A Physical State based DQN Agent for Autonomous Vehicles

Soumyabrata Talukder, Ramij Raja Hossain

***Abstract*—In lieu of the current trend of computer vision based decision inference, agents of autonomous vehicles may alternatively be designed for physical state-based control synthesis. In this work, we train a Deep Q Network (DQN) to estimate Q values corresponding to the physical state and action of the autonomous vehicle, which facilitates selection of optimal control decision at every physical state. We assume constrained continuous state space with a discrete set of actions, and implement our idea on the bicycle model of a commercialized miniaturized autonomous vehicle. Results are reported for multiple trials with different agent architectures and training parameters.**

***Index Terms*—DQN, Reinforcement Learning, Autonomous Vehicle, Landshark, Continuous state space.**

I. Introduction

AUTOMOBILE is an ubiquitous necessity of human life, and bringing *autonomy* in automobile was conceptualized primarily to enhance transportation efficiency and safety, and also to improve human experience. Since its inception, autonomous vehicle had been an active area of research, proliferation and success of which are evident through its recent commercialization, and the optimistic targets of launching autonomous vehicles set by a number of industry leaders.

While the state-of-the-art of autonomous vehicles, invariably embrace computer vision based decision making [1]–[4], it is intriguing to explore how good the physical states of the vehicle are, to make control decisions. The camera inputs are quintessential for a general on-road application, in order to ensure safety and compliance of traffic rules, but alternatively the aforementioned physical state based control could act as a reference *path plan* that is further duly corrected by the real-time image-based inference. On the other hand, there are miniaturized autonomous vehicles (e.g. *Landshark* launched by Black-I Robotics) that are designed to perform transportation of objects between designated points in a fixed environment (akin to robotic application), where physical state based control decision making can eliminate the current need of installing a set of sensory cameras, thus reducing the manufacturing cost.

Various classical approaches (e.g. linear/nonlinear model predictive control (MPC), feedback linearization, control Lyapunov design) exist in literature that deals with the problem of optimal path planning for autonomous vehicles. [5] provides a brief survey of these approaches. The classical techniques offer provable properties of the controller, but generally at the cost of numerous assumptions that are hard to verify in real time practical applications. Reinforcement Learning (RL), in contrast, arguably provides the most general framework that offers a near optimal solution. RL also benefits from the fact that it doesn't rely on an accurate model for control synthesis, rather it learns from the experiences (also termed as *reward*) with the environment.

Motivated by this, we try to solve the problem of optimal control synthesis for the autonomous vehicle by adopting *Deep Q Learning*, the RL approach proposed in [6], which is amenable to perform well for the problem categories with continuous state space, and finite discrete action space.

The rest of the paper is organized as follows. Section II describes the dynamic model of the autonomous vehicle, and defines the environment and its constraints. Section III defines few notations that will be used throughout the rest of the paper. Section IV describes the architecture of the agent, and the implementation setting. Section VI describes the results, and finally section VII concludes the paper.

II. Model Description

A. Autonomous Vehicle Dynamics

We consider the bicycle model [7] of an autonomous vehicle with 5 degrees of freedom (a.k.a. states), where v, ω, and θ denote the three states: forward velocity, yaw rate and angular position respectively, and the pair of the rest two states x and y denotes the position of the vehicle on cartesian coordinate. The model parameters are: m, J, B, B_r, B_s, B_l, and S_l denoting the point mass, moment of inertia, width (wheel to wheel), mechanical resistance to rolling forward, mechanical resistance to sliding forward, mechanical resistance to turning and lateral friction of the track respectively. The input to the vehicle are F_l and F_r denoting the forward effective force acting on the left and right wheel assembly respectively. Fig. 1 shows a schematic diagram of the vehicle model for the sake of clarity.

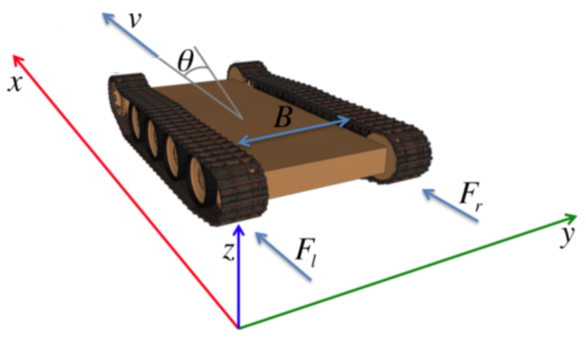

Fig. 1: Schematic diagram of the vehicle model

This paper is submitted in May'19, as a requirement of the final term project of the course ME 592x offered at Iowa State University

S. Talukder and R. R. Hossain are affiliated with the Department of Electrical and Computer Engineering, Iowa State University, Ames, IA 50010, USA (e-mail: talukder@iastate.edu, rhossain@iastate.edu).

In this setting, the dynamics of the vehicle can be represented by the following switched nonlinear dynamic equations:

$$turning := \begin{cases} True, & \frac{B}{2}|F_l - F_r| \geq S_l \\ False, & Otherwise \end{cases} \tag{1}$$

$$\dot{v} = \begin{cases} \frac{1}{m}\{F_l + F_r - (B_r + B_s)v\}, & turning = True \\ \frac{1}{m}(F_l + F_r - B_r v), & turning = False \end{cases} \tag{2}$$

$$\dot{\omega} = \begin{cases} \frac{1}{J}\{\frac{B}{2}(F_l - F_r) - B_l\omega\}, & turning = True \\ 0, & turning = False \end{cases} \tag{3}$$

$$\dot{\theta} = \omega \tag{4}$$

$$\dot{x} = v \sin\theta \tag{5}$$

$$\dot{y} = v \cos\theta \tag{6}$$

We assume that the states of the vehicle are fully observed, and are available to the agent for the purpose of control synthesis.

B. Objective

The goal of the agent is to manipulate the input of the vehicle (F_l and F_r) such that it starts from a point and reaches a designated destination in an environment subjected to obstacles and constraints. It is also desired that the agent drives the vehicle optimally in the sense that the time required to reach the destination, and the extent of hitting obstacles or violating constraints, are both minimized. A layout of the environment, in which the vehicle needs to be maneuvered is shown in Fig. 2. The *start* and *stop* points are indicated by the red dots. The agent needs to ensure that the vehicle doesn't collide with any of the obstacles (Obstacle 1, 2 and 3 in Fig. 2), and that the vehicle motion is contained within the cartesian space $x, y \in [0, 100]$.

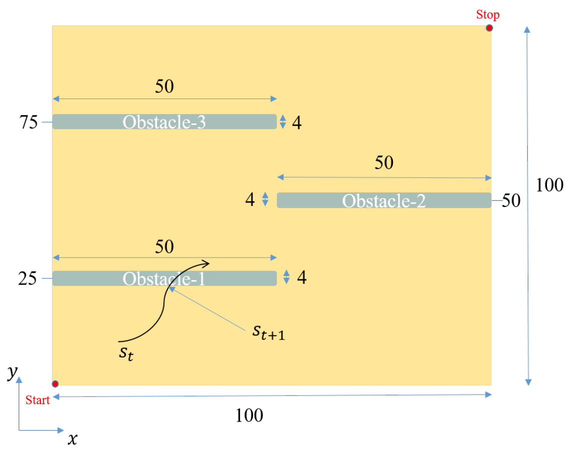

Fig. 2: Schematic diagram of the vehicle model

III. Notations

Let $\mathbb{R}$ denote the space of real numbers, $\mathbb{N}$ denote the set of natural numbers, and $\mathbb{R}^n$ denote the space of real vectors of dimension n. $\tilde{x}_t \in \mathbb{R}^5$ denotes the state vector of the vehicle at the discrete time instant $t \in \mathbb{N}$. $s_t \in \mathbb{R}^2$ denotes the cartesian coordinate of the vehicle at the discrete time instant t. $n_t \in \mathbb{N}$ denotes the zone in which the vehicle belongs (illustrated in Section V-B) at the discrete time instant t.

IV. Agent Model

In order to achieve the goal, the Reinforcement Learning(RL) agent has to take actions depending on the state observations and reward from the environment as shown in Fig. 3. Now, to implement such a RL agent, we have chosen a fully connected Deep Neural Network having 1 input layer, 1 output layer and 2 Hidden layers as shown in Fig. 4, and for the Network 'Adam' is chosen as the 'Optimizer'.

For such a DNN model, there are total 3 categories of parameters, 1. Network parameters, 2. Optimizer Parameters [6], [7], and 3. Training parameters [7]. Each of these parameters has several hyper-parameters. Tuning of these hyper-parameters have significant impact in the performance of the network. The initial values of these hyper-parameters are chosen based on experience and later, those are tuned in a experimental way to achieve better performance. The parameters are detailed in TABLE I, TABLE II & TABLE III. The different tried values of a particular hyper-parameter are separated by '/'.

TABLE I: Network Parameters

Layers	No. of Neurons	Activation
Input Layer	5	-
Hidden Layer 1	64/128	ReLu
Hidden Layer 2	64/128	ReLu/Tanh
Output Layer	9/8	-

TABLE II: Optimizer Parameters

Hyper Parameters	Values
Learning rate (α)	$10^{-3}/10^{-4}/10^{-5}$
Gradient momentum (β_1)	0.9/0.95
RMS momentum (β_2)	0.999/0.95
Gradient Clip-norm	-/10/1
Loss	MSE/ Huber($\delta = 1$)
Fuzz factor(ϵ)	10^{-8}

TABLE III: Training Parameters

Hyper Parameters	Values
Batchsize (B)	32/100/200
Replay Memory size (N)	$10^5/5 \times 10^6/10^6$
Discount factor (γ)	0.99/0.9999
Exploration range([ϵ_{min},ϵ_{max}]	[0.1,1]/[0.01,1]
Exploration decay rate (ϵ_{decay})	0.99/0.995
Training (or exploration decay) start	$4.5{\times}10^4/5 \times 10^4$
Episode Length (No. of Timesteps, T)	100/250/500/1000
Model update frequency (No. of episodes, f_m)	15/20/100

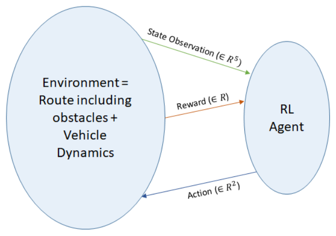

Fig. 3: Interaction between Environment & Agent

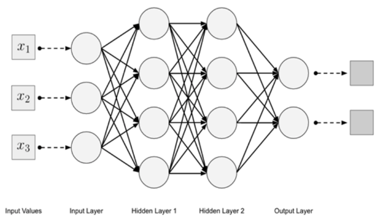

Fig. 4: Network Model

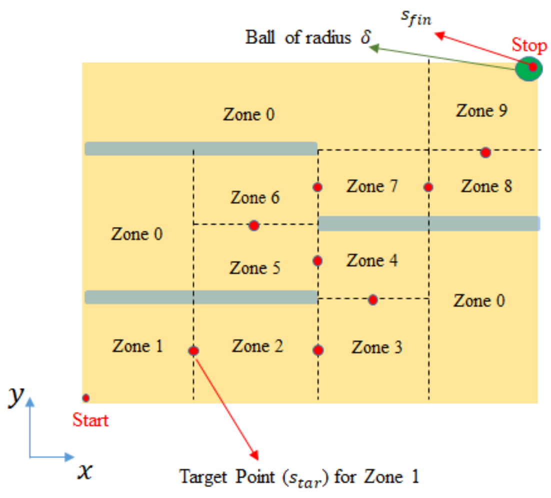

Fig. 5: Zone-wise Segregation of the Environment

V. Architecture and Implementation

In this section, we discuss about various aspects of the implementation framework such as, Action Space, Rewarding Scheme Exploration Scheme and finally, the implementation architecture in details.

A. Action Space

Here, we are considering a discrete action space. As mentioned earlier, there are two control input, F_l & F_r, whose values are limited within lower & upper bounds. Now, we define action space as, $a = \{a_1, a_2\}$, each of a_1 & a_2 can take values either {-1,0 & 1}. Thus updated control inputs are given by (9) & (8).

$$F_l^{\,t+1} = \left[F_l^{\,t} + a_1\right]_{F_l^{min}}^{F_l^{max}}; \tag{7}$$

$$F_r^{\,t+1} = \left[F_r^{\,t} + a_1\right]_{F_r^{min}}^{F_r^{max}}; \tag{8}$$

where, $F_l^{\,t+1}$ & $F_r^{\,t+1}$ denotes the control inputs for next time instant, $F_l^{\,t}$ & $F_r^{\,t}$ denotes the control inputs for current time instant and F_l^{max}, F_l^{min}, F_r^{max} & F_r^{min} denotes respective maximum and minimum value of F_l & F_r.

The structure of action space clearly implies that it is discrete and total possible actions are {(-1,-1); (-1,0); (-1,1); (0,-1); (0,0); (0,1); (1,-1); (1,0); (1,1)}. Among them, we have excluded the action (0,0) to avoid getting stuck at a particular point. So, the cardinality of the action space is 8.

B. Rewarding Scheme

We have segregated the entire layout shown in Fig. 2 into 10 different zones as detailed in Fig. 5. The numbering of the zones are done in such way that a zone with higher number is relatively favorable position to reach the ultimate goal. For each zone, there are separate target points marked as red dots on zone transition boundaries in Fig. 5.

Considering a transition s_t to s_{t+1} in Cartesian coordinate and corresponding zone transition n_t to n_{t+1} following an action a the reward at t^{th} time-step is defined as follow,

$$R_t = (r_z + r_f + m \times n_t \times r_{nz} + r_t + r_p) \times K; \tag{9}$$

where, $K = 0.01$ is a normalizing factor, and R_t has total 5 components (namely r_z, r_f, r_{nz}, r_t and r_p) that are described below,

1) *Reward for zone transition*

$$r_z = \begin{cases} 20n_t, & n_{t+1} > n_t \\ -25n_t, & n_{t+1} < n_t \\ 0, & Otherwise \end{cases} \tag{10}$$

2) *Reward for reaching goal*

$$r_f = \begin{cases} 500, & \mid s_{fin} - s_{t+1} \mid \leq \delta \\ 0, & Otherwise \end{cases} \tag{11}$$

Here, $\mid s_{fin} - s_{t+1} \mid \leq \delta$ defines a small ball (shown as a 'green ball' in Fig.) around the 'Stop' point.

3) *Reward for State Change*

This reward component comes into effect only if transition occurs within a particular zone.

$$r_{nz} = \begin{cases} \frac{k - \|s_{tar} - s_{t+1}\|}{k} \in [0, 1], & n_{t+1} = n_t \\ 0, & Otherwise \end{cases} \tag{12}$$

In connection to this reward component, $m \in \{-1, 1\}$ signifies whether a particular action moves the vehicle towards the zonal target point or away from it.

$$m = sign\{r_{nz}(t) - r_{nz}(t-1)\}, \tag{13}$$

4) *Cost of time spent*

According to this scheme, at each time step a fixed negative reward will be added. Thus, as a consequence of taking too much time to reach the goal, the agent ends up accumulating a large negative reward, thus the agent will avoid aimlessly wandering over the cartesian-space.

$$r_t = -0.5, \tag{14}$$

5) Cost of hitting prohibited zone

It is obvious that the ultimate goal is to reach the end point avoiding any kind of obstacles. To incorporate this in our rewarding scheme, we have specified 4 zones as 'Prohibited Zone', and hitting any of these prohibited zones results in obtaining a large penalty. The Prohibited zones are specified below,

- Obstacle 1
- Obstacle 2
- Obstacle 3
- Outside Grid

The reward component is specified as follows:

$$r_p = \begin{cases} -30, & \text{if prohibited zone is hit} \\ 0, & Otherwise \end{cases} \tag{15}$$

C. *Exploration Scheme*

Initially, an uniform exploration scheme is considered over all the zones. But we observed that such exploration doesn't allow the agent to adequately explore various zones in the cartesian space. Hence, we have adopted a zone-wise exploration scheme, where the initial exploration rate of each zone is set to 1. Now, if in a particular time-step, the vehicle visits Zone n_t, then the exploration rates for all the zones $n \leq n_t$ decays in a pre-defined constant rate. Using zone-wise exploration scheme, it is possible to explore more zones in the environment, hence it in turn helps to populate the replay memory more effectively, i.e increasing the no. of useful training data.

D. *Implementation Flow Chart*

After designing the environment, agent model and rewarding scheme, a framework is built in Python inspired by the one mentioned in [6] to train the RL agent. The algorithm used for training is Deep Q-Learning. The implementation flow-chart is self-explanatory and depicted in Fig. 6.

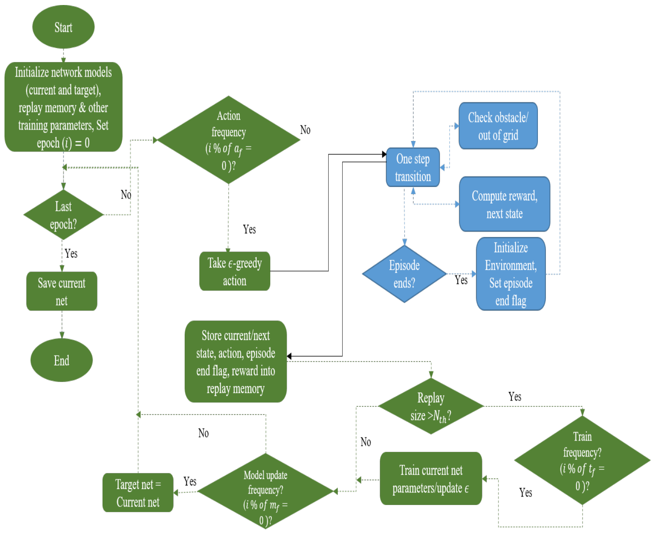
Start
Initialize network models (current and target), replay memory & other training parameters, Set epoch (i) = 0
Last epoch?
No
Yes
Save current net
End
Action frequency (i % of a_f = 0)?
No
Yes
Take ε-greedy action
One step transition
Check obstacle/ out of grid
Compute reward, next state
Episode ends?
Yes
Initialize Environment, Set episode end flag
Store current/next state, action, episode end flag, reward into replay memory
Replay size > N_{th}?
Yes
No
Train frequency? (i % of t_f = 0)?
Yes
Train current net parameters/update ε
Model update frequency? (i % of m_f = 0)?
No
Yes
Target net = Current net

Fig. 6: Implementation Flow Chart

VI. Results

The training is done in HPC cluster, and we found different outcomes for different choice of hyper-parameters. The important results are presented below under the 4 different cases:

A. *Case-I*

- $B = 32$, $T = 250$, $\alpha = 10^{-4}$
- Clip-norm = -, Loss = MSE, $\epsilon_{min} = 0.1$, $N = 10^5$
- No. of Neurons in Hidden Layers = 64, $\gamma = 0.99$
- Uniform exploration/reward scheme

In this case, we observe that the mean score is oscillating after few epochs, and is not following a continually improving trend. We think the primary reason of this failure is non-implementation of zone-wise exploration, which motivated us to enforce zone-wise exploration in the subsequent cases.

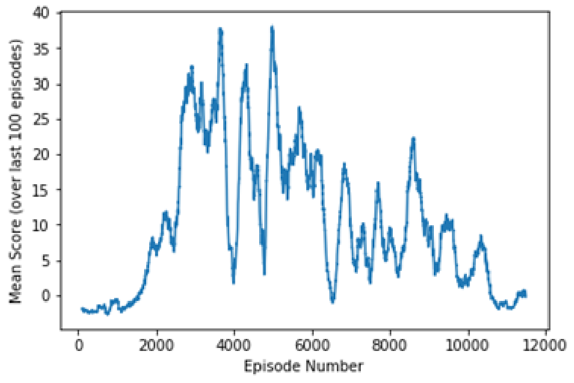

Fig. 7: Mean score (running avg over 100 episodes)

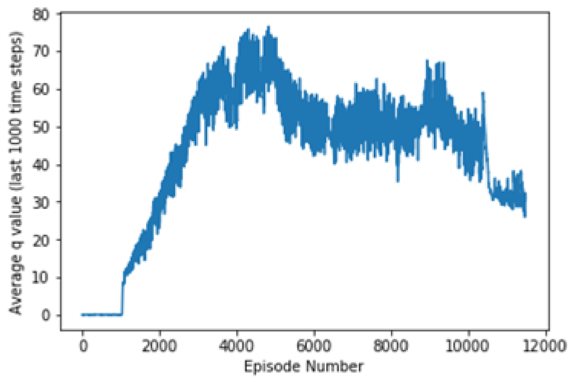

Fig. 8: Mean Q-value (running avg over 1000 episodes)

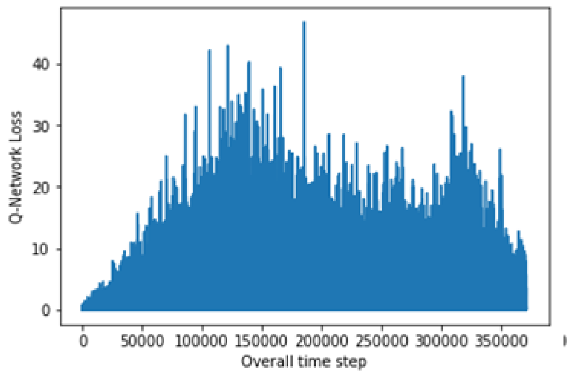

Fig. 9: Network loss

B. *Case-II*

- $B = 200,\ T = 250,\ f_m = 100,\ \alpha = 10^{-5}$
- Clip-norm = 1, Loss = Huber, $\epsilon_{min} = 0.1,\ N = 5\times10^5$
- No. of Neurons in Hidden Layers = 128, $\gamma = 0.9999$
- Zone-wise exploration/reward scheme

In this case, we used zone-wise exploration. In addition, we clipped the gradients at unity norm, and used *Huber loss* (as opposed to *mean squared error* in Case I) to make the process of learning smoother. The learning of the agent has improved over the last case as depicted by the plots below. However, Fig. 10 shows that the trained agent only reaches zone 3; so the training is not yet through.

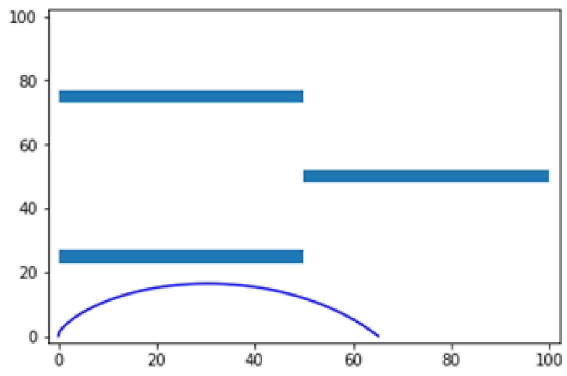

Fig. 10: Performance of Trained Model

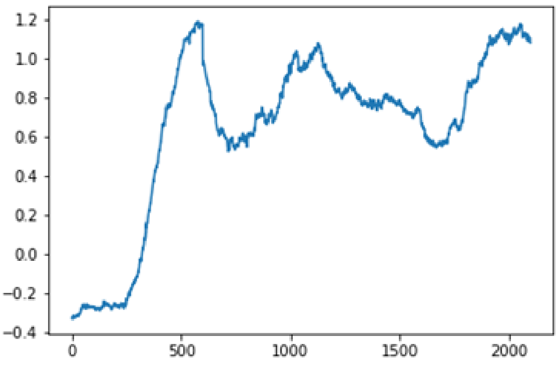

Fig. 11: Mean score (running avg over 100 episodes)

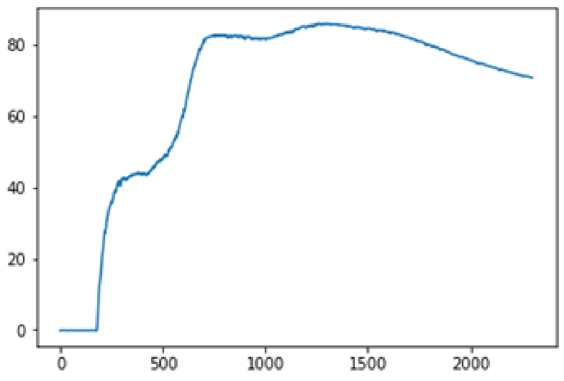

Fig. 12: Mean Q-value (running avg over 1000 episodes)

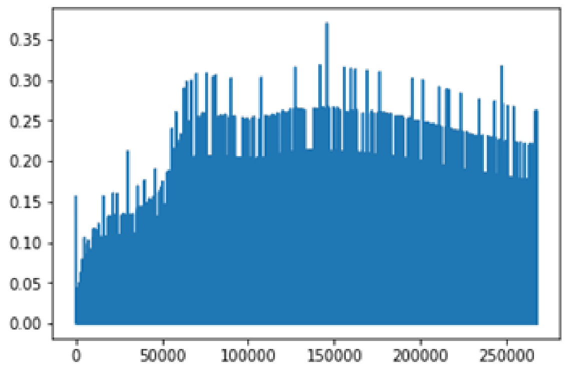

Fig. 13: Network loss

C. *Case-III*

- $B = 200,\ T = 1000,\ f_m = 100,\ \alpha = 10^{-5}$
- Clip-norm = 1, Loss = Huber, $\epsilon_{min} = 0.01,\ N = 5\times 10^5$
- No. of Neurons in Hidden Layers = 128, $\gamma = 0.9999$
- Obstacles removed
- Zone-wise exploration/reward scheme

In order to enable more effective exploration, we eliminate Obstacle 1, 2 and 3 in this case. Our rewarding scheme is designed such that even in absence of these obstacles the optimal policy (if learned effectively) shall drive the vehicle along the desired path (i.e. along the increasing order of zones starting from zone 1). In addition, we increased the episode length and the size of the replay memory (albeit making the computation more memory complex). In this setting the performance of the agent is found better than the previous two cases in that the vehicle reaches nearer to the final goal.

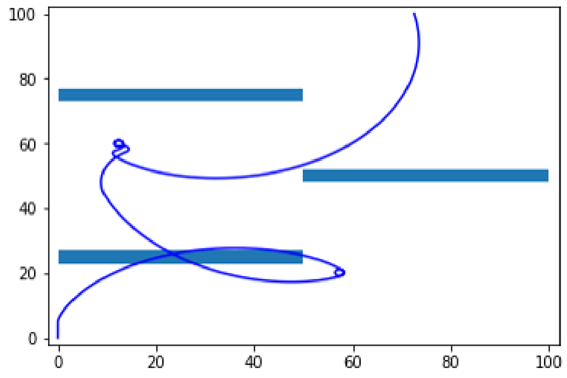

Fig. 14: Performance of Trained Model

D. *Case-IV*

- $B = 200,\ T = 1000,\ f_m = 100,\ \alpha = 10^{-5}$
- Clip-norm = 1, Loss = Huber, $\epsilon_{min} = 0.01,\ N = 5\times 10^5$
- No. of Neurons in Hidden Layers = 128, $\gamma = 0.9999$
- Obstacles removed
- Zone-wise exploration/reward scheme

The training setting in this case is identical to the one in the previous case. Here we retrain the pre-trained network

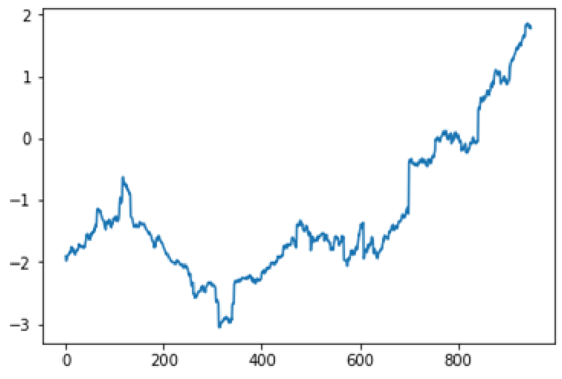
Fig. 15: Mean score (running avg over 100 episodes)

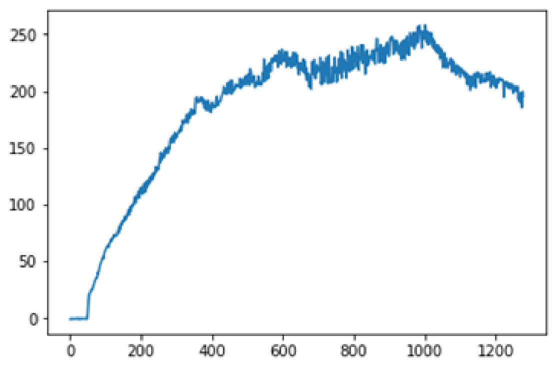
Fig. 16: Mean Q-value (running avg over 1000 episodes)

obtained from Case III, for more number of episodes. As is it is observed in Fig. 19, the mean score continues to increase indicating a positive progress of training.

VII. Conclusion

In this work, we tried to train a deep Q network to infer physical state based optimal action for a miniaturized autonomous vehicle. Our results show that none of the trials could train the network enough to infer the true optimal action from the estimated Q-values. Intuitively, we attribute this failure to the following reasons. First, we observed that the good samples encountered during exploration, are very sparse to be picked frequent enough in the training batches, adversely affecting the training process. Second, we

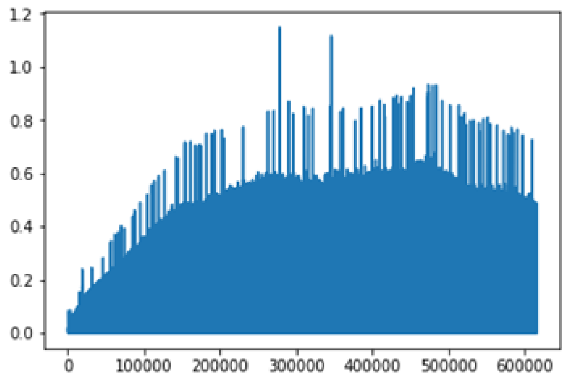
Fig. 17: Network loss

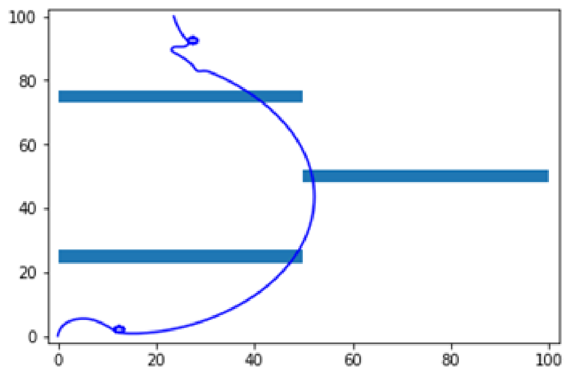
Fig. 18: Performance of Trained Model

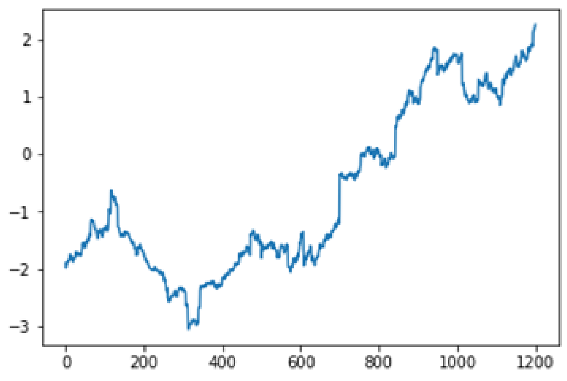
Fig. 19: Mean score (running avg over 100 episodes)

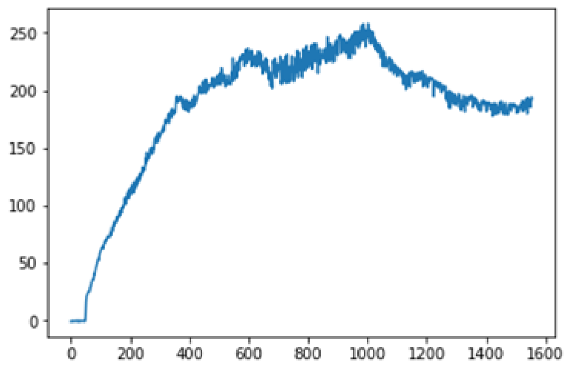
Fig. 20: Mean Q-value (running avg over 1000 episodes)

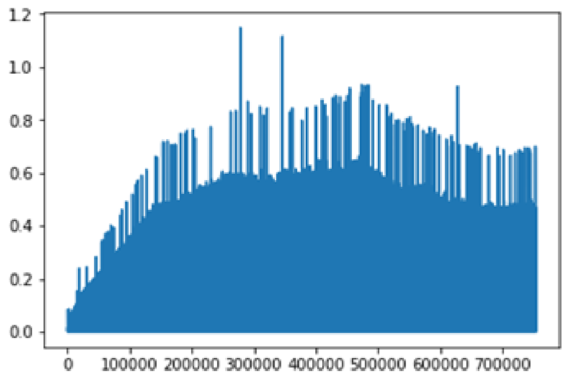
Fig. 21: Network loss

believe that the vehicle dynamics in this particular problem, is more complex (seen in the switched nonlinearity) than the environments typically used to demonstrate performance of RL. As a future extension of this work, we would pursue implementation of *priority sampling*, an approach suggested in [8], [9], to overcome the aforecited first reason. We are optimistic that a more rigorous hyper-parameter tuning on a more complex network architecture would possibly alleviate the concern pertaining to the complex dynamics.

VIII. Acknowledgement

We are thankful to the course-instructor Prof. S Sarkar, and all the teaching assistants for their valuable suggestions, during this work.

References

[1] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[2] D. Howard and D. Dai, "Public perceptions of self-driving cars: The case of berkeley, california," in *Transportation Research Board 93rd Annual Meeting*, vol. 14, no. 4502, 2014, pp. 1–16.

[3] C. Urmson *et al.*, "Self-driving cars and the urban challenge," *IEEE Intelligent Systems*, vol. 23, no. 2, pp. 66–68, 2008.

[4] G. Hee Lee, F. Faundorfer, and M. Pollefeys, "Motion estimation for self-driving cars with a generalized camera," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 2746–2753.

[5] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Transactions on intelligent vehicles*, vol. 1, no. 1, pp. 33–55, 2016.

[6] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[7] J. J. Nutaro, *Building software for simulation: theory and algorithms, with applications in C++*. John Wiley & Sons, 2011.

[8] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[9] A. W. Moore and C. G. Atkeson, "Prioritized sweeping: Reinforcement learning with less data and less time," *Machine learning*, vol. 13, no. 1, pp. 103–130, 1993.