Published 09/23/2017

doi:10.4271/2017-01-1960

Obstacle Avoidance for Self-Driving Vehicle with Reinforcement Learning

Xiaopeng Zong, Guoyan Xu, Guizhen Yu, Hongjie Su, and Chaowei Hu
Beihang University

ABSTRACT

Obstacle avoidance is an important function in self-driving vehicle control. When the vehicle move from any arbitrary start positions to any target positions in environment, a proper path must avoid both static obstacles and moving obstacles of arbitrary shape. There are many possible scenarios, manually tackling all possible cases will likely yield a too simplistic policy. In this paper reinforcement learning is applied to the problem to form effective strategies. There are two major challenges that make self-driving vehicle different from other robotic tasks. Firstly, in order to control the vehicle precisely, the action space must be continuous which can't be dealt with by traditional Q-learning. Secondly, self-driving vehicle must satisfy various constraints including vehicle dynamics constraints and traffic rules constraints. Three contributions are made in this paper. Firstly, an improved Deep Deterministic Policy Gradients (DDPG) algorithm is proposed to solve the problem of continuous action space, so that the continuous steering angle and acceleration can be obtained. Secondly, according to the vehicle constraints include inside and outside, a more reasonable path for obstacle avoidance is designed. Thirdly, the various sensors data are merged into vehicle so as to satisfy the need of the input information of the algorithm, including the vehicle state and surrounding environment state. In addition to that, the algorithm are tested on an open source vehicle simulator for racing called TORCS which stands for The Open Racing Car Simulator. The result demonstrate the effectiveness and robustness of the method.

CITATION: Zong, X., Xu, G., Yu, G., Su, H. et al., "Obstacle Avoidance for Self-Driving Vehicle with Reinforcement Learning," *SAE Int. J. Passeng. Cars – Electron. Electr. Syst.* 11(1):2018, doi:10.4271/2017-01-1960.

INTRODUCTION

The self-driving vehicle is a robot system that can move from any arbitrary start positions to any target position in environment, which contain both static and moving obstacle of arbitrary shape. Obstacle avoidance is a basic function for self-driving vehicle control. In order to improve the intelligence of self-driving vehicle, the control algorithm should consider a variety of environment situation, and imitate the excellent driver's operation. The existing decision making module for self-driving vehicle is generally constructed according to various rules. However, experience programming can't enumerate all situation which may occur in a variety of emergencies, so it's necessary to raise a more intelligent algorithm to deal with the various situation in the driving environment [1].

With the development of reinforcement learning (RL), more and more researcher apply it to self-driving vehicle control. The purpose of RL is to learn how to take the optimal behavior by interacting with the environment. Compared to traditional machine learning, it has the following advantages: Firstly, because there is no need to label samples, it can be more effective to solve the special circumstances in environment. Secondly, it is more robust to some modules as it treat the system as a whole. Finally, RL can be easier to learn a series of behaviors. These features are very suitable for self-driving vehicle decision-making process.

The deep reinforcement learning that combine the deep learning (DL) and RL is a most promising approach to achieve human level control. In Deep Q Networks (DQN) model, DL is responsible for the sensor data processing, while RL is responsible for the policy decision. It has had successful application in playing Atari games at human level using unprocessed pixels for input [2]. However, DQN can only solve problem with discrete and low-dimensional actions spaces, it can't handle continuous and high dimensional action space which most control tasks possess. The self-driving vehicle control is a typical continuous control process which contain continuous steering angle and acceleration value.

An obvious approach to adapting DQN to continuous filed is to simply discretize the action space, but it could raise the problem of dimensionality disaster. Additionally, simple artificial division for the action space will lose the control accuracy. The Deep Deterministic Policy Gradients (DDPG) algorithm which combine Deterministic Policy-Gradient algorithms, Actor-Critic methods and DQN can perform well in the continuous control domain.

In this paper, a framework is proposed for self-driving vehicle model considering vehicle sensor inputs and outputs driving actions. At first, the DDPG algorithm structure is analyzed. Then the internal and external constraints for self-driving vehicle are discussed, and the reward function based on the vehicle constraints is designed.

In order to evaluate the obstacle avoidance method, a variety of testing scenarios in TORCS that involve various tracks and competitive vehicles are constructed. The approach is tested by applying it to tackle two major obstacle including static obstacles and moving obstacles. The result shows that, through a period of self-learning, self-driving vehicle can effectively learn sophisticated behaviors and perform well in new testing environment.

BACKGROUND

Reinforcement Learning

The most basic model of RL is shown as figure 1[3].

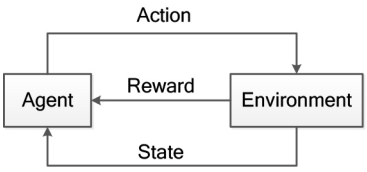

Figure 1. The most basic model of RL.

In the model of RL, the agent interacts with environment. In every discrete time t, the agent choose the action a_t according to current state s_t, and the environment change its state to s_{t+1}, then the agent get reward r_t which used to evaluate the impact of the strategy.

The goal of RL is to maximize the discounted accumulative reward $R_t = \sum_{i=t}^{T} \gamma^{i-t} r(s_i, a_i)$ [4]. Q-learning is the most commonly used algorithm, which utilize the action-value to express the expectation of R_t: $Q(s_t, a_t) = E(R_t|s_t, a_t)$. The optimal action-value function is the maximum achievable expected under the strategy π,

$$Q^*(s,a) = max_\pi E(R_t|s_t = s, a_{t=a}, \pi) \tag{1}$$

It can be expressed by Bellman equations,

$$Q^*(s,a) = E[r + \gamma max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1})|s,a] \tag{2}$$

It is common to use neural networks to estimate action values, which assume the approximators as $Q(s, a; \theta) \approx Q^*(s, a)$ with the parameters θ. The purpose of learning networks is to minimize the loss function,

$$L_t(\theta_t) = E[(r + \gamma max_{a_{t+1}} Q(s_{t+1}, a_{t+1}; \theta_{t-1}) - \cdots \\ -Q(s_t, a_t; \theta_t))^2] \tag{3}$$

Deep Deterministic Policy Gradients

Deep Deterministic Policy Gradients (DDPG) is a model-free, off-policy Actor-Critic algorithm using deep function approximators that can learn policies in continuous action spaces [5].

The Actor-Critic algorithm is essentially a hybrid method to combine the policy gradient method and the value function method together. Policy function can be initialized using method like supervised learning. The learning of value function is very difficult when the environment is more complex. The learning difficulty can be reduced by separating the policy function and value function. The policy function is known as the actor, while the value function is referred to as the critic. The actor produces the action ***a*** given the current state of the environment ***s***, while the critic produces a signal to criticize the actions made by the actor. So the input of actor network is state, output is action, while the input of critic network is state and action, output is Q value. The model of Actor-Critic is shown as figure 2.

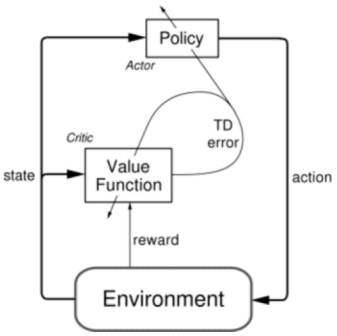

Figure 2. The model of Actor-Critic method.

It is well known that estimating the value function by non-linear function is difficult and unstable. Introducing neural network function approximators is introduced to adapt the large state spaces, but the convergence is no longer guaranteed. The DDPG solve this issue by two improvements. The network is trained off-policy with samples from a replay buffer to minimize correlations between samples. The replay buffer is a finite sized cache R, the tuple (s_t, a_t, r_t, s_{t+1}) sampling from the environment according to the exploration policy is stored in the replay buffer. The network is trained with a target Q network to give consistent targets during temporal difference backups. The weights for actor and critic target networks are updated by having them slowly track the learned network: $\theta' \leftarrow \tau\theta + (1 - \tau)\theta'$ with $\tau \ll 1$. This means that the target values are constrained to change slowly, greatly improving the stability of learning.

A key problem of learning in continuous actions is exploration. The DDPG is an off-policy algorithm which can handle the problem of exploration independently from the learning algorithm. An Ornstein-Uhlenbeck process $\mathcal{N}$ is utilized to generate temporally noise sampled to actor policy:

$$\mu'(s_t) = \mu(s_t|\theta^\mu) + \mathcal{N}_t \tag{4}$$

$\mathcal{N}$ can be chosen to suit the environment.

The DDPG algorithm process is shown as Algorithm 1.

Randomly initialize critic network $Q(s,a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode =1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t=1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss:

$$L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$$

    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s,a|\theta^Q)\,|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

Algorithm 1. DDPG algorithm

Vehicle Constraints

Vehicle Dynamics Constraints

The self-driving vehicle achieve obstacle avoidance function by controlling the steering wheel angle and gas/brake pedals when driving on the road. The vehicle may skidding or even overturned if turn at high speed. In order to keep the vehicle safe, it is necessary to limit the vehicle steering angle based on actual vehicle speed. The relationship between steering angle and vehicle speed should meet the vehicle dynamics constraints. The most important dynamic constraint is that the lateral acceleration is not greater than 0.4g [6].

$$a_y \le 0.4g \tag{5}$$

Where the a_y means the lateral acceleration, the g means the acceleration due to gravity.

The relationship between the lateral acceleration and the front wheel angle and speed is as follows:

$$a_y = u\frac{u/L}{1+Ku^2}\delta \tag{6}$$

Where the u means the vehicle speed, the δ means front wheel angle, and the K is stability factor, which is calculated from the following formula:

$$K = \frac{m}{L}\left(\frac{a}{k_1} - \frac{b}{k_2}\right) \tag{7}$$

Where m is vehicle quality, L is vehicle wheelbase which the distance from the front axle to the rear axle, a is the distance from front axle to the center of gravity, b is the distance from rear axle to the center of gravity, k_1 and k_2 are the lateral stiffness of the front and rear wheels.

According to the constraints: $a_y \le 0.4g$

We know that:

$$u\frac{u/L}{1+Ku^2}\delta \le 0.4g \tag{8}$$

So the steering angle satisfy the following inequality:

$$\delta \le \delta_{max} = 0.4g\frac{1+Ku^2}{u^2/L} \tag{9}$$

Traffic Rules Constraints

In addition to consider vehicle dynamics constraints, the traffic rules constraints also should be contained. The typical traffic rules including traffic light, lane line, speed limit, etc. When self-driving vehicle make the lane change as the obstacle avoidance, it is essential to consider the traffic rules constraints. There is no traffic light in TORCS, so the lane line restrictions are mainly considered. The lane line could be divided into solid line which vehicle can't pass through and dash line which vehicle is allowed to across. This class constraint is considered by setting the form of reward function. The speed are limited to 120 km/h which considering the speed constraint on the highway.

Self-Driving Vehicle Simulator

The simulator plays a very important role in self-driving vehicle development. Simulator can simulate common scenes in the environment, such as lane conditions, road conditions, obstacle distribution and behavior, weather, and so on. The Open Racing Car Simulator is a widely used open source vehicle simulator which own multi-player, multi-agent, several tracks and different game modes (e.g., practice, quick race, championship, etc.). The player's task is to compete with other AI cars and reach the end as far as possible [7]. Although the tasks in TORCS are different from the real autonomous

driving, the feature of high degree of modularity and portability make it an ideal choice for artificial intelligence research. The architecture of the competition software is shown as figure 3[8].

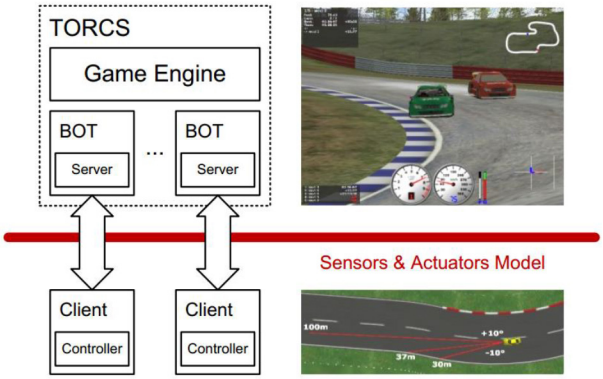

Figure 3. The architecture of the competition software.

Each vehicle can be controlled by an automated driver or bot which user designed for their control algorithm. At every control step, the driver can access the real-time state, including various information about the vehicle and the environment. The other vehicle information on the track also could be obtained. The vehicle is controlled by gas/ brake pedals, the gear stick, and the steering wheel.

ALGORITHM DESIGN

The environment status are used as input of DDPG. The environment reward is defined as the speed of the vehicle along the track at each moment. In addition, the vehicle will get additional penalty if collision occur or vehicle out of track.

Sensors and Variable Definitions

Sensors

The self-driving vehicle are equipped with a variety of sensors to sense the environment information. In this work the major sensors include GPS, LIDAR and ultrasonic radar. GPS can give real-time position information and vehicle driving status, including heading angle, vehicle speed and so on. The LIDAR can detect the obstacle information around the vehicle. It is used to detect the front obstacle vehicle in this paper. Ultrasonic radar are distributed around the vehicle body and are used to detect the road edge in real environment.

Input Variable

In TORCS, the self-driving vehicle perceives the environment status through kinds of sensors which provide useful information about the vehicle status (e.g., the current gear, the fuel level, etc.) and the vehicle surrounding environment (e.g., the track borders, the obstacles, etc.). In order to maximize simulate the real vehicle, the key sensors data are chosen to describe the vehicle status, these data can be obtained by real sensors in world as shown in the table 1.

Table 1. Description of vehicle status and vehicle surrounding environment.

Name	Range (unit)	Description	Sensor
Angle	[-π, +π](rad)	Angle between the vehicle direction and the direction of the track axis.	GPS
SpeedX	(-∞,+∞)(km/h)	Speed of the vehicle along the longitudinal axis of the vehicle.	GPS
SpeedY	(-∞,+∞)(km/h)	Speed of the vehicle along the transverse axis of the vehicle.	GPS
TrackPos	(-∞,+∞)	Distance between the vehicle and the track axis. The value is normalized to the track width: it is 0 when the vehicle is on the axis, -1 when the vehicle is on the right edge of the track and +1 when it is on the left edge of the vehicle. Value greater than 1 or small than -1 mean that the vehicle is outside of the track.	GPS
Track	[0,4](m)	Each sensor return the distance between the road edge and the vehicle within a range of 4 meters.	Ultrasonic radar
Opponents	[0,100](m)	Vector of 36 opponent sensors: each sensor covers a span of 10 degrees within a range 100 meters and return the distance of the closet opponent in the covered area.	LIDAR

Output Variable

The control to self-driving vehicle in TORCS is achieved by a rather typical set of actuators, i.e., the steering wheel, the gas pedal, the brake pedal, and the gearbox as shown in table 2.

Table 2. Description of vehicle out variable.

Name	Range (unit)	Description
acceleration	[0,1]	Virtual gas pedal (0 means no gas, 1 full gas).
brake	[0,1]	Virtual brake pedal (0 means no brake, 1 full brake).
Gear	-1,0,1,⋯,6	Gear value
Steering	[-1,1]	Steering value: -1 and +1 means respectively full right and left.

Design of Learning Algorithm

Framework of Algorithm

In traditional DDPG algorithm, the network is trained off-policy with samples from a replay buffer. The replay buffer is a finite sized cache R, the tuple (s_t, a_t, r_t, s_{t+1}) is stored in the replay buffer and sampling from the environment according to the exploration policy. However, the learning rate will get slowly and the vehicle behavior can't be effectively improved as the reason of fixed size sample. So the size of sample is increased in the second learning period. The improved algorithm is shown as Algorithm.

Randomly initialize critic network $\mathrm{Q}(\mathrm{s},\mathrm{a}|\theta^Q)$ and actor $\mu(\mathrm{s}|\theta^\mu)$ with weights θ^Q and θ^μ
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode =1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t=1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    If t < T/2 then $N \leftarrow N_0$
    Else $N \leftarrow N_1$ $(N_1 > N_0)$
    End if
    Set $y_i = r_i + \gamma Q'\left(s_{i+1}, \mu'\left(s_{i+1}|\theta^{\mu'}\right)|\theta^{Q'}\right)$
    Update critic by minimizing the loss:

$$L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$$

    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)\,|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

Algorithm 2. improved DDPG algorithm

Design of Rewards

Reward function define the goal of RL, it maps the sensed state to an enhanced signal, and evaluate whether the action is good. The reward signal is usually a scalar which positive value indicates reward and negative value indicates penalty. To apply Q-learning, the reward function is designed as follow:

$$r_t = \begin{cases} -10, & if\ collision\ occur \\ -20, & if\ out\ of\ the\ track \\ V_x\cos(\theta) - V_x\sin(\theta) - V_x|\,\mathrm{trackPos}\,|, & \text{others} \end{cases} \quad (10)$$

When the collision occur, reward value is set to -10, and if vehicle go out of the track, the reward value is set to -20. In other conditions, the reward function is designed to maximize vehicle speed of track longitudinal axial and minimize the transverse axial speed, so that vehicle could complete the competition as soon as possible. Considering the traffic rules constraints, which vehicle can't pass through the solid line which is always arranged at the edge of the road, the third term to reward function is added. Because when vehicle crush the solid line, it always deviates from the center of the road.

Each experiment consists of a number of learning episodes, the learning episode ended when the vehicle go out the border or stuck in the minimum which means the speed of vehicle is less than minimum value.

Exploration Strategy

In RL, appropriate exploration is essential, try some more actions can avoid getting into the local optimum, which vehicle always take same action in specific scene. Ornstein-Uhlenbeck process is used for exploration in the continuous domain, and it is a stochastic process which has mean-reverting properties.

$$dx_t = \theta(\mu - x_t)dt + \sigma dW_t \quad (11)$$

In the formula, the θ represents the speed of the variable reverts towards to the mean. The μ represents the equilibrium or mean value. The σ represents the degree of volatility of the process. This process is added to all the steering, acceleration and brake. Basically, the most important parameters are the μ of the acceleration, because the car should have some initial velocity and don't stuck in a local minimum where the car keep pressing the brake and never hit the gas pedal.

EXPERIMENTS

The TORCS platform is used to implement our obstacle avoidance algorithm. There are diverse tracks in TORCS, including various shaped speedway, narrow country road, alpine track, and track for off-road vehicles, etc. These tracks all include static obstacles and moving obstacles. The static obstacles contain road edge, trees and buildings. The competitive vehicles are obvious moving obstacles, which the self-driving vehicle can't collide with. The purpose of self-driving vehicle is to avoid these obstacles and as soon as possible to complete the competition.

The task are divided into two case based on the obstacles situation: case 1 contain only static obstacles and case 2 contain both static and moving obstacles. The different parameters are set when self-driving learn to avoid obstacles in two cases, so that vehicle could learn a more applicable strategy.

Case 1: Only Static Obstacles

Parameters Setting

First the network are trained without other vehicle in the track, the vehicle avoid static obstacles contain road edge, trees and buildings.

Both the Actor network and Critic network are built in Keras. The actor neural network consist of two hidden layers with 300 and 600 units respectively. The final output layer which represent the activation function is distinguishing due to the variable range. The tanh layer is used to implement steering command as the steering is in the range [-1, 1]. However the acceleration and brake activation function are sigmoid since their range is [0, 1]. The policy network learning rate is 10^{-4}. The Critic network also consists of two hidden layers with 300 and 600 units with the learning rate 10^{-3}. The neural network are trained with 600 episodes and the Ornstein-Uhlenbeck process is declined linearly in 10^4 frames.

The random noise parameters for the three output variables are shown in the table 3.

Table 3. The parameter setting of Ornstein-Uhlenbeck process in case 1.

Action	θ	μ	σ
Steering	0.6	0.0	0.30
Acceleration	1.0	0.6	0.10
Brake	1.0	-0.1	0.05

The μ represents the mean value of noise, so the μ of acceleration is set to 0.6 in order to give vehicle an initial speed, and for the purpose of prevent vehicle halt as frequency brake, the μ of acceleration is set to -0.1. These two parameters have a great impact on the experimental results.

Experiment Results

The track of **CG Speedway number 1** is used as learning track, as it possess the typical character of roadway and contain the static obstacle and the lane line. The shape of the track is shown as Figure 4.

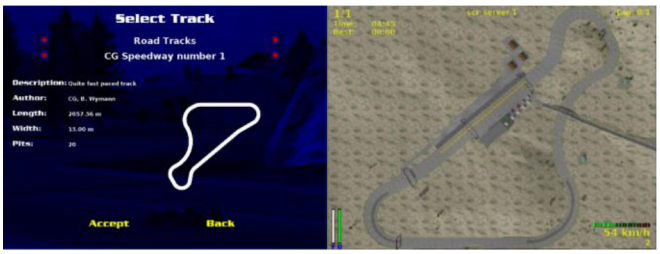

Figure 4. The details of learning track. The CG Speedway number 1 is a quite fast paced track, it is 2057.56 meters long and 15.00 meters wide, and there are 20 pits in the middle.

After approximate 600 episodes learning, the self-driving vehicle could run the entire track without collision. The details of learning process could be found in Figure 5.

The damage of vehicle describe the degree of collision, the higher values indicate more collision. The learning process is monitored by this variable. The damage of every episode is shown as Figure 6.

The average reward of every step in each episode reveal the effect of learning. The purpose of self-driving vehicle is to improve environment reward by continuous learning and get the maximum reward value. So the greater reward indicate the better leaning effect. The details of learning process are shown as Figure 7.

After the vehicle perform well in the learning track of **CG Speedway number 1**, the robustness of algorithm are validated in **Alpine 2** which is longer and more complex than learning track. The details of verification track are shown as Figure 8.

Figure 5. The details of learning process. Scr_server 1 is our self-driving vehicle which controlled by our algorithm. It may pass through the solid line at the beginning of learning, and collide with the road edge. After some learning process, it can complete whole track without collision.

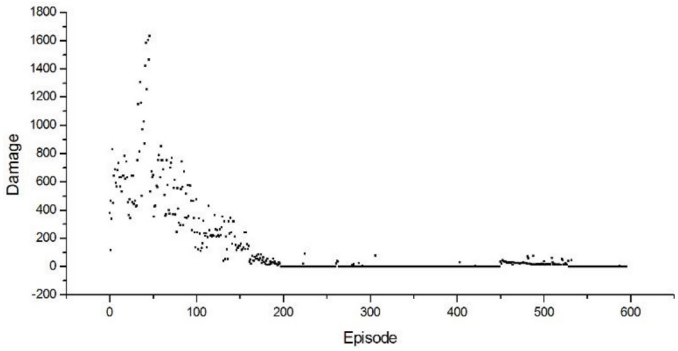

Figure 6. The damage of every episode. At the beginning of experiment, the damage is larger than 600, as the vehicle collide with the edge of road when search the suitable route. After about 100 episode, the damage are getting smaller, and almost close to zero after 200 episode learning. It means that the vehicle could effective avoid static obstacles.

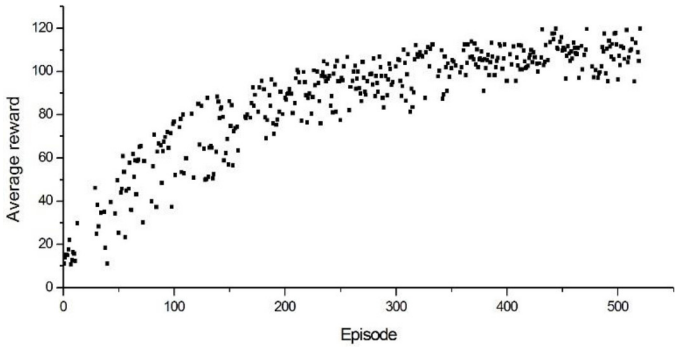

Figure 7. The average reward of every episode. The average reward increase gradually. After 200 episode learning, the average reward is no less than 70 which means vehicle could get positive reward at every step in each episode. And after about 400 episode, the average reward value tend to be stable, which means the vehicle learn a best strategy.

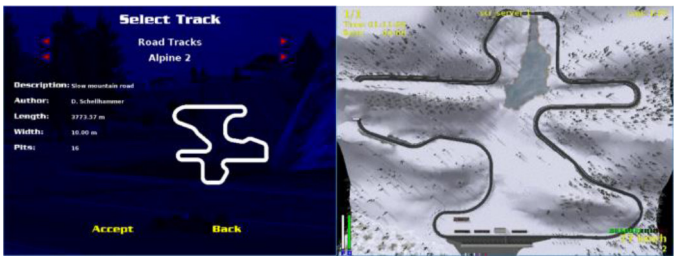

Figure 8. The details of verification track. The Alpine 2 is a slow mountain road, it is 3773.57 meters long and 10.00 meters wide, and there are 16 pits in the middle track.

The self-driving vehicle can successfully complete the entire track without any collide as there is no -10 in reward value of every step. The reward of every step is shown as Figure 9 and the details of driving in verification track are shown as Figure 10.

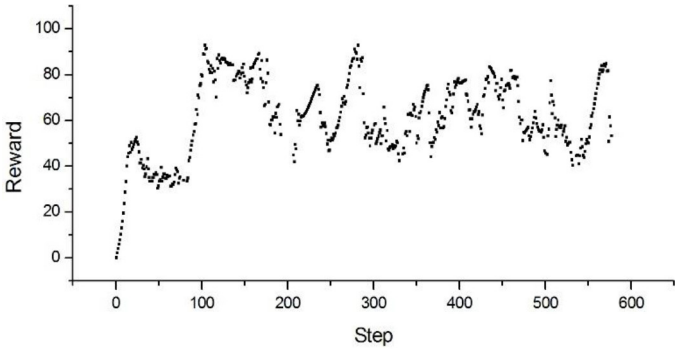

Figure 9. The reward of every step. All reward of every step is positive value which means there is no collision occurred. Self-driving vehicle could complete other track with leaning experience in track of CG Speedway number 1.

Figure 10. The details of driving in verification track.

Case 2: Both Static and Moving Obstacles

Parameters Setting

After train the network in static obstacles environment, the competitive vehicles are added to the environment. The self-driving vehicle should not only avoid the static obstacles, but also should avoid moving obstacles which make it more complicated.

The Actor network and Critic network are built same as case 1. The actor and critic neural network consist of two hidden layers with 300 and 600 units respectively. The random noise parameters for the three output variables are shown in the table 4.

Table 4. The parameter setting of Ornstein-Uhlenbeck process in case 2.

Action	θ	μ	σ
Steering	0.6	0.0	0.30
Acceleration	1.0	0.4	0.10
Brake	1.0	0.1	0.05

The μ of acceleration is changed from 0.6 to 0.4, and the μ of brake is changer from -0.1 to 0.1, because it is found that the vehicle can't decelerate properly at the corner, it need more brake to slow down.

Experimental Results

Track of **Alpine 1** is used as our learning track. The details of the track are shown as Figure 11. The other five vehicle are added as moving obstacle, so the self-driving vehicle must learn how to avoid these vehicles and get the largest reward.

Figure 11. The details of learning track. The Alpine is alpine track, it is 6355.65 meters long and 12.00 meters wide, and there are 16 pits in the middle track.

The other five vehicle are controlled by AI which could make vehicle complete the game. Some vehicles are so extreme that may collide with roadside fences. So the self-driving must learn how to avoid these vehicles and get the effective strategy. After approximate 800 episodes learning, the self-driving vehicle could run the entire track without collision. The details of learning process are shown as Figure 12.

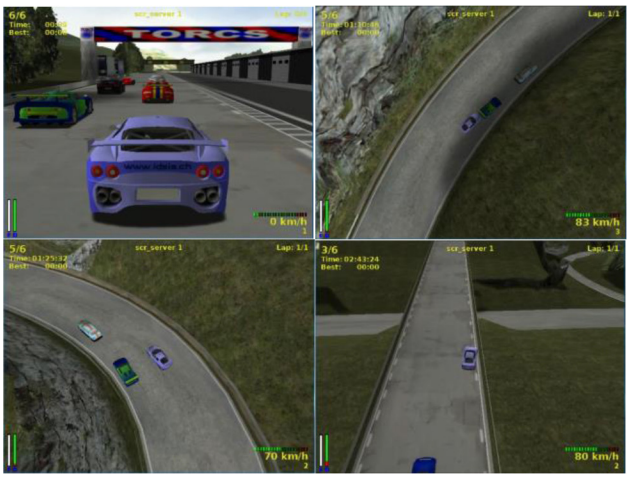

Figure 12. The details of learning process. The vehicle can't avoid other vehicles and collide with them at the beginning of learning. Sometimes it also collide with road edge. It gradually improve its performance by reward and finally learn how to avoid both static and moving obstacles.

The damage of every episode is shown as Figure 13.

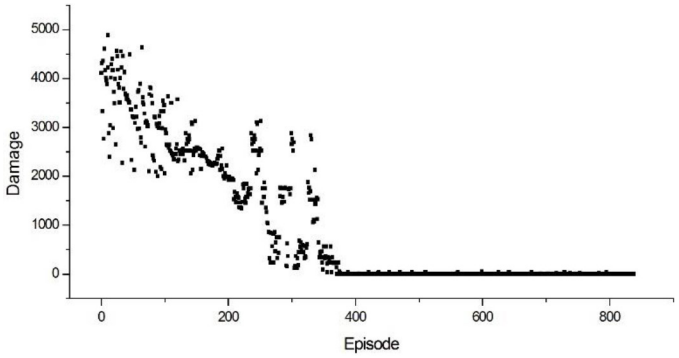

Figure 13. The damage of every episode. At the beginning of experiment, the damage is larger than 2000, as the vehicle collide with the edge of road and other vehicles. After about 380 episode, the damage are getting smaller rapidly and almost close to zero after 400 episode learning. It means that the vehicle could effective avoid static obstacles.

The average reward of every step in each episode of learning process are shown as Figure 14.

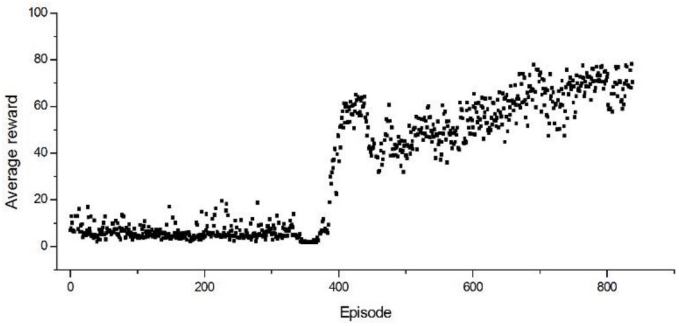

Figure 14. The average reward of every episode. The average reward increase gradually. After 400 episode learning, the average reward increased obviously and no less than 40. And after about 500 episode, the average reward value tend to be stable, which means the vehicle learn a best strategy.

After vehicle perform well in the learning track, the robustness of algorithm are validated in **Alpine 2** which is 3773.57 meters long and 10 meters wide. The self-driving vehicle can successfully complete the entire track without any collide as there is no -10 in reward value of every step. The reward of every step is shown as Figure 15 and the details of driving in verification track are shown as Figure 16.

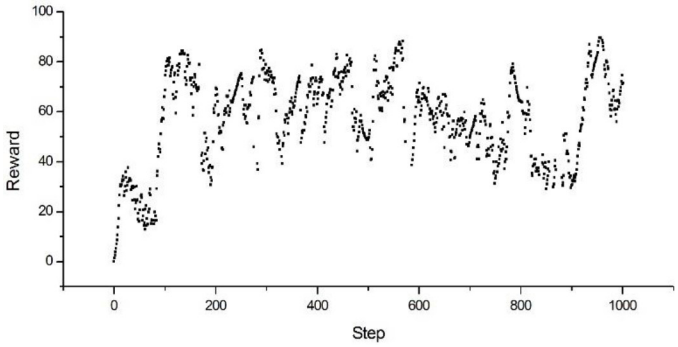

Figure 15. The reward of every step. All reward of every step is positive value which means there is no collision occurred. It turns out that the parameters learned in Alpine 1 is applicable to other track.

Figure 16. The details of driving in verification track.

CONCLUSIONS

In this paper, the Deep Deterministic Policy Gradients, namely DDPG, is applied to develop an advanced obstacle avoidance strategy so that self-driving vehicle could execute continuous actions. The vehicle dynamics constraints and traffic rules constraints are added into learning algorithm, which make vehicle action more effective. The obstacles type are divided into two categories which include static obstacles and moving obstacles. Several experiments are performed to test the proposed approach in different situations. The vehicle is trained in learning track at first, and then the effectiveness and robustness are tested in other different track after enough episode. The result shows that the algorithm could perform well in self-driving vehicle for TORCS environment.

REFERENCES

1. El Sallab, Ahmad, et al. "Deep Reinforcement Learning framework for Autonomous Driving." Autonomous Vehicles and Machines, Electronic Imaging (2017).
2. Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." Nature 518.7540 (2015): 529-533.
3. Sutton, Richard S., and Barto Andrew G.. Reinforcement learning: An introduction. Vol. 1. No. 1. Cambridge: MIT press, 1998.
4. Xiong, X., Wang, J., Zhang, F., & Li, K. "Combining Deep Reinforcement Learning and Safety Based Control for Autonomous Driving." arXiv preprint arXiv:1612.00147, 2016.
5. Lillicrap, Timothy P., et al. "Continuous control with deep reinforcement learning." arXiv preprint arXiv:1509.02971 (2015).
6. Liang-zheng, X. U., Cheng-yong U. I. A. O., and Jian-wu ZHANG. "Analysis of Dynamics Stability of the Tractor/Full Trailer Combination Vehicle and Simulation for Its Control System." Computer Simulation 12 (2003): 034.
7. Loiacono, Daniele, et al. "Learning to overtake in torcs using simple reinforcement learning." Evolutionary Computation (CEC), 2010 IEEE Congress on. IEEE, 2010.
8. Loiacono, Daniele, Luigi Cardamone, and Lanzi Pier Luca. "Simulated car racing championship: Competition software manual." arXiv preprint arXiv:1304.1672 (2013).

CONTACT INFORMATION

Name	Email address	Tel
Xiaopeng Zong	zongxp@qq.com	15011299975
Guoyan Xu	xuguoyan@buaa.edu.cn	15810771808
Guizhen Yu	yugz@buaa.edu.cn	13651164015
Hongjie Su	920457660@qq.com	15652915436
Chaowei Hu	chaowei0820@buaa.edu.cn	15910987739

ACKNOWLEDGMENTS

This work is partially supported by the Beijing Municipal Science and Technology Project under Grant #D161100004116001 and by the Natural Science Foundation of Beijing (No. 71571107).The authors would also like to thank the insightful and constructive comments from anonymous reviewers.

DEFINITIONS/ABBREVIATIONS

RL - Reinforcement leaning

DL - Deep learning

DQN - Deep Q network

DDPG - Deep Deterministic Policy Gradients

TORCS - The Open Racing Car Simulator