the observability requirement. For the *Moving* track, a human operator slowly pulls the gate laterally at around $0.25\,\mathrm{m/s}$; for safety, we disable forward velocity and evaluate average lateral tracking error of the gate centroid as MGE. We use motion capture (mocap) to log quadrotor poses and moving-gate trajectories. While we also use mocap for the state-based controller baseline, it is not used in FalconGym 2.0 construction or in vision-based closed-loop control. Real-world onboard views and quadrotor trajectories are visualized in Figure 6.

Due to safety constraints, differing control modalities, and limited onboard compute, we do not run the visual baselines on hardware. We evaluate the remaining three policies under the same protocol as in FalconGym 2.0: each policy is run on each track with 10 different initial conditions, and we report SR and MGE in Table II. Although both our visual-policy variants perform slightly worse than in simulation, they maintain high success rates (SR $\geq 93\%$) across all tracks. The primary failure modes are (i) perception errors where the U-Net confuses a background ladder with a gate and (ii) latency limits from the 8 Hz closed loop. We mark Moving track results with * to indicate lateral-tracking-only hardware evaluation (no gate crossing for safety), which likely yields lower MGE than full crossing in FalconGym

2.0. Consistent with simulation and the UAV case study, PGR improves SR and shows comparable MGE to uniform sampling across hardware tracks. Both visual-policy variants also track state-based-controller performance closely across all three tracks.

V. CONCLUSIONS

We introduced *FalconGym 2.0*, a GSplat-based photorealistic simulation framework that provides an *Edit API* for programmatic object transforms. Leveraging this editability, we proposed *performance-guided refinement* (PGR), which concentrates visual-policy training on challenging tracks and iteratively improves performance. Across two case studies (fixed-wing UAVs and quadrotors) with different dynamics and environments, our visual policy generalizes across tracks, achieves 100% SR on unseen tracks in FalconGym 2.0, and is more robust to gate-pose perturbations than baselines. Finally, we demonstrate zero-shot sim-to-real transfer on quadrotor hardware with 98.6% SR over three tracks (70 gates).

In future work, we plan to: relax the *observability* assumption to handle occluded gates; evaluate higher-speed flights for the quadrotor case study; incorporate realistic UAV dynamics to enable fixed-wing sim-to-real transfer; and distill the perception and controller neural networks to deploy the full two-network stack onboard.

ACKNOWLEDGMENT

This research was supported by the Air Force Office of Scientific Research (AFOSR) MURI grant FA9550-23-1-0337, HyDDRA: Hybrid Dynamics - Deconstruction and Aggregation, and by the National Science Foundation (NSF) FMitF program, Visual Computing Meets Formal Verification: Certified Rendering, Geometry, and Video Generation (Award No. 2525287). We also acknowledge support from the Center for Autonomy at the University of Illinois Urbana-Champaign for enabling the experimental work.

REFERENCES

[1] J. Low, M. Adang, J. Yu, K. Nagami, and M. Schwager, "Sous vide: Cooking visual drone navigation policies in a gaussian splatting vacuum," *IEEE Robotics and Automation Letters (under review)*, 2024, available on arXiv: https://arxiv.org/abs/2412.16346.

[2] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis, "3d gaussian splatting for real-time radiance field rendering," *ACM Transactions on Graphics*, vol. 42, no. 4, July 2023. [Online]. Available: https://repo-sam.inria.fr/fungraph/3d-gaussian-splatting/

[3] Y. Miao, W. Shen, and S. Mitra, "Falcongym: A photorealistic simulation framework for zero-shot sim-to-real vision-based quadrotor navigation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Hangzhou, China, October 2025.

[4] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, *et al.*, "Nerf: representing scenes as neural radiance fields for view synthesis," *Commun. ACM*, vol. 65, no. 1, p. 99–106, Dec. 2021. [Online]. Available: https://doi.org/10.1145/3503250

[5] I. Geles, L. Bauersfeld, A. Romero, J. Xing, and D. Scaramuzza, "Demonstrating agile flight from pixels without state estimation," in *Robotics: Science and Systems XX, Delft, The Netherlands, July 15-19, 2024*, D. Kulic, G. Venture, K. E. Bekris, and E. Coronado, Eds., 2024. [Online]. Available: https://doi.org/10.15607/RSS.2024.XX.082

[6] Q. Chen, J. Sun, N. Gao, J. Low, T. Chen, *et al.*, "Grad-nav: Efficiently learning visual drone navigation with gaussian radiance fields and differentiable dynamics," 2025. [Online]. Available: https://arxiv.org/abs/2503.03984

[7] P. Soviany, R. T. Ionescu, P. Rota, and N. Sebe, "Curriculum learning: A survey," 2022. [Online]. Available: https://arxiv.org/abs/2101.10382

[8] A. Byravan, J. Humplik, L. Hasenclever, A. Brussee, F. Nori, *et al.*, "Nerf2real: Sim2real transfer of vision-guided bipedal motion skills using neural radiance fields," *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 9362–9369, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:252815541

[9] M. Torne, A. Simeonov, Z. Li, A. Chan, T. Chen, *et al.*, "Reconciling reality through simulation: A real-to-sim-to-real approach for robust manipulation," *Arxiv*, 2024.

[10] Z. Xie, Z. Liu, Z. Peng, W. Wu, and B. Zhou, "Vid2sim: Realistic and interactive simulation from video for urban navigation," *Preprint*, 2024.

[11] S. Yang, W. Yu, J. Zeng, J. Lv, K. Ren, *et al.*, "Novel demonstration generation with gaussian splatting enables robust one-shot manipulation," *arXiv preprint arXiv:2504.13175*, 2025.

[12] O. Shorinwa, J. Tucker, A. Smith, A. Swann, T. Chen, *et al.*, "Splat-mover: Multi-stage, open-vocabulary robotic manipulation via editable gaussian splatting," 2024.

[13] R. Portelas, C. Colas, K. Hofmann, and P.-Y. Oudeyer, "Teacher algorithms for curriculum learning of deep rl in continuously parameterized environments," in *Proceedings of the Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, L. P. Kaelbling, D. Kragic, and K. Sugiura, Eds., vol. 100. PMLR, 30 Oct–01 Nov 2020, pp. 835–853. [Online]. Available: https://proceedings.mlr.press/v100/portelas20a.html

[14] H. Wang, J. Xing, N. Messikommer, and D. Scaramuzza, "Environment as policy: Learning to race in unseen tracks," 2025. [Online]. Available: https://arxiv.org/abs/2410.22308

[15] Y. Chen, Z. Chen, C. Zhang, F. Wang, X. Yang, *et al.*, "Gaussianeditor: Swift and controllable 3d editing with gaussian splatting," 2023. [Online]. Available: https://arxiv.org/abs/2311.14521

[16] Y. Wang, X. Yi, Z. Wu, N. Zhao, L. Chen, *et al.*, "View-consistent 3d editing with gaussian splatting," 2025. [Online]. Available: https://arxiv.org/abs/2403.11868

[17] C. Vachha and A. Haque, "Instruct-gs2gs: Editing 3d gaussian splats with instructions," 2024. [Online]. Available: https://instruct-gs2gs.github.io/

[18] J. L. Schönberger and J.-M. Frahm, "Structure-from-motion revisited," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 4104–4113.

[19] W. Kabsch, "A discussion of the solution for the best rotation to relate two sets of vectors," *Acta Crystallographica Section A*, vol. 34, no. 5, pp. 827–828, Sept. 1976. [Online]. Available: http://dx.doi.org/10.1107/S0567739478001680

[20] M. Tancik, E. Weber, E. Ng, R. Li, B. Yi, *et al.*, "Nerfstudio: A modular framework for neural radiance field development," in *ACM SIGGRAPH 2023 Conference Proceedings*, ser. SIGGRAPH '23, 2023.

[21] M. Owen, R. W. Beard, and T. W. McLain, *Implementing Dubins Airplane Paths on Fixed-Wing UAVs*. Dordrecht: Springer Netherlands, 2015, pp. 1677–1701.

[22] F. Sabatino, "Quadrotor control: modeling, nonlinear control design, and simulation," 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:61413561

[23] Y. Miao, W. Shen, H. Cui, and S. Mitra, "Falconwing: An ultra-light indoor fixed-wing uav platform for vision-based autonomy," 2025. [Online]. Available: https://arxiv.org/abs/2505.01383

[24] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, Eds. Springer International Publishing, 2015, pp. 234–241.

[25] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, *et al.*, "Generative adversarial networks," *Commun. ACM*, vol. 63, no. 11, p. 139–144, Oct. 2020. [Online]. Available: https://doi.org/10.1145/3422622

[26] S. YAGHOUBI and G. FAINEKOS, "Worst-case satisfaction of stl specifications using feedforward neural network controllers: A lagrange multipliers approach," in *2020 Information Theory and Applications Workshop (ITA)*, 2020, pp. 1–20.

[27] K. Shimizu, Y. Ishizuka, and J. F. Bard, *Min-Max Problem*. Boston, MA: Springer US, 1997, pp. 271–279.