Toward Exploring End-to-End Learning Algorithms for Autonomous Aerial Machines

Srivatsan Krishnan†, Behzad Boroujerdian‡, Aleksandra Faust∓, and Vijay Janapa Reddi†

† Harvard University, ‡The University of Texas at Austin, ∓ Robotics at Google
http://bit.ly/2WhhWYz

***Abstract*— We develop AirLearning, a tool suite for end-to-end closed-loop UAV analysis, equipped with a customized yet randomized environment generator in order to expose the UAV with a diverse set of challenges. We take Deep Q networks (DQN) as an example deep reinforcement learning algorithm and use curriculum learning to train a point to point obstacle avoidance policy. While we determine the best policy based on the success rate, we evaluate it under strict resource constraints on an embedded platform such as RasPi 3. Using hardware in the loop methodology, we quantify the policy's performance with quality of flight metrics such as energy consumed, endurance and the average length of the trajectory. We find that the trajectories produced on the embedded platform are very different from those predicted on the desktop, resulting in up to 26.43% longer trajectories. Quality of flight metrics with hardware in the loop characterizes those differences in simulation, thereby exposing how the choice of onboard compute contributes to shortening or widening of 'Sim2Real' gap.**

I. Introduction

Unmanned Aerial Vehicles (UAVs) have shown great promises in various robotics applications such as search and rescue [1], package delivery [2], [3], construction inspection [4], and others. Among these UAVs, rotor based platforms have been exceedingly deployed due to their robust mechanical design and favorable flight characteristics such as vertical take-off/landing and the high degree of agility/maneuverability [5].

To fully explore and exploit such UAVs there is a need to increase their level of autonomy, speed, and agility. In the context of autonomous navigation, end-to-end learning that includes deep reinforcement learning (DRL) is showing promising results in sensory-motor control in cars [6], indoor robots [7], as well as UAVs [8], [9]. Deep RL's ability to adapt and learn with minimum apriori knowledge makes them attractive for use as a controller in complex systems [10]. At the heart of these algorithms is a policy that is approximated by a neural network. Simply put, a policy takes sensor data (RGB/IMU) as input and predicts control actions as an output.

Despite the promises offered by reinforcement learning, there are several challenges in adopting reinforcement learning for UAV control. The challenge is that deep reinforcement learning algorithms are hungry for data. Collecting large amounts of data on real UAVs has logistical issues. For instance, most of the commercial and off the shelf (COTS) UAVs can only operate for less than 30 mins and collecting hundreds of millions of images takes about 1850 fights.[1] Also for effective learning, there is a need to include negative scenarios such as collisions which significantly increases the cost of collecting real data from UAVs [9]. In such scenarios, simulation provides a scalable and cost-effective approach for generating data for reinforcement learning.

The second challenge stems from the limited onboard energy, compute capability and power budget. Since UAVs are mobile machines, they need to be able to accomplish their tasks with the limited amount of energy on board. This problem exacerbates as the size decreases, reducing the total energy onboard while the same level of intelligence is required. For example, a nano-UAV such as a CrazyFlie [11] must have the same autonomous navigation capabilities comparing to its mini counterpart, e.g. DJI-Mavic Pro [12] while its onboard energy is $\frac{1}{15}$th. Hence onboard compute is a scarce resource and, reinforcement learning policies need to be carefully co-designed with underlying hardware so that it meets the real-time requirements under fixed power budget

The third challenge is the metrics that are used for the evaluation of deep reinforcement learning algorithms in context of UAVs. Since reinforcement learning policies are computationally intensive, evaluation metrics of the DRL algorithms need to be expanded beyond the traditional metrics used in OpenAI gym and arcade games. For instance, since energy is severely constrained, in real time, a policy can only be evaluated as long as there is enough battery. Hence the DRL algorithm designed for UAVs needs to include metrics such as energy and flight time to denote the quality of flight.

To address these challenges, we develop the AirLearning suite, which consists of a configurable environment generator with a wide range of knobs in order to enable various difficulty levels. These knobs tune the number of static and dynamic obstacles, their speed (if relevant), their texture and color, and etc. For quantifying the real-time performance of different reinforcement learning policies on resource-constrained compute platforms, we use the hardware-in-the-loop methodology where a policy is evaluated on embedded compute platforms that might potentially be the onboard compute of the UAVs. The tool suite also has metrics such as energy consumed, the average length of the trajectory and endurance so that different reinforcement learning policies can be evaluated on these metrics that quantify the quality of flight. The primary contributions of the AirLearning benchmarking suite are as follows:

- To assist in the dataset generation and generalization of learning algorithms for navigation tasks in Aerial robots, we develop a *configurable environment generator* with parameters to vary the number of static obstacles, arena size, type of objects in the environment, textures of the objects, the color of objects, number of dynamic obstacles and their velocity.
- Using closed loop *Hardware-in-the-Loop methodology*, we characterize the performance of policies on real em-

[1] Assuming flight time between recharge is 30 mins, and camera throughput is 30fps.

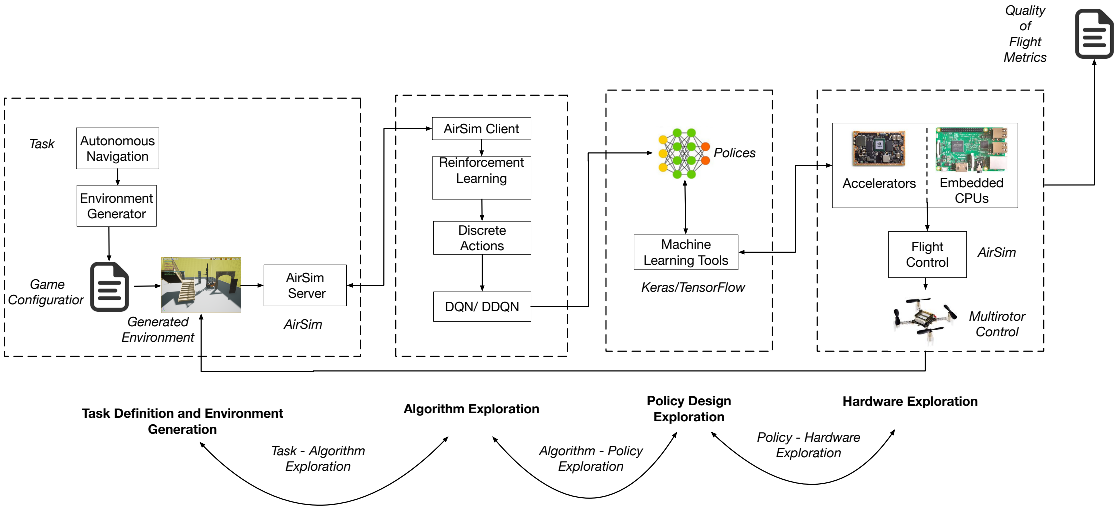


Fig. 1: Air Learning infrastructure and benchmarking suite for end-to-end learning in aerial autonomous machines.

bedded devices that are resource and power constrained.

- We provide *quality of flight metrics* such as energy consumed, endurance and average length of trajectory in the evaluation of learning algorithms specifically designed for aerial robots.

Using our tool, we show that the extent of 'Sim2Real' gap is not just limited to the fidelity of simulation environments but also extends to the choice of onboard compute platform for UAVs. For instance, the best performing algorithm and its policy are often evaluated on high-end desktops, but when ported to onboard compute platform the real-time performance of the policy are different, thus widening the 'Sim2Real' gap [13], [14]. Hence, having an end-to-end closed-loop analysis tool suite such as AirLearning, we can develop policies, characterize the real-time performance of said policies, identify critical bottlenecks, possibly co-design algorithms and hardware to bridge the 'Sim2Real' gap.

II. Air Learning

AirLearning consist of four keys components: the environment generator, algorithm exploration, closed loop real time hardware in the loop setup, and quality of flight metrics that is cognizant of the resource constraints of the UAV. By using all these components in unison, AirLearning allows to carefully fine tune algorithms for the underlying hardware. Figure 1 illustrates the AirLearing infrastructure.

A. Environment Generator

Environment generator is creates high fidelity photo-realistic environments for the UAV's to fly in. It is built on top of UE4 and uses AirSim UE4 [15] plugin for the UAV model and flight physics.

The Environment generator uses a game configuration file to generate the required configuration tailored for the task. Here a task corresponds to navigating from one point to another while avoiding obstacles. Game configuration file is the interface between Unreal and the reinforcement learning algorithms. The runtime program writes a set of parameters into the game configuration file that the Unreal engine parses before starting the game. The Arena size, number of static/dynamic obstacles and their velocity (if relevant), minimum distance between obstacles, obstacle types, colors, texture and the destination of the UAV are some of the parameters that are configurable.

B. Algorithm exploration and Policy Exploration

We seed the AirLearning algorithm suite with Deep Q Network (DQN) [16], a commonly used reinforcement learning algorithm. DQN falls into the discrete action algorithms where the control commands are high-level commands ('move forward,' 'move left' e.t.c.,). For DQN Agent, we use the keras-RL framework. To make it more amenable to develop other reinforcement learning algorithms, we improve upon the work of AirGym [17] which provides the capability of interfacing with OpenAI gym. We extend the original implementation of AirGym by providing an array of new capabilities such as support for curriculum learning, training multi-modal policies with inputs such as depth/RGB/IMU measurements. These extrasensory data are exposed as part of the environment in OpenAI gym and used for developing a new class of RL algorithms and its associated policies.

C. Hardware Exploration

Once the Algorithm and policy are finalized, AirLearning allows for characterizing the performance of the different type of hardware platforms. Often aerial roboticists port the algorithm on real UAVs to validate the functionality of the algorithm. These UAVs might be a custom built[18] or off the shelf UAVs [19], [20]. Using our framework, we can explore the performance of different on-board off-the-shelf compute platforms or even examine the possibilities of designing custom processors (hardware accelerators).

To seed the framework, we use Ras Pi 3 as hardware platforms to evaluate the DQN algorithms. We use an established hardware-in-the-loop (HIL) methodology [21] to characterize the performance of various algorithms and policies. Our HIL methodology enables accurate performance and energy benchmarking of the computation kernels.

D. Quality of Flight Metrics

Algorithms and policies need to be evaluated on the metrics that describe the quality of flight such as mission time, distance flown, etc. We consider the following metrics:

Success rate (S_r)**:** The percentage of the times the UAV reaches the goal state without collisions or running out of battery. Ideally, we expect this number to be close to 100% as it reflects the algorithms' navigation efficiency.

Time to Completion (T_c)**:** The total time UAV spends finishing a mission within the simulated world.

Energy (E) consumed: Total energy spent while carrying the mission. Limited battery on-board constrains the mission time. Hence monitoring energy is of utmost importance and can be a measure of policy's efficiency.

Distance Traveled (D_t): Total distance flown while carrying out the mission. This metric is the average length of the trajectory. This metric can be used to measure the path planning intelligence of a particular policy.

III. Evaluation Methodology

This section discusses our training/testing methodology and DQN results. DQN policy generates high-level discrete actions (e.g., move forward) and these are mapped to low-level controls by the flight controller.

Environments: For the autonomous navigation task, we create an environment with varying levels of static obstacles. The environment size is 50 m x 50 m. The number of obstacles varies from 5 to 10, and it is changed every 4 episodes. The obstacles, start, and goal positions are placed in random locations in every episode to ensure that the policy does not overfit to the environment.

Training Methodology: We train the DQN agent on the environment described above. To speed up the learning, we employ the curriculum learning [22] approach where the goal position is progressively moved farther away from the starting point of the agent. To implement this, we divide the entire arena into multiple zones namely `Zone 1`, `Zone 2` and `Zone 3` as shown in Figure 2. Here `Zone 1` corresponds to the region that is within 16 m from the UAV starting position and `Zone 2` and Zone 3 are within 32 m and what is this number respectively. Initially, the position of goal for the UAV is chosen randomly such that the goal position lies within `zone 1`. Once the UAV agent achieves 50 % success over a rolling window of past 1000 episodes, the position of the goal expands to *zone 2* and so forth. To make sure that the agent does not forget learning in the previous zone, the goal position in the next zone is inclusive of previous zones. We train the agent to progress until zone 3 and upon achieving 50 % success rate in that zone we terminate the training. We checkpoint the policy at every zone so it can be evaluated on how well it has learned to navigate across all three environments. To compare the effectiveness of curriculum learning, we also train using non-curriculum learning where we train for 250,000 steps in the entire arena, the same number of steps where the curriculum learning reached `zone 3`.

Testing: For testing of curriculum learning policies, we evaluate the checkpoints saved at each zone. For testing the non-curriculum learning counterpart, we evaluate the checkpoints saved at fixed intervals of 50,000 steps. To ensure the policy does not overfit to the environment, we evaluate the policy on the unknown zone (`zone 3`) which was not used during training. We also randomly change the position of the goal and obstacles.

IV. Algorithm Evaluation

In this section, we compare the results of curriculum learning approach with the non-curriculum learning approach on the environment described in Section III. The primary metric is the success rate. The policy is evaluated on 100 trajectories.

Figure 3a and Figure 3c show success rate of policy trained using curriculum learning and non-curriculum learning respectively. Here `chkpt1`, `chkpt2`, `chkpt3` corresponds to the checkpoints of policy saved in `Zone1`. `Zone2` and `Zone3` respectively. The confusion matrix represents the success rate of a particular checkpoint in each zone. Here `Zone3` represents a region where the agent was not trained before. In Figure 3b and Figure 3d shows the number of steps it took for the agent to transition from one zone to the other for curriculum learning and non-curriculum learning respectively. Recall that in curriculum learning the criteria for transitioning from one zone to another is to finish the goal 50% of the time over a rolling window of 1000 episodes. For non-curriculum learning, the interval is fixed since we checkpoint at a fixed interval of 50,000 steps. In this evaluation, we use the policy checkpointed at 150,000 steps, 200,000 steps and, 250,000 steps respectively. For curriculum learning, we see that the transition from `Zone0` to `Zone1` happens at 140,000 steps and from `Zone2` to `Zone3` happens at 200,000 steps. Contrasting the success rate of curriculum learning and non-curriculum learning, we find that across different checkpoints and zones, policy trained using curriculum learning generally performs better than non-curriculum learning with a fewer number of steps. Also `chkpt3` policy of curriculum learning is the best policy compared to other policies trained using both approaches.

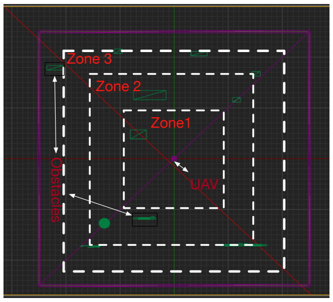


Fig. 2: Top view of the Arena. An Arena is divided into logical partition called Zones. The UAV is first trained into Zone0 and the goals are assigned within this region. The zone gets incremented when the UAV agent has met 50% success in the previous zone.

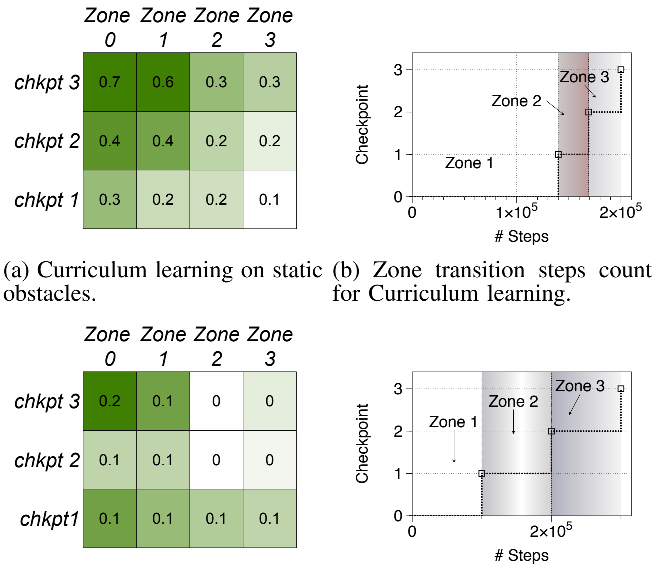
In summary, we use the curriculum learning technique to determine the best performing policy. We use this policy to evaluate the policy performance on a resource constrained Ras-pi 3 platform (Section V).

V. System Evaluation

This section evaluates the best policy obtained in the Section IV, on real embedded platforms such as Ras Pi 3 [23] using the hardware-in-the-loop (HIL) methodology. In HIL methodology, the state information is fed from the simulator running on a high end desktop and policy is evaluated on the Ras Pi 3. The actions determined by the policy on Ras Pi 3 are relayed back to the simulator.

The quality of flight metrics in system evaluation is time to completion (T$_c$), distance traveled (D$_t$) and energy left at the end of the mission (E). Using HIL evaluation and QOF metrics we show two things. First, the choice of onboard compute affects the quality of flight metrics. Second, the evaluation on a high-end machine does not accurately reflect the real-time performance on the onboard compute available on UAVs, thus contributing to a new type of 'Sim2Real' gap.

Table I shows the comparison of performance of the policy a high end desktop and Ras pi 3 averaged over 100 trajectories on `Zone 3`, the region not used during the training. For the policy with 4.4 Million parameters, the



(a) Curriculum learning on static obstacles.

(b) Zone transition steps count for Curriculum learning.

(c) Non-Curriculum learning

(d) Non-Curriculum learning

Fig. 3: Comparison of Curriculum learning and non-curriculum learning. Figure 3a shows the confusion matrix of success rate for curriculum learning. Figure 3b, shows the zone transition interval for curriculum learning. Figure 3c, shows the success rate for non-curriculum learning and Figure 3d shows the zone transition interval for non-curriculum learning.

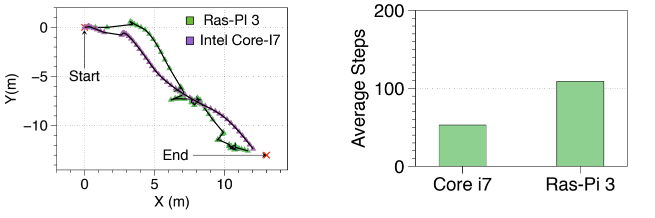
Metric	Intel core i7	Ras Pi 3	Performance Gap (%)
Inference latency (ms)	3.00	68.00	2166.66
Success rate (%)	30.00	25.00	5.00
QOF metrics			
Flight time (s)	64.43	88.86	37.90
Distance Flown (m)	34.92	44.16	26.43
Energy (kJ)	42356.72	54557.36	28.77

TABLE I: Inference time, success rate, and quality of flight metrics comparison between Intel core i7 desktop and Ras-Pi 3 in `Zone3` level. The policy under evaluation is the best policy obtained from Algorithmic evaluation.

inference time on Ras-Pi 3 is 68ms, while on the desktop, equipped with GTX 1080 Ti GPU and Intel core I7 CPU, is 3ms, more than 20 times faster. The policy running on the desktop is 5% more successful. While, some degradation in performance is expected, the magnitude of the degradation is an order of magnitude more severe for the other QoF metrics.

The trajectories on the embedded platform are longer, slower, and less energy efficient. The flight time needed to reach the goal on the high-end desktop on an average is 64.4 s whereas on Ras-pi 3 is 88.86 s, yielding a performance gap of around 38%. The Distance flown for the same policy on the high-end desktop has a trajectory is 34.9 m, whereas on Ras-pi 3 is 44.16 m thereby contributing to a difference of 26.43%. Lastly, the high-end desktop consumes on an average of 42 kJ of energy, while Ras-pi 3 platform consumes on an average of 54 kJ, 28.7% more energy.

To understand how the same policy performs differently in Ras-Pi 3 and desktop, we fix the position of the end goal and obstacles and evaluate 100 trajectories with the same configuration. The trajectory shown in Figure 4a is representative of the trajectories between the start and end goal. We can observe that the two trajectories are very different. The desktop trajectory orients towards the goal and the proceeds directly. The trajectory taken by Ras-Pi 3 starts towards the goal, but then makes a zig-zag pattern resulting in the longer trajectory. The is likely a result of obstacle-avoidance based on more stale sensory information, due to the longer inference time. Also, we notice the distance between each step is smaller, suggesting the agent is yawing more (stuck in the same position). Figure 4b shows that on the average the number of steps taken to reach the goal is higher in Ras-Pi 3 compared to desktop, suggesting that trajectories are slower compared to desktop.



(a) Trajectory

(b) Average Steps

Fig. 4: Trajectory comparison between Ras-Pi 3 and Intel Core i7.

In summary, we find that the choice of on-board compute along with algorithm affects the resulting UAV behavior and shape of the trajectory profoundly, and that the quality of flight metrics captures those differences better than the success rate. Evaluations done purely on high-end desktop might show lower energy consumed per mission but when ported to real robots, might actually consume more energy due to sub-par onboard compute. Using HIL methodology allows us to identify these differences in behavior and performance bottlenecks arising due to the on-board compute without having to port it real robots. Hence having HIL methodology helps bridging the ‘Sim2Real’ gap arising due to the limitation of the onboard compute.

VI. Conclusion

We develop AirLearning, which enables end-to-end analysis of reinforcement learning algorithms, and use it to compare the performance of curriculum learning based DQN vs. non-curriculum learning based DQN on a configurable environment with varying static obstacles. We show that the curriculum learning based DQN has better success rate compared to non-curriculum learning based DQN with the same number of experience (steps). We then use the best policy trained using curriculum learning and expose different UAV behaviour quantify the performance of the policy using hardware-in-the-loop on a resource-constrained Ras-Pi 3. Ras-pi 3 platform acts as a proxy for onboard compute on real UAVs. We evaluate the performance of the best policy using quality of flight metrics such as flight time, energy consumed and total distance traveled. We show that there is a non-trivial behavior change, and up to 38% difference in the performance of policy evaluated in high-end desktop and resource-constrained Ras-Pi 3 signifying that the choice of onboard compute is also a contributing factor for bridging the ‘Sim2Real’ gap.

References

[1] S. Waharte and N. Trigoni, “Supporting search and rescue operations with uavs,” in *2010 International Conference on Emerging Security Technologies*, pp. 142–147, IEEE, 2010.

[2] A. Goodchild and J. Toy, “Delivery by drone: An evaluation of unmanned aerial vehicle technology in reducing co2 emissions in the delivery service industry,” *Transportation Research Part D: Transport and Environment*, vol. 61, pp. 58–67, 2018.

[3] A. Faust, I. Palunko, P. Cruz, R. Fierro, and L. Tapia, “Automated aerial suspended cargo delivery through reinforcement learning,” *Artificial Intelligence*, vol. 247, pp. 381 – 398, 2017. Special Issue on AI and Robotics.

[4] K. Peng, L. Feng, Y. Hsieh, T. Yang, S. Hsiung, Y. Tsai, and C. Kuo, “Unmanned aerial vehicle for infrastructure inspection with image processing for quantification of measurement and formation of facade map,” in *2017 International Conference on Applied System Innovation (ICASI)*, pp. 1969–1972, IEEE, 2017.

[5] M. Fässler, *Quadrotor Control for Accurate Agile Flight*. PhD thesis, Universität Zürich, 2018.

[6] 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*, vol. abs/1604.07316, 2016.

[7] H. L. Chiang, A. Faust, M. Fiser, and A. Francis, “Learning navigation behaviors end-to-end with autorl,” *IEEE Robotics and Automation Letters*, vol. 4, pp. 2007–2014, April 2019.

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

[9] D. Gandhi, L. Pinto, and A. Gupta, “Learning to fly by crashing,” *CoRR*, vol. abs/1704.05588, 2017.

[10] R. M. Kretchmar, *A synthesis of reinforcement learning and robust control theory*. Colorado State University Fort Collins, CO, 2000.

[11] Crazyflie, “Crazyflie 2.0.” https://www.bitcraze.io/crazyflie-2/, 2018.

[12] DJI, “Dji-mavic pro.” https://www.dji.com/mavic, 2018.

[13] S. Koos, J.-B. Mouret, and S. Doncieux, “Crossing the reality gap in evolutionary robotics by promoting transferable controllers,” in *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pp. 119–126, ACM, 2010.

[14] A. Boeing and T. Bräunl, “Leveraging multiple simulators for crossing the reality gap,” in *2012 12th International Conference on Control Automation Robotics & Vision (ICARCV)*, pp. 1113–1119, IEEE, 2012.

[15] S. Shah, D. Dey, C. Lovett, and A. Kapoor, “AirSim: High-fidelity visual and physical simulation for autonomous vehicles,” *CoRR*, vol. abs/1705.05065, 2017.

[16] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing atari with deep reinforcement learning,” *arXiv preprint arXiv:1312.5602*, 2013.

[17] K. Kjell, “Project title.” https:https://github.com/Kjell-K/AirGym, 2018.

[18] Nvidia-Ai-Iot, “Nvidia-ai-iot/redtail.”

[19] A. Hummingbird, “Asctec hummingbird.” https:http://www.asctec.de/en/uav-uas-drones-rpas-roav/asctec-hummingbird/, 2018.

[20] Intel, “Intel aero ready to fly drone.” https:https://www.intel.com/content/www/us/en/products/drones/aero-ready-to-fly.html, 2018.

[21] W. Adiprawita, A. S. Ahmad, and J. Semibiring, “Hardware in the loop simulator in UAV rapid development life cycle,” *CoRR*, vol. abs/0804.3874, 2008.

[22] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, “Curriculum learning,” in *Proceedings of the 26th annual international conference on machine learning*, pp. 41–48, ACM, 2009.

[23] R. pi 3, “Raspberry pi 3.” https://www.raspberrypi.org/products/raspberry-pi-3-model-b/, 2018.