Interactive Learning Final Project

Ali Saeizadeh - 810196477
Kasra Borazjani - 810196662

Outline

What this presentation is going to discuss

- Problem Description
- Methodology
- Experiments
- References

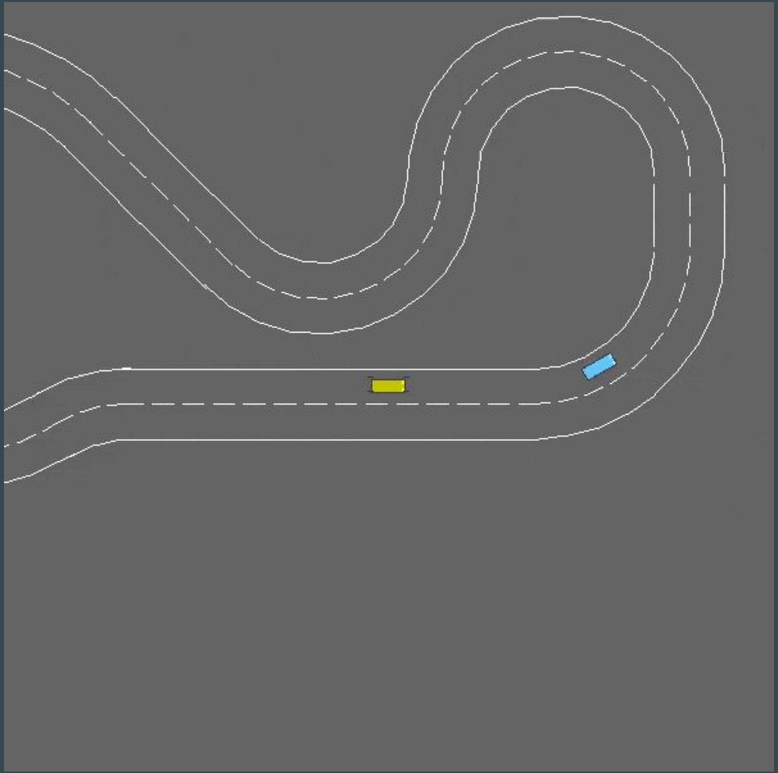
Problem Description

Problem Description

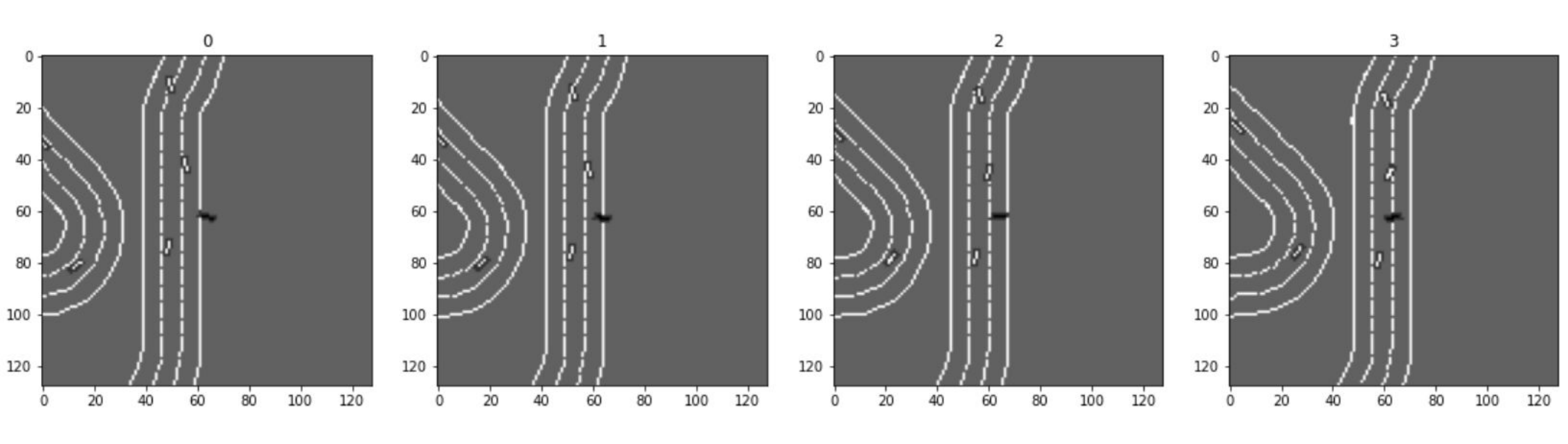
- HardRacetrack Environment
  - RGB
  - 128 x 128
  - Discrete Actions

- States
  - Previous Four Observations
  - Terminal state after 300 steps or car collision

- Actions
  - Steering Wheel Angle ($-1 < \theta < 1$)
  - Acceleration ($-1 < a < 1$)



Observation Sample



Methodology

Function Approximators

Why use function approximators?

- States are made up of previous four observations
- Cannot be processed using basic RL methods due to various positions of the car relative to the track and surrounding cars

How do we solve this problem?

- Using Deep Q-Learning methods that employ deep neural networks to process images, namely CNNs

Deep Q Learning

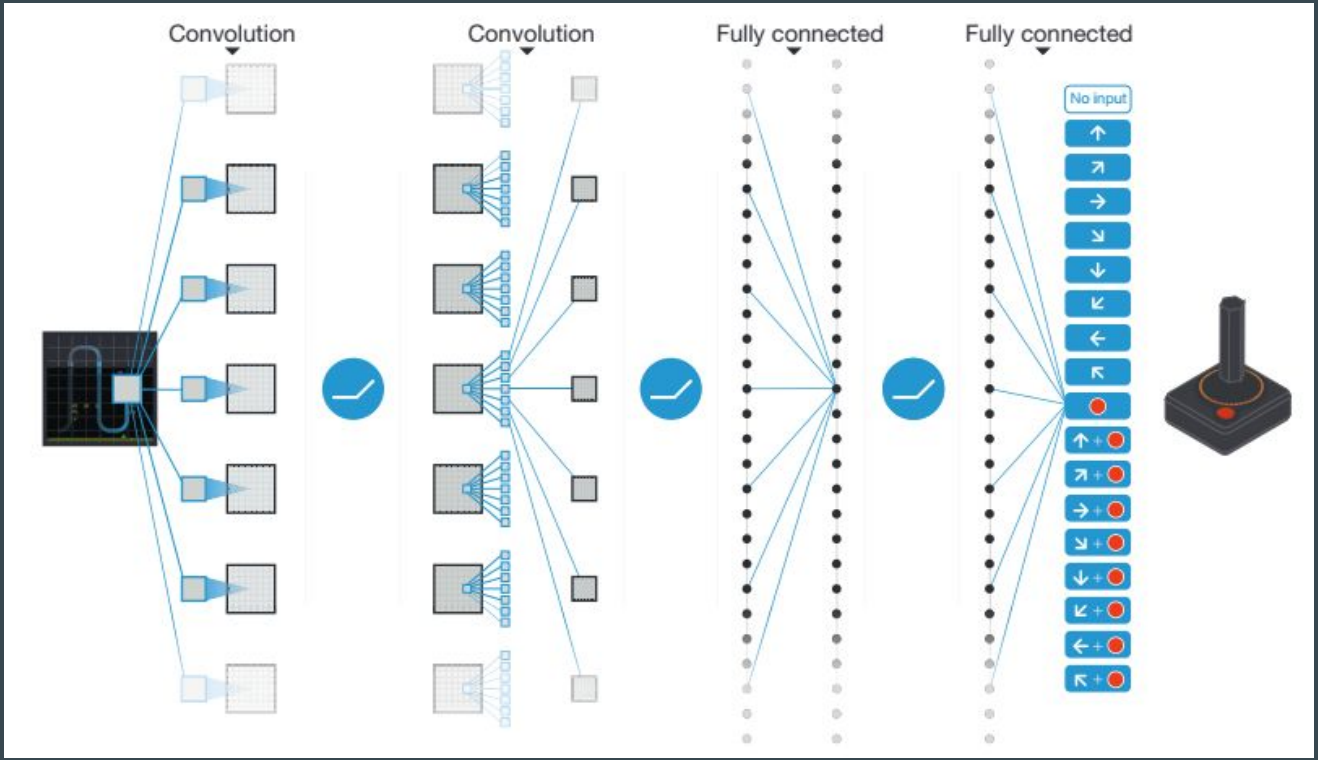
$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot);s'\sim\mathcal{E}}\left[\left(r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) - Q(s,a;\theta_i)\right)\nabla_{\theta_i} Q(s,a;\theta_i)\right]. \quad (3)$$

- Pre-Processing Phase
  - RGB to Grayscale
  - Cropping
- Training Phase
  - Experiencing
  - Mini-Batch Recall
  - DNN Training

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation $\boxed{3}$
  end for
end for

[1]



Deep Q-Learning Network Demonstration [3]

Double Deep Q Networks

DQN Deficiencies:

- Overestimations

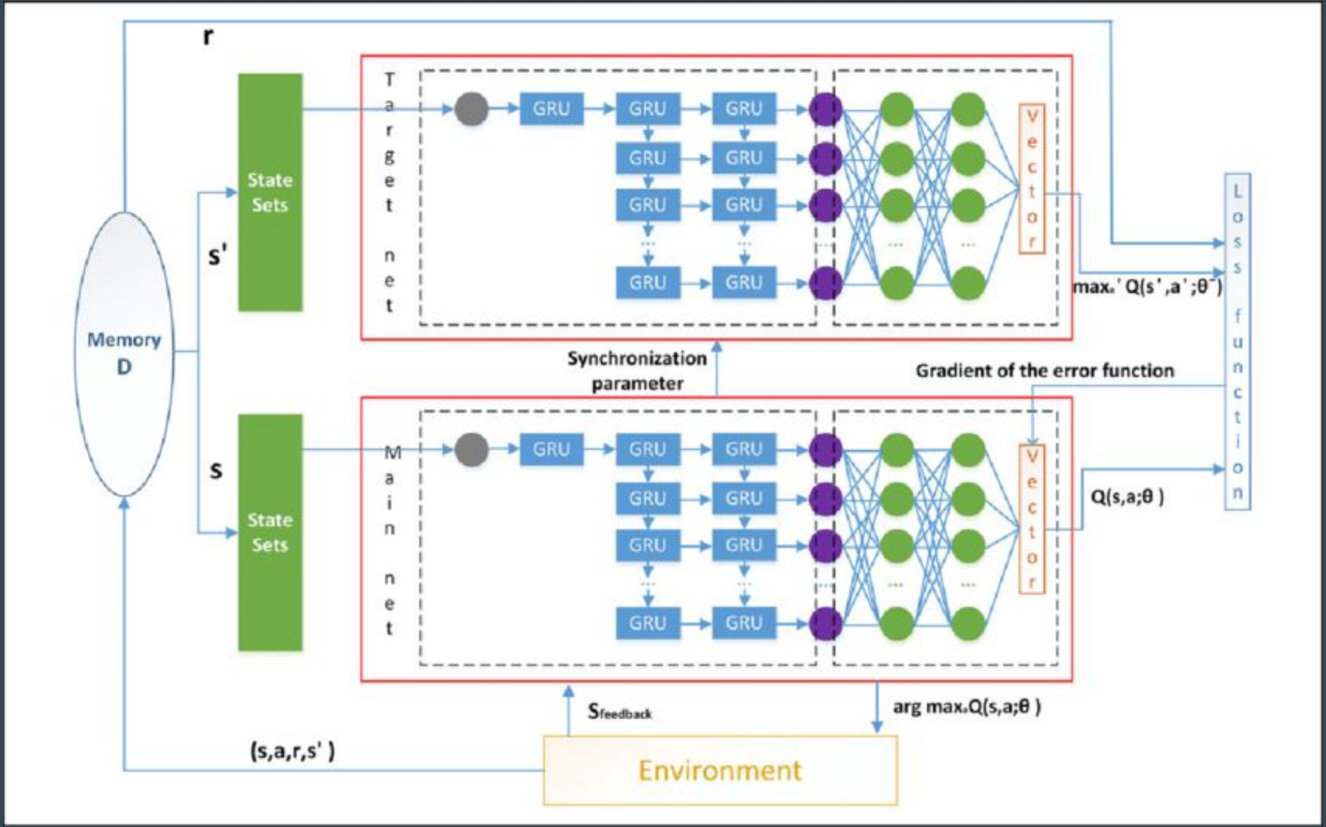
DDQN Improvements:

- Main (Online) net
- Target net

Algorithm 1 : Double Q-learning (Hasselt et al., 2015)

Initialize primary network Q_θ, target network $Q_{\theta'}$, replay buffer $\mathcal{D}$, $\tau << 1$
for each iteration **do**
    for each environment step **do**
        Observe state s_t and select $a_t \sim \pi(a_t, s_t)$
        Execute a_t and observe next state s_{t+1} and reward $r_t = R(s_t, a_t)$
        Store (s_t, a_t, r_t, s_{t+1}) in replay buffer $\mathcal{D}$
    for each update step **do**
        sample $e_t = (s_t, a_t, r_t, s_{t+1}) \sim \mathcal{D}$
        Compute target Q value:
            $Q^*(s_t, a_t) \approx r_t + \gamma\, Q_\theta(s_{t+1}, argmax_{a'} Q_{\theta'}(s_{t+1}, a'))$
        Perform gradient descent step on $(Q^*(s_t, a_t) - Q_\theta(s_t, a_t))^2$
        Update target network parameters:
            $\theta' \leftarrow \tau * \theta + (1 - \tau) * \theta'$

[2]



[4]

Double Deep Q Learning Architecture

DDQN Loss Calculation Algorithm

- Upper Equation:

  Finding estimated Q-value from the online network

- Middle Equation:

  Finding the appropriate action for the current state based on the Q-values of the online network

- Bottom Equation:

  Acquiring the Q-value based on the greedier policy for comparison and loss calculation

$$TD_e = Q^*_{online}(s, a)$$

$$a' = argmax_a \, Q_{online}(s', a)$$

$$TD_t = r + \gamma Q^*_{target}(s', a')$$

DDQN Network Weight Updating Algorithm

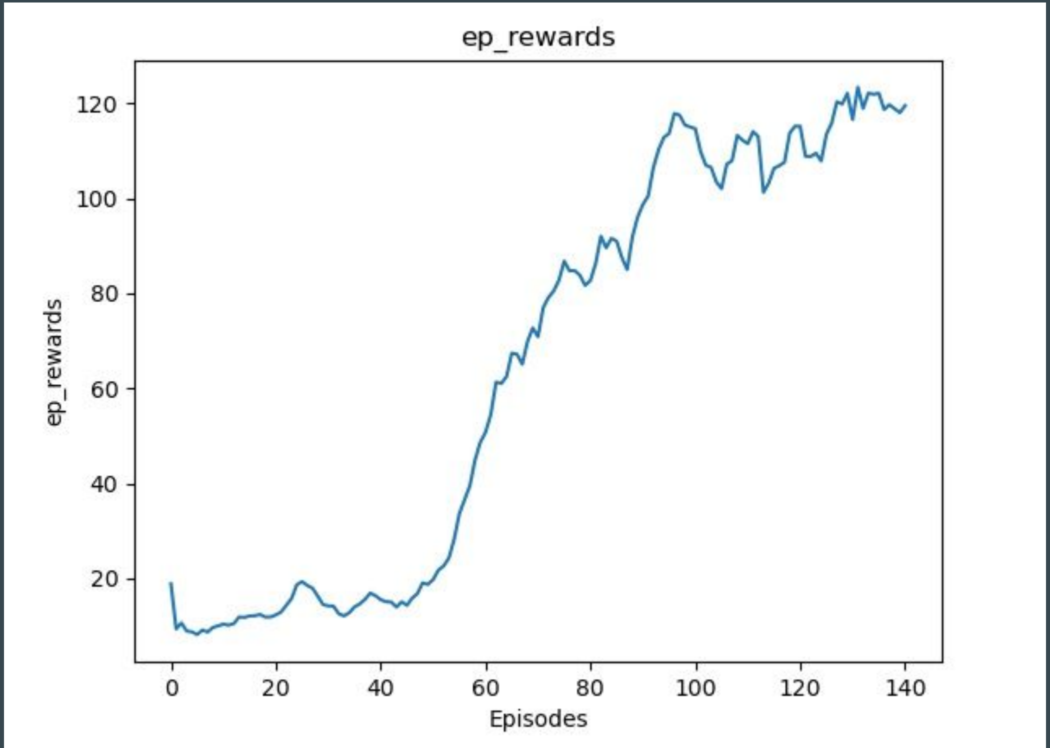
- The Online network gets updated with each episode based on the equation on the right.
- The Target network's weights are synced with the Online network each "n" episodes.

$$\theta_{online} \leftarrow \theta_{online} + \alpha\nabla(TD_e - TD_t)$$

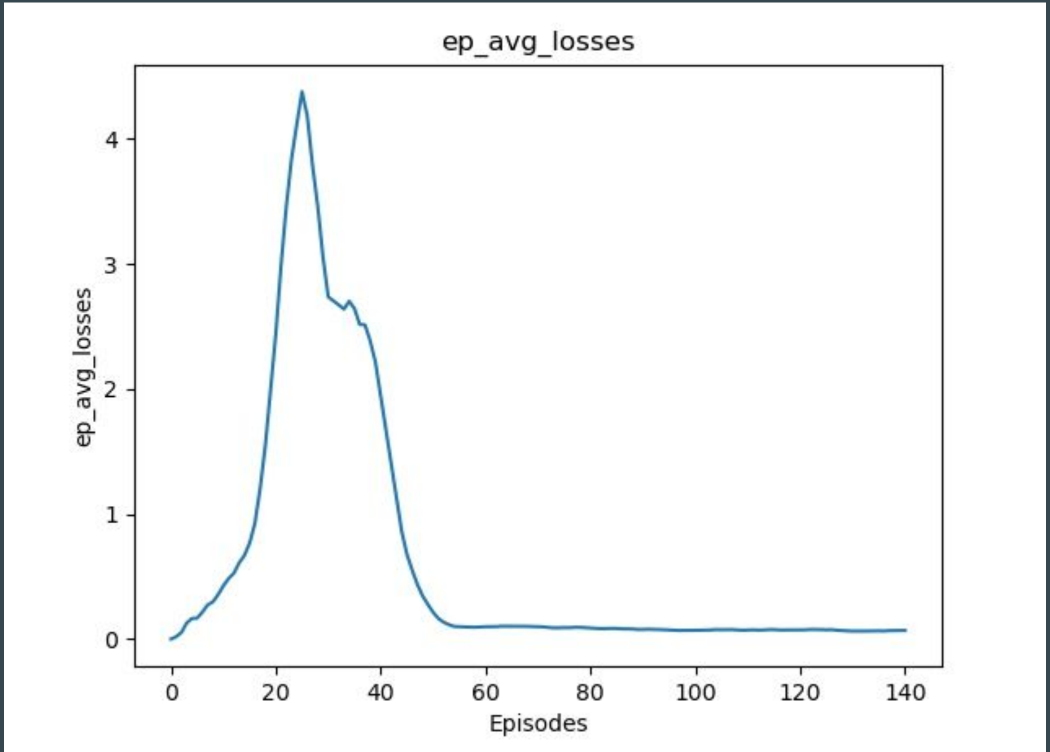
Experiments

Experiment Environment

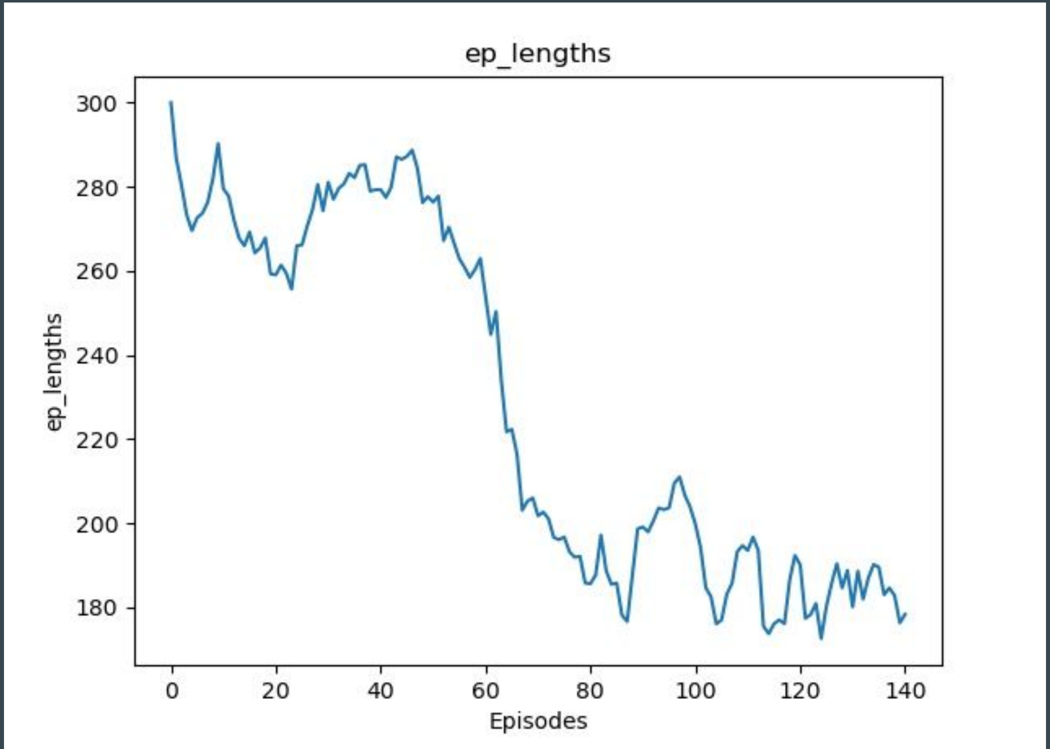
- 16GB RAM
- NVidia 1050 Ti Graphics Card
- Intel Core i7 CPU
- 2800 episodes
- Total run time 6h 40m
- 633247 steps



Average reward per 20 episodes - 2800 episodes in total



Average loss per 20 episodes - 2800 episodes in total



Average length played per 20 episodes - 2800 episodes in total

References

References

[1] Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning." arXiv preprint arXiv:1312.5602 (2013).

[2] Van Hasselt, Hado, Arthur Guez, and David Silver. "Deep reinforcement learning with double q-learning." Proceedings of the AAAI conference on artificial intelligence. Vol. 30. No. 1. 2016.

[3] Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." nature 518.7540 (2015): 529-533.

[4] Quan, Hao, et al., “A novel mobile robot navigation method based on deep reinforcement learning”, International Journal of Advanced Robotic Systems, May 2020