Teaching UAVs to Race Using UE4Sim*

Matthias Mueller, Vincent Casser, Neil Smith, Dominik L. Michels, Bernard Ghanem
Visual Computing Center, KAUST, Thuwal, Saudi Arabia
{matthias.mueller.2, vincent.casser, neil.smith, dominik.michels, bernard.ghanem}@kaust.edu.sa

Abstract

Automating the navigation of unmanned aerial vehicles (UAVs) in diverse scenarios has gained much attention in the recent years. However, teaching UAVs to fly in challenging environments remains an unsolved problem, mainly due to the lack of data for training. In this paper, we develop a photo-realistic simulator that can afford the generation of large amounts of training data (both images rendered from the UAV camera and its controls) to teach a UAV to autonomously race through challenging tracks. We train a deep neural network to predict UAV controls from raw image data for the task of autonomous UAV racing. Training is done through imitation learning enabled by data augmentation to allow for the correction of navigation mistakes. Extensive experiments demonstrate that our trained network (when sufficient data augmentation is used) outperforms state-of-the-art methods and flies more consistently than many human pilots.

1. Introduction

Unmanned aerial vehicles (UAVs) like drones and multicopters are attracting more attention in the graphics community. This development is stimulated by the merging of researchers from robotics, graphics, and computer vision to a common scientific community. Recent UAV-related contributions in computer graphics cover a wide spectrum from computational multicopter design, optimization and fabrication [6] to state-of-the-art video capturing using quadrotor-based camera systems [15] and the generation of dynamically feasible trajectories [38]. While UAV design and point-to-point stabilized flight navigation is becoming a solved problem (as is evident from recent advances in UAV technology from industry leaders such as DJI, Amazon, and Intel), autonomous navigation of UAVs in more complex and real-world scenarios, such as unknown congested environments, GPS-denied areas, through narrow spaces, or around obstacles, is still far from being solved. In fact, only human pilots can reliably maneuver in these environments. This is a complex problem, since it requires *both* the sensing of real-world conditions and the understanding of appropriate policies of response to perceive obstacles through optimal navigation trajectory adjustment. There is perhaps no area where human pilots are more required to control UAVs then in the emerging sport of UAV racing where all these complex sense-and-understand tasks are conducted at neck-break speeds of over 100 km/h. Learning to control racing UAVs is a challenging task even for humans. It takes hours of practice and quite often hundreds of crashes. A more affordable approach to develop professional flight skills is to first train many hours on a flight simulator before going to the field. Since most of the fine motor skills of flight control are developed in simulators, the pilot is able to quickly transition to real-world flights.

In this contribution, we capitalize on this insight from how human pilots learn to sense and react with appropriate controls to their environment to train a deep network that can fly racing UAVs through challenging racing courses, many of which test the capabilities of even professional pilots. Inspired by recent work that trains artificial intelligence (AI) systems through the use of computer games [5], we create a photo-realistic UAV racing game with accurate physics using the Unreal Engine 4 (UE4) and integrate it with UE4Sim [27]. As this is the core learning environment, we develop a photo-realistic and customizable racing area in the form of a stadium based on a three-dimensional (3D) scanned real-world location to minimize discrepancy incurred from transitioning from the simulated to a real-world scenario. Inspired by recent work on self-driving cars [3], our automated racing UAV approach goes beyond simple pattern detection by learning the full control system required to fly a UAV through a racing course (arguably much more complicated than driving a car). As such, the proposed network extends the complexity of previous work to the control of a six degrees of freedom (6-DoF) UAV flying system, enabling the UAV to traverse tight spaces and make sharp turns at very high speeds (a task that cannot be performed by a ground vehicle). Our imitation learn-

*www.airsim.org

Figure 1: Illustration of the trained racing UAV in-flight.

ing based approach simultaneously addresses both problems of perception and policy selection as the UAV navigates through the course, after it is trained from human pilot input on how to control itself (exploitation) and how to correct itself in case of drift (exploration). Our developed simulator is multi-purpose, as it enables the evaluation of a trained network in real-time on racing courses it has not encountered before.

Contributions. Our specific contributions are as follows.

(1) We are the first to introduce a photo-realistic simulator that is based on a real-world 3D environment that can be easily customized to build increasingly challenging racing courses, enables realistic UAV physical behavior, and is integrated with a real-world UAV controller (powered by a human pilot or a synthetic one). Logging video data from the UAV point-of-view and pilot controls is seamless and can be used to effortlessly generate large-scale training data for AI systems targeting UAV flying in particular and self-driving vehicles in general (e.g. self-driving cars).

(2) To facilitate the training, parameter tuning, and evaluation of deep networks on this type of simulated data, we provide a full integration between the simulator and an end-to-end deep learning pipeline (based on TensorFlow) to be made publicly available to the community. Similar to other deep networks trained for game play, our integration will allow the community to fully explore many scenarios and tasks that go far beyond UAV racing in a rich and diverse photo-realistic gaming environment (e.g. obstacle avoidance and path planning).

(3) To the best of our knowledge, this paper is the first which fully demonstrates 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 inexperienced pilots, who can use our system in a learning game play mode to become better pilots.

2. Related Work

In this section, we put our proposed methodology into context, focusing on the most related previous work as follows.

Learning to Navigate. The task of training an actor (e.g. vehicle, animal, human, or UAV) to physically navigate through an unknown environment has traditionally been approached either through supervised learning (SL) [3,4,18,29,35,39,41], reinforcement learning (RL) [22,24, 25, 32, 33, 42], or a combination of the two [1, 5, 9, 10, 21]. A key challenge is learning a high dimensional order of raw sensory input within a complex 3D environment. For the particular case of physics-driven vehicles, SL can be advantageous when high dimensional feedback can be recorded and it has proven to have relative success in the last several years in the field of autonomous driving [3,4,41]. However, the use of neural networks for SL in autonomous driving goes back much earlier to work [29, 35].

In the work of Bojarski et al. [3], a deep neural network (DNN) is trained to map recorded camera views to 3-DoF steering commands (steering wheel angle, throttle, and brake). Seventy-two hours of human driven training data was tediously collected from a forward facing camera and augmented with two additional views to provide data for simulated drifting and corrective maneuvering. The sim-

ulated and on-road results of this pioneering work demonstrate the ability of a DNN to learn (end-to-end) the control process of a self-driving car from raw video data. Similar to our work but for car vehicles, the work of Chen et al. [4] uses TORCS (The Open Racing Car Simulator) [44] to train a DNN to drive at casual speeds through a course and properly pass or follow other vehicles in its lane. This work builds off earlier work using TORCS, which focused on keeping the car on a track [18]. Similar to the work of Bojarski et al. [3] and in contrast to our work, the vehicle controls to be predicted in the work of Chen et al. [4] are limited, since only a small discrete set of expected control outputs are available: turn-left, turn-right, throttle, and brake. Recently, TORCS has also been successfully used in several RL approaches for autonomous car driving [19, 22, 24]; however, in these cases, RL was used to teach the agent to drive specific tracks or all available tracks rather than learning to drive never before seen tracks.

In the work of Smolyanskiy et al. [41], a DNN is trained (in an SL fashion and from real data captured from a head-mounted camera) to navigate a UAV through forest trails and avoid obstacles. Similar to previous work, the expected control outputs of the network are discrete and very limited (simple yaw movements): turn-left, go-straight, or turn-right. Despite showing relatively promising results, the trained network leads to a slow, non-smooth (zig-zag) trajectory at a fixed altitude above the ground. It is worthwhile to note that indoor UAV control using DNNs has also been recently explored [1, 17, 40].

Importance of Exploration in Supervised Learning. In imitation learning [14], the 'expert' training set used for SL is augmented and expanded, so as to combine the merits of both exploitation and exploration. In many sequential decision making tasks of which autonomous vehicle control is one, this augmentation becomes necessary to train an AI system (e.g. DNN) that can recover from mistakes. In this sense, imitation learning can be crudely seen as a supervision guided form of RL. For example, a recent imitation learning method called DAgger (Dataset Aggregation) [39] demonstrated a simple way of incrementally augmenting ground-truth sequential decisions to allow for further exploration, since the learner will be trained on the aggregate dataset and not only the original expert one. This method was shown to outperform state-of-the-art AI methods on a 3D car racing game (Super Tux Kart), where the control outputs are again 3-DoF. Other imitation learning approaches [21] have reached a similar conclusion, namely that a trajectory optimizer can function to help guide a sub-optimal learning policy towards the optimal one. Inspired by the above work, our proposed method also exploits concepts from imitation learning, so as to automatically and effortlessly (through our simulator) generate a richly diverse set of image and control pairs that can be used to train a UAV to robustly and reliably navigate through a racing course.

Simulation. As mentioned earlier, generating diverse 'natural' training data for sequential decision making through SL is tedious and generating such data for exploration purposes using imitation learning or RL (i.e. in scenarios where both input and output pairs have to be generated) is much more so. Therefore, much more attention from the community is being given to simulators (or games) for this source of data. In fact, a broad range of work has exploited them recently for these types of learning, namely in animation and motion planning [10–12, 16, 20, 22, 42], scene understanding [2, 31], pedestrian detection [23], and identification of 2D/3D objects [13, 26, 34]. For example, a physics-based computer game environment (Unity) has been used to teach a simulated bird to fly [16]. Moreover, Hardware in the loop (HILT) simulation has also been used in robotics to develop and evaluate controllers and for visual servoing studies (e.g. JMAVSim [36, 43] and RotorS [7]). The visual rendering in these simulators is often primitive and relies on off-the-shelf simulators (e.g. Realflight, Flightgear, or XPlane). They do not support advanced shading and post-processing techniques, are limited in terms of available assets and textures, and do not support motion capture (MOCAP) or key-frame type animation to simulate natural movement of actors or vehicles. Recent work [8, 37] shows the advantages of exploiting the photo-realism of modern game engines to generate training datasets and pixel-accurate segmentation masks. Since the goal of this work is to build an automated UAV flying system (based on imitation learning) that can relatively easily be transitioned from a simulated world to the real one, we base our own simulator on the recent work of Mueller et al. [28], which provides full hardware and software in-the-loop UAV simulations built on top of the open source game engine UE4. This simulator was designed to generate photo-realistic simulated ground-truth for specific computer vision tasks like UAV tracking.

3. Overview

The fundamental modules of our proposed system are summarized in Figure 2, which represents the end-to-end dataset generation, learning, and evaluation process. In what follows, we provide details for each of these modules, namely how datasets are automatically generated within the simulator, how our proposed DNN is designed and trained, and how the learned DNN is evaluated. Note that this generic architecture can also be applied to any other type of vision-based navigation task made possible using our simulator.

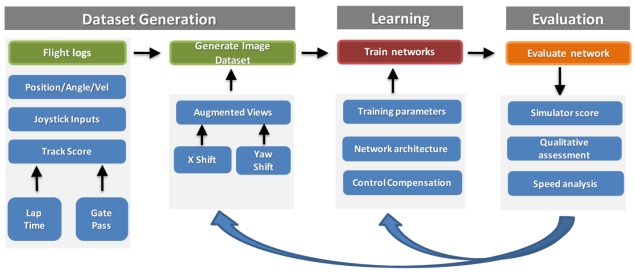

Figure 2: Description of the pipeline of our DNN Imitation Learning System. After recording flights of human pilots, we improve important model parameters like network architecture, number of augmented views and appropriate control compensation for them in an iterative process.

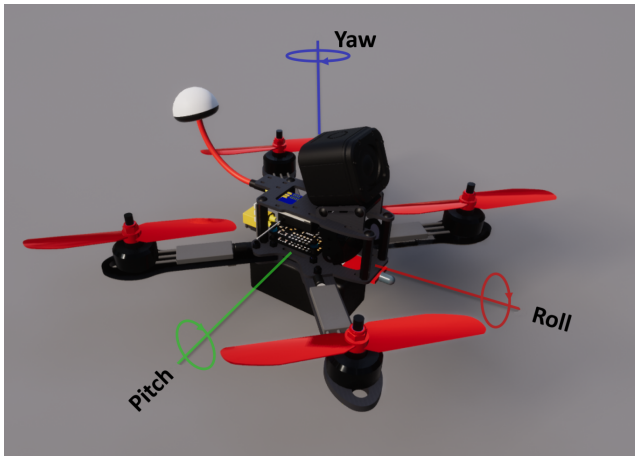

Figure 3: The 3D model of the racing UAV modeled in the simulator, based on a well known 250 class design known within the racing community as the *Hornet*.

4. Simulation Centric Dataset Generation

Our simulation environment allows for the automatic generation of customizable datasets, which comprise rich expert data to robustly train a DNN through imitation learning.

UAV Flight Simulation and Real-World Creation. The core of the system is the application of our UE4 based simulator. It is built on top of the open source UE4 project for computer vision called UAVSim [28]. Several changes were made to adapt the simulator for training our proposed racing DNN. First, we replaced the UAV with the 3D model and specifications of a racing quadcopter (see Figure3). We retuned the PID controller of the UAV to be more responsive and to function in a racing mode, where altitude control and stablization are still enabled but with much higher rates and steeper pitch and roll angles. In fact, this is now a popular racing mode available on consumer UAVs, such as the DJI Mavic. The simulator frame rate is locked at 60 fps and at every frame a log is recorded with UAV position, orientation, velocity, and stick inputs from the pilot. To accommodate for realistic input, we integrated the same UAV transmitter that would be used in real-world racing scenarios. We refer to the **supplementary material** for an example pilot recording. Following paradigms set by UAV racing norms, each racing course/track in our simulator comprises a sequence of gates connected by uniformly spaced cones. The track has a timing system that records time between each gate, lap, and completion time of the race. The gates have their own logic to detect whether the UAV has passed through the gate in the correct direction. This allows us to trigger both the start and ending of the race, as well as, determine the number of gates traversed by the UAV. These metrics (time and percentage of gates passed) constitute the overall per-track performance of a pilot, albeit human or a DNN.

Many professional pilots compete in time trials of well-known tracks such as those posted by the MultiGP Drone Racing League. Following this paradigm, our simulator race course is modeled after a football stadium, where local professional pilots regularly setup MultiGP tracks. Using a combination of LiDAR scanning and aerial photogrammetry, we captured the stadium with an accuracy of 0.5 cm; see Figure 5. A team of architects used the dense point cloud and textured mesh to create an accurate solid model with physics based rendering (PBR) textures in 3DSmax for export to UE4. This manifested in a geometrically accurate and photo-realistic race course that remains low in poly count, so as to run in UE4 at 60 fps, in which all training and evaluation experiments are conducted. We refer to Figure 4 for a side-by-side comparison of the real and virtual stadiums. Moreover, we want the simulated race course to be as dynamic and photo-realistic as possible, since the eventual goal is to transition the trained DNN from the simulated environment to the real-world, particularly starting with a similar venue as learned within the simulator. The concept of generating synthetic clones of real-world data for deep learning purposes has been adopted in previous work [8]. A key requirement for relatively straightforward simulated-to-real world transition is the DNN's ability to learn to automatically detect the gates and cones in the track within a complexly textured and dynamic environment. To this end, we enrich the simulated environment and race track with customizable textures (e.g. grass, snow, and dirt), gates (different shapes and appearance), and lighting.

Automatic Track Generation. We developed a track editor, where a user draws a 2D sketch of the overhead view of the track, the 3D track is automatically generated accordingly, and it is integrated into the timing system. With this editor, we created eleven tracks: seven for training,

Figure 4: *left:* Aerial image captured from an UAV hovering above the stadium racing track. *right:* Rendering of the reconstructed stadium generated at a similar altitude and viewing angle within the simulator.

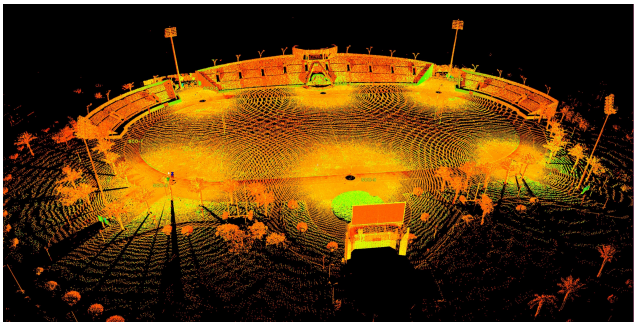

Figure 5: The point cloud from the LiDAR scan of the stadium collected from six locations.

and four for testing and evaluation. Each track is defined by gate positions and track lanes delineated by uniformly spaced racing cones distributed along the splines connecting adjacent gates. To avoid user bias in designing the race tracks, we use images collected from the internet and trace their contours in the editor to create uniquely stylized tracks. Following trends in designing tracks popularly used in the UAV racing community, both training and testing tracks have a large variety of types of turns and strait lengths. From a learning point of view, this track diversity exposes the DNN to a large number of track variations, as well as, their corresponding navigation controls. Obviously, the testing/evaluation tracks are never seen in training, neither by the human pilot nor the DNN.

Acquiring Large-Scale Ground-Truth Pilot Data. We record human pilot input from a Taranis flight transmitter integrated into the simulator through a joystick. This input is solicited from three pilots with different levels of skill: novice (has never flown before), intermediate (a moderately experienced pilot), and expert (a professional racing pilot). 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. For the evaluation tracks, the pilots are allowed to fly the course only as many times needed to complete the entire course without crashing. We automatically score pilot performance based on lap time and percentage of gates traversed.

The simulation environment allows us to log the images rendered from the UAV camera point-of-view and the UAV flight controls from the transmitter. As mentioned earlier and to enable exploration, robust imitation learning requires the augmentation of these ground-truth logs with synthetic ones generated at a user-defined set of UAV offset positions and orientations accompanied by the corresponding controls needed to correct for these offsets. Also, since the logs can be replayed at a later time in the simulator, we can augment the dataset further by changing environmental conditions, including lighting, cone spacing or appearance, and other environmental dynamics (e.g. clouds). Therefore, each pilot flight leads to a large number of image-control pairs (both original and augmented) that will be used to train the UAV to robustly recover from possible drift along each training track, as well as, unseen evaluation tracks. Details of how our proposed DNN architecture is designed and trained are provided in Section 5. In general, more augmented data should improve UAV flight performance assuming that the control mapping and original flight data are noise-free. However, in many scenarios, this is not the case, so we find that there is a limit after which augmentation does not help (or even degrades) explorative learning. Empirical results validating this observation are detailed in Section 6.

Note that assigning corrective controls to the augmented data is quite complex in general, since they depend on many factors, including current UAV velocity, relative position on the track, its weight and current attitude. While it is possible to get this data in the simulation, it is very difficult to obtain it in the real-world in real-time. Therefore, we employ a fairly simple but effective model to determine these augmented controls that also scales to real-world settings. We add or subtract a corrective value to the pilot roll and yaw stick inputs for each position or orientation offset that is applied. For rotational offsets, we do not only apply a yaw correction but also couple it to roll because the UAV is in motion while rotating causing it to wash out due to its inertia.

DNN Interface for Real-Time Evaluation. To evaluate the performance of a trained DNN in real-time at 60 fps, we establish a TCP socket connection between the UE4 simulator and the Python wrapper (TensorFlow) executing the DNN. In doing so, the simulator continuously sends rendered UAV camera images across TCP to the DNN, which in turn processes each image individually to predict the next UAV stick inputs (flight controls) that are fed back to the UAV in the simulator using the same connection. Another advantage of this TCP connection is that the DNN prediction can be run on a separate system than the one running the simulator. We expect that this versatile and multi-purpose interface between the simulator and DNN framework will enable opportunities for the research community to further develop DNN solutions to not only the task of automated UAV navigation (using imitation learning) but to the more general task of vehicle maneuvering and obstacle avoidance (possibly using other forms of learning including RL).

5. Learning

In this section, we provide a detailed description of the learning strategy used to train our DNN, its network architecture and design. We also explore some of the inner workings of one of these trained DNNs to shed light on how this network is solving the problem of automated UAV racing.

5.1. Dataset Preparation and Augmentation

As it is the case for DNN-based solutions to other tasks, a careful construction of the training set is a key requirement to robust and effective DNN training. To this end and as mentioned earlier, we dedicate seven racing tracks (with their corresponding image-control pairs logged from human pilot runs in our simulator) for training and four tracks for testing/evaluation. We design the tracks such that they are similar to what racing professionals are accustomed to and such that they offer enough diversity and capability for exploration for proper network generalization on the unseen tracks. Figure 6 illustrates an overhead view of all these tracks.

As mentioned in Section 4, we log the pilot flight inputs and all other necessary parameters so that we can accurately replay the flights. These log files are then augmented using the specified offsets, replayed within the simulator while saving all rendered images and thus, providing exploratory insights to the racing DNN. For completeness, we summarize the details of the data generated for each training/testing track in Table 1. It is clear that the augmentation increases the size of the original dataset by approximately seven times. In Section 6, we show the effect of changing the amount of augmentation on the UAV's ability to generalize well. Moreover, for this dataset, we choose to use the intermediate pilot only so as to strike a trade-off between style of flight and overall size of the dataset. In Section 6.3, we show the effects of training with different flying styles.

track	type	time (sec)	original	total
track01	train	69.8	4.2K	29.3K
track02	train	100.4	6.0K	42.2K
track03	train	83.1	5.0K	35.0K
track04	train	97.7	5.9K	41.0K
track05	train	99.8	6.0K	42.0K
track06	train	115.4	6.9K	48.5K
track07	train	98.3	5.9K	41.2K
total	train	664.5	39.9K	279.1K

Table 1: Overview of the image-control dataset generated from two laps of flying (by the intermediate pilot) through each of the training tracks. The 'time' column shows the total time taken by the pilot to successfully fly two laps through the track (i.e. passing through all the gates). We also record the number of images rendered from the pilot's trajectory in the simulator, along with the total number of images used for training when data augmentation is applied. For this augmentation, we use the following default settings: roll offset ($\pm$50cm), yaw offset ($\pm 15^\circ$ and $\pm 30^\circ$).

5.2. Network Architecture and Implementation Details

To train a DNN to predict stick controls to the UAV from images, we choose a regression network architecture similar in spirit to the one used by Bojarski et al. [3]; 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 Figure7. The network consists of eight layers, five convolutional and three fully-connected. Since we implicitly want to localize the track and gates, we use striding in the convolutional layers instead of (max) pooling, which would add some degree of translation invariance. The DNN is given a single RGB-image with a 320×180 pixel resolution as input and is trained to regress to the four control/stick inputs to the UAV using a standard L^2-loss and dropout ratio of 0.5. We find that the relatively high input resolution (i.e. higher network capacity), as compared to related methods [3, 41], is useful to learn this more complicated maneuvering task and to enhance the network's ability to look further ahead. This affords the network with more robustness needed for long-term trajectory stability. We arrived to this compact network architecture by running extensive validation experiments that strikes a reasonable tradeoff between computational complexity and predictive performance. This careful design makes the proposed DNN architecture feasible for real-time applications on embedded hardware (e.g. Nvidia TX1) unlike previous architectures [3], if they use the same input size. In Table 2, we show both evaluation time on and technical details of

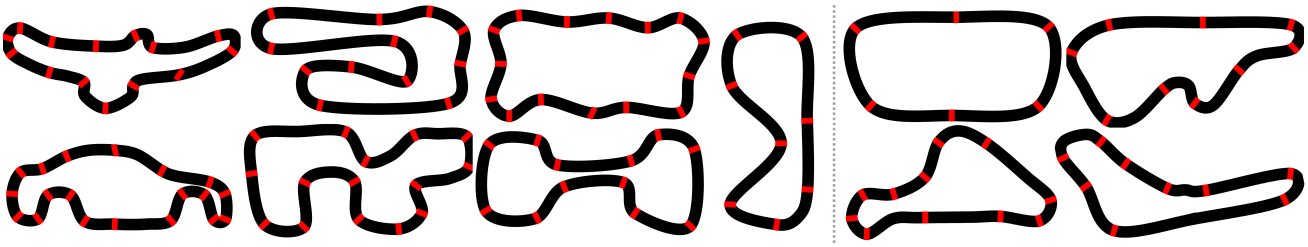

Figure 6: The seven training tracks (left) and the four evaluation tracks (right). Gates are marked in red.

the NVIDIA Titan X, and how it compares to a NVIDIA TX-1. Based on [30], we expect our network to still run at real-time speed with over 60 frames per second on this embedded hardware.

For training, we exploit a standard stochastic gradient descent (SGD) optimization strategy (namely Adam) using the TensorFlow platform. As such, one instance of our DNN can be trained to convergence on our dataset in less than two hours on a single GPU. This relatively fast training time enables finer hyper-parameter tuning.

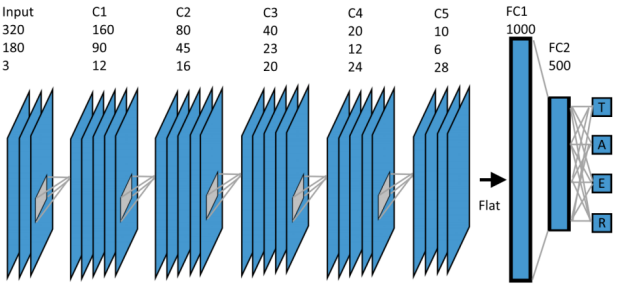

Figure 7: Our network architecture is taking an image of shape 320x180 and regresses to the control outputs throttle (T), elevator (E), aileron (A) and roll (R).

	NVIDIA Titan X	NVIDIA TX-1
CUDA cores	3,840	256
Boost Clock MHz	1,582	998
VRAM	12 GB	4 GB
Memory Bandwidth	547.7 Gbps	25.6 Gbps
Evaluation (ours)	556 fps (ref)	64.6 fps

Table 2: Comparison of the NVIDIA Titan X and the NVIDIA TX-1. The performance of the TX-1 is approximated according to [30].

In contrast to other work where the frame rate is sampled down to 10 fps or lower [3,4,41], our racing environment is highly dynamic (with tight turns, high speed, and low inertia of the UAV), so we use a frame rate of 60 fps. This allows the UAV to be very responsive and move at high speeds, while maintaining a level of smoothness in controls. An alternative approach for temporally smooth controls is to include historic data in the training process (e.g. add the previous controls as input to the DNN). This can make the network more complex, harder to train, and less responsive in the highly dynamic racing environment, where many time critical decisions have to be made within a couple of frames (about 30 ms). Therefore, we find the high learning frame rate of 60 fps a good tradeoff between smooth controls and responsiveness.

Reinforcement vs. Imitation Learning. Of course, our simulator can lend itself useful in training networks using reinforcement learning. This type of learning does not specifically require supervised pilot information, as it searches for an optimal policy that leads to the highest eventual reward (e.g. highest percentage of gates traversed or lowest lap time). Recent methods have made use of reinforcement to learn simpler tasks without supervision [5]; however, they require weeks of training and a much faster simulator (1,000fps is possible in simple non photo-realistic games). For UAV racing, the required task is much more complicated and since the intent is to transfer the learned network into the real-world, a (slower) photo-realistic simulator is mandatory. Because of these two constraints, we decided to train our DNN using imitation learning instead of reinforcement learning.

5.3. Network Visualization

After training our DNN to convergence, we visualize how parts of the network behave in order to get additional insights. Figure8 shows some feature maps in different layers of the trained DNN for the same input image. Note how the filters have automatically learned to extract all necessary information in the scene (i.e. gates and cones), while in higher-level layers they are not responding to other parts of the environment. Although the feature map resolution becomes very low in the higher DNN layers, the feature map in the fifth convolutional layer is interesting as it marks the top, left, and right of parts of a gate with just a single activation each. This clearly demonstrates that our DNN is learning semantically intuitive features for the task of UAV racing.

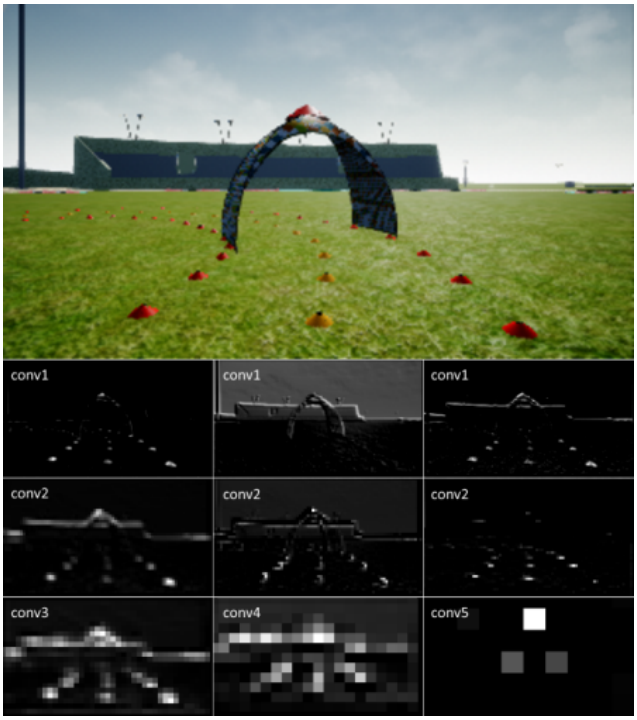

Figure 8: Visualization of feature maps at different convolutional layers in our trained network. Notice how the network activates in locations of semantic meaning for the task of UAV racing, namely the gates and cones.

6. Evaluation

In order to evaluate the performance of our DNN, we create four testing tracks based on well-known race tracks found in TORCS and Gran Turismo. We refer to Figure 6 for an overhead view of these tracks). Since the tracks must fit within the football stadium environment, they are scaled down leading to much sharper turns and shorter straightaways with the UAV reaching top speeds of over 100 km/h. Therefore, the evaluation tracks are significantly more difficult than they may have been originally intended in their original racing environments. We rank the four tracks in terms of difficulty ranging from easy (track 1), medium (track 2), hard (track 3), to very hard (track 4). For all the following evaluations, both the trained networks and human pilots are tasked to fly two laps in the testing tracks and are scored based on the total gates they fly through and overall lap time.

6.1. Effects of Exploration

We find exploration to be the predominant factor influencing network performance. As mentioned earlier, we augment the pilot flight data with offsets and corresponding corrective controls. We conduct grid search to find a suitable degree of augmentation and to analyze the effect it has on overall UAV racing performance. To do this, we define two sets of offset parameters: one that acts as a horizontal offset (roll-offset) and one that acts as a rotational offset (yaw-offset). Figure9 shows how the racing accuracy (percentage of gates traversed) varies with different sets of these augmentation offsets across the four testing tracks. It is clear that increasing the number of rendered images with yaw-offset has the greatest impact on performance. While it is possible for the DNN to complete tracks without being trained on roll-offsets, this is not the case for yaw-offsets. However, the huge gain in adding rotated camera views saturates quickly, and at a certain point the network does not benefit from more extensive augmentation. Therefore, we found four yaw-offsets to be sufficient. Including camera views with horizontal shifts is also beneficial, since the network is better equipped to recover once it is about to leave the track on straights. We found two roll-offsets to be sufficient to ensure this. Therefore, in the rest of our experiments, we use the following augmentation setup in training: horizontal roll-offset set $\{-50°, 50°\}$ and rotational yaw-offset set $\{-30°, -15°, 15°, 30°\}$.

[-75:25:75]	6	0.17	0.45	1.00	1.00	1.00	1.00	0.83	0.85	0.92	1.00
		0.82	0.50	0.95	1.00	1.00	1.00	0.95	1.00	1.00	1.00
[-75:50:75]	4	0.42	0.60	1.00	1.00	1.00	1.00	0.75	1.00	1.00	0.85
		0.82	0.61	0.41	0.78	1.00	0.94	0.91	0.94	1.00	1.00
[-50:50:50]	2	0.17	0.35	0.92	1.00	1.00	1.00	1.00	1.00	1.00	1.00
		0.23	0.28	1.00	1.00	1.00	1.00	1.00	1.00	0.82	1.00
[None]	0	0.00	0.00	0.92	1.00	0.67	1.00	1.00	1.00	0.50	1.00
		0.00	0.00	0.55	0.78	0.73	1.00	0.77	0.89	0.91	0.89
roll offset [cm]	# of cams	0		2		4		6		12	
	yaw offset [°]	[None]		[-20:20:20]		[-30:15:30]		[-30:10:30]		[-30:5:30]	

Figure 9: Effect of data augmentation in training to overall UAV racing performance. By augmenting the original flight logs with data captured at more offsets (roll and yaw) from the original trajectory along with their corresponding corrective controls, our UAV DNN can learn to traverse almost all the gates of the testing tracks, since it has learned to correct for exploratory maneuvers. After a sufficient amount of augmentation, no additional benefit is realized in improved racing performance.

6.2. Comparison to State-of-the-Art

We compare our racing DNN to the two most related and recent network architectures, the first denoted as Nvidia (for self-driving cars [3]) and the second as MAV (for forest path navigating UAVs [41]). While the domains of these works

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. For fair comparison, we scale our dataset to the same input dimensionality and re-train each of the three networks. We then evaluate each of the trained models on the task of UAV racing in the testing tracks. It is noteworthy to point out that both the Nvidia and MAV networks (in their original implementation) use data augmentation as well, so when training, we maintain the same strategy. For the Nvidia network, the exact offset choices for training are not publicly known, so we use a rotational offset set of $\{-30°, 30°\}$ to augment its data. As for the MAV network, we use the same augmentation parameters proposed in the paper, i.e. a rotational offset of $\{-30°, 30°\}$. We needed to 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 would lead to inadequate UAV racing performance.

It should be noted that in the original implementation of the Nvidia network [3] (based on real-world driving data), it was realized that additional augmentation was needed for reasonable automatic driving performance *after* the real-world data was acquired. To avoid recapturing the data again, synthetic viewpoints (generated by interpolation) were used to augment the training dataset, which introduced undesirable distortions. By using our simulator, we are able to extract any number of camera views without distortions. Therefore, we wanted to also gauge the effect of *additional* augmentation to both the Nvidia and MAV networks, when they are trained using our default augmentation setting: horizontal roll-offset of $\{-50°, 50°\}$ and rotational yaw-offset of $\{-30°, -15°, 15°, 30°\}$. We denote these trained networks as Nvidia++ and MAV++.

Table 3 summarizes the results of these different network variants on the testing tracks. Results indicate that the performance of the original Nvidia and MAV networks suffer from insufficient data augmentation. They clearly do not make use of enough exploration. These networks improve in performance when our proposed data augmentation scheme (enabled by our simulator) is used. Regardless, our proposed DNN outperforms the Nvidia/Nvidia++ and MAV/MAV++ networks, where this improvement is less significant when more data augmentation or more exploratory behavior is learned. Unlike the other networks, our DNN performs consistently well on all the unseen tracks, owing to its sufficient network capacity needed to learn this complex task.

6.3. Pilot Diversity & Human vs. DNN

In this section, we investigate how the flying style of a pilot affects the network that is being learned. To do this, we compare the performance of the different networks on the testing set, when each of them is trained with flight data captured from pilots of varying flight expertise (intermediate, and expert). We also trained models using the Nvidia [3] and MAV [41] architectures, with and without our default data augmentation settings. Table 3 summarizes the lap time and accuracy of these networks. Clearly, the pilot flight style can significantly affect the performance of the learned network.

Pilot / Network	**Track 1**	**Track 2**	**Track 3**	**Track 4**
Human-Novice	1.00	1.00	0.95	0.94
Human-Intermediate	1.00	1.00	1.00	1.00
Human-Expert	1.00	1.00	1.00	1.00
Ours-Intermediate	**1.00**	**1.00**	**1.00**	**1.00**
Ours-Expert	1.00	0.95	0.91	0.78
Nvidia-Intermediate	0.17	1.00	0.82	0.83
Nvidia-Intermediate++	1.00	1.00	0.82	1.00
MAV-Intermediate	0.50	0.75	0.73	0.83
MAV-Intermediate++	0.42	1.00	0.91	0.78

Table 3: Accuracy score of different pilots and networks on the four test tracks. The accuracy score represents the percentage of completed racing gates. The networks ending with ++ are variants of the original network with our augmentation strategy.

Figure 10 shows that there is a high correlation regarding both performance and flying style of the pilot used in training and the corresponding learned network. The trained networks clearly resemble the flying style and also the proficiency of their human trainers. Thus, our network that was trained on flights of the intermediate pilot achieves high accuracies but is quite slow, just as the expert network sometimes misses gates but achieves very good lap and overall times. Interestingly, although the networks perform similar to their pilot, they fly more consistently, and therefore tend to outperform the human pilot with regards to overall time on multiple laps. This is especially true for our intermediate network. Both the intermediate and the expert network clearly outperforms the novice human pilot, who takes several hours of practice and several attempts to reach similar performance to the network. Even our expert pilots were not always able to complete the test tracks on the first attempt.

While the percentage of passed gates and best lap time give a good indication about the network performance, they do not convey any information about the style of the pilot. To this end, we visualize the performance of human pilots and the trained networks by plotting their trajectories onto the track (from a 2D overhead viewpoint). Moreover, we encode their speeds as a heatmap, where blue corresponds to the minimum speed and red to the maximum speed. Figure 11 shows a collection of heatmaps revealing several interesting insights. Firstly, the networks clearly imitate the

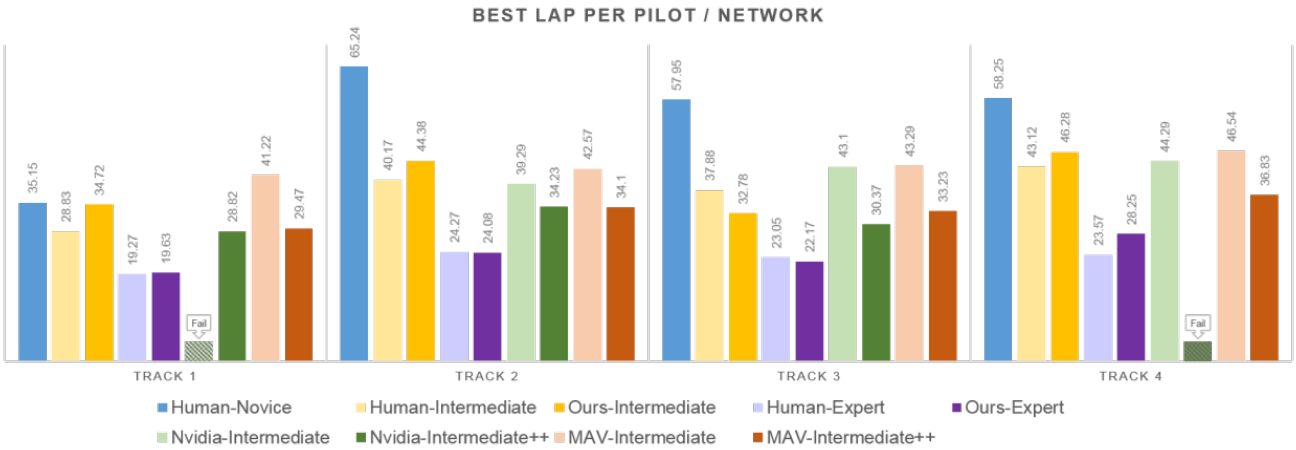

Figure 10: Best lap times of human pilots and networks trained on different flight styles. If there is no lap time displayed, the pilot was not able to complete the course because the UAV crashed. See text for a more detailed description.

style of the pilot they were trained on. This is especially true for the intermediate proficiency level, while the expert network sometimes overshoots, which causes it to loose speed and therefore to not match the speed pattern as well as the intermediate one. We also note that the performance gap between network and human increases as the expertise of the pilot increases. Note that the flight path of the expert network is less smooth and centered than its human correspondent and the intermediate network, respectively. This is partly due to the fact that the networks were only trained on two laps of flying across seven training tracks. An expert pilot has a lot more training than that and is therefore able to generalize much better to unseen environments. However, the experience advantage of the intermediate pilot over the network is much less and therefore the performance gap is smaller. We also show the performance of our novice pilot on these tracks. While the intermediate pilots accelerate on straights, the novice clearly is not able to control speed that well, creating a very narrow velocity range. Albeit flying quite slow, he also gets of track several times. This underlines how challenging UAV racing is, especially for unexperienced pilots.

7. Conclusions and Future Work

In this paper, we proposed a robust imitation learning based framework to teach an unmanned aerial vehicle (UAV) to fly through challenging racing tracks at very high speeds, an unprecedented and difficult task. To do this, we trained a deep neural network (DNN) to predict the necessary UAV controls from raw image data, grounded in a photo-realistic simulator that also allows for realistic UAV physics. Training is made possible by logging data (rendered images from the UAV and stick controls) from human pilot flights, while they maneuver the UAV through racing tracks. This data is augmented with sufficient offsets so as to teach the network to recover from flight mistakes. Extensive experiments demonstrate that our trained network (when sufficient data augmentation is used) outperforms state-of-the-art methods and flies more consistently than many human pilots.

In the future, we aim to transfer the network we trained in our simulator to the real-world 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 between simulator and 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] 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.

[2] 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.

[3] 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.

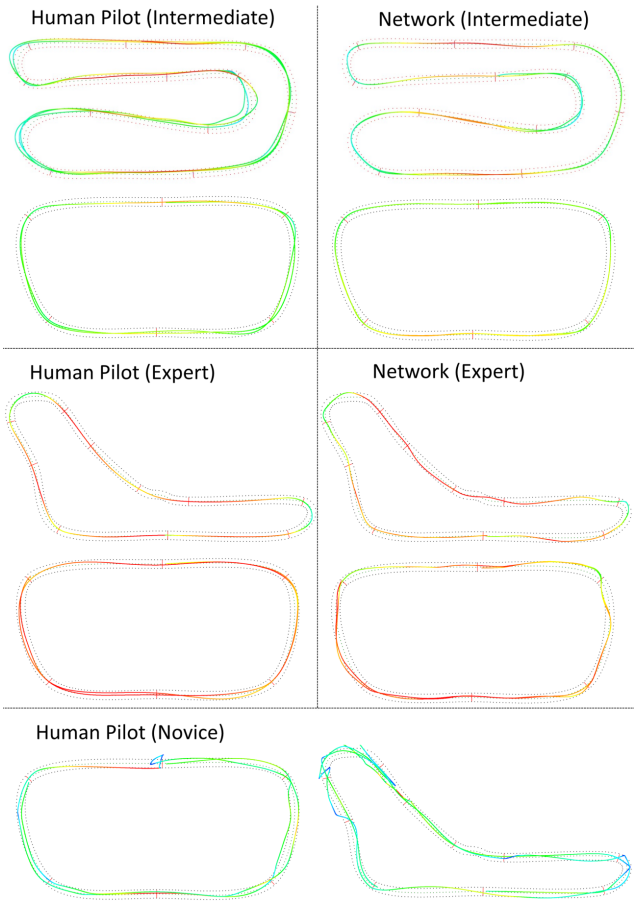

Figure 11: Visualization of human and automated UAV flights super-imposed onto a 2D overhead view of different tracks. The color coding illustrates the instantaneous speed of the UAV. Notice how the UAV learns to speedup on straights and to slow down at turns and how the flying style corresponds, especially with the intermediate pilots.

[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.

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

[6] T. Du, A. Schulz, B. Zhu, B. Bickel, and W. Matusik. Computational multicopter design. *ACM Trans. Graph.*, 35(6):227:1–227:10, Nov. 2016.

[7] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart. *RotorS—A modular gazebo MAV simulator framework*, volume 625 of *Studies in Computational Intelligence*. Springer, Cham, 2016.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne. Imitation learning: A survey of learning methods. *ACM Comput. Surv.*, 50(2):21:1–21:35, Apr. 2017.

[15] N. Joubert, M. Roberts, A. Truong, F. Berthouzoz, and P. Hanrahan. An interactive tool for designing quadrotor camera shots. *ACM Transactions on Graphics (SIGGRAPH Asia 2015)*, 2015.

[16] 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.

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

[18] 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.

[19] 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.

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

[21] 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.

[22] 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.

[23] J. Marín, D. Vázquez, D. Gerónimo, and A. M. López. Learning appearance in virtual scenarios for pedestrian detection. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 137–144, June 2010.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] M. Mueller, N. Smith, and B. Ghanem. *A Benchmark and Simulator for UAV Tracking*, pages 445–461. Springer International Publishing, Cham, 2016.

[29] 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.

[30] Nvidia. Gpu-based deep learning inference: A performance and power analysis. preprint on webpage at `math.rochester.edu/people/faculty/cohf`, November 2015.

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

[32] 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.

[33] 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.

[34] B. Pepik, M. Stark, P. Gehler, and B. Schiele. Teaching 3d geometry to deformable part models. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3362–3369, June 2012.

[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] Y. A. Prabowo, B. R. Trilaksono, and F. R. Triputra. Hardware in-the-loop simulation for visual servoing of fixed wing uav. In *Electrical Engineering and Informatics (ICEEI), 2015 International Conference on*, pages 247–252, Aug 2015.

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

[38] M. Roberts and P. Hanrahan. Generating dynamically feasible trajectories for quadrotor cameras. *ACM Transactions on Graphics (SIGGRAPH 2016)*, 35(4), 2016.

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

[40] 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.

[41] 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.

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

[43] B. R. Trilaksono, R. Triadhitama, W. Adiprawita, A. Wibowo, and A. Sreenatha. Hardware-in-the-loop simulation for visual target tracking of octorotor uav. *Aircraft Engineering and Aerospace Technology*, 83(6):407–419, 2011.

[44] B. Wymann, C. Dimitrakakis, A. Sumner, E. Espié, C. Guionneau, and R. Coulom. TORCS, the open racing car simulator. `http://www.torcs.org`, 2014.