Reinforcement Learning-based Autonomous Driving Agent

Megerősítéses Tanulás Alapú Önvezető Ágens

Training a lane following agent in a Duckietown simulation environment

Márton Moró
TMIT
BME
Budapest, Hungary
moro.marton@edu.bme.hu

Bence Szabó
IIT
BME
Budapest, Hungary
bszabo1652@edu.bme.hu

Balázs Zsolt Ungvárszky
HVT
BME
Budapest, Hungary
balazszsolt.ungvarszky@edu.bme.hu

***Abstract* — This paper describes two solutions to the Duckietown lane following challenge using deep reinforcement learning. In our case Duckietown is a simple environment consisting of roads and green areas. This is where the Duckiebot has to navigate using its camera for lane detection. The solutions are two policy optimization method-based agents. The development was done in an environment with serious limitations but through hyperparameter tuning and other optimizations we achieved increasingly better results. In the following we discuss the DuckieTown framework, environment, the exact building blocks of the agent and how we optimized to use the available computational resources and increase its performance.**

***Keywords* — Duckietown, Reinforcement Learning, Python, Optuna, Stable Baselines, Google Collaboratory, Deep learning, algorithms, PPO, A2C, Hyperparameter Tuning**

I. Introduction

Autonomous-driving is one of the most popular and controversial topics not only among software and hardware developers, but all over the world. Solving the complex task of controlling a vehicle to safely drive it from one point to another is causing a lot of headaches for engineers in the automotive industry. Part of this issue is the implementation of lane-following. Given that we in our team have all been curious about existing lane tracking solutions, we thought we would implement such a lane tracking system in a simple simulation environment using deep reinforcement learning algorithms. During our big homework, we created the simulation environment in which the agent was run, and then we selected some existing reinforcement learning algorithms, specialised them for the task and started teaching the agents. After completing the teachings, we switched to optimizing the hyperparameters of the algorithms in order to achieve the best possible results for our agents during the re-teaching. We also implemented other tools for running, testing, and teaching (Early stopping, RL algorithm seed setup, Callback function, evaluation function). Finally, we compared the solutions and evaluated the results obtained.

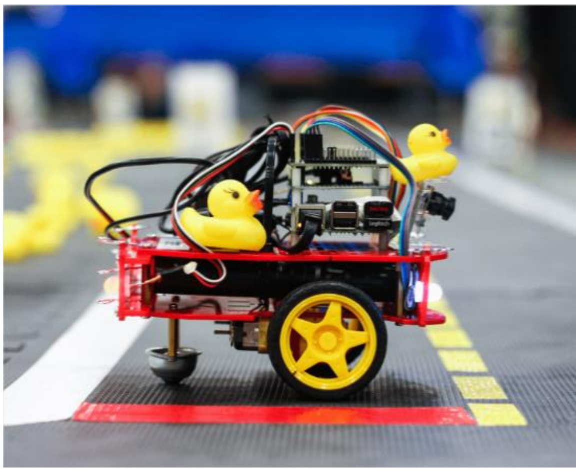


Figure 1. DuckieBot

II. Duckietown Framework

The Duckietown [1] project itself has been in existence since 2016. Initially, it was launched by MIT as a course with the goal of bringing robotics and the world of artificial intelligence closer to the audience. Today, we can talk about a framework for

Duckietown that can navigate those who are interested in the world of AI and Reinforcement Learning from the complete beginner level to the research level. The DuckieTown platform is made up of two elements the environment itself, “DuckieTown” and “DuckieBots”. DuckieBots are low-cost mobile robots equipped with a simple camera and ultrasonic sensor. DuckieTown is a test environment for an urban environment that can be assembled from simple building blocks. Thanks to the flexible installation of the building elements, a variety of environments can be created, from the simple straight two-lane road to the downtown-like environment filled with complex obstacles, traffic lights and signs.

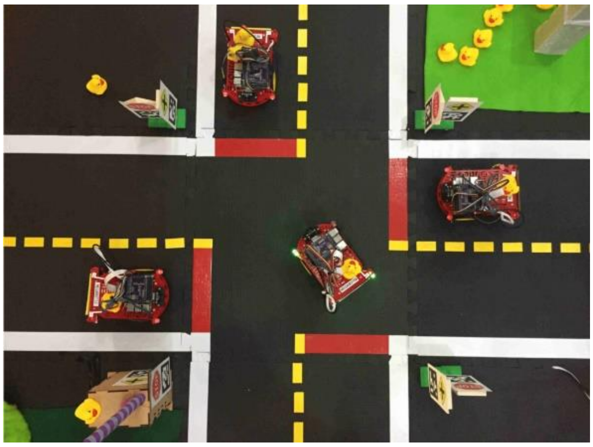


Figure 2. DuckieTown

In addition to these elements, a number of software packages are available for those who are interested in AI development. From the DuckieTown simulation environment, to simple control codes, to more complex evaluation and teaching codes there are several additional documents to the platform. Over the years DuckieTown has grown into a platform for software developers, students and now it is seen as a global competition by people with a simple goal to beat each other's solutions. Realizing this, DuckieTown developers created AIDO. The AI Driving Olympics is a bi-annual global AI and Machine Learning competition where candidates can compete against each other in a variety of challenges. One such task was lane-following. In our project, we addressed this problem in the simulation environment provided by DuckieTown.

III. Reinforcement Learning

Reinforcement Learning [2] is a field of Machine Learning where an agent interacts with the environment. Each action of the agent is based on the state it’s in and its policy. In exchange for an action, it receives the next state and a reward from the environment.

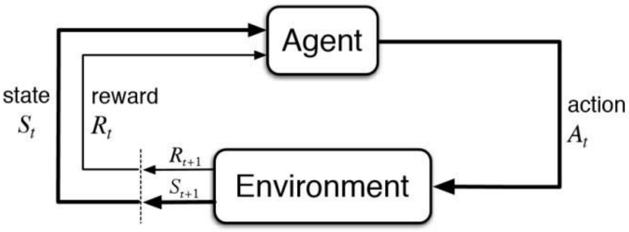


Figure 3. Reinforcement Learning, training flowchart

The aim of the agent is to maximize the expected return for following its policy from the current state. The policy which is used by the agents can be of many kinds. We preferred the methods based on deep neural networks.

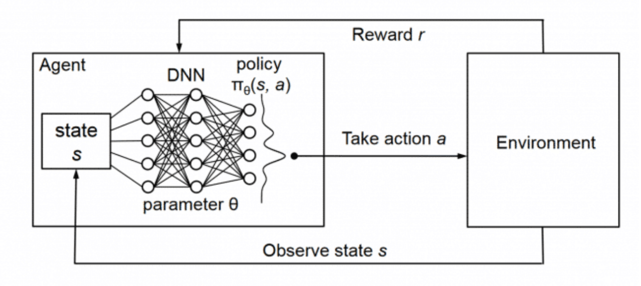


Figure 4. Agent with Neural network policy

There are a lot of methods with such settings so choosing the right algorithm was one of the key aspects of our project. The algorithms will be discussed in the V. chapter. Although the developer sets the reward policy, but he provides no hint for the model (agent) about how to solve the task. Utilising the power of searching and many trials reinforcement learning became a very famous tool demonstrate the ‘creativity’ of the machines. After a huge number of trials, it is possible that the machine can exceed the limitations of the human brain and in some cases achieve better results. Of course, to bring these characteristics forward a sufficiently powerful computer infrastructure is required which can be quite a fortune. Realising this we decided to use virtual machines for the training, these machines were provided by “Google Collaboratory” [3].

IV. Simulation environment

Before finding the right algorithms and training methods we had to create an appropriate environment [4] which can return state of the agent, the calculated reward after the previous action and can react to the current action taken by the agent. There are a lot of developed environments already in the existence but most of them was too simple or too complex for our lane following challenge. Fortunately, as part of a work at MILA the duckietown-gym environment was created. Duckietown-gym is a simulator for Duckietown framework written in Python/OpenGL. Within this gym you can place your agent inside of an instance of DuckieTown. Duckietown-gym [5] grows into a fully-functioning autonomous driving simulator where one can train and test ML, RL, IL models or even classical robotics algorithms. Besides its customizability, there are many additional features which can help in developing better and better models.

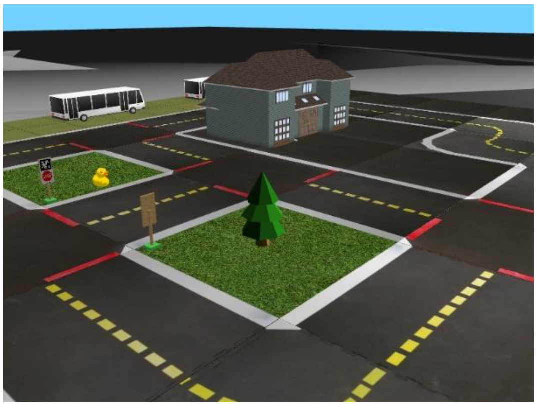


Figure 5. Duckietown-gym simulation environment

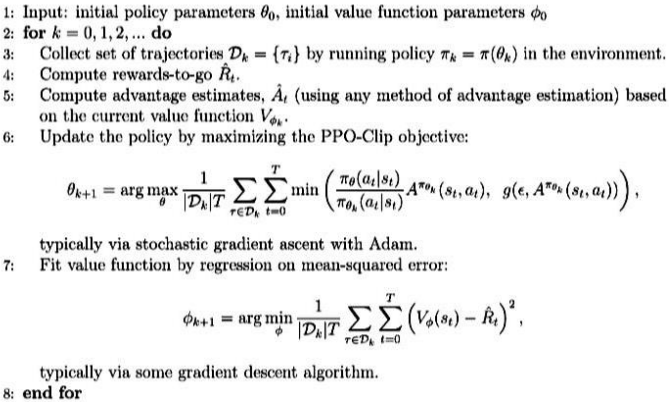
We used two maps to train our agent the first one was a straight road the second was a more complex loop road. The advantages and disadvantages of the environments will be discussed in the VI. Chapter. To improve the training capacity and to exploit the power of reinforcement learning we trained our models headless, which means we did not open the gym environment's display during training. After setting up the simulation environment the next step was to choose an adequate policy for the training.

V. METHODS

One of the first decisions that has to be made is the choice of method used in the model. We used Model-Free Reinforcement Learning and chose two Policy Optimization methods. These two were PPO (Poximal Policy Optimization) and A2C (Advantage Actor Critic). We also tried the DDPG (Deep Deterministic Policy Gradient) algorithm but since we ran into hardware limitations, it was decided not to use it. There will be a deeper dive into hardware limitations in Chapter VIII.

Proximal Policy Learning

PPO [6] uses multiple epochs of stochastic gradient ascent to perform each policy update. It has the stability and reliability of trust-region methods but is much simpler to implement and applicable in more general settings. PPO should require more RAM and it is also computationally more expensive than A2C.

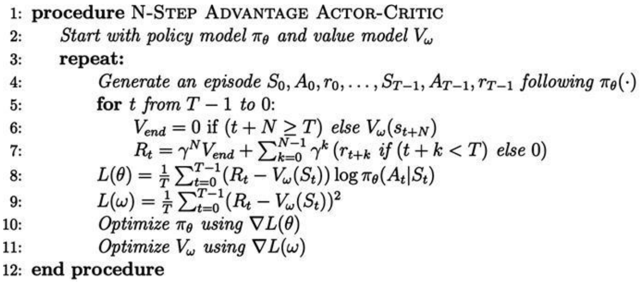


```
Input: initial policy parameters $\theta_0$, initial value function parameters $\phi_0$
for $k = 0, 1, 2, \ldots$ do
    Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
    Compute rewards-to-go $\hat{R}_t$.
    Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based
    on the current value function $V_{\phi_k}$.
    Update the policy by maximizing the PPO-Clip objective:
        $\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left( \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t),\ \ g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$
    typically via stochastic gradient ascent with Adam.
    Fit value function by regression on mean-squared error:
        $\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left( V_\phi(s_t) - \hat{R}_t \right)^2,$
    typically via some gradient descent algorithm.
end for
```

Figure 6. Pseudo Code for PPO

Advantage Actor Critic

A2C, or Advantage Actor Critic, is a synchronous version of the A3C policy gradient method. It is an Actor Critic Method which means that it has a Critic that estimates the value function and an Actor that updates the policy distribution in the direction suggested by the Critic and both the Critic and Actor functions are parameterized with neural networks.



```
procedure N-STEP ADVANTAGE ACTOR-CRITIC
    Start with policy model $\pi_\theta$ and value model $V_\omega$
    repeat:
        Generate an episode $S_0, A_0, r_0, \ldots, S_{T-1}, A_{T-1}, r_{T-1}$ following $\pi_\theta(\cdot)$
        for $t$ from $T-1$ to $0$:
            $V_{end} = 0$ if $(t + N \geq T)$ else $V_\omega(s_{t+N})$
            $R_t = \gamma^N V_{end} + \sum_{k=0}^{N-1} \gamma^k \left(r_{t+k} \text{ if } (t + k < T) \text{ else } 0\right)$
        $L(\theta) = \frac{1}{T} \sum_{t=0}^{T-1} (R_t - V_\omega(S_t)) \log \pi_\theta(A_t|S_t)$
        $L(\omega) = \frac{1}{T} \sum_{t=0}^{T-1} (R_t - V_\omega(S_t))^2$
        Optimize $\pi_\theta$ using $\nabla L(\theta)$
        Optimize $V_\omega$ using $\nabla L(\omega)$
end procedure
```

Figure 7. Pseudo Code for A2C

VI. TRAINING THE AGENT

We used the previously mentioned two methods (PPO, A2C) to create two models using the stable baselines 3 [7] framework and trained only on 1000 environment steps. This was clearly not enough to achieve high enough performance.

If we look at how the objectives are used to quantify how well the embodied task (lane-following) is completed, we can find the following criteria [8]:

VI.I Performance criteria

$$\mathcal{J}_{P-LF(V)}(t) = \int_0^t -v(t)dt$$

Here the performance indicator is the integrated speed along the road over time of the Duckiebot. This gives the number of tiles travelled over t time where t is up to the length of an episode.

VI.II Traffic law objective

$\mathcal{J}_{T-LF/LFV} = \text{median}(\{t_{outside}\})$, where $t_{outside}$ is the time spent outside of drivable zones. This objective is for penalizing actions such as leaving the lane or driving in the wrong lane.

VI.III Comfort objective

$\mathcal{J}_{C-LF/LFV}(t) = \text{median}(\{d_{outside}\}),$

where $d_{outside}$ is the sequence of lateral distances from the centre line.

This measures how "smooth and comfortable" is the driving of the Duckiebot.

Based on our observations we concluded the following behaviour for our agents. First it tries to avoid spending time in illegal driving zones and given sufficient time it aims to increase its speed along the road.

VII. HyperParameter Tuning

For hyperparameter tuning we used a hyperparameter optimization framework called Optuna [9] . For the PPO model we tuned the discount factor gamma, the learning rate, the number of epochs, the entropy coefficient and the number of steps to run for each environment per update.

For sampling we used a TPE (Tree-structured Parzen Estimator) based sampler. On each trial, for each parameter, TPE fits one Gaussian Mixture Model (GMM) l(x) to the set of parameter values associated with the best objective values, and another GMM g(x) to the remaining parameter values. It chooses the parameter value x that maximizes the ratio l(x)/g(x).

To ease the RAM constraint, we used garbage collection during tuning and we also used a percentile pruner which based on stored intermediate values determined if the given trial should be pruned or not.

VIII. Challenges of the development

Although there are a lot of existing optimized solutions [10] still reinforcement learning is one the most computationally demanding field of machine learning hence in order to achieve better performance, we decided to do the development using Google Colab. Although it was better than using our computers it had its own limitations. One of the biggest and most frequent problem was running out of RAM. To solve this during hyperparameter tuning we used a garbage collector function. Also, to speed up the tuning process we used the previously mentioned. Although these did certainly help, we still couldn't achieve the desired performance which is why we decided to try to use the server of the department.

IX. Future Improvements

One particular part where further improvements could clearly be made is image pre-processing [11] where cropping, resizing, colour segmentation, normalization and image sequencing could have increased the performance. Training in multiple parallel environments would result in faster training while also resulting in a better policy.

X. Conclusion

While attempting to solve the lane-following challenge we were constantly trying to see how different algorithms, models and frameworks compete against each other. We tested the DDPG policy by duckietown, other PPO1/2, A2C and DDPG by tensorforce [12] environment realising that we didn't have enough computing capacity to execute the necessary commands appropriately we moved on to stable baselines 3. Here we used PPO and A2C policies for the training. A2C policy happened to be a little less consistent so we chose PPO as the main algorithm for our lane-following challenge. In the next few figures, we will show some of the differences that we found between the two algorithms.

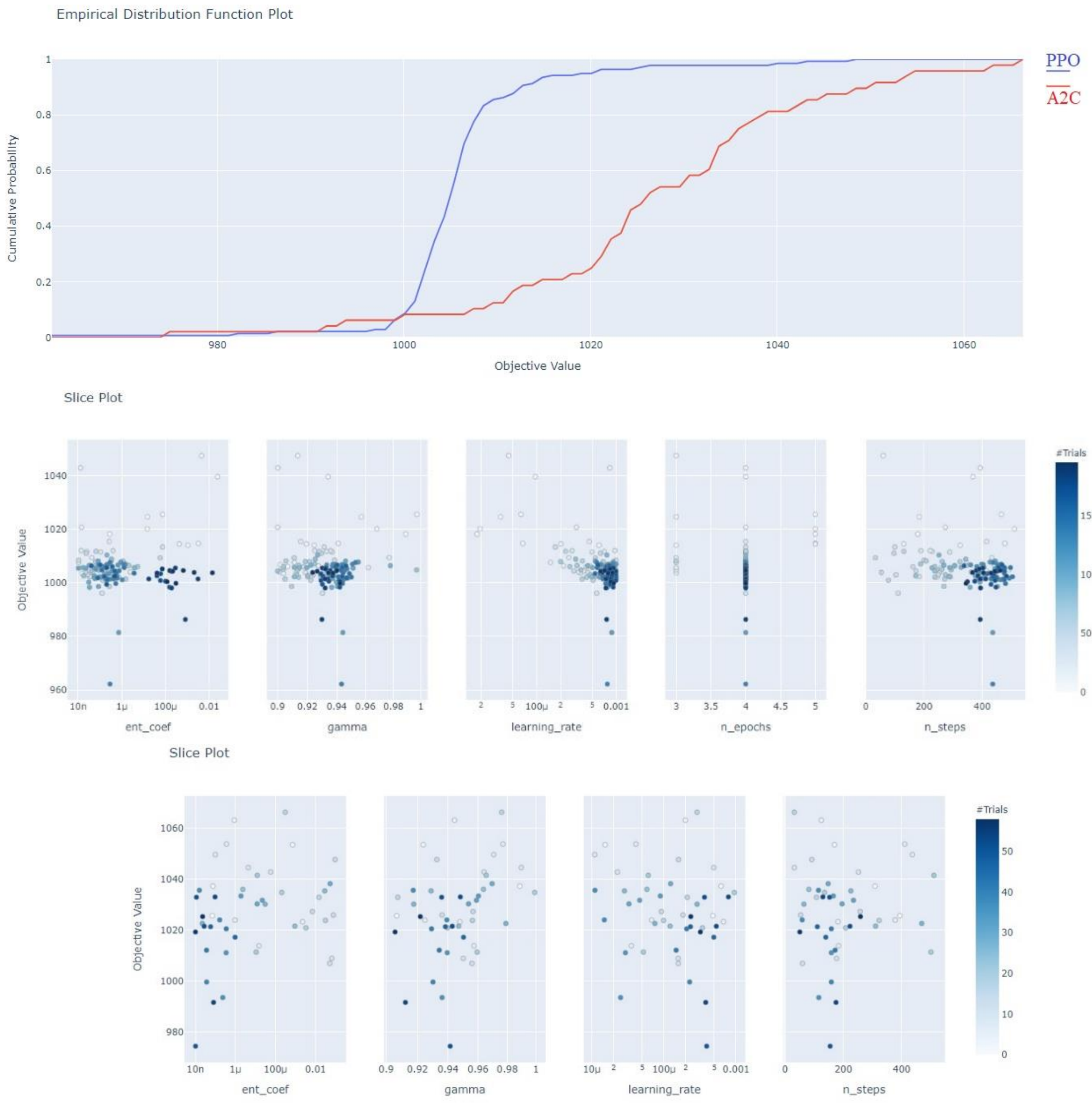


Figure 8 A2C, PPO hyperparameters

Keep in mind that by the time these figures were created, we could only train and optimize our agent for much less episodes than we wanted to due to the already mentioned hardware limitations.

If we look at Figure 8., we can see that the A2C's deviation is much higher than the PPO's. It means that during the evaluation phase, the agent trained by the A2C algorithm has a lot more outlies especially worse value. We can see this just looking at the plot's gradients, the PPO's plot is much steeper, this means there's a lot more results in this value range.

In Figure 8. we can see that, during the hyperparameter optimization, there were a lot more trials nearby the best result. These results lead us to think that PPO should produce much more consistent results during the evaluation. To sum it up from this project we gained a lot of knowledge in the area of reinforcement learning and we are planning on continuing to dive deeper into deep RL learning.

ACKNOWLEDGMENT

We would like to express our gratitude to Róbert Móni and Dr. Bálint Gyires-Tóth, for their patient guidance, enthusiastic encouragement and useful ideas of this research work. We would also like to say thanks to the professors of the course “Deep learning a gyakorlatban Python és LUA alapon”, for their advices and assistances.

REFERENCES

1. L. Paull et al., "Duckietown: An open, inexpensive and flexible platform for autonomy education and research," 2017 IEEE International Conference on Robotics and Automation (ICRA), Singapore, 2017, pp. 1497-1504, doi: 10.1109/ICRA.2017.7989179.
2. Jordi Torres (2020), A gentle introduction to Deep Reinforcement Learning
3. Bisong E. (2019) Google Colaboratory. In: Building Machine Learning and Deep Learning Models on Google Cloud Platform. Apress, Berkeley, CA. https://doi.org/10.1007/978-1-4842-4470-8_7
4. Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J. & Zaremba, W. (2016), 'OpenAI Gym', cite arxiv:1606.01540.
5. Chevalier-Boisvert, Maxime and Golemo, Florian and Cao, Yanjun and Mehta, Bhairav and Paull, Liam, Duckietown Environments for OpenAI Gym. 2018.
6. Dhariwal, Prafulla and Hesse, Christopher and Klimov, Oleg and Nichol, Alex and Plappert, Matthias and Radford, Alec and Schulman, John and Sidor, Szymon and Wu, Yuhuai and Zhokhov, Peter, OpenAI Baselines, Proximal Policy Optimization. 2017.
7. Raffin, Antonin and Hill, Ashley and Ernestus, Maximilian and Gleave, Adam and Kanervisto, Anssi and Dormann, Noah, stable-baselines3. 2019.
8. D. D. at Mila, The AI Driving Olympics. 2018
9. Takuya Akiba and Shotaro Sano and Toshihiko Yanase and Takeru Ohta and Masanori Koyama, Optuna: Next-generation Hyperparameter Optimization Framework. 2019.
10. Tani, Jacopo & Paull, Liam & Zuber, Maria & Rus, Daniela & How, Jonathan & Leonard, John & Censi, Andrea. (2016). Duckietown: An Innovative Way to Teach Autonomy. Advances in Intelligent Systems and Computing. 560. 10.1007/978-3-319-55553-9_8.
11. P. Almási, R. Moni and B. Gyires-Tóth, "Robust Reinforcement Learning-based Autonomous Driving Agent for Simulation and Real World," 2020 International Joint Conference on Neural Networks (IJCNN), Glasgow, United Kingdom, 2020, pp. 1-8, doi: 10.1109/IJCNN48605.2020.9207497.
12. Kuhnle, Alexander and Schaarschmidt, Michael and Fricke, Kai, Tensorforce: a TensorFlow library for applied reinforcement learning. 2017.