International Journal of Applied Engineering Research ISSN 0973-4562 Volume 14, Number 23 (2019) pp. 4325-4328

A Simulator for Training Path-Following Agents Using Reinforcement Learning

Jo Myoung Kang[1], Hyunsuk Chang[2], Byung Joon Park[1*]

[1]Dept. of Computer Science, Kwangwoon University, Seoul 01897, Korea.

[2]MARUSYS, Seoul 06222, Korea.

[1] Corresponding Author, Dept. of Computer Science, Kwangwoon University, Seoul 01897, Korea.*

Abstract

This paper presents a software simulator for training path-following agents using deep reinforcement learning for energy-efficient autonomous driving. The simulator provides a graphical interface with which the user observes how the agent behaves while the hyper parameters or the reward mechanism vary. The simulator is developed with Unity, a 3D editor and the reinforcement learning mechanism was incorporated into the system using Unity's Machine Learning agent. Image data obtained from Camera Object was saved as a rendered texture to provide the input data for learning. We also discuss experimental results obtained from training a path-following agent with the implemented simulator.

Keywords: Path-following, reinforcement learning, simulator, autonomous driving

1. INTRODUCTION

Path-following is a key component in autonomous driving technology. Autonomous driving is one of the best-known applications of artificial intelligence and is making a lot of progress. Autonomous driving technology is being used not only for cars but also for unmanned aircraft such as drones. However, it costs a lot to test self-driving in a physical environment with a real vehicle. This paper describes a simulator that enables a certain agent to learn to move along a given path using reinforcement learning. Through a graphical interface, it is possible for a user to observe how the agent behaves differently when the hyper-parameter or reward method of reinforcement learning varies. If path-following capability can be acquired from training in a simulated environment, it could save both the physical resources and efforts. In this paper, we present a Unity environment where we could train path-following agent for autonomous driving with deep reinforcement learning. The input data used to train the path-following agent the image data received from the camera object of agent.

In Section 2, we provide some background related to our work such as autonomous driving techniques and deep reinforcement learning to be implemented in this paper. We also examine Unity, the environment with which we implement the simulator. We describe the actual implementation of the simulator in section 3, the simulation environment, and its learning agent. Section 4 describes the hyper-parameters set-up for deep reinforcement learning and how experiments were conducted. Section 5 examines analyzes the agent's learning process and the result of experiments. Section 6 draws a conclusion and discuss future work.

2. RELATED WORK

2.1. Path Following

There are several possible ways to build an agent that can follow a given path for autonomous driving technology. One simple method is to clone the motion of a car running in front [1]. Path-following is carried out by cloning the movement of the vehicle in front using the vehicle's camera. Another way is to utilize the image data from the agent's camera. For example, by taking the picture below as an input, a path-following agent has to go along the white road. In Figure 1, the blue circles represent the center of the camera and red circles represent the center line of the road. The path-following agent tries to narrow the difference between the blue and red circles by issuing a suitable directional command. Repeatedly going through this process, the vehicle moves to match its center with the center of the road. Yet another way to build a path-following agent it to train it with some learning mechanism such as reinforcement learning. The image data entered the camera will be input to learning. Using machine learning method requires a large amount of data to be covered and a lot of trial and error is required to proceed. Therefore, it is not possible to experiment with a real vehicle, and the results of learning through simulation can be used to apply it to a real vehicle. Later in this paper, we discuss our implemented simulator for conducting in deep reinforcement learning in the Unity environment.

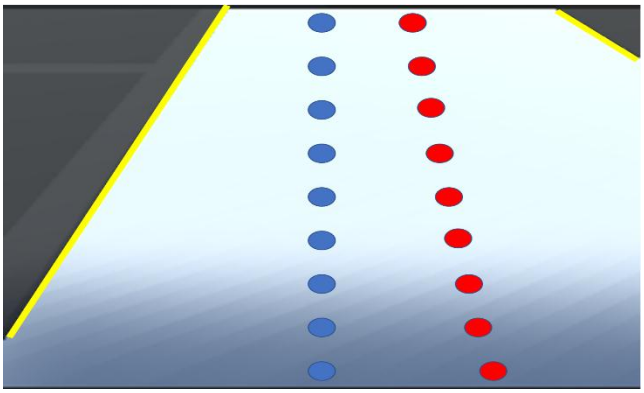

Figure 1. An example of creating a autonomous driving algorithm based on image data

2.2. Deep reinforcement learning

Reinforcement learning is a branch of machine learning. Reinforcement learning consists of states that are utilized as input data, agents that take appropriate actions using data, and rewards that are expected to be obtained if action is taken. Given a state, there are possible actions that can be taken by the agent, and each action has a different reward. The agent then acts in a way that maximizes the value of the return that can be obtained. This process creates a function of state, action, and reward, and the final purpose of the reinforcement learning is to find this function. However, the size of the input data becomes very large when using reinforcement learning in real-life situations. An example is the study result of applying reinforcement learning to Atari games conducted at Deepmind.[2] If you look at the game 'Breakout' among Atari games, the image data from game screen is 84×84 pixel data. Since it is almost impossible to create a function to process input data of this size, a method for processing input data using the neural network has been devised. Also, using neural network, data on the screen can be used as raw without processing. This is the basis of deep reinforcement learning. Deep reinforcement learning has been an active area of research[3].

2.3. OpenAI Gym

OpenAI Gym[4] is a Python package created by OpenAI, a non-profit organization. It supports many test environments compared to existing packages for enhanced learning. These include Atari's games and simulations of the 3D environment using the MuJoCo physics engine.

2.4. Unity ML-Agents

Machine Learning Agents Toolkit[5] is a tool for processing reinforcement learning provided by Unity. There are agents controlled by several brains, which allow the agent to take appropriate actions based on state and environment. Even a user with a shallow knowledge of reinforcement learning can utilize the learning mechanism by following the provided samples if he or she is familiar with the Unity environment. Users can also speed up the learning process because it can do multiple parallel processing. The ML-Agents Toolkit consists of two packages. The Unity SDK is a package used in the Unity Editor that is included in the Unity Editor's work environment to help set up agents and states and create learning environments. Another package is a Python package. In Unity, when information about the state's data is passed to python, it uses this to learn and deliver the appropriate command to the agent. The process of learning utilizes Tensorflow. All of these packages are open-source and anyone can access the github package if necessary[6]. ML-agents SDK consists of agent, brain and academy. The agent collects the data of the environment that acts as a state to perform the learning. The collected data is then used by the brain to give proper instructions. Brain can be used in many ways: internal, heuristic, and external. These brains are managed in one academy. ML-agents are available in the Unity Editor. The Unity Editor is optimized for creating programs in 3d environments and is widely known for making games, but recently Unity's development team has been very helpful in creating simulations for research, such as ML-agents.. In addition, programs such as Airsim[7], which help create driving simulations such as unmanned aircraft, are also open source, helping researchers. In addition, physical expressions are not difficult to implement in the Unity environment. Physical phenomena such as the mass of objects and the movement of wind can also be implemented in Unity. For this reason, the simulation was created using Unity ML-agents.

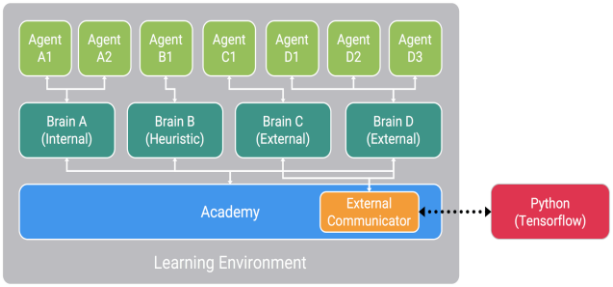

Figure 2. Composition diagram of Unity ML-Agents

3. IMPLEMENTATION

3.1. Environment

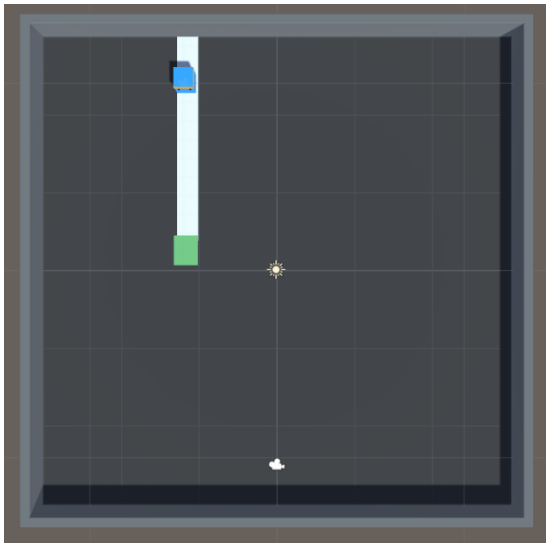

Figure 3. The configuration of the environment for training

The environment for the experiment was implemented as shown in the picture above. First, the space for the experiment was implemented in the form of a box without a ceiling, and the physical part was eliminated in the bottom part. Using this, if the agent, which looks like a blue small box, escapes from the road represented by a white band, it will fall through the bottom of the box. In this case, it is recognized that the learning has failed, and the position is initialized again to continue the learning. And at the end of the course, we made a goal point in green. When it reaches this point, it recognizes that the agent has properly followed the path, rewards it, and resumes learning at the initial location.

3.2. Agents

The learning agent is implemented as in Figure 4. The agent has one camera. Views on the camera can be viewed in the lower right corner of the picture. Agent's behavior was most simplified, making it a forward, right, and left turn. The image shown on the agent's camera is saved by the unity editor to a file of a 32×32 pixel rendered texture.

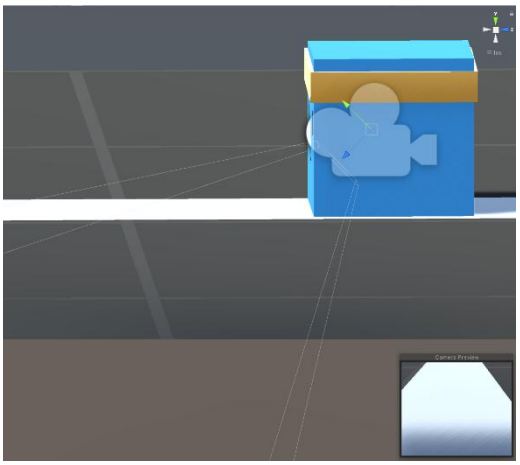

Figure 4. Agent with one Camera object

4. DESIGN OF EXPERIMENT

The agent's goal is to earn extra rewards when he reaches the green goal while driving continuously without getting off the white path. The agent recognizes the path using only the rendered texture data obtained from the camera, analyzes the data, and uses ML-agents to learn. In the course of learning or when learning is complete, a * .nn file is created. You can use this file to see how well the agent is learning. Reapplying this file to the simulation ensures that it is well trained and follows well. In addition, if you use Tensorboard of Tensorflow in the process of learning, you can check the value of mean reward etc. during the learning process.

The version of Unity used in the experiment is Unity 2017 4.33.f1, and the experiment was conducted in the environment of Tensorflow 1.7 and ML-agents 0.11. In addition, the main hyper parameters used to actually do the learning in Python are shown in the table below. Initially, the number of steps was set up to 2,000,000 times. But as the value of mean reward can be checked in real time through the console in the course of the learning process, if it is deemed to have been learned properly, the learning was stopped and we tested with the learned neural network

Table 1. the hyper parameter used in the experiment.

hyper parameter	value
algorithm	PPO
size of batch	1024
num of epochs	3
num of layers	2

5. EXPERIMENTAL RESULTS

In the first production environment, 50,000 move decisions were executed. After 50,000 commands were executed according to the received image data, the agent continued to move off the road. This is a behavior that does not get any better than when commanded randomly. In our initial experiment, we set a reward of 50 when the agent reaches the goal, and -5 when the agent falls off the road. Figure 5 shows the mean rewards obtained during learning. The average of rewards does not increase or change significantly, indicating that the experiment needs to be improved.

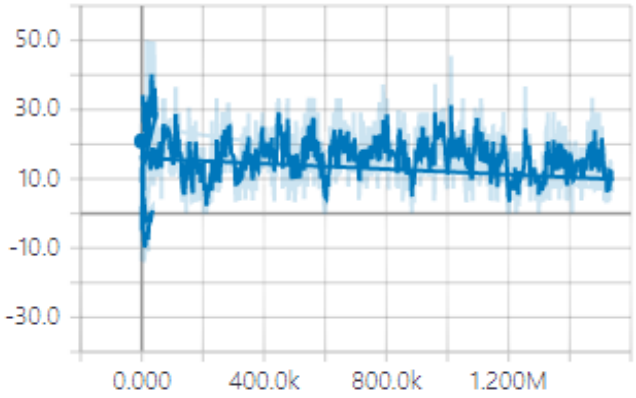

Figure 5. Mean rewards of the agent during learning

Increasing the number of steps doesn't seem to improve the result. So the user can experiment with another rewarding scheme such that if agent stays on the road without falling, it receives a reward of 0.01. In this situation, the agent was expected to act to keep himself from falling off the road.

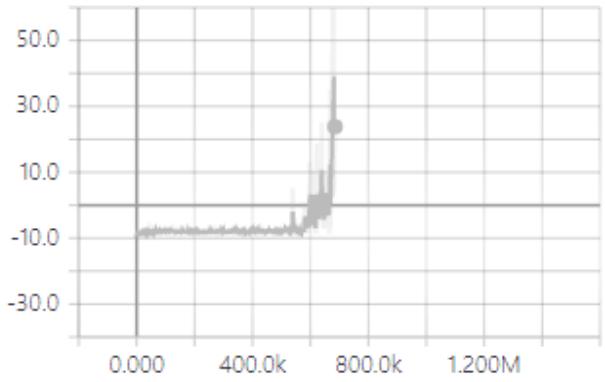

Figure 6. Mean rewards when the agent receives some rewards for not falling off the road

The graph suddenly shows an exponential increase in compensation because it was only aimed at keeping things from falling.The results of the experiment are as shown in the graph above. At first, the experiment appeared to be problematic because the average of rewards did not increase, but over time the average value increased. But after testing in Unity using

these learned neural network data, the agent, whose goal is not to get off the road, took a swirling motion instead of going for goal. It seems to have chosen that action because the agent can get enough reward even if it doesn't get off the road.

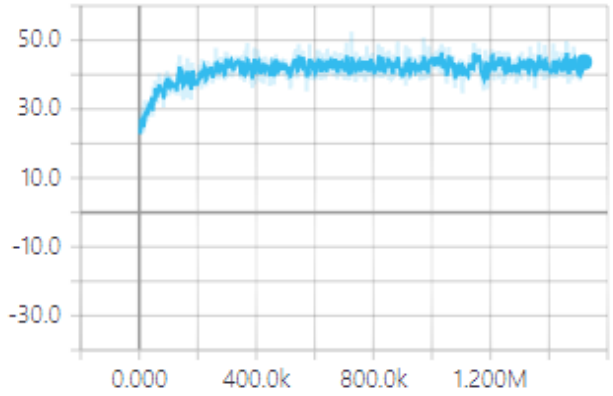

Figure 7. Graphs of normal experimentation. Too many steps resulted in an error in over-fitting

.

Next, we experimented with another rewarding scheme that gives a negative reward of -0.01 to the agent for every action it makes without just giving a reward for the agent's keeping itself on the road. This setting causes the agent to aim to reach the goal with fewer actions. The above graph shows the rewards of the results of the learning process. This graph seems to have been properly learned, but it has too many repeated steps a great number of times. The experiment result shows some sign of over-fitting. In testing at Unity, the agent has already learned that he can reach his goal by going straight because he has only learned to move in a straight line. Thus, no matter how the image data came in, only straight ahead was seen as a action. We applied a little variation to the course and confirmed that we only gave direct orders even though we changed the target position. This could be done by adjusting the size of the steps low.

6. CONCLUSION

In this paper, we have described the process of producing simulations essential to the development of autonomous driving technology and have produced simulations that can proceed with deep reinforcement learning. In the course of in-depth reinforcement learning using Unity ML-agents, even researchers with relatively little knowledge of Unity, physical effects, or reinforcement learning were able to create and study the environment of reinforcement learning relatively easily. In addition, the reinforcement learning was conducted using visual-based image data rather than simple input values used in existing reinforcement learning examples. There have been many trials and errors in the process of conducting the experiment. In the process of going through these trials and errors, the users had to experiment with adjusting reward or the value of the hyper parameters to achieve good results.

The simulation implemented in this paper was an agent that could follow a very simple form of path. If a user wants to experiment with different types of path or environment, he or she could create an agent that can follow more different forms of pathways. There are also ways to create and provide a variety of courses directly, and the method to use a random generator may be used. It is also possible to take full advantage of the Unity environment to be created in a more sophisticated environment. Next, it is possible for ML-Agents to determine which commands to issue based on given data. And if the method implemented in this paper utilizes the rendered texture, the information coming from the external camera, rather than the simulated environment, will also be available for analysis. This allows the simulation to take the learned model and give the image received from the camera of the actually moving vehicle as the input data. Then it can send the resulting commands back to the actual vehicle. In this way, image-based autonomous driving can be applied to actual vehicle based on data learned from the simulations. To do this, the vehicle to be applied and the object to be used in the simulations should be made to have a similar degree of behavior.

ACKNOWLEDGMENT

This work was supported by "Human Resources Program in Energy Technology" of the Korea Institute of Energy Technology Evaluation and Planning (KETEP), granted financial resource from the Ministry of Trade, Industry & Energy, Republic of Korea. (No. 20194010201830). The present research has been conducted by the Research Grant of Kwangwoon University (No. 2017-0264).

REFERENCES

[1] Torabi, F., Warnell, G., & Stone, P. (2018, July). Behavioral cloning from observation. In Proceedings of the 27th International Joint Conference on Artificial Intelligence (pp. 4950-4957). AAAI Press.

[2] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602.

[3] Arulkumaran, K., Deisenroth, M. P., Brundage, M., & Bharath, A. A. (2017). Deep reinforcement learning: A brief survey. IEEE Signal Processing Magazine, 34(6), 26-38.

[4] Gym[website]. Retrieved from https://gym.openai.com/

[5] Juliani, A., Berges, V., Vckay, E., Gao, Y., Henry, H., Mattar, M., Lange, D. (2018). Unity: A General Platform for Intelligent Agents. arXiv preprint arXiv:1809.02627.

[6] Unity-Technologies/ml-agents[website]. Retrieved from https://github.com/Unity-Technologies/ml-agents.

[7] Shah, S., Dey, D., Lovett, C., & Kapoor, A. (2018). Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In Field and service robotics (pp. 621-635). Springer, Cham.