Abstract

Recent work has explored 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 Observational Imitation Learning (OIL), a novel imitation learning variant that supports online training and automatic selection of optimal behavior by observing multiple imperfect teachers. We apply our proposed methodology to the challenging problems of autonomous driving and UAV racing. For both tasks, we utilize the Sim4CV simulator [18] that enables the generation of large amounts of synthetic training data and also allows for online learning and evaluation. We train a perception network to predict waypoints from raw image data and use OIL to train another network to predict controls from these waypoints. Extensive experiments demonstrate that our trained network outperforms its teachers, conventional imitation learning (IL) and reinforcement learning (RL) baselines and even humans in simulation.

1. Introduction

In the machine learning community, solving complex sequential prediction problems usually follows one of two different approaches: reinforcement learning (RL) or supervised learning (SL), more specifically imitation learning (IL). On the one hand, the learner in conventional IL is required to trust and replicate authoritative behaviors of a teacher. The drawbacks are primarily the need for extensive manually collected training data and the inherent subjectivity to potential negative behaviors of teachers, since in many realistic scenarios they are imperfect. IL often does not generalize well to completely new environments or differing control scenarios that are not well represented during training. On the other hand, RL does not specifically require supervision by a teacher, as it searches for an optimal policy that leads to the highest eventual reward. However, a good reward function, which offers the agent the opportunity to learn desirable behaviors, requires tedious and meticulous reward shaping [20]. Recent methods have used RL to learn simpler tasks without supervision [5], but they require excessive training time and a very fast simulation (e.g., 1,000 fps). In this paper, we demonstrate that state-of-the-art performance can be achieved by incorporating RL concepts into direct imitation to learn only the successful actions of multiple teachers. We call this approach Observational Imitation Learning (OIL). Unlike conventional IL, OIL enables learning from multiple teachers with a method for discarding bad maneuvers by using a reward based online evaluation of the teachers at training time. Furthermore, this approach enables a greater control over training in diverse environments that require different control dynamics. In our experiments, it is shown that this approach leads to greater robustness and improved performance, as compared to various state-of-the-art IL and RL methods. Moreover, OIL allows for control networks to be trained in a fully self-supervised fashion requiring no human annotation but rather can be trained using automated agents.

We apply OIL to both autonomous driving and UAV high speed racing in order to demonstrate the diverse scenarios at which it can be applied to solve sequential prediction
problems. We follow recent work [5] that tests AI systems through the use of computer games. We use Sim4CV based on the Unreal Engine 4 (UE4), which has both a residential driving environment with a physics based car and gated racing tracks for UAV racing [18]. The simulator is multi-purpose, as it enables the generation of synthetic image data, reinforcement based training in real-time, and evaluation on unseen tracks. We demonstrate that OIL makes possible a self-supervised and modular neural network capable of learning how to control both vehicular driving and the more complex task of UAV racing in the simulated Sim4CV environment. Through extensive experiments, we demonstrate that OIL outperforms its teachers, conventional IL and RL approaches and even humans in simulation.

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 imperfect teachers, but only updates the policy with the best maneuvers of each teacher, eventually outperforming all of them. (2) We introduce a flexible network architecture that adapts well to different control scenarios and complex navigation tasks (e.g. autonomous driving and UAV racing) using OIL in a self-supervised manner without any human training demonstrations. (3) To the best of our knowledge, this paper is the first to apply imitation learning to multiple teachers while being robust to teachers that exhibit bad behavior.

2. Related Work

The task of training an actor (e.g. ground vehicle, human, or UAV) to physically navigate through an unknown environment has traditionally been approached either through supervised learning (SL) and in particular Imitation Learning (IL), Reinforcement Learning (RL), or a combination of the two. 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 such as [7, 22, 23, 31, 11] 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. Subsequently, we give a brief overview of relevant work within the domain of IL, RL and its intersection.

Imitation Learning (IL). For the particular case of physics based navigation, IL can be advantageous when high dimensional feedback can be recorded. It has been applied to both autonomous driving [2, 4, 32, 37] and UAV navigation [32, 9, 19]. In particular, DAGGER [26] has been widely used for many robotic control tasks. However, there are some limitations that prevent DAGGER to scale up to some problems in practice. Data augmentation 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. Obtaining optimal or near-optimal expert data can be costly or even infeasible. Moreover, the flaws and mistakes of the teachers are learned along with differing responses. See Section 5.2 for the comparison of two popular IL methods (Behavioral Cloning [35] and Dagger) to OIL. Follow-up work such as AGGREVATE [25] and Deeply AggreVaTed [33] are trying to mitigate this problem by introducing exploratory actions and measuring actual induced cost instead of optimizing for expert imitation only. They also claim exponentially higher sample efficiency than many classical RL methods. A number of improvements in other respects have been published, such as SafeDagger [38] that aims to make DAgGER more (policy) query-efficient, and LOLS [3] that aims to improve upon cases where the reference policy is suboptimal.

Reinforcement Learning (RL). RL provides an alternative to IL by using rewards and many iterations of exploration to help discover the proper response through interactive trial and error. Recent work on autonomous car driving has employed RL [15, 14, 10, 28, 27, 8, 6]. One of the main disadvantages is that RL networks may not be able to discover the optimal outputs in higher-order control tasks. For ex-
ample, Dosovitskiy et al. [6] find RL to under-perform in vehicular navigation due to the extensive hyperparameter space. RL methods can be divided into three classes: value-based, policy-based, and actor-critic based methods [34]. In particular, actor-critic based methods, e.g. A3C [15] and DDPG [14], are notably the most popular algorithms in the RL community. However, achieving strong results with RL is difficult, since it is very sensitive to the reward function, it can be sample inefficient, and it requires extensive training time due to the large policy space (see Section 5.2 for comparison of DDPG to OIL). Methods such as TRPO [30] has been developed to provide monotonic policy improvements in most cases, but still require extensive training time.

**Combined approaches.** Several methods exist that combine the advantages of IL and RL. Most of them focus on tackling the problem of low RL sample efficiency by pre-initializing with suitable expert demonstrations (e.g. CIRL [13] and Guided Policy Search [12, 5]). Others focus on risk awareness as real-world deployment failures can be costly [1]. We draw inspiration for OIL from these hybrid approaches. In contrast to pure IL, OIL can prevent itself from learning bad demonstrations from imperfect teachers by observing teachers’ behaviours and estimating the advantage or disadvantage to imitate them. Unlike pure RL, it converges to a high performance policy without too much exploration since it is guided by the best teacher behaviors. While sharing the advantage of higher sample efficiency with other hybrid approaches, our method has the specific advantage of inherently dealing well with bad demonstrations, which is a common occurrence in real-world applications.

### 3. Methodology

After giving a brief review of related learning strategies for sequential decision making (i.e. Markov Decision Process, Imitation Learning, and Reinforcement Learning), we introduce our proposed Observational Imitation Learning (OIL), which enables automatic selection of the best teacher (from multiple teachers) at each time step of online learning.

#### 3.1. Markov Decision Process

OIL is a method that enables a learner to learn from multiple sub-optimal or imperfect teachers and eventually to outperform all of them. To achieve this goal, training needs to be done by repeatedly interacting with an environment $\mathcal{E}$. We consider the problem as a Markov Decision Process (MDP) consisting of an agent and environment. At every time step $n$, the agent observes a state $s_n$ or a partial observation $o_n$ of state $s_n$. In our setting, we assume the environment is partially-observed but we use $s_n$ to represent the state or the partial observation for simplicity. Given $s_n$, the agent performs an action $a_n$ within the available action space $\mathcal{A}$ based on its current policy $\pi$, where $\theta$ is the parameter of the policy. Then, the environment provides the agent a scalar reward $r_n$ according to a reward function $R(s_n, a_n)$ and transfers to a new state $s_{n+1}$ within state space $\mathcal{S}$ under the environment $\mathcal{E}$ transition distribution $p(s_{n+1}|s_n, a_n)$. After receiving an initial state $s_1$, the agent generates a trajectory $\tau = (s_1, a_1, s_2, a_2, ..., s_N)$ after $N$ time steps. The trajectory can end after a certain number of time steps or after the agent reaches a goal or terminal state. The objective of solving a MDP problem is to find an optimal policy $\pi^*$ from policy space $\Pi$ that maximizes the expected sum of discounted future rewards, $R_n$ at time $n$. It is popularly known as the value function:

$$V^\pi(s) = \mathbb{E} \left[ R_n | s_n = s, \pi \right]$$

$$= \mathbb{E} \left[ \sum_{n'=n}^{N} \gamma^{n'-n} r_{n'} | s_n = s, \pi \right],$$

where $\gamma \in [0, 1]$ is a discounted factor that trades-off the importance of immediate and future rewards, $N$ is the time step when the trajectory $\tau$ terminates. The optimal policy $\pi^*$ maximizes the value function for all $s \in \mathcal{S}$:

$$\pi^* = \operatorname{argmax}_{\pi \in \Pi} V^\pi(s)$$  

#### 3.2. Imitation Learning

Imitation learning (IL) is a supervised learning approach to solve sequential decision making problems by mimicking an expert policy $\pi^e$. Instead of directly optimizing the value function, IL minimizes a surrogate loss function $\mathbb{L}(s, \pi, \pi^e)$. Let $d_{\pi}$ denote the average distribution of visited observations when an arbitrary policy $\pi$ is executed for $T$ time steps. Behavioral Cloning, one of the simplest IL algorithms, trains a learner policy network $\pi^l$ to fit the input (observations) and output (actions) of the expert policy by minimizing the surrogate loss [26].

$$\pi^l = \operatorname{argmin}_{\pi \in \Pi} \mathbb{E}_{s \sim d_{\pi^e}} \mathbb{L}(s, \pi, \pi^e)$$

However, this leads to poor performance because the encountered observation spaces of the learner and the expert are different thus, violating the independent, identically distributed (i.i.d) assumption of statistical learning approaches and causes compounding errors [24]. DAGGER (Dataset Aggregation) [26] alleviates these errors in an iterative fashion by collecting the state-action pairs visited by the learned policy, but labeled by the expert. Its goal is to find a policy $\pi^l$ that minimizes the surrogate loss under the observation distribution $d_{\pi}$ induced by the current policy $\pi$.

$$\pi^l = \operatorname{argmin}_{\pi \in \Pi} \mathbb{E}_{s \sim d_{\pi}} \mathbb{L}(s, \pi, \pi^e)$$

A major drawback of DAGGER is that it highly depends on the expert’s performance, where a near-optimal expert is hard to acquire in most tasks.
3.3. Reinforcement Learning

Reinforcement Learning (RL) is another category of methods to solve the MDP problem by trial and error. RL methods can be divided into three classes: value-based, policy-based, and actor-critic-based methods [34]. Specifically, actor-critic-based methods, e.g. A3C [15], and DDPG [14], are currently the most popular algorithms in the RL community. The most related actor-critic method to our work is the Advantage Actor-critic approach [15], which uses an advantage function for policy update instead of the typical value function or Q-function [16]. Intuitively, this advantage function evaluates how much improvement is obtained if action a is taken at state s as compared to the expected value. It is formally defined as follows [21]:

$$A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$$  \hspace{1cm} (5)

3.4. Observational Imitation Learning (OIL)

As discussed in Section 3.2 imitation learning requires a near-optimal teacher and extensive augmentation for exploration. Getting labeled expert data is expensive and not scalable. While RL approaches (Section 3.3) do not require supervision and can freely explore an environment, they are time-consuming and may never learn a good policy unless the reward is very well designed. In an effort to combine the strengths of both approaches we propose Observational Imitation Learning (OIL). Inspired by advantage actor-critic learning [15], we learn to imitate only the best behaviours of several sub-optimal teachers. We do so by estimating the value function of each teacher and only keeping the best to imitate. As a result, we can learn a policy that outperforms each of its teachers. Learning from multiple teachers also allows for exploration, but only of feasible states, leading to faster learning than in RL.

Since we do not require expert teachers we can obtain labeled data much more cheaply. We assume easy access to K cheap sub-optimal teacher policies (e.g. simple PID controllers). We denote the teacher policy set as $\Pi^t = \{\pi^t_1, \pi^t_2, ..., \pi^t_K\}$, where $\pi^t_k$ is the teacher policy corresponding to teacher k. Let $\pi^t$ denote the learner policy. Denote the advantage function of current learner policy $\pi^t$ compared to teacher policy $\pi^t$ at state s as:

$$A(s, \pi^t, \pi^*) = V^{\pi^*}(s) - V^{\pi^t}(s)$$  \hspace{1cm} (6)

We define our training in terms of observation rounds, each of which is divided into three phases, observing, rehearsing and acting.

**Observe.** We estimate the value functions $V^\pi(s)$ of the learner policy as well as of all teacher policies using Monte-Carlo sampling (rolling-out the trajectory of each policy to get the returns $R_t$). We then select the most critical teacher policy $\pi^t_* = \arg\min_{\pi^t \in \Pi^t} V^\pi(s)$ as the critic. Then we compute the advantage function $A(s, \pi^t, \pi^t_*)$ between the learner policy and the most critical teacher policy. If the advantage $A(s_1, \pi^t_1, \pi^t_*) < 0$ (i.e. there exists a sub-optimal teacher policy with a higher advantage than the current learner policy), we enter the rehearsing phase. Otherwise, we go to the acting phase directly.

**Rehearse.** After computing $A(s_1, \pi^t_1, \pi^t_*)$, we can optimize the policy by actor-critic methods or by optimizing the surrogate loss. In order to benefit from the fast convergence of IL instead of optimizing the advantage function directly, we optimize the surrogate loss 7 iteratively as follows:

$$\pi^{l(i+1)} = \arg\min_{\pi \in \Pi} \mathbb{E}_{s \sim d_{\pi^{l(i)}}}[L(s, \pi^{l(i)}, \pi^*)]$$  \hspace{1cm} (7)

Where $\pi^{l(i)}$ is the learner policy at the i-th iteration. In our implementation, we use a DNN to represent our learner policy as $\pi^*(s|\theta)$, where $\theta$ is the parameter of the neural network. In order to minimize the surrogate loss, we roll out the learner (actor) policy and using the selected teacher (critic) to correct the learner’s actions. In other words, we minimize the surrogate loss with states encountered by the learner policy and actions labeled by the most critical teacher policy. We minimize the surrogate loss by performing gradient descent with respect to $\theta$ on collected data D and update learner policy until $A(s, \pi^t_1, \pi^t_*) > \epsilon$ or I episodes.

**Act.** After rehearsing, the learner policy will perform well at the current trajectory, then we roll out the current policy $\pi^t_1$ to new state $s'_1$ with acting steps J.

4. Network and Training Details

In this section we present a modular network architecture for autonomous driving and UAV racing that solves the navigation task as a high dimensional perception module (trained with self-supervision) and a low dimensional control module (trained with OIL).

4.1. Modular Architecture

The fundamental modules of our proposed system are summarized in Figure 2. We use modular architecture to reduce the required training time by introducing an intermediate representation layer. The overall neural network consists of two modules: a Perception Network $\phi$ and a Control Network $\varphi$. The input state includes image $\mathcal{I}$ and the
physical measurements $M$ of the vehicle’s state (i.e., current orientation and velocity). The action is a control signal for cars (G: Gas/Brake, S: Steering) or UAV (T: Throttle, A: Aileron/Roll, E: Elevator/Pitch, R: Rudder/Yaw). The Perception Network is parameterized by $\theta_p$ and the control network is parameterized by $\theta_c$. The control network takes the intermediate representation predictions of the perception network and vehicle’s state as input and outputs the final control predictions. The whole policy can be described as follows:

$$\pi'(s|\theta) = \pi'(s|\theta_p, \theta_c) = \varphi(\phi(s_I|\theta_p), s_M|\theta_c) \quad (8)$$

The overall loss is defined in Equation 9 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 intermediate representation $z^*$ and predicted intermediate representation $\phi(s_I|\theta_p)$, while the control loss in Equation 10 comes from applying OIL to learn from multiple imperfect teachers (automated PID controllers in our case) by minimizing the surrogate loss in Equation 7, where $s \sim d_{at}$ and $a^* \sim \pi'\pi'$. In experiments, we choose waypoints as the intermediate representation. Note that it can also be segmentations [17]. Depth images, affordances [4] [29] or the combination of them.

$$\mathbb{L} = \mathbb{L}_c(\pi(s|\theta), a^*) + \lambda \mathbb{L}_p(\phi(s_I|\theta_p), z^*) \quad (9)$$

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

$$\frac{\partial \mathbb{L}}{\partial \theta_p} = \mathbb{L}_c \frac{\partial \mathbb{L}_c}{\partial \theta_p} + \lambda \mathbb{L}_p \frac{\partial \mathbb{L}_p}{\partial \theta_p} + \lambda \mathbb{L}_p \frac{\partial \mathbb{L}_p}{\partial \theta_c} \frac{\partial \mathbb{L}_p}{\partial \theta_c} \quad \frac{\partial \mathbb{L}_p}{\partial \theta_c} = \mathbb{L}_c \frac{\partial \mathbb{L}_c}{\partial \theta_c} \frac{\partial \mathbb{L}_c}{\partial \theta_p} \quad (11, 12)$$

A good perception network is essential to achieving good controls. To maintain modularity and reduce training time, we first optimize only for $\theta_p$ while ignoring the control loss. After the perception network converges, we fix $\theta_p$ and optimize for $\theta_c$.

This modular approach has several advantages over an end-to-end approach (see also [17, 9]). Since only the control module is specific to the vehicle’s dynamics, the perception module can simply be swapped out, allowing the vehicle to navigate in completely different environments without any modification to the control module. Similarly, if the reward function is changed in different tasks, we can simply retrain the control module to learn different policy. 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 MDPs problems (e.g. other vision-based sequential decision making problems).

4.2. Perception

In our case the perception module takes raw RGB images as an input and predicts a trajectory of waypoints relative to the vehicle’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 vehicle state (velocity and orientation).

**Waypoint Encoding.** 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 vehicle 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 vehicle position and orientation by projecting them onto the viewing axis. 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 [36]. Please refer to the supplementary material for further details and an illustration of the waypoint encoding method.

Network Architecture. We choose a regression network architecture similar in spirit to the one used by Bojarski et al. [2]. 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. The 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 vehicle) using a standard \(\ell_2\)-loss and dropout ratio of 0.5 in the fully-connected layers. As compared to related methods [2, 32], 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.

4.3. OIL for Control

Here, we present the details of training the control network using OIL, including network architecture and learning strategy.

Teachers and Learner. In our experiments, we use multiple naive PID controllers as the teachers for control policy. The PID parameters are tuned in only a couple of minutes and validated on a training track to perform well. As the system is very robust to learn from imperfect teachers we don’t need to spend much effort tuning these parameters or achieve optimal performance of the teachers on all training tracks. Although an unlimited number of PID based teachers can be created, we find empirically five to be sufficient for the two control scenarios (Autonomous driving and UAV racing) (For further details see evaluations which demonstrate how the different number of teachers affect learning).

We use a three-layer fully connected network to approximate the control policy \(\varphi\) of the learner. The MDP state of the learner is a vector concatenation of the predicted intermediate representation \((e.g.\,\text{waypoints})\, \tilde{z} = \phi(s_{\mathcal{T}}|\theta_p)\) and the vehicle state (physical measurements for vehicle’s speed and orientation) \(s_{\mathcal{M}}\).

Network Architecture. The goal of the control network is to find a control policy \(\varphi\) that minimizes the control loss \(\mathbb{L}_{\mathcal{C}}\):

\[
\varphi^* = \arg\min_{\varphi} \mathbb{L}_{\mathcal{C}}(\varphi(\tilde{z}, s_{\mathcal{M}}|\theta_c), a^*)
\]

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 \(\mathbb{L}_{\mathcal{C}}\) is a standard \(\ell_2\)-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 vehicle runs through a set of training tracks.

As such, the control network is learning from experiences (akin to reinforcement learning), but it is supervised throughout by multiple teachers to minimize the surrogate loss. An advantage to our approach is that multiple teachers are able to teach throughout the control network’s exploration. The control network never becomes dependent on the teachers, but gradually becomes independent and eventually learns to outperform them.

5. Experiments

5.1. Experimental Setup.

The Sim4CV [18] (see Figure 1) environment provides capabilities to generate labeled datasets for offline training \((e.g.\,\text{imitation learning})\), and interact with the simulation environment for online learning \((e.g.\,\text{OIL or reinforcement learning})\) as well as online evaluation. For each application, we design six environments for training and four for testing. For fair comparison, human drivers/pilots are given as much time as needed to practice on the training tracks, before attempting the test tracks.

For the autonomous driving scenario, the PID teachers, learned baselines and human drivers have to complete one lap. They are scored based on the average error to the center line and the time needed to complete the course. In addition, they need to pass through invisible checkpoints placed at every 50 meters to make sure they stay on the course and do not take any shortcuts. The vehicle is reset at the next checkpoint if it did not reach it within 15 seconds.

For the UAV racing task, all pilots have to complete two laps on the test tracks and are scored based on the percentage of gates they maneuver through and the overall time. Similar to the autonomous driving scenario we reset the UAV at the next checkpoint if it was not reached within 10 seconds. Here, the checkpoints correspond to the racing gates.
**Training Details.** For fair comparison we train each network until convergence or for at most 800k steps. We estimate the value functions for all the teachers and the learner to select the most critical teacher with \( N \) steps rollouts \((N = 180 \text{ for car, } N = 200 \text{ for UAV})\) during observing phase. We execute the learner policy trajectory with also \( N \) steps during rehearsing phase until advantage function \( A(s_t, \pi^l, \pi^t) > \epsilon \) or \( I = 50 \) episodes. We choose \( \epsilon = -0.1 \ast V^t \) for three and five teacher experiments, \( \epsilon = 0 \) for one teacher experiments. We choose \( J = 60 \) as the acting step size.

For OIL we score both the learner and the teachers using a reward function that trades off between the trajectory error and speed. For the UAV we simply use its forward speed and penalize it for going outside the track. For the car we compute the progression along the center of the road and the average error. More details of the reward function are provided in the supplementary material.

### 5.2. Results

**Comparison to State-of-the-Art Baselines.** We compare OIL for both autonomous driving and UAV racing to several DNN baselines. These include both IL approaches: Behaviour cloning, Dagger; and RL: DDPG. For each comparison we implement both learning from the single best teacher and an ensemble of teachers. This essentially allows a broad baseline comparison of 6 different state-of-the-art learning approaches evaluating various IL and RL approaches.

In Tables 1 and 2, the learned approaches are compared to OIL. The evaluations demonstrate that OIL outperforms all learned baselines in both score and timing. Both Behaviour Cloning and Dagger become worse in score or timing with more teachers learning both good and bad behaviours. Moreover, they do not achieve scores better than their teachers in single teacher training. In contrast, OIL improves upon scores both with a single teacher and multiple.

In comparison to DDPG, OIL converges quickly without extensive hyperparameter search and still learns to fly/drive much more precisely and faster.
Comparison to Teachers and Human Performance. We compare our OIL trained control network to the teachers it learned from and human control. The perception network is kept the same for all learned models. The summary of this comparison to OIL is given in Tables 1 and 2. The evaluations demonstrate that OIL outperforms all teachers and novice to intermediate human pilots/drivers. Compared to the expert driver, OIL is 3.83 seconds slower but only has 35.91% error in term of the distance to the center.

Teacher1 has the least error of all teachers and is the only one to perfectly complete all gates in the UAV racing evaluation. However, OIL not only completes all gates at higher speeds but becomes even more precise in centering along the middle of the road in the autonomous driving evaluation. In comparison to humans, OIL is better than novice and intermediate levels and is still slower than expert. A note-able difference between the expert driver and OIL is that OIL has a much lower error in driving. It is able to maintain high speeds while centering most accurately in the center of the tracks.

Ablation study. We investigate the importance of the trajectory length and the number of teachers and report the results in Table 3.

6. Conclusions and Future Work

In this paper, we present Observational Imitation Learning (OIL), a new approach for training a stationary deterministic policy that is not bound by imperfect or inexperienced teachers but rather updates its policy by selecting only the best maneuvers leading to improved performance. The flexible network architecture affords modularity in perception and control, and can be applied to many different types of complex vision-based sequential prediction problems. We demonstrate the ability of the OIL framework to train without supervision a DNN to autonomously drive a car and to fly an unmanned aerial vehicle (UAV) through challenging tracks. The extensive experiments demonstrate that OIL outperforms single and multiple teacher learned IL methods (Behavior Cloning, DAGGER) and RL approaches (DDPG). OIL performance is better than its teachers and experienced humans pilots/drivers.

OIL provides a new learning-based approach that can replace traditional control methods especially in robotics and control systems. We expect our framework can be adapted for other robotic tasks such as visual grasping tasks or visual placing tasks where teachers may be imperfect or unavailable except through automation.

One very interesting avenue for future work is to apply OIL outside simulation. The transferability of the perception network can be tested with a conservative PID controller handling controls. Since OIL online training functions at 60fps, the control module can be trained using an IMU with GPS to calculate waypoints and state, then feed them to a real-time control network in the field. Although Sim4CV uses a high quality gaming engine for rendering, the differences in appearance between the simulated and real-world will need to be reconciled. Moreover, real-world physics, weather and road conditions, and sensor noise will present new challenges to training the control network.

Since OIL as an improved DNN learning approach is generic in nature, we expect it will open up unique opportunities for the community to develop better self-navigation and control systems expanding its reach to other fields of autonomous navigation, and to benefit other interesting AI complex sequential prediction problems (e.g. obstacle avoidance).

References

[1] 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
[2] 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. 2, 6
[3] K.-W. Chang, A. Krishnamurthy, A. Agarwal, H. Daume III, and J. Langford. Learning to search better than your teacher. 2015. 2
[4] 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. 2, 5
[5] A. Dosovitskiy and V. Koltun. Learning to act by predicting the future. volume abs/1611.01779, 2017. 1, 2, 3
[6] 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. 2, 3
[7] 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. 2
[8] D. Isele, A. Cosgun, K. Subramanian, and K. Fujimura. Navigating intersections with autonomous vehicles using deep reinforcement learning. CoRR, abs/1705.01196, 2017. 2
[9] E. Kaufmann, A. Loquercio, R. Ranftl, A. Dosovitskiy, V. Koltun, and D. Scaramuzza. Deep drone racing: Learning agile flight in dynamic environments. In A. Billard, A. Dragan, J. Peters, and J. Morimoto, editors, Proceedings of The 2nd Conference on Robot Learning, volume 87 of Proceedings of Machine Learning Research, pages 133–145. PMLR, 29–31 Oct 2018. 2, 5
**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 controls 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.** 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/course as a vertical offset that is measured as the distance between the vehicle 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. Please refer to Figure 3 for a visualization of the encoding method.

**Perception Network Visualization.** After training the perception network to convergence, we visualize how parts of the network behave in order to get additional insights on what it has intrinsically learned. Figures 4 and 5 show some feature maps for each convolutional layer of the trained networks for an input image. Note how the learned filters are able to extract all necessary information in the scene (i.e. road boundaries or gates and cones). While filters from early layers already produce a reasonable activation heatmap, the deeper layers abstract the relevant features even more clearly. Although the feature map resolution becomes very low in the deeper layers, high activations can still be identified for parts of the image with semantic meaning for autonomous driving and UAV racing.

**Detailed Experimental Results.** We show detailed results for the autonomous driving experiments for each course and report the average error to the center of the lane and the time to complete the course in Tables 4, 5 and 6. In Table 4, we show the results for each of the teachers and provide human baselines. In Table 5, we compare our approach OIL to several learned baselines. In Table 6, we show an ablation study of OIL in regard to trajectory length and number of teachers.

We show detailed results for the autonomous UAV racing experiments for each track and report the number of missed gates and the time to complete two laps in Tables 7 and 8. In Table 7, we show the results for each of the teachers and provide human baselines. In Table 8, we compare our approach OIL to several learned baselines.
Figure 4: Visualization of selected feature maps from convolutional layers 1 to 5 (top to bottom) in our trained perception network. Top image is input at $180 \times 320$ resolution. Notice how the network activates in locations of semantic meaning for the task of autonomous driving, namely the boundaries of the road.

Figure 5: Visualization of selected feature maps from convolutional layers 1 to 5 (top to bottom) in our trained perception network. Top image is input at $180 \times 320$ resolution. Notice how the network activates in locations of semantic meaning for the task of UAV racing, namely the gates and cones.
Reward function of OIL. The reward function of OIL is used to compute the value functions and advantage functions of learner and teachers. For OIL, we use reward functions that score the whole trajectory. Assume the length of the trajectory is $N$. Let $\tau = (s_1, a_1, s_2, a_2, ..., s_N)$ denote the generated trajectory and $S_{\text{end}}$ denote all the terminal states when vehicles leave the course/track or hit obstacles.

For autonomous driving, preciseness is an important measurement that reflects the smoothness and safety of the algorithm. Hence, we use a reward function that considers both speed and trajectory error as follows:

$$R(\tau) = \begin{cases} 
\alpha \sum_{n=1}^{N} e_{n+1} + r_{\text{penalty}} & s_N \notin S_{\text{end}} \\
\alpha \sum_{n=1}^{N} e_{n+1} + r_{\text{penalty}} & s_N \in S_{\text{end}} 
\end{cases}$$

where $\alpha$ is the trajectory length of the vehicle along the course and $e_n$ is the distance error to the center of the course at state $s_n$. We choose $\alpha = 0.5$ and choose a large negative reward $r_{\text{penalty}} = -15000$ for infractions (e.g. crossing the road boundary).

For UAV racing, it is not necessary to stay in the center of the track. We use the following reward function that considers forward speed $v_f$ and $S_{\text{end}}$:

$$R(\tau) = \begin{cases} 
\sum_{n=1}^{N} v_{f_n} & s_N \notin S_{\text{end}} \\
\sum_{n=1}^{N} v_{f_n} + r_{\text{penalty}} & s_N \in S_{\text{end}} 
\end{cases}$$

where $v_{f_n}$ is the forward speed of the vehicle after executing $a_n$ at state $s_n$ and $r_{\text{penalty}}$ is a large negative reward. We choose $r_{\text{penalty}} = -15000$.

Reward function of DDPG. In our experiments, DDPG cannot converge using the same simple reward function as OIL. Therefore, we design a more complex reward function to induce a dense normalized reward as follows.

$$r(s_n, a_n) = \begin{cases} 
\frac{e_{n} - e_{n+1}}{e_{n} + 1} & s_{n+1} \notin S_{\text{end}} \\
\frac{1}{e_{n+1} + 1} + \beta \frac{v_{f_{n+1}}}{v_{\text{norm}}} & s_{n+1} \in S_{\text{end}} 
\end{cases}$$

where $e_n$ and $e_{n+1}$ are the distance errors to the center of the track at state $s_n$ and state $s_{n+1}$ respectively, $e_{\text{norm}}$ is a distance error normalization factor, $v_{f_n}$ is the forward speed the vehicle, $v_{\text{norm}}$ is a velocity normalization factor and $r_{\text{penalty}}$ is a negative reward for when vehicles reaches a terminal state. Furthermore, we find it is hard to learn both gas and steering controls for the car. Therefore, we fix the gas value to $E = 0.5$ and choose $\beta = 0$ to only score the agent by the distance error. For the UAV, we choose $\beta = 1$ to score the agent by both the velocity and distance error and choose $v_{\text{norm}} = 800$. We set $e_{\text{norm}} = 15$ and $r_{\text{penalty}} = -0.2$ for both the car and the UAV experiments.

Controller network architecture of learned baselines. For fair comparisons, we use the same control policy network with hidden units $\{64, 32, 16\}$ for all the learned baselines. As for the critic network of DDPG, we use the same number of hidden units and use an output layer with a one dimensional output to predict the Q value.
Table 4: Regular Baselines - CAR.

| PID | Human | teacher1 | teacher2 | teacher3 | teacher4 | teacher5 | novice | intermediate | expert |
|-----|-------|----------|----------|----------|----------|----------|--------|-------------|--------|
|     |       | error    | time     | error    | time     | error    | time    | error       | time   |
| Track1 | 22.26 | 131.59  | 68.19    | 117.89  | 13.61    | 84.30    | 102.21  | 83.37       | 664.79 |
| Track2 | 25.24 | 140.59  | 371.56   | 89.40   | 15.57    | 82.98    | 42.49   | 58.97       | 830.63 |
| Track3 | 26.17 | 161.57  | 484.35   | 120.36  | 26.91    | 158.95   | 120.36  | 19.74       | 147.94 |
| Avg.  | 24.14 | 151.39  | 539.55   | 109.95  | 19.53    | 84.45    | 76.70   | 76.68       | 568.36 |

Table 5: Learned Baselines - CAR.

| Trajectory Length | Number of Teachers | Behaviour Cloning | Dagger | DDPG | OIL |
|-------------------|-------------------|-------------------|--------|------|-----|
| N=60              | N=120             | best              | time   | error| time |
| Track1            | 11.07             | 87.92             | 411.11 | 127.77| 74.35 |
| Track2            | 15.67             | 85.38             | 367.69 | 110.20| 70.80 |
| Track3            | 11.97             | 86.63             | 325.79 | 115.42| 74.15 |
| Track4            | 15.22             | 94.48             | 449.74 | 97.37 | 73.53 |
| Avg.              | 13.85             | 88.60             | 388.58 | 112.49| 73.85 |

Table 6: Ablation Study - CAR. The hyperparameter selection for our final model is highlighted in bold.

| PID | Human | teacher1 | teacher2 | teacher3 | teacher4 | teacher5 | novice | intermediate | expert |
|-----|-------|----------|----------|----------|----------|----------|--------|-------------|--------|
|     |       | error    | time     | error    | time     | error    | time    | error       | time   |
| Track1 | 12/12 | 130.76  | 12/12    | 40.04    | 12/12    | 40.04    | 12/12  | 83.37       | 12/12  |
| Track2 | 20/20 | 136.19  | 17/20    | 77.41    | 20/20    | 81.08    | 12/12  | 78.13       | 20/20  |
| Track3 | 22/22 | 121.54  | 11/22    | 149.45   | 22/22    | 77.77    | 16/22  | 78.13       | 22/22  |
| Track4 | 18/18 | 139.09  | 15/18    | 81.08    | 16/18    | 104.99   | 12/18  | 78.13       | 18/18  |
| Avg.  | 100%  | 131.90  | 76.38    | 87.00    | 97.22%   | 87.57    | 80.56% | 90.29       | 97.22% |

Table 7: Regular Baselines - UAV.

| PID | Human | Behaviour Cloning | Dagger | DDPG | OIL |
|-----|-------|-------------------|--------|------|-----|
|     |       | score              | time   | score| time |
| Track1 | 12/12 | 130.76             | 12/12  | 40.04| 12/12 |
| Track2 | 18/20 | 152.04             | 15/20  | 109.19| 109.19 |
| Track3 | 22/22 | 115.32             | 10/18  | 126.45| 126.45 |
| Track4 | 16/18 | 139.09             | 15/18  | 81.08| 104.99 |
| Avg.  | 100%  | 139.37             | 72.22% | 101.57| 103.45 |

Table 8: Learned Baselines - UAV.