Bachelor's Thesis

A Study of Reinforcement Learning for Self-driving RC Car using AWS DeepRacer and Unity ML-agent

School of Mechanical and Control Engineering

Handong Global University

Do-Yeon Kim

Joo-Ho Kim

Ye-Chan Song

A Study of Reinforcement Learning for Self-driving RC Car using AWS DeepRacer and Unity ML-agent

A Bachelor's Thesis

Submitted to the School of

Mechanical and Control Engineering of

Handong Global University

Do-Yeon Kim

Joo-Ho Kim

Ye-Chan Song

June 2021

This certifies that the bachelor's thesis is approved.

_____

Thesis Advisor: Ph.D. Yeong-Keun Kim

_____

The Dean of Faculty: Ph.D. Jong-Sun Lee

School of Mechanical and Control Engineering

Handong Global University

June 2021

Extended Abstract

A Study of Reinforcement Learning for Self-driving RC Car using AWS DeepRacer and Unity ML-agent

Recently, all industries and all academic fields use the word artificial intelligence. The words artificial intelligence and deep learning are drawing attention around the world. Especially, in 2016, Google's DeepMind launched an artificial intelligence called AlphaGo, which shocked people around the world. DeepMind also recognized that there is no entirely human field by creating machines that surpass human abilities with Go. Through this incident, people became very interested in artificial intelligence, especially reinforcement learning. Due to the increased interest, many reinforcement learning algorithms are published and studies are actively underway to apply them to the real environment. Reinforcement learning is a type of AI machine learning inspired by behavioral psychology. It is a method by which an agent defined in a given environment recognizes the current state and chooses an action or sequence of actions that maximizes rewards among the selectable actions. Because of these learning methods, computers can make actions that maximize the sum of rewards for tasks without human intervention and without having to write explicit programs to achieve their goals. Because of these other advantages, reinforcement learning is expanding in various fields such as robots, games, control, statistics, genetics, and new drug development. This much-attention study of reinforcement learning has difficulties in implementing it in reality. However, most studies of reinforcement learning often stop at simulation rather than realistic implementation. For this reason, most of the research in reinforcement learning is often a step of checking the evaluation of the learned algorithms and models only on the computer. This work proceeds reinforcement learning with environmental creation, setting of reward functions, and learning of models using simulations. After that, we go a step further and show real-world implementation. This study was conducted in two ways. First, the theoretical reinforcement learning algorithm used Proximal Policy Optimization (PPO). Because PPO is an algorithm that is simple to learn, easy to implement, and has a better sample complexity, PPO was chosen to implement the code. First, part 1 used AWS DeepRacer. Learning was conducted using tracks and simulation tools provided by AWS. In particular, the reward function is important in reinforcement learning, and by designing the reward function in our own way, we can confirm that the learning time is drastically reduced. And the direction of learning was presented through the adjustment of hyperparameters. Furthermore, after simulation, we proceed to experiments where hardware runs on real tracks, showing

that it can also be implemented in reality in reinforcement learning. In actual driving, the track on the fourth floor of Newton Hall at Handong University was used. Experimental verification used tracks with different configurations than those that were simulated. By using a different track from the simulation, we prove that the model is a model that makes appropriate behavior even in an environment that we have never encountered before. Although reinforcement learning was conducted in simulation and implementation was shown through DeepRacer, it had a disadvantage that it was difficult to customize. To compensate for these shortcomings and enable them to be used in more fields, this work used ML-agent in Unity environments in part2. By using the Unity environment, users can configure the environment they want and set their own returns. Using these unity features, a simple track was created. Simulation was carried out through image preprocessing, reward function setting, and hyperparameter setting on the constructed track. Earlier in the simulation, the car's heading faltered or the agent was unable to determine the proper steering, and as the learning progressed, it was confirmed that the sum of the rewards increased linearly. As a result, a satisfactory simulation result was confirmed when a simulation of around 400,000 steps was conducted. The future plan is designed to drive the RC car on a real-world track through communication between Unity and RC car. The composition of these experiments resulted in the creation of the environment and the establishment of reward functions, simulation, and real-world implementation through the PPO algorithm. In particular, this paper is written in tutorial format so that readers can easily follow through the techniques of detailed experimental processes. Through this study, guidelines are presented that can be easily followed by students or researchers who are new to reinforcement learning or want to implement it in the real-world rather than in simulation.

Contents

I. Introduction

1.1. Research Outline and Purpose

Machine learning is one of the most sought-after areas of research in many fields. It is a word that anyone living in 2021 should know, as most industries and fields conduct research with the keyword machine learning and apply on the field.

Figure 1. Types of Machine Learning

Machine learning can be classified as supervised learning, unsupervised learning, and reinforcement learning, such as **Figure1**, according to the learning method. Supervised learning is a method of learning a model by putting user-labeled input data into every situation, and unsupervised learning is a method of giving input data to the model, classifying and learning without labeling. However, there are fatal drawbacks of supervised and unsupervised learning. This means that data needs to be collected and processed. The reason why this is a disadvantage is that it takes a lot of time and money to collect data. Furthermore, the process of processing data for the model to learn is also time-consuming and expensive. Reinforcement learning is a machine learning method that can solve these problems and train models without input of labeled data. In particular, reinforcement learning involves the collection of data directly by the agent according to the environmental return, and the model is trained towards receiving maximum rewards through an algorithmized reward function. If circumstantial returns are established only in a certain environment, appropriate reward functions, and well-established algorithms, reinforcement learning will be economical and efficient beyond comparison with supervised learning and unsupervised learning. Furthermore, supervised learning is vulnerable when new data comes in. However, reinforcement learning can be said to become more and more robust over time, as reinforcement learning proceeds even in the new data environment, depending on the reward function. Due to these advantages of reinforcement learning, a lot of research is being

conducted and a lot of attention is being paid to it. However, there is a disadvantage that it is difficult to implement in the real-world, a disadvantage of reinforcement learning. Solving these problems is currently a problem that reinforcement learning must address. Most studies tend to quit in simulations without addressing these problems. This work demonstrates that reality implementation of reinforcement learning is possible by showing implementations using hardware in the real world, not just simulating the process of experiments.

1.2. Research Background

Many studies of reinforcement learning are being conducted throughout the world. In particular, Google has shown considerable interest in reinforcement learning by acquiring DeepMind Technology, a British venture, for $4 billion. What's more surprising about this investment is that DeepMind is a small company with only 50 employees. DeepMind's announcement of reinforcement learning after the acquisition shook the world. In particular, in 2016, AlphaGo and Lee Sedol, who are known by everyone around the world, are the result of reinforcement learning produced by Deep Mind.

Figure 2. AlphaGo of Google DeepMind

Many studies are being conducted on this reinforcement learning to enter all of these reinforcement learning. At the AI with Google 2018 conference, Jeff Dean, Google's AI general manager, said: "Google's goal is to provide AI for everyone. "By utilizing 'artificial intelligence' and 'machine learning', we are going to overcome the huge challenges that not only Google but also humanity faces." As such, experts in many fields emphasize the importance of machine learning. Deep Q-learning algorithms, algorithms developed before AlphaGo in DeepMind, demonstrate that simple and simple tasks or games have already outperformed humans, while proving that artificial intelligence with reinforcement learning has outperformed humans.

1.3. Research Summary

In this study, Experiments are conducted in a way that simulates reinforcement learning using Proximal Policy Optimization (PPO) and shows it in reality. For this experiment to proceed, it is first necessary to understand the theoretical PPO algorithm. After understanding the theoretical PPO algorithm, implementation and reward functions of the algorithm are established by utilizing AWS DeepRacer. The learned model is applied to the DeepRacer hardware and experiments are conducted as a driving process on the real track. In part 2, environmental creation using unity ml-agent is carried out. Unity, which can customize the virtual environment, is returned by the agent through a track set by the user. Part2 is designed to provide more robust virtual environments and rewards through the design of reward functions after the preprocessing of images in environmental returns. As simulations progress after these tasks, the model's learning progresses, and the model's learning results are observed and analyzed.

II. Method of Research

2.1 Proximal Policy Optimization Algorithms (PPO)

The Proximal Policy Optimization (PPO) algorithm is a policy gradient method for reinforcement learning that optimally update based on the two networks of Actor and Critic. It is similar to well-known algorithms such as Actor-Critic, A2C, and A3C. However, PPO re-uses a batch of experience data even after newly updating the policy. PPO leads to a less variance in training and helps the agent deviating to meaningless actions.

PPO aims to find the optimal parameter θ that maximizes the expected rewards by updating θ to minimize the cost function by using Trust Region method. It uses the ratio between the new policy p_θ and old policy $p_{\theta_{old}}$.

1. Importance Sampling

One of the most characteristically used methods in PPO is Importance Sampling. The formula for importance sampling is as follows.

$$\int_x x\,P(x)dx \approx \frac{1}{N}\sum_{i=1}^{N} x_i = \int_x x\frac{p(x)}{q(x)}q(x)dx \approx \frac{1}{N}\sum_{i=1}^{N} x_i \frac{p(x_i)}{q(x_i)}$$

Importance sampling is a method of producing results that are almost identical to those originally calculated where probabilities are calculated elsewhere.

Reinforcement Learning usually aims to find the parameter θ that makes up the policy that maximizes the Expected Rewards and updates θ in the gradient direction. The policy-gradient expression in A3C is as follows.

$$\nabla_\theta J_\theta = \sum_{t=0}^{\infty}\int_\tau \nabla_\theta \ln p_\theta(a_t|s_t)A_t p_\theta(s_t,a_t)p(s_{t+1}|s_t,a_t)\,ds_t a_t s_{t+1}$$

Since integration is difficult in the above expression, the sample is extracted from $p_\theta(s_t,a_t)$and used. The problem here is that you have to discard the data from which you extracted the sample and select a new sample. The PPO uses the importance sampling described above to produce an effect extracted from p_θ without having to extract it from p_θ. It is $p_{\theta_{old}}$ that emerges from this idea.

If the formula has a fixed $p_{\theta old}$ at one point in time, only the fixed $p_{\theta old}$ is sampled when θ is updated after the step. Multiply $\frac{p_{\theta old}(s_t,a_t)}{p_{\theta old}(s_t,a_t)}$ from the A3C formula above. If the expression is represented again using Importance sampling, it is shown in the following expression.

$$\frac{p_\theta(s_t,a_t)}{p_{\theta old}(s_t,a_t)} = \frac{p_\theta(s_t)p_\theta(s_t,a_t)}{p_{\theta old}(s_t)p_{\theta old}(s_t,a_t)}$$

In this formula, if $\theta \approx \theta_{old}$, It can be expressed in the following formula.

The formula above expresses the formula in which θ is updated.

$$\theta \leftarrow \theta + \nabla_\theta + \alpha \nabla_\theta \sum_{i=t-N+1}^{t} \frac{p_\theta(s_t,a_t)}{p_{\theta old}(s_t,a_t)} A_i$$

The conditions for PPO for updating theta like this

$$p_{\theta_{old}}(s_t) \approx p_\theta(s_t) \rightarrow \theta_{old} \approx \theta$$

While satisfying this condition, the PPO algorithm proceeds. Assuming that this condition is satisfactory, the gradient formula is rearranged as follows.

$$\nabla_\theta J_\theta = \sum_{t=0}^{\infty} \int_\tau \frac{\nabla_\theta p_\theta(a_t|s_t)}{p_{\theta_{old}}(a_t|s_t)} A_t p_{\theta_{old}}(s_t,a_t) p(s_{t+1}|s_t,a_t)\, ds_t a_t s_{t+1}$$

Eventually, the key ideas in the PPO algorithm can be summarized using Importance Sampling by the following two formulas.

$$max \sum_{i=t-N+1}^{t} \frac{p_\theta(s_t,a_t)}{p_{\theta old}(s_t,a_t)} A_i$$

$$constraint \quad : E_{st}\left[KL\left[p_{\theta_{old}}, p_\theta\right]\right] \leq \delta$$

2. Clipping

The most important technique in PPO is clipping. TRPO solved the constrain optimization problem using Second-order KL-Divergence. This method gives computers a lot of computation. PPO forces the confidence interval to be clipped when updating function. For simplification of formulas $\frac{p_\theta(s_t,a_t)}{p_{\theta old}(s_t,a_t)} = r_i$ If Clip is used and the formula that needs to be maximized is rewritten as described in 1.

$$max \sum_{i=t-N+1}^{t} J_i^{\phi}$$

$$J_i^{\phi} \triangleq \min[r_i A_i, \phi \quad (r_i, 1-\varepsilon, 1+\varepsilon)A_i]$$

This representation of the formula allows for consideration by maximizing only J_i^{ϕ} unconditionally.

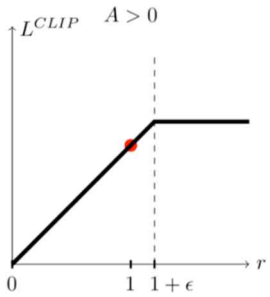

Figure 3. Clipping of Clipping

The expression clip ϕ $(r_i, 1-\varepsilon, 1+\varepsilon)A_i$ can be represented as **Figure 3**. A_i is positive means that the assessment of the action is in the right direction. The larger the r_i, the more maximized it is, the more clip it is used to prevent this. The reason for this clip is to ensure that it does not deviate from the constraints of $p_\theta \approx_{\theta old}$.

3. GAE (Generalized Advantage Estimation)

It can be said that GAE is the synthesis of n-step errors.

$$A_t^1 = R_t + \gamma V(s_{t+1}) - V(s_t) \triangleq \delta_t$$

$$A_t^2 = R_t + \gamma R_{t+1} + \gamma^2 V(s_{t+2}) - V(s_t) \triangleq \delta_t + \gamma\delta_{t+1}$$

$$\vdots$$

Like the above format, the n-step TD-Error can be represented mathematically. When organized using Exponential Moving Average, the following formula is used:

$$A_t^n = \sum_{n=1}^{t+n-1} \gamma^{k-t}\ \delta_k$$

Put the formula above into the $TD(\lambda)$ formula below.

$$TD(\lambda) = \sum_{k=t}^{\infty} (1-\lambda)\lambda^{n-1} A_t^n$$

The expression is summarized to represent $A_t^{GAE(\gamma,\lambda)}$.

$$A_t^{GAE(\gamma,\lambda)} = \sum_{k=t}^{\infty} (\gamma\lambda)^{k-t}\ \delta_k$$

In the PPO paper, GAE is represented as follows:

$$A_t^{GAE(\gamma,\lambda)} = \sum_{\boldsymbol{k=l}}^{\boldsymbol{t}} (\boldsymbol{\gamma\lambda})^{\boldsymbol{k-i}}\ \boldsymbol{\delta_k}$$

2.2 Part 1 – DeepRacer Simulation and Implementation

2.2.1 Simulation

AWS DeepRacer is a recent system that has a high integration between the simulation environment and the real world for the autonomous driving of a mini car. It provides an online service to train and evaluates the reinforcement learning models in a simple simulated environment. The trained model can be easily deployed on the DeepRacer car of 1/18th scale RC car.

The simulation environment of AWS console provides the car data of the 2D position (x_{car} , y_{car}),heading angle (H), and steering angle (θ_s), The reward function from these parameter values is designed as shown in below **Figure 4.**

Reward Function

Initializing & Updating Paramters

H = *car heading angle*
θ_s = *steering angle of car*
x_t, x_y = *target point*
x_{car}, y_{car} = *car point*
eps = *error disired by the user*
$dx = x_t - x_{car}$, $dy = y_t - y_{car}$
$\theta_t = polar(dx, dy)$
$\theta_{best\ angle} = \theta_t - H$

reward
if $(\theta_{best\ angle} - \theta_s < eps)$
get *reward*

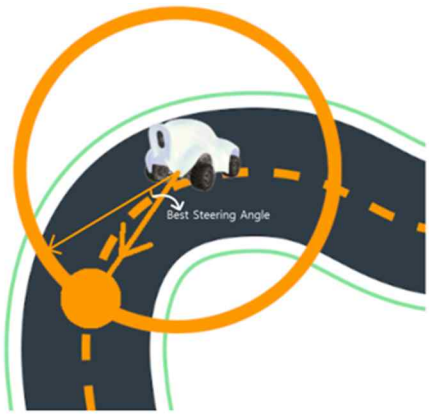

Figure 4. Overview of the Reward Function

The key idea is giving a positive reward if the steering angle of the wheels (θ_s) is similar to the net heading angle (θ_{net}) towards the target point from the current position of the car. The target points are set as the points on the center line of the track at a certain distance away. If the steering angle (θ_s) is exactly the same as the net heading angle (θ_{net}), then the car would drive directly to the target point.

Hyperparameter	Value
Gradient descent batch size	64
Entropy	0.01
Discount factor	0.95
Loss type	Huber
Learning rate	0.0003
Number of experience episodes between each policy-updating iteration	20
Number of epochs	10

Figure 5. Hyperparameter

The hyperparameters for training are set up as shown in **Figure 5**. The batch size was 64, not a large size, and the epoch count was 10 to speed up the learning. In addition, the discount factor was set to 0.95, giving a large weight to the present state. It used Huber loss that is a compromised loss function between MSE and MAE.

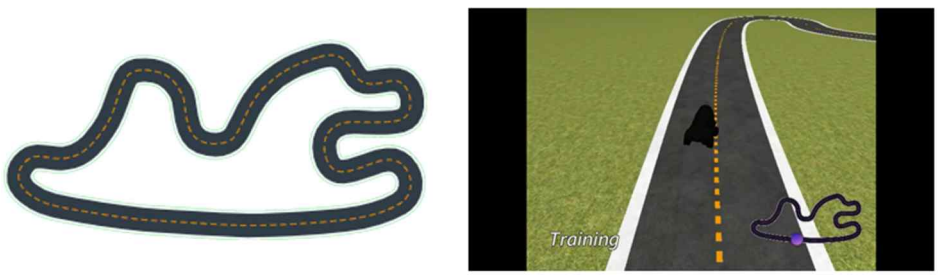

Figure 6. Simulation Track and Driving

The left side of the **Figure 6.** shows the overall track, and the right side shows the agent in training. Learning time varies depending on how the reward function is designed.

2.2.2 Real-World Implementation

The trained model was deployed on the DeepRacer car with the specification as shown in **Figure 7**.

Car	18th scale of real car
CPU	Intel atom Processor
Memory	4GB RAM
Storage	32GB Memory
Wi-Fi	802.11ac
Camera	4 MP Camera with MJPEG
Softwatre	Ubuntu OS 20.04.3

Figure 7. DeepRacer and Specs

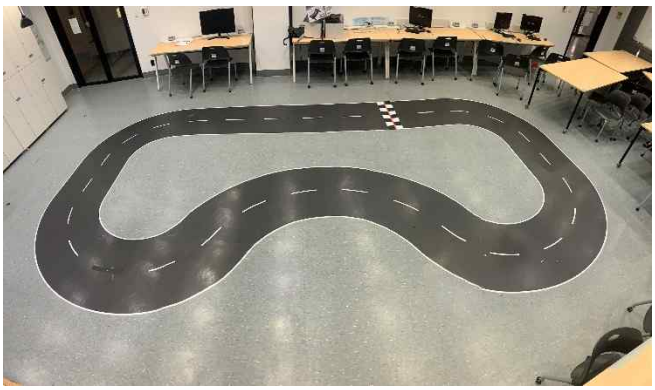

Figure 8. Real Track

Figure 9. DeepRacer Driving within Straight and Curve Lanes

The DeepRacer car was tested on a small-scale RC car track, as shown in **Figure 9**. Although it was a different environment from the simulation track, the car managed to drive well without deviating off from the lanes. With the four-hour trained model, the car could be driven both the straight and curved lanes without any lane departure.

2.3 Part 2 – Unity Simulation

Unity is a game engine that provides 3D and 2D game development environments, and is a program for the production of 3D animation, virtual reality (VR) and augmented reality (AR). Numerous games and simulations that are currently commercially available have been developed through this Unity and are very useful tools for their high accessibility and freedom. It offers C# and Java scripts(Unity script), and in most cases, it is used in conjunction with visual studio.

In this research, it used Unity ML-Agents toolkit, one of the additional packages of Unity. ML-Agents are an open-source tool developed on a Pytorch basis that allows intelligent agents to be a trained in many ways in a customized environment in a game or simulation. Some examples included in the package allow us to understand the overall learning flow.

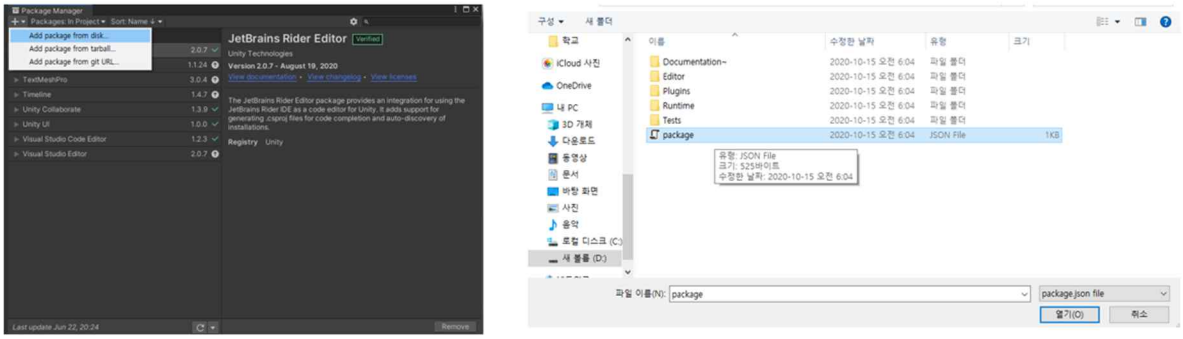

Figure 10. Add ML-Agents Package

It is important to set Unity and its corresponding Python version and ML-Agents release version. It can be downloaded from the GitHub and ML-Agent-Release 8 in this research. The package inside the downloaded release\com.unity.ML-Agents can be installed through the package manager of the unity. After that, create a project in unity and copy paste the entire release X\project\Assets\ML-Agents folder inside your Assets folder.

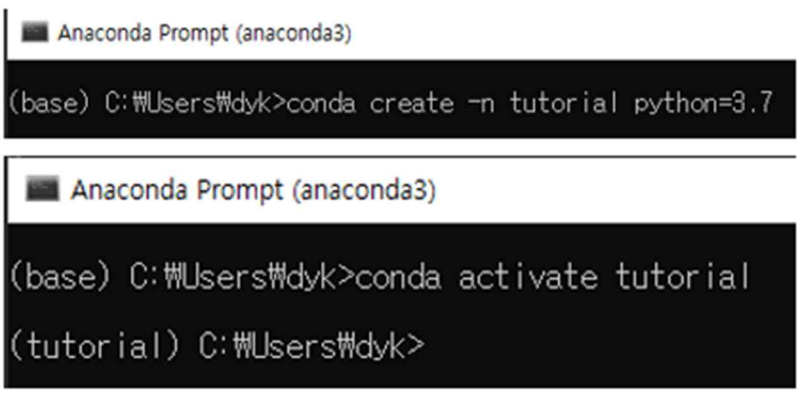

Figure 11. Create the Virtual Environment

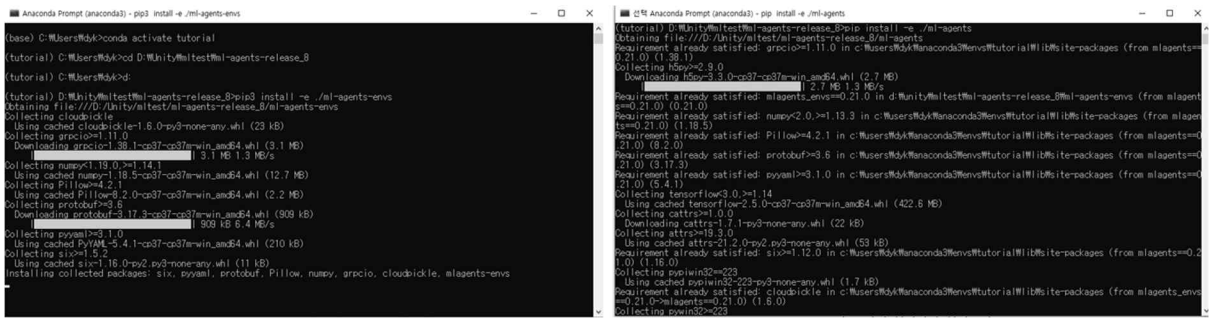

Figure 12. Set-up Necessary PIP

Use Python-api for learning with ML-Agents and install Python 3.7 in a new virtual environment for version compatibility. In the release folder, type ‘‘pip3 install -e ./ml-agents-envs’, ’pip install -e ./ML-Agents’ to install various pips, such as tensorflow, numpy, etc.

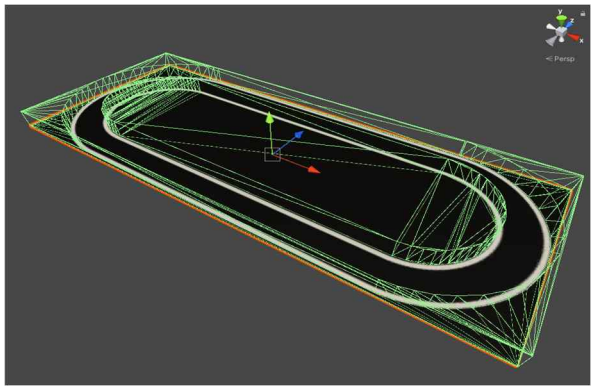

Figure 13. Configuration of Learning Environment

To create an environment in earnest, new scenes were created in the project, creating plane objects to be used as track in the Hierarchy. Drag and drop the track image on the generated plane and resize it. Since the important thing in driving learning is that the car does not deviate from the line, use the Probuilder tool to build invisible walls along the line.

Figure 14. Car Agent

The track and environment for the car to drive has been completed, an agent must be set. Import a low poly car asset from unity asset store. When downloading the created car's assets, it is recommended rather than creating your own because the default settings such as car collider, wheel collider, and design are set. The script to enter the agent uses C# language. The script initializes all variables in initializing(), reset each episode from OnEpisodeBegin(). OnActionReceived() determines the agents' action. The car's motion is determined by the position of the wheel, so it functions in Updatewheelpos(). Finally, there is a heuristic () that moves the agent through keyboard manipulation so that the user can decide what to do.

The reward for Agent's positive action on learning is continuously increase by 0.01 per step until it departs the lane. End condition of episode is the agent leaves the lane. Action space consists of 3 discrete actions, with the max steering angle initially set to 30, it is as follows:

1. Move forward: steering angle x 0
2. Move left: steering angle x (-1)
3. Move right: steering angle x (+1)

By not adding stopping motion to the motion, it only compensates for the passing of the step and prevents the car from stopping.

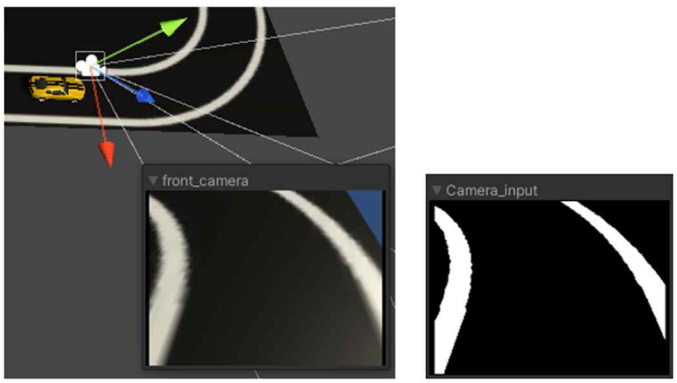

Figure 15. Observation – Image in Front of the Car

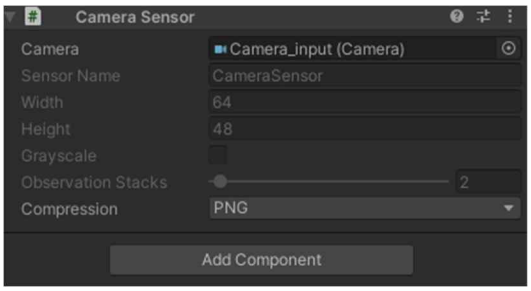

Figure 16. Observation – Camera Sensor

The environment and agent are prepared for reinforcement learning, and the agent must set up an observation to receive information from the environment. In this simulation, the camera is attached to the front of the car to set the image of the line to observation. Set up an invisible cube that maintains distance from the car and enter the road conditions (line) at the front of the car, focusing on the cube.

It is adjusted that the resolution of the front camera from 640×480 to 64×48 to export the weight file of the simulation results to the embedded GPU for real-world implementation and to reduce the amount of information, the three-channel BGR image was replaced with grayscale and thresholding black and white. Because this image is used as an observation by the agent, the camera object is set as a child object of car and the camera sensor script is added to the inspector. Observation stack is a value indicating how many observations to perform or how many tasks should be selected, set to 2 to select tasks using the current image and previous step image.

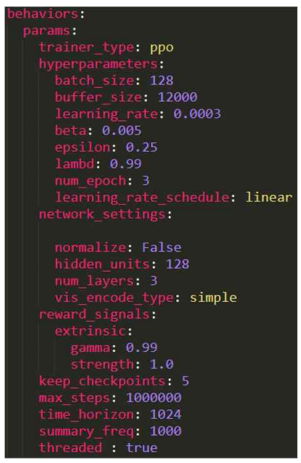

Figure 17. Hyperparameter

It is necessary to set various parameters for learning. First, the learning rate corresponds to the strength of gradient descent updating each step. Decaying this value until max step, then learning converges more stable. Next, the batch size, which is the number of experiences in each iteration of gradient descent, was appropriately adjusted to meet the specifications of the computer. Parameter beta means intensity of entropy normalization, making the policy random. this allows the agent to navigate the action space during training. This value should be adjusted as the entropy consistently decreases alongside increases in reward.

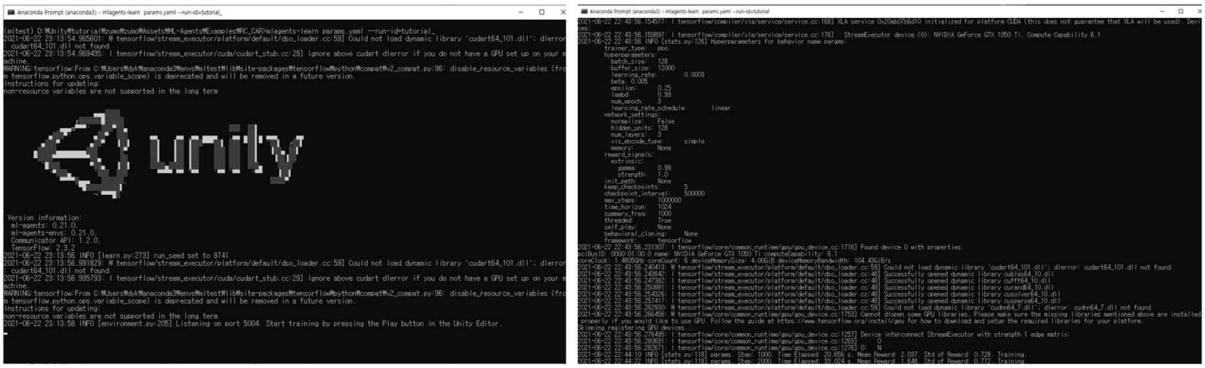

Figure 18. On-Run Screen

When you are ready for reinforcement learning, move to the current folder and enter the command in a virtual environment created using anaconda prompt. As shown in the **figure** above, information from the tools required for learning, such as the unity logo and Tensorflow is printed. When simulation is started at unity, each parameter set and the mean reward and std reward (standard deviation of the reward) for each step are printed to the console.

Figure 19. Multi-training

The advantage of simulation in unity is that it can speed up learning by copying the agent and environment and performing multiple trainings at the same time. Training by multiple cars may be faster than training by one car, but it is necessary to select an appropriate number of training as memory may be exceeded depending on your own RAM specification.

III. Result and Discussion

3.1 Evaluation of the trained AWS-DeepRacer model

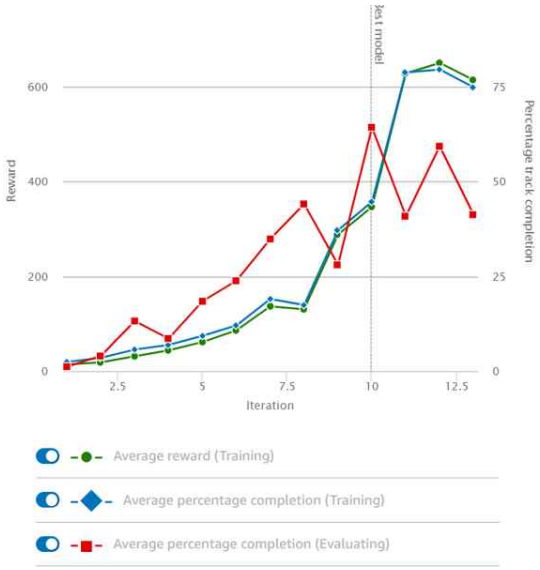

Figure 20. Result Graph of AWS DeepRacer Simulation

The training in the AWS console took four hours and the mean reward is shown to increases with the iterations, as shown in **Figure 20.** Furthermore, the average percentage completion of training has also seemed to be increasing continuously. The DeepRacer car was tested on a small-scale RC car track, as shown in **Figure 20.** Although it was a different environment from the simulation track, the car managed to drive well without deviating off from the lanes. With the four hours trained model, the car could be driven both the straight and curved lanes without any lane departure.

3.2 Evaluation of the trained Unity 3D ML-Agents model

The Tensorboard provides summary statistics related to the training of the Agent. Among the graphs provided by the Tensorboard, graphs such as Cumulative Reward, Entropy, Episode Length, Learning Rate, Policy Loss, Value Loss, and Value Estate can confirm that the Agent's training was successful.

Table 1 Explanation for each result value of PPO

PPO result value	Explanation
Cumulative Reward	The mean cumulative episode reward over all agents.
Entropy	How random the decisions of the model are.
Episode Length	The mean length of each episode in the environment for all agents.
Learning Rate	How large a step the training algorithm takes as it searches for the optimal policy.
Policy Loss	The mean magnitude of policy loss function.
Value Loss	The mean loss of the value function update.
Value estimate	The mean value estimate for all states visited by the agent.

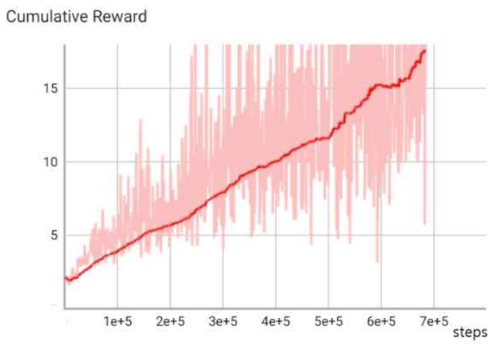

Figure 21. Cumulative reward per step graph & Episode length per step graph

Analyzing each graph representing the results of the Agent training shows that the cumulative rewards have steadily increased since training began in **Figure 21.** As mentioned above, Rewards accumulate more as the agent continues to drive for a longer period without leaving the line, thus driving more and more reliably as the training progresses. As the training progresses, the Agent will drive longer during one Episode, so the graph shows that Episode Length also increases in a similar shape to the cumulative reward.

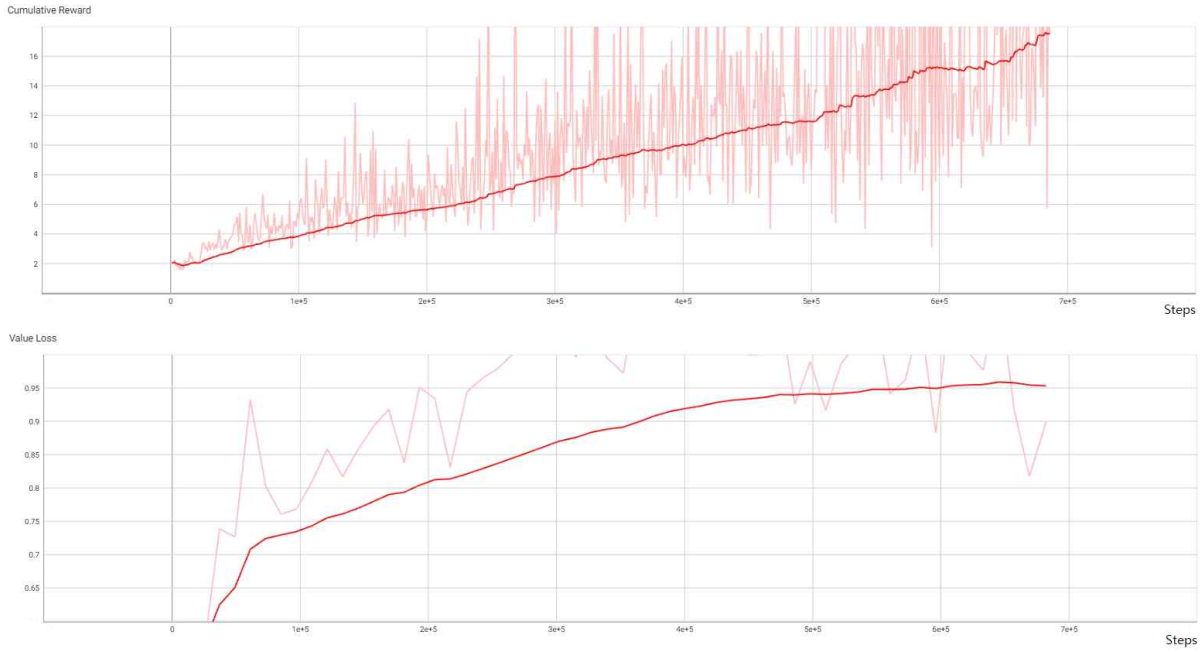

Figure 22. Cumulative Reward per step graph & Value Loss per step graph

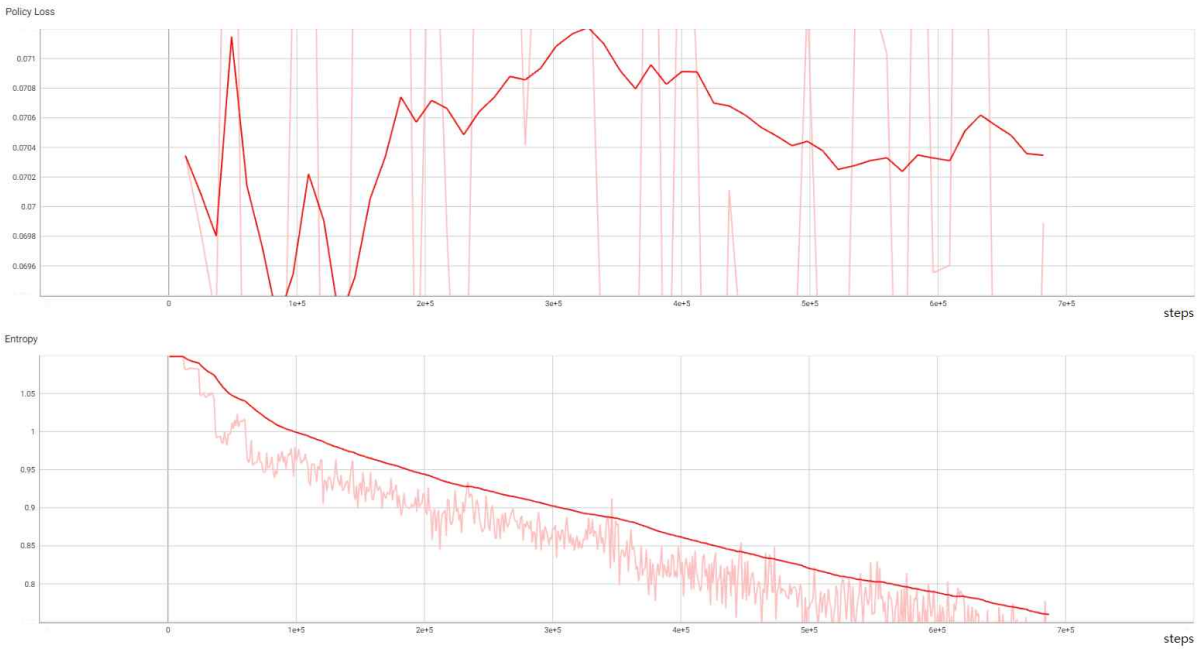

Figure 23. Policy Loss per step graph & Value Loss per step graph

The aspects of both graphs in **Figure 22.** correlate with the amount of change in the policy as learning progresses. Policy loss is calculated to reflect the stability of a policy in which the amount of change in the policy gradually decreases as learning about good policies progresses. Initially, the Agent

visits various state-action spaces to find a good policy, and the rewards it receives are not stable, resulting in high variance, which increases value loss in proportion to the increase in Cumulative Rewards. Policy loss has a stable cumulative reward as the agent approaches a good policy as learning progresses. As the value loss decreases, so does the policy loss change, and the amplitude gradually decreases as the learning progresses compared to the initial appearance of the policy loss graph with a large amplitude. Entropy's graph representing randomness of the model in **Figure 23**. is determined by the beta value, which determines the degree of entropy normalization as the initial set learning rate decreases, allowing the agent to explore the action space appropriately early in learning and converges to good policies. The Agent stopped training early when the value of the Cumulative Reward, which is constantly circulating the track, exceeded 10. During approximately 400,000 steps of training, policy loss and value loss were seen in **Figure 23.** as the learning progressed, policy loss fluctuated and value loss gradually increased, and the agent found better policies and improved learning.

V. Conclusion

In this paper, an autonomous driving reinforcement learning model based on PPO algorithms is trained and implemented directly. As an autonomous driving learning model, we used DeepRacer car and Unity 3D ML-Agent toolkit provided by Amazon Web Services (AWS) to train automotive agents. Both models successfully implemented autonomous driving models by establishing an appropriate return function and hyperparameter in each given simulation environment.

For DeepRacer provided by AWS, there are about 18 different environment return variables, each environment return variable can be designed by the user and learned within the track provided by AWS, but the user cannot customize the environment. Therefore, learning to directly implement autonomous car agents and environments was conducted using Unity 3D ML-Agent toolkit, which can additionally design the environment directly to set environment variables and enable customization of users. The console provided allows agents to design reward functions for various environmental return variables and to learn more quickly and accurately for a given environment, and to train faster despite shorter learning times. Moreover, models learned in virtual environments have succeeded in implementing autonomous driving with similar performance when driven in real-world environments.

On the other hand, models implemented using Unity 3D ML-Agent toolkit had the difficulty of creating everything new from the configuration of the environment, but they had the advantage of creating and learning the environment like the actual environment. Based on the PPO reinforcement learning algorithm, the car agent received only input images that came through the camera attached to the front of the car, and by setting the appropriate reward function, the agent remembered the input image and determined the correct action for the state and succeeded in implementing autonomous driving within the virtual environment. Both models were successful in implementing in virtual environments, but for models implemented in Unity 3D ML-agents toolkit, there was no problem driving off the track without leaving the track, reducing unnecessary actions, and not implementing stable driving. For AWS, more information about the environment can be obtained through approximately 18 environmental return variables and the reward function can be set more specifically, enabling more stable driving.

Future research plans are to implement models trained in virtual environments that are like real-world ones with similar performance in real-world environments. In this research, there was a problem with the model that was trained, which failed to drive reliably even within the line. To solve this problem, we plan to add various sensors in addition to the camera and increase the environment return variable

so that the agent can learn to recognize the given environment better so that it can drive more stably. As a result, we expect to implement a reliably drivable model even in real-world environments if input of environment returns variables like input.

References

[1] Bharathan Balaji, Sunil Mallya, Sahika Genc, SaurabhGupta, Leo Dirac, Vineet Khare, Gourav Roy, Tao Sun, Yun-zhe Tao, Brian Townsend, Eddie Calleja, Sunil Muralidhara,and Dhanasekar Karuppasamy. DeepRacer: Educational au-tonomous racing platform for experimentation with sim2realreinforcement learning.CoRR, abs/1911.01562, 2019.

[2] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jor-dan, and Philipp Moritz. Trust region policy optimization. InInternational conference on machine learning, pages 1889–1897. PMLR, 2015.

[3] John Schulman, Philipp Moritz, Sergey Levine, Michael Jor-dan, and Pieter Abbeel. High-dimensional continuous con-trol using generalized advantage estimation.arXiv preprintarXiv:1506.02438, 2015.

[4] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Rad-ford, and Oleg Klimov. Proximal policy optimization algo-rithms.arXiv preprint arXiv:1707.06347, 2017.

[5] John John Schulman, Filip Wolski, Prafulla Dhariwal, AlecRadford, and Oleg Klimov. Proximal policy optimizationalgorithms.arXiv preprint arXiv:1707.06347, 2017.

[6] Arthur Juliani, Vincent-Pierre Berges, Ervin Teng, AndrewCohen, Jonathan Harper, Chris Elion, Chris Goy, YuanGao, Hunter Henry, Marwan Mattar, et al.Unity: Ageneral platform for intelligent agents.arXiv preprintarXiv:1809.02627, 2018.

[7] Micheal Lanham.Learn Unity ML-Agents–Fundamentalsof Unity Machine Learning: Incorporate new powerful MLalgorithms such as Deep Reinforcement Learning for games.Packt Publishing Ltd, 2018.

Appendix

2021-1 Spring Semester Post-Capstone

A Study of Reinforcement Learning for Self-driving RC Car using AWS DeepRacer and Unity ML-agent

Supervisor: Young-Keun Kim
Doyeon Kim Jooho Kim Yechan Song

Team: Timesquare

Introduction

- Study of Reinforcement Learning(PPO) algorithm for autonomous driving
- Design and train PPO model in simulation environment
- Implementation of PPO in real-environment on small-scaled RC car

aws

- Simulation environment provided
- Hard to customize environment
- Applies on DeepRacer car only

unity

- Can create customized environment
- Can be applied to various RC car platform
- Hard to implement on embedded processor

Theory

1. PPO
   - The most popular reinforcement learning algorithms
   - Easy implementation and high performance
   - High data efficiency

2. Find optimal parameter θ

$$\theta \leftarrow \theta + \nabla_\theta \sum_{i=t-N+1}^{t} \frac{p_\theta(s_i,a_i)}{p_{\theta_{old}}(s_i,a_i)} A_t$$

$$constraint: r(\theta) = \frac{p_\theta}{p_{\theta_{old}}} < \varepsilon$$

3. GAE

$$J_t = \sum_{i=t-N+1}^{t} \frac{p_\theta(s_i,a_i)}{p_{\theta_{old}}(s_i,a_i)} A_t$$

$$A_t \triangleq Q(a_t|s_t) - V(s_t) = \sum_{i=0}^{\infty} (\gamma\lambda)^i \delta_i$$

$$\delta_t = R_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$

4. Clipping

$$J_t^{clip} \triangleq \min(ratio_t \cdot A_t, clip(r_t, 1-\varepsilon, 1+\varepsilon) A_t)$$

5. PPO Update Algorithm

0. *Initialize* θ, w
Repeat 1~4
1. *Collect N Sample* $(sample: \{s_i, a_i, s_{i+1}\})$
Repeat 2~3*(Epoch)*
2. *Actor update:* $\theta \leftarrow \theta + \alpha\nabla_\theta \sum_{i=t-N+1}^{t} J_i^{clip}$
3. *Critic update:* $w \leftarrow w - \beta\nabla_\theta \sum_{i=t-N+1}^{t} \left(A_i^{GAE}\right)^2$
4. *Clear the batch*

Part1 – AWS DeepRacer

Reward Function

```
Reward Function
Initializing & Updating Parameters
    H = car heading angle
    θ_s = steering angle of car
    x_g, y_g = target point
    x_car, y_car = car point
    eps = error disired by the user
    dx = x_g − x_car,  dy = y_t − y_car
    θ_t = polar(dx, dy)
    θ_best angle = θ_t − H

reward
    if(θ_best angle − θ_s < eps)
        get reward
```

Fig 1. Reward Function

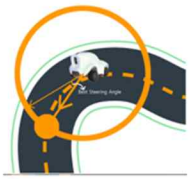

Fig 2. Schematic of Reward Function

Simulation

Fig 3. Simulation Program of AWS

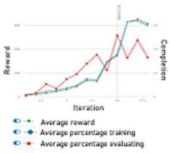

Fig 4. Result Graph of Simulation

Implementation

CPU : Intel atom Processor
Camera : 4MP(2688x1520)

Fig 5. Track and AWS DeepRacer

Fig 6. Implementation of Driving

Part2 – Unity-ML agent

Reward & Hyperparameter

Episode end condition:
- When the agent leaves the lane
- When average reward per 10,000 steps is over 20

Reward:
- Increase by 0.01 per step if driving within lanes

Main Hyperparameters:
- beta: 0.005
- Number of hidden layers: 128
- Learning rate: 0.0003

Image Processing

Fig 7. Using Thresholding to Make Binary Image

Simulation

Fig 8. Unity Track and Car

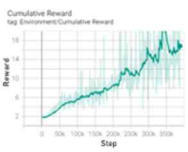

Fig 9. Result of Simulation

Future Plan

Fig 10. Track and RC car

- Implement on embedded processor
- Test driving on RC car track

Conclusion

- Studied reinforcement learning algorithm (PPO) for a simple autonomous driving
- Used AWS DeepRacer and Unity for training agent in Simulation Environment
- Deployed RL model on DeepRacer RC Car for successful driving
- Need to implement on our RC Car for future plane

References

[1] Bharathan Balaji, Sunil Mallya, Sahika Genc, Saurabh Gupta, Leo Dirac, Vineet Khare, Gourav Roy, Tao Sun, Yun zhe Tao, Brian Townsend, Eddie Calleja, Sunil Muralidhara,and Dhanasekar Karuppasamy. Deepracer: Educational au tonomous racing platform for experimentation with sim2realreinforcement learning. CoRR, abs/1911.01562, 2019.

[2] John Schulman, Philipp Moritz, Sergey Levine, Michael Jor dan, and Pieter Abbeel. High-dimensional continuous con trol using generalized advantage estimation. arXiv preprintarXiv:1506.02438, 2015.

[3] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Rad ford, and Oleg Klimov. Proximal policy optimization algo rithms. arXiv preprint arXiv:1707.06347, 2017.