Vision-Guided Outdoor Flight and Obstacle Evasion via Reinforcement Learning

Shiladitya Dutta[1], Aayush Gupta[1], Varun Saran[1], Avideh Zakhor[1]

***Abstract*— Although quadcopters boast impressive traversal capabilities enabled by their omnidirectional maneuverability, the need for continuous pilot control in complex environments impedes their application in GNSS and telemetry-denied scenarios. To this end, we propose a novel sensorimotor policy that uses stereo-vision depth and visual-inertial odometry (VIO) to autonomously navigate through obstacles in an unknown environment to reach a goal point. The policy is comprised of a pre-trained autoencoder as the perception head followed by a planning and control LSTM network which outputs velocity commands that can be followed by an off-the-shelf commercial drone. We leverage reinforcement and privileged learning paradigms to train the policy in simulation through a two-stage process: 1) initial training with optimal trajectories generated by a global motion planner acting as a supervisory backbone, 2) further fine-tuning in a curriculum environment. To bridge the sim-to-real gap, we employ domain randomization and reward shaping to create a policy that is both robust to noise and domain shift. In actual outdoor experiments, our approach achieves successful zero-shot transfer to both a drone platform (DJI M300) and environments with obstacles that were never encountered during training.**

I. INTRODUCTION

Remote-controlled quadcopters are popular within commercial and enterprise markets for their unparalleled mobility in a small, cheap form factor. However, they have several key drawbacks stemming from their control process including bounds to where they can traverse due to wireless connection limitations and the need for constant operator attention. While autopilot for following preset simple paths is a common feature, in obstacle-rich, dynamic or unknown environments autonomous navigation still is not fully realized. This impedes their utilization in GNSS and telemetry denied applications such as sub-canopy forest flight, underground mapping, war zones, and industrial inspection.

As such, an open area of research is autonomous navigation in unknown environments using only onboard computation and sensors, typically in the form of vision-based systems. While traditional methods decompose the navigation task into discrete planning, perception and control units, new end-to-end learning-based methods using privileged information are promising [17] [25]. These involve creating an expert policy with full environmental and state data to generate optimal paths, then distilling it into a student policy using supervised learning. However, these methods use body rates or trajectories followed by a model predictive controller (MPC) to achieve agile flight which cannot be used as-is across different drone models without further tuning [17].

[1]Department of Electrical Engineering and Computer Science, College of Engineering, University of California Berkeley

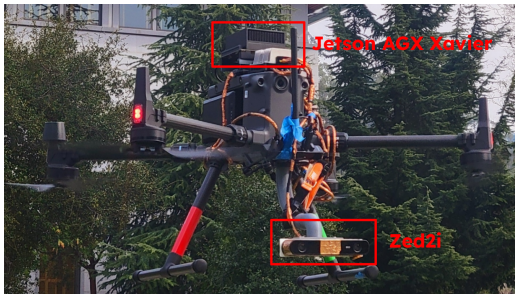

Fig. 1. Actual testbed is a DJI M300 with an attached Zed2i sensor for depth estimation & VIO and a Jetson AGX Xavier for compute.

This paper seeks to train a policy that can be deployed onto real drones of different sizes and classes in a zero-shot fashion without any modification. We accomplish this by pairing a deep Reinforcement Learning (RL) paradigm and privileged learning to train an end-to-end model that outputs reference velocity commands which can be followed by off-the-shelf consumer drones through built-in APIs. We take a modular approach to constructing this network where a pre-trained AutoEncoder acts as a perception head by mapping the depth image data to a low-dimensional latent space. We then freeze the perception component and train an LSTM planning/control network first with optimal trajectories as a supervisory backbone for the reward function before further fine-tuning on a curriculum environment. To cross the Sim2Real gap, we employ domain randomization and policy smoothing to ensure successful transfer on varying environments and platforms. Domain randomization teaches the policy to be robust to sensor noise and trains the LSTM to account for differing system dynamics such as latency, inertia, etc. across time-steps [7]. Meanwhile, reward shaping prevents large action fluctuations that would degrade the drone's stability and controllability.

The result is a system that can navigate through obstacles to a goal in an outdoor environment using stereo-vision depth and visual inertial odometry (VIO). We deploy this system onto a hardware testbed depicted in Figure 1. With real-world experiments covering a total of 650m in field trials, the policy overcomes Sim2Real gaps such as scenery changes, sensor noise, background clutter and crosswinds demonstrating successful collision-free navigation in new environments and on an off-the-shelf drone platform –the DJI M300 RTK– that was never encountered during training.

II. RELATED WORK

There has been extensive research on methods for collision-free drone flight in cluttered environments. In sce-

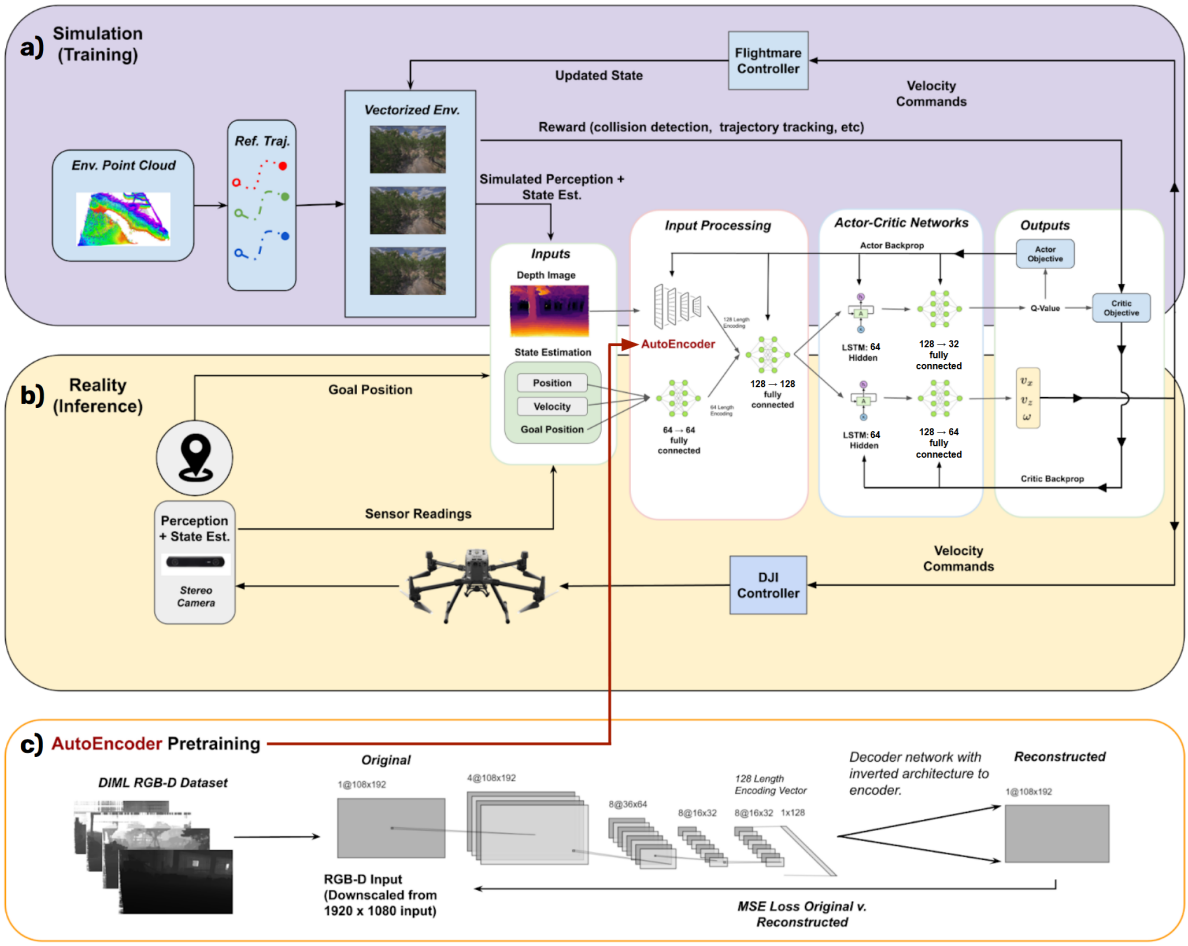

Fig. 2. Overview of System. In (c), we first pre-train an auto-encoder to process depth images which is used as the perception head of the policy in (a) and (b). We then generate environments and calculate optimal trajectories to be used as a supervisory backbone. To train the input processing network and actor-critic networks in (a), we use PPO for optimization and Flightmare for rendering/dynamics simulation. When deploying the policy in reality with (b), we interface with the drone using the high-level DJI Controller and collect state estimates + depth image using the stereo RGB-D camera.

narios where perfect environment information e.g. 3D point cloud or map is provided, many existing approaches take a global planning perspective where dynamically feasible trajectories are pre-computed using a sampling-based or hierarchical algorithm [15] [20]. A related task, in terms of goals though not necessarily methods, is autonomous racing in known environments where the drones fly through gates. State-of-the-art methods employ learning-based approaches to handle noisy perception and dynamics [11].

For navigation in unknown environments, traditional methods can be separated into 3 components: perception, planning, and control. A common strategy is to use a reactive planner which searches through a set of feasible trajectories to generate a local plan. These can be split into three approaches: building a map from past observations, directly using past observations, and memoryless methods which only use the current observation. Much of the research in this area focuses on optimizing the trajectory search space and collision-checking processes. Examples include bitwise trajectory elimination [26], sampling-based schemes to generate a free-space skeleton [9], and rectangular pyramid partitioning for efficient collision checking [2]. Other advancements include safe-stopping trajectory planning [14] and integrating semantic SLAM [16]. Despite these optimizations, end-to-end neural policies remain architecturally more efficient, as their inference is significantly simpler than tasks such as map fusion. Furthermore, hardware acceleration—leveraging NVIDIA DLA Cores or Mythic AMP—further enhances inference speed. In addition, many classical approaches – when integrated together– react adversely to compounding errors in state estimation, system latency, and sensor noise. As such, learning-based approaches replacing some or all of this 3-part stack have become a focal point of research in recent years [8]. In particular, end-to-end sensorimotor policies that directly map vision to action have proven to be effective in agile high-speed scenarios [6].

The architecture of these end-to-end networks can be similarly broken up into a perception section and a planning/control section. For the perception network, common practices are to employ convolution layers or an image autoencoder pretrained on a separate image dataset, though [1] suggests that Vision Image Transformers have better OOD generalizability and performance. Other approaches use neural monocular depth estimation to eliminate the need for stereo cameras [28][29] and to train the autoencoder with collision meshes so that the perception network outputs can

be collision-aware [12]. For control, many monocular approaches either have the perception network directly predict a steering angle and collision probability [18] or follow a state machine [19]. For end-to-end sensorimotor approaches, while fully connected networks are commonly used for the planning/control portion, some studies have proposed using Long Short-Term Memory (LSTM) [7] or attention layers [22] for their ability to implicitly compensate for sim2real factors and remembering partially observed environments over multiple time-steps. A diverse range of action outputs have been explored such as generating local trajectories in a receding horizon [17], outputting body rates [27], and directly outputting motor commands [5].

To train these networks, either reinforcement learning or imitation learning is typically used. Deep RL methods employ a drone dynamics simulator such as Flightmare [24] or AerialGym [13] to train a policy, though some works also integrate real-world data as part of the training loop to account for the sim2real gap in flight dynamics [10][4]. With imitation learning, a privileged expert provides a supervisory signal to a learned student policy, usually employing a distillation approach where the mean squared error between the student and teacher output is optimized. For the expert, some works use traditional global planning methods [17] while others use a Deep RL agent trained with privileged information and a perception-aware reward [25].

Our work is closely related to [17] and [25] in that they both aim to achieve real-world vision-based flight in cluttered outdoor environments using a learning-based approach. [25] focuses on optimizing flight in a *known* environment whereas we focus on flight in unknown environments; also [25] uses RL to train the expert rather than the policy itself. Our approach differs from [17] in that: (a) [17] does not employ any RL techniques, and (b) [17] uses an MPC tuned to their custom-built first-person view (FPV) drone to follow outputted polynomial trajectories whereas we aim to use reference velocity commands that are drone-agnostic. To train the deployed policy, [17] and [25] use imitation learning to match the expert exactly. On the other hand, we use the relative position of the optimal vs. rollout trajectory acts as a component of the reward function in RL.

III. METHODOLOGY

We first provide an overview of our approach. We then discuss the RL setup including reward function and policy architecture. We then describe the training details such as the simulation loop, domain randomization and learning phases.

A. Summary of Approach

Our approach –illustrated in Figure 2– trains a neural network policy to autonomously navigate a drone to a target while evading obstacles. The policy processes depth images and state estimates from onboard sensors to generate velocity commands for the drone to follow. We construct this policy in two stages. We first pre-train an autoencoder, leveraging its encoder to embed depth images into a low-dimensional latent representation. We then freeze the image encoder while training an LSTM actor and state processing network in simulation using Proximal Policy Optimization (PPO). We begin by generating a randomized training environment and computing optimal trajectories using a global motion planner. We then train the policy in this environment where the optimal trajectories are a component of the reward function. Finally, we introduce a more complex curriculum environment and fine-tune the policy without optimal trajectory rewards, enhancing the policy's generalization.

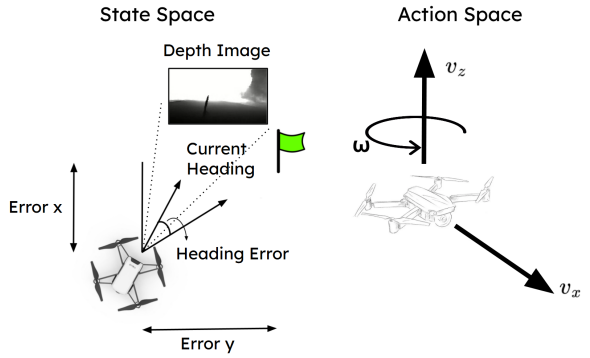

Fig. 3. The state space consists of various measures of drone position relative to goal position alongside the depth image. The action space consists of 3 reference velocities: vertical, horizontal, and angular yaw.

B. Model Architecture

State and Action Space We define the drone's position at time t as $\vec{p}_t = \{p_{t,x}, p_{t,y}, p_{t,z}\}$ for x, y, and z respectively. Likewise, the goal position is $\vec{g} = \{g_x, g_y, g_z\}$. As seen on the left of Figure 3, the state vector at time t is $\vec{s}_t = \{g_x - p_{t,x}, g_y - p_{t,y}, g_z - p_{t,z}, \phi_t, \psi_t, \omega_t, \Delta\psi, z_t\}$ where ϕ_t, ψ_t, and ω_t are current roll, pitch, and yaw rates respectively in rad/s; $\Delta\psi$ is the difference between heading towards goal and current heading in radians; and z_t is the flattened 192×108 depth image. As seen on the right of Figure 3, the action at time t $\vec{a}_t = \{v_{t,x}, v_{t,z}, a_{\omega,t}\}$ consists of horizontal velocity, vertical velocity, and yaw angular velocity. These velocity commands can be directly inputted or translated for the built-in APIs of many off-the-shelf systems such as DJI or AR Parrot drones.

AutoEncoder Pretraining and Feature Extraction We pre-train a denoising autoencoder using the process shown in Figure 2(c). The encoder portion of this network acts as a perception head for the policy by extracting a low dimensional encoding $\mathbb{R}_{128}$ from the high-dimensional depth image input $\mathbb{R}_{192\times108}$. This aids with policy function convergence by reducing the parameters that need to be trained and allows us to tune the perception module separate from the RL simulation loop. For the training dataset, we use DIML [3] which contains depth images from a range of settings e.g. brook, building, construction, overpass, street, trail and inject Gaussian noise into the training input scaled by the dataset's per-pixel depth confidence metrics. This approach helps with robustness by training the encoder on both a wide-range of environment types and for the noise characteristics of stereo depth measurements. As visualized in the Input Processing portion of Figure 2(a), to extract features from the observations we feed the first 7 state values $s_{t,1:7}$ (distance

TABLE I

REWARD TERMS. *ONLY USED WITH PRIVILEGED LEARNING

Term	Expression	Weight
Survival	$-\lambda_1$	$\lambda_1 = 10^{-3}$
Distance to Goal	$\lambda_2(1 - \frac{\|\vec{g}-\vec{p}_t\|_2}{\|\vec{g}-\vec{p}_0\|_2})$	$\lambda_2 = 10^{-3}$
Heading Error	$-\lambda_3 h(\psi_t, \zeta)$	$\lambda_3 = 1/3000$
Z-position Error	$-\lambda_4 (g_z - p_{t,z})^2$	$\lambda_4 = 10^{-3}$
ω Magnitude	$-\lambda_5 a_{t,\omega}^2$	$\lambda_5 = 1/25$
v_z Direction	$\lambda_6 \begin{cases} -1.0, & \text{if } v_{t,z}(g_z - p_{t,z}) < 0 \\ 0.03, & \text{otherwise} \end{cases}$	$\lambda_6 = 1/5000$
Velocity Towards Goal	$\lambda_7 (v_{t,x} \cos(h(\psi_t, \zeta_t)))$	$\lambda_7 = 1/8000$
Acceleration	$-\vec{\lambda_8} \cdot (\vec{a_t} - \vec{a_{t-1}})$	$\vec{\lambda_8} = \begin{bmatrix} 1/20000 \\ 1/15000 \\ 1/20000 \end{bmatrix}$
Yaw Jerk	$-\lambda_9 \|(a_{t,\omega} - a_{t-1,\omega}) - (a_{t-1,\omega} - a_{t-2,\omega})\|$	$\lambda_9 = 10^{-3}$
Obstacle Proximity	$-\lambda_{10} \text{ReLU}(1 - \frac{min_{o \in O} d(\vec{p}_t, o)}{3})$	$\lambda_{10} = 1/3000$
Trajectory Proximity*	$\lambda_{11} \text{ReLU}(1 - \frac{min_{\vec{r} \in R} \|\vec{p}_t - \vec{r}\|_2}{5})$	$\lambda_{11} = 1/2000$

to goal in x, y, and z; roll, pitch, and yaw rates; difference between current heading and heading to goal) through two fully-connected layers before being concatenated with the depth image encoding and passed through two more fully-connected layers. This yields a $\mathbb{R}_{256}$ feature vector which is fed into the actor-critic module of the policy.

Actor-Critic Networks and Policy Optimizer As seen in the Actor-Critic portion of Figure 2(a), both the actor and critic networks take in the feature vector and pass it through a LSTM layer with 64-dimensional hidden and memory units. The actor/critic networks then pass the LSTM outputs through two fully-connected layers (128, 32 neurons in actor and 128, 64 neurons in critic) to obtain the action $\pi(s_t)$ and Q-value $Q^{\pi}(s_t, a_t)$. Note, that the policy outputs range from $[-1, 1]$ and are multiplied by $v_{x,\max}, v_{z,\max}, \omega_{\max}$ to obtain the action $a_t = \{v_{t,x}, v_{t,z}, \omega_t\}$. During training, we use Proximal Policy Optimization (PPO2) [21] to optimize the actor-critic and state mixing networks while freezing the encoder.

Reward Function We define the beginning position of the drone as $\vec{p}_0$ and the desired heading of the drone as $\zeta = \arctan((g_x - p_{t,x})/(g_y - p_{t,y}))$. We use $o \in O$ to represent the set of all obstacles in the environment and $\vec{r} \in R$ to represent the x,y,z positions of points on the optimal trajectory. We define function $d(p, o)$ to be the closest distance between an obstacle o and position $\vec{p}$. We define $h(a_1, a_2)$ to represent the difference between two yaw angles a_1 and a_2 when accounting for phase unwrapping. The reward for a state transition $R(s_{t+1}, s_t, a_t)$ is calculated by adding the reward terms using the definition shown in Table I. The parameters $\lambda_{1...11} \geq 0$ are empirically chosen weights. Intuitively, these rewards terms reflect either proximity to a desired state (e.g. distance to goal) or whether the agent is approaching a desired state (e.g. velocity towards goal).

There are four possible terminal rewards/states: reached goal (2.0), out of bounds (0.16), crashed (0.08), and time-out (-1.0). Reached goal is triggered if the drone reaches within 1.0m of the goal i.e. $\|\vec{g} - \vec{p}_t\|_2 \leq 1.0$. Crashed is triggered if the drone is within 1.25m of an obstacle i.e. $min_{o \in O} d(\vec{p}_t, o) \leq 1.25$. Timeout is if the drone is still flying for more than a maximum amount of time of 40 seconds i.e. $t_{max} \leq t, t_{max} = 480$. Lastly, the goal and start points form the diagonal corners of a rectangle in the simulation xy frame and if the drone flies more than 8m outside this rectangle then it is considered to be out-of-bounds.

C. Simulation and Environments

Simulation Setup We use the open-source Flightmare platform, specifically from the DodgeDrone ICRA 2022 Competition, which contains both a high-fidelity physics engine for quadrotor dynamics simulation and a graphics engine built on Unity that handles camera rendering [23]. The simulation cycle represented in Figure 2 is as follows: the current drone state is used by the Unity module to render the depth image view which is fed alongside the state information into the RL policy which then outputs velocity commands. The velocity commands are processed by Flightmare's built-in low-level controller to issue rotor outputs, which the dynamics modeling engine then uses to calculate the next drone state after a Δt timestep. The simulated dynamics are based on a 0.752kg Kingfisher drone. The state is fed back into Unity and this loop continues at a rate of 12 iterations per simulated second ($\Delta t = 0.085$). During training, an environment with 150 independently simulated drones is used to collect rollouts in parallel.

Environment Generation We generate two environment types: privileged learning and curriculum environments. The privileged learning environment consists of randomly placed pillars with randomized start/end locations throughout. The curriculum environment as depicted in the fine-tuning portion of Figure 4 consist of three regions in increasing order of difficulty. Region 1 is a grid of clusters where a random number of obstacles are placed within an inner circle and start/end points are placed at varying radii in an outer circle. Region 2 consists of variable-length walls placed at random angles with start/end points being placed randomly throughout at a maximum of 40m away from each other. Region 3 has variable-length walls being placed in rectangular sub-region with start/end positions being placed on either side of each sub-region. It is arranged so that the drone's trajectory passes orthogonal to the walls' orientations (+/- some angle) and as such is forced to make larger path adjustments to avoid them.

D. Training

We train our policy using the 3 steps visualized in Figure 4: trajectory planning, privileged learning, and fine-tuning.

Optimal Trajectory Planning Given the privileged learning environment, we use Unity to convert the environment mesh into a 3D point cloud with a resolution of 0.15m. The point cloud is fed into a hierarchical motion planner [15] that generates a set of dynamically feasible optimal trajectories for the given start/goal pairs.

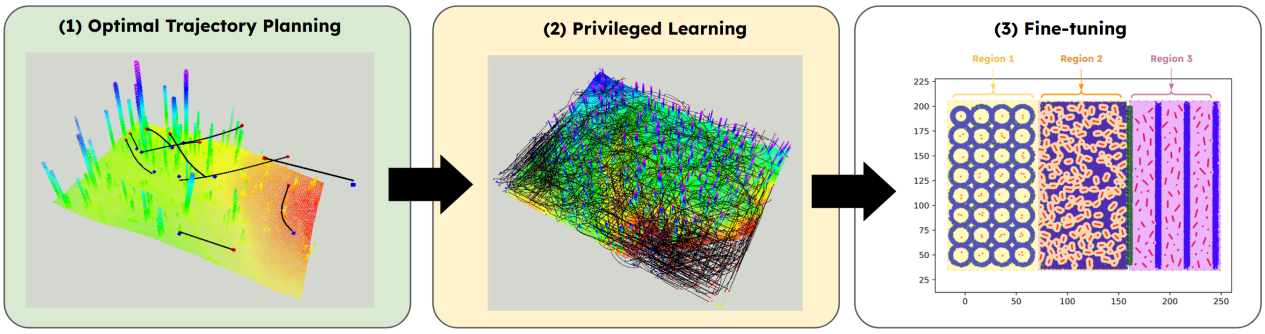

Fig. 4. An overview of the 3 stages of the training pipeline. In (1), we generate trajectories using a motion planner with perfect environment knowledge (3D Point Cloud). Then in (2), we train an initial policy in the pillar environment with the optimal trajectories as supervisory signal. Finally in (3), we fine-tune the policy in the mixed curriculum environment without any optimal trajectories.

Privileged Learning with Trajectories We teach the policy the basics of optimal time flight behavior in a simple environment of spaced-out pillars. This is aided via privileged learning whereby if the drone's position at each time step is near the optimal trajectory then it is given a reward. This differs from distillation methods since we are not trying to emulate the trajectories exactly, but rather they are a guide that helps the policy converge by giving it intermediate rewards. Furthermore, the position-based rewards rather than action-based rewards means that the supervisory signal acts more on the planning behavior rather than the control.

Fine-tuning in Curriculum Environment After a base network is trained, we further train the policy in the more complex curriculum environment –illustrated in Figure 4– to teach the policy navigation strategies, to make it robust to potential environment variations, and to prevent memorization. Over the course of fine-tuning, we start off in the easiest region (1) and gradually mix in start/goal pairs from the harder regions (2/3) as training progresses. The reason we do not use optimal trajectories during fine-tuning is that the space of optimal trajectories is multi-modal (e.g. equally valid to go right or left around a single obstacle). While this effect is negligible with the simpler environments of the privileged learning step, in the more complex curriculum environment it is exacerbated since the range of near-optimal trajectories is exponential, therefore meaning that rewards for adhering to a particular trajectory is a poor signal that actually impedes learning.

E. Bridging Sim2Real

To close the Sim2Real gap, we focus on improving the policy's noise robustness and action smoothness. The reason for the former is intuitive – in the real-world there is noise across the state and action space: noise in the depth image, drift in the IMU/VIO measurements, randomness in latency, errors in rotor control, etc. The intuition behind striving for smoother policies is twofold. Firstly, while jerky movements may be optimal in simulation with an idealized model of flight dynamics, in the real-world they strain the rotors and degrade control authority. Secondly, jerky movement profiles do not generalize well to quadrotors with different characteristics (moments of inertia, thrust maps, body drags, etc.), thereby impeding zero-shot transfer.

Reward Shaping A key problem with using reference velocities as an output is that we can not enforce trajectory smoothness via methods such as interpolation. Thus, to discourage high-frequency jerky commands, we apply penalties to yaw rate magnitude, acceleration, and jerk. To discourage low-frequency oscillations which lead to wavy motion, we apply a reward if the drone is angled towards the goal and if the its trajectory matches the smooth optimal trajectory.

Domain Randomization We apply a small amount of normal noise ($\pm 2\%$) to positional state measurements and a 5×5 Gaussian Blur with a randomized σ in the interval [0.1, 0.7] to the depth input image. We have found empirically that Gaussian Blur degradation most closely approximates the type of noise encountered in real-world outdoor testing where the long range (>20m) and variable lighting lead to large variations in measurements near edges if the disparity between the foreground and background depth is large. For the action space, we apply normal noise ($\pm 5\%$) to the drone's simulated movement at each time step to emulate inaccuracies in control output and outdoor factors such as wind. In addition, we also varied the flight dynamics parameters such as mass and moments of inertia to simulate different drone testbed characteristics. We also add a lag factor to approximate system delays such as forward pass processing time and publish-poll time differences between ROS nodes.

IV. EXPERIMENTS

A. Simulation Evaluation

We conduct a simulation study to analyze the policy's capabilities in a more elaborate environment than can be set up in the real world. To do so, we conduct 4 experiments across varying maximum speeds ($v_{x,max}$) of 1.0m/s - 4.0m/s. In each experiment, we measure the policy's performance over 1000 runs in an obstacle course randomly generated with the same region types as the curriculum environment, thereby capturing a variety of situations. Example trajectories are shown in Figure 5. The success rate shown in Table II stays relatively constant across 1.0 - 3.0m/s, however it drops off at 4.0m/s due to the lower agility and decreased necessary reaction time at higher speeds.

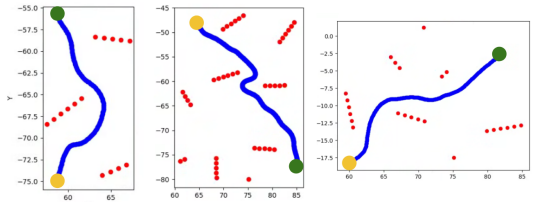

Fig. 5. Visualization of example simulation trials. Yellow is the start and green is the goal.

TABLE II

SIMULATION OUTCOMES ACROSS MAXIMUM SPEEDS

Speed ($v_{x,max}$)	Success	Crash	Timeout	Out of Bounds
1.0 m/s	992	8	0	0
2.0 m/s	985	13	1	1
3.0 m/s	989	10	1	0
4.0 m/s	967	28	5	2

B. Flight System Overview

All real-world tests are conducted on a DJI Matrice 300 RTK (6.3kg, 0.81m span) with additional mounted hardware as seen in Figure 1. A Zed2i device with a stereo camera and built-in IMU is used to collect 672×376 resolution depth images and state estimates augmented by Visual Intertial Odometry (VIO). In the depth image, we remove any measurements over 20m to account for the Zed2i's maximum effective range and under 0.2m to account for the rotor blades being in view before downscaling to 192×108. The stereo depth calculations and policy are run on a Jetson AGX Xavier orchestrated with ROS. The policy interfaces with the drone using DJI's onboard SDK to send velocity commands. The full pipeline, from the depth image capture to the command being sent to the drone, runs at 12Hz. This system can ideally be deployed on other drones given the proper mounting hardware and software interface.

C. Real-World Experiments

To validate that the policy successfully transfers to the real-world, we constructed 3 obstacle formations: wall, triangle, and inverted triangle. Each consists of foam pillars being positioned in different locations with the drone having to navigate through them to reach the goal point. For each environment, we test 3 different scenarios where the start and goal point are in different locations relative to the obstacle formation: left, center, and right. The environments and scenarios are shown in Figure 7 and the results are shown in Table III. For the first set of experiments with VIO device positioning, we performed 2 trials for each combination of scenario and environment for a total of 18 trials. During the VIO trials, we only use VIO for navigation, however we still collect GNSS data for post-flight analysis as a form of ground-truth positioning. We also performed a second set of experiments with GNSS positioning of 2 trials for each of the environments as shown in Figure 8 and Table IV. For VIO trials the goal was relative to the drone's starting orientation e.g. +25m forward, whereas for GNSS trials the goal point was a lat/long coordinate. A run is declared to be successful if the drone stops within 1.0m of the goal as declared by the positioning device used by the policy. The device for the VIO trials is the Zed2i and the device for the GNSS trials is GPS. All runs operate at a speed $v_{x,max}$ of 3.0 m/s.

Fig. 6. Pictures of the actual testing environments at (a) Berkeley Marina and (b) Hearst Mining Circle on the UC Berkeley campus.

The experiments occurred in two locations shown in Figure 6: a field at Berkeley Marina and the Hearst Mining courtyard on the UC Berkeley campus. The field was open with no obstacles in view whereas the courtyard had uneven terrain, a sculpture, a pond, and surrounding buildings/trees. The GNSS experiments in Table IV were conducted in Hearst Mining courtyard in addition to the 6 center scenario trials for the VIO results of Table III, whose measurements are italicized for reference. The remaining 12 VIO trials of Table III were carried out in Berkeley Marina.

Beyond the new obstacle configurations themselves, the system had to contend with multiple domain shifts to successfully cross the Sim2Real gap. The most obvious one is that the policy had never trained with the dynamics of a DJI M300 and as such had to compensate for the different flight characteristics. The environment scenery itself was also substantially different from training with the sloped terrain and clutter such as trees, a sculpture, light poles, and buildings. There were out-of-distribution environmental factors such as crosswinds reaching up to 14 kph and non-ideal lighting conditions leading to additional sensor noise. Lastly, the VIO positioning from the Zed2i was oftentimes substantially inaccurate drifting up to 3m as evidenced by GNSS measurements.

In spite of the aforementioned Sim2Real domain gaps, the system achieved a high success rates across all 3 environments. The results can be seen in Figure 7 and Table III where s[m/s] is average speed across the run as measured by GNSS, t[s] is time from start to goal, and d[m] is actual measured distance to goal. As seen in Table III, the success rate across all environments and scenarios is 100% while the average distance to goal is 2.27m averaged over all three environments. This relatively large average distance to goal, also shown in the discrepancy between the orange and blue trajectories in Figure 7, are despite the fact that the VIO device believes it is within 1.0 meter of the goal 100% of the time. This reflects the positioning inaccuracy of the chosen VIO device which fundamentally limits the drone's ability to accurately reach the end goal.

The drone had an average speed of 2.58m/s across the trials which is close to the maximum speed limit of 3.0m/s, especially when considering the acceleration phase at the start and deceleration phase near the goal. This indicates that the policy was aiming to achieve the highest possible speed during the trials. To decouple the inaccuracy of the VIO device from the performance of our policy, we conducted

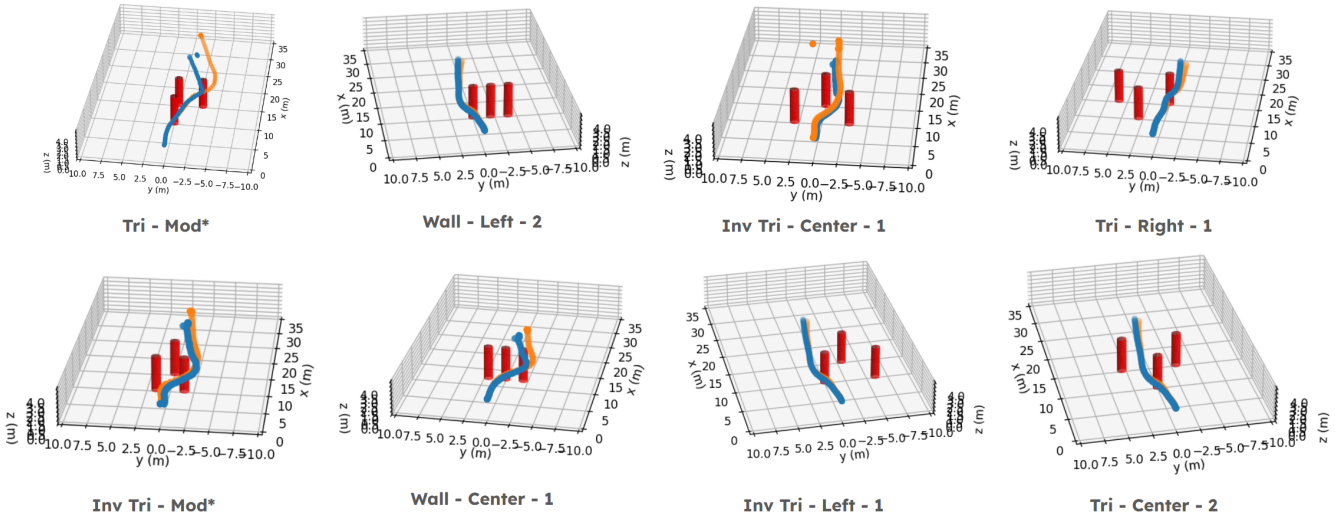

Fig. 7. 3D visualization of 8 runs, labeled in the form of environment - scenario - trial. Depicted are also modified environments labeled as Tri - Mod and Inv Tri - Mod which are described in the Discussion. The red pillars are the obstacles, the orange line is the GPS-measured trajectory, and the blue line is VIO device measured trajectory. Note that the blue line may seem like it is passing through obstacles, however that is due to VIO's inaccuracy.

TABLE III

VIO DEVICE POSITIONING TRIALS ACROSS ENVIRONMENT AND STARTING POSITION

Environment	**Scenario**	**Trial 1**			**Trial 2**			**Trial Avg.**			**Env Avg.**		**Overall**
		s [m/s]	t [s]	d [m]	s [m/s]	t [s]	d [m]	s [m/s]	t [s]	d [m]	Success [%]	d [m]	$\bar{d}$ [m]
Wall	Left	2.53	8.95	2.90	2.63	8.98	2.40	2.58	8.97	2.65			
	Center	*2.49*	*9.25*	*3.30*	*2.51*	*9.56*	*2.30*	2.50	9.41	2.80	**100% (6/6)**	2.65	
	Right	2.46	9.33	1.90	2.54	9.00	3.10	2.50	9.17	2.50			
Triangle	Left	2.54	8.99	1.05	2.58	9.05	1.85	2.56	9.02	1.40			2.27
	Center	*2.61*	*9.26*	*1.90*	*2.59*	*9.23*	*2.10*	2.60	9.25	2.00	**100% (6/6)**	1.95	
	Right	2.54	8.84	2.70	2.60	9.19	2.20	2.57	9.02	2.45			
Inverted Triangle	Left	2.63	8.94	2.60	2.71	8.83	2.90	2.67	8.89	2.75			
	Center	*2.62*	*9.1*	*1.60*	*2.53*	*9.08*	*2.10*	2.575	9.09	1.85	**100% (6/6)**	2.21	
	Right	2.55	9.28	1.10	2.83	8.07	3.00	2.69	8.68	2.05			

TABLE IV

GNSS-POSITIONING TRIALS ACROSS ENVIRONMENTS

Environment	**Trial 1**			**Trial 2**			**Trial Avg.**				Overall
	s [m/s]	t [s]	d [m]	s [m/s]	t [s]	d [m]	s [m/s]	t [s]	d [m]	Success [%]	$\bar{d}$ [m]
Wall	2.67	8.26	1.30	2.58	8.45	0.80	2.63	8.36	1.05	100% (2/2)	0.81
Triangle	2.46	8.60	0.52	2.55	8.80	0.78	2.51	8.70	0.65	100% (2/2)	
Inverted Triangle	2.56	8.34	0.60	2.60	8.46	0.84	2.58	8.40	0.72	100% (2/2)	

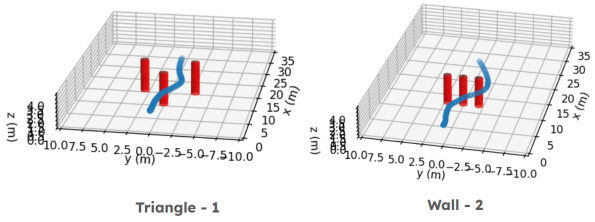

Fig. 8. 3D visualization of GPS runs. Blue line is GPS-measured trajectory.

trials using GNSS for positioning. The results shown in Figure 8 and Table IV indicate a 100% success rate and an average distance to goal of 0.81 which is about 2.8 times smaller than that of VIO device experiments.

D. Discussion and Conclusion

Overall, the key takeaway is that the system is able to perform in largely out-of-distribution conditions. The policy is able to both track a goal point and avoid obstacles in spite of degraded state estimation. It is also able to generalize from the simulation dynamics of a 0.7kg drone to a real-world drone 10 times the weight at a total of 7kg, thereby lending credence to the idea that robustly trained policies can be combined with low-level drone controllers without specific tuning. This is in contrast to [17] which tunes its policy specifically for the dynamics of a custom-built drone. These results can likely be attributed to the autoencoder

perception being trained on diverse environments and the use of regularization/randomization to train the policy to be robust to noise across the state-action space.

In addition to the standard trials, we built modified versions of the environments to test behavioral changes in response to certain factors. For instance, as shown with "Tri-Mod" in Figure 7, we created a variation of the Triangle environment where an additional obstacle was concealed behind the first, positioned along the drone's expected trajectory. Upon encountering the hidden obstacle within its field of view, the drone adapted its normal path to avoid it. Similarly, we tested a modified version of the Inverted Triangle environment shown with "Inv Tri-Mod" in Figure 7, reducing the gap from 6m to 3m to assess whether the drone would avoid passages with insufficient clearance relative to its size. As expected, the drone avoided the gap.

In terms of transferability to different hardware systems, one of the benefits of a learning-based approach is the extremely low computational cost of the policy itself. Our system (stereo camera input to action command output) has a latency of about 50ms. However, the policy including all surrounding ROS processes is able to complete a pass in only 1.5ms or about >600Hz on the Jetson GPU. The primary bottleneck of our system is the Zed2i's depth image computation process which occurs off-device on the Jetson and takes >90% of the latency i.e. 45ms. However, a depth sensor with on-device depth computation such as the Intel Realsense D435 or Luxonis OAK-D can result in a very fast and compute efficient system.

REFERENCES

[1] Anish Bhattacharya, Nishanth Rao, Dhruv Parikh, Pratik Kunapuli, Yuwei Wu, Yuezhan Tao, Nikolai Matni, and Vijay Kumar. Vision transformers for end-to-end vision-based quadrotor obstacle avoidance. *arXiv preprint arXiv:2405.10391*, 2024.

[2] Nathan Bucki, Junseok Lee, and Mark W. Mueller. Rectangular pyramid partitioning using integrated depth sensors (rappids): A fast planner for multicopter navigation. *IEEE Robotics and Automation Letters*, 5(3):4626–4633, 2020.

[3] Jaehoon Cho, Dongbo Min, Youngjung Kim, and Kwanghoon Sohn. Diml/cvl rgb-d dataset: 2m rgb-d images of natural indoor and outdoor scenes. *arXiv preprint arXiv:2110.11590*, 2021.

[4] Alessandro Devo, Jeffrey Mao, Gabriele Costante, and Giuseppe Loianno. Autonomous single-image drone exploration with deep reinforcement learning and mixed reality. *IEEE Robotics and Automation Letters*, 7(2):5031–5038, 2022.

[5] Robin Ferede, Christophe De Wagter, Dario Izzo, and Guido C. H. E. de Croon. End-to-end reinforcement learning for time-optimal quadcopter flight. *arXiv preprint arXiv:2311.16948*, 2023.

[6] Jiawei Fu, Yunlong Song, Yan Wu, Fisher Yu, and Davide Scaramuzza. Learning deep sensorimotor policies for vision-based autonomous drone racing. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5243–5250. IEEE, 2023.

[7] Sven Gronauer, Daniel Stümke, and Klaus Diepold. Comparing quadrotor control policies for zero-shot reinforcement learning under uncertainty and partial observability. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7508–7514. IEEE, 2023.

[8] Drew Hanover, Antonio Loquercio, Leonard Bauersfeld, Angel Romero, Robert Penicka, and et al. Autonomous drone racing: A survey. *IEEE Transactions on Robotics*, 2024.

[9] Jialin Ji, Zhepei Wang, Yingjian Wang, Chao Xu, and Fei Gao. Mapless-Planner: A robust and fast planning framework for aggressive autonomous flight without map fusion. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11720–11726, 2021.

[10] Katie Kang, Suneel Belkhale, Gregory Kahn, Pieter Abbeel, and Sergey Levine. Generalization through simulation: Integrating simulated and real data into deep reinforcement learning for vision-based autonomous flight. In *2019 international conference on robotics and automation (ICRA)*, pages 6008–6014. IEEE, 2019.

[11] Elia Kaufmann, Leonard Bauersfeld, Antonio Loquercio, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Champion-level drone racing using deep reinforcement learning. *Nature*, 620(7976):982–987, 2023.

[12] Mihir Kulkarni and Kostas Alexis. Reinforcement learning for collision-free flight exploiting deep collision encoding. *arXiv preprint arXiv:2402.03947*, 2024.

[13] Mihir Kulkarni, Theodor J. L. Forgaard, and Kostas Alexis. Aerial gym – isaac gym simulator for aerial robots, 2023.

[14] Jonathan Lee, Abhishek Rathod, Kshitij Goel, John Stecklein, and Wennie Tabib. Rapid quadrotor navigation in diverse environments using an onboard depth camera. In *2024 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, November 2024.

[15] Sikang Liu, Kartik Mohta, Nikolay Atanasov, and Vijay Kumar. Search-based motion planning for aggressive flight in se (3). *IEEE Robotics and Automation Letters*, 3(3):2439–2446, 2018.

[16] Xu Liu, Guilherme V. Nardari, Fernando Cladera Ojeda, Yuezhan Tao, Alex Zhou, Thomas Donnelly, Chao Qu, Steven W. Chen, Roseli A. F. Romero, Camillo J. Taylor, and Vijay Kumar. Large-scale autonomous flight with real-time semantic slam under dense forest canopy. *IEEE Robotics and Automation Letters*, 7(2):5512–5519, 2022.

[17] Antonio Loquercio, Elia Kaufmann, René Ranftl, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Learning high-speed flight in the wild. *Science Robotics*, 6(59):eabg5810, 2021.

[18] Antonio Loquercio, Ana I. Maqueda, Carlos R. Del Blanco, and Davide Scaramuzza. Dronet: Learning to fly by driving. *IEEE Robotics and Automation Letters*, 3:1088–1095, 2018.

[19] Kimberly McGuire, Guido de Croon, Christophe De Wagter, Karl Tuyls, and Hilbert Kappen. Efficient optical flow and stereo vision for velocity estimation and obstacle avoidance on an autonomous pocket drone. *IEEE Robotics and Automation Letters*, 2(2):1070–1076, 2017.

[20] Daniel Mellinger and Vijay Kumar. Minimum snap trajectory generation and control for quadrotors. In *2011 IEEE international conference on robotics and automation*, pages 2520–2525. IEEE, 2011.

[21] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[22] Abhik Singla, Sindhu Padakandla, and Shalabh Bhatnagar. Memory-based deep reinforcement learning for obstacle avoidance in uav with limited environment knowledge. *IEEE transactions on intelligent transportation systems*, 22(1):107–118, 2019.

[23] Yunlong Song, Elia Kaufmann, Leonard Bauersfeld, Antonio Loquercio, and Davide. Scaramuzza. Icra 2022 dodgedrone challenge: Vision-based agile drone flight. 2022.

[24] Yunlong Song, Selim Naji, Elia Kaufmann, Antonio Loquercio, and Davide Scaramuzza. Flightmare: A flexible quadrotor simulator. In *Conference on Robot Learning*, pages 1147–1157. PMLR, 2021.

[25] Yunlong Song, Kexin Shi, Robert Penicka, and Davide Scaramuzza. Learning perception-aware agile flight in cluttered environments. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1989–1995. IEEE, 2023.

[26] Vaibhav Viswanathan, Eric Dexheimer, Guanrui Li, Giuseppe Loianno, Michael Kaess, and Sebastian Scherer. Efficient trajectory library filtering for quadrotor flight in unknown environments. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2510–2517, 2020.

[27] Wei Xiao, Zhaohan Feng, Ziyu Zhou, Jian Sun, Gang Wang, and Jie Chen. Time-optimal flight in cluttered environments via safe reinforcement learning. *arXiv preprint arXiv:2406.19646*, 2024.

[28] Xin Yang, Jingyu Chen, Yuanjie Dang, Hongcheng Luo, Yuesheng Tang, Chunyuan Liao, Peng Chen, and Kwang-Ting Cheng. Fast depth prediction and obstacle avoidance on a monocular drone using probabilistic convolutional neural network. *IEEE Transactions on Intelligent Transportation Systems*, 22(1):156–167, 2019.

[29] Haokun Zheng, Sidhant Rajadnya, and Avideh Zakhor. Monocular depth estimation for drone obstacle avoidance in indoor environments. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 10027–10034. IEEE, 2024.