Robot Navigation using Deep Reinforcement Learning

Presented by

- Ahmed Mohsen Ali
- Felix Sihitshuwam
- Sohaila Ahmed

November 20, 2022

Outline

- Introduction
- Problem Formulation
- Related Work
- Methodology
- Results and Discussions
- Conclusion

Members' Contribution

- Felix Dalang
  - Environment, reward, and control design
  - Environment implementation using OpenAI Gym
  - Abstract, Introduction, Problem formulation
- Sohaila Ahmed
  - DQN network implementation
  - Integration with discrete action space
  - Methodology (DQN), results and conclusion
- Ahmed Mohsen
  - DDPG and TD3 implementation
  - Integration with continuous action space
  - Related work, Methodology (TD3 & DDPG), and Results

Introduction

- Robot navigation involves the movement of a mobile robot from an initial pose $A\ (x, y, \theta)$ to a target pose $B\ (x_g, y_g, \theta_g)$. This is generally done via path planning algorithms.
- Can we apply Reinforcement learning (RL) to robot navigation? RL refers to a class of algorithms from which an agent learns to make optimal decisions by interacting with the environment.
- RL supports the model free approach, in which we can disregard the kinematic model of the robot. If successfully applied, RL might guarantee better accuracy compared to existing path planning approaches for navigation through the environment.

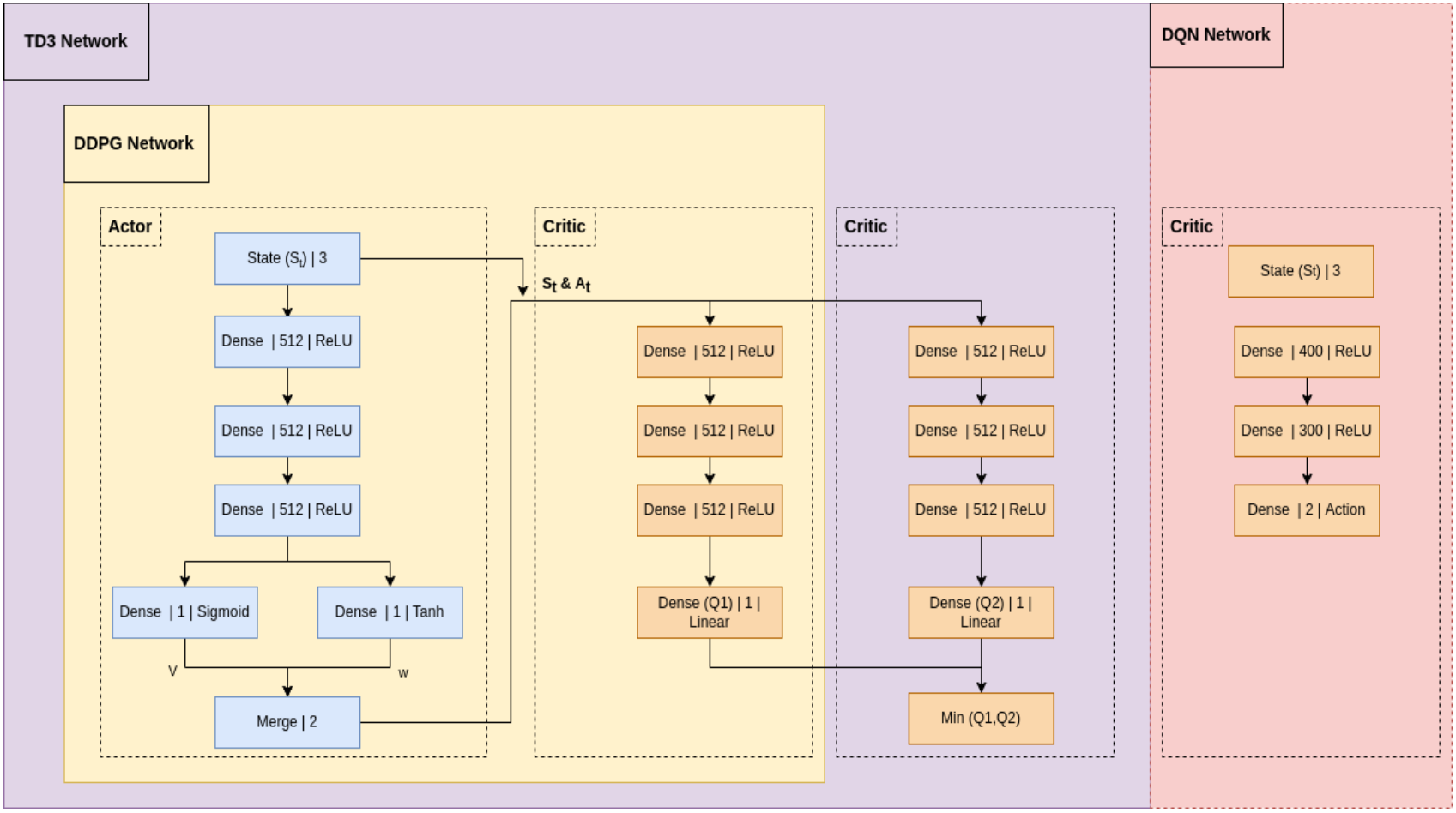
Problem Formulation

- Minimize the distance between the robot's initial pose, and the target location, also align the robot's orientation towards the target location by issuing out appropriate actions (*discrete* and *continuous linear* and *angular* velocities) to update the robot's state.
- Reward the robot in increasing measures for being progressively close to the target. Also reward the robot proportionally to how it aligns itself to the target. Penalize the robot for hitting walls and obstacles.
- Episodes are terminated when robot gets to target, hits a wall or reaches 1000 steps.

Related Work

- Several approaches were used to solve robot navigation.
- The approach differs depending on the type of map, action space, and sensors used.
- Models used are: DQN, ADDPG, A3C, and TD3.

Methodology



DQN

Algorithm 1 Deep Q-Network

Initialize replay memory D with size N
Initialize action-value function Q with random weights
for episode $\leftarrow 1$ to M **do**
  for $t \leftarrow 1$ to T **do**
    if ε **then**
      $a_t \leftarrow$ select random action with probabilityε
    else if $(1-\varepsilon)$ **then**
      $a_t \leftarrow \arg\max_a Q(s,a)$
    end if
    Execute action a_t
    Observe reward r_t and next state s_{t+1}
    $D \leftarrow s_t, a_t, r_t, s_{t+1}$
    Sample a random mini batch from D
    Set $y_j = \left\{ \begin{array}{ll} r_j, & \text{if episode terminates} \\ r_j + \gamma \max_{a'} Q(s',a'), & \text{otherwise} \end{array} \right\}$
    Perform gradient descent on $(y_j - Q(s_j, a_j))^2$
  end for
end for

DDPG

Algorithm 2 Deep Deterministic Policy Gradient

Initialize main critic Q and main actor networks μ with weights θ^Q, θ^μ
Initialize target critic Q_{targ} and actor μ_{targ} with weights $\theta^Q_{targ}, \theta^\mu_{targ}$
Initialize replay memory D with size N
for episode $\leftarrow 1$ to M **do**
  for $t \leftarrow 1$ to T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}$ from main policy μ with noise
    Execute action a_t and observe reward r_t and new state s_{t+1}
    $D \leftarrow s_t, a_t, r_t, s_{t+1}$
    Select mini batch b from D
    Update main Critic and Actor losses
    Update target network with a smaller rate:

$$\theta^\mu_{targ} \Longleftarrow \tau\theta^\mu_{targ} + (1-\tau)\theta^\mu$$
$$\theta^Q_{targ} \Longleftarrow \tau\theta^Q_{targ} + (1-\tau)\theta^Q$$

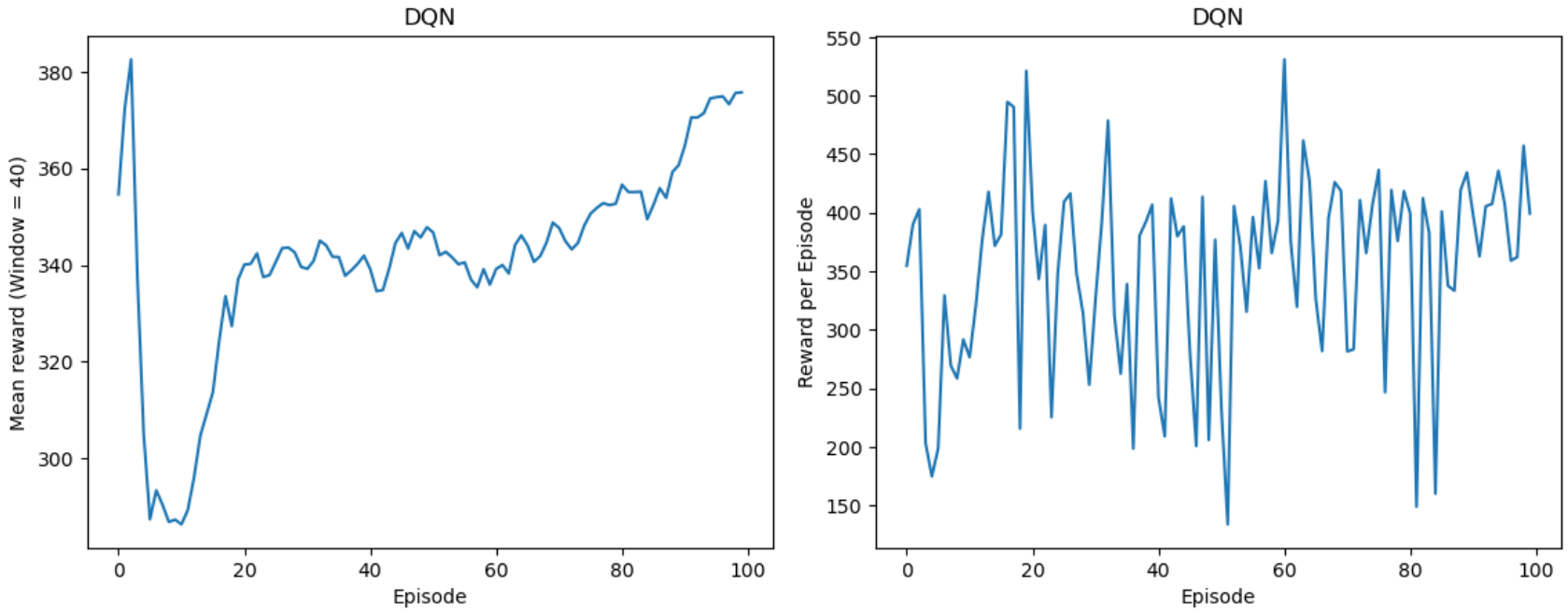
  end for
end for

TD3

Algorithm 3 Temporal-Difference 3

Initialize main critic Q and main actor networks μ with weights θ^Q, θ^μ
Initialize two target critic $Q_{targ1,2}$ and actor μ_{targ} with weights $\theta^Q_{targ}, \theta^\mu_{targ}$
Initialize replay memory D with size N
for episode $\leftarrow$ 1 to M **do**
  for $t \leftarrow$ 1 to T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}$ from main policy μ with noise
    Execute action a_t and observe reward r_t and new state s_{t+1}
    $D \leftarrow s_t, a_t, r_t, s_{t+1}$
    Select mini batch b from D
    Select action $a_{targ} = \mu(s_t|\theta_{targ})$ for TD target
    if $TargetUpdate$ **then**
      Update Policy network μ
      Update target networks $Q_{\theta_1}, Q_{\theta_2}, \mu_{targ}$
    end if
  end for
end for

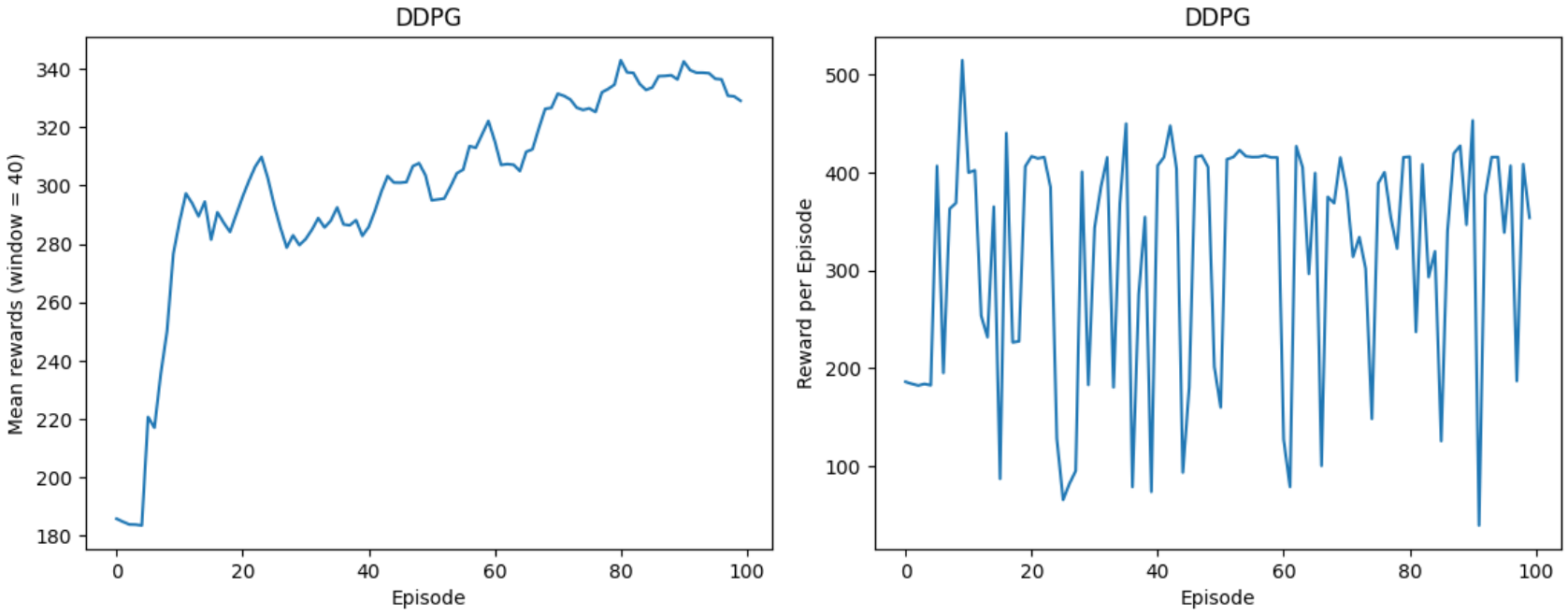
Results and Discussion: DQN



(a) Average reward (100 episodes) (b) Reward per episode (100 episodes)

Figure: (a) shows the average rewards for the DQN for 100 episodes while (b) show the cumulative reward per episode (return).

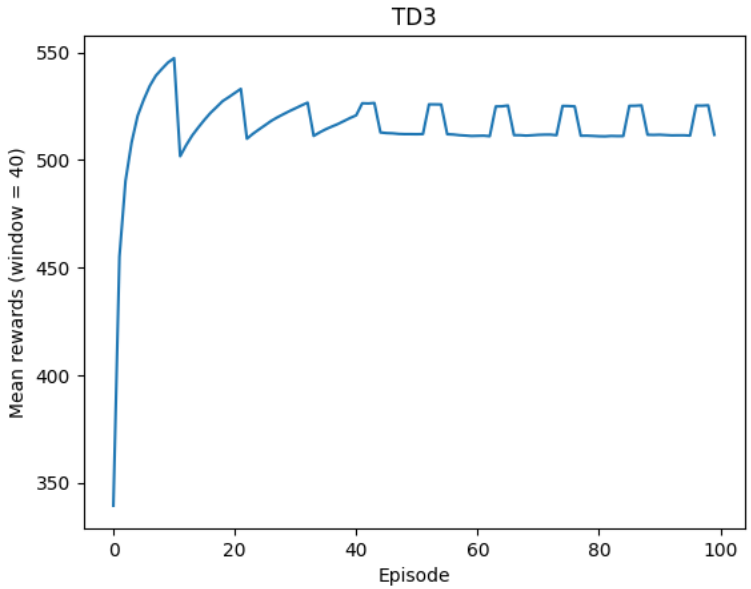
Results and Discussion: DDPG



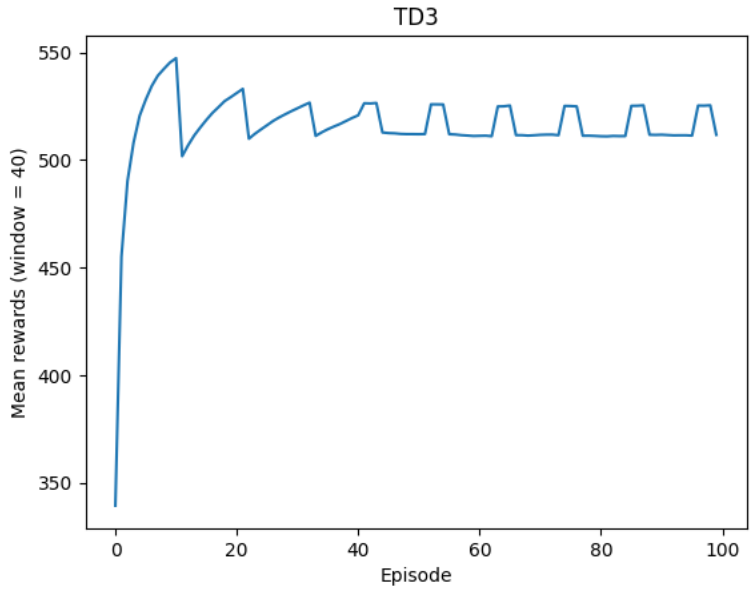
(a) Average reward (100 episodes) (b) Reward per episode (100 episodes)

Figure: (a) shows the average rewards for the DDPG for 100 episodes while (b) show the cumulative reward per episode (return).

Results and Discussion: TD3

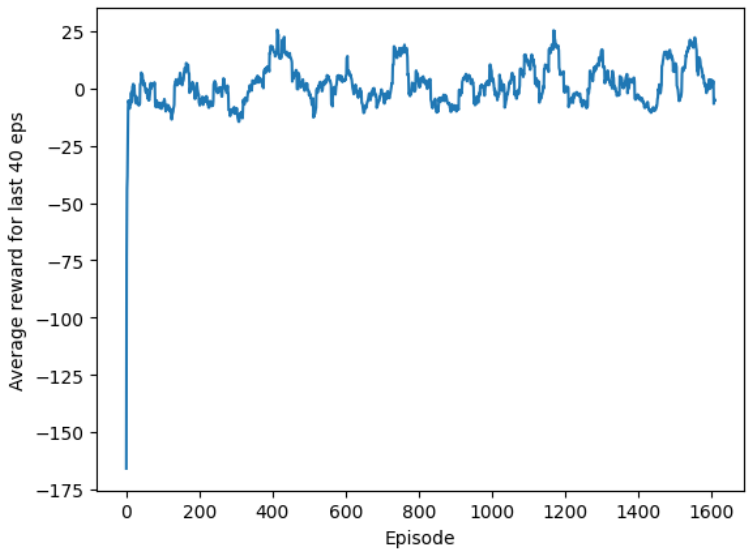


(a) Average reward (100 episodes)

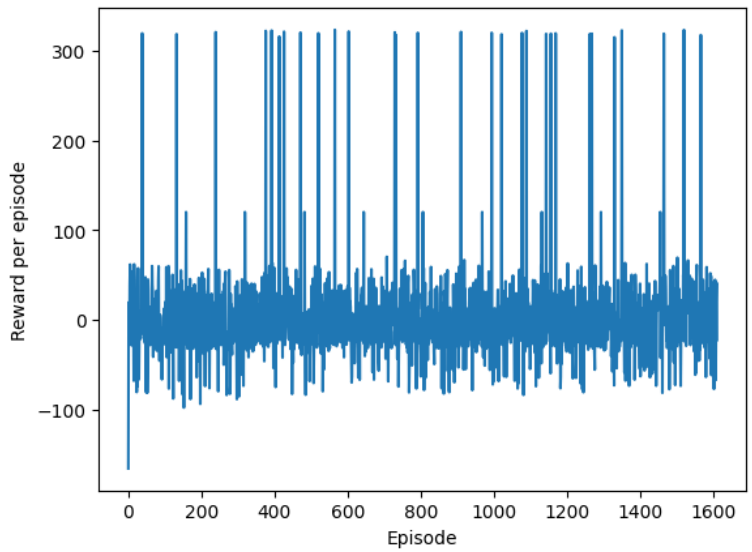


(b) Reward per episode (100 episodes)

Results and Discussion: TD3



(c) Average Reward (1600 episodes)



(d) Reward per episode (1600 episodes)

Conclusion

- Successfully built an environment from scratch using OpenAI gym
- We set the robot kinematics, control inputs, and rewards.
- Created two versions of the world for both discrete and continuous action spaces.
- Integrated DQN with discrete world, in which the results were promising in terms of convergence.
- Integrated DDPG and TD3 with continuous world, the models reach sub-optimal convergence.
- Evaluation of models was through calculating mean rewards and reward per episode.
- **Future work:** increasing the number of episodes and iteration steps for continuous model.

Thank You for Your Attention!