Self-driving in DuckieTown Environment - Önvezetés DuckieTown Környezetben

Botond Gergő Ungvárszki, Barna Radnai, Gábor Lévai
Budapest - Hungary - 2021
Email: ungbotond@gmail.com, radnaibarna@gmail.com, levaigabor.net@gmail.com

***Abstract*—[EN] The following paper reflects on the implementation of a work where reinforcement learning was used to move around a robot in a simulated DuckieTown environment. The task consists of guiding a vechicle along a predefined road network by tracking it. The robot could move in different directions depending on the input image under the influence of neural networks. In the model, this was solved by a simple convolutional network with a single camera source using different action wrappers. Another task was to make the simulation and development environment compatible with the latest software versions and plugins. The success of the robot's driving was measured by simple metrics such as average survival time and distance travelled.**

[HU] A következő tanulmány egy olyan munka implementálását tükrözi ahol reinforcement learning segítségével egy szimulált DuckieTown környezetben mozoghattunk. A feladat lényege, hogy az előre meghatározott tesztpályán sávkövetéssel végigvezessünk egy robotot, amely a bemeneti kép függvényében neurális hálók hatására különböző irányokba mozoghat. A modellben ezt egy egykamerás forrással üzemelő egyszerű konvolúciós hálózattal oldottuk meg különböző Action Wrapperek felhasználásával. Feladatunk volt még, hogy a szimulációs és fejlesztőkörnyezetet kompatibilissá tegyük a legfrissebb szoftververziókkal és beépülő csomagokkal. A robot vezetésének sikerességét egyszerű mérőszámokkal mértük mint átlagos túlélési idő és megtett távolság.

Keywords: DuckieTown, action wrappers, population based training, neural networks, machine learning, sim-to-real

I. Introduction

DuckieTowns are urban environments: roads, constructed from exercise mats and tape, and the signage which the robots use to navigate around. DuckieTowns can be transformed into smart cities by adding traffic lights and watchtowers [1]. It is an open, inexpensive and flexible platform for autonomy education and research [2]. This environment could be simulated by graphical 3D virtualization.

The simulation environment from which the data for the neural network development was extracted was provided by Gym-Duckietown. In this work, we updated the software environment to the latest version and reimplemented the work of our predecessors in depth [3]. We then performed hyperoptimization learning and testing using various action wrappers.

II. System Design

We used pre-defined wrapper classes in the model for training. The following files are containing the required classes:

- actionwrappers.py: Responsible for transforming the action space of Gym-Duckietown to alternate representations. These alternative spaces are: Discrete actions(left, right, forward); Wheel velocity (Braking); Wheel velocity(Clipped to 0-1 interval); Steering.
- observartionwrappers.py: Responsible for transforming the observations to alternative representations. The alternatives are the following: Clipping; Resizing; Normalizing; Stacking buffers; Grayscaling images; Simple motion blur simulation.
- rewardwrappers.py: Responsible for defining reward functions or performing reward transformations.

We used the following options in the configuration file to use different wrappers:

- Action: **heading; heading_smooth**: These options change the heaviness of an action. If we use *heading_smooth* the influence of the action is the cube of the influence if we used the *heading* flag.
- Observ: **grayscale on; grayscale off**: If we set the *grayscale* flag to *true* the processed image frame is converted to grayscale otherwise the processed image frame is left untouched.
- Reward: **posangle; lane_distance**: If we use the *posangle* option, the network is rewarded the most if the simulated robot faces towards the center of it's lane. If we used *lane_distance*, the wrapper gave positive reward directly proportional with the travelled distance.

For training this environment uses the *Proximal Policy Optimization (PPO)* algorithm, because of it's stability and ability to take advantage on multiple parallel workers.

III. Hyperparameter Optimization

The result of the training depends on the configuration of hyperparameters. The values of these parameters can be set manually, however it's more accurate to use an algorithm that chooses the best values through optimization.

A. The algorithm

We chose *Ray*'s *Population Based Training (PBT)* algorithm [4] to optimize hyperparameters. PBT runs multiple trial variants and creates checkpoints of their structure at predefined intervals. Comparing the trials PBT then clones the top-performing trial's values into the other trials and perturbs their

hyperparameters in a specified range (Table 1) to find a better configuration.

B. Hyperparameters

We chose the following parameters for optimization:

- lambda: PPO uses *Generalized Advantage Estimation (GAE)* [5] to calculate the advantage of each action during training, *lambda* is a smoothing factor which makes the training more stable by reducing GAE's variance [6].
- KL coefficient: PPO uses *Kullback–Leibler (KL)* divergence to calculate the penalty of the system during training, *KL coefficient* is the initial weight given to calculate the divergence.
- KL target: it is the target KL divergence that the system tries to reach.
- learning rate: it is the coefficient that determines how much the calculated loss should modify the weights of the network.
- entropy coefficient: during training the maximum possible value of entropy of different actions are added to loss to prevent one action from dominating the system in an early phase. *Entropy coefficient* determines how much we want the entropy to modify the loss.

TABLE I: Hyperparameter perturbation ranges

Hyperparameter	Perturbation Range
lambda	(0.9, 1.0)
KL coefficient	(0.9, 1.0)
KL target	(0.003, 0.03)
learning rate	(0.00001, 0.001)
entropy coefficient	(0, 0.01)

Hyperparameters and their range were selected based on Schulman et al. [7].

IV. Training

Eight networks with different configuration were trained each for 50 iterations with 2 trial variants, we chose these numbers because a single training took for several hours and our goal was to compare the configurations rather than creating a strong model (Figure 1).

V. Testing

We ran the test script from the original project on all eight neural networks. The script runs a real-time simulation of a duckie-bot on five different circular maps where the bot is controlled by the loaded network (Figure 2). The test on a map have two types of ending, the bot is drove into a wall or it survives for longer than the preset time limit which we chose to be 60 seconds.

Not surprisingly all network resulted in a crash within 60 seconds as we didn't train them long enough, but we could still use the results to compare them. To measure the results of a network we observed the *mean survival time (MST)* and *mean distance travelled (MDT)* on the five test maps (Table II).

Based on the results the best config had *posangle* as reward function *heading* as action wrapper and *grayscale* was

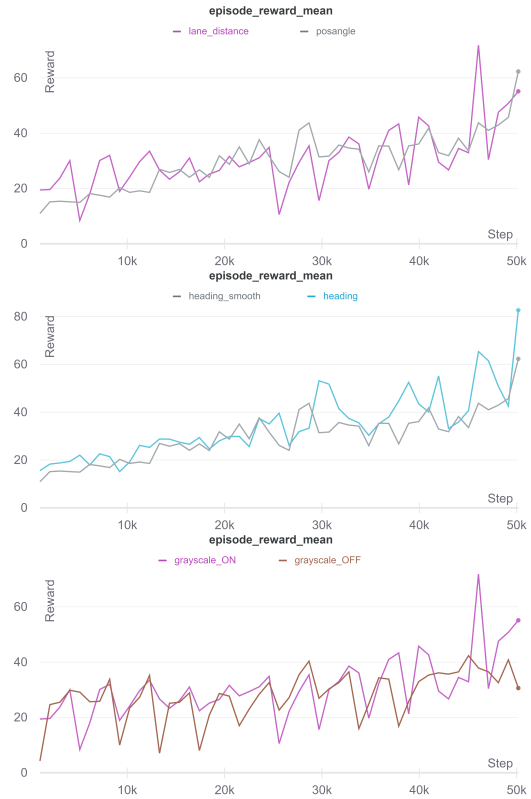

Fig. 1: Graphs showing the difference of mean reward between configurations during training: reward functions (top), action wrappers (middle), observation wrappers (bottom)

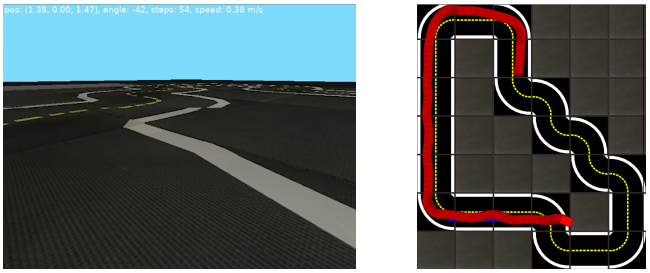

Fig. 2: Running test in the simulated environment (left), visualised test result (right).

turned on in it's observation wrapper. To test whether this configuration could achieve better results if trained for longer we set it's hyperparameters' values to the ones PBT gave as optimal values, we restarted the training and this time ran it for 100 iterations and without hyperparameter optimization (Figure 3).

Running the same test on this final network resulted in *mean survival time* of 18.78 seconds and *mean distance travelled* of 7.823 distance units.

VI. Future Plans

We have many opportunities to develop this project further. The most obvious of these seems to be the implementation of neural networks, which are emerging year after year and offer the highest overall accuracy, so that the vehicle travelling on

TABLE II: Test results

RF	**AW**	**OW**	**MST** (s)	**MDT** (unit)
lane_distance	*heading*	*grayscale on*	4.753	1.835
lane_distance	*heading*	*grayscale off*	2.9	1.2
lane_distance	*heading_smooth*	*grayscale on*	3.373	1.553
lane_distance	*heading_smooth*	*grayscale off*	1.26	0.555
posangle	*heading*	*grayscale on*	8.287	3.443
posangle	*heading*	*grayscale off*	8.46	3.606
posangle	*heading_smooth*	*grayscale on*	3.327	1.552
posangle	*heading_smooth*	*grayscale off*	2.227	0.976

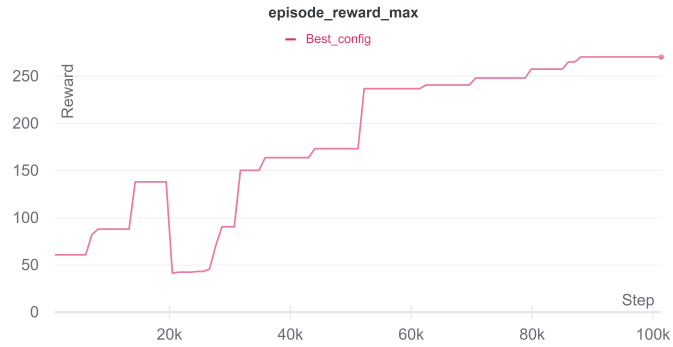

Fig. 3: Change of maximum reward during the training of the best model.

the test track encounters the fewest obstacles or passes through them with the greatest precision [8] and possibly the fastest [9]. The project could be taken to a new level by modelling the simultaneous movement of several vehicles in a more complex environment.

The DuckieTown project is a general research and education project around self-driving cars, so to take this further, an option has been developed to produce a gradient-based [10] activation HeatMap [11] by counting back the output of the network layers to the input frame. Which would serve the purpose of allowing us humans to determine which image region in the original image caused the highest output value in the neural network used.

VII. Summary

In this work, we learned how hyperparameter optimization can be used to evolve neural networks using wrappers to track a band in both the virtual and real environments of DuckieTown. The training and the networks were constructed using the PBT model [12] and achieved end-to-end lane tracking by vehicles. Different wrappers allowed us to use different agents and then we evaluated the results of these.

References

[1] K. Krinkin, K. Chayka, A. Filatov, and A. Filatov, "Autonomous wheels and camera calibration in duckietown project," *Procedia Computer Science*, vol. 186, pp. 169–176, 2021, 14th International Symposium "Intelligent Systems. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050921009509

[2] L. Paull, J. Tani, H. Ahn, J. Alonso-Mora, L. Carlone, M. Cap, Y. F. Chen, C. Choi, J. Dusek, Y. Fang, D. Hoehener, S.-Y. Liu, M. Novitzky, I. F. Okuyama, J. Pazis, G. Rosman, V. Varricchio, H.-C. Wang, D. Yershov, H. Zhao, M. Benjamin, C. Carr, M. Zuber, S. Karaman, E. Frazzoli, D. Del Vecchio, D. Rus, J. How, J. Leonard, and A. Censi, "Duckietown: An open, inexpensive and flexible platform for autonomy education and research," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 1497–1504.

[3] A. Kalapos, C. Gor, R. Moni, and I. Harmati, "Sim-to-real reinforcement learning applied to end-to-end vehicle control," *2020 23rd International Symposium on Measurement and Control in Robotics (ISMCR)*, Oct 2020. [Online]. Available: http://dx.doi.org/10.1109/ISMCR51255.2020.9263751

[4] V. Dalibard and M. Jaderberg, "Faster improvement rate population based training," 2021.

[5] A. Li, O. Spyra, S. Perel, V. Dalibard, M. Jaderberg, C. Gu, D. Budden, T. Harley, and P. Gupta, "A generalized framework for population based training," 2019.

[6] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," 2018.

[7] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017.

[8] A. Kalapos, C. Gór, R. Moni, and I. Harmati, "Vision-based reinforcement learning for lane-tracking control," *ACTA IMEKO*, vol. 10, no. 3, pp. 7–14, 2021.

[9] Thiang, G. R. Purwanto, P. Santoso, and H. Khoswanto, "Autonomous mobile robot development based on duckietown platform for recognizing and following the traffic sign," *Journal of Physics: Conference Series*, vol. 1921, no. 1, p. 012065, may 2021. [Online]. Available: https://doi.org/10.1088/1742-6596/1921/1/012065

[10] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.

[11] J. Kim and J. Canny, "Interpretable learning for self-driving cars by visualizing causal attention," 2017.

[12] M. Jaderberg, V. Dalibard, S. Osindero, W. M. Czarnecki, J. Donahue, A. Razavi, O. Vinyals, T. Green, I. Dunning, K. Simonyan, C. Fernando, and K. Kavukcuoglu, "Population based training of neural networks," 2017.