# Apprenticeship Learning Based on Deep Deterministic Policy Gradients for Autonomous Driving

Yongming Qin (yq9vc), Qiyu Zhang (qz5uw), Linfeng Wu (lw4fk), Zixuan Lin (zl7qk), Mengmeng Ye(my5an)

***Abstract*—Autonomous cars gain much attention over these years. In this reinforcement learning project, to produce continuous control variables for autonomous cars, the Deep Deterministic Policy Gradients (DDPG) algorithm is applied. Based on DDPG, we consider the apprenticeship learning in order to reduce the learning time before getting an acceptable agent and take advantage of expert experience. For offline case, the expert behavior is not involved in the learning process. For online case, at first both the expert and the actor of the algorithm provide actions. Finally the expert will not take part in the control anymore. Other than learning the actions that the expert demonstrates, we also make the agent learn actions that the expert does not demonstrate. The algorithms are tested in TORCS environment. Experiment results show reduction of learning time and effectiveness of learning more actions by the agent itself while learning the actions expert demonstrates using online learning. The offline learning does not have a good result. We state the possible reasons.**

***Index Terms*—DDPG, Apprenticeship Learning, TORCS, Autonomous Cars**

## I. Introduction

Autonomous Driving means a vehicle sense its environment around it and run without human input [1]. Driving is complex behavior for a human which needs skills of focusing, reacting and so on. With the development of artificial intelligence, achieving self-driving technology has become a long term goal. To deal with the potential influence of self-driving technique, a classification system based on six different levels of driving behavior (ranging from fully manual to fully automated systems) was published in 2014 by SAE International [2].

From the artificial intelligence aspect, there are three steps to achieve autonomous driving [3]. First, to sense the environment around the vehicle. Identifying components of the surrounding environment such as pedestrian, traffic sign, lane and so one is basic but import task for an autonomous car. At this step, we usually solve it with computer vision technology. Computer vision tasks include methods for acquiring, processing, analyzing and understanding digital images, and extraction of high-dimensional data from the real world in order to produce numerical or symbolic information [4]. For example, Deep learning models are able to learn complex feature representations from raw input data, omitting the need for handcrafted features [5]. Second, to predict. Recognizing is not enough to drive in a real road. It should have an inner model to predict the near future states of the environment. And the third is to plan which combine with recognizing and predicting together. The main difficulty is that it must be able to know about the environment and its dynamics in the lane and then try to generate the best reward to achieve best actions [3].

In our project we mainly focus on the second and third step. The simulator we used is TORCS whose full name is "The Open Racing Car Simulator". It is a highly portable multi-platform car racing simulation. It is used as ordinary car racing game, as AI racing game and as research platform. It could be used in different operation system, like linux, windows, Mac os, etc. [6]. The advantage of it is that it provide varies interfaces to offer data of the racing road, which means that developers don't need to deal with the recognizing part. Therefore, this simulator helped us to focus more on second and third step to build and train model. As for the algorithms of autonomous driving, DQN is a very popular and useful algorithm to achieve the goal. In 2015, Mnih et al. a member of DeepMind successfully created a convolutional neural network to calculate the Q function. And this framework could successfully plays more than Atari 2600 games at or significantly above professional human player ability [7].

Also, this algorithm has been adapted to the autonomous driving field. There are several successful work in different simulator using DQN [3], [7], [9]. However, there are also some shortcomings of DQN, one of them is that it can only deal with discrete space. In the real case, the driving behavior is more like a continuous behavior. Therefore, there always require more time and episodes to train a pure DQN model. So some improved algorithms came up, like Double DQN, Prioritised Replay, Dueling Network and so on [10]. Our work is based on DDPG algorithm [11] and we made some improvements with the knowledge of apprenticeship learning which shows in the following report [12].

## II. DDPG Algorithm

Traditional reinforcement learning algorithms were designed to deal with discrete space problems. For example, the outputs for Deep-Q-Network (DQN) are discrete. Since DQN is so powerful to solve high-dimensional problem, the biggest challenges for DQN is it will generate a huge amount of states-action pairs, which will result in the curse of dimensionality. In the autonomous vehicle environment, the action such as steering is continuous. If we discretize the problem, it will obvious encounter the curse of dimensionality problem. A continuous algorithm should be implemented to the autonomous vehicle racing. Google DeepMind has developed an solid algorithm for tackling the continuous action space problem, which is called Deep Deterministic Policy Gradient(DDPG) algorithm. DDPG is a model-free and off-policy algorithm. The basic idea of this algorithm is combing 3 algorithms together: 1) Deterministic Policy-Gradient Algorithms, 2) Actor-Critic Methods, 3)Deep-Q-Network [19].

### A. *Deterministic Policy-Gradient Algorithms*

The goal for this project is to define a policy network that implements autonomous driving. The policy is defined as below.

$$\pi_\theta(s,a) = P[a|s,\theta]$$

The input of the policy network is the state(for example: velocity of the car), the output of the policy is the action of the car(for example: turn left or right or brake). How to find the policy network? We'll define a policy objective function, which is the total discount reward, defined as below:

$$L(\theta) = E[r_1 + \gamma r_2 + \gamma^2 r_3 + ...|\pi_\theta(s,a)]$$

Which can also be written as:

$$L(\theta) = E_{x\sim_p(x|\theta)}[R]$$

The gradient of this deterministic policy is

$$\frac{\partial L(\theta)}{\partial \theta} = E_{x\sim_p(x|\theta)}[\frac{\partial Q}{\partial \theta}]$$

### B. *Actor-Critic Methods*

The actor-Critic algorithm combined the policy gradient algorithm and value function together. The policy function is called actor. The value function is called critic. Basically the actor will produce an action, the critic will criticize the action. Then based on the criticism, the actor will adjust its action and doing better next time. In our project, we are going to use policy gradient algorithm as actor model, SARSA as critic model.

### C. *Deep-Q-Network*

Deep-Q-Network is the first deep reinforcement learning method proposed by Google DeepMind. In DDPG algorithm, DQN is used to replace Q-function [17] [18]. The the Q function is written as

$$Q^\pi(s,a) \approx Q(s,a,w)$$

Where, w is the weight of the neural network. After the combination of these three different algorithms, the DDPG algorithm is show as below:

$$\frac{\partial L(\theta)}{\partial \theta} = \frac{\partial Q(s,a,w)}{\partial a}\frac{\partial a}{\partial \theta}$$

## III. Apprenticeship Learning of Our Project

### A. *Motivation*

While doing this project, we find the reinforcement learning process takes a lot of time before we see an acceptable agent behavior. Generally speaking, there are also many cases that people need an acceptable agent before a fully expert agent in practical. For example, if people do not have a simulation environment for their application. They need to use real devices to complete the reinforcement learning process. But as we know, experiments of real devices can cause damages to the devices especially for the first several episodes.

Also, sometimes before we want to have an good agent we human have the experience of how to do the task. This can be called expert experience. Taking advantage of expert experience can benefit us in the reinforcement learning process.

In this project, we use apprenticeship learning to get an acceptable agent in a shorter time than normal reinforcement learning [13]. In addition, we take advantage of expert experience and see if this can provide any insights. The idea is to "benefit from the expert agent but leave open the possibility of eventually performing even better" [14].

It is noted that this is not inverse reinforcement learning that tries to get one reward function [15]. In our project, we do not attempt to get this kind of reward functions. Instead we implant the expert experience into the neural network that DDPG uses.

### B. *Revisit of the training of DDPG*

Fig. 1 shows the training of the neural networks (NN) of the actor and the critic in DDPG. For the critic, normal NN training is used to create the relation from states and actions to the evaluation Q. The evaluation value is calculated using the reward function we created manually. The NN of the actor is trained using the gradient of the states with respect ot the actions.

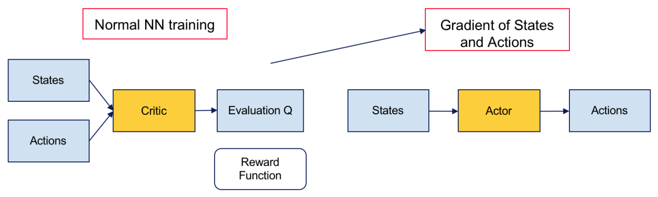


Fig. 1. Training of the neural networks of the actor and critic in DDPG

### C. *Training of neural networks in apprenticeship learning*

As for our apprenticeship learning algorithms, two major cases are considered – offline learning and online learning. For the **offline learning**, we use the behaviors of the experts to train the NN of the actor and critic. Then we start the reinforcement learning process based on the NN we trained. As the expert behavior is not involved in the reinforcement learning process, we call this offline learning.

As shown in Fig. 2, before the reinforcement learning process, both the critic NN and the actor NN are trained using normal NN training while the expert is demonstrating and we use the producing data.

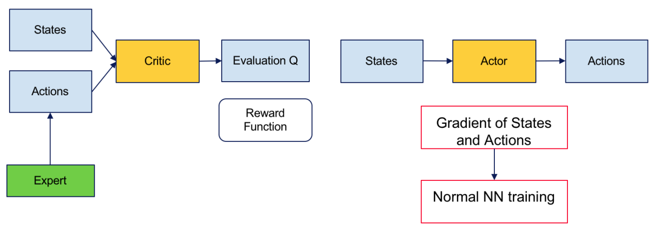


Fig. 2. Training of the neural networks of the actor and critic before reinforcement learning process in offline learning

For the **online learning**, the expert behavior is involved in the reinforcement learning process. We use a policy to choose actions between the actor and the expert. This policy ensures higher probability of participation of the expert in the first stage of learning and lower probability of participation in the following stage of learning. Finally, the expert does not affect the actions anymore. We hope till then the NN of the actor has mastered the experience of the expert.

In the online learning case, we consider two scenarios – **learning of actions from the expert behavior** and **learning of actions the expert does not demonstrate**. For the first learning scenario as shown in Fig. 3, the actions of the actor and the actions of the expert are same. For the second learning scenario as shown in Fig. 4, there are more actions of the actor than the expert's. This is to say we want the actor to learn actions by itself while learn the actions the expert demonstrates. Specifically in our project, we make the actor learn to use the brake while the expert never uses the brake.

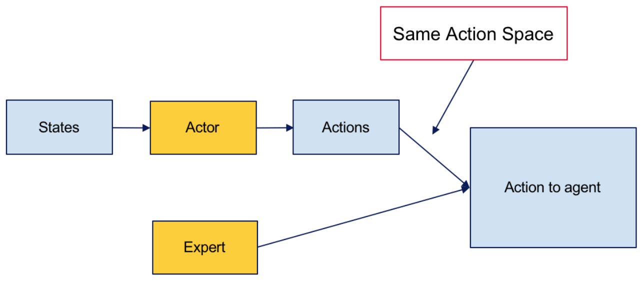


Fig. 3. Online learning of the neural network of the actor directly from the expert's behavior

## IV. Experiments

### A. Simulation Environment

To implement and test algorithms applied to autonomous vehicles, simulating the real driving scenes is necessary. Several simulators for drones as well as vehicles are available, including AirSim developed by Microsoft [20], Carla developed by Computer Vision Center and Intel [21], etc. They are both open-source simulators specially designed for autonomous driving research. Users are easy to add and edit code for their research projects. In both platform, several

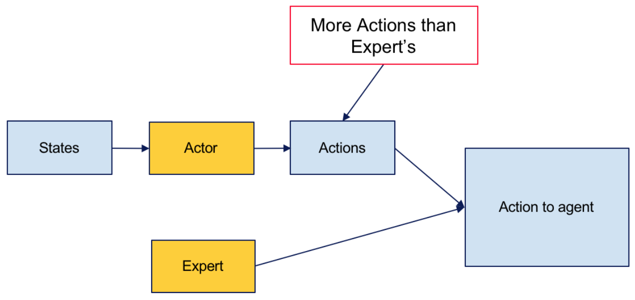


Fig. 4. Online learning of more actions compared with the expert's behavior

driving environments are provided which satisfy the needs of testing algorithms in different scenarios. Another advantage is they are cross-platform platforms compatible with Linux, Windows and Mac.

However, both AirSim and Carla have relatively high requirements for the computer configuration. Moreover, the environments provided involve a lot in computer vision area which requires more training time. For this project, we are only focused on reinforcement learning so these resources are not required.

The Open Racing Car Simulator (TORCS) is a highly portable car racing simulation [22]. Its high degree of modularity and portability render it ideal for articial intelligence research. There are several tracks available in TORCS, users can select different tracks individually for training and validation, which is helpful in avoiding over-fitting. Unlike AirSim and Carla depending on images of neighbourhood, the inputs for drivers in TORCS are a series of sensors measuring car status and car surroundings, which is more flexible to be processed. In this project, we use Gym-TORCS, a python wrapper of TORCS for RL experiment with the simple interface compatible with OpenAI-gym environments.

### B. Design of Experiments

*1) State Space:* TORCS perceives the racing environment through a number of sensor readings which provide information both about the surrounding game environment (e.g., the tracks, the opponents, the speed, etc.) and the current state of the race (e.g., the current lap time and the position in the race, etc.). There are totally 18 sensors. We selected 8 among them which are useful in autonomous driving. Their ranges and descriptions can be found in TABLE I.

Some of the sensors are vectors of multiple values. Combining these values together, there are 28 values among all the selected sensors. The agent uses these 28 values as the state space.

*2) Action Space:* We define three continuous variables as our action space: steering, acceleration and brake. They are all single-unit. For steering, it ranges from -1 to +1, where -1 means max right turn and +1 means max left turn. For acceleration, it ranges from 0 to +1, where 0 means no gas, 1 means full gas. For brake, it also ranges from 0 to +1, where 0 means no brake, 1 full brake.

TABLE I
DESCRIPTION OF SELECTED SENSORS. RANGES ARE REPORTED WITH THEIR UNIT OF MEASURE (WHERE DENED).

| Name | Range (unit) | Description |
|---|---|---|
| angle | [$-\pi, +\pi$]$(rad)$ | Angle between the car direction and the direction of the track axis. |
| track | [0,200] (m) | Vector of 19 range nder sensors: each sensors returns the distance between the track edge and the car within a range of 200 meters. |
| trackPos | $(\infty, +\infty)$ | Distance between the car and the track axis. The value is normalized w.r.t to the track width: it is 0 when car is on the axis, -1 when the car is on the right edge of the track and +1 when it is on the left edge of the car. Values greater than 1 or smaller than -1 mean that the car is outside of the track. |
| speedX | $(\infty, +\infty)(km/h)$ | Speed of the car along the longitudinal axis of the car. |
| speedY | $(\infty, +\infty)(km/h)$ | Speed of the car along the transverse axis of the car. |
| speedZ | $(\infty, +\infty)(km/h)$ | Speed of the car along the Z axis of the car. |
| wheelSpinVel | [0,+$\infty$]$(rad/s)$ | Vector of 4 sensors representing the rotation speed of wheels. |
| rpm | [0,+$\infty$)$(rpm)$ | Number of rotation per minute of the car engine. |

*3) Rewards:* In car racing, we want to maximize the velocity of the car, meanwhile, keep the car in the center of the track and not hit the edge. Thus, we get the reward function as following:

$$R = V_x cos(\theta) - V_x sin(\theta)) - V_x \left|trackPos\right|$$

We want to maximize the longitudinal velocity (first term), minimize transverse velocity (second term), and we also penalize the agent if it constantly drives very off center of the track (third term).

*4) Exploration:* In this project, $\epsilon$-greedy policy doesn't work well, since we have three actions. If we just randomly choose the action from uniform random distribution, we will generate some combinations that make no sense. For example, the value of the brake is greater than the value of acceleration. Here, we add noise using Ornstein-Uhlenbeck process to do the exploration. OU process is simply a stochastic process which has mean-reverting properties. We add noise which has the following format:

$$dx_t = \theta(\mu - x_t)dt + \sigma dW_t$$

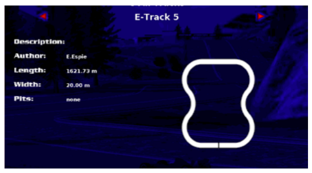
here, $\theta$ means the how "fast" the variable reverts towards to the mean, $\mu$ represents the equilibrium or mean value, $\sigma$ is the degree of volatility of the process. The values we use can be found in TABLE II.

TABLE II
PARAMETERS USED IN ORNSTEIN-UHLENBECK PROCESS

| Action | $\theta$ | $\mu$ | $\sigma$ |
|---|---|---|---|
| Steering | 0.6 | 0.0 | 0.30 |
| Acceleration | 1.0 | [0.3-0.6] | 0.10 |
| Brake | 1.0 | -0.1 | 0.05 |

## V. RESULTS AND CONCLUSION

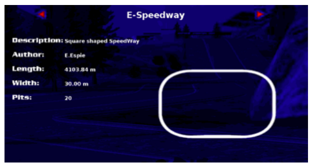
In order to test DDPG algorithm and Apprenticeship Learning, different speedways are chosen. Fig. 5 and Fig. 6 show the overview graphs of two tracks.



E-Track 5

Description:
Author: E.Espie
Length: 1621.73 m
Width: 20.00 m
Pits: none

Fig. 5. The E-Track 5 track



E-Speedway

Description: Square shaped SpeedWay
Author: E.Espie
Length: 4103.84 m
Width: 30.00 m
Pits: 20

Fig. 6. The E-Speedway track

### A. Metrics

To evaluate the performance and compare the results of different algorithms, below metrics are chosen.

- Learning time.
- Relation between episode number and step number.
- How many episodes does it take for the learning of the car to pass the first S curve of the speedway shown in Fig. 5?
- The behavior of using brake.

### B. Result of original DDPG

We train the DDPG neural network for 1000 episodes and allow the OU process decay linearly. In this process it is important to test the AI agents in other tracks before overfitting. We evaluated the policy periodically during training by testing its exploration noise. In order to perform well across all tasks, both of these additions are necessary. In general, learning with original DDPG is very slow.

Fig. 7 and Fig. 8 show the performance of DDPG algorithm for E-Speedway track, the termination condition is finishing 8 laps. After approximately 250 episodes, the car can learn to complete 8 laps. Fig. 9 shows the information of each lap.

From the results we can see DDPG algorithm can learn a reasonable policy. It is a powerful, model-free, continuous control algorithm. However, when the number of actions increase, the number of combinations increase and it is not obvious to me how to do the exploration.

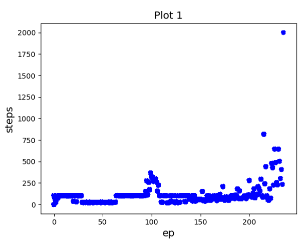


Fig. 7. Steps of each episode

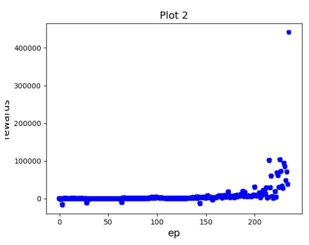
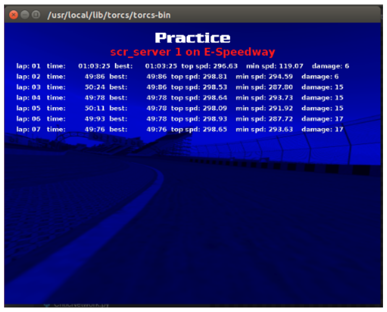


Fig. 8. Rewards of each episode



```
/usr/local/lib/torcs/torcs-bin
Practice
scr_server 1 on E-Speedway
lap: 01  time:  01:03:25  best:  01:03:25  top spd: 296.63  min spd: 119.07  damage: 6
lap: 02  time:     49:86  best:     49:86  top spd: 298.81  min spd: 294.59  damage: 6
lap: 03  time:     50:24  best:     49:86  top spd: 298.53  min spd: 287.80  damage: 15
lap: 04  time:     49:78  best:     49:78  top spd: 298.64  min spd: 293.73  damage: 15
lap: 05  time:     50:11  best:     49:78  top spd: 298.09  min spd: 291.92  damage: 15
lap: 06  time:     49:93  best:     49:78  top spd: 298.93  min spd: 287.72  damage: 17
lap: 07  time:     49:76  best:     49:76  top spd: 298.65  min spd: 293.63  damage: 17
```

Fig. 9. Testing result for 7 laps

## C. *Offline Learning of apprenticeship learning*

For offline learning, it turns out that it does not have an obvious effect on reducing the learning time. Using the NN trained by observing the expert behavior, if we only use the NN without further training (which is a pure training algorithm), the agent performs very well interestingly. However, after starting the learning process, the can runs longer than normal DDPG but does not have a reasonable behavior. It goes straight towards the boarder of the track at the first corner, but does not show appropriate wheel action in following episodes. The reason we think is that **the NN trained based on expert behavior does not contain the information dealing with bad situations.**

## D. *Online Learning of apprenticeship learning*

In our experiments, online learning produces good results.

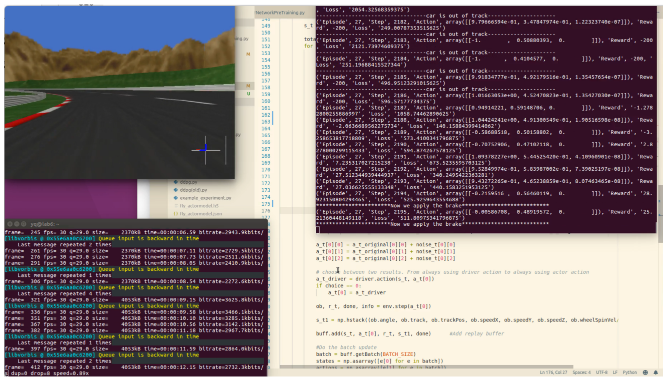
*1) Learning of actions from the expert behavior:* TABLE III shows the learning time of getting an acceptable agent of original DDPG and Apprenticeship Learning. Here, the judgment is based on the metrics mentioned in section V-A.

TABLE III
THE LEARNING TIME OF GETTING AN ACCEPTABLE AGENT OF DDPG AND APPRENTICESHIP LEARNING

| Method | DDPG | Apprenticeship Learning |
|---|---|---|
| Time (E-Speedway) | 3 hours | 30 minutes |
| Time (E-Track 5) | - | 30 minutes |
| Episode | 200 | 80 |

We can see that the apprenticeship learning (Online) helps us to get an acceptable agent quickly. The car can even pass the first S curve of E-Track 5 track at the 20th episode for some tests. The result verifies the intuitive expectation – some rules can help the AI learn quickly.

*2) Learning of actions the expert does not demonstrate:* Fig. 10 shows the behavior of the agent after learning and exploration. This is to say the action of braking is caused by the actor based on previous learning. We can see the agent uses brake when it is passing a curve. This is a reasonable behavior from the view of our human drivers.



```
, 'Loss', '2054.32568359375')
---------------------------------car is out of track--------------------
('Episode', 27, 'Step', 2182, 'Action', array([[9.79666594e-01, 3.47847974e-01, 1.22323740e-07]]), 'Rewa
rd', -200, 'Loss', '249.00787353515625')
---------------------------------car is out of track--------------------
('Episode', 27, 'Step', 2183, 'Action', array([[-1.        ,  0.50880393,  0.        ]]), 'Reward', -200
, 'Loss', '2121.73974609375')
---------------------------------car is out of track--------------------
('Episode', 27, 'Step', 2184, 'Action', array([[-1.        ,  0.4104577,  0.        ]]), 'Reward', -200, '
Loss', '251.19688415527344')
---------------------------------car is out of track--------------------
('Episode', 27, 'Step', 2185, 'Action', array([[9.91834777e-01, 4.92179516e-01, 1.35457654e-07]]), 'Rewa
rd', -200, 'Loss', '496.95123291015625')
---------------------------------car is out of track--------------------
('Episode', 27, 'Step', 2186, 'Action', array([[1.01663053e+00, 4.52470823e-01, 1.35427030e-07]]), 'Rewa
rd', -200, 'Loss', '596.57177734375')
('Episode', 27, 'Step', 2187, 'Action', array([[0.94914221, 0.59148706, 0.        ]]), 'Reward', '-1.278
280025586997', 'Loss', '1058.74462890625')
('Episode', 27, 'Step', 2188, 'Action', array([[1.04424241e+00, 4.91300549e-01, 1.90516598e-08]]), 'Rewa
rd', '-2.0636689562275734', 'Loss', '140.15884399414062')
('Episode', 27, 'Step', 2189, 'Action', array([[-0.58688518,  0.50158802,  0.        ]]), 'Reward', '-3.
258653817718809', 'Loss', '573.4100341796875')
('Episode', 27, 'Step', 2190, 'Action', array([[-0.70752906,  0.47102118,  0.        ]]), 'Reward', '2.8
278000299115433', 'Loss', '594.874267578125')
('Episode', 27, 'Step', 2191, 'Action', array([[1.09378227e+00, 5.44525420e-01, 4.10960901e-08]]), 'Rewa
rd', '7.235317027215238', 'Loss', '673.5235595703125')
('Episode', 27, 'Step', 2192, 'Action', array([[9.52849974e-01, 5.83987002e-01, 7.39025197e-08]]), 'Rewa
rd', '27.51234493944937', 'Loss', '340.2495422363281')
('Episode', 27, 'Step', 2193, 'Action', array([[9.43272265e-01, 4.65238859e-01, 8.07463465e-08]]), 'Rewa
rd', '27.036625555133348', 'Loss', '440.1583251953125')
('Episode', 27, 'Step', 2194, 'Action', array([[-0.2159516 ,  0.56460119,  0.        ]]), 'Reward', '28.
923150804294465', 'Loss', '525.9259643554688')
**********************Now we apply the brake***************************
('Episode', 27, 'Step', 2195, 'Action', array([[-0.00586708,  0.48919572,  0.        ]]), 'Reward', '25.
21360448149118', 'Loss', '511.80975341796875')
**********************Now we apply the brake***************************
```

```python
a_t[0][0] = a_t_original[0][0] + noise_t[0][0]
a_t[0][1] = a_t_original[0][1] + noise_t[0][1]
a_t[0][2] = a_t_original[0][2] + noise_t[0][2]

# choose between two results. From always using driver action to always using actor action
a_t_driver = driver.action(s_t, a_t[0])
if choice == 0:
    a_t[0] = a_t_driver

ob, r_t, done, info = env.step(a_t[0])

s_t1 = np.hstack((ob.angle, ob.track, ob.trackPos, ob.speedX, ob.speedY, ob.speedZ, ob.wheelSpinVel/

buff.add(s_t, a_t[0], r_t, s_t1, done)      #Add replay buffer

#Do the batch update
batch = buff.getBatch(BATCH_SIZE)
states = np.asarray([e[0] for e in batch])
```

```
yq@lab6: ~
frame=  245 fps= 30 q=29.0 size=    2370kB time=00:00:06.59 bitrate=2943.9kbits/
[libvorbis @ 0x55e6aa0c6200] Queue input is backward in time
    Last message repeated 2 times
frame=  261 fps= 30 q=29.0 size=    2370kB time=00:00:07.11 bitrate=2729.5kbits/
frame=  276 fps= 30 q=29.0 size=    2370kB time=00:00:07.73 bitrate=2511.6kbits/
frame=  291 fps= 30 q=29.0 size=    2370kB time=00:00:08.05 bitrate=2410.9kbits/
[libvorbis @ 0x55e6aa0c6200] Queue input is backward in time
    Last message repeated 1 times
frame=  306 fps= 30 q=29.0 size=    2370kB time=00:00:08.54 bitrate=2272.6kbits/
[libvorbis @ 0x55e6aa0c6200] Queue input is backward in time
    Last message repeated 4 times
frame=  321 fps= 30 q=29.0 size=    4053kB time=00:00:09.15 bitrate=3625.8kbits/
[libvorbis @ 0x55e6aa0c6200] Queue input is backward in time
frame=  336 fps= 30 q=29.0 size=    4053kB time=00:00:09.58 bitrate=3466.1kbits/
frame=  351 fps= 30 q=29.0 size=    4053kB time=00:00:10.10 bitrate=3285.1kbits/
frame=  367 fps= 30 q=29.0 size=    4053kB time=00:00:10.56 bitrate=3142.1kbits/
frame=  382 fps= 30 q=29.0 size=    4053kB time=00:00:11.18 bitrate=2967.7kbits/
[libvorbis @ 0x55e6aa0c6200] Queue input is backward in time
    Last message repeated 1 times
frame=  397 fps= 30 q=29.0 size=    4053kB time=00:00:11.59 bitrate=2864.0kbits/
[libvorbis @ 0x55e6aa0c6200] Queue input is backward in time
    Last message repeated 2 times
frame=  412 fps= 30 q=29.0 size=    4053kB time=00:00:12.15 bitrate=2732.3kbits/
s dup=0 drop=8 speed=0.89x
```

Fig. 10. The car learns to use brake which the expert does not demonstrate

## VI. Challenge and Future Works

### A. *Challenge*

During the project, we met lots of problems. The most difficult one is that we found that the online learning (DDPG) sometimes will not converge or will work in a very long time. We tried to find reasons and solutions. In the issues of DDPG-Keras-Torcs project (https://github.com/yanpanlau/DDPG-Keras-Torcs/issues), we summarized several reasons of this problem. (1) The actions at beginning steps are random, so that some incredible actions, such as run to the wall directly, may influence the following trend. It's true that it has possibility to succeed in a short episodes but sometimes it may take a longer time or never converge at all; (2) There is lag between the controller(your computer) and the game simulator. So sometimes the network in the model can not receive data on time and to feedback the actions. Thus, sometimes it would go wrong.

To solve this, we came up with solutions containing two methods. (1) Training and testing the model in a simple map and using a better computer (with better CPU) will solve this problem in some degrees. (2) Using offline model will be much better. Actually, DDPG algorithm is not that reasonable enough. The first steps are stochastic because of the policy network. However, we know that in the real-life case if someone want to learn to drive, he or she will not drive just randomly because there are some basic rules of driving that everyone should know. And that's why we used Apprenticeship Learning. method. So first we trained a basic driving policy and then let the DDPG use the model directly. The result is much better than before. We have updated this method in our Github. (https://github.com/yongming-qin/auto_car_experience_training )

### B. Future Works

About the future work, we have three ways to improve our project. (1) First, We would like to promote the reward function. The current reward function is simple and may influence the performance of vehicle. Our next step is to contain apprenticeship learning in our reward function. For example, if the action generate by our model is similar or same with the action of the "expert" (a given trained model), it will get more reward. We believe combining with "expert", the vehicle may learn quicker and perform better; (2) Second, we are planning to deal with more difficult and complex simulator which means longer time to do that. Specially, some complex simulator like Airsim and Carla will be hard to achieve our goal because the scene is so realistic and in it there are some shadows or curb that is hard for vehicle to detect and avoid. (3) To solve this problem the third method is using computer vision. Our current autonomous vehicle gets the data from the interface instead of image processing. So our next steps is to realize an end to end project which contains recognizing.

## Acknowledgment

Our ideas are inspired by [14] and [16]. The test platform is built according to a blog called using Keras and deep deterministic policy gradient to play TORCS.

## References

[1] Gehrig, Stefan K., and Fridtjof J. Stein. "Dead reckoning and cartography using stereo vision for an autonomous car." Intelligent Robots and Systems, 1999. IROS'99. Proceedings. 1999 IEEE/RSJ International Conference on. Vol. 3. IEEE, 1999.

[2] Automated driving levels of driving automation are defined in new SAE international statdrd J3016 (PDF). 2017. Archived from the original (PDF) on 20 November 2016.

[3] Sallab, Ahmad EL, et al. "Deep reinforcement learning framework for autonomous driving." Electronic Imaging 2017.19 (2017): 70-76.

[4] Reinhard Klette (2014). Concise Computer Vision. Springer. ISBN 978-1-4471-6320-6.

[5] Badrinarayanan, V., Kendall, A., Cipolla, R. (2015). SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. arXiv preprint arXiv:1511.00561.

[6] https://sourceforge.net/projects/torcs/

[7] Matiisen, Tambet (December 19, 2015). "Demystifying Deep Reinforcement Learning — Computational Neuroscience Lab". neuro.cs.ut.ee. Retrieved 2018-04-06.

[8] Wang, Pin, and Ching-Yao Chan. "Formulation of Deep Reinforcement Learning Architecture Toward Autonomous Driving for On-Ramp Merge." arXiv preprint arXiv:1709.02066 (2017).

[9] Isele, David, et al. "Navigating Occluded Intersections with Autonomous Vehicles using Deep Reinforcement Learning." arXiv preprint arXiv:1705.01196 (2017).

[10] "Information about duelingnetwork.com: Dueling Network". dig.do. Retrieved 19 March 2015.

[11] Lillicrap, Timothy P., et al. "Continuous control with deep reinforcement learning." arXiv preprint arXiv:1509.02971 (2015).

[12] Pieter Abbeel, Andrew Ng, "Apprenticeship learning via inverse reinforcement learning." In 21st International Conference on Machine Learning (ICML). 2004.

[13] Argall, Brenna D., et al. "A survey of robot learning from demonstration." Robotics and autonomous systems 57.5 (2009): 469-483.

[14] Sutton, Richard S., and Andrew G. Barto. "Reinforcement learning: An introduction." (2011), Chapter 17.

[15] Abbeel, Pieter, and Andrew Y. Ng. "Apprenticeship learning via inverse reinforcement learning." Proceedings of the twenty-first international conference on Machine learning. ACM, 2004.

[16] Sallab, Ahmad EL, et al. "Deep reinforcement learning framework for autonomous driving." Electronic Imaging 2017.19 (2017): 70-76.

[17] V. Mnih, K. Kavukcuoglu, D. Silver, et al, *Human-level Control Through Deep Reinforcement Leanring*. Nature, 2015.

[18] V. Mnih, K. Kavukcuoglu, D. Silver, et al, *Playing Atari with Deep Reinforcement Learning*. NIPS, 2013.

[19] D. Silver, G. Lever, N. Heess, et al,*Deterministic Policy Gradient Algorithms*.

[20] S. Shah, D. Dey, C. Lovett and A. Kapoor, *AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles*. In Field and Service Robotics, 2017. URL https://arxiv.org/abs/1705.05065.

[21] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez and V. Koltun, *CARLA: An Open Urban Driving Simulator*. Proceedings of the 1st Annual Conference on Robot Learning:1–16, 2017

[22] B. Wymann, C. Dimitrakakis, A. Sumner, E. Espie and C. Guionneau, *TORCS: The open racing car simulator*. http://www.torcs.org, 2014.

[23] D. Loiacono, L. Cardamone and P. L. Lanzi *Simulated Car Racing Championship Competition Software Manual*. Technical report.Italy: Dipartimento di Elettronica e Informazione, Politecnico di Milano, 2011.