Autonomous Parking Simulation Using Reinforcement Learning

1. Abstract

Autonomous driving has been researched a lot in recent years and most of improvements have been achieved. However, autonomous parking has been a practical problem so far. So, this paper deals about a simulation program that visualizes the autonomous parking algorithm by using reinforcement learning (RL). ML-Agent, a plugin of Unity is used for the RL.

2. Introduction

Autonomous vehicle is a vehicle that drives by itself without human inputs. It detects several environments via sensors so that it could move safely. Autonomous driving is usually classified with 6 levels: from level 0 (no self-driving) to level 5 (hands, eyes, and mind-off at any environment). The latest studies are in the 1(hands-on) or 2(hands-off) level now. The autonomous vehicle may lead to some better experiences, such as decreasing traffic accidents, road rages and vehicle crimes.

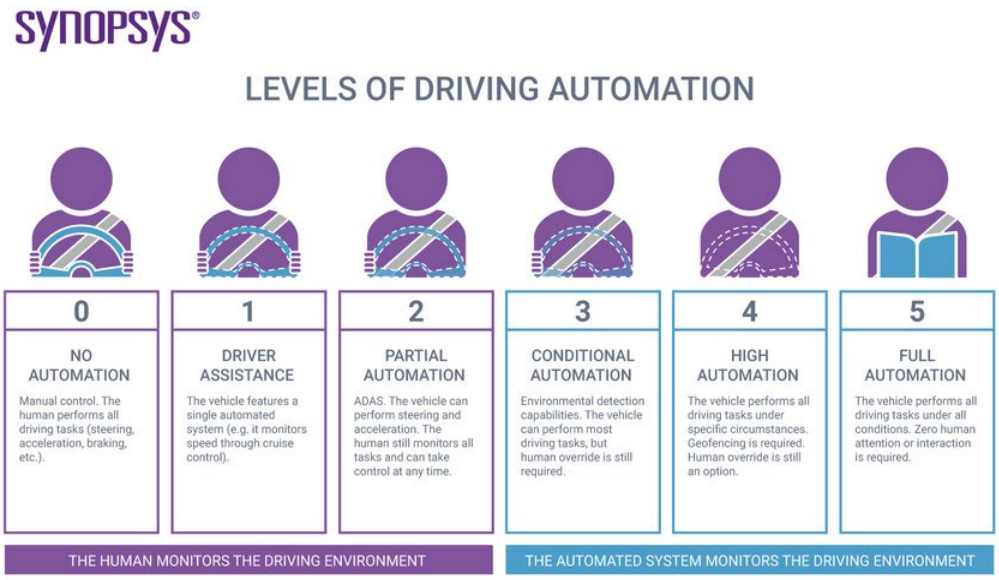

Figure 1 Autonomous driving levels

Since 1960s, Mercedes-Benz, the car company has founded and led the studies about this concept. Early autonomous vehicle model just only could follow lanes of a test track which does not have obstacles. After 1990s, autonomous driving including obstacles has been developed in earnest as computation technology grows. The applications that practically completed are the following: (adaptive) cruise control, lane keep, lane change, highway self-driving, and intersection self-driving.

However, most studies in the field of autonomous vehicle have focused on autonomous driving. Autonomous parking has been not carried much as autonomous driving. Therefore,

autonomous parking will be able to be commercialized around 2027 which is slower than driving. Generally, actual car model is used to implement it. Major problems with this kind of experiment are the following: danger in crash, space constraints, and high research expenses. So, the main purpose of this study is to develop a software that simulates autonomous parking machine learning.

Reinforcement learning is used for the research. In reinforcement learning, an agent which is the target object takes action under its policy, and the environment gives a reward for the action. As the episodes go by, the policy of the agent is optimized by maximizing the reward, and the action is improved by that.

A game engine named Unity is selected to visualize the simulation. And ML-Agent (Machine Learning Agent) which is a plugin program is also used. To be specific, agent, environment, and reward are modeled in Unity Editor by editing game objects and C# scripts. The reinforcement learning is executed on a Linux virtual environment with python ML-Agent scripts. A behavior parameter is printed after the learning is completed, and the result can be visualized with it.

ML-Agent supports various kind of RL algorithms. PPO algorithm is used in this research.

There are 3 major aims of the software:

- Safety in simulation:

  Before the experiment using the actual car model, the software simulates all the process and makes algorithms without any safety problem.

- Generalized simulation:

  All possible situations are simulatable by modeling on Unity Editor.

- Porting behavior parameter on actual cars:

  Unity ML-Agent prints its result in onnx file (Open Neural Network Exchange). Onnx file is a standard of machine learning framework that is written in C++, C, Python, C#, and Java API. It is compatible with Azure environment. If an OS such as Window or Azure is ported in an imbedded system of a car, onnx file can be ported as well.

3. Research Process

(1) Components

A. Parking Lot

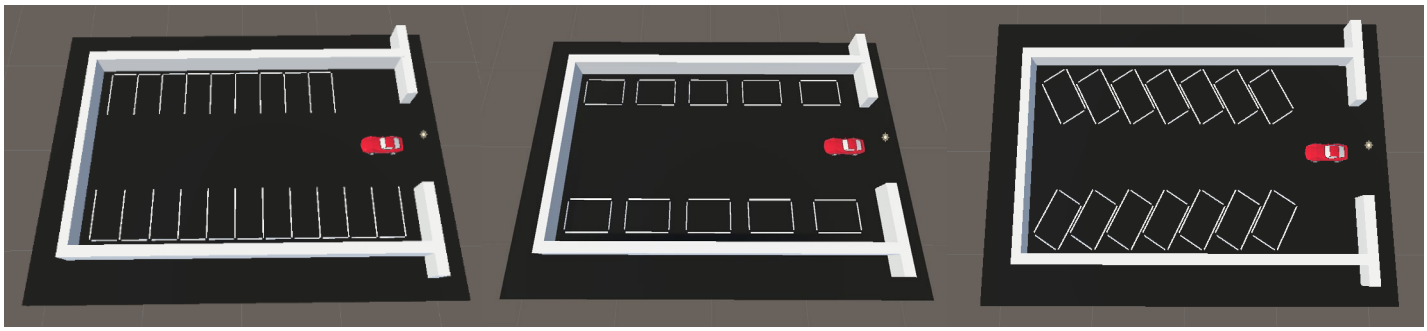

Figure 2 Front/back parking **Figure 3 Parallel parking** **Figure 4 Diagonal parking**

Parking lot game objects are modeled for each of 3 situations: front/back, parallel, and diagonal parking. Universal gravity is applied, and the floor object is set static to hold car objects.

B. Parking Slot

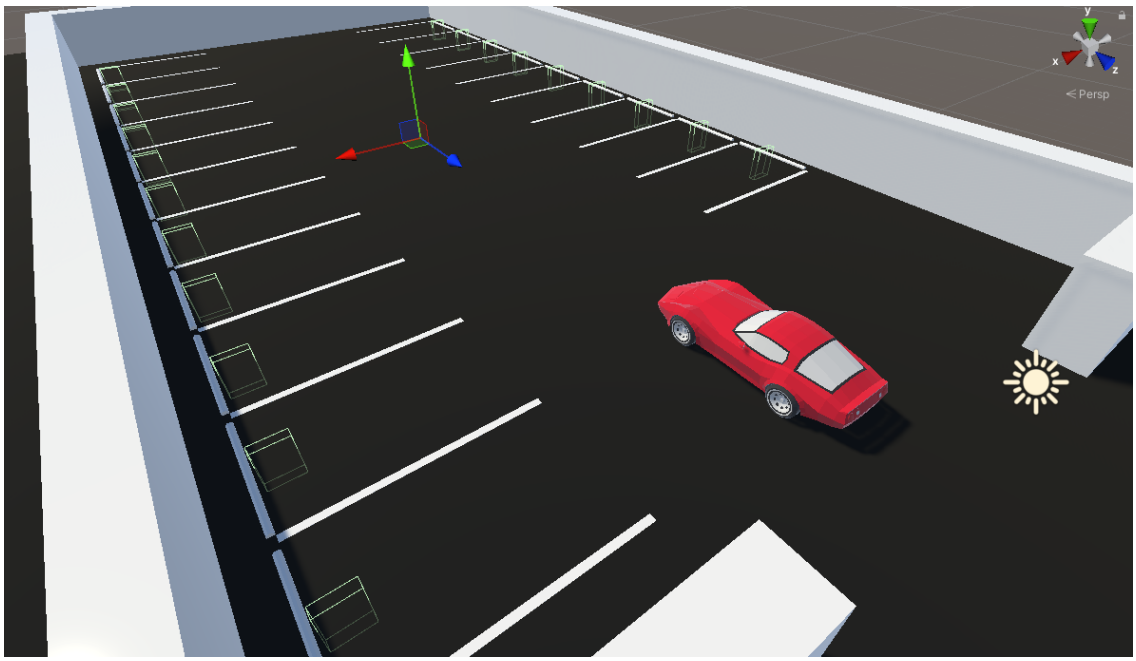

Figure 5 Parking slot objects

Parking slot objects are modeled to determine whether parking is completed or not. Each of the paring slot objects has its own variables that are angle alignment and linear distance between the agent car object. The simulator monitors these variables.

C. Agent

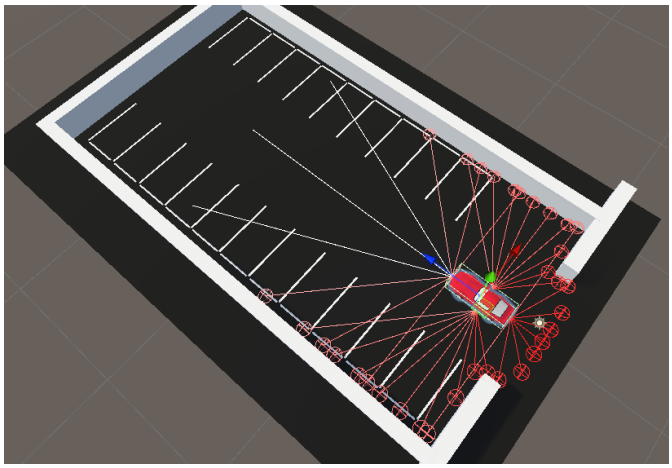

Figure 6 Overview of agent object with ray sensors

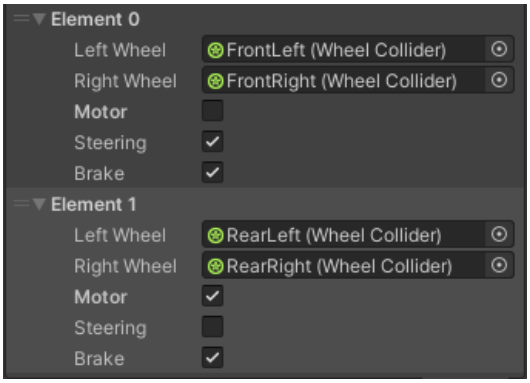

Figure 7 Movement elements for agent object

Agent object is modeled to imitate an actual car. Unity 3D ray sensors are used to implement LiDAR sensor of autonomous vehicle. These sensors detect the environments such as obstacles (parked cars and walls) and empty slots during RL episodes. And the object is made to move like an actual car. The object has two axles: front and rear. The front axle steers, and the rear axle accelerates just like a normal rear wheel drive car.

D. Reward

The simulator monitors the alignment and distance variables of every parking slot object per frame. If a parking slot has a certain condition of alignment and distance, the simulator decides that parking is succeeded and gives following reward:

$$\frac{Alignment}{Distance \times Time} = \frac{\overrightarrow{r_{parkingslot}} \cdot \overrightarrow{r_{agent}}}{Distance \times EpisodeTime}$$

E. Parked Cars

Random cars are pre-parked at random slots at every episode start for generalized episodes. If the agent collides these parked cars or the walls, the simulator ends the episode and subtracts certain amount of reward.

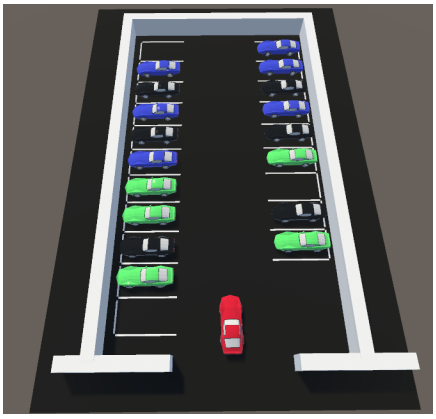

Figure 8 Parking lot packed with random cars

F. Algorithm

The algorithm for the entire simulation program introduced so far is the following:

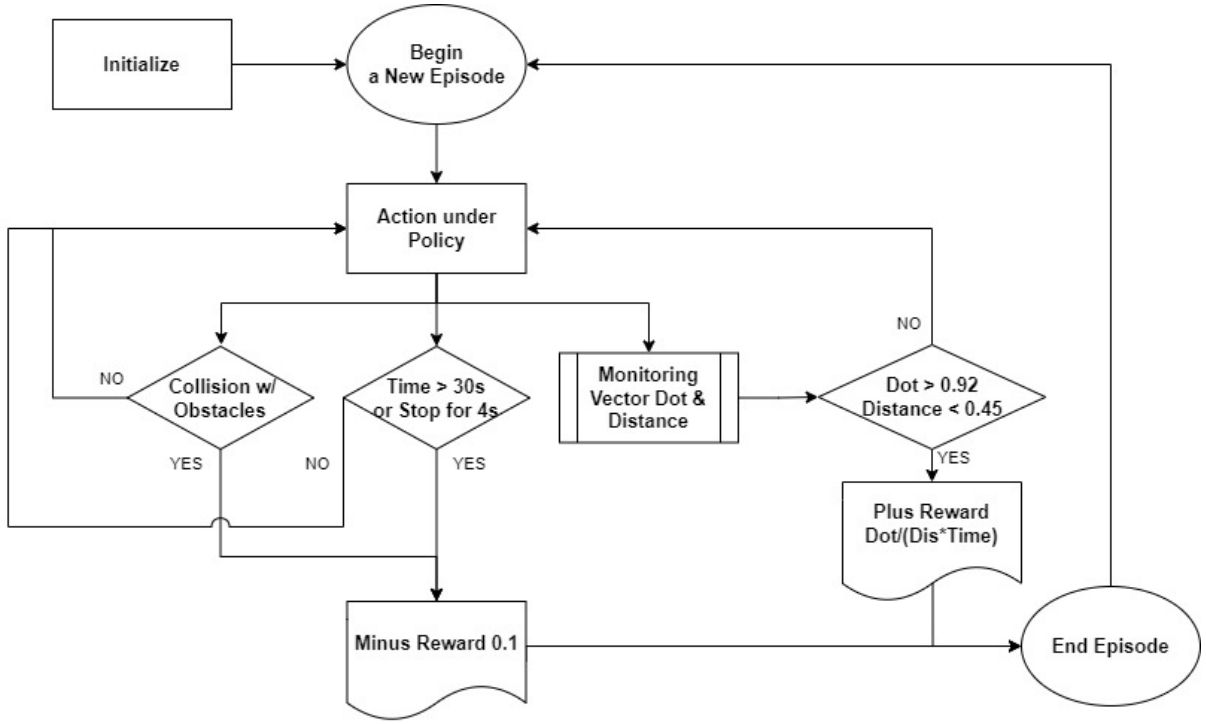

Figure 9 Algorithm for the simulator

(2) Learning

Reinforcement learning is done for each 3 of situations. The number of episodes and elapsed time are the following:

	Front/back	**Parallel**	**Diagonal**
Episodes	10,000,000	6,450,000	7,350,000
Elapsed Time	5h 57m 35s	12h 32m 43s	22h 38m 15s

Table 1 Learning process summary

(3) Result

A. Reward Graph

Episode-reward graphs for each situation are the followings:

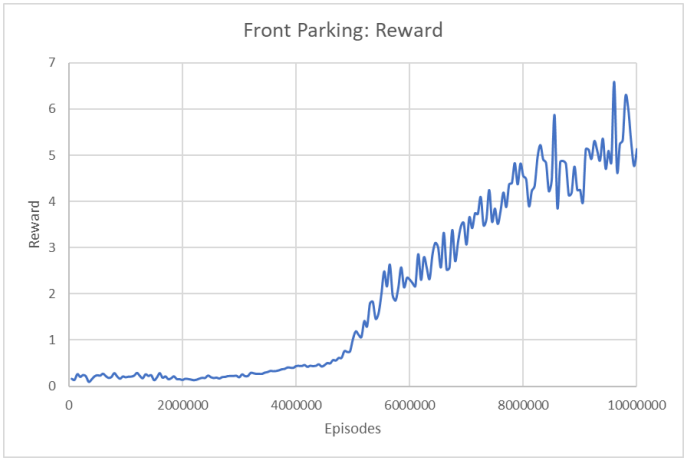

Figure 10 Reward graph for front/back parking

Figure 11 Reward graph for parallel parking

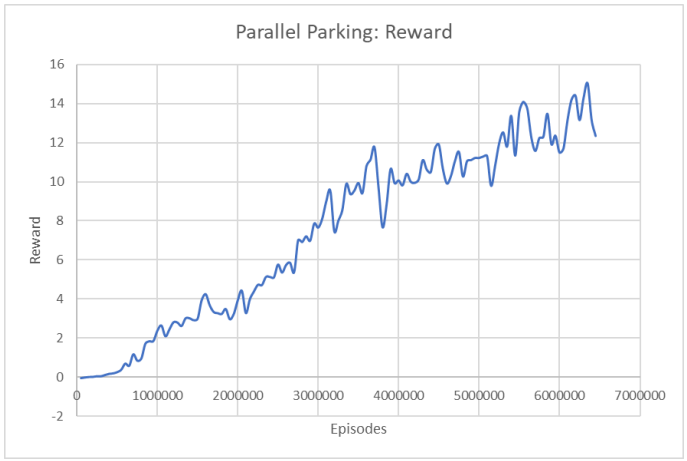

Figure 12 Reward graph for diagonal parking

B. Application

The result is exported as a PC application to show the autonomous parking.

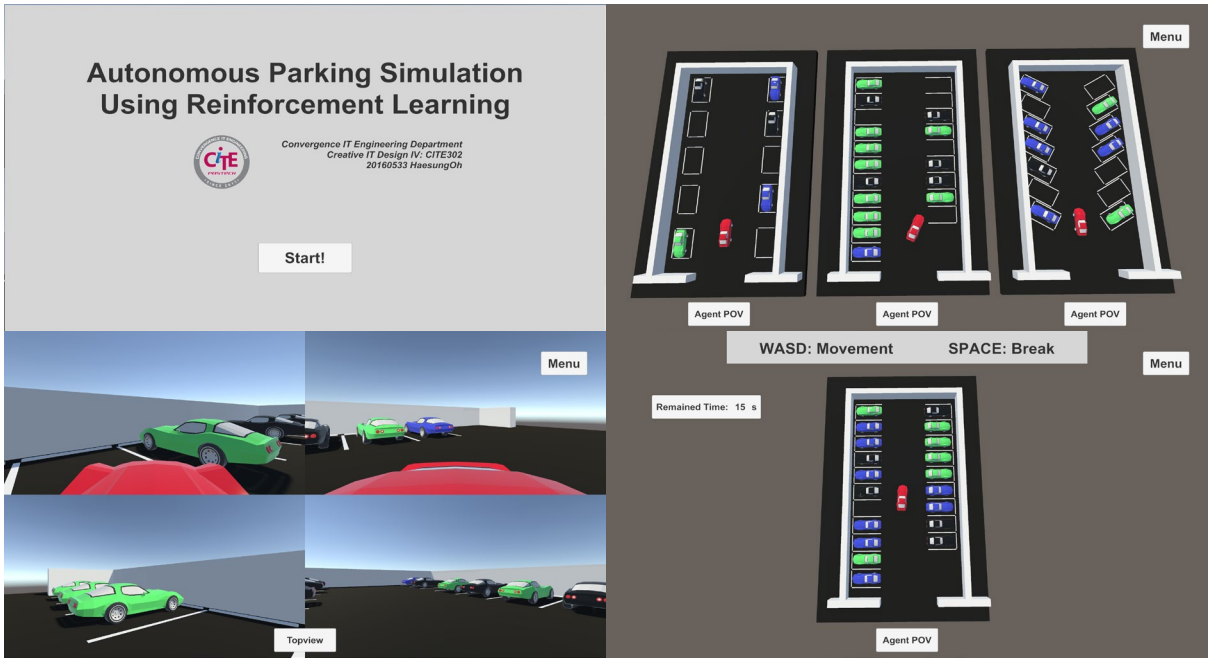

Figure 13 Overview of Autopark application

4. Summary

A. Conclusion

This program determines an autonomous parking algorithm using reinforcement learning and visualizes it. Not only the 3 situations in this program, but also other situations could be learned with this tool. And it could be implemented in real model by importing the behavior parameters. In summary, these results provide safe, generalized, and flexible autonomous parking simulation.

B. Limitation of the study, Future work

Due to practical reasons, this study could not unify the behavior parameters. Reinforcement learning has been done for each of 3 situations separately so that there 3 parameters exist. In further study should focus more generalized environments and episodes.