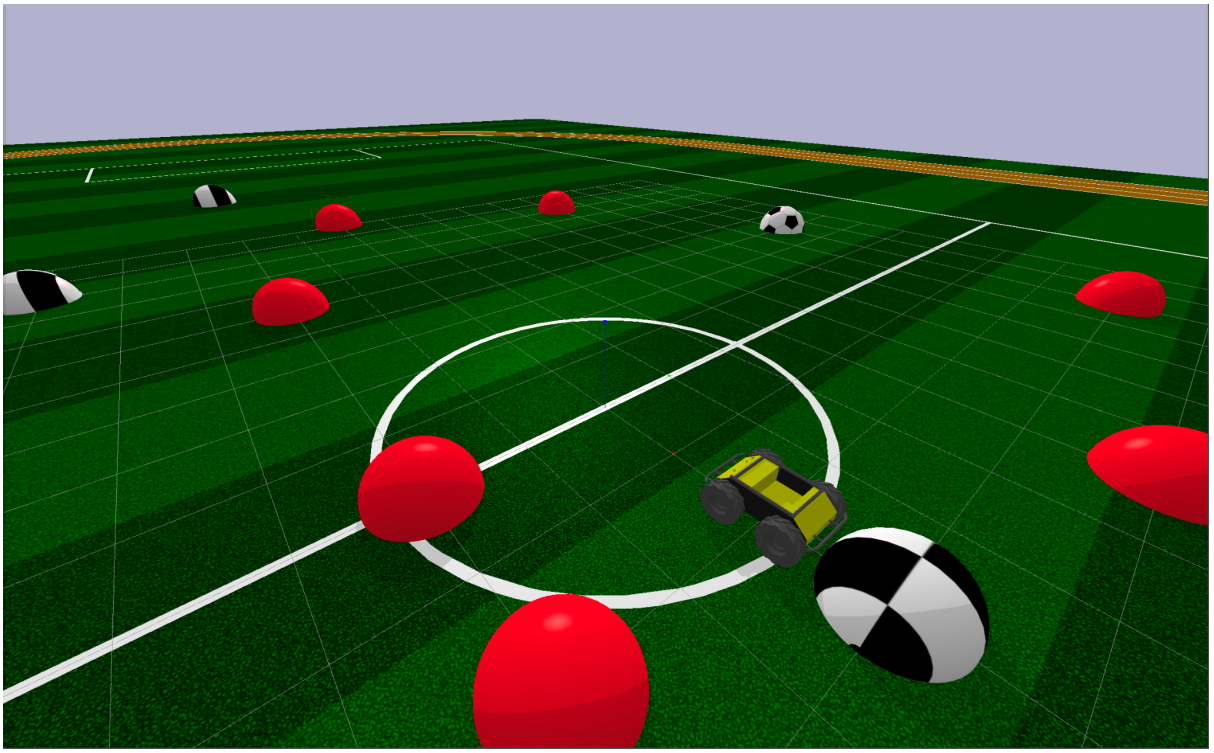
Deep Deterministic Policy Gradients in Pytorch with Simulation in PyBullet

Objective:
Use Deep Deterministic Policy Gradients to find the optimal path to the destination using an environment created using pybullet. The environment contains obstacles and a goal point(soccerball). The input states to the network are images and the total area of obstacles in the scene (found using segmentation masks).

The environment in PyBullet (Car with obstacles & Soccerball Goal point)



Theory

Why do we need DDPG?

Deep Q networks only work for low dimensional discrete action space. It relies on finding the action that maximizes the action-value function, which in the continuous-valued case requires an iterative optimization process at every step.

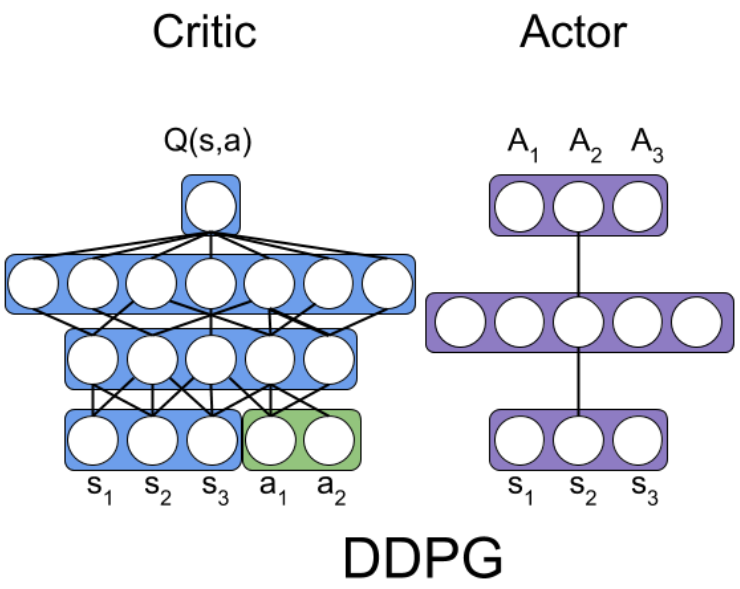
Therefore, we use Deep Deterministic Policy Gradients(DDPG) for continuous-valued high dimensional action space.

Deep Deterministic Policy Gradients(DDPG)

Deep Deterministic Policy Gradients (DDPG) algorithm uses deep networks to regress to the most optimum action in continuous high dimensional action space. DDPG combines both deep Q networks and policy gradients.
DDPG is an "**off-policy**" method. DDPG is "**deterministic**" since the actor computes the action directly instead of a probability distribution over actions.

Model

It uses an actor-critic model. The actor network is the policy network that takes states as input and outputs a continuous action value between [-1,1]. The policy is **deterministic** since it directly outputs the action. In order to promote **exploration** some Ornstein Uhlenbeck noise is added to the action determined by the policy.



The critic network is a Q-value network that takes as input both the state and the action value(from the actor) + noise, then outputs a Q-value. We try to maximize the Q-value obtained for the action decided by the actor network, and minimize the critic loss. The critic loss is computed by the TD error where target networks are used to compute Q-value for the next state.

To stabilize learning we create target networks for both critic and actor. These target networks will have soft-updates based on main networks.

Algorithm

Algorithm 1 DDPG algorithm

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

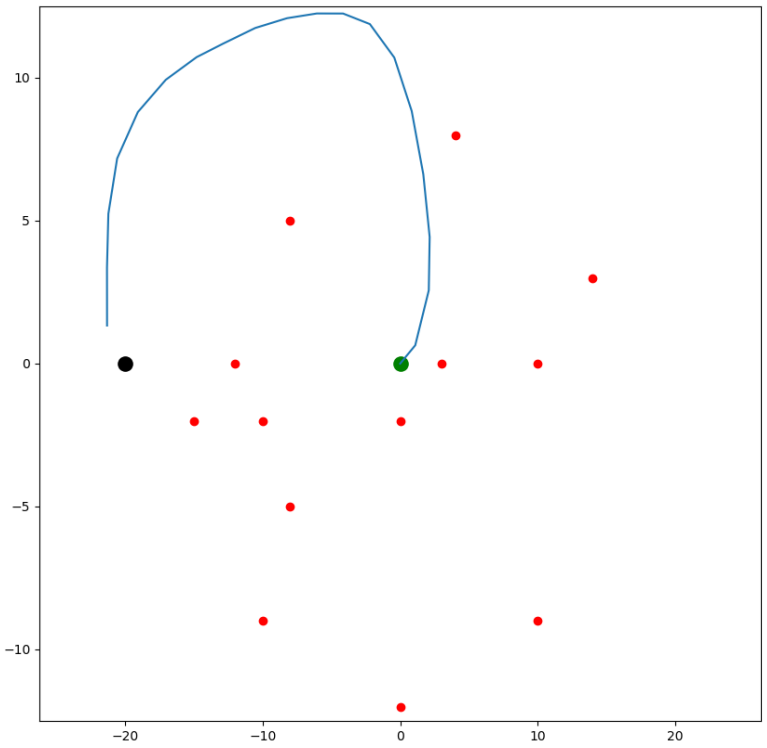
Training Details

Hyperparameters Used:

- Input State: **image captured + total area of all obstacles obtained using segmentation masks**
- Output Action - **Steering Angle [-1,1] ->ranging from -90 degree left to +90 right**
- Reward
  - On reaching goal = 5000
  - On collliding with obstacles = -0.01*area of obstacles
  - Outside boundary = -100
- Threshold distance to goal = 2
- Episodes = 500
- Max_Steps = 60
- Replaymemory size = 1000
- Ornstein Uhlenbeck noise
  - Sigma = 0.2
  - Theta = 0.15
- Initialization of network weights = 3e-4
- Actor LR = 0.0001
- Critic LR = 0.001

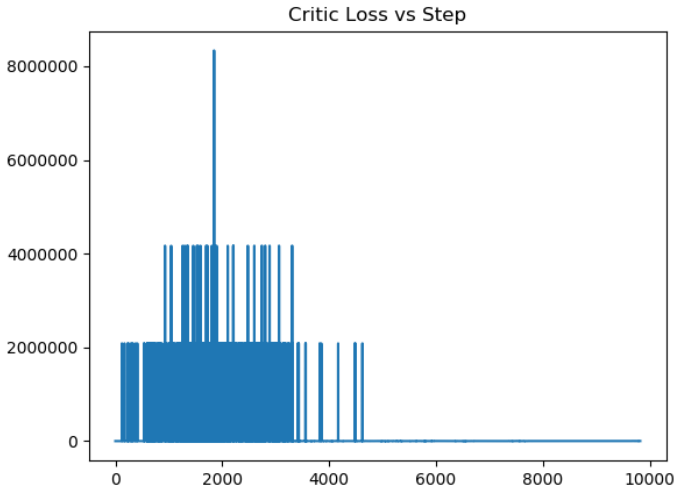
Results for two different Goal points without a continuous reward function

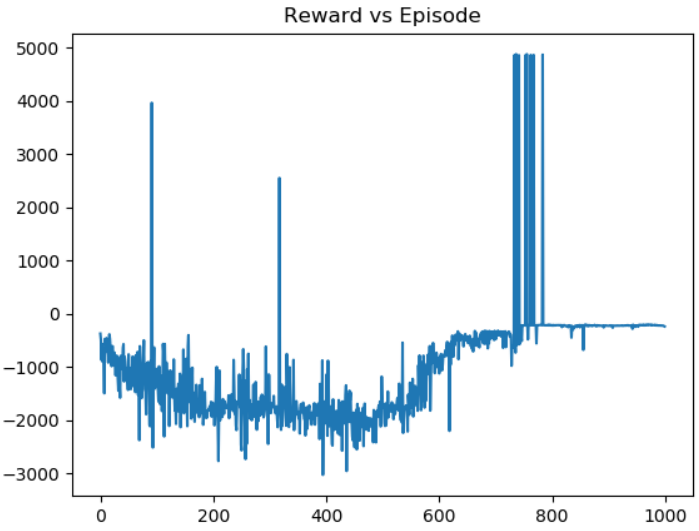
1. **Goal point = (-20, 0)**
   a. **Trajectory**



   b. **Video - Car reaching the goal**
   https://iiitaphyd-my.sharepoint.com/:v:/g/personal/gowri_lekshmy_research_iiit_ac_in/Ed1I5LB9KudLu0JB2PI08zUBMXV-5wMBuKLbkHzGdS6VDw?e=hmvrWD

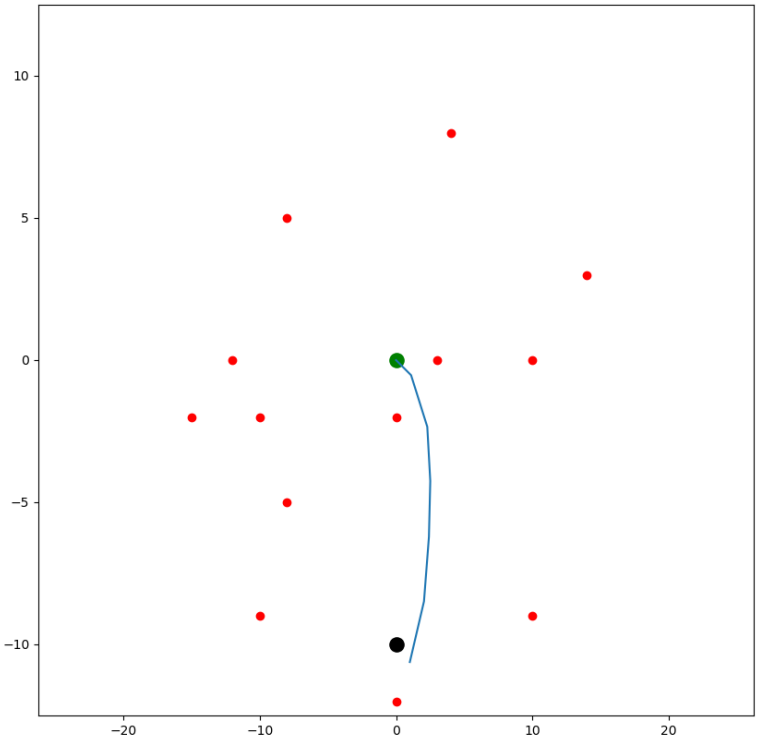
   c. **Plots**





2. Goal point = (0, -10)

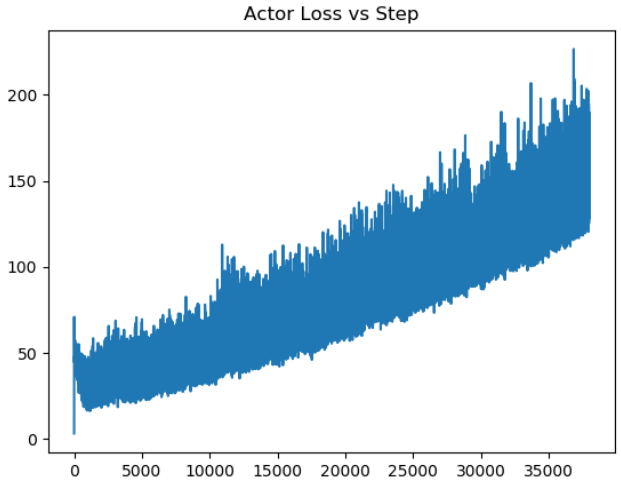
a. Trajectory

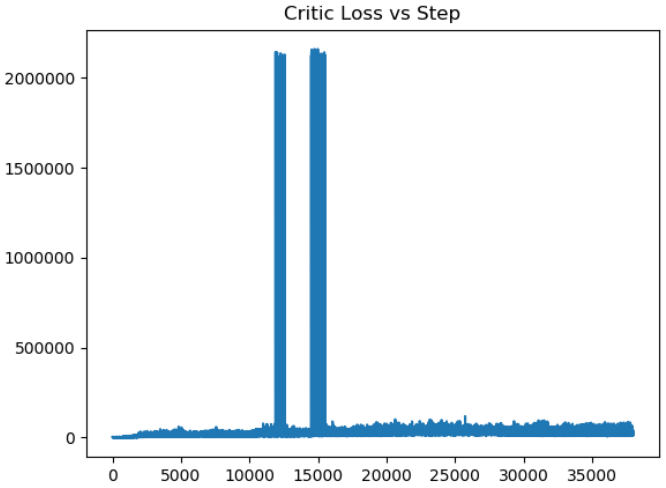


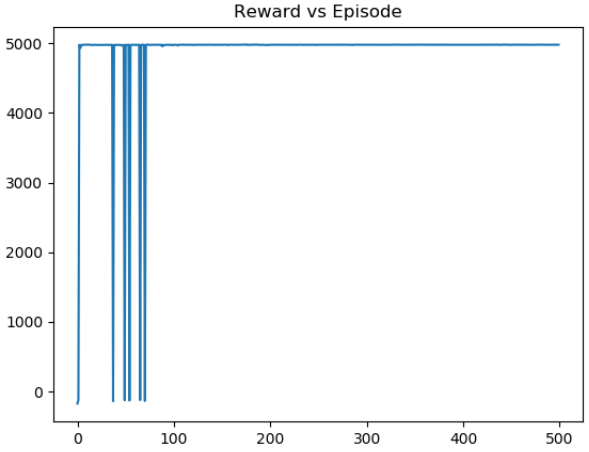
b. Video - Car reaching the goal

https://iiitaphyd-my.sharepoint.com/:v:/g/personal/gowri_lekshmy_research_iiit_ac_in/EVqrnRGLMHxHiPzIyrY44M8BT0A6AbAeMhjqTdW6-MCcdQ?e=DH93gr

c. Plots


Actor Loss vs Step
200
150
100
50
0
0
5000
10000
15000
20000
25000
30000
35000


Critic Loss vs Step
2000000
1500000
1000000
500000
0
0
5000
10000
15000
20000
25000
30000
35000


Reward vs Episode
5000
4000
3000
2000
1000
0
0
100
200
300
400
500

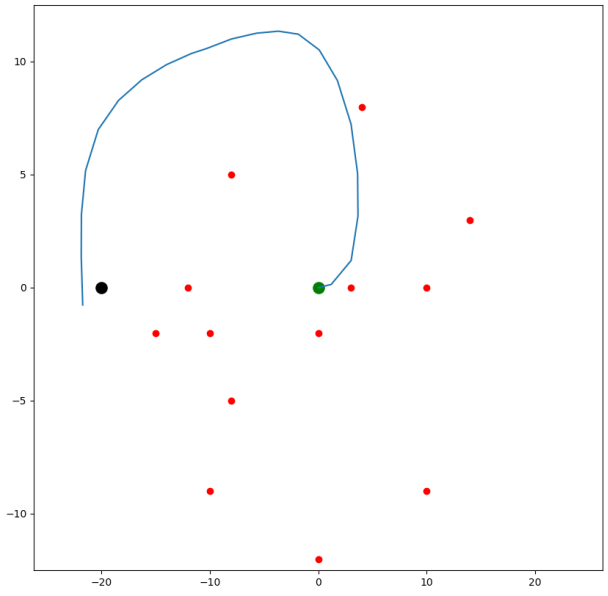
Reward function:

Initially, the reward function used only binary rewards for good action and bad action. It used +ve binary reward for reaching the goal and -ve rewards for both going out of boundary & colliding with obstacles. This non-continuous reward function was observed to give sudden peaks in the reward function so a new continuous reward function was used.
Old Reward = F(Goal reward, outside Boundary)

The new reward function considered the distance to the goal and the visible obstacles into account.
New Reward = -dist_scale*(distance to goal) + obstacle_scale*(total distance to all the visible obstacles)

Results for two different Goal points with a continuous reward function

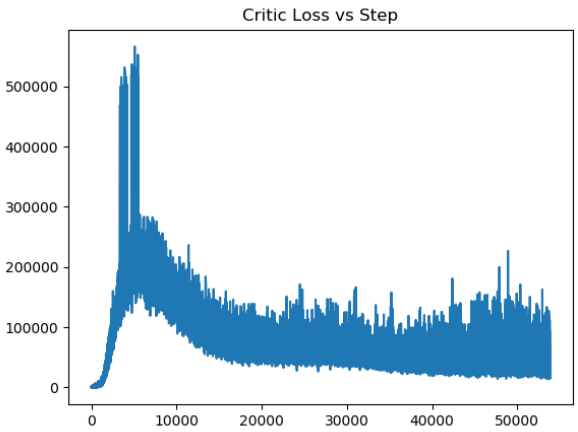
1. **Goal point = (-20, 0)**
    a. **Trajectory**

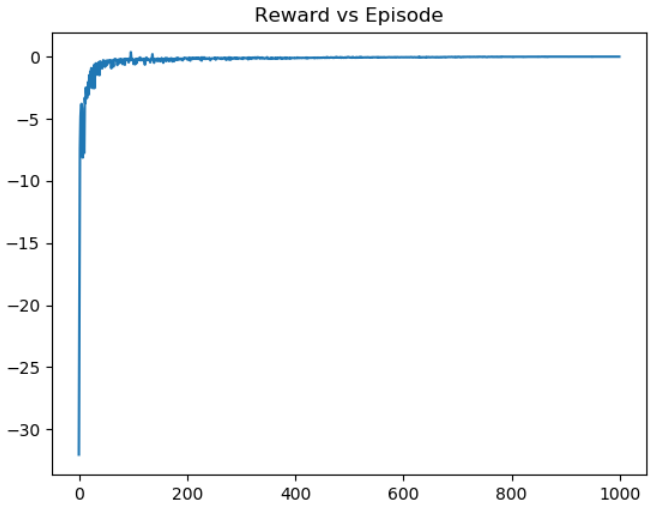


    b. **Video - Car reaching the goal**
    https://iiitaphyd-my.sharepoint.com/:v:/g/personal/gowri_lekshmy_research_iiit_ac_in/EWdviWZdChJHpdJCJShQae8Bl-cr8WROX4RNpPslQja5JA?e=TEuSe8

c. Plots

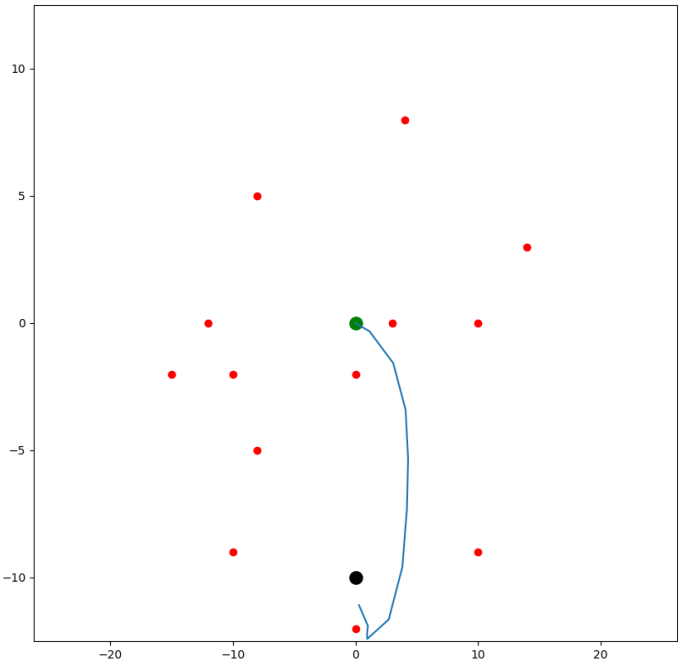

Actor Loss vs Step
350
300
250
200
150
100
50
0
0
10000
20000
30000
40000
50000


Critic Loss vs Step
500000
400000
300000
200000
100000
0
0
10000
20000
30000
40000
50000


Reward vs Episode
0
−5
−10
−15
−20
−25
−30
0
200
400
600
800
1000

2. Goal point = (0, -10)

a. Trajectory



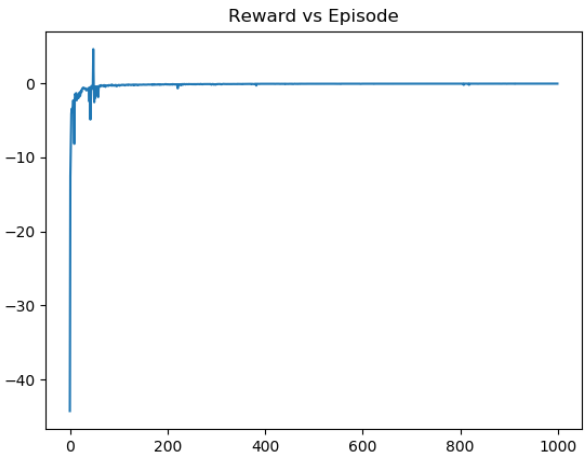
b. Video - Car reaching the goal

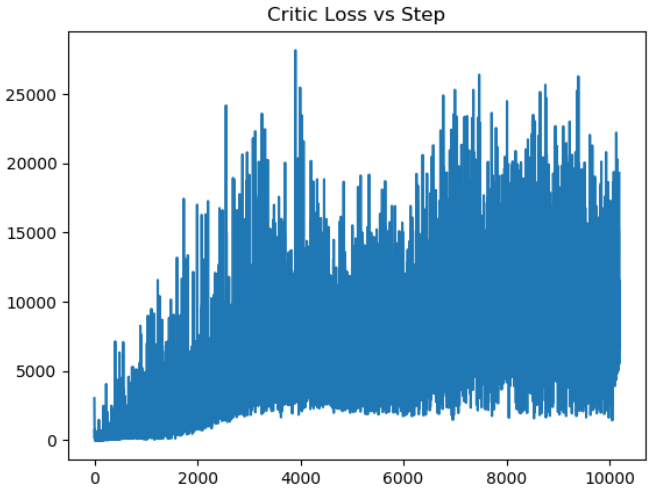
https://iiitaphyd-my.sharepoint.com/:v:/g/personal/gowri_lekshmy_research_iiit_ac_in/EVt6i7RLPUxDlrx5oaMr2sEB0uvjxxfcx4vnUeuhFOy4CA?e=7aqoL2

c. Plots

-

-





References

1. Lillicrap, et al. Continuous control with Deep Reinforcement Learning
2. Silver, et al. Deterministic Policy Gradients