Final Project Report

CS5180 Reinforcement Learning

Autonomous Robot Navigation using Deep RL

Authors:
Rahul Sha Pathepur Shankar
Vaishnav Raja

Submitted to:
Dr. Robert Platt

Autonomous Robot Navigation using Deep RL

1st Rahul Sha Pathepur Shankar
Master of Science in Robotics
Northeastern University
Boston, USA
pathepurshankar.r@northeastern.edu

2nd Vaishnav Raja
Master of Science in Robotics
Northeastern University
Boston, USA
raja.krv@northeastern.edu

***Abstract*—This final project explores deep reinforcement learning for autonomous robot navigation like 2D environment using the Deep Deterministic Policy Gradient (DDPG) algorithm and Twin Delayed DDPG (TD3) algorithm. The agent is a differential-drive TurtleBot robot is trained in a custom simulator build using pygame to reach a target goal without collisions. We develop a shaped reward function that penalizes collisions and inefficient motion while rewarding progress toward the goal. The DDPG agent uses continuous wheel speed commands as actions and learns via an actor-critic approach. The agent demonstrates improved navigation performance, significantly reducing collision occurrences and increasing its cumulative rewards over time. However, training instabilities in the Q-value estimates were observed, and this was solved by the integration of the Twin Delayed DDPG (TD3) algorithm for improvements over Q-values instabilities. This project highlights the potential of model-free deep reinforcement learning for robot navigation and identifies challenges and future directions for enhancing learning for Autonomous Navigation in Robots.**

I. Introduction

Autonomous navigation is a fundamental problem in robotics, requiring an agent to reach a goal location while avoiding obstacles both static and dynamic. Traditional navigation approaches rely on mapping and global and local planning algorithms, but recent advances in reinforcement learning introduce the ability for robots to learn navigation policies from experience in the simulation. For instance, deep reinforcement learning can handle multi-dimensional inputs and learn complex policies, making it useful for continuous end to end robot control. This project is about applying deep RL to train an agent (Mobile Robot – TurtleBot) to navigate in a custom simulated environment (EscapeRoomEnv). The environment presents an escape room scenario where the robot must find its way to an exit or goal area within a walled space like escaping a room.
We employ the Deep Deterministic Policy Gradient (DDPG) algorithm [1] and Twin Delayed DDPG (TD3) algorithm [2] continuous control task. DDPG is an actor-critic method that has shown success in continuous action domains by learning a deterministic policy with the help of a learned critic Q-function. It is well-suited for our navigation problem where the action space which are left wheel and right wheel velocities is continuous. The goal is to train the robot's policy to maximize cumulative reward by encourage reaching the goal quickly while minimizing collisions and inefficient movements which is the ratio of the shortest path to the longest path taken.

II. Technical Definition

Our problem involves training a differential drive robot (agent) operating in a continuous state-action space.

A. State Space

The agent's state s_t at time step t includes the robot's position (x_t, y_t), orientation θ_t, linear velocities (v_{x_t}, v_{y_t}), and angular velocity ω_t. Also, the state encodes the relative position and orientation towards the goal, specifically the distance to the goal d_t and angle difference $\Delta\theta_t$. Thus, the state vector is the following

$$s_t = [x_t,\ y_t,\ \theta_t,\ v_{x_t},\ v_{y_t},\ \omega_t,\ d_t,\ \Delta\theta_t]$$

B. Action Space

The agent's action a_t is a continuous two-dimensional vector controlling the left and right wheel velocities, normalized to the range $[-1, 1]$. Internally, actions are scaled by the maximum wheel velocity $V_{\max}$ Differential drive kinematics define robot motion equal wheel speeds yield straight-line motion, unequal speeds result in turning, and opposite directions facilitate rotations in place.

$$a_t = (u_L, u_R), \quad u_L, u_R \in [-1, 1]$$

$$v_L = u_L \cdot V_{\max}, \quad v_R = u_R \cdot V_{\max}$$

C. Transition Dynamics

Robot motion follows deterministic kinematic equations for a differential drive system. Given current state s_t and action a_t, the robot's next state s_{t+1} is computed by integrating wheel velocities over a discrete time interval Δt and collisions with walls or boundaries trigger penalties, reverting the robot's state to the last valid configuration.

$$x_{t+1} = x_t + v_t \cos(\theta_t)\Delta t$$

$$y_{t+1} = y_t + v_t \sin(\theta_t)\Delta t$$

$$\theta_{t+1} = \theta_t + \omega_t \Delta t$$

D. Reward Function

The shaped reward r_t guides robot behavior with several following components

- **Step Penalty -** A small negative reward $-\alpha$ per step, discouraging slow navigation
$$r_{\text{step}} = -\alpha$$
- **Distance-based Reward -** Rewards getting closer to the goal and penalizes moving away, scaled logarithmically based on distance improvement $\Delta d = d_{\text{old}} - d_{\text{new}}$
$$r_{\text{distance}} = \begin{cases} +\log(1+\Delta d), & \text{if } \Delta d > 0 \\ -\log(1-\Delta d), & \text{if } \Delta d \leq 0 \end{cases}$$
- **Heading Alignment Penalty -** Penalizes large deviations ($\Delta\theta$) greater than a threshold ($\pi/6$), plus an extra penalty if angular velocity (ω) is high
$$r_{\text{heading}} = \begin{cases} -\log(1+\Delta\theta) - \alpha \cdot \mathbf{1}[\omega > \frac{\pi}{6}], & \text{if } \Delta\theta > \frac{\pi}{6} \\ 0, & \text{otherwise} \end{cases}$$
- **Goal Reward -** Large positive reward upon reaching the goal, including a time-based bonus encouraging quicker completion
$$r_{\text{goal}} = B + \log\left(1 + \frac{\text{max\_steps} - t}{t}\right) \cdot B \cdot \alpha$$
where $B = 10000$.
- **Collision Penalty -** Immediate negative reward for collisions; minor collisions yield small penalties (e.g., -5), boundary or severe collisions incur larger penalties and potentially terminate the episode.

E. Episode Termination Conditions

Episodes terminate under the following conditions

- Goal reached (successful termination).
- Irrecoverable collision or boundary violation (failure termination).
- Maximum allowed steps per episode exceeded (failure termination).

The goal is to learn an optimal policy $\pi^*(s)$ that balances efficient goal-reaching with obstacle avoidance, given the constraints of differential drive dynamics and partial observability.

III. Simulation Design

EscapeRoomEnv is a custom 2D simulation environment built following the OpenAI Gym interface `reset()` and `step()` designed specifically for training differential-drive robot navigation in escape-room scenarios.

A. Environment Layout

The environment is a bounded 2D rectangular plane with dimensions 600×500 units. Boundaries act as solid walls, and this internal walls create obstacles. Walls are represented as rectangles or line segments.

B. Robot Model

The robot, modeled as a differential-drive TurtleBot, is characterized by a circular body and pose (x, y, θ), indicating its position and orientation. The robot's wheel velocities (v_L, v_R) are directly controlled through agent actions. At episode start, the robot initializes at a predefined location and orientation, which near one corner, with its size allowing free passage through openings. Robot kinematics for position updates per timestep Δt
$$x_{t+1} = x_t + v_t \cos(\theta_t)\Delta t,$$
$$y_{t+1} = y_t + v_t \sin(\theta_t)\Delta t,$$
$$\theta_{t+1} = \theta_t + \omega_t \Delta t.$$

C. Goal

The robot aims to reach a designated goal position represented as a circular region with a fixed radius. The goal's position is fixed for training consistency but could be randomized for generalization.

D. Observation Space

The observation state provided to the agent consists of two key components

- **Distance to goal:**
$$d_{\text{goal}} = \|\text{goal.position} - \text{robot.position}\|$$
- **Relative angle to goal:**
$$\Delta\theta = \text{normalize}(\theta_{\text{robot}} - \theta_{\text{goal}})$$
where $\Delta\theta \in [-\pi, \pi]$, calculated via
$$\theta_{\text{goal}} = \text{atan2}(y_{\text{goal}} - y_{\text{robot}},\ x_{\text{goal}} - x_{\text{robot}})$$

E. Action Space

Actions are continuous two-dimensional vectors (u_L, u_R), normalized to $[-1, 1]$ and scaled to the robot's maximum wheel velocity V_{max}
$$v_L = u_L \cdot V_{\text{max}}, \quad v_R = u_R \cdot V_{\text{max}}$$
Examples of actions are the following

- Forward motion: $(1.0, 1.0)$
- In-place rotation: $(1.0, -1.0)$

F. Physics and Collision Handling

Each simulation step (Δt) updates robot position and orientation according to differential-drive kinematics. Collision checking occurs immediately after state updates. Collision logic includes the following

- Compute tentative next position (x', y').
- Check boundary (out-of-bounds) and internal obstacle collisions (circle-rectangle intersection tests).
- Assign penalties:
  - Boundary collision: $p = -10$, flag $O =$ True
  - Obstacle collision: $p = -5$, flag $O =$ False
- If a collision occurs, revert robot position to the previous safe state.

G. Reset Function

At env.reset(), the robot and environment are reinitialized

- Robot is positioned at initial location with zero velocity.
- Goal and walls positions remain constant.
- Internal state (e.g., step count $t = 0$, distance to goal D_{old}) is reset.
- Returns the initial observation (distance and angle to goal).

H. Rendering

An optional render function provides real-time visualization of the robot, goal, and walls for debugging and demonstration purposes.

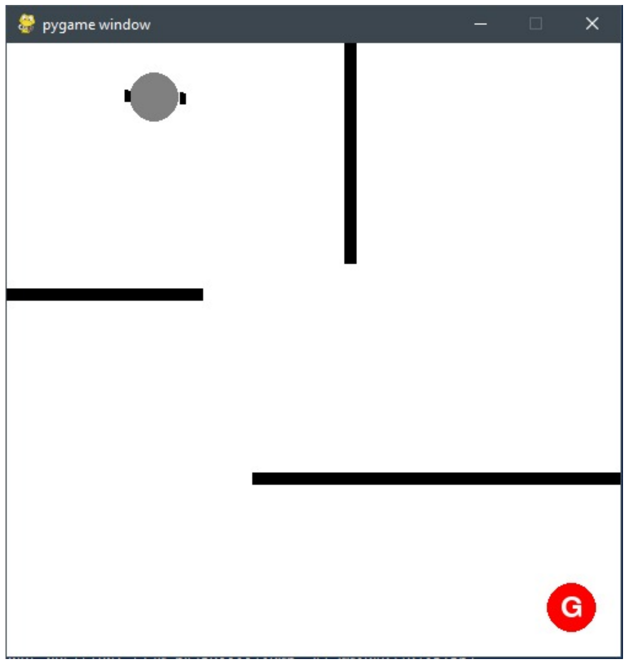

Fig. 1. Rendering of the Environment

IV. RL Algorithms

We utilize two reinforcement learning algorithms *Deep Deterministic Policy Gradient (DDPG)* and its improved variant *Twin Delayed Deep Deterministic Policy Gradient (TD3)* due to their effectiveness in continuous-action tasks like robot navigation.

A. Deep Deterministic Policy Gradient (DDPG)

DDPG is an actor-critic method designed for continuous action spaces. It involves two neural networks:

- **Actor network** $\mu(s|\theta^{\mu})$, outputs deterministic actions given state s.
- **Critic network** $Q(s,a|\theta^{Q})$, estimates expected cumulative rewards (Q-values) given state-action pairs.

In our implementation, both networks are feed-forward with ReLU activations. The actor outputs actions in the range $[-1,1]$ (scaled by V_{max}).

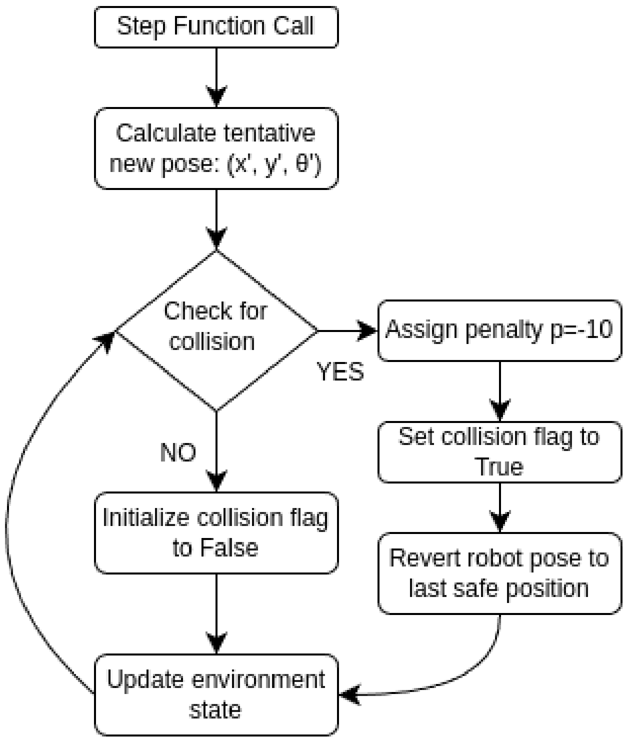
Step Function Call
Calculate tentative new pose: (x', y', θ')
Check for collision
YES
Assign penalty p=-10
Set collision flag to True
Revert robot pose to last safe position
NO
Initialize collision flag to False
Update environment state

Fig. 2. Flowchart of the robot's movement update and collision checking logic

- **Actor network structure:**

$$\mu(s|\theta^{\mu}) : s \rightarrow [64N] \rightarrow [64N] \rightarrow \tanh(\text{output})$$

- **Critic network structure:**

$$Q(s,a|\theta^{Q}) : [s,a] \rightarrow [64N] \rightarrow [64N] \rightarrow [\text{output Q-value}]$$

Training Procedure: We employ experience replay and soft target networks ($\mu_{\text{targ}}, Q_{\text{targ}}$) to stabilize training. Each training episode follows:

1) **Action selection with exploration noise with OU process**

$$a_t^{\text{exec}} = \mu(s_t|\theta^{\mu}) + \mathcal{N}_t, \quad \mathcal{N}_t \sim \text{OU}(\theta = 0.15, \sigma = 0.2)$$

2) **Environment interaction -** Execute a_t^{exec}, obtain new state s_{t+1}, reward r_t, and done flag d_t.
3) **Store transition -** $(s_t, a_t, r_t, s_{t+1}, d_t)$ in replay buffer.
4) **Critic update -** Minimize MSE loss for batch of transitions

$$y_i = r_i + \gamma Q_{\text{targ}}(s_i', \mu_{\text{targ}}(s_i')) \cdot (1 - d_i)$$

Critic loss

$$L_Q(\theta^Q) = \frac{1}{N}\sum_i \left(Q(s_i, a_i|\theta^Q) - y_i\right)^2$$

5) **Actor update -** Maximize expected Q-value by minimizing actor loss

$$L_{\mu}(\theta^{\mu}) = -\frac{1}{N}\sum_i Q(s_i, \mu(s_i|\theta^{\mu})|\theta^Q)$$

6) **Soft target updates**

$$\theta^{\mu}_{\text{targ}} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu}_{\text{targ}}$$

$$\theta^{Q}_{\text{targ}} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q}_{\text{targ}}$$

We set $\tau = 0.001$, discount factor $\gamma = 0.99$, actor learning rate 10^{-4}, critic learning rate 10^{-3}, and batch size 64. Inputs are normalized for stability.

B. Twin Delayed Deep Deterministic Policy Gradient (TD3)

TD3 improves upon DDPG by addressing overestimation and instability through three primary techniques:

- **Clipped Double Q-learning -** Maintains two critic networks Q_1, Q_2, using the minimum to compute targets

$$y_i = r_i + \gamma \min_{j=1,2} Q_{\text{targ},j}(s'_i, \mu_{\text{targ}}(s'_i))$$

  Critic loss

$$L_Q(\theta^{Q_j}) = \frac{1}{N}\sum_i \left(Q_j(s_i, a_i|\theta^{Q_j}) - y_i\right)^2, \quad j = 1,2$$

- **Delayed policy updates -** Actor and target networks updated less frequently (e.g., once per two critic updates), ensuring stable Q-value estimates.
- **Target policy smoothing -** Adds clipped noise to actions during target calculation, regularizing Q-estimates:

$$a' = \text{clip}(\mu_{\text{targ}}(s') + \epsilon, -1, 1), \quad \epsilon \sim \mathcal{N}(0,\sigma), \quad |\epsilon| \le c$$

  TD3's target update becomes

$$y = r + \gamma \min_{j=1,2} Q_{\text{targ},j}(s', a')$$

C. Training Tools

Training was conducted using our custom *EscapeRoomEnv* with PyTorch implementations of DDPG and planned TD3 methods. Training duration was about 1500 episodes, taking a few hours on Nvidia RTX4070 hardware. Metrics tracked include episode rewards, actor loss, and critic loss, providing clear indications of learning progression.

DDPG established baseline agent performance. TD3's integration is to mitigate observed DDPG instabilities, enhancing the robustness and effectiveness of the trained navigation policy.

V. Empirical Results

Analysis of DDPG

We trained the DDPG agent in the *EscapeRoomEnv* for 1500 episodes, evaluating its performance via cumulative rewards (episode scores) and actor-critic losses. We present key observations below.

DDPG - Episode Reward Analysis

The primary metric of success is the cumulative reward per episode, defined as:

$$R_{\text{episode}} = \sum_{t=0}^{T} r_t$$

- Initially (episodes 0–200), scores were substantially negative (-8000 to -10000), indicating frequent collisions and ineffective navigation.
- Mid-training (episodes $\sim$500), performance improved significantly (scores between -2000 and 0), showing decreased collisions and improved navigation toward the goal.
- Late training (episodes $\sim$1200–1500), scores plateaued near 0, occasionally dipping to large negative values (around -6000), suggesting persistent instability and incomplete task learning. Importantly, the goal reward (+10000) was rarely achieved, indicating partial but not complete convergence.

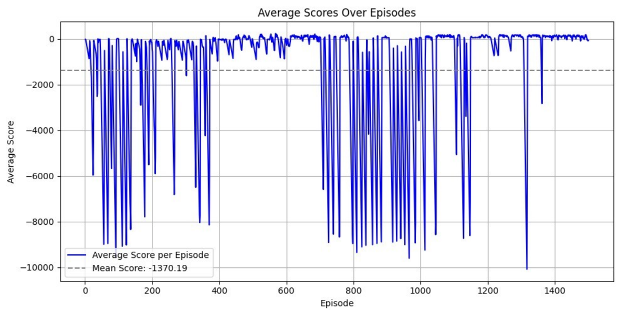

Fig. 3. Episode rewards over the course of training

DDPG - Actor Loss Analysis

The actor network loss (L_{actor}) is defined as the negative mean Q-value predicted by the critic for the actions chosen by the actor:

$$L_{\text{actor}}(\theta^{\mu}) = -\frac{1}{N}\sum_{i=1}^{N} Q(s_i, \mu(s_i|\theta^{\mu})|\theta^{Q})$$

- The actor loss smoothly decreased throughout training from approximately 0 to around -1700 by the final episodes.
- The steady decrease indicates improved action selection, as lower (more negative) actor loss implies higher predicted Q-values, suggesting the actor was effectively optimizing actions toward higher returns.

DDPG - Critic Loss Analysis

The critic loss (L_{critic}) measures the mean squared error (MSE) between predicted Q-values and target values:

$$L_{\text{critic}}(\theta^{Q}) = \frac{1}{N}\sum_{i=1}^{N}\left(Q(s_i, a_i|\theta^{Q}) - y_i\right)^2$$

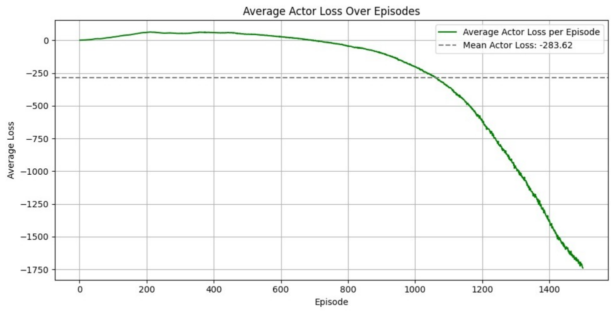

Fig. 4. Actor loss (policy loss) over episodes

where the target y_i is

$$y_i = r_i + \gamma Q_{\text{targ}}(s'_i, \mu_{\text{targ}}(s'_i))(1 - d_i)$$

- The critic loss showed significant volatility and increased dramatically during training, reaching very high values (exceeding 50,000), signaling instability and difficulty approximating accurate Q-values.
- Such instability likely arose due to DDPG's inherent issues like overestimation of Q-values, exacerbated by large and sparse goal rewards.

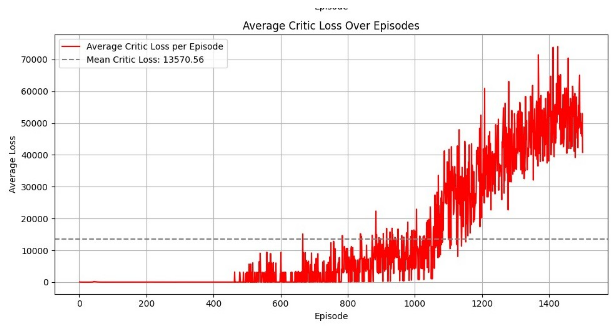

Fig. 5. Actor loss (policy loss) over episodes

DDPG - Qualitative Observations

- Early episodes - The robot exhibited random and ineffective behaviors, frequently colliding or spinning aimlessly.
- Post-training behavior - Improved navigation with reduced collisions, cautious maneuvers near obstacles, and active search behaviors around obstructions.
- Challenges Without explicit obstacle sensing, the robot relied solely on collision feedback and reward signals, resulting in occasional local minima traps and inefficient obstacle circumvention strategies.

Analysis of TD3

We trained the Twin Delayed Deep Deterministic Policy Gradient (TD3) agent for a total of 5000 episodes in our custom *EscapeRoomEnv*. During training, we monitored three critical metrics: episode scores (cumulative rewards), critic loss, and actor loss. The results, shown in Figure 1, illustrate significant learning trends and some notable instabilities.

TD3 - Episode Reward Analysis

The episode reward R_{episode} is defined as:

$$R_{\text{episode}} = \sum_{t=0}^{T} r_t$$

- Initial Training (Episodes 0–500) The agent begins with highly negative scores (frequently below -5000, with some episodes reaching as low as -14000), indicating frequent and severe collisions with obstacles or boundary walls, and ineffective navigation strategies.
- Intermediate Training (Episodes 500–2000) Episode rewards steadily improved and stabilized around values close to 0, typically ranging from -2000 to slightly below 0. This improvement suggests significant progress in collision avoidance and goal-directed behavior. Although major collisions decreased, the agent still frequently incurred small penalties, indicating partial but incomplete learning.
- Late Training (Episodes 2000–5000) The rewards plateaued near zero but retained high variance. Occasional severe drops (below -6000 and some rare extreme negative spikes around -12000) occurred, suggesting periodic instability or exploratory failures. The mean cumulative reward across the entire training period stabilized at approximately -609.28. Despite these fluctuations, the general trend toward near-zero average scores indicated significant learning and substantial reduction in collisions.

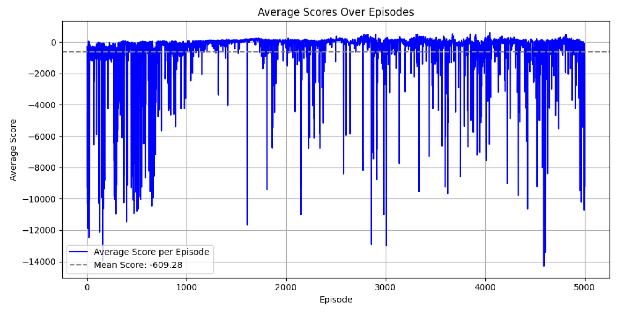

Fig. 6. Actor loss (policy loss) over episodes

TD3 - Critic Loss Analysis

The critic loss L_{critic}, defined by mean squared error (MSE) between predicted and target Q-values, is calculated as

$$L_{\text{critic}}(\theta^Q) = \frac{1}{N}\sum_{i=1}^{N}\left(Q(s_i, a_i|\theta^Q) - y_i\right)^2,$$

$$y_i = r_i + \gamma \min_{j=1,2} Q_{\text{targ},j}(s'_i, a')$$

- Early Training (Episodes 0–1000) initially, the critic loss started low and briefly increased around episodes 200–500, reaching peaks between 500 and 1000. Early instability is typical due to exploration variance, but it subsequently stabilized significantly between episodes

1000–2000, indicating that the critic network learned more accurate Q-value predictions.

- Mid Training Stability (Episodes 1000–3000) between episodes 1000–3000, the critic loss remained relatively stable and low, suggesting accurate Q-function approximation during this period, contributing positively to policy improvement.
- Late Training Instability (Episodes 3000–50000 post episode 3000, critic loss exhibited notable volatility, significantly increasing and regularly spiking, occasionally exceeding values of 2000. These spikes indicate periodic overestimation issues and instability in the critic's Q-value predictions. The overall mean critic loss was approximately 148.57, indicating moderate average critic stability, though late-stage instability poses challenges for consistent learning.

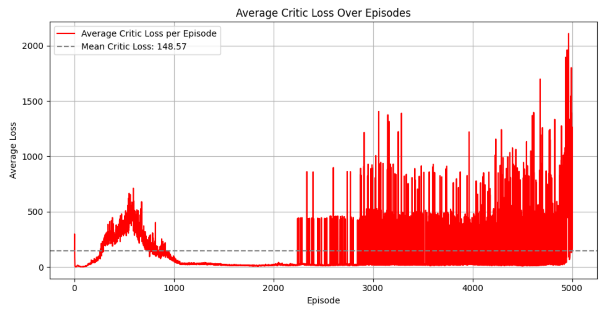

Fig. 7. Actor loss (policy loss) over episodes

TD3 - Actor Loss Analysis

The actor loss L_{actor}, representing the negative mean Q-value of actor-selected actions (thus maximizing Q-value through minimization), is expressed as:

$$L_{\text{actor}}(\theta^{\mu}) = -\frac{1}{N}\sum_{i=1}^{N} Q(s_i, \mu(s_i|\theta^{\mu})|\theta^{Q})$$

- Early Training (Episodes 0–1000) initially, actor loss quickly rose positively from 0 to approximately $+150$, indicating initially poor actions evaluated negatively by the critic (low Q-values). This initial rise was followed by a steady and consistent reduction, indicating the actor rapidly learned to select better-valued actions.
- Mid Training Improvement (Episodes 1000–3000) the actor loss showed steady improvement, crossing below zero around episode 1500 and stabilizing between episodes 2000 and 3000. This clear downward trajectory suggests effective actor learning, with increasingly favorable Q-value evaluations for its actions.
- Late Training Stability and Improvement (Episodes 3000–5000) despite critic volatility during later episodes, the actor maintained relatively stable and gradually improving loss values. Towards the end of training (episode 5000), the actor loss significantly dipped below -100, indicating that the actor continued to identify and choose actions that the critic deemed highly valuable. The overall mean actor loss across training was approximately -34.11, reflecting consistent improvement in policy quality.

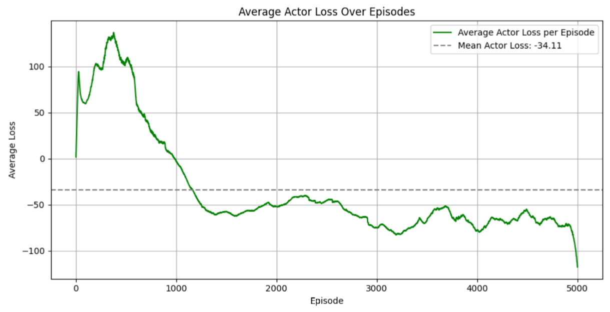

Fig. 8. Actor loss (policy loss) over episodes

TD3 - Qualitative Observations

- Improved Collision Avoidance - The TD3 agent significantly improved collision avoidance compared to initial performance, demonstrating learned caution and better trajectory planning, even without explicit obstacle sensors.
- Cautious but Incomplete Goal reaching - Although consistently avoiding severe collisions by late-stage training, the agent still occasionally exhibited ineffective exploration strategies, resulting in near-goal hesitation and missed goal completions. The agent generally reached vicinity regions but rarely fully accomplished goal objectives.
- Critic-induced Policy Variability - Periodic critic loss spikes correlated with episodes of drastically lower rewards, suggesting that critic instability directly influenced temporary policy degradation episodes. Yet, the actor consistently recovered from these episodes, indicating resilience in learned behavior.

Summary of Findings

While the DDPG algorithm demonstrated notable improvement in navigational performance (fewer collisions, higher rewards over time), it exhibited significant instability in value-function estimation. Actor performance steadily improved, while critic predictions diverged, a common limitation of vanilla DDPG.

The empirical TD3 results demonstrate significant improvements over initial DDPG performance.TD3 substantially stabilized actor learning, leading to consistently better action selection and smoother policy behavior. However, despite notable improvement, the critic network showed periodic instability especially late in training—reflecting inherent challenges in accurately approximating highly variable and sparse rewards in the environment.

VI. Conclusion and Future Work

This project explored the use of deep reinforcement learning—specifically DDPG and TD3 for training a differential-drive robot to navigate an escape-room-style environment. Both algorithms, particularly TD3, enabled the agent to significantly reduce collisions and demonstrate goal-seeking behavior through smooth, continuous control. Actor-critic architectures, paired with shaped rewards, facilitated optimal policy learning over thousands of episodes. The agent did not consistently reach the goal. Limitations such as sparse rewards, partial observability, and critic instability constrained optimal policy convergence. To address these challenges and further enhance performance, we propose the following future directions:

- **Curriculum Learning -** Gradually increase environment complexity, beginning with simple layouts and introducing obstacles incrementally. This approach can help the agent build foundational skills before tackling harder tasks.
- **Hindsight Experience Replay (HER) -** Apply HER to DDPG and TD3 to combat sparse reward issues. Reframing failed episodes as successful with alternative goals offers richer learning signals and encourages exploration.
- **Reward Reshaping -** Refine the reward structure by incorporating intermediate proximity rewards or subgoal bonuses. Potential-based reward shaping can provide learning guidance without altering the optimal policy.
- **Algorithmic Refinements -** Further evaluate DDPG and TD3 while exploring alternatives like Soft Actor-Critic (SAC). Hyperparameter tuning and architecture scaling will also be considered to match environment complexity.

In summary, combining curriculum learning, HER, and improved reward shaping with robust RL algorithms like DDPG and TD3 offers a promising path toward developing generalizable, stable, and real-world-capable navigation agents.

References

[1] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[2] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *Proc. Int. Conf. Machine Learning (ICML)*, 2018, pp. 1587–1596.

[3] M. Shahid, S. N. Khan, F. I. Khawaja, S. Ali, and Y. Ayaz, "Dynamic goal tracking for differential drive robot using deep reinforcement learning," *Sensors*, vol. 21, no. 23, p. 8036, 2021.

[4] L. Butyrev, T. Edelhaußer, and C. Mutschler, "Deep reinforcement learning for motion planning of mobile robots," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, 2020, pp. 3499–3506.

[5] Vaishnav Raja, Rahul Sha, "RL Escape Room," GitHub repository, Available: https://github.com/VaishnavRaja11/rl-escape-room.