arXiv:1803.01129v1 [cs.CV] 3 Mar 2018

Teaching UAVs to Race With Observational Imitation Learning

Guohao Li*, Matthias Mueller*, Vincent Casser, Neil Smith, Dominik L. Michels, Bernard Ghanem
Visual Computing Center, KAUST, Thuwal, Saudi Arabia

Abstract

Recent work has tackled the problem of autonomous navigation by imitating a teacher and learning an end-to-end policy, which directly predicts controls from raw images. However, these approaches tend to be sensitive to mistakes by the teacher and do not scale well to other environments or vehicles. To this end, we propose a modular network architecture that decouples perception from control, and is trained using Observational Imitation Learning (OIL), a novel imitation learning variant that supports online training and automatic selection of optimal behavior from observing multiple teachers. We apply our proposed methodology to the challenging problem of unmanned aerial vehicle (UAV) racing. We develop a simulator that enables the generation of large amounts of synthetic training data (both UAV captured images and its controls) and also allows for online learning and evaluation. We train a perception network to predict waypoints from raw image data and a control network to predict UAV controls from these waypoints using OIL. Our modular network is able to autonomously fly a UAV through challenging race tracks at high speeds. Extensive experiments demonstrate that our trained network outperforms its teachers, end-to-end baselines, and even human pilots in simulation.

1. Introduction

UAV racing is an emerging global sport, where highly skilled human pilots compete to be the fastest and most agile pilots. While UAV design and point-to-point stabilized flight navigation is becoming a solved problem, the complexity of racing remains a human only task. Autonomous navigation for both cars and UAVs at racing speeds is still a challenge, since it requires *both* the sensing of the environment and the execution of appropriate policies for interaction at very fast update rates. In UAV racing, these complex sense-and-understand tasks are conducted at neck-break speeds reaching over 100 km/h. Learning to control racing UAVs is a challenging task even for humans. Inspired by recent work on self-driving cars [4], we demonstrate that the development of a broadly applicable new category of imitation learning, dubbed *observational imitation learning* (OIL), enables learning more complex tasks. We apply this new method to the problem of automated UAV racing, which requires control over six degrees of freedom (6-DoF) at high speeds for traversing tight spaces, and achieving perfect racing gate scores. Our approach is a self-supervised modular neural network that teaches itself how to race UAVs in a physics-based simulated environment.

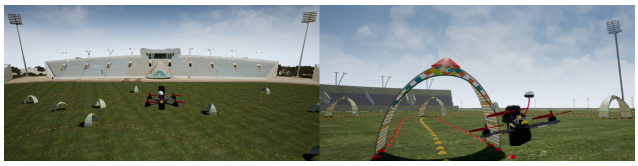

Figure 1: Aerial view of simulated stadium (*left*) and OIL-trained UAV racing through gates on a test track (*right*).

UAV racing can be regarded as a sequential prediction problem. Human pilots or agents are required to sequentially control the UAV to fly through a race track based on the feedback (visual information, physical measurements, or both) of previous actions. In the machine learning community, reinforcement learning (RL) or supervised learning (SL), more specifically imitation learning (IL), can be an avenue to solve such sequential prediction problems. In conventional IL, the learner is required to trust and replicate authoritative behaviors of a teacher. The drawbacks are primarily the need for extensive training data and the inherent subjectivity to potential negative behaviours of teachers. RL does not specifically require supervision by a pilot, as it searches for an optimal policy that leads to the highest eventual reward (*e.g.* most gates traversed or lowest lap time). However, a good reward function, which enables agent learning of desirable behaviors, requires tedious and meticulous reward shaping [31]. It is quite difficult to design a feasible reward function in this challenging high speed racing setting. Recent methods have made use of reinforcement to learn simpler tasks without supervision [7]; however, they require weeks of training and a very fast simulator (1,000fps is possible in simple non photo-realistic games). For UAV racing, the task is much more complicated and since the intent is to eventually transfer the learned network into a real-world setting, a photo-realistic (slower) simulator is necessary.

*equal contribution

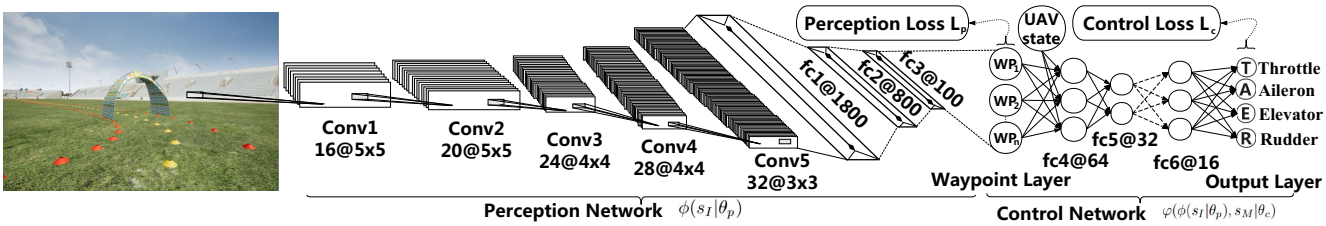

Figure 2: Pipeline of our modular network for autonomous UAV racing. The Perception Network ϕ takes the raw image as input and predicts waypoints. The Control Network φ takes predicted waypoints and UAV state as input and outputs a 4D control signal: T (Throttle), A (Aileron), E (Elevator), and R (Rudder).

In this paper, we propose OIL as a solution that incorporates reinforcement learning concepts to imitate only the successful actions of multiple teachers. Unlike conventional imitating learning, OIL enables learning from multiple teachers and also provides a method for disregarding bad maneuvers by using a reward-like online evaluation of each teacher at training time. Further, we propose a modular architecture enabling the decoupling of observation from control, separating them into a perception and control module, which can be easily substituted depending on the environment and vehicle dynamics. Both perception and control networks can be trained in a fully self-supervised fashion requiring no need for human annotation. We demonstrate that our approach outperforms other state-of-the art end-to-end network architectures and human pilots.

In order to validate OIL for UAV racing, we follow recent work [7] that trains AI systems through the use of computer games. We create a UAV racing game with accurate physics using the Unreal Engine 4 (UE4) and integrate it with UE4Sim [29] (see Figure 1). We also develop a customizable racing area in the form of a stadium based on a 3D scan of a real-world location. Race tracks are generated within a GUI interface and can be loaded at game-time to allow training on multiple tracks automatically. The simulator is multi-purpose, as it enables the generation of synthetic image data, reinforcement based training in real-time, and evaluation on unseen dynamic racing tracks. For more details on the simulator, refer to the **supplementary material**.

Contributions. **(1)** We propose observational imitation learning (OIL) as a new approach for training a stationary deterministic policy that overcomes shortcomings of conventional imitation learning by incorporating reinforcement learning ideas. It learns from an ensemble of teachers but only updates the policy with the best maneuvers of each teacher. Learning is done online, while a buffer/memory also allows for semi-offline training. **(2)** We propose a flexible network architecture that affords modularity in perception and control, applied to UAV racing. It is trained using OIL in a self-supervised manner without any human racing demonstrations. **(3)** To the best of our knowledge, this paper is the first to fully demonstrate the capability of deep networks in learning how to master the complex control of UAVs at racing speeds through difficult flight scenarios. Experiments show that our trained network can reach near-expert performance, while outperforming several end-to-end baselines and less experienced human pilots.

2. Related Work

The task of training an actor (*e.g.* vehicle, human, or UAV) to physically navigate through an unknown environment has traditionally been approached either through supervised learning (SL), in particular Imitation Learning (IL) [4, 6, 8, 20, 30, 35, 38, 44, 47], reinforcement learning (RL) [1, 25–27, 33, 34, 39, 45], or a combination of the two [2, 7, 10, 11, 23, 24]. A key challenge is learning a high dimensional representation of raw sensory input within a complex 3D environment. Similar to our approach in this paper, many recent works [9, 36, 37] use modern game engines or driving simulators, where the 3D environment can be controlled, synthetic datasets for self-driving can be generated, and other navigation based tasks can be evaluated (see [3, 11–13, 15, 17, 22, 25, 28, 32, 45]).

For the particular case of physics based navigation, SL can be advantageous when high dimensional feedback can be recorded. It has proven to be relatively successful in the last several years in the field of autonomous driving [4, 6, 44, 47] and in initial work on UAVs [44]. For self-driving, the vehicle controls are low in complexity: turn-left, turn-right, throttle, and brake. Although these approaches use data augmentation to allow for robustness, it is only corrective. It cannot predict the controls a human driver may choose and requires significant fine-tuning, which may not be applicable at different speeds or environments. As with other SL/IL approaches, the flaws and mistakes of the teachers are learned along with differing responses. We compare the end-to-end approaches of Nvidia [4] and MAV [44] to our own OIL-based method in Section 5.

RL provides an alternative to SL by using rewards and many iterations of exploration to help discover the proper response. The disadvantage is that RL networks may not be able to discover the optimal outputs in higher-orders of

control. TORCS and 2D driving simulators are used in several RL approaches for autonomous car driving [16, 21, 25, 26, 40, 41]. Recently, Dosovitskiy *et al*. [8] released a UE4 simulator to evaluate SL and self-supervised RL approaches for self-driving. They find RL to under-perform due to the extensive hyperparameter search required for vehicular navigation. One approach to reduce the large state space is to apply inverse RL with a deep Q-network (DQN) as a refinement step [43]. Another approach is to use a DQN with an accurate teacher to start. In [40, 41], a p-controller is provided as input to the DQN and over many iterations gradually adapts until the model has fully learned to steer on its own. Finally, DAgger (Dataset Aggregation) [38] presents a unique approach of SL/IL where sequential decisions are incrementally augmented allowing for the type of extensive exploration usually seen in RL. This method outperforms state-of-the-art AI methods on a 3D car racing game (Super Tux Kart), where control outputs are again limited to 3DoF.

The use of deep neural networks (DNNs) for UAVs dates back to work that focuses on learning acrobatic helicopter flight [1]. More recent work has studied both SL and RL approaches with a focus on indoor flight, collision avoidance, and trajectory optimization [2, 18, 19, 24, 39, 42, 44]. An important insight from [18, 23, 24, 39] is that a trajectory optimizer such as a Model Predictive Controller (MPC) can function similar to traditional SL to help regress a learner's sub-optimal policy towards the optimal one with much fewer iterations. By jointly learning perception and control with the self-supervision of MPC, full end-to-end navigation and collision avoidance can be learned.

Our work exploits a similar concept by using PID controllers as teachers to allow for both self-supervision and extensive exploration. Our approach differs in that our control network does not learn strictly from the teacher nor is its training limited by the exploration of the teacher. Rather, similar to Borsa *et al*. [5] we apply observational learning that allows the network to observe teacher behavior and not necessarily duplicate it, a key difference with SL. Our network observes the actions of the PID controllers at each time step and selects the best predictions based on the perceived rewards. It also differs from [23] in that the teachers never have to deviate from their optimal control to induce exploration. By design, our RL motivated training process enables extensive exploration by only observing the best behavior from various teachers. Also, unlike trajectory optimization [14], the trajectory in our setup is a spline of local 2D waypoints with unknown global 3D position. This enables the prediction of local waypoints without needing precise knowledge of the UAV's current state and dynamics, which are rarely available in the real-world. Similar to adaptive trajectory optimization, our predicted waypoints are updated every time step allowing for adaptation to environment changes.

3. Methodology

In this section, we introduce Observational Imitation Learning (OIL) which enables automatic removal of bad demonstrations from imperfect or inexperienced teachers by observing interactions between multiple teachers and the environment before the learner's policy is updated. We then present a modular network architecture for autonomous UAV racing that separates the navigation task into a high dimensional perception module (trained with self-supervsion) and a low dimensional control module (trained with OIL).

3.1. Observational Imitation Learning (OIL)

We define *Observational Imitation Learning* as a learning strategy that integrates knowledge from multiple teachers and the dynamic environment into the learning process (refer to Algorithm 1 for a detailed description). At each time step t, the agent (learner or the teacher) receives a state (or partial state) s_t from the environment and executes an action a_t. Thus, the trajectory of the agent behaviors are denoted as $\tau = (s_1, a_1, s_2, a_2, ..., s_n)$. The learner policy is a parameterized function $\pi(s|\theta)$ mapping the state to a deterministic action that can be continuous or discrete. In our case, the action is a 4D continuous control signal for a UAV. The teacher policy $\mu(s)$ also maps the state to an action. In practice, the teacher can be either an automated controller or a human, and teaching can be performed online or offline from a recording. In this paper, we use two PID (Proportional-Integral-Derivative) controllers as teachers, thus, avoiding the need for hours of human recorded control. Note that the states that the teachers and learner observe can be different, if the learner can physically reproduce the teacher's action by viewing different states.

Unlike conventional imitation learning, OIL does not assume that the teachers are infallible. Although teachers might not be perfect, it is still useful to learn from them. OIL is not simply a process of behavior cloning like imitation learning, since it does not require the learner to duplicate the teacher's behaviors without any self-awareness. The reason why the consciousness of the learner is non-trivial during the learning process is that the demonstrations of the task from the teacher might sometimes be unsuccessful. If the learner is aware of negative behavior like this, it can prevent itself from learning from those behaviors. OIL can overcome the drawbacks of conventional imitation learning, since it empowers the learner to learn the best controls of each teacher. To achieve this goal, a straightforward approach is to observe the consequent feedback generated from interactions between the teacher and the environment before learning from teacher behavior.

To clarify which demonstrations we should learn from, we call the demonstrations that lead the learner to good behavior as *desirable* demonstrations, the ones that lead to bad behavior as *unwanted* demonstrations, and the uncertain ones as *unforeseeable* demonstrations. The goal is to in-

Algorithm 1: Observational Imitation Learning (OIL)

Initialize Teachers $\{\mu(s)\}_{i=1}^{n}$;
Initialize Learner $\pi(s_t|\theta)$ with random weights θ;
Initialize Learner training database $D \leftarrow \emptyset$;
Initialize Learner temporary observational buffers $\{B_i \leftarrow \emptyset\}_{i=1}^{n}$ corresponding to the Teachers;
// for each teacher
for *episode* $\leftarrow 1$ **to** M **do**
  Receive initial state s_1 from the environment;
  for $t \leftarrow 1$ **to** T **do**
    Predict Learner action $a_t = \pi(s_t; \theta)$;
    Teachers demonstrate $\{a_{\mu_i t}\}_{i=1}^{n}$=$\{\mu_i(s_t)\}_{i=1}^{n}$;
    // Or execute learner action with or without exploration noise;
    Execute teacher action $a_{\mu_i t}$; observe feedback
    Update $B_i \leftarrow B_i \bigcup\{(s_t, a_{\mu_i t})\}\ \forall i$
    Discard unwanted demonstrations from $\{B_i\}_{i=1}^{n}$ (according to *buffering strategy*);
    Add desirable demonstrations from $\{B_i\}_{i=1}^{n}$ to D;
    Sample mini-batch (s, a) from D and perform SGD to minimize $\mathbb{L}(\pi(s|\theta), a)$ with respect to θ;
    Break if s_t is terminal state;
  end
end

troduce a virtual consciousness represented by a *Temporary Observational Buffer* B to store the *unforeseeable* demonstrations before collecting them into training data. At each time step t, by observing the feedback of the interactions between teacher and environment, the learner can determine whether to discard some *unwanted* demonstrations from B, retain *unforeseeable* demonstrations in B, or augment its training data D by adding *desirable* demonstrations from B. We call this operation a *Buffering Strategy*. Note that the learner maintains a temporary observational buffer for each teacher.

The training data D can be viewed as a distilled set of demonstrations collected by applying the buffering strategy on B at each time step. Our goal is to find a policy π^* to minimize the loss function $\mathbb{L}(\pi(s|\theta), a)$:

$$\pi^*(s|\theta) = \arg\min_{\pi}\ \mathbb{E}_{(s,a)\sim D}[\mathbb{L}(\pi(s|\theta), a)] \tag{1}$$

3.2. Modular Architecture for UAV Racing

The fundamental modules of our proposed system are summarized in Figure 2. Unlike an end-to-end system, our modular architecture decouples perception and control into separate modules. The overall neural network consists of two modules: *Perception Network* ϕ and *Control Network* φ. The input state includes image I and the physical measurements M of the UAV state (*i.e.* current orientation and velocity). The action is a 4-channel control signal (T: Throttle, A: Aileron/Roll, E: Elevator/Pitch, R: Rudder/Yaw). The Perception Network is parameterized by θ_p and the control network is parameterized by θ_c. The control network takes the predictions of the perception network and UAV state as input and outputs the final control predictions. The whole policy can be described as follows:

$$\pi(s|\theta) = \pi(s|\theta_p, \theta_c) = \varphi(\phi(s_I|\theta_p), s_M|\theta_c) \tag{2}$$

The overall loss is defined in Equation 3 as a weighted sum of perception and control loss. Note that the perception loss comes from self-supervision by minimizing the difference between the ground truth and predicted waypoints, while the control loss in Equation 4 comes from applying OIL to learn from basic teachers (automated PID controllers in our case).

$$\mathbb{L} = \underbrace{\mathbb{L}_c(\pi(s|\theta), a)}_{\text{control loss}} + \lambda \underbrace{\mathbb{L}_p(\phi(s_I|\theta_p), \text{wp}_{gt})}_{\text{perception loss}} \tag{3}$$

$$\mathbb{L}_c(\pi(s|\theta), a) = \mathbb{L}_c(\varphi(\phi(s_I|\theta_p), s_M|\theta_c), a) \tag{4}$$

In general, this optimization problem can be solved by minimizing the overall loss with respect to θ_p and θ_c at the same time. The gradients are as follows:

$$\frac{\partial \mathbb{L}}{\partial \theta_p} = \frac{\partial \mathbb{L}_c}{\partial \theta_p} + \lambda \frac{\partial \mathbb{L}_p}{\partial \theta_p} = \frac{\partial \mathbb{L}_c}{\partial \varphi}\frac{\partial \varphi}{\partial \phi}\frac{\partial \phi}{\partial \theta_p} + \lambda \frac{\partial \mathbb{L}_p}{\partial \phi}\frac{\partial \phi}{\partial \theta_p} \tag{5}$$

$$\frac{\partial \mathbb{L}}{\partial \theta_c} = \frac{\partial \mathbb{L}_c}{\partial \varphi}\frac{\partial \varphi}{\partial \theta_c} \tag{6}$$

In our case, a good perception network is essential to achieving good controls. To maintain modularity and reduce training time, we first optimize only for θ_p while ignoring the control loss. After the perception network converges, we fix θ_p and optimize for θ_c. The intermediate self-supervision through waypoint predictions ensures a clear separation of perception and control and allows for better evaluation of each module.

This modular approach has several advantages. Since only the control module is specific to the UAV dynamics, the perception module can simply be swapped out allowing the UAV to fly in completely different environments without any modification to the control module. Similarly, the perception module can be used on UAVs with different dynamics with their own corresponding control module, which could be a simple PID controller, a model-predictive controller as in [23], or a learned DNN control policy as proposed in this work. In this case, it is possible to add links between the perception and control networks and then finetune the joint network in an end-to-end fashion. One could also connect the two networks and use the waypoint labels as intermediate supervision for the perception part while training the joint model end-to-end. While these variants are interesting, we specifically refrain from such connections to safeguard the attractive modularity properties.

In what follows, we provide details of the architecture, implementation, and training procedure for both the perception and control modules. Note that OIL and the proposed

architecture can also be applied to other types of vision-based navigation tasks (*e.g.* autonomous driving).

4. Network and Training Details

The perception module is a DNN with five convolutional and three fully connected layers. It takes raw RGB images as an input and predicts waypoints relative to the UAV's current position, which remains unknown in 3D. The waypoints predicted by the perception module are input into the control network along with the current UAV state (velocity and orientation). The control network is lightweight with only 3 fully connected layers. The task of the control network is to output a 4-channel control signal for the UAV, thus, replicating a human pilot's stick input.

4.1. Perception

Here, we provide a detailed description of our waypoint encoding, network architecture, and training parameters.

Drawbacks of predicting controls directly. Note that it is common practice to create an end-to-end DNN to directly predict controls from single images. However, this has several limitations. **(i)** A teacher is required for data collection and the controls are tightly coupled to the teacher and vehicle dynamics, as opposed to our proposed modular DNN. **(ii)** There is no unique mapping from images to controls, since different sequences of controls can lead to the same image. This can result in conflicts and less stable training, if data from multiple teachers is used. **(iii)** When using camera views in addition to the view used for data acquisition (a common augmentation step), it is unclear how the control corresponding to these augmented views should be adjusted, given the underlying complex nature of the task. For the case of driving, it might be sufficient to only predict a steering angle and acceleration. Since the car is confined to a 2D plane (road) and the friction of the tires limits drift, one can easily come up with a simple model, *e.g.* offset the steering by the rotation of the augmented camera view. However, for more complex scenarios where the vehicle moves in 3D and is only minimally constrained by friction, it is hard to come up with such a simple model.

Waypoint Encoding. In contrast, the mapping from image to waypoints is deterministic and unique. For every camera view, the corresponding waypoints can easily be determined and are independent of the vehicle state. We define waypoints along the track as a vertical offset that is measured as the distance between the UAV position and the projected point along the viewing axis, and a horizontal offset that is defined as the distance between the original and projected point along the viewing axis normal. We then encode these waypoints relative to the UAV position and orientation by projecting them onto the viewing axis. Figure 3 illustrates the encoding method.

Predicting waypoints rather than controls does not only facilitate network training, but it also allows for the automatic collection of training data without human intervention (self-supervision). Within the simulator, we simply sample/render the entire training track from multiple views and calculate the corresponding waypoints along the track. Note that it is still possible to use recordings from teachers, as one can use future positions to determine the waypoint for the current frame similar to [46].

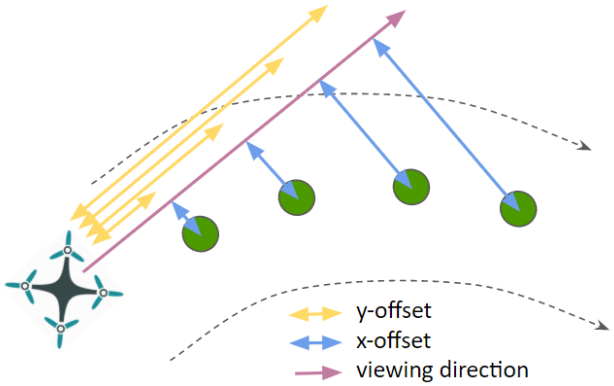

Figure 3: Waypoint encoding.

Network Architecture and Training Parameters. In order to train a DNN to predict waypoints from images, we choose a regression network architecture similar in spirit to the one used by Bojarski *et al.* [4]. However, we make changes to accommodate the complexity of the task at hand and to improve robustness in training. Our DNN architecture is shown in Figure 2 as the perception module. It consists of eight layers: five convolutional with $\{20, 24, 28, 30, 32\}$ filters and three fully-connected with $\{1800, 800, 100\}$ hidden units. Since we implicitly want to localize the track and gates, we use a stride of 2 in the convolutional layers instead of (max) pooling, which adds some degree of translation invariance. This DNN consumes a single RGB-image with 180×320 pixel resolution and is trained to regress the next five waypoints (x-offset and y-offset with respect to the local position of the UAV) using a standard L^2-loss and dropout ratio of 0.5 in the fully-connected layers. As compared to related methods [4, 44], we find that the relatively high input resolution (equivalently high network capacity) is useful to improve the network's ability to look further ahead. This affords the network more robustness for long-term trajectory stability. Visualization results for predicted waypoints on sample track images and the learned perception filters are given in the **supplementary material**.

We implement our model in TensorFlow and train it with a learning rate of 0.0005 using the Adam optimizer. In contrast to other work where the frame rate is sampled down to 10 fps or lower [4, 6, 44], we use the maximum frame rate of 60 fps because the racing environment is much more dynamics (*e.g.* tight turns and high speed). Note that when predicting controls directly from the input image, a high frame rate can be harmful, since fast transitions in control

can lead to similar images with different labels. Since the image to waypoint correspondence is well defined, our approach is not affected by frame rate.

4.2. Control

Here, we present the details of our control network, including network architecture and learning strategy. This network takes the predicted waypoints from the perception network and the UAV state (orientation and velocity) as input and predicts the four UAV controls.

OIL for Control Network. In our experiments, we use two naive PID controllers as the teachers and denote them as μ_1 and μ_2. The first PID controller μ_1 is conservative, as it accurately follows the center of the track and flies through gates precisely but it is relatively slow. Its output control values are a function of the first predicted waypoint wp_1 and UAV state. The second PID control μ_2 is aggressive, as it flies at maximum speed but can often overshoot gates on sharp turns due to inertia and the limits of the UAV. Its output control values are a function of the fourth predicted waypoint wp_4 and the UAV state. We use a three-layer fully connected network to approximate the policy φ of the learner. The state of the learner is a vector concatenation of the predicted waypoints and the UAV state (physical measurements): $s = [s_{wp_{1-n}}, s_M] = [\phi(s_I|\theta_p), s_M]$.

As such, the states of the learner φ, the conservative PID μ_1, and the aggressive PID μ_2, are $[s_{wp_{1-n}}, s_M]$, $[s_{wp_1}, s_M]$ and $[s_{wp_4}, s_M]$ respectively. At each time step, the state-action pairs (s, a) in the *Temporary Observational Buffers* B_1 and B_2 are $([s_{wp_{1-n}}, s_M], \mu_1(s_{wp_1}))$ and $([s_{wp_{1-n}}, s_M], \mu_2(s_{wp_4}))$.

An illustration of the *buffering strategy* is shown in Figure 4 with buffer size $k = 3$. At time step t, there are k unforeseeable samples $\{(s_t, a_t)\}_{t-k}^{t-1}$ in B. The next sample is (s_t, a_t). The samples ahead of (s_{t-k}, a_{t-k}) are stored in the ground truth training database D. Figure 4 illustrates two cases of buffering operations.

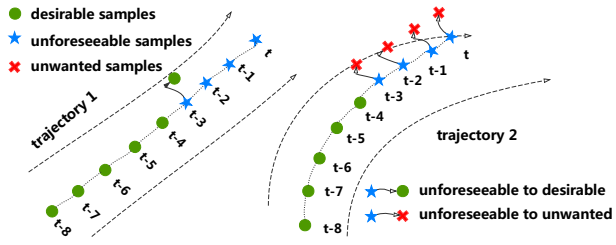

Figure 4: Illustration of Buffer Strategy. In Trajectory 1 the teacher remains in the track: the sample (s_{t-k}, a_{t-k}) will become a desirable sample and will be added to the ground truth database D. In Trajectory 2 the teacher leaves the track: the samples $\{(s_t, a_t)\}_{t-k}^{t}$ will be discarded.

Network Architecture and Training Details. The goal of the control network is to find a control policy φ that minimizes the control loss $\mathbb{L}_c$:

$$\varphi^*(s|\theta_c) = \arg\min_{\varphi} \mathbb{E}_{(s,a)\sim D}[\mathbb{L}_c(\varphi(s|\theta_c), a)] \quad (7)$$

The control module is also implemented in Tensorflow. It consists of three fully-connected layers with hidden units $\{64, 32, 16\}$, and a dropout in the second layer with dropout ratio of 0.5. The loss function $\mathbb{L}_c$ is a weighted ℓ_1-loss optimized by the Adam Optimizer with a learning rate of 10^{-4}. The control network is updated by OIL in an online fashion, while the UAV runs through a training track. A *temporary observational buffer* with size k is used to temporally store the last k observational state-action pairs from each teacher ($k_1 = 1$ for B_1 and $k_2 = 50$ for B_2). To improve learning, we add an Ornstein Uhlenbeck process [25] noise to the output of the control network to allow for exploration at the beginning and move the UAV by the learner's control predictions, but the action is labeled by the conservative PID. After the initial exploration, the teachers take over control. At each time step, if a teacher leaves the track, its temporary observational buffer is flushed and no state-action pairs are added to D for training. If the teacher keeps the UAV within the track boundaries, the oldest state-action pair in its temporary observational buffer is added to D, as shown in Figure 4. At each time step, the network parameters θ_c are updated by back propagation to minimize the difference between the learner's UAV policy φ and the demonstrations (state-action pairs) in D, which are considered to be believable after discarding unwanted behaviors.

As such, the control network is learning from experiences (akin to reinforcement learning), but it is supervised throughout by multiple teachers (PID controllers). An advantage to our approach is that multiple teachers are able to teach throughout the control network's exploration. There is no need to develop approaches as in [18] to have the teacher diverge from its optimal output to induce exploration. Similarly, the control network never becomes dependent on the teachers, but gradually becomes independent and eventually learns to outperform them. The UAV improves with multiple teachers, while simultaneously reducing the number of training iterations and hyperparameter searches needed.

5. Experiments

Experimental Setup. Our UAV racing environment in UE4Sim [29] (see Figure 1) provides capabilities to generate labeled datasets for offline training (*e.g.* imitation learning), and interact with the simulation environment for online learning (*e.g.* OIL or reinforcement learning) as well as online evaluation. We design seven racing tracks for training and seven tracks for testing. To avoid user bias, we collect online images and trace their contours to create uniquely stylized tracks. We select the tracks with two aspects in mind. (1) The tracks should be similar to what racing professionals are accustomed to, and (2) they should

offer enough diversity for network generalization on unseen tracks (see Figure 5). For all of the following evaluations, both the trained networks and human pilots are tasked with flying two laps on the test tracks and are scored based on the percentage of gates they fly through, their overall time, and the number of required resets. We reset the UAV at the next gate, if it has not reached it within 10 seconds after passing through the previous gate. This occurs if the UAV crashes beyond recovery or drifts off the track. Human pilots are given as much time as needed to practice on the training tracks, before attempting the test tracks. Due to space limits, we only report average results across all test tracks for the three metrics. Detailed track results and visualizations of the UAV racing are in the **supplementary material**.

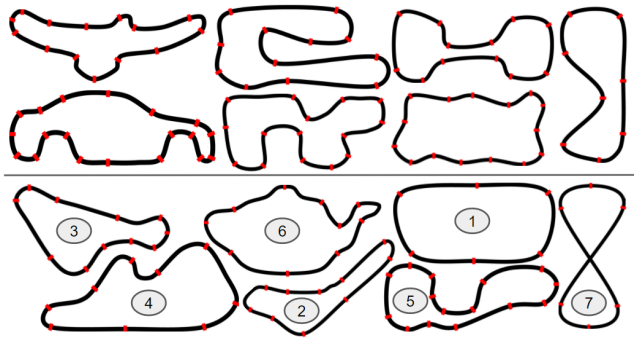

Figure 5: The seven training tracks (top) and the seven testing tracks (bottom). Gates are marked in red.

Comparison to State-of-the-Art Baselines. We compare our system for UAV racing to the two most related and recent network architectures, the first denoted as Nvidia (for self-driving cars [4]) and the second as MAV (for forest path navigating UAVs [44]). Both the Nvidia and MAV networks use data augmentation from an additional left and right camera. For the Nvidia network, the exact offset choices for training are not publicly known, so we use a rotational offset of $\{-30°, 30°\}$. For the MAV network, we use the same augmentation parameters in the paper, *i.e.* a rotational offset of $\{-30°, 30°\}$. We modify the MAV network to allow for a regression output instead of its original classification (left, center and right controls). This is necessary, since our task is much more complex and discrete controls lead to poor performance. We assign corrective controls to the augmentation views using a fairly simple but effective strategy, with details reported in the **supplementary material**.

While the domains of these methods are similar, it should be noted that flying a high-speed racing UAV is a particularly challenging task, especially since the effect of inertia is much more significant and there are more degrees of freedom. To ensure a strong end-to-end baseline, we build an end-to-end network that takes the state of the UAV (exactly like our control module) as an input along with the image. We also augment the data with 18 additional camera views (exactly like our perception module) and assign the best corrective controls after extensive cross-validation search.

Table 1 (Baselines) compares the results of these baselines against our method. MAV needs more than 7 resets and only completes about 60% of gates on average, while taking more than twice the time. Nvidia performs slightly better, but still needs about 4 resets and only completes 80% of gates on average. While the end-to-end trained version of our network achieves better performance than MAV and Nvidia, our modular (self-supervised) and OIL trained network easily outperforms it without the need for supervised corrective controls. In fact, it outperforms all baselines by a considerable margin in all three evaluation metrics.

Ablation Study. Here, we compare our OIL trained control network to the teachers it learned from. The perception network is kept the same. The summary of this comparison is given in Table 1 (Ablation), where we find that our learned policy is able to outperform both teachers (PID1 and PID2). Further, as Figure 7 clearly shows, both teachers are imperfect. PID1 completes most gates while flying very slowly and PID2 misses many gates while flying very quickly. However, since our control module is designed to learn only from the best behaviour of both teachers, it completes all gates at a high speed. We also highlight the importance of the *Temporary Observational Buffer* in Table 1 (Ablation), which shows that removing this buffer results in a significant drop in performance, since the learner is forced to learn from all the teacher demonstrations including the undesirable ones.

Comparison to Human Performance. We also compare our system to three pilots with different levels of skill: novice (has never flown before), intermediate (a moderately experienced pilot), and a professional (a competitive racing pilot with many years of experience). The pilots are given the opportunity to fly the seven training tracks as many times as needed until they successfully complete the tracks at their best time while passing through all gates. The pilots are then scored on the test tracks in the same fashion as the trained networks. The results are summarized in Table 1 (Human vs. Machine). Our network is able to beat both the novice and intermediate pilots by a margin and is not far off from the professional pilot. Interestingly, our network flies more consistently than even the professional racing pilot while remaining reliably on the track (see Figure 7).

Adaptation through Modularity. We replace the low-quality grass textures with high-quality grass and show that our perception network generalizes without any modification. If our perception network is trained on diverse environments/textures, it would learn even more invariance to the background, as demonstrated in [39]. However, generalization only works up to some extent and usually requires heavy data augmentation. In some applications, it might not even be desirable to generalize too broadly as the performance in the target domain often suffers as a result. For

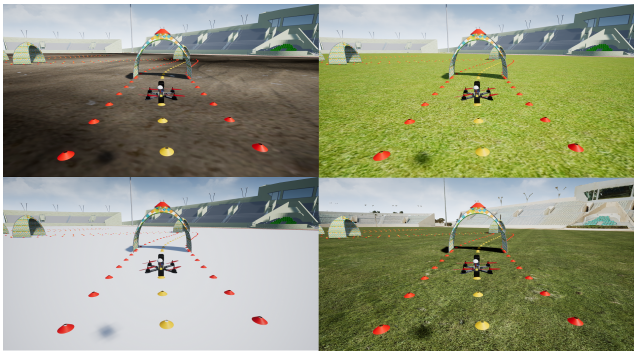

Figure 6: Simulated UAV racing stadium with different textures (mud, grass, snow, and HD grass)

such cases, our modular approach allows to simply swap out the perception module to adapt to any environment (see Figure 6). We train the perception network on different environments while keeping the control network fixed and show almost perfect transfer of the control policy. Average performance on these environment variations across all seven testing tracks are summarized in Table 1 (Adaptation).

Table 1: Results for the following experiments: Comparison to State-of-the-Art Baselines, Ablation Study, Comparison to Human Performance, and Adaptation through Modularity

UAV Pilot	Score	Time	Resets
		Baselines	
End2End (MAV)	62.69%	140.72	7.14
End2End (Nvidia)	79.85%	126.43	3.86
End2End (Ours)	95.52%	98.99	0.86
		Ablation	
PID1 (Conservative)	98.51%	137.61	0.29
PID2 (Aggressive)	65.67%	107.48	6.57
Ours (No Buffer)	79.85%	93.45	3.86
		Human vs. Machine	
Human (Novice)	98.51%	133.89	0.29
Human (Intermediate)	100%	83.32	0
Human (Professional)	100%	49.73	0
		Adaptation	
Ours (HD Grass)	100%	69.75	0
Ours (Mud)	100%	67.54	0
Ours (Snow)	100%	68.26	0
Ours (WP + OIL)	**100%**	**66.94**	**0**

6. Conclusions and Future Work

In this paper, we propose a framework based on observational imitation learning (OIL) and self-supervision to teach an unmanned aerial vehicle (UAV) to fly through challenging racing tracks at very high speeds, an unprecedented

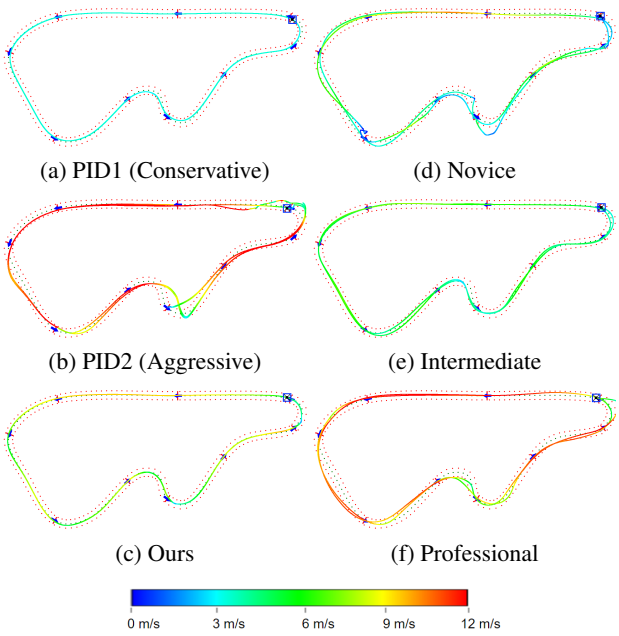

Figure 7: Comparison between our learned policy and its teachers (*column1*) and human pilots (*column2*), on a test track. Color encodes speed as a heatmap, where blue is the minimum speed and red is the maximum speed.

and difficult task. To do this, we train a network to predict UAV controls from raw image data, made possible by a photo-realistic simulator with realistic UAV physics. Extensive experiments demonstrate that our modular network with OIL trained control module outperforms state-of-the-art methods and flies more consistently than human pilots.

Since perception is decoupled from control, one very interesting avenue for future work is to apply OIL to a real UAV platform. The transferability of the perception network can be tested with a conservative PID controller handling controls. Similarly, the control module can be evaluated using a GPS to calculate waypoints and feeding them to the control network to evaluate its predictive response. In the future, we aim to transfer the network trained in the simulator to compete against human pilots in real-world racing scenarios. Although we accurately modeled the simulated racing environment, the differences in appearance between the simulated and real-world will need to be reconciled. Therefore, we will investigate deep transfer learning techniques to enable a smooth transition to the real-world. Since our developed simulator and its seamless interface to deep learning platforms is generic in nature, we expect that this combination will open up unique opportunities for the community to develop better automated UAV flying methods, to expand its reach to other fields of autonomous navigation such as self-driving cars, and to benefit other interesting AI tasks (*e.g.* obstacle avoidance).

References

[1] P. Abbeel, A. Coates, M. Quigley, and A. Y. Ng. An application of reinforcement learning to aerobatic helicopter flight. In P. B. Schölkopf, J. C. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 1–8. MIT Press, 2007.

[2] O. Andersson, M. Wzorek, and P. Doherty. Deep learning quadcopter control via risk-aware active learning. In *Thirty-First AAAI Conference on Artificial Intelligence (AAAI), 2017, San Francisco, February 4-9. :*, 2017. Accepted.

[3] P. W. Battaglia, J. B. Hamrick, and J. B. Tenenbaum. Simulation as an engine of physical scene understanding. *Proceedings of the National Academy of Sciences*, 110(45):18327–18332, 2013.

[4] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba. End to end learning for self-driving cars. *CoRR*, abs/1604.07316, 2016.

[5] D. Borsa, B. Piot, R. Munos, and O. Pietquin. Observational learning by reinforcement learning. *CoRR*, abs/1706.06617, 2017.

[6] C. Chen, A. Seff, A. Kornhauser, and J. Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV '15, pages 2722–2730, Washington, DC, USA, 2015. IEEE Computer Society.

[7] A. Dosovitskiy and V. Koltun. Learning to act by predicting the future. volume abs/1611.01779, 2017.

[8] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun. CARLA: An open urban driving simulator. In S. Levine, V. Vanhoucke, and K. Goldberg, editors, *Proceedings of the 1st Annual Conference on Robot Learning*, volume 78 of *Proceedings of Machine Learning Research*, pages 1–16. PMLR, 13–15 Nov 2017.

[9] A. Gaidon, Q. Wang, Y. Cabon, and E. Vig. Virtual worlds as proxy for multi-object tracking analysis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4340–4349, 2016.

[10] X. Guo, S. Singh, H. Lee, R. Lewis, and X. Wang. Deep learning for real-time atari game play using offline monte-carlo tree search planning. In *Proceedings of the 27th International Conference on Neural Information Processing Systems*, NIPS'14, pages 3338–3346, Cambridge, MA, USA, 2014. MIT Press.

[11] S. Ha and C. K. Liu. Iterative training of dynamic skills inspired by human coaching techniques. *ACM Trans. Graph.*, 34(1):1:1–1:11, Dec. 2014.

[12] P. Hamalainen, S. Eriksson, E. Tanskanen, V. Kyrki, and J. Lehtinen. Online motion synthesis using sequential monte carlo. *ACM Trans. Graph.*, 33(4):51:1–51:12, July 2014.

[13] P. Hamalainen, J. Rajamaki, and C. K. Liu. Online control of simulated humanoids using particle belief propagation. *ACM Trans. Graph.*, 34(4):81:1–81:13, July 2015.

[14] M. Hehn and R. DAndrea. Real-time trajectory generation for quadrocopters. *IEEE Transactions on Robotics*, 31(4):877–892, Aug 2015.

[15] M. Hejrati and D. Ramanan. Analysis by synthesis: 3d object recognition by object reconstruction. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 2449–2456, June 2014.

[16] D. Isele, A. Cosgun, K. Subramanian, and K. Fujimura. Navigating intersections with autonomous vehicles using deep reinforcement learning. *CoRR*, abs/1705.01196, 2017.

[17] E. Ju, J. Won, J. Lee, B. Choi, J. Noh, and M. G. Choi. Data-driven control of flapping flight. *ACM Trans. Graph.*, 32(5):151:1–151:12, Oct. 2013.

[18] G. Kahn, T. Zhang, S. Levine, and P. Abbeel. PLATO: policy learning using adaptive trajectory optimization. *CoRR*, abs/1603.00622, 2016.

[19] D. K. Kim and T. Chen. Deep neural network for real-time autonomous indoor navigation. *CoRR*, abs/1511.04668, 2015.

[20] J. Koutník, G. Cuccu, J. Schmidhuber, and F. Gomez. Evolving large-scale neural networks for vision-based reinforcement learning. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, GECCO '13, pages 1061–1068, New York, NY, USA, 2013. ACM.

[21] J. Koutník, J. Schmidhuber, and F. Gomez. *Online Evolution of Deep Convolutional Network for Vision-Based Reinforcement Learning*, pages 260–269. Springer International Publishing, Cham, 2014.

[22] A. Lerer, S. Gross, and R. Fergus. Learning Physical Intuition of Block Towers by Example, 2016. arXiv:1603.01312v1.

[23] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *J. Mach. Learn. Res.*, 17(1):1334–1373, Jan. 2016.

[24] S. Levine and V. Koltun. Guided policy search. In S. Dasgupta and D. McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1–9, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

[25] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *ICLR*, abs/1509.02971, 2016.

[26] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937, 2016.

[27] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, Feb. 2015.

[28] Y. Movshovitz-Attias, Y. Sheikh, V. Naresh Boddeti, and Z. Wei. 3d pose-by-detection of vehicles via discriminatively reduced ensembles of correlation filters. In *Proceedings of the British Machine Vision Conference*. BMVA Press, 2014.

[29] M. Mueller, V. Casser, J. Lahoud, N. Smith, and B. Ghanem. UE4Sim: A Photo-Realistic Simulator for Computer Vision Applications. *ArXiv e-prints*, Aug. 2017.

[30] U. Muller, J. Ben, E. Cosatto, B. Flepp, and Y. L. Cun. Off-road obstacle avoidance through end-to-end learning. In Y. Weiss, P. B. Schölkopf, and J. C. Platt, editors, *Advances in Neural Information Processing Systems 18*, pages 739–746. MIT Press, 2006.

[31] A. Y. Ng, D. Harada, and S. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, volume 99, pages 278–287, 1999.

[32] J. Papon and M. Schoeler. Semantic pose using deep networks trained on synthetic RGB-D. *CoRR*, abs/1508.00835, 2015.

[33] X. B. Peng, G. Berseth, and M. van de Panne. Terrain-adaptive locomotion skills using deep reinforcement learning. *ACM Transactions on Graphics (Proc. SIGGRAPH 2016)*, 35(4), 2016.

[34] X. B. Peng, G. Berseth, K. Yin, and M. van de Panne. Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Transactions on Graphics (Proc. SIGGRAPH 2017)*, 36(4), 2017.

[35] D. A. Pomerleau. Advances in neural information processing systems 1. chapter ALVINN: An Autonomous Land Vehicle in a Neural Network, pages 305–313. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1989.

[36] S. R. Richter, V. Vineet, S. Roth, and V. Koltun. Playing for data: Ground truth from computer games. In *ECCV*, 2016.

[37] G. Ros, L. Sellart, J. Materzynska, D. Vazquez, and A. Lopez. The SYNTHIA Dataset: A large collection of synthetic images for semantic segmentation of urban scenes. 2016.

[38] S. Ross, G. J. Gordon, and J. A. Bagnell. No-regret reductions for imitation learning and structured prediction. *CoRR*, abs/1011.0686, 2010.

[39] F. Sadeghi and S. Levine. (cad)$ˆ2$rl: Real single-image flight without a single real image. *CoRR*, abs/1611.04201, 2016.

[40] A. E. Sallab, M. Abdou, E. Perot, and S. Yogamani. End-to-end deep reinforcement learning for lane keeping assist. *CoRR*, abs/1612.04340, 2016.

[41] A. E. Sallab, M. Abdou, E. Perot, and S. Yogamani. Deep reinforcement learning framework for autonomous driving. *Electronic Imaging*, 2017(19):70–76, 2017.

[42] U. Shah, R. Khawad, and K. M. Krishna. Deepfly: Towards complete autonomous navigation of mavs with monocular camera. In *Proceedings of the Tenth Indian Conference on Computer Vision, Graphics and Image Processing*, ICVGIP '16, pages 59:1–59:8, New York, NY, USA, 2016. ACM.

[43] S. Sharifzadeh, I. Chiotellis, R. Triebel, and D. Cremers. Learning to drive using inverse reinforcement learning and deep q-networks. *CoRR*, abs/1612.03653, 2016.

[44] N. Smolyanskiy, A. Kamenev, J. Smith, and S. Birchfield. Toward Low-Flying Autonomous MAV Trail Navigation using Deep Neural Networks for Environmental Awareness. *ArXiv e-prints*, May 2017.

[45] J. Tan, Y. Gu, C. K. Liu, and G. Turk. Learning bicycle stunts. *ACM Trans. Graph.*, 33(4):50:1–50:12, July 2014.

[46] H. Xu, Y. Gao, F. Yu, and T. Darrell. End-to-end learning of driving models from large-scale video datasets. *CoRR*, abs/1612.01079, 2016.

[47] H. Xu, Y. Gao, F. Yu, and T. Darrell. End-to-end learning of driving models from large-scale video datasets. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.