Learning to Navigate from Synthetic Data for Friction-Adaptive Autonomous Driving

Dai-Ying Hsieh, Hsiao-Han Lu, Ru-Fen Jheng, Hung-Chen Chen, Hong-Han Shuai, Wen-Huang Cheng
National Chiao Tung University, Hsinchu, Taiwan
karta1297963.eed02@g2.nctu.edu.tw, {hsiaohan.eed05, ruby2332ruby.eed05, gkfd45.eed05, hhshuai,whcheng}@nctu.edu.tw

Abstract

We introduce AutoBench, an open-source driving simulator for reinforcement learning research on navigation tasks. The goal of AutoBench is to create a virtual configurable environment with different difficulty levels and environmental conditions, e.g., road width, the friction coefficient of ground, weather condition, day/dusk/night, for training and testing the learning models. As a demo case, we propose Friction-adaptive Autonomous Driving via Reinforcement Learning (FAD-RL) that learns to navigate from synthetic data for friction-adaptive autonomous driving. Based on AutoBench, the proposed FAD-RL is trained and evaluated on the standard driving task, i.e., driving forward and backward on a curving road (S-bend) of grounds with different friction coefficients. Experimental results show that the proposed FAD-RL significantly increases the success rate of arriving at the finishing point.

Introduction

With the development of sensing technology and artificial intelligence, Artificial Intelligence of Things (AIoTs) becomes popular solutions for many applications, while autonomous navigation is an essential ability shared among different AIoT applications, e.g., object tracking (Fu et al. 2014), disaster rescue (Birk et al. 2011), wildlife protection (Olivares-Mendez et al. 2015), parcel delivery (Wang et al. 2019). Conventionally, navigation has been tackled by simultaneous localization and mapping (SLAM) in two stages: (1) mapping the environment with a 3D point cloud derived from LiDAR, and (2) planning a path through the map. However, it is time-consuming to build and update the map. Moreover, without the map, the conventional methods are not able to perform an accurate navigation.

Recently, the deep reinforcement learning (DRL) approaches have achieved great success in different tasks (Lillicrap et al. 2016; Mnih et al. 2016; 2015; Schulman et al. 2015; 2017; Chiang et al. 2019) due to its flexibility and robustness to different situations. Specifically, DRL provides an end-to-end learning procedure, in which agents jointly learn the representation and action policy to improve the limitation of hand-crafted features and predefined rules. Moreover, DRL can learn difficult tasks through trial-and-error experiences with environments while conventional criterion (e.g., shortest path or minimum energy path of path planning) may be too hard to be satisfied. For example, (Tai, Paolo, and Liu 2017) applies a 10-dimensional range sensor as input and employed Deep Deterministic Policy Gradient (DDPG) to train the agent for autonomous navigation.

Learning and evaluating reinforcement learning for different tasks on simulators have become popular nowadays since 1) reinforcement learning requires simulating different situations for deriving a robust model, and 2) the cost of failure is too expensive, e.g., traffic collision of autonomous driving. For example, OpenAI Gym provides both 2D environment such as Atari, Box2D and 3D MuJoCo environments for continuous control tasks (Brockman et al. 2016). DeepMind Lab provides a 3D navigation task based on Quake III Arena engine (Beattie et al. 2016). Moreover, ViZDoom utilizes Doom to built a First-Person Shooter environment for visual reinforcement learning (Kempka et al. 2016). A variety of state-of-the-art algorithms have been proposed to solve these tasks, and even outperform human capability in some cases, e.g., (Mnih et al. 2015; Lample and Chaplot 2017).

On the other hand, for autonomous driving, Gazebo creates a 3D dynamic multi-robot environment with physics engine and rendering capabilities, and is highly-compatible with Robot Operating System (ROS) (Koenig and Howard 2004). Furthermore, AirSim provides 3D environment with highly-detailed visualization and physical models for UAVs (Shah et al. 2018). Recently, CARLA focuses on creating a simulator for urban area driving and weather control (Dosovitskiy et al. 2017). Despite providing the sophisticated environments, these projects are still lack of user-friendly difficulty controls, e.g., inputting from 1 to 4 as the difficulty levels.

Therefore, in this paper, we introduce AutoBench, aiming to provide a difficulty and reward configurable testbed with different environment conditions for building autonomous driving benchmark. With the plug-and-play interface, various time/weather selections and visual observation types,

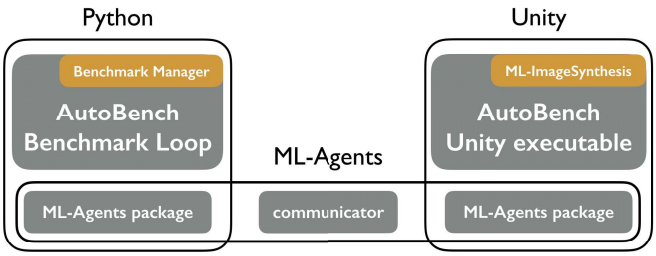


Figure 1: System Architecture of AutoBench

users can train and validate their algorithms without building the environments, which breaks the barriers to entry into reinforcement learning. Moreover, since the curriculum learning (Bengio et al. 2009), which incrementally increases the difficulty to train the learning model, has been widely adopted and proved to be effective on many tasks, the proposed difficulty-configurable environment can also serve as the environment to test different curriculum learning algorithms.

To demonstrate the ability of AutoBench, we formulate a new research problem of autonomous driving on different friction grounds and propose Friction-adaptive Autonomous Driving via Reinforcement Learning (FAD-RL). Specifically, we propose a new RL method by theoretically analyzing the relation between control actions and friction coefficients to automatically adjust the control actions without re-training. The results not only manifest the effectiveness of the proposed FAD-RL but also the usefulness of AutoBench.

AutoBench

To provide a difficulty and reward configurable training environment for autonomous driving, we propose AutoBench[1] based on Unity Engine and Unity ML-Agents (Juliani et al. 2018). Unity ML-Agents is an open source framework for training AI agents and provides a plug-and-play framework for both Python and Unity. It is worth noting that ML-Agents also provides a wrapper class that supports the OpenAI Gym interface for further integration, i.e., users are able to test any OpenAI-Gym-compatible algorithms on AutoBench.

The system architecture of AutoBench is shown in Figure 1, where ML-Agents forms the bottom layer of AutoBench including packages for both Unity and Python environments and a communicator for exchanging data between Python and Unity. On top of that, AutoBench on the Python side provides a training loop for learning, which offers the flexibility of adopting different ML approaches to solve the task. Moreover, *BenchmarkManager* is constructed to keep track of the benchmark progress and analyze the results. On the other hand, AutoBench on the Unity side constructs a compiled instance of the environment and automatically launches when the training starts.

In addition to building the environment, we provide a pre-trained model as shown in Figure 2, where vector inputs are the coordinates of the vehicle. The architecture follows the

[1]https://github.com/karta1297963/AutoBench

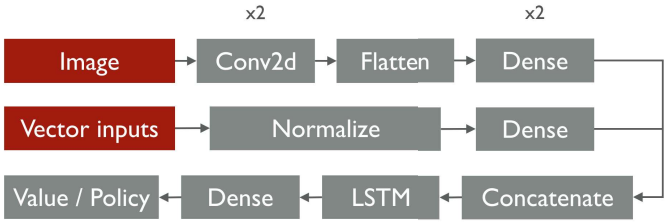
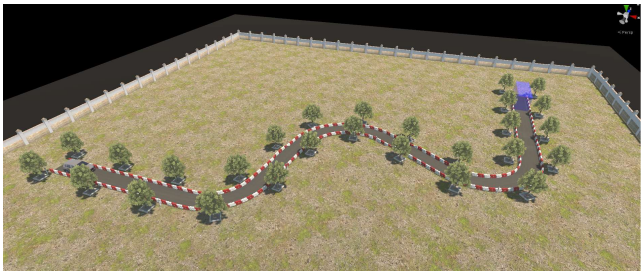
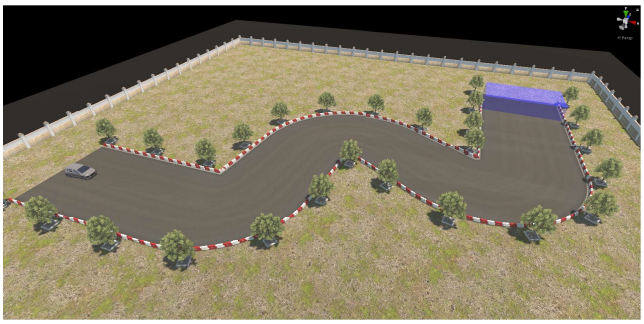


Figure 2: Pre-trained Model Architecture



(a) Road Width = 3m



(b) Road Width = 15m

Figure 3: Configurable Difficulty w.r.t. Road Width

standard PPO (Schulman et al. 2017) implementation with the following modification: 1) adding Convolutional Neural Networks (CNN) as the visual encoder, 2) adding Long Short Term Memory (LSTM) (Hochreiter and Schmidhuber 1997) layer after the concatenation of visual and vector latent features to implement recurrent function in Deep Recurrent Q-Network (Hausknecht and Stone 2015). The results can be found in https://youtu.be/Ptg1hnLxy9U.

Configurable Difficulty

With the configurable difficulty levels, different algorithms can be benchmarked through the success rates of different-level tasks. Here, three parameters are adjustable for the difficulty control of the environment: 1) road width, 2) visual details, and 3) visual observation types, which can be easily specified through the JSON file in the python project. Specifically, variants of the road width are shown in Figure 3. For the visual observation types, we integrate *ML-ImageSynthesis* (ML-ImageSynthesis 2017) plugin to provide image types for raw, segmentation, depth, optical flow, and normal ones. AutoBench also provides weather and time

(a) 11:00, Cloudy: Level 3 (b) 11:00, Foggy: Level 2

(c) 7:00, Cloudy: Level 1 (d) 18:00, Cloudy: Level 1

Figure 4: Configurable Difficulty w.r.t. Weather and Time

controls to add more diversity to the environment. Take Figure 4 as an example. The weather condition includes clear sky, cloudy, foggy, rainy, storm and snowy of different levels. Moreover, the time of the day is a continuous option with corresponding sun angle and lighting color change.

Environmental Details

The environment consists a 50x50m ground surrounding by 3m walls, and a car model approximately 4.3x1.8x1.4m (referred to Figure 3). The goal is to navigate to the finishing area[2] through the S curve path without colliding with the barriers. To improve training efficiency, AutoBench trains 10 agents simultaneously and independently. Moreover, AutoBench provides 3 cameras setups for each agent's visual observations: one front-facing camera mounted on the top center of the vehicle, and two rear-facing cameras simulating two side mirrors of the vehicle. Users can specify each camera with any type of visual observations or simply disable the camera visual output. The training information of agents is listed below.

- Observations: Vector observations with 7 variables corresponding to the velocity of the agent, relative position of the target and Y-axis (yaw) rotational angle; Visual observations corresponding to the 3 camera outputs.
- Actions: 9 discrete actions corresponding to 3 options of throttle levels (+1, 0, -0.3) magnitude of throttle and 3 options of steering angles (+30,0,-30) degree.
- Rewards: 5 parts including the position reward, velocity reward, success reward, time penalty and collision penalty. To utilize the reward shaping, the reward coefficient can be specified in the configuration file.

[2]The blue box is only for visualizing finishing area, and agents cannot see it.

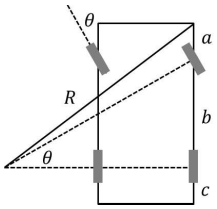


Figure 5: physical diagram of the vehicle

Demo Case: Friction-adaptive Autonomous Driving

In this section, we study the problem of autonomous driving on grounds with different friction coefficients. One of the possible solutions is to train an RL model by adjusting the environment parameters so that the agent can learn from different situations. However, it is infeasible to specify all the friction coefficients. Moreover, the training time of this basic approach significantly increases, which is severe for RL training. Therefore, we propose a method to train the RL in one environment and theoretically analyze the relation between control actions and friction coefficients for automatically adjusting the actions.

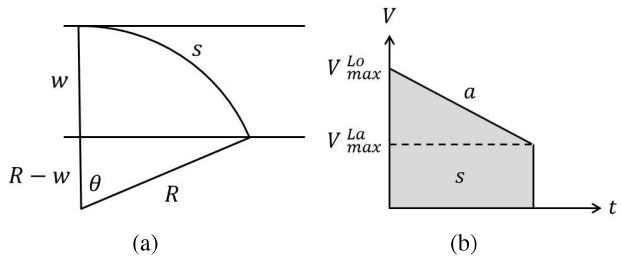
Lateral control

The effect of skid on the turn is greater than the effect on driving forward and backward when the friction force is reduced since the changing direction is at an angle with the direction of motion, i.e., the force the tires are exerting is also at an angle with the momentum vector. Therefore, to safely make a turn, the velocity of vehicle requires being limited on the aspect of turn at first. It is known that the vehicle is in a circular motion when conducting a turn. In the condition that the road surface is horizontal, the centripetal force of the circular motion, denoted as f_c, is all provided by the road friction. Let f_{fmax} and μ denote the maximum road friction and the friction coefficient, respectively. The maximum speed limit, V_{max}^{La}, is obtained when cornering.

$$\begin{aligned} f_c &= \frac{m \times {V_{max}}^2}{R} = f_{fmax} \\ f_{fmax} &= \mu \times mg \\ \Rightarrow \quad V_{max}^{La} &= \sqrt{g \times \mu \times R} \end{aligned} \tag{1}$$

where R denotes the radius of gyration. In other words, when raining, the friction coefficient μ is smaller, which results in 1) a smaller maximum speed V_{max}^{La} or 2) increasing the radius of gyration to maintain the speed. Next, we aim to find the radius of gyration R of the vehicle. Figure 5 shows a physical diagram of the vehicle, where a stands for the front suspension of the vehicle, b is the wheelbase, and c is the rear suspension of the vehicle. Since a, b, c and θ can be found from the environment, R can be derived from the figure as follows.

$$R = \sqrt{(a+b)^2 + b^2 \cot^2 \theta}. \tag{2}$$



(a) (b)

Figure 6: (a) The curve of the turning process (observed from the front of the vehicle) and (b) Velocity-time diagram of the turning process

By Equation 1, we can get V_{max}^{La} by setting the friction coefficient μ as

$$V_{max}^{La} = \sqrt{g \times \mu \times \sqrt{(a+b)^2 + b^2 \cot^2 \theta}}. \quad (3)$$

Longitudinal control

After deriving the maximum speed limit of the turn into operation, the newly converted model is still insufficient since the speed may be too fast before entering the turn. In this case, the vehicle will be too fast and cannot intermediately reduce to V_{max}^{La}, and thus may hit the edge of the road. In other words, since the proposed method judges whether the speed of the car is greater than the maximum speed limit and then decelerates in the moment when the car makes a turn, the velocity of the car is likely too high before entering the action of turning action. On the other hand, if we apply the maximum speed limit to the entire route during the turning period, it will spend more time to reach the end, and even exceed the timeout set in the environment, which is judged as a failure.

To solve this problem, another limitation on velocity is proposed, i.e., restricting the velocity when the vehicle moves forward. If the vehicle starts to turn under this velocity, it can decelerate to the limit velocity in time without hitting the edge of the road. Suppose that the vehicle should drive in the middle of the road. Whenever it starts to turn, the minimum distance between the front of the vehicle and road side should be larger than half of the width of road: w meters. Then, we take the worst case as an example. The vehicle is driving toward the road side, and starts to turn at the time the distance between vehicle and road side is w meters left as Figure 6(a) shown. When the vehicle is completely parallel to the road, it will be the time that it is the closest distance to the edge of the road. Before that condition happened, the speed of the vehicle must be reduced to the maximum speed of the turn, so that the car no longer continues slipping to hit the roadside.

Because the whole process time of turning process is extremely short and the velocity with limitation is not fast, we ignore the distance of lateral skid during the period of turning. Knowing that the distance between the radius of gyration R and the edge of the road at the start of the car is w meters, we first calculate the angle of θ in the figure with inverse trigonometric function as the formula 4 below. Then,

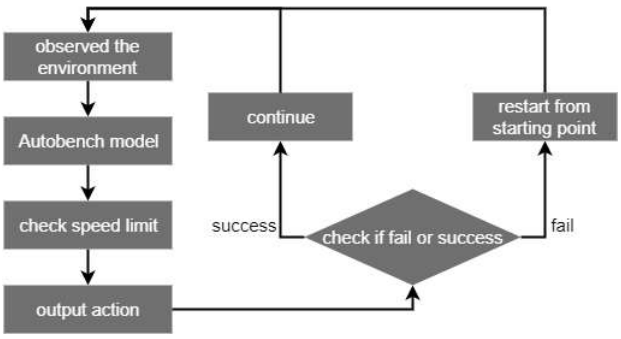


Figure 7: action flow chart

we get the driving distance s of the whole process with θ.

$$\begin{aligned} \theta &= \arccos \frac{R-w}{R} \\ s &= 2R\pi \times \frac{\theta}{2\pi} = R\theta \end{aligned} \quad (4)$$

During the turning process, the vehicle performs an uniform deceleration motion. It can easily found the maximum linear velocity by v-t graph as Figure 6(b). The v-t graph of uniform deceleration motion is a shape of trapezoid. The area of this trapezoid is the distance vehicle traveling. The slope of this trapezoid is the acceleration value of the vehicle during braking. Also, the maximum velocity of the vehicle turning V_{max}^{La} is known in the previous part. Accordingly, we can deduct the maximum linear velocity V_{max}^{Lo}.

$$\begin{aligned} & {V_{max}^{La}}^2 = {V_{max}^{Lo}}^2 + 2as \\ \Rightarrow \quad & V_{max}^{Lo} = \sqrt{{V_{max}^{La}}^2 - 2as} \end{aligned} \quad (5)$$

System flow

The idea is to adaptively restrict the speed limit on output for driving on the ground with different friction coefficients. In other words, the process before inputting the observation to the AutoBench model is same as the original. After AutoBench model decides the output action, the proposed FADRL first determines whether the action is turning. Afterward, it compares the present velocity to the lateral or longitudinal speed limit according to the action. If it is larger than the maximum velocity, change the item of speed control in the action to deceleration. If it is not speeding, then do not do any action. After this, return the action back to the vehicle and execute it. System flow chart is illustrated in Figure 7.

To construct an environment with different friction coefficient for a model by converting the former environment, we use the Wheel Collider function in the Unity environment to set up a less frictional environment. Therefore, the vehicle in the environment would easily slip, it can mimic a road surface with a small friction coefficient. We modify several parameters in Wheel Collider: (1) Extremum Slip, Extremum Value, Asymptote Slip and Asymptote Value of forward friction, (2) Extremum Slip, Extremum Value, Asymptote Slip and Asymptote Value of sideways friction. Forward

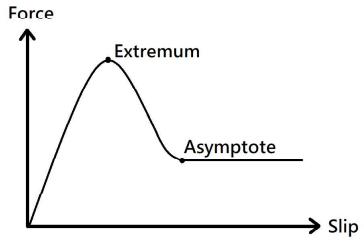


Figure 8: slip to tire force

and sideway frictions represent the properties of tire friction when the wheel is rolling forward/sideways respectively.

On the basis of Forward Friction and Sideways Friction, we can make line graphs respectively as below. The horizontal axis represents slip, which refers to how much the tire is slipping, that is, the difference between the surface velocity of tire and the moving velocity of vehicle. Due to the value of slip, when the vehicle is moving, it corresponds to different tire force unleashed on the connecting dot. The vertical axis of the graph shows the size of the tire force. According to the properties of tires, we can draw a curve which approximately to a two-piece spline as Figure 8. This curve can be separated into two main parts. In the first section, the curve would climb steadily to the point (Extremum Slip , Extremum Value). Besides, the curve's tangent at the point (Extremum Slip , Extremum Value) is zero. In the second part, curve would begin to go down from the point (Extremum Slip , Extremum Value) to the point (Asymptote Slip , Asymptote Value). Similarly, the curve's tangent at the point (Asymptote Slip , Asymptote Value) is zero. When we lower down Extremum Slip, Extremum Value, Asymptote Slip and Asymptote Value, tire force exerted on the contact point would decrease, implying the smaller tire friction. Therefore, the design reaches our expectation of skiddy road.

Experimental results

First of all, we re-train a new model in the default environment with AutoBench as a converted model. Next, we convert the model by our method according to the friction coefficient (0.375). For the baseline, we train a new model on AutoBench again based on new environment (friction coefficient=0.375) and compare new model to the proposed FAD-RL, which applies speed limit, to see if our method can achieve the same effect.

From Table 1, the original model in new environment fails to arrive the destination; thus, it is necessary to retrain a new one.If we retrain the new model with our method, it can reach 70% success rate and save lots of training time. Although the time step is increased by a factor of 0.5, the speed reduction in a slippery environment is required for safe driving, so the increase of time is acceptable. We retrain a model that can adapt to the friction coefficient in the new environment, and we found that the time step has indeed increased. Otherwise, we found that the average time step has little difference between models above. Thus, we successfully save training time by transforming the model.

Round	In original environment	In drift environment	In drift environment with speed limit
1	143	F	F
2	133	F	223
3	139	F	208
4	133	F	193
5	128	F	F
6	138	F	F
7	134	F	210
8	F	F	199
9	135	F	214
10	130	F	210
avg time step	134.8	0	208.1
success rate	90%	0%	70%

Table 1: Model trained with 30M steps in different environments. (F: fail) (Unit: time step)

Round	New trained Model with 10M steps	Original model with speed limit
1	241	F
2	241	223
3	234	208
4	238	193
5	243	F
6	239	F
7	244	210
8	244	199
9	237	214
10	235	210
avg time step	239.6	208.1
success rate	100%	70%

Table 2: Model transformed (with speed limit) compared to model trained with 10M steps in drift environment. (F: fail) (Unit: time step)

Conclusion and Future Work

In this paper, we introduce AutoBench, an open-source driving simulator for reinforcement learning research on navigation tasks. To demonstrate the usefulness, we formulate a new problem of navigation on the ground with different friction coefficients. Experimental results manifest that 1) the proposed FAD-RL is effective and 2) AutoBench provides an useful environment for RL training, which facilitates many applications. We plan to extend AutoBench by adding curvature as the configurable options since the curvature of the road is intuitively related to difficulty of the task. Moreover, we plan to design an algorithm that is able to adjust the difficulty dynamically based on the performance while eliminating the operational cost of human for adjusting the difficulty manually.

References

Beattie, C.; Leibo, J. Z.; Teplyashin, D.; Ward, T.; Wainwright, M.; Küttler, H.; Lefrancq, A.; Green, S.; Valdés, V.; Sadik, A.; et al. 2016. Deepmind lab. *arXiv preprint arXiv:1612.03801*.

Bengio, Y.; Louradour, J.; Collobert, R.; and Weston, J. 2009. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, 41–48. ACM.

Birk, A.; Wiggerich, B.; Bülow, H.; Pfingsthorn, M.; and Schwertfeger, S. 2011. Safety, security, and rescue missions with an unmanned aerial vehicle (uav). *Journal of Intelligent & Robotic Systems* 64(1):57–76.

Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. Openai gym. *arXiv preprint arXiv:1606.01540*.

Chiang, H.-T. L.; Faust, A.; Fiser, M.; and Francis, A. 2019. Learning navigation behaviors end-to-end with autorl.

Dosovitskiy, A.; Ros, G.; Codevilla, F.; Lopez, A.; and Koltun, V. 2017. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, 1–16.

Fu, C.; Carrio, A.; Olivares-Mendez, M. A.; Suarez-Fernandez, R.; and Campoy, P. 2014. Robust real-time vision-based aircraft tracking from unmanned aerial vehicles. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Hausknecht, M., and Stone, P. 2015. Deep recurrent q-learning for partially observable mdps.

Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. *Neural computation* 9(8):1735–1780.

Juliani, A.; Berges, V.-P.; Vckay, E.; Gao, Y.; Henry, H.; Mattar, M.; and Lange, D. 2018. Unity: A general platform for intelligent agents. *arXiv preprint arXiv:1809.02627*.

Kempka, M.; Wydmuch, M.; Runc, G.; Toczek, J.; and Jaśkowski, W. 2016. Vizdoom: A doom-based ai research platform for visual reinforcement learning. In *Computational Intelligence and Games (CIG), 2016 IEEE Conference on*, 1–8. IEEE.

Koenig, N., and Howard, A. 2004. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Lample, G., and Chaplot, D. S. 2017. Playing fps games with deep reinforcement learning. In *Thirty-First AAAI Conference on Artificial Intelligence*.

Lillicrap, T. P.; Hunt, J. J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; and Wierstra, D. 2016. Continuous control with deep reinforcement learning. In *International Conference on Learning Representations (ICLR)*.

ML-ImageSynthesis. 2017. Ml-imagesynthesis repository. https://bitbucket.org/Unity-Technologies/ml-imagesynthesis.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning.

Mnih, V.; Badia, A. P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; and Kavukcuoglu, K. 2016. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning (ICML)*.

Olivares-Mendez, M. A.; Fu, C.; Ludivig, P.; Bissyandé, T. F.; Kannan, S.; Zurad, M.; Annaiyan, A.; Voos, H.; and Campoy, P. 2015. Towards an autonomous vision-based unmanned aerial system against wildlife poachers. *Sensors* 15(12):31362–31391.

Schulman, J.; Levine, S.; Moritz, P.; Jordan, M. I.; and Abbeel, P. 2015. Trust region policy optimization. In *International Conference on Machine Learning (ICML)*.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal policy optimization algorithms. volume abs/1707.06347.

Shah, S.; Dey, D.; Lovett, C.; and Kapoor, A. 2018. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and service robotics*, 621–635. Springer.

Tai, L.; Paolo, G.; and Liu, M. 2017. Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 31–36. IEEE.

Wang, D.; Hu, P.; Du, J.; Zhou, P.; Deng, T.; and Hu, M. 2019. Routing and scheduling for hybrid truck-drone collaborative parcel delivery with independent and truck-carried drones. In *IEEE Internet oF Things Journal (JIOT)*.