
Reinforcement learning based self-driving vehicles in the Duckietown environment

Megerősítéses tanulás alapú önvezető járművek a Duckietown környezetben

Attila Pethő

Olivér István Farkas

Gyula Faragó

Abstract

In this project we tried to bring a solution to driving a small robot with vision-based, end-to-end reinforcement learning in a simulated environment. The robot relies on images from its front-facing camera. Our models generate continuous actions that control a differential-drive robot vehicle. To train the policy we have selected two state-of-the-art algorithms (A2C and PPO). A2C is an Actor Critic model, while PPO uses Proximal Policy Optimization. In order to train these models we have to feed in the images, which we preprocess first. To train the model better for real-world conditions we used domain randomization. Further analysis of our work is discussed in the document.

Kivonat

Ebben a projektben megpróbáltunk megoldást találni egy kis robot vezetésére látásalapú megerősített tanulással egy szimulált környezetben. A robot az elől elhelyezkedő kamerájának képeire támaszkodik. Modelljeink folyamatos cselekvéseket generálnak, amelyek egy differenciálhajtású robotjárművet irányítanak. A stratégia betanításához két modern algoritmust választottunk (A2C és PPO). Az A2C egy Actor Critic modell, míg a PPO a Proximal Policy Optimizationt használja. A modellek a környezettől kapott képkockák alapján tanulnak, amelyeket először előfeldolgozunk. Ahhoz, hogy a modell valós környezetbe átültetését elősegítsük, domain randomizálást használtunk. Munkánk további elemzését a dokumentumban tárgyaljuk.

1 Introduction

Self-driving vehicles are clearly the future of transportation. If you look at Waymo's [1] safety report you can see two things. First, it is obvious that AI driven vehicles are more safe than human driven ones. The other thing which - makes development hard - is the amount of data needed for the training of the AI. The report mentions that the AI used in the cars have learned from over 20 million driven miles. It is quite a big number and an astounding achievement given the fact that the car fleet only had 8 bigger accident during this time. The report shows that all accident had a similar cause, human error. In three times the car was stationary and someone ran into it. This clearly shows the potential of self driving vehicles.

In the field of self-driving cars, reinforcement learning is a very promising technique. Training self-driving agents in an end-to-end fashion is getting more and more attention among researchers. For example Minh et al. [2] proposed a vehicle-controller policy, that operates with discrete actions, based on a forward-facing camera on the robot. Although Kendall et al. [3] demonstrated real-world driving with a lane-following policy on a real vehicle, a safety driver was needed for supervision. Training models on real vehicles raises both safety and financial reasons, so a better solution is to experiment in a simulator. One example is the work of Jaritz et al. [4], who used the World Rally Championship 6 racing simulator to train their vision-based road-following policy. Balaji et al. [5] also presented a method for training vehicle-controller policies in a simulator, but their primary platform was the DeepRacer platform. Our choice was the Duckietown simulator, which

is a fully-functioning autonomous driving simulator that can be used to train and test different vehicle-controller algorithms [6].

2 Methods

2.1 Reinforcement Learning

In a reinforcement learning setting, an agent interacts with its environment by choosing actions (a_t) at every timestep, based on information called state (s_t), and receives back the next state (s_{t+1}) and the reward for the current action (r_{t+1}), which will be used to select the next action (a_{t+1}) [7]. Figure 1 summarizes this process. The agent tries to gradually improve its policy by exploring new ways to behave in the environment and observing the rewards received for its actions. The goal of the agent is to maximize the expected future rewards over a finite amount of steps. There are many ways of achieving this goal, which is reflected in the number of different reinforcement learning algorithms. A fairly simple one is Deep Q-Networks (DQN) [8] which chooses actions based on an estimate of the action-value function that is calculated from previous steps in the environment. Although it has been used in a similar setting [9], DQN would not be the best choice for this type of problem, because it can only utilize discrete actions, which could potentially cause wobbling and unstable behaviour [4], [10]. For this reason, we chose two algorithms, Proximal Policy Optimization (PPO) [11] and a synchronous, deterministic variant of Asynchronous Advantage Actor Critic (A2C) [2], which both support continuous actions.

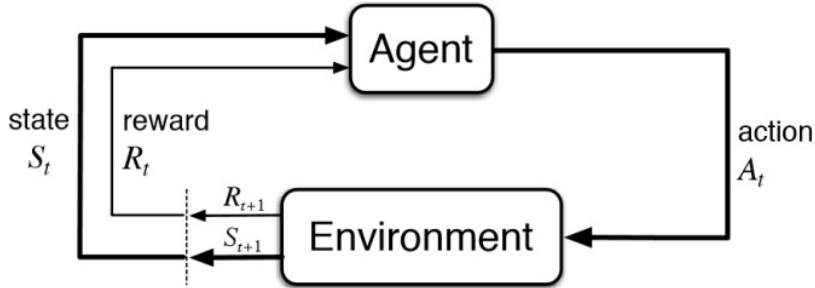


Figure 1: Illustration of Reinforcement Learning

Implementing the above mentioned algorithms would be very time consuming, so we decided to use the Stable Baselines 3 library [12], which is a simplified version of OpenAI baselines. The SB3 library contains several implementations of state-of-the-art RL algorithms and useful tools to train, retrain and evaluate the models.

2.2 Environment

In each step, the environment gives us a 640x480 RGB image. We have to preprocess this image, because feeding the original images to the CNN would be a waste of resources because it makes the training process much slower, and the network can learn from smaller images just as well.

2.2.1 Observations

Preprocessing steps are applied to the observations. These included resizing the image from 640x480 pixels to 84x84, cropping the top 24 pixels of the image (since this only shows the sky), grayscaling the image, or optionally segment the colors in it, and finally stack the last 3 frames. The preprocessing steps were implemented using wrappers, that were applied to the environment layer-by-layer.



Figure 2: (a)



Figure 3: (b)

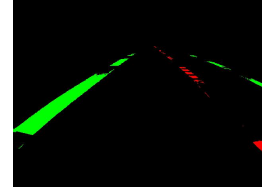


Figure 4: (c)

Figure 5: Observations from the environment. (a) shows a raw observation, (b) shows a grayscale image, (c) shows a color segmented image

2.2.2 Actions

The Duckiebots are two-wheel differential-drive vehicles, and the default actions in the simulator are interpreted as a list of the left and the right wheel angular velocities, where 1 and -1 correspond to the maximum angular velocities forward and backward. This continuous action space has left and right velocity pairs that are not particularly useful (e.g. low velocities), so we experimented with two new action representations, introduced by Kalapos et al. [13].

Wheel velocity with braking: Since negative velocities are not required for moving forward, these are clipped, and the predicted values are interpreted as the amount of braking from full speed at each wheel. This creates a bias for moving forward at full speed.

Steering: Instead of predicting a vector containing the two wheel velocities, the agent predicts a scalar value between -1 and 1, where 0.0 represents going forward at full speed, and -1 and 1 correspond to turning left or right with one wheel completely stopped and the other spinning at maximum speed. Intermediate actions are continuously mapped to wheel velocities using linear interpolation. In this case the speed is always maximal for a given steering value.

2.2.3 Rewards

Reward shaping is essential in RL, since this is the basis for the agent’s learning. Without a good reward function the agent will maximize its score, but won’t learn the intended behaviour. The goal of the robot should be to move forward in the right lane as fast as possible. To encourage this behaviour we used a reward function [13] that considers both the orientation and the velocity of the Duckiebot. The *orientation* reward is based on the lateral position of the robot, so it should always face towards the center of the right lane, while the *velocity* reward is given based on the fastest moving wheel to encourage high velocity in both straight and curved sections.

2.3 Evaluation

There are many ways to measure the performance of the trained agents, and for the sake of simplicity we chose two metrics for this purpose: survival time (in timesteps) and rewards received from the environment. Survival time tells us how many steps the agent took before either it went out of the road or the episode finished. A well-trained agent should stay on the road for the whole evaluation episode. Although, this provides valuable information, this metric does not tell much about performance on its own. Hence, the rewards are also stored and evaluated to gain richer knowledge about the agents performance. The models were trained on the ‘zigzag_dists’ map, because it offers a good balance between left and right turns and straight sections. In order to test how well the models learned to generalize, evaluation was done on three different maps: ‘zigzag_dists’, ‘loop_empty’, and ‘udem1’.

2.4 Hyperparameter Optimization

Reinforcement learning models are extremely sensitive to hyperparameter settings. A poor choice of hyperparameters can cause unstable performance or it could prevent convergence completely. Finding the right parameters for training can be a difficult job, and it is usually not a good idea to start tuning the parameters manually. Instead you can use an optimization library, that sets up sophisticated experiments to find the right hyperparameters for your usecase.

For our project we used the Optuna library, which offers several tools to dynamically explore the hyperparameter-space in a define-by-run style. The library features state-of-the-art algorithms for sampling and pruning, easy parallelization of experiments and visualization tools. In order to optimize our models, we used the Tree-structured Parzen Estimator (TPE) for sampling and the Median pruner for early termination of less promising trials. The studies were conducted with a budget of 100 trials and a maximum of 50000 steps, and evaluations at every 10000 steps. For optimizing the PPO model, we used an RTX 6000 workstation with 24GB VRAM and 46GB RAM, but for the A2C study, we had to use a GTX 1060OC with 6GB of VRAM and 16GB RAM which was not enough to finish the study, so in that case the results are based on 47 finished trials. Tables 1(b) and 1(a) shows the optimized hyperparameters for the models.

Learning rate schedule	constant
Learning rate	0.1324
Norm. advantage	False
n_steps	64
Entropy coef.	0.0015
Use RMS prop.	False
Gae. lambda	0.98
Max gradient norm.	0.7
vf coef.	0.7373
Activation	tanh

1(a) A2C

Learning rate schedule	linear
Learning rate	0.8764
Batch size	256
n_steps	32
Entropy coef.	6.5699
Clip range	0.2
Epochs	5
Gae. lambda	0.92
Max gradient norm.	0.8
vf coef.	0.2425
Activation	relu

1(b) PPO

Table 1: Optimized model hyperparameters

3 Results

The PPO and the A2C agents were trained in the gym-duckietown simulator, with concatenating the last 3, grayscale observations, and using the *steering* actions and the *orientation* and *velocity* rewards described in subsubsection 2.2.2 and subsubsection 2.2.3. The agents were trained on the 'zigzag_dists' map for one million steps, using only 1 environment. Each episode lasted a maximum of 500 steps, because this proved to be enough to take at least a full round on the map for a reasonable agent. In order to examine the effects of the hyperparameter optimization, first we trained the models with mostly the default SB3 parameters, the only exception was the learning rate, which was set to a constant 0.0005 value. This was ment to be a baseline training to compare it with the optimized models. The main difference - besides the optimized parameters - was the use of 4 parallel environments and the 2 million steps long training. Tables 2, 3 and 4 show the evaluation results on the different maps.

Table 2: Evaluation results on the 'zigzag_dists' (seen) map

	PPO (default)	A2C (default)	PPO (optimized)	A2C (optimized)
Mean episode reward	11.51	74.24	128.19	74.24
Mean episode length	71.5	308.0	452.9	308.0

Table 3: Evaluation results on the 'loop_empty' (unseen) map

	PPO (default)	A2C (default)	PPO (optimized)	A2C (optimized)
Mean episode reward	14.83	61.21	93.78	61.21
Mean episode length	69.8	219.5	311.6	219.5

Table 4: Evaluation results on the 'udem1' (unseen) map

	PPO (default)	A2C (default)	PPO (optimized)	A2C (optimized)
Mean episode reward	6.01	98.27	70.79	98.27
Mean episode length	38.2	359.6	262.7	359.6

The evaluation results show us, that in the case of the PPO model, hyperparameter optimization significantly helped achieving higher rewards and longer runs, both in previously seen and unseen environments. But regarding the A2C model, the optimization seemingly did not help at all. It is unclear why this has occurred, which means further examination of the training and evaluation conditions are needed for the A2C agent. Another interesting result to consider, is the fact that the A2C model performed better on a map that it has never seen, which is also a more complex version of the 'loop_empty' map, with intersections and previously unseen objects like houses and vehicles in the background. Furthermore, the A2C model achieved much better results with its "default" hyperparameters than the PPO, which could mean it needs less optimization trials.

4 Conclusion

In this work two reinforcement learning algorithms were trained to drive a two-wheel robot in a simulated environment. The models were compared with each other and their optimized versions using basic metrics from the simulation. It was found that optimization did help regarding the PPO model, but the A2C agent performed the same way. In order to understand the reasons behind this phenomena more in-depth experiments are needed. To further improve the results, longer optimization trials are necessary, with at least 100000 steps, and a variety of maps to train on, which can improve generalization and help avoid overfitting to a single map. More evaluation methods could be used to elaborate further on the models performance, like transversal and angular deviations from the right lane centerline, or distance travelled in the ego lane.

References

- [1] W. LLC, "Waymo safety report," 2021. <https://waymo.com/safety/>.
- [2] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," *CoRR*, vol. abs/1602.01783, 2016.
- [3] A. Kendall, J. Hawke, D. Janz, P. Mazur, D. Reda, J.-M. Allen, V.-D. Lam, A. Bewley, and A. Shah, "Learning to drive in a day," 2018.
- [4] M. Jaritz, R. de Charette, M. Toromanoff, E. Perot, and F. Nashashibi, "End-to-end race driving with deep reinforcement learning," *CoRR*, vol. abs/1807.02371, 2018.
- [5] B. Balaji, S. Mallya, S. Genc, S. Gupta, L. Dirac, V. Khare, G. Roy, T. Sun, Y. Tao, B. Townsend, E. Calleja, S. Muralidhara, and D. Karuppasamy, "Deepracer: Educational autonomous racing platform for experimentation with sim2real reinforcement learning," *CoRR*, vol. abs/1911.01562, 2019.
- [6] "Gym-duckietown," 2021. <https://github.com/duckietown/gym-duckietown>.
- [7] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. 2018.
- [8] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013.
- [9] P. Almási, R. Moni, and B. Gyires-Tóth, "Robust reinforcement learning-based autonomous driving agent for simulation and real world," *CoRR*, vol. abs/2009.11212, 2020.
- [10] B. Osinski, A. Jakubowski, P. Milos, P. Ziecina, C. Galias, S. Homoceanu, and H. Michalewski, "Simulation-based reinforcement learning for real-world autonomous driving," *CoRR*, vol. abs/1911.12905, 2019.
- [11] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017.
- [12] "Stable baselines docs," 2021. <https://stable-baselines.readthedocs.io/en/master/index.html>.

- [13] A. Kalapos, C. G3r, R. Moni, and I. Harmati, “Sim-to-real reinforcement learning applied to end-to-end vehicle control,” *arXiv:2012.07461*, 2020. <https://github.com/kaland313/Duckietown-RL>.