This paper has been accepted for publication at Robotics: Science and Systems, 2024.

Demonstrating Agile Flight from Pixels without State Estimation

Ismail Geles*, Leonard Bauersfeld*, Angel Romero, Jiaxu Xing, Davide Scaramuzza

Robotics and Perception Group, University of Zurich, Switzerland

***Abstract*—Quadrotors are among the most agile flying robots. Despite recent advances in learning-based control and computer vision, autonomous drones still rely on explicit state estimation. On the other hand, human pilots only rely on a first-person-view video stream from the drone onboard camera to push the platform to its limits and fly robustly in unseen environments. To the best of our knowledge, we present the first vision-based quadrotor system that autonomously navigates through a sequence of gates at high speeds while directly mapping pixels to control commands. Like professional drone-racing pilots, our system does not use explicit state estimation and leverages the same control commands humans use (collective thrust and body rates). We demonstrate agile flight at speeds up to 40 km/h with accelerations up to 2 g. This is achieved by training vision-based policies with reinforcement learning (RL). The training is facilitated using an asymmetric actor-critic with access to privileged information. To overcome the computational complexity during image-based RL training, we use the inner edges of the gates as a sensor abstraction. This simple yet robust, task-relevant representation can be simulated during training without rendering images. During deployment, a Swin-transformer-based gate detector is used. Our approach enables autonomous agile flight with standard, off-the-shelf hardware. Although our demonstration focuses on drone racing, we believe that our method has an impact beyond drone racing and can serve as a foundation for future research into real-world applications in structured environments.**

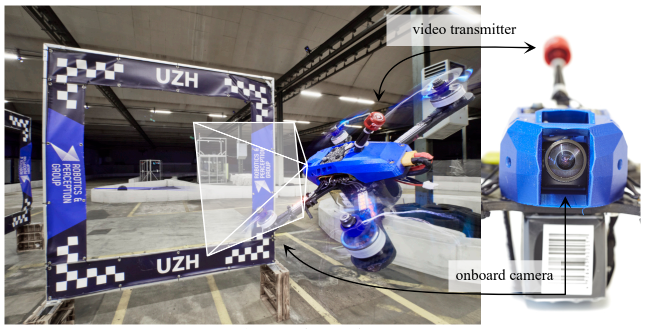
Supplementary Material

A narrated video with real-world experiments is available at: https://youtu.be/a1MSkTD-Tl8

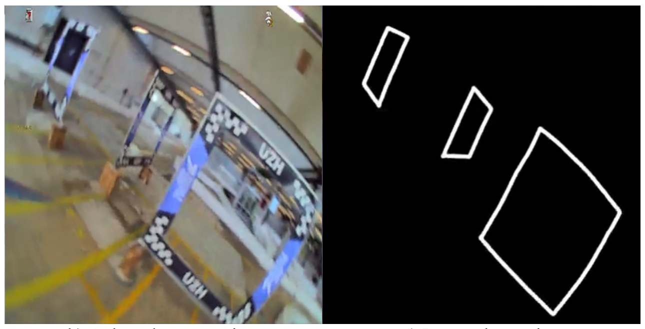
I. Introduction

Over fifteen years after the first autonomous, vision-based quadrotor flight [1], today's most agile autonomous vision-based quadrotors still rely on explicit state estimation [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]. This approach requires powerful, specialized hardware to perform all sensing and computation on board, as the fusion of visual and inertial information requires consistent and extremely low latencies [13]. This starkly contrasts with professional human pilots, who instead control the drone only based on a first-person-view (FPV) video stream from the drone's onboard camera. These pilots display impressive robustness and agility as they rely on very low-level commands: collective thrust and body rates.

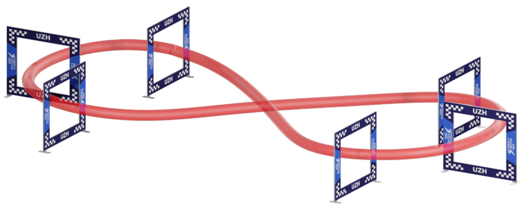
The ability to control agile autonomous drones directly from image pixels without explicit state estimation and without access to IMU measurements opens tremendous possibilities. First and foremost, it enables shifting the computations to a powerful ground-station PC (or the cloud), raising the opportunity to run complex algorithms and large neural networks in real time without being constrained by the computationally-limited hardware onboard the drone. Secondly, not requiring specialized hardware makes a one-to-one replacement of human pilots possible, thus yielding a very scalable approach for industrial applications.



a) Experimental setup



b) Onboard camera view c) Image abstraction



d) Figure 8 racetrack

Fig. 1. Our autonomous quadrotor can fly through a racetrack purely based on images from an onboard camera, without explicit state estimation. **a)** overview of our experimental setup consisting of racing gates, a quadrotor equipped with an onboard camera, and a video transmitter, **b)** onboard view from the camera, **c)** image abstraction used by the policy, **d)** overview over the racetrack.

Prior work on learning agile drone flight from visual input has focused on a setting where an explicit state estimate was input to the navigation policy [10, 11, 12]. In this same setting, an autonomous drone flown by a neural-network controller trained with RL beat the human world champions of drone racing in a head-to-head race [12]. In the context of learning a navigation policy purely from image pixels, prior works rely on the presence of a stabilizing onboard controller providing attitude and velocity estimates, severely limiting the achievable agility to below 2 m/s [14, 15].

Despite the recent achievements in reinforcement learning for mobile robotics control [12, 16, 17, 18, 19, 20] and the recent advances in deep visual odometry [21, 22, 23, 24], applying RL to learn agile-flight policies directly from pixels remains a challenging endeavor due to multiple factors. First, RL acquires knowledge through millions of trial-and-error interactions with the environment. For vision-based RL, sample efficiency is particularly important, as (i) the dimensionality of observations will be greatly increased, making exploration more challenging, and (ii) if image rendering is required during the interactions, it will significantly raise computational costs. Therefore, the task of exploring and learning efficiently in vision-based RL becomes more challenging. Second, in contrast with ground robots, flying robots capture images from very different viewpoints. Their ability to move freely in 3D space and the lack of constraints on the environment's appearance lead to diverse sensory information, exacerbating the difference between simulation and real-world deployment. Third, motion blur and rolling shutter effects degrade image quality at high speeds. Fourth, the simulation-to-reality gap is widened by highly nonlinear aerodynamic forces and the variability of environments, affecting both vehicle dynamics and sensory feedback. Fifth, quadrotors are unstable systems and the critical interdependence of perception and action necessitates rapid response times.

A. Contributions

To our knowledge, we demonstrate the first agile vision-based autonomous quadrotor flight in the real world without explicit state estimation in a structured environment featuring known racing gate landmarks. Our drone successfully flies a racing track (Fig. 1d) at speeds up to 40 km/h and accelerations up to 2 g's while only relying on a video stream transmitted by the drone's onboard camera (see Fig. 1a,b) to an offboard ground-station PC. We achieve a transmission latency of 33 ms at a rate of 60 Hz (1280 $\times$ 720 resolution).

The key enablers of this breakthrough are as follows. First, directly training vision-based policies from scratch is instrumental in ensuring robust control policies that explore large regions of the observation and action spaces during training. Reinforcement learning directly from pixels is made possible by our Atari-game-inspired [25] pixel-level abstraction (see 1c): inner gate edges are both task-relevant and efficient to simulate during training. Second, our reinforcement learning framework leverages an asymmetric actor-critic, where the critic has access to privileged information about the full state of the drone. Third, robust gate detections in the real world are achieved by a Swin-transformer-based [26] gate-detector trained on both rendered and real-world images but do not require real-world data from the deployment environment.

II. Related Work

A. Reinforcement Learning from Pixels

Deep reinforcement learning (RL) algorithms have achieved remarkable feats in simplified environments, such as games, providing efficient simulation platforms to benchmark the capabilities and limitations of AI agents [27]. This success is evident in surpassing human performance in Atari games [25] and mastering complex games like Go [28]. However, these environments, primarily offering discrete actions, pose challenges for continuous control tasks. The DeepMind Control Suite [29] addresses this gap, by presenting diverse environments for continuous control tasks. Yet, even in simulation, learning purely from pixels in these scenarios proves less sample-efficient, often yielding suboptimal performance compared to state-based learning. Recent efforts aim to improve this by refining low-dimensional visual representations alongside RL agents. Works such as SAC-AE [30], PlaNet [31], and CURL [32] explore techniques, including auto-encoders, future predictions, and contrastive unsupervised representations to bridge the performance-gap pixel-based to state-based policy performance gap.

Addressing the challenge of learning complex behaviors efficiently and robustly from pixels, some algorithms focus on data augmentation [33, 34]. Despite substantial progress in learning directly from pixels, these methods often cater to simulation benchmarking environments.

Diverging from the prevalent trend of simulation-centric methodologies, [35] adopts a model-based RL framework with a latent state-space model to learn from images, employing the Dreamer algorithm [36]. The learned tasks involve vision-based pick-and-place object manipulation and 2D maneuvering with a wheeled robot. The authors emphasize the challenges of mastering real-world application complexities with pixel-based control as already their quadrupedal robot relies on explicit state information.

Sensorimotor policies, mapping camera observations to actions, are predominantly learned through extensive training data simulation, incorporating domain randomization, dynamics randomization, or additional pose information like joint angles [37, 38, 39]. Tasks of higher complexity in robotics often necessitate expert demonstrations, prompting a shift towards imitation learning (IL) methodologies as a prevalent choice, as opposed to the paradigm of learning from scratch using RL, showcased by recent works such as [40].

B. Vision-based flight

There has been a series of works that aim to fly a drone by using pixel-to-high-level commands from visual data [41, 14, 42, 43, 44]. These approaches are not trained through reinforcement learning and are more focused on the general navigation task, and therefore are far from demonstrating agile flight. In [10], the authors demonstrate low-level commands directly from feature representations in the image plane to perform aggressive flight maneuvers learning control policies via imitation learning.

Leaving the vision-based aspect aside, RL has been applied to low-level control of quadrotors [45, 46] and has been demonstrated to be superior to classical methods. In state-based agile flight, and more particularly, drone racing, RL methods have recently demonstrated high-speed flight [47, 48], even being able to outperform the state-of-the-art in model-based control for drone racing [20]. Furthermore, the versatility of RL-based controllers has allowed their adoption in autonomous, vision-based flight and several recent works have been showing ever-growing success.

One of the first uses of RL for quadrotor navigation is [49], where the authors train a vision-based RL policy using a CAD model to produce discrete 'forward', 'left', or 'right' velocity commands directly from pictures. However, this was only possible due to the drone's stabilizing onboard controller and VIO pipeline. Very recently, an autonomous quadrotor combining vision-based state estimation and RL-based control has beaten the human world champions of drone racing [12]. While this represents a milestone for robotics, the drone requires highly specialized hardware to run the state estimation and control onboard the vehicle and shows very limited robustness, crashing in 40 % of the races against human pilots. Additionally, to finetune the control policies of the racetrack, a motion-capture system is required for data collection, leading to reduced versatility.

In this work, we aim to overcome the limitation of specialized hardware and onboard computation by flying like professional human drone pilots—only from a video stream. This also increases the robustness of the system as much more complex algorithms can be run on an offboard ground station.

III. Methodology

We consider the task of vision-based agile quadrotor flight directly from pixels. The goal is to navigate through a sequence of gates (see Fig. 1) in the correct order as quickly as possible while only relying on a video stream from an onboard camera. In this section, we first describe the learned controller, its observation and action space, and its training procedure. Then, we detail the simulation environment for policy training, and finally, we discuss the gate detector, which is an essential part of successful real-world deployment.

A. Neural Controller

The neural controller has access to pixel observations from an onboard camera which are pre-processed in the form of a gate segmentation mask. Based on these observations, the controller has to infer control commands comprising a collective thrust and body rates (CTBR). By relying on the CTBR control modality, the drone's agility is maximized [50].

We consider the standard RL setting, consisting of an agent acting in an environment in discrete time steps within the Markov Decision Process (MDP) formulation. The MDP is described by the tuple $(\mathcal{O}, \mathcal{A}, p, p_0, \mathcal{R}, \gamma)$ with observation space $\mathcal{O}$, action space $\mathcal{A}$, transition probabilities between states p, initial state distribution p_0 and rewards $\mathbb{R}$ with discount factor γ.

1) Action Space: At each timestep t, the policy needs to output an action $\mathbf{a_t}$, also referred to as the control command, which is four-dimensional. It consists of a mass-normalized collective thrust c (the acceleration of the drone) and a body rate setpoint $\boldsymbol{\omega}_{\mathcal{B},\text{ref}}$. These commands are then mapped to the individual motor speeds by a low-level controller onboard the drone. This control modality is also used by expert human pilots [12, 51] and, in contrast to high-level control commands such as linear velocities, ensures maximal agility [50]. Furthermore, the low-level controller on the drone is not required to estimate the state (e.g. position, attitude, velocity) of the vehicle with this control modality.

2) Observation Space of the Actor: To compute the action, the policy has access to an observation $\boldsymbol{o}_t$ to infer the action $\boldsymbol{a}_t$. In this work, the observation consists of a continuous pixel-level gate-segmentation mask (see Fig. 1). Here, continuous refers to a non-binary segmentation where the mask value corresponds to a confidence estimate. Our choice of observation is motivated by two aspects: first, inner gate edges are a highly task-relevant abstraction that encodes information about the task (fly through gates) as well as about the current state of the vehicle (how the gates are seen depends on where the drone is and how it is oriented). Second, during training, the inner gate edges can be obtained using perspective projection without the need to render the complete image, leading to a significant speedup. Taking inspiration from [25], we downsample our observations to a resolution of 84×84 pixels.

In addition to the pixel observations, the control policy has access to a history of three past actions. This enables the policy to produce smooth control commands, as the controller is not purely reactive and aware of the past commands.

3) Observation Space of the Critic: To train the control policies with PPO we leverage an asymmetric critic that has access to privileged information. In addition to the observations provided to the actor, the critic also has access to the full simulation state. We define this full simulation state be the 20-dimensional vector $\mathbf{s} = [\mathbf{p}, \tilde{\mathbf{R}}, \mathbf{v}, \boldsymbol{\omega}, \mathbf{i}, \mathbf{d}]$, where $\mathbf{p} \in \mathbb{R}^3$ is the position of the drone, $\tilde{\mathbf{R}} \in \mathbb{R}^6$ is a vector consisting of the first two columns of the $\mathbf{R}_{\mathcal{WB}}$ [52], $\mathbf{v} \in \mathbb{R}^3$ and $\omega \in \mathbb{R}^3$ denote the linear and angular velocity of the drone, and $\mathbf{d} \in \mathbb{R}^3$ is the position of the next gate center relative to the current drone position.

The vector $\mathbf{i} \in \mathbb{R}^2$ encodes the gate index using a sine-cosine encoding to account for the periodic nature of flying multiple laps around the track. To avoid discontinuities when a gate is passed, we use a continuous gate index. Let i denote the (discrete) number of gates passed so far, and let $d = ||\mathbf{d}||$ denote the (scalar) distance to the next gate center. The continuous gate index i_c is given by

$$i_c = i + \frac{2}{1 + \exp(k \cdot d)} \ , \tag{1}$$

where $k = 5$ is an empirical smoothing factor. From this, the observation $\mathbf{i}$ can be computed as follows:

$$\mathbf{i} = \begin{bmatrix} \exp(-i_c) + \cos(\alpha \cdot i_c) \\ \exp(-i_c) + \sin(\alpha \cdot i_c) \end{bmatrix} \ , \tag{2}$$

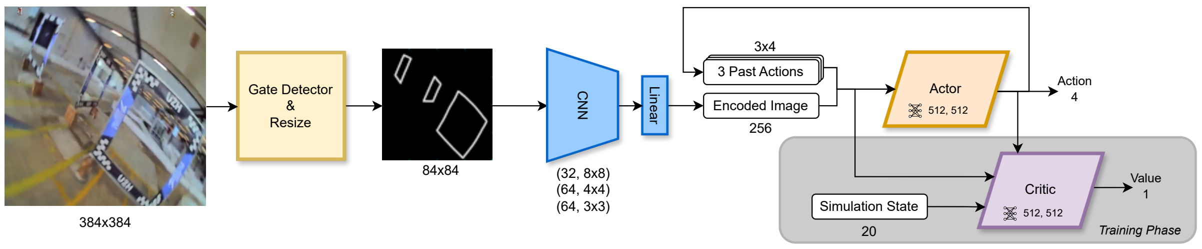


Fig. 2. The architecture of our method consists of a gate detector, which is trained to segment the inner gate edges. The gate detection is downsampled to a size of 84×84 and given as input to the three-layer CNN acting as a shared feature extractor for the asymmetric actor-critic framework. While training, we efficiently simulate the detected gates instead of using the detector. Both, the actor and critic are 2 hidden layer MLPs with 512 neurons each. The actor network has access to the current image encoding and the past three actions. The critic network, which is only used when training the policy, additionally receives privileged information about the state of the simulation environment.

where n_G is the total number of gates of the racetrack, and the frequency factor is computed as $\alpha = 2\pi/n_G$. The decaying exponential summand helps distinguish between the start of a flight and the subsequent laps.

4) Rewards: Similar to our previous works on drone racing, we use a dense reward to guide policy learning and encode the task of navigating through the racetrack. At each timestep, the reward r_t is computed as

$$r_t = r_t^{\text{prog}} + r_t^{\text{perc}} + r_t^{\text{pass}} - r_t^{\text{cmd}} - r_t^{\text{crash}} , \tag{3}$$

where the individual components are calculated as follows:

$$\begin{aligned} r_t^{\text{prog}} &= \lambda_1 \left(d_{t-1} - d_t\right) \\ r_t^{\text{perc}} &= \lambda_2 \exp\left(-\delta_{\text{cam}}^4\right) \\ r_t^{\text{cmd}} &= \lambda_3 ||\boldsymbol{a}_t|| + \lambda_4 ||\boldsymbol{a}_t - \boldsymbol{a}_{t-1}||^2 \\ r_t^{\text{pass}} &= \begin{cases} 1.0 - d_t, & \text{if passed gate at current timestep} \\ 0, & \text{otherwise} \end{cases} \\ r_t^{\text{crash}} &= \begin{cases} -4.0, & \text{if } p_z < 0 \text{ or in collision with gate} \\ 0, & \text{otherwise} \end{cases} \end{aligned} \tag{4}$$

where δ_{cam} is the angle between the optical axis of the camera and the vector pointing from the drone to the next gate center. The hyperparameters $\lambda_1 = 0.5$, $\lambda_2 = 0.025$, $\lambda_3 = 0.0005$, and $\lambda_4 = 0.0002$ are chosen empirically and trade-off speed and smoothness of a policy. The progress reward r^{prog} encourages fast flight to maximize progress along the track, the perception reward r^{perc} encourages orienting the camera towards the next gate to be passed, and the command smoothness penalty r^{cmd} penalizes large control actions as well as abrupt changes in the control command. The sparse gate pass reward and collision penalty guide the policy towards safe flight.

5) Network Architecture: The neural controller, shown in Fig. 2, consists of two components: A convolutional neural network (CNN) is used to extract a low-dimensional feature embedding from the high-dimensional pixel gate segmentation masks. Together with the last three actions, this embedding is used as an input to the two-layer MLP actor network. For training, the critic network has access to the CNN embedding and the full simulation state described above.

The reasoning behind including the last three actions in the MLP is to provide the controller with some information on which commands were sent before; this short-term memory enables smoother control commands.

B. Simulation

RL algorithms are known for being data-intensive to train. However, data collection in real-world scenarios is often impractical, especially considering the brittle nature of quadrotors. To circumvent this issue, we train our control policies exclusively in simulated environments. This approach not only protects the hardware but also allows for a more controlled training process.

In vision-based RL, the training process involves simulating not only the dynamics of quadrotors but also generating sensory observations, such as pixel-based visual data. This is a key distinction from state-based RL, where such a sensor simulation is not necessary. When it comes to rendering images for training an RL agent, there's a trade-off to consider: opting for lower-quality rendering can speed up the training process but at the cost of less realistic images. On the other hand, aiming for high-fidelity, photorealistic rendering results in a reduced sim2real gap, but significantly slows down the training due to the rendering complexity.

We overcome this obstacle through the use of inner gate edges as a task-related abstraction. These segmentation masks can be simulated very efficiently by projecting the gate edges onto the image plane.

1) Gate Observation: To compute the pixel observations, the simulation environment requires knowledge about the intrinsic parameters of the camera, which are obtained using the popular Kalibr [53] calibration toolbox. The lenses available for first-person-view drone racing cameras are not rectilinear but exhibit strong barrel distortion. Consequently, we use a double-sphere camera model for calibration [54], which is well suited to capture the distortion of wide-angle lenses.

To simulate the pixel observations, all gate edges in view are first sorted by their distance to the camera to correctly handle occlusions. Next, each visible gate edge is discretized into 5 points which are projected into the image plane and connected with a line. By subdividing each line into multiple segments before projecting them, the resulting simulated observations correctly account for lens distortion. To robustify the policy to poor gate detections 10% of the edge segments are corrupted and drawn at a random place in the image.

Our simulated gate observations are very efficient to compute and take less than 100 µs per frame, a speed unattainable with high-quality rendering.

2) Quadrotor Dynamics: In addition to the sensor observations, the simulator must compute the dynamics of the quadrotor. This section presents a brief overview of the simulator, but the reader is referred to [55, 12] for an in-depth explanation. The dynamics of the quadrotor are simulated as

$$\dot{\boldsymbol{x}} = \begin{bmatrix}\dot{\boldsymbol{p}}_{\mathcal{WB}}\\ \dot{\boldsymbol{q}}_{\mathcal{WB}}\\ \dot{\boldsymbol{v}}_{\mathcal{W}}\\ \dot{\boldsymbol{\omega}}_{\mathcal{B}}\\ \dot{\boldsymbol{\Omega}}\end{bmatrix} = \begin{bmatrix} \boldsymbol{v}_{\mathcal{W}} \\ \boldsymbol{q}_{\mathcal{WB}}\cdot\begin{bmatrix}0\\ \boldsymbol{\omega}_{\mathcal{B}}/2\end{bmatrix} \\ \frac{1}{m}\Big(\boldsymbol{q}_{\mathcal{WB}}\odot\left(\boldsymbol{f}_{\text{prop}}+\boldsymbol{f}_{\text{aero}}\right)\Big)+\boldsymbol{g}_{\mathcal{W}} \\ \boldsymbol{J}^{-1}\left(\boldsymbol{\tau}_{\text{prop}}+\boldsymbol{\tau}_{\text{aero}}-\boldsymbol{\omega}_{\mathcal{B}}\times\boldsymbol{J}\boldsymbol{\omega}_{\mathcal{B}}\right) \\ \frac{1}{k_{\text{mot}}}\left(\boldsymbol{\Omega}_{\text{ss}}-\boldsymbol{\Omega}\right)\end{bmatrix}, \quad (5)$$

where $\odot$ represents quaternion rotation, $\boldsymbol{p}_{\mathcal{WB}}$, $\boldsymbol{q}_{\mathcal{WB}}$, $\boldsymbol{v}_{\mathcal{W}}$, and $\boldsymbol{\omega}_{\mathcal{B}}$ denote the position, orientation quaternion, inertial velocity, and bodyrates of the quadrotor, respectively. The motor time constant is k_{mot} and the motor speeds Ω and Ω_{ss} are the actual and steady-state motor speeds, respectively. The matrix $\boldsymbol{J}$ is the quadcopter's inertia and $\boldsymbol{g}_{\mathcal{W}}$ denotes the gravity vector. The force and torque contributions of the propeller/motor unit as well as aerodynamic effects are denoted by $\boldsymbol{f}_{\text{prop}}, \boldsymbol{\tau}_{\text{prop}}$ and $\boldsymbol{f}_{\text{aero}}, \boldsymbol{\tau}_{\text{aero}}$, respectively.

To compute the force and torque contributions introduced above, we utilize either a first-principles model or a data-driven model. The first-principles model is based on blade-element-momentum (BEM) theory [55, 56, 57] and while it is very accurate, it is too slow to use for training the RL controller. For this reason, we use the widespread quadratic thrust and torque propeller model with a data-driven augmentation similar to [12] to obtain an accurate and computationally lightweight model. We refer to this simulation as our augmented simulator and exclusively train the controller in this setting.

C. Gate Detector

To deploy the control policies in the real world, we need a gate detector that takes as input images from the onboard camera and segments the inner gate edges. Compared to prior work that leverages gate detections [58, 12], this work does not rely on detected corners but uses a dense segmentation mask as an input. Following the advances in computer vision, the gate detector relies on a SwinTransformerV2 [26]. This architecture has proven to achieve very high performance in terms of accuracy while being very fast to compute on modern GPU architectures.

Obtaining a robust gate detector that reliably segments gates independent of the background, adverse lighting conditions, motion blur, and rolling shutter effects is crucial for this work. To ensure this, the gate detector is trained in a supervised fashion until convergence on a diverse dataset (see Fig. 3) comprising 80,000 labeled images. The dataset is composed as follows:

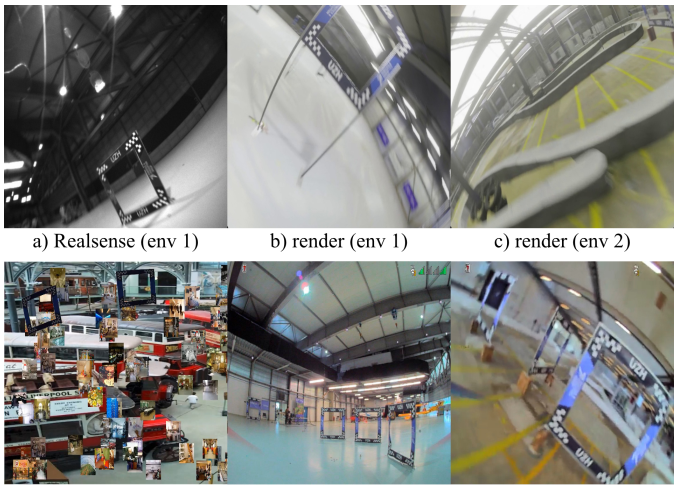


Fig. 3. The gate detector is trained on data collected from real and synthetic environments; however, it has never seen real images from environment 2, in which the system is deployed (f).

- 25,000 real-world images, taken with a different camera (Intel Realsense) to the one used in this work. They are reprojected to match the FPV-camera's intrinsics (Fig. 3a)
- 25,000 images are photorealistic renders with strong motion blur based on digital twins of two environments (Fig. 3b,c). Environment 2 corresponds to the deployment environment.
- 25,000 images are generated by rendering the gates on a diverse background obtained through randomly combining images from ImageNet [59] together (Fig. 3d)
- 5,000 real-world images from the FPV-camera are included in the dataset (Fig. 3e).

Note that none of these images are obtained with the FPV camera in the deployment environment, and consequently, the gate detector is not trained on real images from the deployment environment (Fig. 3f).

For deployment, the gate detector network is implemented in C++ using TensorRT to achieve maximum performance. The inference time of the gate detector (SwinV2-B model) is only 4 ms when deployed on an RTX 3090.

D. Policy Training

The control policy is trained through model-free reinforcement learning relying on PPO [60]. The training is entirely done in simulation, where a policy is trained for 400 million environment interactions.

To obtain a policy that is capable of zero-shot transfer to the real world, it must explore a sufficiently large part of the observation space as well as exhibit robustness to a sim2real gap. To address these aspects, we employ an improved sampling strategy that relies on an initial state buffer as well as domain randomization techniques.

1) Initial State Buffer: Reinforcement learning can suffer from catastrophic forgetting [61], where an agent becomes very good at performing the task and then unlearns how to recover from non-optimal states. This is caused by the policy only seeing a very narrow observation once it performs the task reliably and repeatedly. A strategy to mitigate this problem is the use of an initial state buffer [62] from which the environment samples a state when it resets itself after an episode is terminated.

In this work, we use a simplified version. The state buffer is used to store quadrotor states during training such that episodes are started with physically plausible, yet diverse states. We maintain a buffer of 10 possible initial states per gate. At the start of the training, the states are all initialized, such that the drone is positioned in the gate center and flies forward at a velocity of 2 m/s. When the agent passes a gate, its current state is added to the state buffer if it passes the gate with a sufficient margin to the gate edges. When the environment is reset, it samples a state from the buffer and perturbs the position, attitude, velocity, and body rate randomly. This effectively prevents mode collapse and catastrophic forgetting.

2) Domain Randomization: Similar to the RL agents presented in previous works [20, 12], we employ domain randomization to robustify a control policy against a sim2real gap in the system dynamics. Despite having access to very accurate models, drone-to-drone variation and complex non-linear aerodynamic effects necessitate robust policies. Furthermore, domain randomization prevents a policy essentially from memorizing a sequence of control commands and forces it to react to the environment.

During training, we vary the drone dynamics by $\pm 20\,\%$ in the thrust, body drag, and inertia parameters. The mass is randomized by $\pm 5\,\%$. The starting pose of the drone undergoes uniform sampling around the starting point, with $\pm 0.8\,\mathrm{m}$ variations in the $x - y$ plane, $\pm 0.6\,\mathrm{m}$ in the z axis, and $\pm 20\,\mathrm{deg}$ in attitude. The initial velocity is sampled within $\pm 0.8\,\mathrm{m/s}$, and initial body rates are randomized in $\pm 45\,\mathrm{deg/s}$. To better generalize to uncertainties in gate positions, we also randomize the gate position by $\pm 5\,\mathrm{cm}$ in each axis (x, y, z).

Therefore, the domain randomization procedure affects not only the drone's physical dynamics but also the environment conditions. It is thus fully incorporated into the full simulation state, which is used by the privileged critic and state-based policies.

IV. Experiments

After presenting the methodology in the previous section, we now aim to quantitatively evaluate the performance of our proposed system. The metrics used during this comparison will be (i) the success rate, quantifying how many runs successfully finish a three-lap race, (ii) the mean-gate-passing-error quantifying how far from the gate center the drone passes the gate, and (iii) the lap-time, measuring how agilely the control policy flies. Leveraging these metrics, we want to answer the following three research questions: (i) how does our pixel-based agent compare to a state-based agent on different track layouts, (ii) how sensitive is our agent to variations in the track layout and (iii) is our method able to robustly fly in the real world.

A. Simulation & HIL Experiments

To obtain reproducible results, we conduct the first part of our analysis in simulation as well as using hardware-in-the-loop (HIL) experiments. During such HIL experiments the physical drone is controlled in the real world, however, instead of using the gate detector for segmentation, we use our simulated pixel-observations. This is possible since a motion-capture system gives us access to the pose of the drone in real time. This HIL approach enables differentiating between gate-detector performance and the performance of the RL agent.

To assess the effectiveness of our approach, we conduct a comparative analysis between our pixel-based policy and two baselines. The first baseline is a pixel-based policy which we have trained with a symmetric architecture, e.g. omitting the privileged information in the critic. The second baseline is a state-based policy where both actor and critic are trained exclusively on the full simulation state. Figure 4 visualizes the training rewards of the three approaches together with the corresponding racetrack. From these plots, it is already obvious that the symmetric pixel-based agent is not able to fly the drone. The state-based agents learn much faster, thanks to the smaller and hightly task relevant observation space. This speed difference is especially pronounced when comparing training times, as the state-based agent is also eight times faster to train. However, given enough environment interactions, our asymmetric pixel-based agent performs similarly. For results on racetracks where no laps are flown (i.e. start to finish only), see appendix VI-A.

Table I shows a detailed evaluation of the agents' performance on all three racetracks. We conduct the evaluation both in the BEM simulator, our augmented simulator (training environment), as well as in the real world with hardware-in-the-loop simulation. The simulation evaluation is conducted by simulating policy rollouts in each setting using 64 environments with slightly perturbed starting positions and 1000 time steps. For hardware in the loop, we run the policy multiple times and fly three laps during each run.

The results clearly show that the asymmetric critic is crucial for the method's success. Furthermore, our asymmetric pixel-based agents come very close to the performance of the state-based agents in all metrics. Most notably, it achieves a 100% success rate during hardware-in-the-loop deployment of multiple experiments.

In addition, we can notice that a state-based policy trained using the methodology of Song et al. [63] is superior in terms of lap time. However, these policies complete the tracks independently of their viewing direction, which gives them the advantage of not necessarily looking toward the gates the drone has to pass through. Moreover, the actions are not as regularized as our approach, resulting in more aggressive maneuvers, sharper turns, and noticeably higher mean-gate-passing-errors (MGE). The MGE can be interpreted as a measure of the safety margin when passing gates, as a higher

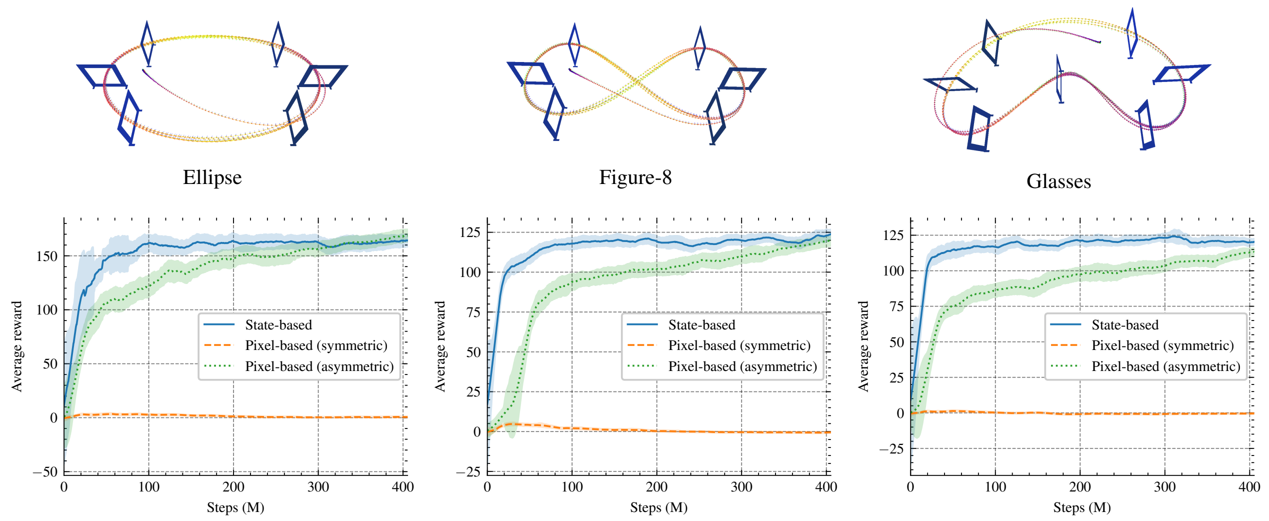


Fig. 4. The top row shows the three different racetracks with trajectories flown by the pixel-based policy in the augmented simulator, while the bottom row shows the reward progress during training. With the asymmetric actor-critic architecture, the average reward of our pixel-based agent converges to a similar value as the state-based agent. Excluding the privileged information for the critic network results in unsuccessful learning. The training process requires roughly 3 hours for the state-based policy, while the pixel-based policies take approximately one day to train for 400 million steps.

TABLE I
Simulation and HIL results. We compare the success rate (SR), mean-gate-passing-error (MGE), and the lap-time (LT) of our asymmetric pixel-based policy against four baselines. Three different racetracks are evaluated in simulation and HIL experiments, each with three laps. Pixel-based policies trained with our asymmetric actor-critic architecture perform significantly better than the symmetric architecture. We compare against the state-based approach of Song et al. [63] and a modified version, denoted by +Perc, which uses the same observations as [63] but with our perception-aware reward design. The best state-based result is underlined and the best pixel-based result is bold.

		BEM			Augmented			HIL		
Racetrack	Observation	SR [%]	MGE [m]	LT [sec]	SR [%]	MGE [m]	LT [sec]	SR [%]	MGE [m]	LT [sec]
Ellipse	State-based (Song et al. [63])	100.00	0.516	2.800	100.00	0.531	2.731	100.00	0.540	2.816
	State-based (Song et al. [63]) +Perc	100.00	0.199	2.810	100.00	0.196	2.729	100.00	0.221	2.817
	State-based (ours)	100.00	0.296	2.846	100.00	0.219	2.756	100.00	0.389	2.570
	Pixel-based (sym.) (ours)	0.00	**0.328**	-	0.00	0.268	-	0.00	-	-
	Pixel-based (asym.) (ours)	**93.75**	0.350	**3.072**	**90.60**	**0.154**	**2.902**	**100.00**	**0.381**	**3.318**
Figure-8	State-based (Song et al. [63])	100.00	0.446	3.600	100.00	0.438	3.588	100.00	0.414	3.648
	State-based (Song et al. [63]) +Perc	100.00	0.190	4.819	100.00	0.180	4.727	100.00	0.184	4.900
	State-based (ours)	100.00	0.200	4.833	100.00	0.190	4.741	100.00	0.367	4.703
	Pixel-based (sym.) (ours)	0.00	0.409	-	0.00	0.378	-	0.00	-	-
	Pixel-based (asym.) (ours)	**100.00**	**0.198**	**4.626**	**95.30**	**0.186**	**4.644**	**100.00**	**0.238**	**4.777**
Glasses	State-based (Song et al. [63])	100.00	0.471	3.479	100.00	0.451	3.497	100.00	0.455	3.570
	State-based (Song et al. [63]) +Perc	100.00	0.119	5.065	100.00	0.121	5.018	100.00	0.163	5.090
	State-based (ours)	100.00	0.151	5.102	100.00	0.163	4.985	100.00	0.191	5.157
	Pixel-based (sym.) (ours)	0.00	0.402	-	0.00	0.396	-	0.00	-	-
	Pixel-based (asym.) (ours)	**100.00**	**0.240**	**5.267**	**89.10**	**0.209**	**5.162**	**100.00**	**0.273**	**5.586**

deviation from the center of the gates makes crashes more likely but yields faster lap times. If we add the proposed perception-aware reward and action regularization to [63], then the overall performance of [63] is comparable to our state-based policy (see +Perc entries). The remaining lap-time difference to the pixel-based policies is caused by [63]+Perc being deployed with state-based information from a motion-capture system.

1) Sensitivity to Gate Positions: Next, we investigate the sensitivity of our pixel-based policy to slightly changed gate positions. By introducing gate position randomization, we aim to assess the generalizability of our policies across varying scenarios. This ablation provides valuable insights into how well our pixel-based approach adapts to changes in gate positions. In Figure 5, the results are visualized. At the beginning of each simulation experiment, the gates are randomly displaced by up to 50 cm in each direction. Consistent with the methodology employed in previous simulation experiments, each metric is derived from the average of 64 environment rollouts, each spanning 1000 time steps. Notably, even without explicit gate position information, our pixel-based approach achieves successful flights, demonstrating resilience in the face of

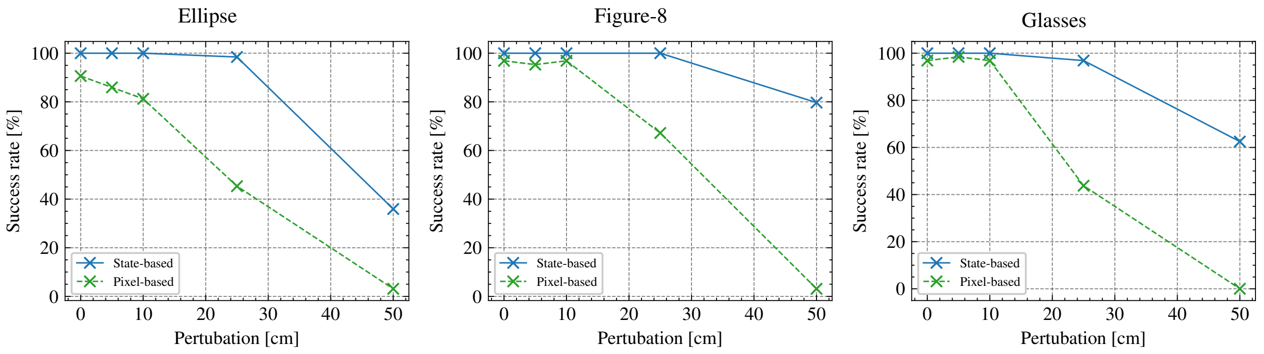


Fig. 5. The rate of successful trials in the augmented simulator ablated by perturbing the gate position in positive and negative x, y, z direction by a uniform distribution. While the state-based policy benefits from direct access to gate position information, the pixel-based policy demonstrates robustness by effectively inferring gate position solely from image observations, even in scenarios deviating from the training environment.

substantial gate-position perturbations. The state-based policy is superior because its input directly contains the ground-truth relative position of the next gate to be passed.

B. *Real-World Experiments*

In a final setup of experiments, we deploy our pixel-based policy together with the gate detector in the real world on the Figure 8 track. We use a modification of the *Agilicious* platform [64] for the real-world deployment. On this modified platform, the onboard computer is replaced with an RF receiver which receives the control commands from the ground station. The RF receiver is connected to the flight controller[1], which then executes the transmitted collective thrust and body rate commands. Additionally, the quadrotor is equipped with a low-latency video transmission system which sends the live video stream to the base computer. This configuration is similar to the one used by professional drone racing pilots and we measure at video stream latency of 33 ms. The gate detector is run directly on the images sent to the ground station.

Table II summarizes the real-world flights. For a more dynamic impression, the reader is advised to watch the supplementary video showcasing these experiments. We achieve direct zero-shot simulation-to-reality transfer of our pixel-based control policies. The success rate across 6 runs and a total of 20 flown laps is 100%. Compared to the state-based policy, the asymmetric pixel-based policy exhibits a slightly larger gate-passing error but flies a similar lap-time.

This result means that we have demonstrated agile flight directly from pixels without explicit state estimation, as our asymmetric pixel-based agent robustly navigates the track at speeds up to 40 km/h and accelerations up to 2 g.

[1]https://www.betaflight.com

TABLE II
Results in the real world flying without state estimation for 6 trials with 3 laps each on the Figure-8 racetrack. We compare the success rate (SR), mean-gate-passing-error (MGE), and the lap-time (LT) in the real-world using state-based and pixel-based observations.

Observation	SR [%]	MGE [m]	LT [sec]
State-based	100.00	0.367	4.703
Pixel-based (asym.)	100.00	0.491	4.683

V. Discussion and Conclusion

In this paper, we presented, to our knowledge, the first vision-based, agile quadrotor system that learns to directly map pixels to low-level control commands without explicit state estimation or access to IMU. This combination of observation and action space has been a long-standing milestone for visuomotor robotic intelligence. Through extensive experiments in simulation and real-world drone racing at speeds up to 40 km/h with accelerations up to 2 g, our methodology showcases the direct transfer of policies from simulation to reality with the appropriate abstractions of the visual input feed, akin to professional human drone pilots.

As shown in the experiments, the asymmetric actor-critic architecture plays a vital role in the approach's success. By leveraging privileged information about the drone's state in the environment, the critic network enhances its precision in assessing the actions taken by the actor, particularly in the context of pixel observations. This framework enables performance similar to a state-based policy, achieving comparable but marginally lower success rates in our evaluations. The inner gate edge abstraction enables training policies with up to 400 million environment interactions within a day. This makes it possible to train a pixel-based agent directly with RL, contributing to the robustness of the agent.

At present, a primary limitation of our approach is the limited memory of the neural controller, only considering three past actions. When the drone is oriented so that no gate is visible for multiple frames, the success rate drops drastically as our architecture's relatively short action history hinders recovery. This challenge can be effectively addressed by incorporating a recurrent architecture, allowing for prolonged memory retention. Furthermore, enhancing sample efficiency could be achieved by employing specialized algorithms tailored for learning from pixels.

While this work exclusively demonstrates autonomous vision-based flight without state-estimation for the task of drone racing, we believe that it has broader implications and will serve as a foundation for more application-oriented extensions. In tasks like autonomous indoor navigation or ship inspection, salient landmarks like doorways or manholes could be used instead of gates as their positions are known and their appearance is relatively consistent. Here, employing a separate the vision-encoder and RL-controller is beneficial as the landmark detector can be trained independently of the navigation policy. The latter can be trained in a simulation with access to a floorplan or blueprint of the deployment environment, potentially enabling diverse inspection tasks.

Another potential application of our method is to overcome situations where IMU information is not reliable, such as a straight-line flight at a constant speed. This flight profile, prevalent during powerline inspection, renders the IMU bias drift in a state-estimation pipeline unobservable. Our method on the other hand could rely on detecting powerline pylons and always fly towards the next visible pylon, effectively traversing along the powerline.

We realize that future research dedicated to obstacle avoidance, dynamic objects and unforseen changes to the environment is required for many real-world applications. Nevertheless, we believe that our work represents a major step forward in the development and understanding of fully autonomous robots that rely solely on visual data.

Acknowledgments

This work was supported by the European Union's Horizon Europe Research and Innovation Programme under grant agreement No. 101120732 (AUTOASSESS) and the European Research Council (ERC) under grant agreement No. 864042 (AGILEFLIGHT). The authors thank Chunwei Xing for his assistance in the development of the Swin-transformer-based gate detector.

References

[1] Michael Bloesch, Stephan Weiss, Davide Scaramuzza, and Roland Siegwart. Vision based mav navigation in unknown and unstructured environments. In *2010 IEEE International Conference on Robotics and Automation*, pages 21–28, 2010.

[2] Giuseppe Loianno and Davide Scaramuzza. Special issue on future challenges and opportunities in vision-based drone navigation. *Journal of Field Robotics*, 37(4):495–496, June 2020. ISSN 1556-4959. doi: 10.1002/rob.21962.

[3] Shaojie Shen, Yash Mulgaonkar, Nathan Michael, and Vijay Kumar. Vision-based state estimation and trajectory control towards high-speed flight with a quadrotor. In *Robotics: Science and Systems (RSS)*, 2013.

[4] Sikang Liu, M. Watterson, S. Tang, and V. Kumar. High speed navigation for quadrotors with limited onboard sensing. In *IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 1484–1491, 2016.

[5] G. Loianno, C. Brunner, G. McGrath, and V. Kumar. Estimation, control, and planning for aggressive flight with a small quadrotor with a single camera and IMU. *IEEE Robot. Autom. Lett.*, 2017.

[6] Davide Falanga, Elias Mueggler, Matthias Faessler, and Davide Scaramuzza. Aggressive quadrotor flight through narrow gaps with onboard sensing and computing using active vision. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 5774–5781. IEEE, 2017.

[7] Kartik Mohta, Michael Watterson, Yash Mulgaonkar, Sikang Liu, Chao Qu, Anurag Makineni, Kelsey Saulnier, Ke Sun, Alex Zhu, Jeffrey Delmerico, Konstantinos Karydis, Nikolay Atanasov, Giuseppe Loianno, Davide Scaramuzza, Kostas Daniilidis, Camillo Jose Taylor, and Vijay Kumar. Fast, autonomous flight in gps-denied and cluttered environments. *Journal of Field Robotics*, 35(1):101–120, 2018.

[8] Yi Lin, Fei Gao, Tong Qin, Wenliang Gao, Tianbo Liu, William Wu, Zhenfei Yang, and Shaojie Shen. Autonomous aerial navigation using monocular visual-inertial fusion. *Journal of Field Robotics*, 35(1):23–51, 2018.

[9] Yingjian Wang, Jialin Ji, Qianhao Wang, Chao Xu, and Fei Gao. Autonomous flights in dynamic environments with onboard vision. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1966–1973, 2021. doi: 10.1109/IROS51168.2021.9636117.

[10] Elia Kaufmann, Antonio Loquercio, René Ranftl, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Deep drone acrobatics. In *Proceedings of Robotics: Science and Systems*, Corvalis, Oregon, USA, July 2020.

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

[12] 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, Aug 2023. ISSN 1476-4687.

[13] Paul Furgale, Joern Rehder, and Roland Siegwart. Unified temporal and spatial calibration for multi-sensor systems. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1280–1286, 2013.

[14] Antonio Loquercio, Ana I Maqueda, Carlos R Del-Blanco, and Davide Scaramuzza. Dronet: Learning to fly by driving. *IEEE Robotics and Automation Letters*, 3(2):1088–1095, 2018.

[15] Fereshteh Sadeghi and Sergey Levine. CAD2RL: Real single-image flight without a single real image. In *Robotics: Science and Systems (RSS)*, pages 48–55, 2017.

[16] Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning quadrupedal locomotion over challenging terrain. *Science robotics*, 5(47):eabc5986, 2020.

[17] Gabriel Margolis, Ge Yang, Kartik Paigwar, Tao Chen, and Pulkit Agrawal. Rapid locomotion via reinforcement learning. In *Robotics: Science and Systems*, 2022.

[18] Takahiro Miki, Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning robust perceptive locomotion for quadrupedal robots in the wild. *Science Robotics*, 7(62):eabk2822, 2022.

[19] Zipeng Fu, Xuxin Cheng, and Deepak Pathak. Deep whole-body control: Learning a unified policy for manipulation and locomotion. In Karen Liu, Dana Kulic, and Jeff Ichnowski, editors, *Proceedings of The 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pages 138–149. PMLR, 14–18 Dec 2023.

[20] Yunlong Song, Angel Romero, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Reaching the limit in autonomous racing: Optimal control versus reinforcement learning. *Science Robotics*, 8(82):eadg1462, 2023.

[21] Zachary Teed and Jia Deng. Droid-slam: Deep visual slam for monocular, stereo, and rgb-d cameras. *Advances in Neural Information Processing Systems*, 34:16558–16569, 2021.

[22] Zachary Teed, Lahav Lipson, and Jia Deng. Deep patch visual odometry. *arXiv preprint arXiv:2208.04726*, 2022.

[23] Shunkai Li, Wang Xin, Yingdian Cao, Fei Xue, Zike Yan, and Hongbin Zha. Self-supervised deep visual odometry with online adaptation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6338–6347, 06 2020.

[24] Sen Wang, Ronald Clark, Hongkai Wen, and Niki Trigoni. DeepVO: Towards end-to-end visual odometry with deep recurrent convolutional neural networks. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 2043–2050. IEEE, 2017.

[25] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015. ISSN 0028-0836, 1476-4687.

[26] Ze Liu, Han Hu, Yutong Lin, Zhuliang Yao, Zhenda Xie, Yixuan Wei, Jia Ning, Yue Cao, Zheng Zhang, Li Dong, Furu Wei, and Baining Guo. Swin transformer v2: Scaling up capacity and resolution. In *International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[27] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, June 2013. ISSN 1076-9757.

[28] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, January 2016. ISSN 1476-4687. Number: 7587 Publisher: Nature Publishing Group.

[29] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy Lillicrap, and Martin Riedmiller. DeepMind Control Suite, January 2018. URL http://arxiv.org/abs/1801.00690. arXiv:1801.00690 [cs].

[30] Denis Yarats, Amy Zhang, Ilya Kostrikov, Brandon Amos, Joelle Pineau, and Rob Fergus. Improving sample efficiency in model-free reinforcement learning from images. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(12): 10674–10681, May 2021. ISSN 2374-3468, 2159-5399.

[31] Danijar Hafner, Timothy P. Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 2555–2565. PMLR, 2019.

[32] Michael Laskin, Aravind Srinivas, and Pieter Abbeel. CURL: Contrastive unsupervised representations for reinforcement learning. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 5639–5650. PMLR, 13–18 Jul 2020.

[33] Michael Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, Pieter Abbeel, and Aravind Srinivas. Reinforcement learning with augmented data. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[34] Denis Yarats, Rob Fergus, Alessandro Lazaric, and Lerrel Pinto. Mastering visual continuous control: Improved data-augmented reinforcement learning. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022.

[35] Philipp Wu, Alejandro Escontrela, Danijar Hafner, Pieter Abbeel, and Ken Goldberg. Daydreamer: World models for physical robot learning. In *Conference on Robot Learning (CoRL)*. PMLR, 2022.

[36] Danijar Hafner, Timothy P. Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[37] Andrei A. Rusu, Matej Vecerík, Thomas Rothörl, Nicolas Heess, Razvan Pascanu, and Raia Hadsell. Sim-to-real robot learning from pixels with progressive nets. In *1st Annual Conference on Robot Learning, CoRL 2017, Mountain View, California, USA, November 13-15, 2017, Proceedings*, volume 78 of *Proceedings of Machine Learning Research*, pages 262–270. PMLR, 2017.

[38] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2017, Vancouver, BC, Canada, September 24-28, 2017*, pages 23–30. IEEE, 2017.

[39] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *J. Mach. Learn. Res.*, 17:39:1–39:40, 2016.

[40] Zipeng Fu, Tony Z. Zhao, and Chelsea Finn. Mobile aloha: Learning bimanual mobile manipulation with low-cost whole-body teleoperation. In *arXiv*, 2024.

[41] Elia Kaufmann, Antonio Loquercio, Rene Ranftl, Alexey Dosovitskiy, Vladlen Koltun, and Davide Scaramuzza. Deep drone racing: Learning agile flight in dynamic environments. In Aude Billard, Anca Dragan, Jan Peters, and Jun Morimoto, editors, *Proceedings of The 2nd Conference on Robot Learning*, volume 87 of *Proceedings of Machine Learning Research*, pages 133–145. PMLR, 29–31 Oct 2018.

[42] Dhruv Shah, Ajay Sridhar, Arjun Bhorkar, Noriaki Hirose, and Sergey Levine. GNM: A General Navigation Model to Drive Any Robot. In *International Conference on Robotics and Automation (ICRA)*, 2023. URL https://arxiv.org/abs/2210.03370.

[43] Dhruv Shah, Ajay Sridhar, Nitish Dashora, Kyle Stachowicz, Kevin Black, Noriaki Hirose, and Sergey Levine. ViNT: A foundation model for visual navigation. In *7th Annual Conference on Robot Learning*, 2023. URL https://arxiv.org/abs/2306.14846.

[44] Ajay Sridhar, Dhruv Shah, Catherine Glossop, and Sergey Levine. NoMaD: Goal Masked Diffusion Policies for Navigation and Exploration. *arXiv pre-print*, 2023. URL https://arxiv.org/abs/2310.07896.

[45] William Koch, Renato Mancuso, Richard West, and Azer Bestavros. Reinforcement learning for uav attitude control. *ACM Transactions on Cyber-Physical Systems*, 3(2):1–21, 2019.

[46] Nathan O Lambert, Daniel S Drew, Joseph Yaconelli, Sergey Levine, Roberto Calandra, and Kristofer SJ Pister. Low-level control of a quadrotor with deep model-based reinforcement learning. *IEEE Robotics and Automation Letters*, 4(4):4224–4230, 2019.

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

[48] Jonas Eschmann, Dario Albani, and Giuseppe Loianno. Learning to fly in seconds. *arXiv e-prints*, pages arXiv–2311, 2023.

[49] Fereshteh Sadeghi and Sergey Levine. Cad2rl: Real single-image flight without a single real image. *arXiv preprint arXiv:1611.04201*, 2016.

[50] Elia Kaufmann, Leonard Bauersfeld, and Davide Scaramuzza. A benchmark comparison of learned control policies for agile quadrotor flight. In *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022.

[51] Christian Pfeiffer and Davide Scaramuzza. Human-piloted drone racing: Visual processing and control. *IEEE Robotics and Automation Letters*, 6(2):3467–3474, 2021.

[52] Yi Zhou, Connelly Barnes, Jingwan Lu, Jimei Yang, and Hao Li. On the continuity of rotation representations in neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 5745–5753. Computer Vision Foundation / IEEE, 2019.

[53] Luc Oth, Paul Furgale, Laurent Kneip, and Roland Siegwart. Rolling shutter camera calibration. In *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1360–1367, 2013.

[54] Vladyslav C. Usenko, Nikolaus Demmel, and Daniel Cremers. The double sphere camera model. *2018 International Conference on 3D Vision (3DV)*, pages 552–560, 2018.

[55] Leonard Bauersfeld, Elia Kaufmann, Philipp Foehn, Sihao Sun, and Davide Scaramuzza. Neurobem: Hybrid aerodynamic quadrotor model. *RSS: Robotics, Science, and Systems*, 2021.

[56] Raymond W Prouty. *Helicopter performance, stability, and control*. Krieger Pub Co, 1995.

[57] Rajan Gill and Raffaello D'Andrea. Propeller thrust and drag in forward flight. In *2017 IEEE Conference on Control Technology and Applications (CCTA)*, pages 73–79. IEEE, 2017.

[58] Philipp Foehn, Dario Brescianini, Elia Kaufmann, Titus Cieslewski, Mathias Gehrig, Manasi Muglikar, and Davide Scaramuzza. Alphapilot: Autonomous drone racing. *Autonomous Robots*, 46(1):307–320, 2022.

[59] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[60] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv e-prints*, 2017.

[61] Prakhar Kaushik, Alex Gain, Adam Kortylewski, and Alan Yuille. Understanding catastrophic forgetting and remembering in continual learning with optimal relevance mapping, 2021.

[62] Nico Messikommer, Yunlong Song, and Davide Scaramuzza. Contrastive initial state buffer for reinforcement learning, 2023.

[63] Yunlong Song, Angel Romero, Mathias Mueller, Vladlen Koltun, and Davide Scaramuzza. Reaching the limit in autonomous racing: Optimal control versus reinforcement learning. *Science Robotics*, page adg1462, 2023.

[64] Philipp Foehn, Elia Kaufmann, Angel Romero, Robert Penicka, Sihao Sun, Leonard Bauersfeld, Thomas Laengle, Giovanni Cioffi, Yunlong Song, Antonio Loquercio, and Davide Scaramuzza. Agilicious: Open-source and open-hardware agile quadrotor for vision-based flight. *Science Robotics*, 7(67): eabl6259, 2022.

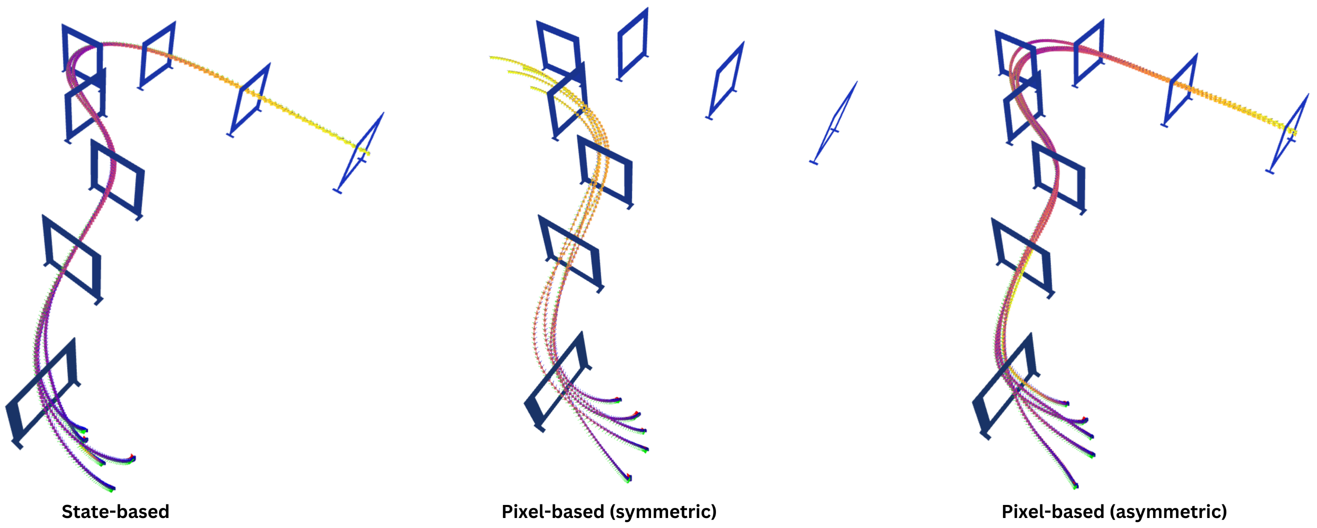


Fig. 6. Comparison of multiple rollouts with different initial conditions on an acyclic racetrack. From left to right: rollouts with a state-based policy, a pixel-based symmetric actor-critic policy, and the proposed pixel-based asymmetric architecture. The symmetric pixel-based policy is not able to learn this racetrack successfully.

VI. Appendix

A. Acyclic racetrack

In a drone-racing scenario, it is a common assumption that the race lasts multiple laps; thus, the task becomes cyclic. However, outside of drone racing, a drone might be confronted with an acyclic task consisting of flying from one point to another. To underline that our method also allows for this type of mission, we also evaluate it on an acyclic racetrack, where the episode terminates after the last gate. Besides this change, the observations, rewards, and actions remain the same. A racecourse without cycles demonstrates that this method is not limited to drone racing and can, therefore, generalize beyond other maps that are used for vision-based navigation. Notice that the symmetric actor-critic architecture failed to complete this track entirely. In contrast, the proposed asymmetric architecture successfully navigated the track in multiple runs with varying starting positions, similar to the state-based policy, see Figure 6.

The corresponding average reward while training the acyclic track is visualized in Figure 7. Since the episode terminates after the last gate with a terminal reward of $r_t^{\text{term}} = 10$, the total reward is lower than the previously discussed cyclic racetracks reported in Figure 4, where episodes could go up to 30 seconds (assuming no collision before) with a simulation step frequency of 50 Hz.

B. Hyperparameters

We train our state- and vision-based policies using Proximal Policy Optimization (PPO), simulating 100 environments in parallel to collect rollouts. The hyperparameters listed in Table III are used to train all the policies. The only exception is the acyclic track, which requires a lower discount factor $\gamma = 0.98$ for robust learning, since the episodes are significantly shorter than tracks where the drone may complete multiple laps in up to 30 seconds.

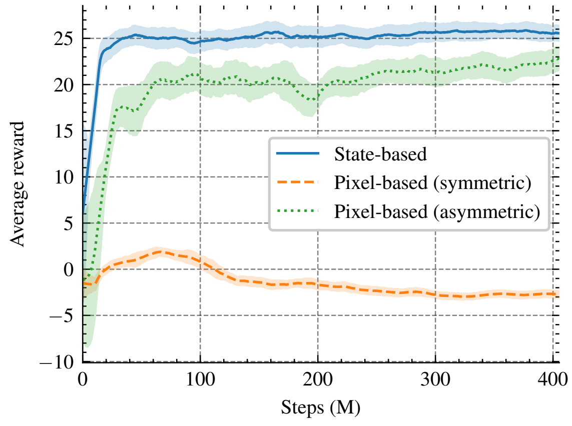


Fig. 7. Reward progress over the number of simulation time-steps for the acyclic track. The symmetric actor-critic is not able to complete the whole track.

TABLE III
PPO hyperparameters to train state- and vision-based policies.

Parameter	Value
learning rate	3e-4 linear decay to 1e-5
discount factor	0.995
GAE-λ	0.95
learning epochs	10
clip range	0.2
entropy coefficient	0.001
batch size	25000
policy network MLP	[512, 512]
value network MLP	[512, 512]
CNN-encoder latent dimension	256