Autonomous Navigation and Obstacle Avoidance Using Deep Reinforcement Learning

AmirHossein Taslimi
Department of Computer Science
Shiraz University

1. Abstract

This project explores the application of reinforcement learning, specifically the REINFORCE algorithm, to solve autonomous navigation and obstacle avoidance tasks using the Pioneer 3-DX robot within the Webots simulation environment. The primary objective was to enable the robot to navigate an unknown environment, reaching a designated goal while avoiding obstacles. Various modifications and enhancements to the REINFORCE algorithm, including the causality trick, discount factor, gradient clipping, baseline adjustment, and curriculum learning, were implemented and evaluated across three experiments. The results demonstrated that while the baseline implementation struggled with high variance and inconsistent learning, the incorporation of gradient clipping and baseline adjustment led to improved stability. The most effective approach was curriculum learning, which significantly improved the agent's ability to reach the goal, though some instability remained. These findings highlight the potential of reinforcement learning in robotic navigation tasks and underscore the importance of refining the training process to achieve optimal performance.

2. INTRODUCTION

2.1 Background and Motivation

MOTION planning for autonomous vehicle functions is a vast and long-researched area using a wide variety of approaches such as different optimization techniques, modern control methods, artificial intelligence, and machine learning.

Using deep neural networks for self-driving cars gives the possibility to develop “end-to-end” solutions where the system operates like a human driver: its inputs are the travel destination, the knowledge about

the road network and various sensor information, and the output is the direct vehicle control commands, e.g., steering, torque, and brake. However, on the one hand, realizing such a scheme is quite complicated, since it needs to handle all layers of the driving task, on the other hand, the system itself behaves like a black box, which raises design and validation problems.

2.2 Objectives

The primary goal of this project is to develop a robust reinforcement learning model that enables the Pioneer-3 robot to navigate autonomously in an unknown environment while effectively avoiding obstacles. By leveraging the REINFORCE algorithm with targeted improvements, the project aims to create an adaptive navigation strategy that allows the robot to learn from its interactions with the environment, optimizing its path to reach predefined goals without collisions. The model is designed to operate in unpredictable environments, where traditional navigation methods that rely on pre-mapped data may fail.

In addition to developing the navigation model, the project also aims to rigorously test and validate the system within the Webots simulation environment. This includes evaluating the robot's performance across various scenarios with different obstacle configurations and assessing the improvements introduced to the basic REINFORCE algorithm. The ultimate objective is to demonstrate that the enhanced algorithm can achieve efficient and reliable autonomous navigation, providing a foundation for future applications in real-world robotic systems.

3. Methodology

3.1 Environment Setup

Webots is an open source and multi-platform desktop application used to simulate robots. It provides a complete development environment to model, program and simulate robots “Figure 3.1.1”.

webots advantages :

Webots was selected for this project primarily due to its user-friendly interface and ease of setup, which distinguish it from other robotics simulators such as Gazebo or CoppeliaSim (formerly V-REP).

Also, webots has a set of features that are well-suited for developing and testing autonomous robotics applications. As a professional-grade robot simulator, Webots offers a realistic 3D environment with precise physics modeling, which is crucial for accurately simulating the dynamics of the Pioneer-3 robot and its interactions with the environment. Additionally, Webots supports a wide range of sensors and actuators, allowing for detailed and accurate simulation of the robot's sensory inputs and movement capabilities. The platform's compatibility with various programming languages and its robust API further facilitate the integration of custom reinforcement learning algorithms, making it an ideal choice for this project.

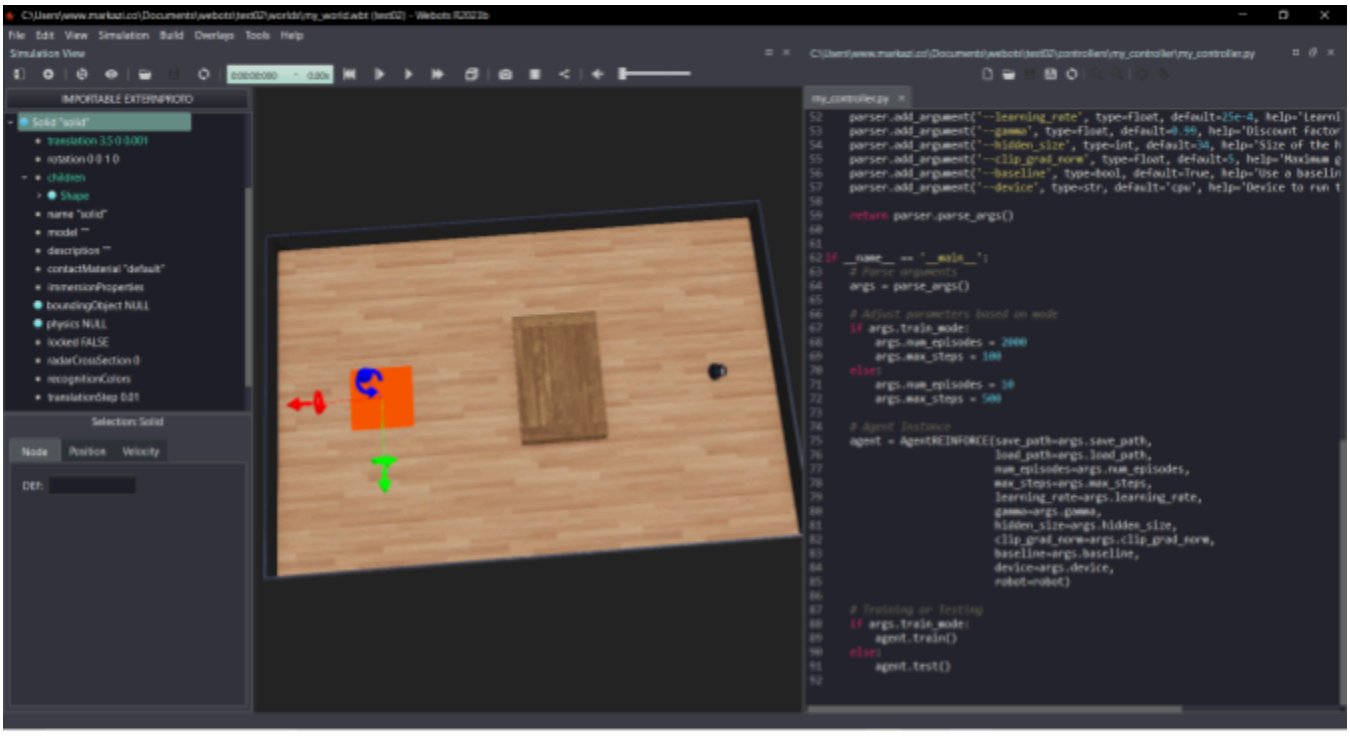


Figure 3.1.1 Webots Environment

For this project, the Pioneer 3-DX robot was selected, a versatile wheeled robot equipped with differential drive, making it well-suited for precise maneuvering in complex environments. The robot is outfitted with a GPS module that provides real-time coordinates, crucial for calculating the robot's distance to the goal within the simulation environment. Additionally, the Pioneer 3-DX is equipped with 16 sonar sensors distributed around its body, allowing for accurate distance measurements to nearby obstacles, which is essential for effective obstacle avoidance.

To further enhance its sensory capabilities, the robot includes a touch sensor that detects physical collisions with obstacles, providing immediate feedback on contact events. This combination of sensors offers a comprehensive view of the robot's surroundings, enabling it to make informed decisions about its movements. The sensor data—comprising GPS coordinates, sonar readings, and collision status—serves as the input (observation) to the neural network, allowing the reinforcement learning

model to learn and adapt to the environment, ultimately guiding the robot toward its goal while avoiding obstacles.

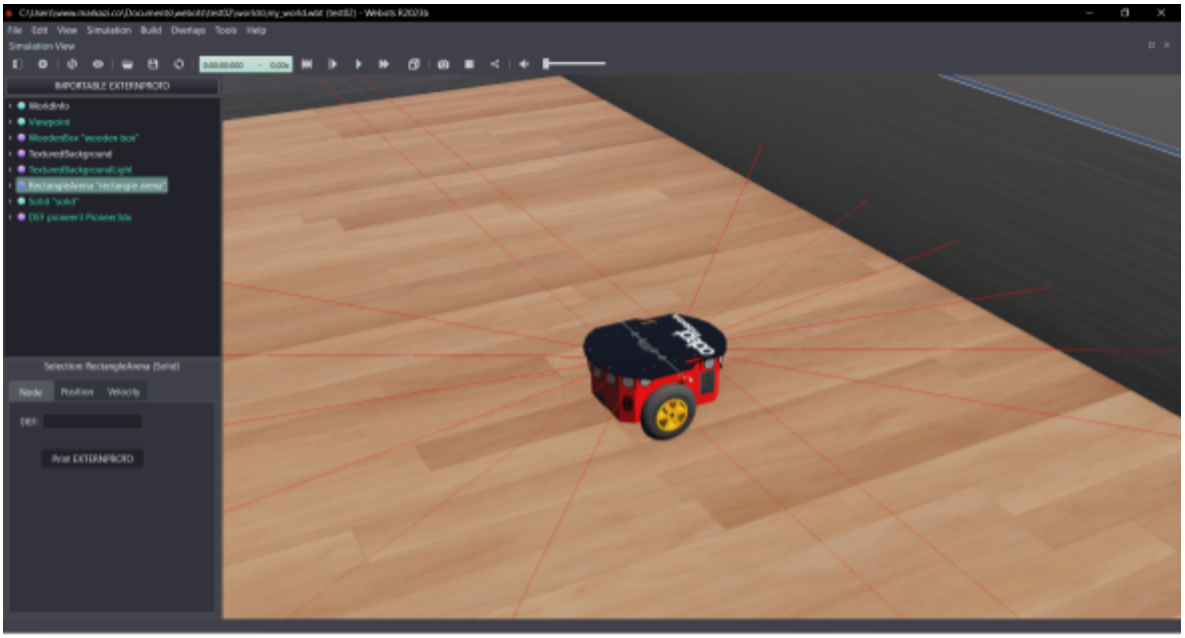


Figure 3.1.2 Pioneer 3-DX

3.2 Algorithm

Policy Gradient Methods in Reinforcement Learning

Policy gradient methods are a class of reinforcement learning (RL) algorithms that directly optimize the policy—the strategy that the agent uses to decide its actions—by adjusting its parameters to maximize cumulative rewards. Unlike value-based methods, which focus on estimating the value function (e.g., Q-learning), policy gradient methods work by representing the policy as a parameterized function (often a neural network) and optimizing these parameters to improve the policy's performance.

In policy gradient methods, the policy is typically represented as a probability distribution over actions given a state, denoted as $\pi\theta(a|s)$, where θ represents the parameters of the policy. The goal is to find the

optimal policy parameters θ * that maximize the expected cumulative reward (or return) over time.

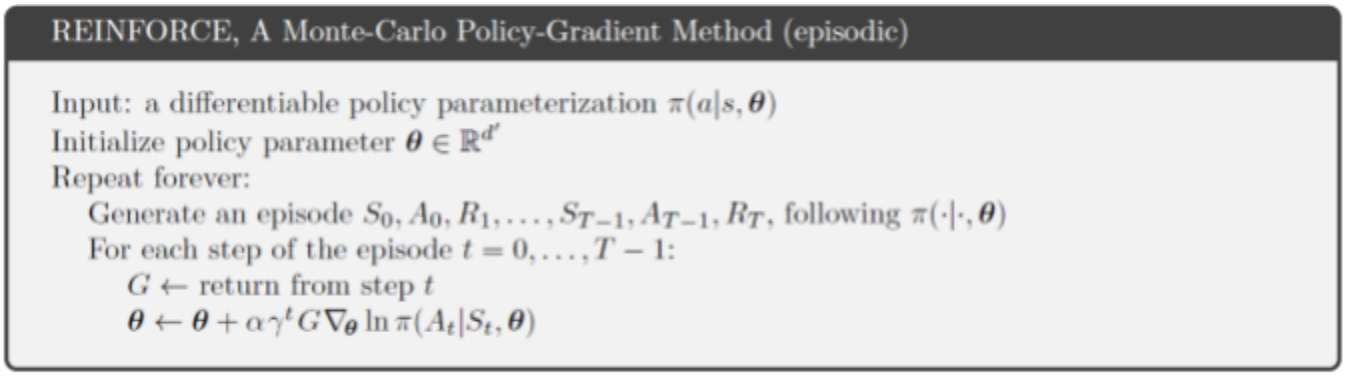
To maximize $J(\theta)$, policy gradient methods compute the gradient of $J(\theta)$ with respect to the policy parameters θ and use gradient ascent to update the parameters iteratively:

$$\theta \leftarrow \theta + \alpha \nabla \theta J(\theta)$$

where α is the learning rate.

REINFORCE Algorithm

The REINFORCE algorithm is a Monte Carlo method, meaning it uses complete episodes of interaction with the environment to update the policy. The key idea behind REINFORCE is to use the rewards obtained in an episode to directly adjust the policy parameters, encouraging actions that lead to higher rewards and discouraging those that lead to lower rewards.


REINFORCE, A Monte-Carlo Policy-Gradient Method (episodic)

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$
Repeat forever:
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
  For each step of the episode $t = 0, \ldots, T-1$:
    $G \leftarrow$ return from step t
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla_{\boldsymbol{\theta}} \ln \pi(A_t|S_t, \boldsymbol{\theta})$

Figure 3.2.1, Reinforcement Learning: An Introduction, Richard S. Sutton and Andrew G. Barto

Limitations of REINFORCE:

- **High Variance**: Since REINFORCE updates the policy based on full returns from episodes, the

gradient estimates can have high variance, leading to unstable learning.

- **Slow Convergence**: The high variance and reliance on complete episodes can result in slow convergence, particularly in environments with delayed rewards.

Improvements of REINFORCE:

Causality

Causality in reinforcement learning refers to the idea that actions taken by an agent should only be credited (or blamed) for future rewards, not past ones. This principle is critical because rewards received after an action are influenced by that action, while rewards received before an action are not. In the context of the REINFORCE algorithm, this means that when updating the policy, only the rewards that occur after an action are considered in calculating the return. This helps ensure that the policy learns the correct relationships between actions and their outcomes, improving the accuracy and effectiveness of the learning process.

Discount Factor

The discount factor determines the importance of future rewards relative to immediate ones. A discount factor between 0 and 1 is applied to future rewards to reduce their impact on the current decision, reflecting the principle that immediate rewards are often more valuable than delayed rewards. In the REINFORCE algorithm, incorporating a discount factor helps to balance the trade-off between short-term and long-term rewards, encouraging the agent to make decisions that maximize cumulative future rewards rather than focusing solely on immediate gains. This leads to more strategic behavior, where the agent learns to plan ahead and consider the long-term consequences of its actions.

Baseline

A baseline is a value subtracted from the expected return (or cumulative reward) in the policy gradient calculation to reduce the variance of the gradient estimates.

The intuition behind subtracting the mean of the rewards (baseline) is to reduce the variance of the policy gradient estimates while retaining the expected value of the update.

By subtracting the mean, the algorithm emphasizes how much better or worse an action performed relative to the average outcome the agent has experienced. If the return is greater than the baseline, the advantage is positive, and the policy is adjusted to increase the likelihood of the action that led to that outcome. Conversely, if the return is less than the baseline, the advantage is negative, and the policy is adjusted to decrease the likelihood of that action.

3.3 Reward Design

Proximity to Goal: The primary reward is based on the robot's distance to the goal. If the robot moves closer to the goal, it receives a positive reward, while moving away results in a penalty. This encourages the robot to consistently move towards the goal.

Distance Thresholds: Additional rewards are provided based on specific distance thresholds. As the robot gets closer to the goal, it receives progressively higher rewards. This tiered approach incentivizes the robot to not just move closer, but to continuously strive for proximity to the goal.

Goal Achievement: A significant reward is given when the robot reaches the goal, signaling successful

task completion. This large positive reward is designed to reinforce the desired behavior of reaching the goal.

Collision Penalty: A substantial penalty is imposed if the robot collides with an obstacle. This discourages reckless behavior and encourages the robot to navigate carefully around obstacles.

Obstacle Proximity Penalty: A smaller penalty is applied if the robot gets too close to obstacles, even without colliding. This prevents the robot from taking risky paths that might lead to collisions.

Alignment with Goal Direction: The robot is rewarded for maintaining a heading that aligns with the direction towards the goal. A small positive reward is given when the robot is well-aligned, and a minor penalty is applied when it deviates from this path. This encourages the robot to not only move towards the goal but to do so in a direct and efficient manner.

4. Implementation

4.1 Policy Network

The policy network in this project is a neural network that serves as the agent's decision-making model. Its primary role is to map the state observations received from the environment to a probability distribution over possible actions. This allows the agent to make decisions based on the learned policy, aiming to maximize cumulative rewards over time.

Network Architecture

The policy network is composed of three fully connected (dense) layers, each contributing to the processing of the input data:

Input Layer:

The network takes an input vector representing the state of the environment, which includes sensor

readings, position data, and other relevant features. The size of this input vector corresponds to the dimensionality of the state space.

Hidden Layers:

The network includes two hidden layers that sequentially transform the input data through linear transformations followed by non-linear activation functions (ReLU).

The first hidden layer processes the input using a set of weights and biases, mapping it to a new representation of the state with a size defined by the hidden_size parameter.

The second hidden layer further refines this representation, expanding it to twice the size of the first hidden layer, allowing the network to capture more complex features and patterns within the input data.

Output Layer:

The final layer of the network reduces the dimensionality of the data to match the number of possible actions the agent can take in the environment.

This layer outputs a vector where each element represents the raw score (logits) for a particular action. To convert these scores into probabilities, a softmax function is applied, ensuring that the sum of the probabilities is 1, and each value lies between 0 and 1.

4.2 REINFORCE Agent

The AgentREINFORCE class initializes several important parameters and components:

Network and Environment: The agent initializes a policy network (PolicyNetwork) that takes the current state as input and outputs a probability distribution over possible actions. The environment instance, created using the Environment class, is responsible for simulating the robot's interactions with its surroundings.

Hyperparameters: Key hyperparameters such as the learning rate (learning_rate), discount factor (gamma), number of episodes (num_episodes), and others are initialized.

Saving and Loading the Model: The agent includes methods for saving and loading the model parameters. This is essential for preserving the best-performing policy and for continuing training from a previous state.

Compute Returns: The compute_returns method calculates the discounted return for each time step in an episode. This involves discounting future rewards by a factor (gamma) and optionally subtracting a baseline (mean reward) to reduce variance and improve learning stability

Compute Loss: The compute_loss function calculates the loss for the REINFORCE algorithm. This loss is based on the log probabilities of the actions taken by the agent during an episode and the corresponding returns, which represent the total discounted future reward.

Key Steps in **compute_loss**:

- Log Probabilities (log_probs): These are the log of the probabilities of the actions that the policy network selected during the episode. The log probabilities are crucial because they allow us to calculate the gradient of the log policy, which is used to update the policy in the direction that increases the probability of actions that lead to higher rewards.
- Returns (returns): This is a tensor representing the total discounted reward that the agent expects to receive from each time step onward in the episode. The returns are pre-computed using the compute_returns function and represent the future reward that the agent is trying to maximize.
- Policy Gradient Loss Calculation: The loss for the REINFORCE algorithm is computed as the negative sum of the product of the log probabilities and the corresponding returns. This is done for each action taken during the episode. The negative sign is used because we want to maximize

the expected return, and in gradient-based optimization, we typically minimize a loss function..

Learn: The learn function orchestrates the process of updating the policy network after each episode. It uses the computed returns and log probabilities to calculate the loss and then applies backpropagation to update the network's parameters.

Key Steps in **learn**:

- Zero Gradients: Before performing the backpropagation, the gradients of the network parameters are reset to zero using self.optimizer.zero_grad(). This is standard practice in PyTorch to prevent gradient accumulation from previous updates.
- Compute Returns: The compute_returns function is called to calculate the returns for the episode, which are needed to compute the loss.
- Compute Loss: The compute_loss function is called with the log probabilities of actions taken and the computed returns. This function returns the total loss for the episode.
- Backpropagation: The .backward() method is called on the loss to compute the gradients of the loss with respect to the network's parameters. This step is crucial as it calculates how much each parameter should be adjusted to reduce the loss.
- Gradient Clipping: The gradients are clipped using torch.nn.utils.clip_grad_norm_ to prevent exploding gradients, which can destabilize the training process. This is particularly important when dealing with deep networks or complex environments.
- Parameter Update: Finally, the optimizer's step() function is called to update the network's parameters based on the computed gradients. This step applies the changes to the network, making it more likely to select actions that lead to higher rewards in the future.

Train: The train method handles the entire training process. It iterates through a predefined number of episodes, collecting rewards and log probabilities of actions taken at each step. After each episode, the agent learns from the accumulated rewards and updates its policy network. The model is saved periodically based on its performance.

5. Experiments and Results

To evaluate the performance of the proposed reinforcement learning algorithm, we conducted a series of experiments in a simulated environment, as illustrated in Figure 5.1. The objective was for the Pioneer 3-DX robot to autonomously navigate from a starting position to an orange rectangular goal while avoiding obstacles and following an optimal path.

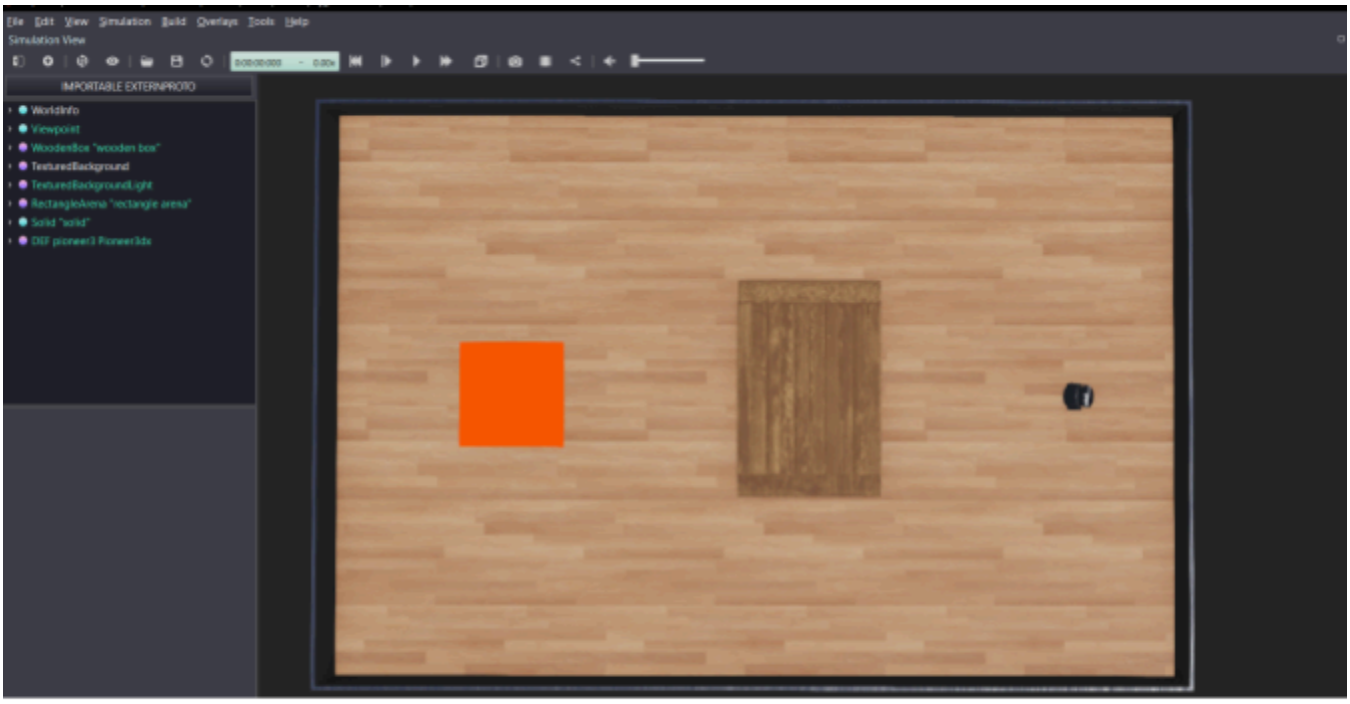


Figure 5.1 Experiment Setup

Hyperparameters :

Number of Episodes	2000
Maximum Steps per Episode	100 (Training Mode), 500 (Default)
Learning Rate	25e-4
Discount Factor (γ)	0.99
First Layer Size	34 neurons
Second Layer Size	64 neurons
Gradient Clipping Norm	5
Robot Speed	3 units

Evaluation Metric

The results of each experiment were analyzed using a Simple Moving Average (SMA) to smooth out the variability in the robot's performance over episodes. The SMA provided a clear view of the agent's learning progress, showing how its ability to navigate the environment improved as it gained more experience.

The SMA plots illustrate the trend in the robot's performance, indicating how quickly the agent was able to learn an effective policy and how consistently it could reach the goal without collisions. The convergence of the SMA curve towards a higher value indicates that the agent's policy became more effective and stable over time.

Experiment 1: Baseline Implementation

In the first experiment (Figure 5.2 SMA), the REINFORCE algorithm was implemented with the causality trick and a discount factor to prioritize more immediate rewards. However, this approach exhibited high variance, and the agent struggled to consistently learn an effective policy. As a result, the agent often failed to reach the goal, and the rewards did not show a significant increase over episodes, indicating that the learning process was not effective in this configuration.

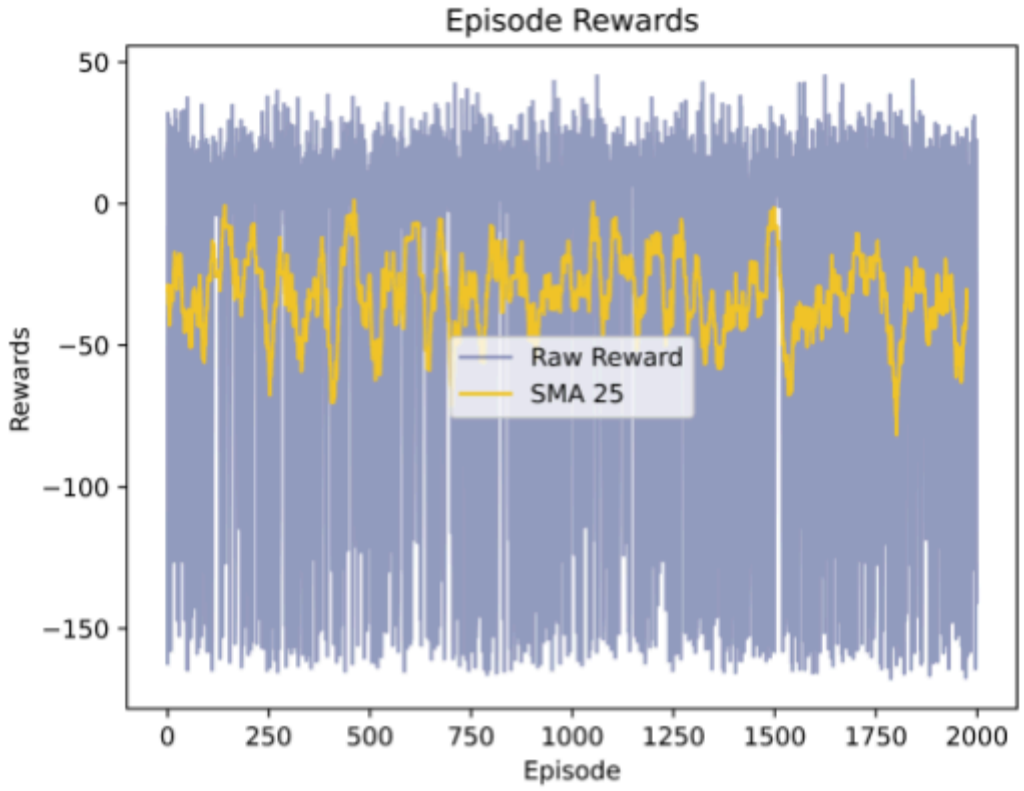


Figure 5.2, Experiment 1

Experiment 2: Gradient Clipping and Baseline Adjustment

To address the high variance observed in the first experiment, we introduced two key modifications in the second experiment (Figure 5.3 SMA). First, gradient clipping was applied with a threshold of 5 to prevent excessively large gradients, which can destabilize the learning process. Second, we added a baseline, computed as the mean of the rewards history, to reduce variance by normalizing the returns. These adjustments led to a lower variance in the results, but the agent still struggled to consistently solve the task and reach the goal, resulting in a relatively low reward return.

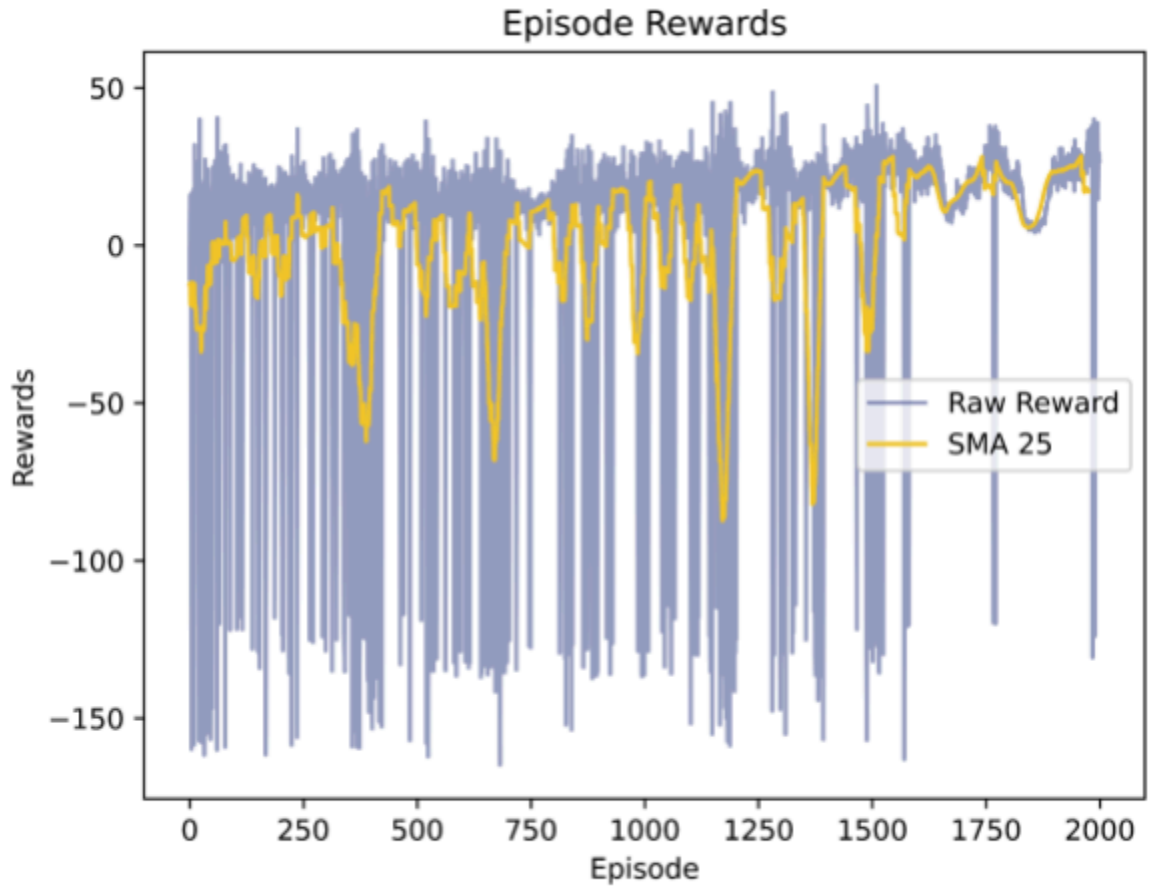


Figure 5.3, experiment 2

Experiment 3: Curriculum Learning

In the third experiment (Figure 5.4 SMA), we employed a curriculum learning strategy to progressively increase the difficulty of the environment. Initially, the agent was trained in an environment without obstacles for the first 500 episodes. Afterward, it was introduced to the primary environment with obstacle. This approach led to a higher average reward, and the agent demonstrated an improved ability to reach the goal. However, the performance occasionally dropped, suggesting that while curriculum learning significantly enhanced the agent's performance, some instability remained in the learning process.

Figure 5.4, experiment 3

6. CONCLUSION

In this project, we successfully applied and evaluated the REINFORCE algorithm for autonomous navigation and obstacle avoidance using the Pioneer 3-DX robot in a simulated environment. The experiments demonstrated the challenges inherent in reinforcement learning, such as high variance and instability in learning. Through a series of refinements, including gradient clipping, baseline adjustments, and curriculum learning, we were able to mitigate some of these issues and significantly improve the robot's performance.

The initial experiments revealed that the agent struggled to consistently learn an effective policy due to high variance, which prohibited its ability to reach the goal. By incorporating gradient clipping and a baseline, the learning process became more stable, though the agent still faced challenges in solving the task. The most promising results were achieved with curriculum learning, where the agent was gradually introduced to more complex environments. This approach enabled the agent to learn more effectively, resulting in higher rewards and improved navigation capabilities, although occasional performance drops highlighted the need for further refinement.

Overall, the findings suggest that while reinforcement learning, particularly the REINFORCE algorithm, holds significant potential for robotic navigation tasks, careful consideration of training strategies and algorithmic adjustments is crucial to achieving consistent and optimal results. Future work could explore additional strategies to further stabilize learning and enhance the robustness of the agent's policy.

7. REFERENCES

[1] Reinforcement Learning: An Introduction Richard S. Sutton and Andrew G. Barto

[2] Survey of Deep Reinforcement Learning for Motion Planning of Autonomous Vehicles University of Maryland at College Park.

[3] Reinforcement Learning: An introduction (Part ¾) (medium)