Deep Drone Racing: From Simulation to Reality With Domain Randomization

Antonio Loquercio, Elia Kaufmann, René Ranftl, Alexey Dosovitskiy, Vladlen Koltun, and Davide Scaramuzza

***Abstract*—Dynamically changing environments, unreliable state estimation, and operation under severe resource constraints are fundamental challenges that limit the deployment of small autonomous drones. We address these challenges in the context of autonomous, vision-based drone racing in dynamic environments. A racing drone must traverse a track with possibly moving gates at high speed. We enable this functionality by combining the performance of a state-of-the-art planning and control system with the perceptual awareness of a convolutional neural network. The resulting modular system is both platform independent and domain independent: it is trained in simulation and deployed on a physical quadrotor without any fine-tuning. The abundance of simulated data, generated via domain randomization, makes our system robust to changes of illumination and gate appearance. To the best of our knowledge, our approach is the first to demonstrate *zero-shot sim-to-real* transfer on the task of agile drone flight. We extensively test the precision and robustness of our system, both in simulation and on a physical platform, and show significant improvements over the state of the art.**

***Index Terms*—Drone racing, learning agile flight, learning for control.**

I. Introduction

DRONE racing is a popular sport in which professional pilots fly small quadrotors through complex tracks at high speeds (see Fig. 1). Drone pilots undergo years of training to master the sensorimotor skills involved in racing. Such skills would also be valuable to autonomous systems in applications such as disaster response or structure inspection, where drones must be able to quickly and safely fly through complex dynamic environments [1].

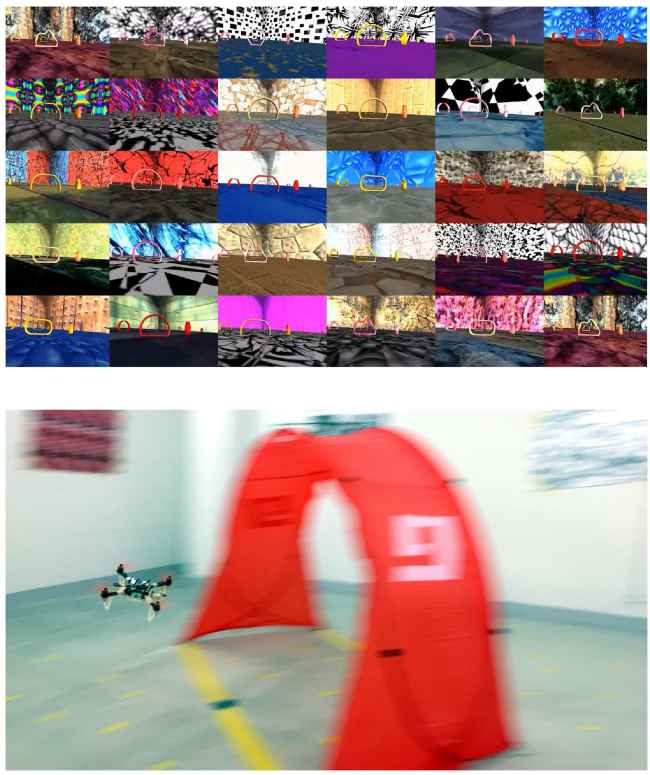

Fig. 1. The Perception block of our system, represented by a convolutional neural network (CNN), is trained *only* with nonphotorealistic simulation data. Due to the abundance of such data, generated with domain randomization, the trained CNN can be deployed on a physical quadrotor without any fine-tuning.

Developing a fully autonomous racing drone is difficult due to challenges that span dynamics modeling, on-board perception, localization and mapping, trajectory generation, and optimal control. For this reason, autonomous drone racing has attracted significant interest from the research community, giving rise to multiple autonomous drone racing competitions [2], [3].

One approach to autonomous racing is to fly through the course by tracking a precomputed global trajectory. However, global trajectory tracking requires to know the race-track layout in advance, along with highly accurate state estimation, which current methods are still not able to provide [4]–[6]. Indeed, visual-inertial odometry (VIO) [4], [5] is subject to drift in

Manuscript received March 20, 2019; accepted August 7, 2019. Date of publication October 21, 2019; date of current version February 4, 2020. This work was supported in part by the Intel Network on Intelligent Systems, the Swiss National Center of Competence Research Robotics, through the Swiss National Science Foundation, and in part by the Swiss National Science Foundation European Research Council Starting Grant. This article was recommended for publication by Editor A. Billard upon evaluation of the reviewers' comments. *(Antonio Loquercio and Elia Kaufmann contributed equally to this work.) (Corresponding author: Antonio Loquercio.)*

A. Loquercio, E. Kaufmann, and D. Scaramuzza are with the Robotic and Perception Group, Department of Informatics, University of Zürich, 8006 Zürich, Switzerland, and also with the Department of Neuroinformatics, University of Zürich and ETH Zürich, 8057 Zürich, Switzerland (e-mail: loquercio@ifi.uzh.ch; elia.kaufmann92@gmail.com; davide.scaramuzza@ieee.org).

R. Ranftl, A. Dosovitskiy, and V. Koltun are with the Intelligent Systems Lab., Intel (e-mail: rene.ranftl@intel.com; adosovitskiy@gmail.com; vkoltun@gmail.com).

This article has supplementary downloadable material available at http://ieeexplore.ieee.org, provided by the authors. Supplementary videos, source code, and trained networks can also be found on the project page: http://rpg.ifi.uzh.ch/research_drone_racing.html

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TRO.2019.2942989

estimation over time. Simulateneous Localization and Mapping (SLAM) methods can reduce drift by relocalizing in a previously generated, globally consistent map. However, enforcing global consistency leads to increased computational demands that strain the limits of on-board processing. In addition, regardless of drift, both odometry and SLAM pipelines enable navigation only in a predominantly static world, where waypoints and collision-free trajectories can be statically defined. Generating and tracking a global trajectory would therefore fail in applications where the path to be followed cannot be defined *a priori*. This is usually the case for professional drone competitions since gates can be moved from one lap to another.

In this article, we take a step toward autonomous, vision-based drone racing in dynamic environments. Instead of relying on globally consistent state estimates, our approach deploys a CNN to identify waypoints in local body-frame coordinates. This eliminates the problem of drift and simultaneously enables our system to navigate through dynamic environments. The network-predicted waypoints are then fed to a state-of-the-art planner [7] and tracker [8], which generate a short trajectory segment and corresponding motor commands to reach the desired location. The resulting system combines the perceptual awareness of CNNs with the precision offered by state-of-the-art planners and controllers, getting the best of both worlds. The approach is both powerful and lightweight: all computations run fully onboard.

An earlier version of this article [9] (Best System Paper Award at the Conference on Robotic Learning, 2018) demonstrated the potential of our approach both in simulation and on a physical platform. In both domains, our system could perform complex navigation tasks, such as seeking a moving gate or racing through a dynamic track, with higher performance than state-of-the-art, highly engineered systems. In this article, we extend the approach to generalize to environments and conditions not seen at training time. In addition, we evaluate the effect of design parameters on closed-loop control performance, and analyze the computation-accuracy tradeoffs in the system design.

In the earlier version [9], the perception system was track specific: it required a substantial amount of training data from the target race track. Therefore, significant changes in the track layout, background appearance, or lighting would hurt performance. In order to increase the generalization abilities and robustness of our perception system, we propose to use domain randomization [10]. The idea is to randomize during data collection all the factors to which the system must be invariant, i.e., illumination, viewpoint, gate appearance, and background. We show that domain randomization leads to an increase in closed-loop performance relative to our earlier work [9] when evaluated in environments or conditions not seen at training time. Specifically, we demonstrate performance increases of up to 300% in simulation (see Fig. 6) and up to 36% in real-world experiments (see Fig. 14).

Interestingly, the perception system becomes invariant not only to specific environments and conditions but also to the training domain. We show that after training purely in nonphotorealistic simulation, the perception system can be deployed on a physical quadrotor that successfully races in the real world. On real tracks, the policy learned in simulation has comparable performance to one trained with real data, thus alleviating the need for tedious data collection in the physical world.

II. Related Work

Pushing a robotic platform to the limits of handling gives rise to fundamental challenges for both perception and control. On the perception side, motion blur, challenging lighting conditions, and aliasing can cause severe drift in vision-based state estimation [4], [11], [12]. Other sensory modalities, e.g., LIDAR or event-based cameras, could partially alleviate these problems [13], [14]. Those sensors are however either too bulky or too expensive to be used on small racing quadrotors. Moreover, state-of-the-art state estimation methods are designed for a predominantly static world, where no dynamic changes to the environment occur.

From the control perspective, plenty of work has been done to enable high-speed navigation, both in the context of autonomous drones [7], [15], [16] and autonomous cars [17]–[20]. However, the inherent difficulties of state estimation make these methods difficult to adapt for small, agile quadrotors that must rely solely on on-board sensing and computing. We will now discuss approaches that have been proposed to overcome the aforementioned problems.

A. Data-Driven Algorithms for Autonomous Navigation

A recent line of work, focused mainly on autonomous driving, has explored data-driven approaches that tightly couple perception and control [21]–[24]. These methods provide several interesting advantages, e.g., robustness against drifts in state estimation [21], [22] and the possibility to learn from failures [24]. The idea of learning a navigation policy end-to-end from data has also been applied in the context of autonomous, vision-based drone flight [25]–[27]. To overcome the problem of acquiring a large amount of annotated data to train a policy, Loquercio *et al.* [26] proposed to use data from ground vehicles, whereas Gandhi *et al.* [27] devised a method for automated data collection from the platform itself. Despite their advantages, end-to-end navigation policies suffer from high sample complexity and low generalization to conditions not seen at training time. This hinders their application to contexts where the platform is required to fly at high speed in dynamic environments. To alleviate some of these problems while retaining the advantages of data-driven methods, a number of works propose to structure the navigation system into two modules: perception and control [28]–[32]. This kind of modularity has proven to be particularly important for transferring sensorimotor systems across different tasks [29], [31] and application domains [30], [32].

We employ a variant of this perception-control modularization in this article. However, in contrast to prior work, we enable high-speed, agile flight by making the output of our neural perception module compatible with fast and accurate model-based trajectory planners and trackers.

B. Drone Racing

The popularity of drone racing has recently kindled significant interest in the robotics research community. The classic solution to this problem is image-based visual servoing, where a robot is given a set of target locations in the form of reference images or patterns. Target locations are then identified and tracked with hand-crafted detectors [33]–[35]. However, the handcrafted detectors used by these approaches quickly become unreliable in the presence of occlusions, partial visibility, and motion blur. To overcome the shortcomings of classic image-based visual servoing, recent work proposed to use a learning-based approach for localizing the next target [36]. The main problem of this kind of approach is, however, limited agility. Image-based visual servoing is reliable when the difference between the current and reference images is small, which is not always the case under fast motion.

Another approach to autonomous drone racing is to learn end-to-end navigation policies via imitation learning [37]. Methods of this type usually predict low-level control commands, in the form of body rates and thrust, directly from images. Therefore, they are agnostic to drift in state estimation and can potentially operate in dynamic environments, if enough training data are available. However, despite showing promising results in simulated environments, these approaches still suffer from the typical problems of end-to-end navigation: 1) limited generalization to new environments and platforms and 2) difficulties in deployment to real platforms due to high computational requirements (desired inference rate for agile quadrotor control is much higher than what current on-board hardware allows).

To facilitate robustness in the face of unreliable state estimation and dynamic environments, while also addressing the generalization and feasibility challenges, we use modularization. On one hand, we take advantage of the perceptual awareness of CNNs to produce navigation commands from images. On the other hand, we benefit from the high speed and reliability of classic control pipelines for generation of low-level controls.

C. Transfer From Simulation to Reality

Learning navigation policies from real data has a shortcoming: high cost of generating training data in the physical world. Data need to be carefully collected and annotated, which can involve significant time and resources. To address this problem, a recent line of work has investigated the possibility of training a policy in simulation and then deploying it on a real system. Work on transfer of sensorimotor control policies has mainly dealt with manual grasping and manipulation [38]–[43]. In driving scenarios, synthetic data were mainly used to train perception systems for high-level tasks, such as semantic segmentation and object detection [44], [45]. One exception is the work of Müller *et al.* [32], which uses modularization to deploy a control policy learned in simulation on a physical ground vehicle. Domain transfer has also been used for drone control: Sadeghi and Levine [25] learned a collision avoidance policy by using three-dimensional (3-D) simulation with extensive domain randomization.

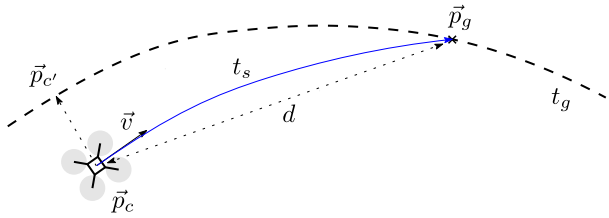

Fig. 2. Pose $\vec{p}_c$ of the quadrotor is projected on the global trajectory t_g to find the point $\vec{p}_{c'}$. The point at distance d from the current quadrotor position $\vec{p}_c$, which belongs to t_g in the forward direction with respect to $\vec{p}_{c'}$, defines the desired goal position $\vec{p}_g$. To push the quadrotor toward the reference trajectory t_g, a short trajectory segment t_s is planned and tracked in a receding horizon fashion.

Akin to many of the aforementioned methods, we use domain randomization [10] and modularization [32] to increase generalization and achieve sim-to-real transfer. Our work applies these techniques to drone racing. Specifically, we identify the most important factors for generalization and transfer with extensive analyses and ablation studies.

III. Method

We address the problem of robust, agile flight of a quadrotor in a dynamic environment. Our approach makes use of two subsystems: perception and control. The perception system uses a CNN to predict a goal direction in local image coordinates, together with a desired navigation speed, from a single image collected by a forward-facing camera. The control system uses the navigation goal produced by the perception system to generate a minimum-jerk trajectory [7] that is tracked by a low-level controller [8]. In the following, we describe the subsystems in more detail.

Perception system: The goal of the perception system is to analyze the image and provide a desired flight direction and navigation speed for the robot. We implement the perception system by a convolutional network. The network takes as input a 300×200 pixel RGB image, captured from the on-board camera, and outputs a tuple $\{\vec{x}, v\}$, where $\vec{x} \in [-1, 1]^2$ is a 2-D vector that encodes the direction to the new goal in normalized image coordinates, and $v \in [0, 1]$ is a normalized desired speed to approach it. To allow for on-board computing, we employ a modification of the DroNet architecture of Loquercio *et al.* [26]. In Section IV-C, we will present the details of our architecture, which was designed to optimize the tradeoff between accuracy and inference time. With our hardware setup, the network achieves an inference rate of 15 frames per second while running concurrently with the full control stack. The system is trained by imitating an automatically computed expert policy, as explained in Section III-A.

Control system: Given the tuple $\{\vec{x}, v\}$, the control system generates low-level commands. To convert the goal position $\vec{x}$ from 2-D normalized image coordinates to 3-D local frame coordinates, we back-project the image coordinates $\vec{x}$ along the camera projection ray and derive the goal point at a depth equal to the prediction horizon d (see Fig. 2). We found setting d proportional to the normalized platform speed v predicted by

the network to work well. The desired quadrotor speed v_{des} is computed by rescaling the predicted normalized speed v by a user-specified maximum speed $v_{\max}$: $v_{\text{des}} = v_{\max} \cdot v$. This way, with a single trained network, the user can control the aggressiveness of flight by varying the maximum speed. Once p_g in the quadrotor's body frame and v_{des} are available, a state interception trajectory t_s is computed to reach the goal position (see Fig. 2). Since we run all computations onboard, we use computationally efficient minimum-jerk trajectories [7] to generate t_s. To track t_s, i.e., to compute the low-level control commands, we employ the control scheme proposed by Faessler *et al.* [8].

A. Training Procedure

We train the perception system with imitation learning, using automatically generated globally optimal trajectories as a source of supervision. To generate these trajectories, we make the assumption that at training time the location of each gate of the race track, expressed in a common reference frame, is known. Additionally, we assume that at training time the quadrotor has access to accurate state estimates with respect to the latter reference frame. Note however that at test time no privileged information is needed and the quadrotor relies on image data only. The overall training setup is illustrated in Fig. 2.

Expert policy: We first compute a global trajectory t_g that passes through all gates of the track, using the minimum-snap trajectory implementation from Mellinger and Kumar [15]. To generate training data for the perception network, we implement an expert policy that follows the reference trajectory. Given a quadrotor position $\vec{p}_c \in \mathbb{R}^3$, we compute the closest point $\vec{p}_{c'} \in \mathbb{R}^3$ on the global reference trajectory. The desired position $\vec{p}_g \in \mathbb{R}^3$ is defined as the point on the global reference trajectory the distance of which from $\vec{p}_c$ is equal to the prediction horizon $d \in \mathbb{R}$. We project the desired position $\vec{p}_g$ onto the image plane of the forward facing camera to generate the ground truth normalized image coordinates $\vec{x}_g$ corresponding to the goal direction. The desired speed v_g is defined as the speed of the reference trajectory at $\vec{p}_{c'}$ normalized by the maximum speed achieved along t_g.

Data collection: To train the network, we collect a dataset of state estimates and corresponding camera images. Using the global reference trajectory, we evaluate the expert policy on each of these samples and use the result as the ground truth for training. An important property of this training procedure is that it is agnostic to how exactly the training dataset is collected. We use this flexibility to select the most suitable data collection method when training in simulation and in the real world. The key consideration here is how to deal with the domain shift between training and test time. In our scenario, this domain shift mainly manifests itself when the quadrotor flies far from the reference trajectory t_g. In simulation, we employed a variant of DAgger [46], which uses the expert policy to recover whenever the learned policy deviates far from the reference trajectory. Repeating the same procedure in the real world would be infeasible: allowing a partially trained network to control a unmanned aerial vehicle (UAV) would pose a high risk of crashing and breaking the platform. Instead, we manually carried the quadrotor through the track and ensured a sufficient coverage of off-trajectory positions.

Generating data in simulation: In our simulation experiment, we perform a modified version of DAgger [46] to train our flying policy. On the data collected through the expert policy (see Section III-A) (in our case, we let the expert policy fly for 40 s), the network is trained for ten epochs on the accumulated data. In the following run, the trained network is predicting actions, which are only executed if they keep the quadrotor within a margin ϵ from the global trajectory. In case the network's action violates this constraint, the expert policy is executed, generating a new training sample. This procedure is an automated form of DAgger [46] and allows the network to recover when deviating from the global trajectory. After another 40 s of data generation, the network is retrained on all the accumulated data for ten epochs. As soon as the network performs well on a given margin ϵ, the margin is increased. This process repeats until the network can eventually complete the whole track without help of the expert policy. In our simulation experiments, the margin ϵ was set to 0.5 m after the first training iteration. The margin was incremented by 0.5 m as soon as the network could complete the track with limited help from the expert policy (less than 50 expert actions needed). For experiments on the static track, 20 k images were collected, whereas for dynamic experiments 100 k images of random gate positions were generated.

Generating data in the real world: For safety reasons, it is not possible to apply DAgger for data collection in the real world. Therefore, we ensure sufficient coverage of the possible actions by manually carrying the quadrotor through the track. During this procedure, which we call *handheld* mode, the expert policy is constantly generating training samples. Due to the drift of on-board state estimation, data are generated for a small part of the track before the quadrotor is reinitialized at a known position. For the experiment on the static track, 25 k images were collected, whereas for the dynamic experiment additional 15 k images were collected for different gate positions. For the narrow gap and occlusion experiments, 23 k images were collected.

Loss function: We train the network with a weighted MSE loss on point and velocity predictions

$$L = \|\vec{x} - \vec{x}_g\|^2 + \gamma(v - v_g)^2 \tag{1}$$

where $\vec{x}_g$ denotes the ground truth normalized image coordinates and v_g denotes the ground truth normalized speed. By cross-validation, we found the optimal weight to be $\gamma = 0.1$, even though the performance was mostly insensitive to this parameter (see the Appendix for details).

Dynamic environments: The described training data generation procedure is limited to static environments since the trajectory generation method is unable to take the changing geometry into account. How can we use it to train a perception system that would be able to cope with dynamic environments? Our key observation is that training on multiple static environments (for instance with varying gate positions) is sufficient to operate in dynamic environments at test time. We collect data from multiple layouts generated by moving the gates from their initial position. We compute a global reference trajectory for each layout and

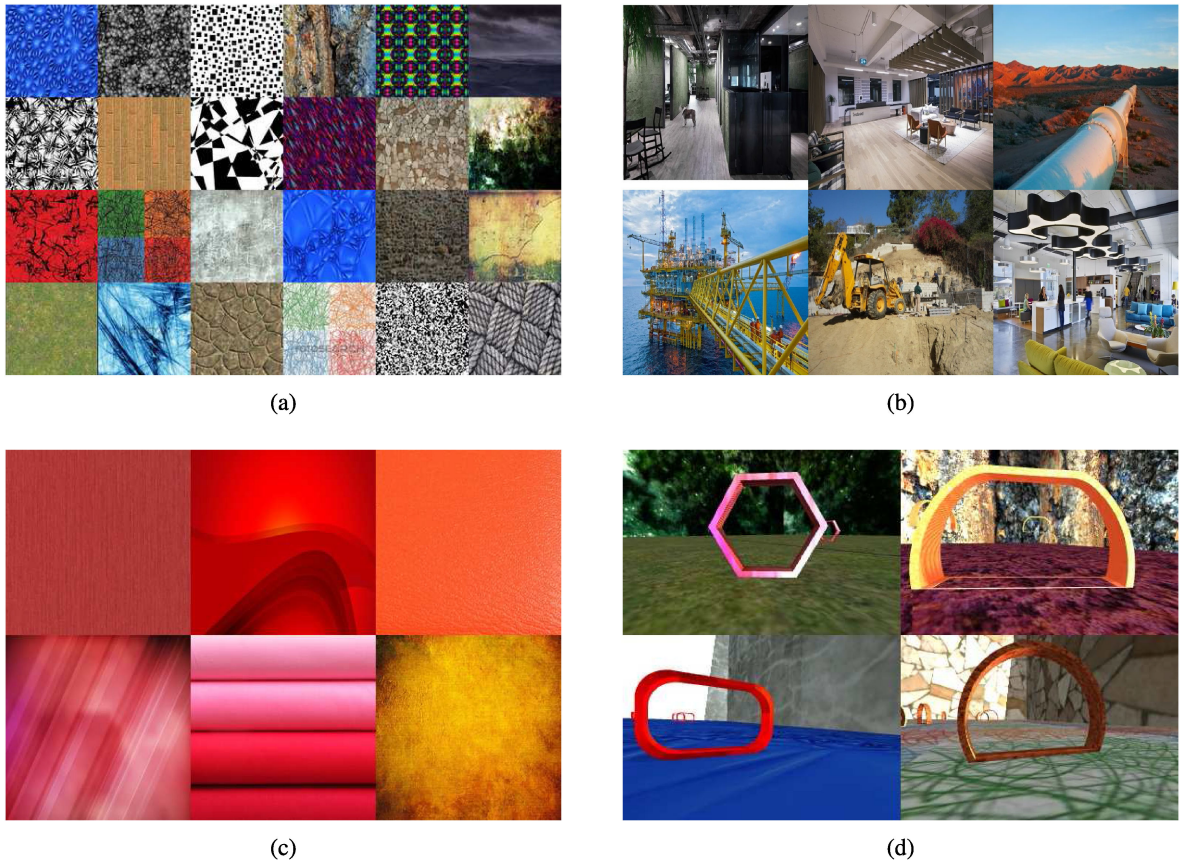

Fig. 3. To test the generalization abilities of our approach, we randomize the visual properties of the environment (background, illumination, gate shape, and gate texture). This figure illustrates the random textures and shapes applied both at training (a) and test time (b). For space reasons, not all examples are shown. In total, we used 30 random backgrounds during training and ten backgrounds during testing. We generated six different shapes of gates and used five of them for data generation and one for evaluation. Similarly, we used ten random gate textures during training and a different one during evaluation. (a) Random backgrounds used during training data generation. (b) Random backgrounds used at test time. (c) Gate textures. (d) Selection of training examples illustrating the gate shapes and variation in illumination properties.

train a network jointly on all of these. This simple approach supports generalization to dynamic tracks with the additional benefit of improving the robustness of the system.

Sim-to-real transfer: One of the big advantages of perception-control modularization is that it allows training the perception block exclusively in simulation and then directly applying on the real system by leaving the control part unchanged. As we will show in the experimental section, thanks to the abundance of simulated data, it is possible to train policies that are extremely robust to changes in environmental conditions, such as illumination, viewpoint, gate appearance, and background. In order to collect diverse simulated data, we perform visual scene randomization in the simulated environment while keeping the approximate track layout fixed. Apart from randomizing visual scene properties, the data collection procedure remains unchanged.

We randomize the following visual scene properties:

i) the textures of the background, floor, and gates;
ii) the shape of the gates;
iii) the lighting in the scene.

For i), we apply distinct random textures to background and floor from a pool of 30 diverse synthetic textures [see Fig. 3(a)]. The gate textures are drawn from a pool of ten mainly red/orange textures [see Fig. 3(c)]. For gate shape randomization ii), we create six gate shapes of roughly the same size as the original gate. Fig. 3(d) illustrates four of the different gate shapes used for data collection. To randomize illumination conditions iii), we perturb the ambient and emissive light properties of all textures (background, floor, gates). Both properties are drawn separately for background, floor, and gates from uniform distributions with support $[0, 1]$ for the ambient property and $[0, 0.3]$ for the emissive property.

While the textures applied during data collection are synthetic, the textures applied to background and floor at test time represent common indoor and outdoor environments [see Fig. 3(b)]. For testing we use held-out configurations of gate shape and texture not seen during training.

B. Trajectory Generation

Generation of global trajectory: Both in simulation and in real-world experiments, a global trajectory is used to generate ground truth labels. To generate the trajectory, we use the implementation of Mellinger and Kumar [15]. The trajectory is generated by providing a set of waypoints to pass through a

maximum velocity to achieve, as well as constraints on maximum thrust and body rates. Note that the speed on the global trajectory is not constant. As waypoints, the centers of the gates are used. Furthermore, the trajectory can be shaped by additional waypoints, for example, if it would pass close to a wall otherwise. In both simulation and real-world experiments, the maximum normalized thrust along the trajectory was set to 18 ms^{-2} and the maximum roll and pitch rate to 1.5 rad s^{-1}. The maximum speed was chosen based on the dimensions of the track. For the large simulated track, a maximum speed of 10 ms^{-1} was chosen, whereas on the smaller real-world track 6 ms^{-1} was chosen.

Generation of trajectory segments: The proposed navigation approach relies on constant recomputation of trajectory segments t_s based on the output of a CNN. Implemented as state-interception trajectories, t_s can be computed by specifying a start state, goal state, and a desired execution time. The velocity predicted by the network is used to compute the desired execution time of the trajectory segment t_s. While the start state of the trajectory segment is fully defined by the quadrotor's current position, velocity, and acceleration, the end state is only constrained by the goal position p_g, leaving velocity and acceleration in that state unconstrained. This is, however, not an issue since only the first part of each trajectory segment is executed in a receding horizon fashion. Indeed, any time a new network prediction is available, a new state interception trajectory t_s is calculated.

The goal position p_g is dependent on the prediction horizon d (see Section III-A), which directly influences the aggressiveness of a maneuver. Since the shape of the trajectory is only constrained by the start state and end state, reducing the prediction horizon decreases the lateral deviation from the straight-line connection of start state and end state but also leads to more aggressive maneuvers. Therefore, a long prediction horizon is usually required on straight and fast parts of the track, whereas a short prediction horizon performs better in tight turns and in proximity of gates. A long prediction horizon leads to a smoother flight pattern, usually required on straight and fast parts of the track. Conversely, a short horizon performs more agile maneuvers, usually required in tight turns and in the proximity of gates.

The generation of the goal position p_g differs from training to test time. At training time, the quadrotor's current position is projected onto the global trajectory and propagated by a prediction horizon d_{train}. At test time, the output of the network is back-projected along the camera projection ray by a planning length d_{test}.

At training time, we define the prediction horizon d_{train} as a function of distance from the last gate and the next gate to be traversed

$$d_{\text{train}} = \max\left(d_{\min}, \min\left(\|s_{\text{last}}\|, \|s_{\text{next}}\|\right)\right) \tag{2}$$

where $s_{\text{last}} \in \mathbb{R}^3$ and $s_{\text{next}} \in \mathbb{R}^3$ are the distances to the corresponding gates and $d_{\min}$ represents the minimum prediction horizon. The minimum distance between the last and the next gate is used instead of only the distance to the next gate to avoid jumps in the prediction horizon after a gate pass. In our simulated track experiment, a minimum prediction horizon of $d_{\min} = 1.5$ m was used, whereas for the real track we used $d_{\min} = 1.0$ m.

At test time, since the output of the network is a direction and a velocity, the length of a trajectory segment needs to be computed. To distinguish the length of trajectory segments at test time from the same concept at training time, we call it *planning length* at test time. The planning length of trajectory segments is computed based on the velocity output of the network (computation based on the location of the quadrotor with respect to the gates is not possible at test time since we do not have knowledge about gate positions). The objective is again to adapt the planning length such that both smooth flight at high speed and aggressive maneuvers in tight turns are possible. We achieve this versatility by computing the planning length according to this linear function

$$d_{\text{test}} = \min\left[d_{\max}, \max\left(d_{\min}, m_d v_{\text{out}}\right)\right] \tag{3}$$

where $m_d = 0.6$ s, $d_{\min} = 1.0$ m, and $d_{\max} = 2.0$ m in our real-world experiments, and $m_d = 0.5$ s, $d_{\min} = 2.0$ m, and $d_{\max} = 5.0$ m in the simulated track.

IV. Experiments

We extensively evaluate the presented approach in a wide range of simulated and real scenarios. We first use a controlled, simulated environment to test the main building blocks of our system, i.e., the convolutional architecture and the perception-control modularization. Then, to show the ability of our approach to control real quadrotors, we perform a second set of experiments on a physical platform. We compare our approach to state-of-the-art methods, as well as to human drone pilots of different skill levels. We also demonstrate that our system achieves zero-shot simulation-to-reality transfer. A policy trained on large amounts of cheap simulated data shows increased robustness against external factors, e.g., illumination and visual distractors, compared to a policy trained only with data collected in the real world. Finally, we perform an ablation study to identify the most important factors that enable successful policy transfer from simulation to the real world.

A. Experimental Setup

For all our simulation experiments we use Gazebo as the simulation engine. Although nonphotorealistic, we have selected this engine since it models with high fidelity the physics of a quadrotor via the RotorS extension [47].

Specifically, we simulate the AscTec Hummingbird multirotor, which is equipped with a forward-looking 300×200 pixels RGB camera.

The platform is spawned in a flying space of cubical shape with side length of 70 m, which contains the experiment-specific race track. The flying space is bounded by background and floor planes whose textures are randomized in the simulation experiments of Section IV-E.

The large simulated race track [see Fig. 4(b)] is inspired by a real track used in international competitions. We use this track layout for all of our experiments, except the comparison against end-to-end navigation policies. The track is traveled in the same

(a) (b)

Fig. 4. Illustration of the simulated tracks. The small track (a) consists of four gates and spans a total length of 43 m. The large track (b) consists of eight gates placed at different heights and spans a total length of 116 m.

direction (clockwise or counterclockwise) at training and testing time. We will release all code required to run our simulation experiments upon acceptance of this manuscript.

For real-world experiments, except for the ones evaluating sim-to-real transfer, we collected data in the real world. We used an in-house quadrotor equipped with an Intel UpBoard and a Qualcomm Snapdragon Flight Kit. While the latter is used for VIO, the former represents the main computational unit of the platform. The Intel UpBoard was used to run all the calculations required for flying, from neural network prediction to trajectory generation and tracking.

B. Experiments in Simulation

Using a controlled simulated environment, we perform an extensive evaluation to i) understand the advantages of our approach with respect to end-to-end or classical navigation policies, ii) test the system's robustness to structural changes in the environment, and iii) analyze the effect of the system's hyperparameters on the final performance.

Comparison to end-to-end learning approach: In our first scenario, we use a small track that consists of four gates in a planar configuration with a total length of 43 m [see Fig. 4(a)].

We use this track to compare the performance to a naive deep learning baseline that directly regresses body rates from raw images. Ground truth body rates for the baseline were provided by generating a minimum snap reference trajectory through all gates and then tracking it with a low-level controller [8]. For comparability, this baseline and our method share the same network architecture. Our approach was always able to successfully complete the track. In contrast, the naive baseline could never pass through more than one gate. Training on more data (35 k samples, as compared to 5 k samples used by our method) did not noticeably improve the performance of the baseline. We believe that end-to-end learning of low-level controls [37] is suboptimal for the task of drone navigation when operating in the real world. Since a quadrotor is an unstable platform [48], learning the function that converts images to low-level commands has a very high sample complexity. Additionally, the network is constrained by computation time. In order to guarantee stable control, the baseline network would have to produce control commands at a higher frequency (typically 50 Hz) than the camera images arrive (30 Hz) and process them at a rate that is computationally infeasible with existing on-board hardware. In our experiments, since the low-level controller runs at 50 Hz, a network prediction is repeatedly applied until the next prediction arrives.

In order to allow on-board sensing and computing, we propose a modularization scheme that organizes perception and control into two blocks. With modularization, our approach can benefit from the most advanced learning-based perceptual architectures and from years of study in the field of control theory [49]. Because body rates are generated by a classic controller, the network can focus on the navigation task, which leads to high sample efficiency. Additionally, because the network does not need to ensure the stability of the platform, it can process images at a lower rate than required for the low-level controller, which unlocks on-board computation. Given its inability to complete even this simple track, we do not conduct any further experiments with the direct end-to-end regression baseline.

Performance on a complex track: In order to explore the capabilities of our approach of performing high-speed racing, we conduct a second set of experiments on a larger and more complex track with eight gates and a length of 116 m [see Fig. 4(b)]. The quantitative evaluation is conducted in terms of average task completion rate over five runs initialized with different random seeds. For one run, the task completion rate linearly increases with each passed gate while 100% task completion is achieved if the quadrotor is able to successfully complete five consecutive laps without crashing. As a baseline, we use a pure feedforward setting by following the global trajectory t_g using state estimates provided by VIO [4].

The results of this experiment are shown in Fig. 5(a). We can observe that the VIO baseline, due to accumulated drift, performs worse than our approach. Fig. 5(b) illustrates the influence of drift on the baseline's performance. While performance is comparable when one single lap is considered a success, it degrades rapidly if the threshold for success is raised to more laps. On a static track [see Fig. 5(a)], a SLAM-based state estimator [5], [11] would have less drift than a VIO baseline, but we empirically found the latency of existing open-source SLAM pipelines to be too high for closed-loop control. A benchmark comparison of latencies of monocular visual-inertial SLAM algorithms for flying robots can be found in [50].

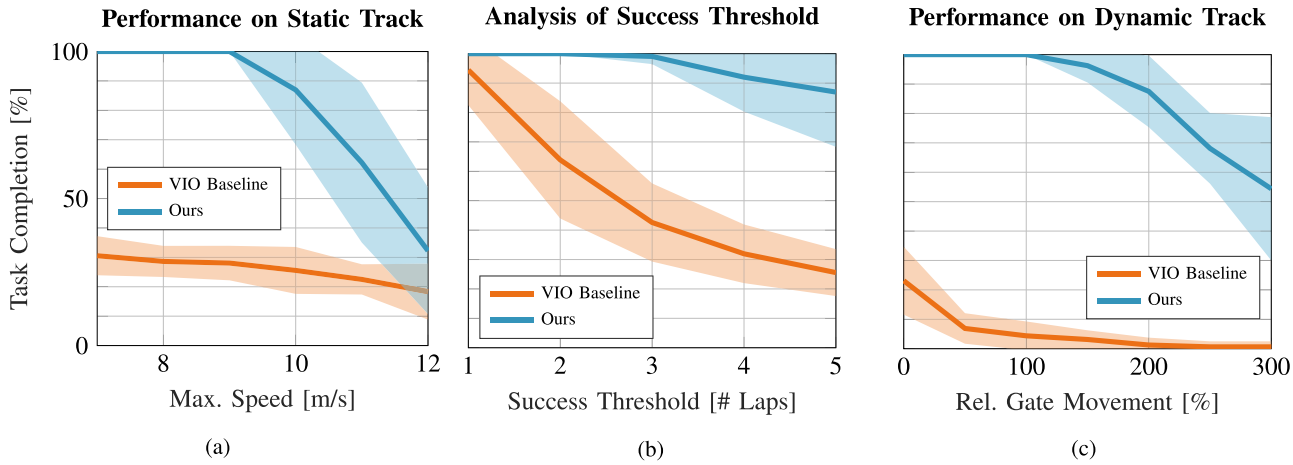

Fig. 5. (a) Results of simulation experiments on the large track with static gates for different maximum speeds. *Task completion rate* measures the fraction of gates that were successfully completed without crashing. A task completion rate of 100% is achieved if the drone can complete five consecutive laps without crashing. For each speed ten runs were performed. (b) Analysis of the influence of the choice of success threshold. The experimental setting is the same as in Fig. 5(a), but the performance is reported for a fixed maximum speed of 10 ms^{-1} and different success thresholds. The y-axis is shared with Fig. 5(a). (c) Result of our approach when flying through a simulated track with moving gates. Every gate independently moves in a sinusoidal pattern with an amplitude proportional to its base size (1.3 m), with the indicated multiplier. For each amplitude ten runs were performed. As for the static gate experiment, a task completion rate of 100% is achieved if the drone can complete five consecutive laps without crashing. Maximum speed is fixed to 8 ms^{-1}. The y-axis is shared with Fig. 5(a). Lines denote mean performance, whereas the shaded areas indicate one standard deviation. The reader is encouraged to watch the supplementary video to better understand the experimental setup and the task difficulty.

Our approach works reliably up to a maximum speed of 9 ms^{-1} and performance degrades gracefully at higher velocities. The decrease in performance at higher speeds is mainly due to the higher body rates of the quadrotor that larger velocities inevitably entail. Since the predictions of the network are in the body frame, the limited prediction frequency (30 Hz in the simulation experiments) is no longer sufficient to cope with the large roll and pitch rates of the platform at high velocities.

Generalization to dynamic environments: The learned policy has a characteristic that the expert policy lacks of: the ability to cope with dynamic environments. To quantitatively test this ability, we reuse the track layout from the previous experiment [see Fig. 4(b)], but dynamically move each gate according to a sinusoidal pattern in each dimension independently. Fig. 5(c) compares our system to the VIO baseline for varying amplitudes of gates' movement relative to their base size. We evaluate the performance using the same metric as explained in Section IV-B. For this experiment, we kept the maximum platform velocity $v_{\max}$ constant at 8 ms^{-1}. Despite the high speed, our approach can handle dynamic gate movements up to 1.5 times the gate diameter without crashing. In contrast, the VIO baseline cannot adapt to changes in the environment, and fails even for small gate motions up to 50% of the gate diameter. The performance of our approach gracefully degrades for gate movements larger than 1.5 times the gate diameter, mainly due to the fact that consecutive gates get too close in flight direction while being shifted in other directions. Such configurations require extremely sharp turns that go beyond the navigation capabilities of the system. From this experiment, we can conclude that the proposed approach reactively adapts to dynamic changes in the environment and generalizes well to cases where the track layout remains roughly similar to the one used to collect training data.

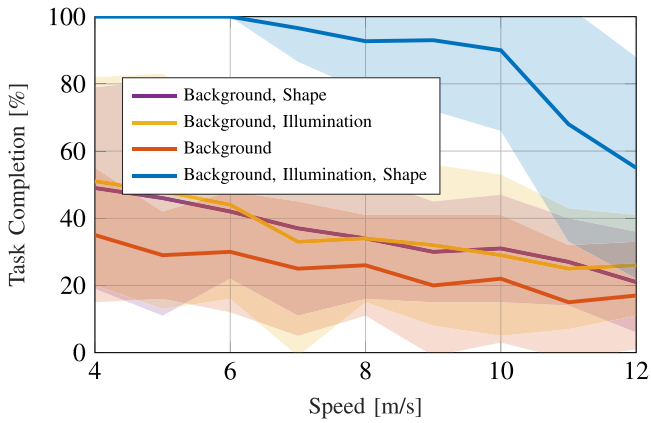

Fig. 6. Generalization tests on different backgrounds after domain randomization. More comprehensive randomization increases the robustness of the learned policy to unseen scenarios at different speeds. Lines denote mean performance, whereas the shaded areas indicate one standard deviation. Background randomization has not been included in the analysis: without it the policy fails to complete even a single gate pass.

Generalization to changes in the simulation environment: In the previous experiments, we have assumed a constant environment (background, illumination, gate shape) during data collection and testing. In this section, we evaluate the generalization abilities of our approach to environment configurations not seen during training. Specifically, we drastically change the environment background [see Fig. 3(b)] and use gate appearance and illumination conditions held out at training time.

Fig. 6 shows the result of this evaluation. As expected, if data collection is performed in a single environment, the resulting policy has limited generalization (red line). To make the policy environment-agnostic, we performed domain randomization

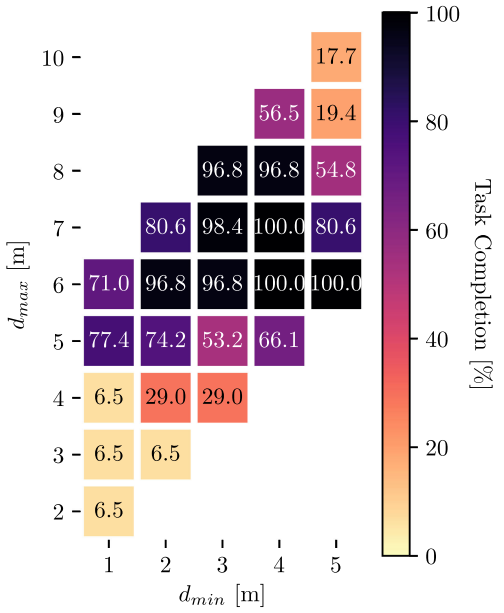

Fig. 7. Sensitivity analysis of planning length parameters $d_{\min}$ and $d_{\max}$ on a simulated track. Maximum speed and (static) track layout are kept constant during the experiment.

while keeping the approximate track layout constant (details in Section III-A). Clearly, both randomization of gate shape and illumination lead to a policy that is more robust to new scenarios. Furthermore, while randomization of a single property leads to a modest improvement, performing all types of randomization simultaneously is crucial for good transfer. Indeed, the simulated policy needs to be invariant to all of the randomized features in order to generalize well.

Surprisingly, as we show in the following, the learned policy can not only function reliably in simulation but is also able to control a quadrotor in the real world. In Section IV-E, we present an evaluation of the real-world control abilities of this policy trained in simulation, as well as an ablation study to identify which of the randomization factors presented above are the most important for generalization and knowledge transfer.

Sensitivity to planning length: We perform an ablation study of the *planning length* parameters $d_{\min}$ and $d_{\max}$ on a simulated track. Both the track layout and the maximum speed ($10.0\ \mathrm{ms}^{-1}$) are kept constant in this experiment. We varied $d_{\min}$ between 1.0 and 5.0 m and $d_{\max}$ between $(d_{\min} + 1.0)$ m and $(d_{\min} + 5.0)$ m. Fig. 7 shows the results of this evaluation. For each configuration, the average *task completion rate* (see Section IV-B) over five runs is reported. Our systems performs well over a large range of $d_{\min}$ and $d_{\max}$, with performance dropping sharply only for configurations with very short or very long planning lengths. This behavior is expected since excessively short planning lengths result in very aggressive maneuvers, whereas excessively long planning lengths restrict the agility of the platform.

C. Analysis of Accuracy and Efficiency

The neural network at the core of our perception system constitutes the biggest computational bottleneck of our approach. Given the constraints imposed by our processing unit, we can guarantee real-time performance only with relatively small CNNs. Therefore, we investigated the relationship between the capacity (hence the representational power) of a neural network and its performance on the navigation task. We measure performance in terms of both prediction accuracy on a validation set, and closed-loop control on a simulated platform, using, as above, completion rate as metric. The capacity of the network is controlled through a multiplicative factor on the number of filters (in convolutional layers) and number of nodes (in fully connected layers). The network with capacity 1.0 corresponds to the DroNet architecture [26].

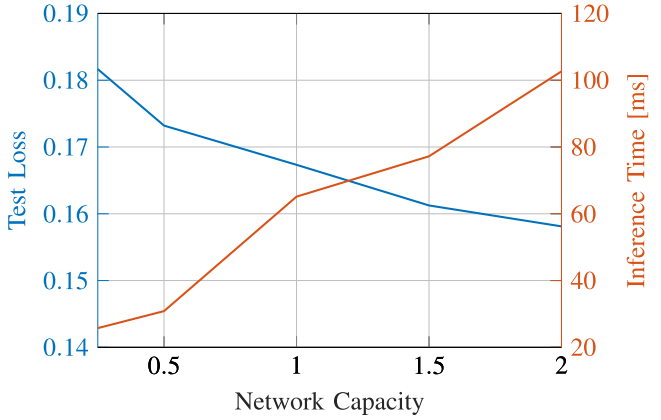

Fig. 8. Test loss and inference time for different network capacity factors. Inference time is measured on the actual platform.

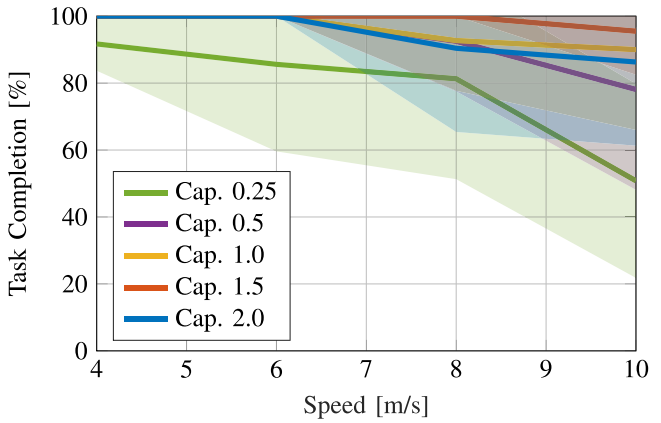

Fig. 9. Comparison of different network capacities on different backgrounds after domain randomization.

Fig. 8 shows the relationship between the network capacity, its test loss [root-mean-square error (RMSE)] on a validation set, and its inference time on an Intel UpBoard (our on-board processing unit). Given their larger parameterization, wider architectures have a lower generalization error but largely increase the computational and memory budget required for their execution. Interestingly, a lower generalization loss does not always correspond to a better closed-loop performance. This can be observed in Fig. 9, where the network with capacity 1.5 outperforms the one with capacity 2.0 at high speeds. Indeed, as shown in Fig. 8, larger networks entail smaller inference rates, which result in a decrease in agility.

Fig. 10. Setup of the narrow gap and occlusion experiments.

TABLE I
SUCCESS RATE FOR FLYING THROUGH A NARROW GAP FROM DIFFERENT INITIAL ANGLES

Relative angle range [°]	Handcrafted detector	Network
[0, 30]	70%	100%
[30, 70]	0%	80%
[70, 90]*	0%	20%

Each row reports the average of ten runs uniformly spanning the range. The gate was completely invisible at initialization in the experiments marked with*.

In our previous conference paper [9], we used a capacity factor of 1.0, which appears to have a good time-accuracy tradeoff. However, in the light of this article, we select a capacity factor of 0.5 for all our new sim-to-real experiments to ease the computational burden. Indeed, the latter experiments are performed at a speed of 2 ms^{-1}, where both 0.5 and 1.0 have equivalent closed-loop control performance (see Fig. 9).

D. Experiments in the Real World

To show the ability of our approach to function in the real world, we performed experiments on a physical quadrotor. We compared our model to state-of-the-art classic approaches to robot navigation, as well as to human drone pilots of different skill levels.

Narrow gate passing: In the initial set of experiments the quadrotor was required to pass through a narrow gate, only slightly larger than the platform itself. These experiments are designed to test the robustness and precision of the proposed approach. An illustration of the setup is shown in Fig. 10. We compare our approach to the handcrafted window detector of Falanga *et al.* [34] by replacing our perception system with the handcrafted detector and leaving the control system unchanged.

Table I summarizes a comparison between our approach and the baseline. We tested the robustness of both approaches to the initial position of the quadrotor by placing the platform at different starting angles with respect to the gate (measured as the angle between the line joining the center of gravity of the quadrotor and the gate, respectively, and the optical axis of the forward facing camera on the platform). We then measured the average success rate at passing the gate without crashing. The experiments indicate that our approach is not sensitive to the initial position of the quadrotor. The drone is able to pass the gate consistently, even if the gate is only partially visible. In contrast, the baseline sometimes fails even if the gate is fully visible because the window detector loses tracking due to platform vibrations. When the gate is not entirely in the field of view, the handcrafted detector fails in all cases.

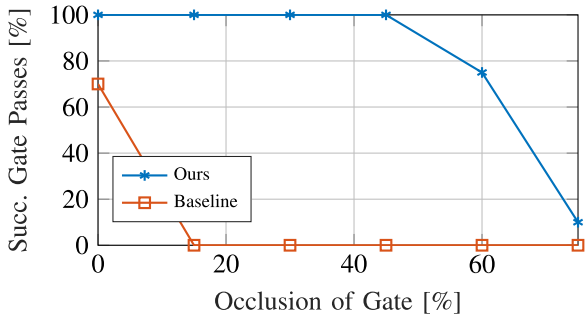

Fig. 11. Success rate for different amounts of occlusion of the gate. Our method is much more robust than the baseline method that makes use of a hand-crafted window detector. Note that at more than 60% occlusion, the platform has barely any space to pass through the gap.

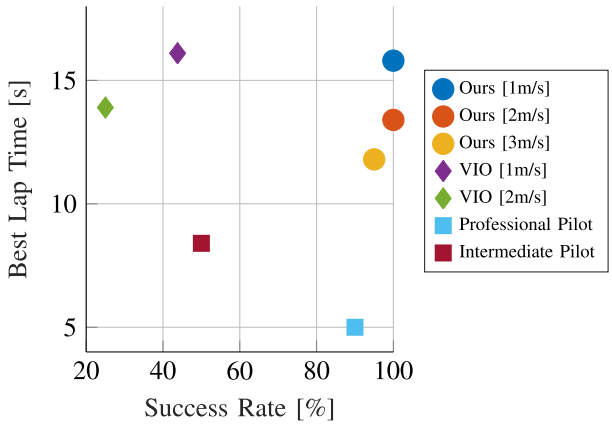

Fig. 12. Results on a real race track composed of four gates. Our learning-based approach compares favorably against a set of baselines based on visual-inertial state estimation. Additionally, we compare against an intermediate and a professional human pilot. We evaluate success rate using the same metric as explained in Section IV-B.

In order to further highlight the robustness and generalization abilities of the approach, we perform experiments with an increasing amount of clutter that occludes the gate. Note that the learning approach has not been trained on such occluded configurations. Fig. 11 shows that our approach is robust to occlusions of up to 50% of the total area of the gate (see Fig. 10), whereas the handcrafted baseline breaks down even for moderate levels of occlusion. For occlusions larger than 50%, we observe a rapid drop in performance. This can be explained by the fact that the remaining gap was barely larger than the drone itself, requiring very high precision to successfully pass it. Furthermore, visual ambiguities of the gate itself become problematic. If just one of the edges of the window is visible, it is impossible to differentiate between the top and bottom part. This results in overcorrection when the drone is very close to the gate.

Experiments on a race track: To evaluate the performance of our approach in a multigate scenario, we challenge the system to race through a track with either static or dynamic gates. The

Fig. 13. Track configuration used for the real-world experiments.

TABLE II
COMPARISON OF OUR APPROACH WITH A PROFESSIONAL HUMAN PILOT ON A STATIC AND A DYNAMIC TRACK

Method	Task completion (average) static	Task completion (average) dynamic	Best lap time [s] static	Best lap time [s] dynamic
Ours	95%	95%	12.1	15.0
Professional pilot	90%	80%	5.0	6.5

We evaluate the performance using the same metric as explained in Section IV-B.

track is shown in Fig. 13. It is composed of four gates and has a total length of 21 m.

To fully understand the potential and limitations of our approach, we compared to a number of baselines, such as a classic approach based on planning and tracking [51] and human pilots of different skill levels. Note that due to the smaller size of the real track compared to the simulated one, the maximum speed achieved in the real-world experiments is lower than in simulation. For our baseline, we use a state-of-the-art VIO approach [51] for state estimation in order to track the global reference trajectory.

Fig. 12 summarizes the quantitative results of our evaluation, where we measure success rate (completing five consecutive laps without crashing corresponds to 100%), as well as the best lap time. Our learning-based approach outperforms the VIO baseline, whose drift at high speeds inevitably leads to poor performance. In contrast, our approach is insensitive to state estimation drift since it generates navigation commands in the body frame. As a result, it completes the track with higher robustness and speed than the VIO baseline.

In order to see how state-of-the-art autonomous approaches compare to human pilots, we asked a professional and an intermediate pilot to race through the track in first-person view. We allowed the pilots to practice the track for ten laps before lap times and failures were measured (see Table II). It is evident from Fig. 12 that both the professional and the intermediate pilots were able to complete the track faster than the autonomous systems. However, the high-speed and aggressive flight by human pilots comes at the cost of increased failure rates. The intermediate pilot in particular had issues with the sharp turns present in the track, leading to frequent crashes. Compared with the autonomous systems, human pilots perform more agile maneuvers, especially in sharp turns. Such maneuvers require a level of reasoning about the environment that our autonomous system still lacks.

Dynamically moving gates: We performed an additional experiment to understand the abilities of our approach to adapt to dynamically changing environments. In order to do so, we manually moved the gates of the race track (see Fig. 13) while the quadrotor was navigating through it. Flying the track under these conditions requires the navigation system to reactively respond to dynamic changes. Note that moving gates break the main assumption of traditional high-speed navigation approaches [52], [53], specifically that the trajectory can be preplanned in a static world. They could thus not be deployed in this scenario. Due to the dynamic nature of this experiment, we encourage the reader to watch the supplementary video.[1] Table II provides a comparison in term of task completion and lap time with respect to a professional pilot. Due to the gates' movement, lap times are larger than the ones recorded in static conditions. However, while our approach achieves the same performance with respect to crashes, the human pilot performs slightly worse, given the difficulties entailed by the unpredictability of the track layout. It is worth noting that training data for our policy were collected by changing the position of only a single gate, but the network was able to cope with movement of any gate at test time.

E. Simulation to Real-World Transfer

We now attempt direct simulation-to-real transfer of the navigation system. To train the policy in simulation, we use the same process to collect simulated data as in Section IV-B, i.e., randomization of illumination conditions, gate appearance, and background. The resulting policy, evaluated in simulation in Fig. 6, is then used without any fine-tuning to fly a real quadrotor. Despite the large appearance differences between the simulated environment [see Fig. 3(d)] and the real one (see Fig. 13), the policy trained in simulation via domain randomization has the ability to control the quadrotor in the real world. Thanks to the abundance of simulated data, this policy can not only be transferred from simulation to the real world but is also more robust to changes in the environment than the policy trained with data collected on the real track. As can be seen in the supplementary video, the policy learned in simulation can not only reliably control the platform but is also robust to drastic differences in illumination and distractors on the track.

To quantitatively benchmark the policy learned in simulation, we compare it against a policy that was trained on real data. We use the same metric as explained in Section IV-B for this evaluation. All experiments are repeated ten times and the results averaged. The results of this evaluation are shown in Fig. 14. The data that were used to train the "real" policy were recorded on the same track for two different illumination conditions: *easy* and *medium*. Illumination conditions are varied by changing the number of enabled light sources: 4 for the easy, 2 for the medium, and 1 for the difficult. The supplementary video illustrates the different illumination conditions.

[1] [Online]. Available: http://youtu.be/8RILnqPxo1s

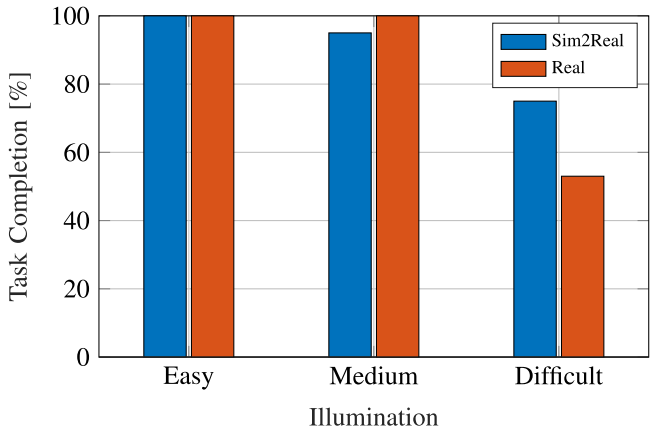

Fig. 14. Performance comparison (measured with task completion rate) of the model trained in simulation and the one trained with real data. With *easy* and *medium* illumination (on which the real model was trained on), the approaches achieve comparable performance. However, with *difficult* illumination the simulated model outperforms the real one since the latter was never exposed to this degree of illumination changes at training time. The supplementary video illustrates the different illumination conditions.

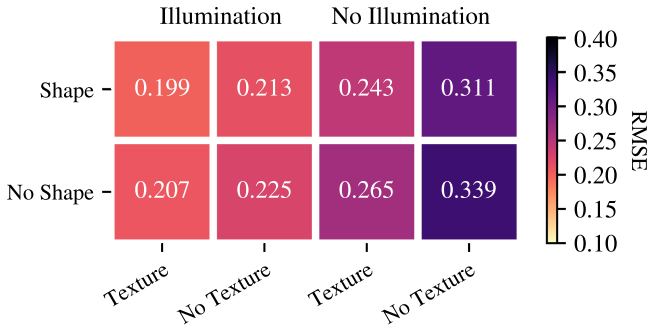

Fig. 15. Average RMSE on testing data collected in the real world (lower is better). Headers indicate what is randomized during data collection.

The policy trained in simulation performs on par with the one trained with real data in experiments that have the same illumination conditions as the training data of the real policy. However, when the environment conditions are drastically different (i.e., with very challenging illumination), the policy trained with real data is outperformed by the one trained in simulation. Indeed, as shown by previous work [41], the abundance of simulated training data makes the resulting learning policy robust to environmental changes. We invite the reader to watch the supplementary video to understand the difficulty of this last set of experiments.

What is important for transfer? We conducted a set of ablation studies to understand what are the most important factors for transfer from simulation to the real world. In order to do so, we collected a dataset of real-world images from both indoor and outdoor environments in different illumination conditions, which we then annotated using the same procedure as explained in Section III. More specifically, the dataset is composed of approximately 10 k images and is collected from three indoor environments under different illumination conditions. Sample images of this dataset are shown in the Appendix.

During data collection in simulation, we perform randomization of background, illumination conditions, and gate appearance (shape and texture). In this experiment, we study the effect of each of the randomized factors, except for the background that is well known to be fundamental for transfer [10], [25], [41]. We use as metric the RMSE in prediction on our collected dataset. As shown in Fig. 15, illumination is the most important of the randomization factors, whereas gate shape randomization has the smallest effect. Indeed, while gate appearance is similar in the real world and in simulation, the environment appearance and illumination are drastically different. However, including more randomization is always beneficial for the robustness of the resulting policy (see Fig. 6).

V. Conclusion

In this article, we presented a new approach to autonomous, vision-based drone racing. Our method used a compact CNN to continuously predict a desired waypoint and speed directly from raw images. These high-level navigation directions were then executed by a classic planning and control pipeline. As a result, the system combined the robust perceptual awareness of modern machine learning pipelines with the precision and speed of well-known control algorithms.

We investigated the capabilities of this integrated approach over three axes: precision, speed, and generalization. Our extensive experiments, performed both in simulation and on a physical platform, show that our system is able to navigate complex race tracks, avoids the problem of drift that is inherent in systems relying on global state estimates, and can cope with highly dynamic and cluttered environments.

Our previous conference work [9] required collecting a substantial amount of training data from the track of interest. Here, instead we proposed to collect diverse simulated data via domain randomization to train our perception policy. The resulting system can not only adapt to drastic appearance changes in simulation but can also be deployed to a physical platform in the real world even if only trained in simulation. Thanks to the abundance of simulated data, a perception system trained in simulation can achieve higher robustness to changes in environment characteristics (e.g., illumination conditions) than a system trained with real data.

It is interesting to compare the two training strategies—on real data and sim-to-real—in how they handle ambiguous situations in navigation, for instance when no gate is visible or multiple gates are in the field of view. Our previous work [9], which was trained on the test track, could disambiguate those cases by using cues in the environment, for instance discriminative landmarks in the background. This can be seen as implicitly memorizing a map of the track in the network weights. In contrast, when trained only in simulation on multiple tracks (or randomized versions of the same track), our approach can no longer use such background cues to disambiguate the flying direction and has instead to rely on a high-level map prior. This prior, automatically inferred from the training data, describes some common characteristics of the training tracks, such as, for instance, to always turn right when no gate is visible. Clearly, when ambiguous cases cannot be

resolved with a prior of this type (e.g., an eight-shaped track), our sim-to-real approach would likely fail. Possible solutions to this problem are fine-tuning with data coming from the real track, or the use of a metric prior on the track shape to make decisions in ambiguous conditions [54].

Due to modularity, our system can combine model-based control with learning-based perception. However, one of the main disadvantages of modularity is that errors coming from each submodule degrade the full system performance in a cumulative way. To overcome this problem, we plan to improve each component with experience using a reinforcement learning approach. This could increase the robustness of the system and improve its performance in challenging scenarios (e.g., with moving obstacles).

While our current set of experiments was conducted in the context of drone racing, we believe that the presented approach could have broader implications for building robust robot navigation systems that need to be able to act in a highly dynamic world. Methods based on geometric mapping, localization, and planning have inherent limitations in this setting. Hybrid systems that incorporate machine learning, like the one presented in this article, can offer a compelling solution to this task, given the possibility to benefit from near-optimal solutions to different subproblems. However, scaling our proposed approach to more general applications, such as disaster response or industrial inspection, poses several challenges. First, due to the unknown characteristics of the path to be flown (layout, presence and type of landmarks, obstacles), the generation of a valid teacher policy would be impossible. This could be addressed with techniques such as *few-shot learning*. Second, the target applications might require extremely high agility, for instance in the presence of sharp turns, which our autonomous system still lacks of. This issue could be alleviated by integrating learning deeper into the control system [22].

REFERENCES

[1] G.-Z. Yang *et al.*, "The grand challenges of science robotics," *Sci. Robot.*, vol. 3, no. 14, 2018, Art. no. eaar7650.

[2] H. Moon, Y. Sun, J. Baltes, and S. J. Kim, "The IROS 2016 competitions," *IEEE Robot. Autom. Mag.*, vol. 24, no. 1, pp. 20–29, Mar. 2017.

[3] H. Moon *et al.*, "Challenges and implemented technologies used in autonomous drone racing," *Intell. Service Robot.*, vol. 1, no. 1, pp. 611–625, 2019.

[4] C. Forster, M. Pizzoli, and D. Scaramuzza, "SVO: Semi-direct visual odometry for monocular and multi-camera systems," *IEEE Trans. Robot.*, vol. 33, no. 2, pp. 249–265, Apr. 2017.

[5] T. Qin, P. Li, and S. Shen, "VINS-Mono: A robust and versatile monocular visual-inertial state estimator," *IEEE Trans. Robot.*, vol. 34, no. 4, pp. 1004–1020, Aug. 2018.

[6] C. Cadena *et al.*, "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age," *IEEE Trans. Robot.*, vol. 32, no. 6, pp. 1309–1332, Dec. 2016.

[7] M. W. Mueller, M. Hehn, and R. D'Andrea, "A computationally efficient algorithm for state-to-state quadrocopter trajectory generation and feasibility verification," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2013, pp. 3480–3486.

[8] M. Faessler, A. Franchi, and D. Scaramuzza, "Differential flatness of quadrotor dynamics subject to rotor drag for accurate tracking of high-speed trajectories," *IEEE Robot. Autom. Lett.*, vol. 3, no. 2, pp. 620–626, Apr. 2018.

[9] E. Kaufmann, A. Loquercio, R. Ranftl, A. Dosovitskiy, V. Koltun, and D. Scaramuzza, "Deep drone racing: Learning agile flight in dynamic environments," in *Proc. Conf. Robot Learn.*, 2018, pp. 133–145.

[10] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2017, pp. 23–30.

[11] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "ORB-SLAM: A versatile and accurate monocular SLAM system," *IEEE Trans. Robot.*, vol. 31, no. 5, pp. 1147–1163, Oct. 2015.

[12] S. Lynen, T. Sattler, M. Bosse, J. Hesch, M. Pollefeys, and R. Siegwart, "Get out of my lab: Large-scale, real-time visual-inertial localization," in *Proc. Robot., Sci. Syst.*, 2015, doi:10.15607/RSS.2015.XI.037.

[13] A. Bry, A. Bachrach, and N. Roy, "State estimation for aggressive flight in GPS-denied environments using onboard sensing," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2012, pp. 1–8.

[14] A. Rosinol Vidal, H. Rebecq, T. Horstschaefer, and D. Scaramuzza, "Ultimate SLAM? Combining events, images, and IMU for robust visual SLAM in HDR and high speed scenarios," *IEEE Robot. and Autom. Lett.*, vol. 3, no. 2, pp. 994–1001, Apr. 2018.

[15] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2011, pp. 2520–2525.

[16] B. Morrell *et al.*, "Differential flatness transformations for aggressive quadrotor flight," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2018, pp. 1–7.

[17] K. Kritayakirana and J. C. Gerdes, "Autonomous vehicle control at the limits of handling," *Int. J. Vehicle Auton. Syst.*, vol. 10, no. 4, pp. 271–296, 2012.

[18] N. R. Kapania, "Trajectory planning and control for an autonomous race vehicle," Ph.D. dissertation, Dept. Mech. Eng., Stanford Univ., Stanford, CA, USA, 2016.

[19] J. C. Kegelman, L. K. Harbott, and J. C. Gerdes, "Insights into vehicle trajectories at the handling limits: Analysing open data from race car drivers," *Vehicle Syst. Dyn.*, vol. 55, no. 2, pp. 191–207, 2017.

[20] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou, "Aggressive driving with model predictive path integral control," in *Proc. IEEE Int. Conf. Robot. Autom.*, Stockholm, Sweden, May 2016, pp. 1433–1440.

[21] P. Drews, G. Williams, B. Goldfain, E. A. Theodorou, and J. M. Rehg, "Aggressive deep driving: Combining convolutional neural networks and model predictive control," in *Proc. Conf. Robot Learn.*, 2017, pp. 133–142.

[22] Y. Pan *et al.*, "Agile autonomous driving using end-to-end deep imitation learning," in *Proc. Robot., Sci. Syst.*, 2018, doi: 10.15607/RSS.2018.XIV.056.

[23] C. Richter and N. Roy, "Safe visual navigation via deep learning and novelty detection," in *Proc. Robot., Sci. Syst.*, 2017, doi: 10.15607/RSS.2017.XIII.064.

[24] G. Kahn, A. Villaflor, B. Ding, P. Abbeel, and S. Levine, "Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2018, pp. 5129–5136.

[25] F. Sadeghi and S. Levine, "CAD2RL: Real single-image flight without a single real image," in *Proc. Robot., Sci. Syst.*, 2017, doi: 10.15607/RSS.2017.XIII.034.

[26] A. Loquercio, A. I. Maqueda, C. R. D. Blanco, and D. Scaramuzza, "DroNet: Learning to fly by driving," *IEEE Robot. Autom. Lett.*, vol. 3, no. 2, pp. 1088–1095, Apr. 2018.

[27] D. Gandhi, L. Pinto, and A. Gupta, "Learning to fly by crashing," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2017, pp. 3948–3955.

[28] R. Hadsell *et al.*, "Learning long-range vision for autonomous off-road driving," *J. Field Robot.*, vol. 26, no. 2, pp. 120–144, 2009.

[29] C. Devin, A. Gupta, T. Darrell, P. Abbeel, and S. Levine, "Learning modular neural network policies for multi-task and multi-robot transfer," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2017, pp. 2169–2176.

[30] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "Deepdriving: Learning affordance for direct perception in autonomous driving," in *Proc. Int. Conf. Comput. Vision*, 2015, pp. 2722–2730.

[31] I. Clavera, D. Held, and P. Abbeel, "Policy transfer via modularity and reward guiding," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2017, pp. 1537–1544.

[32] M. Müller, A. Dosovitskiy, B. Ghanem, and V. Koltun, "Driving policy transfer via modularity and abstraction," in *Proc. Conf. Robot Learn.*, 2018, pp. 1–15.

[33] O. Tahri and F. Chaumette, "Point-based and region-based image moments for visual servoing of planar objects," *IEEE Trans. Robot.*, vol. 21, no. 6, pp. 1116–1127, Dec. 2005.

[34] D. Falanga, E. Mueggler, M. Faessler, and D. Scaramuzza, "Aggressive quadrotor flight through narrow gaps with onboard sensing and computing using active vision," in *Proc. IEEE Int. Conf. Robot. and Autom.*, 2017, pp. 5774–5781.

[35] S. Li, M. Ozo, C. De Wagter, and G. de Croon, "Autonomous drone race: A computationally efficient vision-based navigation and control strategy," 2018, *arXiv:1809.05958*.
[36] S. Jung, S. Hwang, H. Shin, and D. H. Shim, "Perception, guidance, and navigation for indoor autonomous drone racing using deep learning," *IEEE Robot. Autom. Lett.*, vol. 3, no. 3, pp. 2539–2544, Jul. 2018.
[37] M. Müller, V. Casser, N. Smith, D. L. Michels, and B. Ghanem, "Teaching UAVs to race: End-to-end regression of agile controls in simulation," in *Proc. Eur. Conf. Comput. Vision Workshops*, 2018, pp. 1–18.
[38] A. Gupta, C. Devin, Y. Liu, P. Abbeel, and S. Levine, "Learning invariant feature spaces to transfer skills with reinforcement learning," in *Proc. Int. Conf. Learn. Representat.*, 2017.
[39] M. Wulfmeier, I. Posner, and P. Abbeel, "Mutual alignment transfer learning," in *Proc. Conf. Robot Learn.*, 2017, pp. 281–290.
[40] K. Bousmalis *et al.*, "Using simulation and domain adaptation to improve efficiency of deep robotic grasping," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2018, pp. 4243–4250.
[41] S. James, A. J. Davison, and E. Johns, "Transferring end-to-end visuomotor control from simulation to real world for a multi-stage task," in *Proc. Conf. Robot Learn.*, 2017, pp. 334–343.
[42] A. A. Rusu, M. Vecerik, T. Rothörl, N. Heess, R. Pascanu, and R. Hadsell, "Sim-to-real robot learning from pixels with progressive nets," in *Proc. Conf. Robot Learn.*, 2017, pp. 262–270.
[43] F. Sadeghi, A. Toshev, E. Jang, and S. Levine, "Sim2Real viewpoint invariant visual servoing by recurrent control," in *Proc. Conf. Comput. Vision Pattern Recognit.*, 2018, pp. 4691–4699.
[44] S. R. Richter, V. Vineet, S. Roth, and V. Koltun, "Playing for data: Ground truth from computer games," in *Proc. Eur. Conf. Comput. Vision*, 2016, pp. 102–118.
[45] M. Johnson-Roberson, C. Barto, R. Mehta, S. N. Sridhar, K. Rosaen, and R. Vasudevan, "Driving in the matrix: Can virtual worlds replace human-generated annotations for real world tasks?" in *Proc. IEEE Int. Conf. Robot. Autom.*, 2017, pp. 746–753.
[46] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proc. Int. Conf. Artif. Intell. and Statist.*, 2011, pp. 746–753.
[47] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart, "RotorS—A modular Gazebo MAV simulator framework," in *Robot Operating System*. Berlin, Germany: Springer, 2016, pp. 595–625.
[48] K. Narendra and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," *IEEE Trans. Neural Netw.*, vol. 1, no. 1, pp. 4–27, Mar. 1990.
[49] R. Mahony, V. Kumar, and P. Corke, "Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor," *IEEE Robot. Autom. Mag.*, vol. 19, no. 3, pp. 20–32, Sep. 2012.
[50] J. Delmerico and D. Scaramuzza, "A benchmark comparison of monocular visual-inertial odometry algorithms for flying robots," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2018, pp. 2502–2509.
[51] 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.*, vol. 2, no. 2, pp. 404–411, Apr. 2017.
[52] A. Bry and N. Roy, "Rapidly-exploring random belief trees for motion planning under uncertainty," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2011, pp. 723–730.
[53] P. Furgale and T. D. Barfoot, "Visual teach and repeat for long-range rover autonomy," *J. Field Robot.*, vol. 27, no. 5, pp. 534–560, 2010.
[54] E. Kaufmann *et al.*, "Beauty and the beast: Optimal methods meet learning for drone racing," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2019, pp. 690–696.
[55] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-CAM: Visual explanations from deep networks via gradient-based localization," in *Proc. Int. Conf. Comput. Vision*, 2017, pp. 618–626.

Antonio Loquercio received the M.Sc. degree in robotics, systems and control from ETH Zürich, Zürich, Switzerland, in 2017. He is currently working toward the Ph.D. degree in computer science with the Robotics and Perception Group, University of Zürich, Zürich, Switzerland, under the supervision of Prof. D. Scaramuzza.

His current research focuses on data-driven methods for perception and control in robotics.

Mr. Loquercio was the recipient of the ETH Medal for outstanding master thesis in 2017.

Elia Kaufmann was born in 1992 in Switzerland. He received the B.Sc. degree in mechanical engineering in 2014, and the M.Sc. degree in robotics, systems and control from ETH Zürich, Zürich, Switzerland, in 2017. Since 2017, he has been working toward the Ph.D. degree in computer science with the University of Zürich, Zürich, under the supervision of D. Scaramuzza.

His current research focuses on the application of machine learning to improve perception and control of autonomous mobile robots.

René Ranftl received the M.Sc. and Ph.D. degrees in computer science from the Graz University of Technology, Graz, Austria, in 2010 and 2015, respectively.

He is currently a Senior Research Scientist with the Intelligent Systems Laboratory, Intel, Munich, Germany. His research interests include computer vision, machine learning, and robotics.

Alexey Dosovitskiy received the M.Sc. and Ph.D. degrees in mathematics (functional analysis) from Moscow State University, Moscow, Russia, in 2009 and 2012, respectively.

He is currently a Research Scientist with the Intelligent Systems Laboratory, Intel, Munich, Germany. From 2013 to 2016, he was a Postdoctoral Researcher, with Prof. T. Brox, with the Computer Vision Group, University of Freiburg, Breisgau, Germany, working on various topics in deep learning, including self-supervised learning, image generation with neural networks, motion, and 3-D structure estimation. In 2017, he joined Intel Visual Computing Laboratory led by Dr. V. Koltun, where he worked on applications of deep learning to sensorimotor control, including autonomous driving and robotics.

Vladlen Koltun is currently a Senior Principal Researcher and the Director of the Intelligent Systems Laboratory, Intel. He has mentored more than 50 Ph.D. students, postdocs, research scientists, and Ph.D. student interns, many of whom are now successful research leaders. His lab conducts high-impact basic research on intelligent systems, with emphasis on computer vision, robotics, and machine learning.

Davide Scaramuzza was born in 1980 in Italy. He received the Ph.D. degree in robotics and computer vision from ETH Zürich, Zürich, Switzerland, in 2008.

He was a Postdoc with the University of Pennsylvania, Philadelphia, PA, USA, in 2011. He is a Professor of Robotics with the University of Zürich, Zürich, Switzerland, where he does research at the intersection of robotics, computer vision, and neuroscience. From 2009 to 2012, he led the European project sFly, which introduced the world's first autonomous navigation of microdrones in GPS-denied environments using visual-inertial sensors as the only sensor modality. He coauthored the book *Introduction to Autonomous Mobile Robots* (MIT Press, 2004).

Dr. Scaramuzza was the recipient of an SNSF-ERC Starting Grant, the IEEE Robotics and Automation Early Career Award, and a Google Research Award for his research contributions.