Simultaneous Path planning and Path Tracking for Autonomous Ground Vehicles using Deep Q Network

Yu Zheng, Hansong Zhou

1 Background

Autonomous driving is expected to revolutionize road traffic attenuating current externalities, especially accidents and congestion [1]. A fully autonomous driving platform contains sensing, perception, path planning, path tracking and motion control. Path planning and path tracking are called the decision making system of a smart car where the former generates the optimal path based on the environment information and vehicle's states, while the latter decides the vehicle's actions, including steering angle and motor commend, to follow the given path. The traditional A star path planner and model-free based path tracking controller, such as PID controller, could not robustly control the car to handle unseen events happens on the road in real-time, such obstacles. Model predictive control is a type of model-based controllers which could handle complexity road environment but is computationally expensive due to online optimization solving. Thus, how to design an intelligent and computation-friendly decision making system is crucial for driving a car in complex road environment.

In path planning and path tracking, decision is made based on the observation only from current environment and the online policy in each timestamp, and such process is called Markov decision process (MDP). This process is represented by four tuples (S, A, P, R), where S and A denote the state space and action space respectively. P is the state transition probability function whereas R is the reward function [2]. Deep reinforcement learning (DRL) provides a promising solution to fully adapt to the dynamic environment via trial and error and is widely used to deal with the policy design problem in MDP. As mentioned before, tradition model-based controller suffers from high computation demand for learning unknown environment. To deal with this challenge, a variant of DRL called Deep Q-Learning Network (DQN) is adopted in the path planning and path tracking problem. DQN is efficient in MDP problem because it is a model-free based network and learns the policy by interacting with the environment online. The following part introduces the specifics of our project.

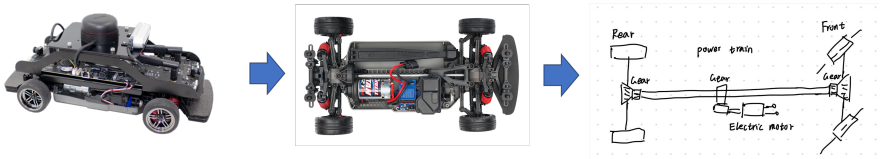

Figure 1: Vehicle

2 Problem statement

Although there exists a lot of reinforcement learning-based path planning in grid-world, autonomous driving in a real scenario is much different. The most difference is: after RL agent decides a target waypoint, the vehicle could not immediately arrive at the target place. The vehicle has its own dynamics and kinematics, and the motion controller also needs time to track the waypoints. This makes those reinforcement learning methods for gridworld path planning be not sufficient for real autonomous driving.

In this report, we will present a successful deep Q network case for **lane change policy learning in order to overtaking other moving cars tracking target waypoints**.

2.1 Vehicle model

A 1/10 vehicle is used in this project, shown in figure 1. Since we use reinforcement learning only to learn the the lateral control, the dynamical model is omitted. Instead, a discrete forward kinematic model is used for training:

$$\begin{aligned} x_{k+1} &= x_k + v_K cos(\beta_k + \theta_k)dt \\ y_{k+1} &= y_k + v_k sin(\beta_k + \theta_k)dt \\ \theta_{k+1} &= \theta_k + v_k tan(\delta_k)cos(\beta_k)dt/L \end{aligned} \tag{1}$$

where $tan(\beta_k) = \frac{1}{2}tan(\delta_k)$, x_k, y_k, θ_k represent the vehicle's x, y coordinators and yaw angle respectively, dt is the controller sampling time, $v_k = 2$ is the vehicle's longitudinal velocity that is controlled to be constant value by cruise control, δ_k denotes the steering angle which is controlled by Stanley lateral controller. To simplify, we write the model in the following form:

$$\mathbf{z}_{k+1} = \mathbf{z}_k + f(\mathbf{z}_k, \mathbf{u}_k)dt \tag{2}$$

where $\mathbf{z}_k = \begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix}$, $\mathbf{u}_k = \begin{bmatrix} v_k \\ \delta_k \end{bmatrix}$.

2.2 Environment setup

The experiment performs on a two-lane highway, shown in 2. The goal is to train DQN to learn how to correctly change lane in order to overtake the ahead

car and change back to the original lane. The lane width is $W_L = 1/\sqrt{2}$, and this local map range is $x \in [0, 7], y \in [-1, 8]$. The width and length of cars are same $W_c = L_c = 0.5$.

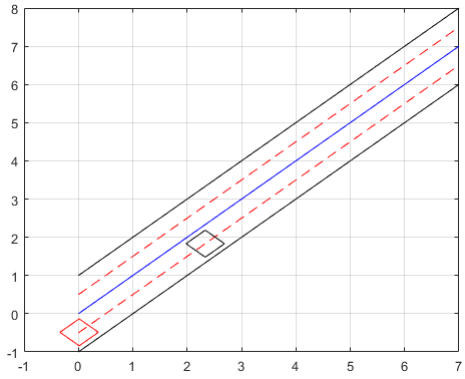

Figure 2: Two-lane highway environment. (**red dotted lines** are the middle line of two lanes respectively, **blue line** is the dividing line between two lanes, and the **black lines** are the boundaries of highway. **red square** is the host car, **black square** is a surrounding car.)

We call the upper lane as lane 2, while the below lane as lane 1. The lanes are described as

$$\text{Lane } 1 : g_1(x) = 0 * x^2 + 1 * x - 0.5$$
$$\text{Lane } 2 : g_1(x) = 0 * x^2 + 1 * x + 0.5$$

The yaw angles of two lanes are same as $\pi/4$. The surrounding cars are assumed to run with a slower speed as $v_j = 1$. The upper and bottom lane boundary can be described by $g_1(x) - 0.5$ and $g_2(x) + 0.5$. Thus, if the vehicle is inside the lane, the following condition holds:

$$g_1(x) - 0.5 \leq y \leq g_2(x) + 0.5$$

where x, y are the x, y coordinators of the vehicle respectively.

2.3 Problem formulation

The goals of the car include:

1. track a target waypoint at the end of lane 1 $(14, 13.5)$;
2. change lane to pass surrounding cars;

The task of DQN is to decide which lane the vehicle should follow, such that

$$\lambda \in \{1(\text{bottom lane}),\ \ 2(\text{upper lane})\}.$$

Once the target lane is given, the lateral distance from the lane e_1 and the difference of yaw angle e_2 should be controlled to be zero. The corsstrack error e_1 is approximated as the shortest lateral distance from the car to the middle of the lane, solved as following. Assume the lane is described as a second-order polynomial function $g(x) = ax^2 + bx + c$, e_1 is given by solving the below optimization program:

$$\text{Minimize}_x \ \frac{1}{2}[(x - x_0)^2 + (g(x) - y_0)^2], \tag{3}$$

where (x_0, y_0) is the location of the vehicle, then $e_1 = \sqrt{(x_\star - x_0)^2 + (g(x_\star) - y_0)^2}$. A closed-form solution is given by

1. $x_\star = root\left([2a^2 \ \ 3ab \ \ 1 + 2a(c - y_0) + b^2 \ \ b(c - y_0) - x_0]\right)$,
2. $e_1 = min_j \sqrt{(x_{\star j} - x_0)^2 + (g(x_{\star j}) - y_0)^2}$.

The yaw difference is easily calculated as

$$e_2 = \theta - atan(2 * ax_\star + b).$$

Then Stanley lateral controller is used to track the target lane:

$$\delta_k = -e_2 + atan\left(k * \frac{e_1}{k_s + v}\right) \tag{4}$$

where $k = 0.1$ is a control gain, $k_s = 1e{-}4$ is a softening parameter. To simplify, the controller is described as

$$\mathbf{u}_k = h(\lambda, \mathbf{z})$$

which means once DQN choose the lane index λ, the controller will track the target lane based on vehicle's current location $\mathbf{z}$.

To learn the lane change policy, the following optimization program should be solved

$$\begin{aligned} \min_{\lambda_k} \quad & a_1 D(\mathbf{z}_k, \mathbf{z}^\star) + a_2 \sum_{j \in V} P(d_{j,k}, V_{j,k}) \\ \text{s.t.} \quad & Z_{k+1} = Z_k + f(Z_k, h(\lambda, \mathbf{z}))dt \\ & \lambda \in \{1, 2\} \\ & g_1(x) - 0.5 \le y \le g_2(x) + 0.5 \\ & d_{j,k} > \max(W_c, L_c) \ \ \forall j \in V \end{aligned} \tag{5}$$

1. $a_1 D(\mathbf{z}_k, \mathbf{z}^\star)$ is a target pursuit penalty, so we should minimize it to achieve the target. Using the distance from host car to target as this penalty term is not a good choice since the distances are not much different when the host car is on different lane.
2. $a_2 \sum_{j \in V} P(d_{j,k}, V_{j,k})$ is the collision probability term based on the distance $d_{j,k}$ from the host car to the surrounding cars indexed in V.

3. $Z_{k+1} = Z_k + f(Z_k, h(\lambda, \mathbf{z}))dt$ is the vehicle kinematics.

4. $\lambda \in \{1, 2\}$ is the action set;

5. $g_1(x) - 0.5 \leq y \leq g_2(x) + 0.5$ is the lane bound

6. Although we minimize the collision probability in the objective function, the collision could not be prevented completely. Thus, the last constraint $d_{j,k} > \max(W_c, L_c) \ \ \forall j \in V$ is used to avoid crashing.

3 DQN-based model

The problem of path planning and path tracking can be formulated as Markov decision process (MDP) because the vehicle makes decision only based on current environment. The MDP is represented by four tuples (S, A, P, R), where S and A denote the state space and action space respectively. P is the state transition probability function whereas R is the reward function. DRL provides a promising solution to fully adapt to the dynamic environment via trial and error and is widely used to deal with the policy design problem in MDP. Compared to the tradition model-based controller which suffers from high computation demand for learning unknown environment, Deep Q-Learning Network (DQN) is more efficient in MDP problem because it learns policy be directly interacting with the environment online without modeling. The definition of each tuple in MDP is given in the following.

3.1 MDP specification

3.1.1 State Space

We formulate a 15mx15m map for the vehicle environment. There are three vehicles in the map including a target car and two surrounding cars. After each step, the target car will observed environment on three aspects.

1. Self current information: Location, direction $\{x_k, y_k, \theta_k\}$

2. Target waypoint information: Location and direction $\{x_{target}, y_{target}\}$

3. surrounding car info: Location and direction $\{x_k^s, y_k^s, \theta_k^s\}$, velocity v_k^s where i is the number of surrounding car.

In total, there are 9 features of the current environment. The set of these features at step k is defines as the environment state:
$\mathbf{S}(k) = \{x_k, y_k, \theta_k, x_{target}, y_{target}, x_k^s, y_k^s, \theta_k^s, v_k^s\}$

3.1.2 Action Space

When the vehicle is getting closer to the surrounding car on its current lane, it needs to switch to another lane to pass by the obstacled vehicle. In our experiment, the action is defines as $\alpha(k) = \{1, 2\}$, where 1 represents starting switching to lane 1 and 2 to lane 2. It is worth noting that the switching behavior is controlled by the motor of the vehicle. If the vehicle cannot make right decision early enough, it still will collide with the surrounding car.

3.1.3 Reward Function

The immediate reward R_k is composed of 5 parts:

$$R_k = R_{target} + R_{collision} + R_{lane} + R_{map} + R_{sucess} \tag{6}$$

where each component is defined as follow:

$$R_{target} = \begin{cases} 5 & \text{host car is on the target lane} \\ -5 & \text{host car is not on the target lane} \end{cases} \tag{7}$$

$$R_{collision} = \begin{cases} 0 & \text{host car is ahead surrounding car or not in same lane} \\ f(d_{h,s}) & d_{h,s} <= 10 \\ -15 & d_{h,s} <= 10 * \max(W, H) - -risky \\ -50 & d_{h,s} <= \max(W, H) - -crash \end{cases} \tag{8}$$

where $f(d_{h,s}) = -10e^{-|d_{h,s} - \max(W,H)|^2}$, the $d_{h,s}$ is the distance between the host car and the surrounding car, and the $\max(W, H)$ is size of each vehicle.

$$R_{lane} = \begin{cases} 0 & \text{host car is in any line} \\ -20 & \text{host car is out side the lane} \end{cases} \tag{9}$$

$$R_{map} = \begin{cases} 0 & \text{host car is in the map} \\ -20 & \text{outside the map but not through the target waypoint} \end{cases} \tag{10}$$

$$R_{success} = \begin{cases} 300 & \text{host car arrives at the target} \\ 0 & \text{host car has not arrives} \end{cases} \tag{11}$$

This reward function reflects the distance to the targets, the rationality of lane selection, the probability of collision. By maximizing the value of reward function, we could have a more stable vehicle control and purse a safer path planning.

3.2 DQN algorithm

In order to reduce the frequency of parameter updating as well as increase the stability and the probability of convergence, the double DQN network is deployed. It includes two networks with the same structure: the target DQN network with parameter θ' and the training DQN with parameter θ. At the beginning of training, the parameters of networks are initialized as $\theta' = \theta$. The outputs of target network and the training network are denoted as $Q^*_\pi(\mathbf{S}(k)', \alpha(k)'; \theta')$ and $Q_\pi(\mathbf{S}(k), \alpha(k); \theta)$ separately. In the training stage, a mini-batch of experience is used to train the training DQN and update the parameter θ based on the following loss function:

$$L = (R_k + \gamma \max Q^*_\pi(\mathbf{S}'_k, \alpha'_k; \theta') - Q_\pi(\mathbf{S_k}, \alpha_k; \theta))^2 \tag{12}$$

where R_t is the reward of taking action α_t. After D steps of training, the parameter θ' is updated to be equal to θ again. The training algorithm is given in the following Algorithm.1.

Algorithm 1 DQN-based path planning algorithm

Initialize the training network and target network with same random parameter $\theta = \theta'$.
Set a empty replay buffer with size G for the DQN.
for $m = 1, 2, \ldots, M$ **do** episodes
    Initialize the state S of the system, including the location of all vehicles.
    for $n = 1, 2, \ldots, N$ **do** steps
        Vehicle observe the state $\mathbf{S}(l)$ and input it to its training network;
        Calculates the corresponding reward $R(l)$;
        Take action $\alpha(l)$ based on the $\varepsilon - greedy$ policy.
        Observe the next state $\mathbf{S}(l+1)$ after taking the action.
        Update the experience replay buffer with the experience $\varsigma(t) = \{\mathbf{S}(L), \alpha(L), R(L), \mathbf{S}(L+1)\}$;
        if R **then**enew B times of buffer:
            Randomly select a mini-batch Ω of from the replay buffer;
            Train the training network with Ω with the loss function (12) and update the parameter θ of training network;
            Count of training steps increases by 1.
        end if
        if Training steps = D **then**
            Update θ' of target network equals to θ
            Training steps resets to 0.
        end if
    end for
end for

We use 3 fully-connected layers as the DQN model for training and with 32, 32, 2 neurons in each layer. The activation function of the first two FC layers is ReLU. The output layer after the third FC layer is a regression layer.

4 Simulation

4.1 Experiment scenario

We run a simple experiment scenario in this simulation: the host car start from $(0, -0.5)$ (lane 1) and run with a constant speed $v = 2m/s$ while one surrounding car start from $(2.3263, 1.8263)$ (lane 1) and run with a lower speed $v = 1m/s$. In the real highway environment, once the host car passes the surrounding car, then the surrounding car cannot catch the host car anymore. In other word, there is no risk of collision after the host car passes surrounding car in the real highway scenario. However, since we only simulate a slice of highway, we should assume the surrounding car stops after running certain time steps in order to avoid the risk of collision after the host car passes the surrounding car. Thus, we assume the surrounding car stops at $(3.4965, 2.9965)$ (lane 1). The target way point for the host car is $(7, 6.5)$ (lane 1).

4.2 Network setting

The parameter of the DQN network is given as follows:

Parameters	**Value**
Optimizer	ADAM
Learning rate	0.001
Batche size G	16
Training interval	16 steps
Target renew interval D	200 steps
Buffer size	4096
Initial ϵ	0.5
ϵ decay rate	0.99
Final ϵ	0.1
Reward discount γ	0.9
Soft update τ	0.01
Loss evaluation	RMSE

Table 1: Parameter of DQN-based algorithm

5 Software

The files are introduced below:

1. Requirement: Matlab
2. `test_lateral_controller.m`
   : main files where the DQN is trained. Run this file.

3. `environment_param.m`

   : load all environment parameters

4. `DQN_param.m`

   : load all DQN parameters

5. `dqn_model.m`

   : create DQN model

6. `kinematic_model.m`

   : kinematic model of vehicle used for state propagation

7. `Trajectory_generator.m`

   : create 2-lane highway environment

8. `draw_car.m`

   : plot car on the map based on the car's coordinator

After running

`test_lateral_controller.m`

, the training process will perform 500 episodes, and if the host car success 10 times, then the training process will end and conclude that DQN is successfully trained.

5.1 Result

As shown in the figure 3, DQN first drive the car in the lane 1. After detecting the surrounding car, it drives the car to change the lane. Then it follows the lane 2 to pass the surrounding car. After passing, it changes back to lane 1 and arrives the target waypoint. And the figure 4 shows the whole positions of the host car and the surrounding car in the experiment without time information. These results show DQN has learned the lane change policy to avoid collision and pursue the target waypoint.

References

[1] Martínez-Díaz, Margarita, and Francesc Soriguera. "Autonomous vehicles: theoretical and practical challenges." Transportation Research Procedia 33 (2018): 275-282.

[2] Y. Lin, M. Wang, X. Zhou, G. Ding, and S. Mao, "Dynamic spectrum interaction of uav flight formation communication with priority: A deep reinforcement learning approach," IEEE Transactions on Cognitive Communications and Networking, vol. 6, no. 3, pp. 892–903, 2020.

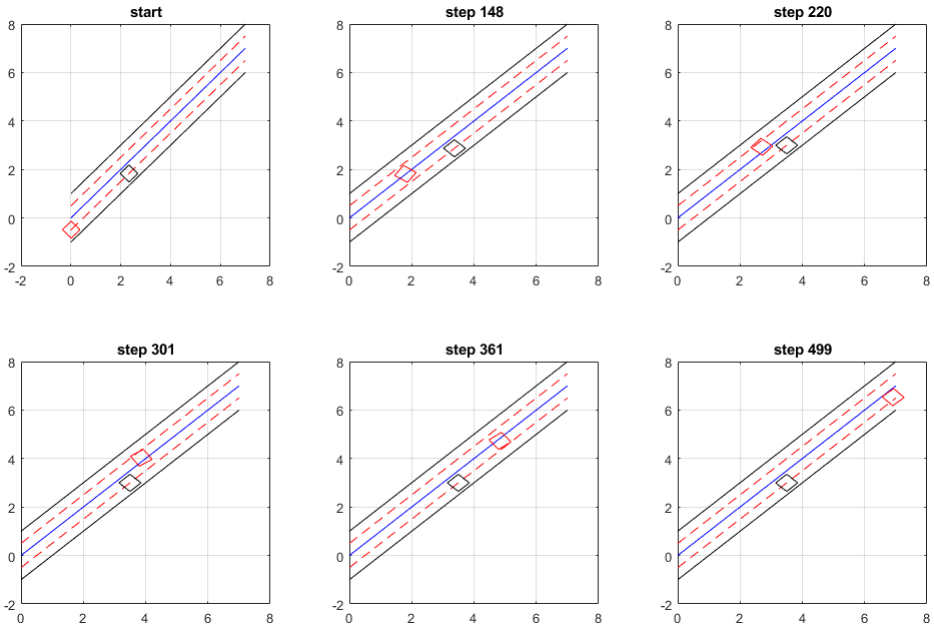

Figure 3: The intermediate spots of the host car and surrounding car in a successful experiment

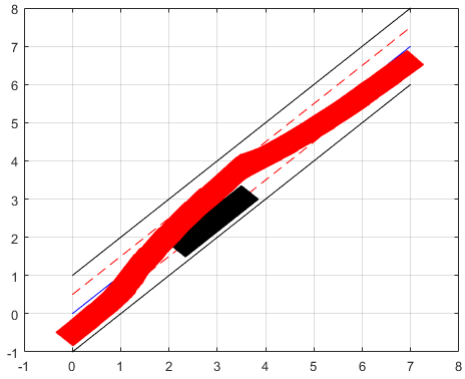

Figure 4: The whole plot of an example successful experiment controlled by DQN