# Contents

**1 Summary and Introduction** **iii**
1.1 Enhancements to the Env class iii

**2 Model 1: Action Value-Function with Linear Function Approximation with Sarsa and Sarsa** **iv**
2.1 Implementation Details iv
2.2 Reward Function iv
2.3 Parameter Tuning v
2.4 Model Performance v
2.5 Further details vi
2.6 Eligibility Traces vi

**3 Model 2: Policy Gradient with Non-linear Function Approximation Implementation Details, REINFORCE** **vii**
3.1 Implementation Details vii
3.2 Parameter tuning vii
3.3 Model Performance viii

**4 Conclusion** **ix**

# 1 Summary and Introduction

This report details the development and evaluation of two robot navigation models using reinforcement learning (RL). The goal is to navigate a robot in an environment using ROS for real-time interaction. Two models are explored: the first utilizes an action value-function with linear function approximation, and the second employs a policy gradient method with non-linear function approximation. Each model is tuned and assessed based on its ability to effectively navigate while avoiding obstacles, showcasing distinct approaches within the field of RL.

In the project, we employ the actor-critic method to enhance the navigation capabilities of the robot. The actor method implemented actively learns and proposes actions based on the current policy, directly influencing the robot's decisions on movement and strategy. Concurrently, the critic evaluates these actions by estimating their value. Additionally, tile encoding is used to manage the state space effectively. This technique discretizes the continuous state space into a finite number of tiles. By combining tile encoding with the actor-critic framework, we achieve a robust method for encoding complex state spaces.

## 1.1 Enhancements to the Env class

To optimize the interaction with the ROS environment and improve the robot's navigation capabilities, several significant modifications were introduced to the Env class:

- Sensor Resolution: Increased the number of laser scan beams to 60 to enhance environmental perception and detect finer details, which is crucial for navigating through narrow spaces.
- Movement Speed: Reduced the robot's speed to improve its ability to detect walls and avoid collisions, acknowledging a trade-off with increased computational demand.
- State Encoding: Integrated Tile Coding for state representation, crucial for managing continuous state spaces in dynamic environments, especially for neural networks.
- Reward Function: Refined the reward logic to enhance decision-making, incorporating specific adjustments for goal achievement and efficient angular distance calculations to the goals.

# 2 Model 1: Action Value-Function with Linear Function Approximation with Sarsa and Sarsa

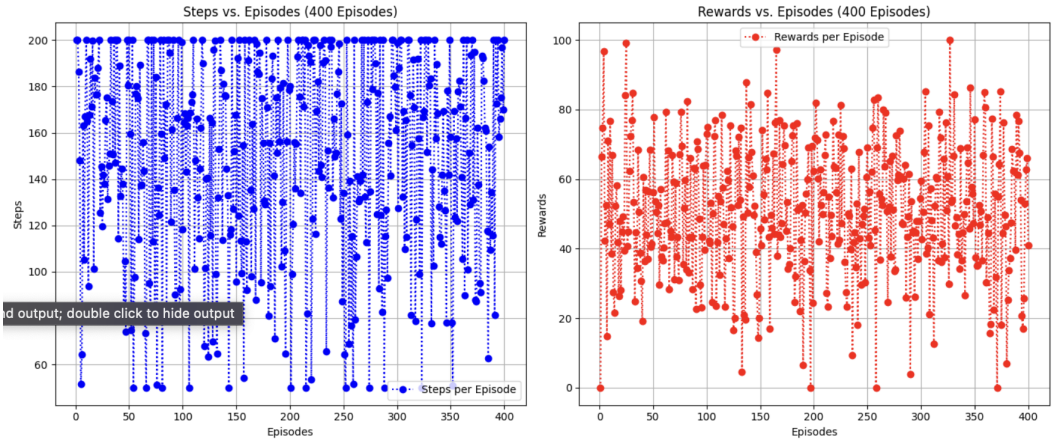
## 2.1 Implementation Details

The first model utilizes the SARSA function from RLv.py, focusing on leveraging immediate feedback from environmental scans.

Key features include:

- State Representation: Employs tile coding to transform sensor inputs into a manageable state representation, enabling effective processing.
- Input Integration: Laser scans to adjust the robot's strategy.
- Action-Value Function: Uses a linear approximation to update value estimates, balancing computational efficiency and navigation performance.

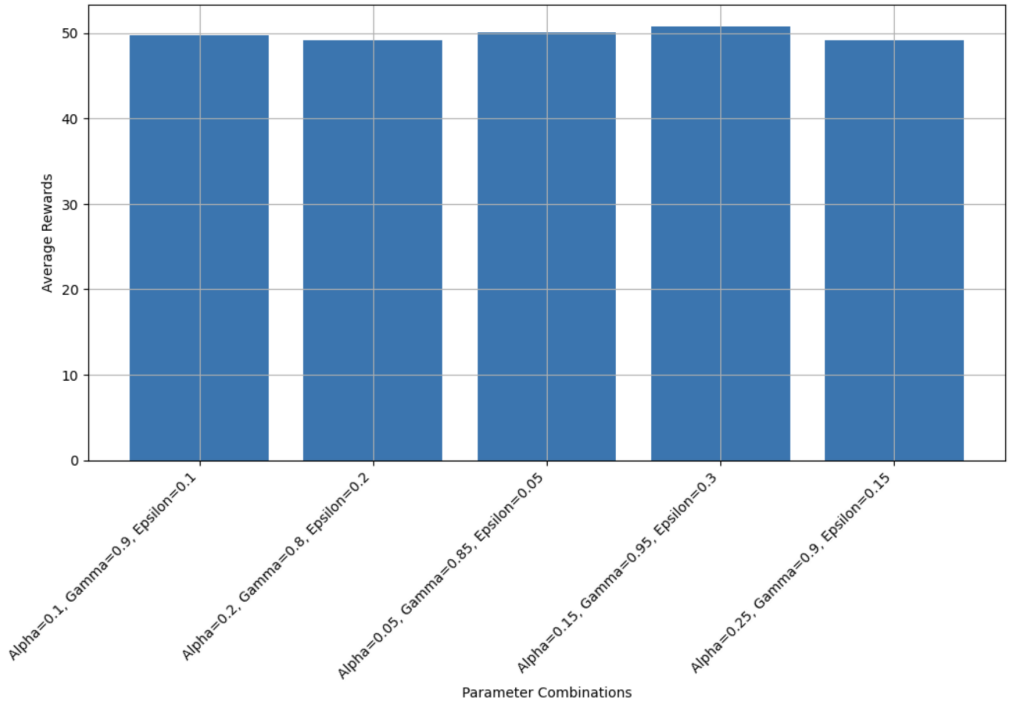
An attempt at using Sarsa was also done. The environment updates its state, which, in this case, consists of updating the laser data. There's a 10% chance (np.random.rand() < 0.1) that the episode ends (done = True). When this happens, a high reward of +400 is given, which could simulate the robot successfully completing a task or reaching a goal. There's another independent 10% chance of receiving a significant penalty (-400), which could simulate the robot encountering hitting a wall. By default, each step incurs a small penalty (reward = -1), encouraging the agent to reach its goal quickly.




## 2.2 Reward Function

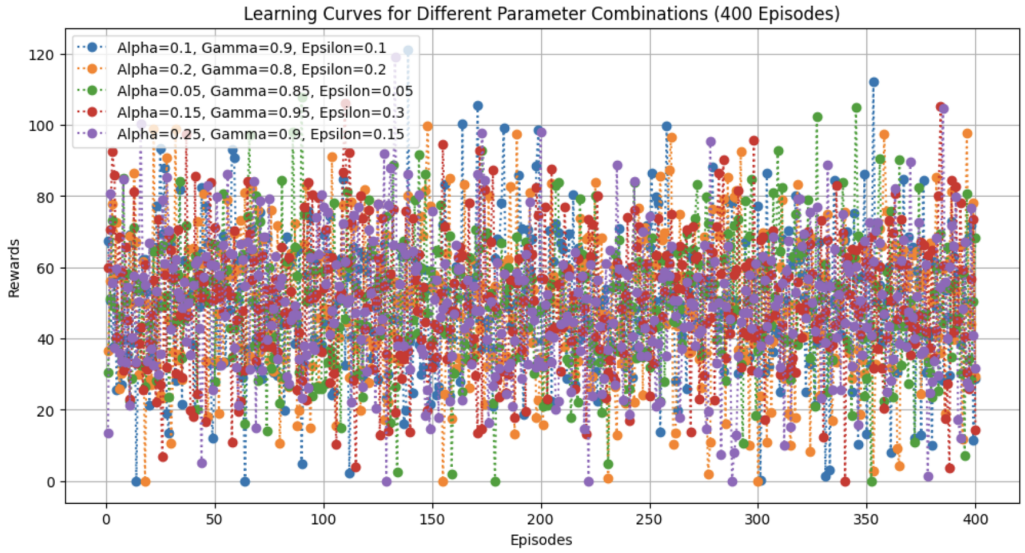
The crafted reward function encourages goal-oriented navigation while penalizing wall collisions, with positive rewards for goal proximity and negative impacts for obstacle contact.

## 2.3 Parameter Tuning




Alpha values seen in the chart such as 0.1, 0.15, 0.2, and 0.25 suggest a range of moderately low to medium settings. In some scenarios, a slightly higher Alpha could be beneficial if combined properly with Gamma and Epsilon. Gamma values like 0.9 and 0.95 in the chart help in promoting a forward-looking policy. The performance across Epsilon settings indicates that a moderate Epsilon might strike the best balance between exploring and exploiting.

## 2.4 Model Performance



## 2.5 Further details

```
actor_input = Input(shape=(env.state_space_size,))
actor_hidden = Dense(24, activation='relu')(actor_input)
actor_output = Dense(env.action_space_size, activation='softmax')(actor_hidden)
self.actor = Model(actor_input, actor_output)

critic_input = Input(shape=(env.state_space_size,))
critic_hidden = Dense(24, activation='relu')(critic_input)
critic_output = Dense(1)(critic_hidden)
self.critic = Model(critic_input, critic_output)
```

## 2.6 Eligibility Traces

Eligibility traces bridge the gap between one-step temporal difference methods like SARSA in this case.They keep track of visited states.

```
for state, action in visited_states:
    eligibility_trace[state][action] *= gamma * lambda_
    update_amount = alpha * td_error * eligibility_trace[state][action]
    Q[state][action] += update_amount
```

# 3 Model 2: Policy Gradient with Non-linear Function Approximation Implementation Details, REINFORCE

## 3.1 Implementation Details

Model 2 leverages a sophisticated policy gradient approach utilizing the RLNN method. This approach optimizes the robot's policy directly from experiences, rather than estimating action values, allowing for more nuanced and complex policy development. The model's key characteristics include:

- Neural Network Architecture: A deep neural network with multiple hidden layers and non-linear activation functions serves as the policy approximator. This network is designed to process and integrate high-dimensional input data from odometry and laser scans effectively.
- Experience-Replay Mechanism: The model includes an experience replay mechanism that stores past transition data. This allows the network to learn from a more diverse set of experiences, improving learning stability and preventing overfitting.

The policy is represented by a neural network (referred to as policy_network) that outputs the probability of taking each action in a given state. The network takes the state of the environment as input and outputs a probability distribution over actions. The last layer of the network uses a softmax activation function to ensure that the output values are valid probabilities. There's an exploration mechanism via an -greedy approach for action selection. The loss for training the policy network is calculated using the policy gradient loss, specifically a form of the REINFORCE algorithm. Gradients of the loss function with respect to the policy parameters are computed and used to update the parameters via backpropagation.

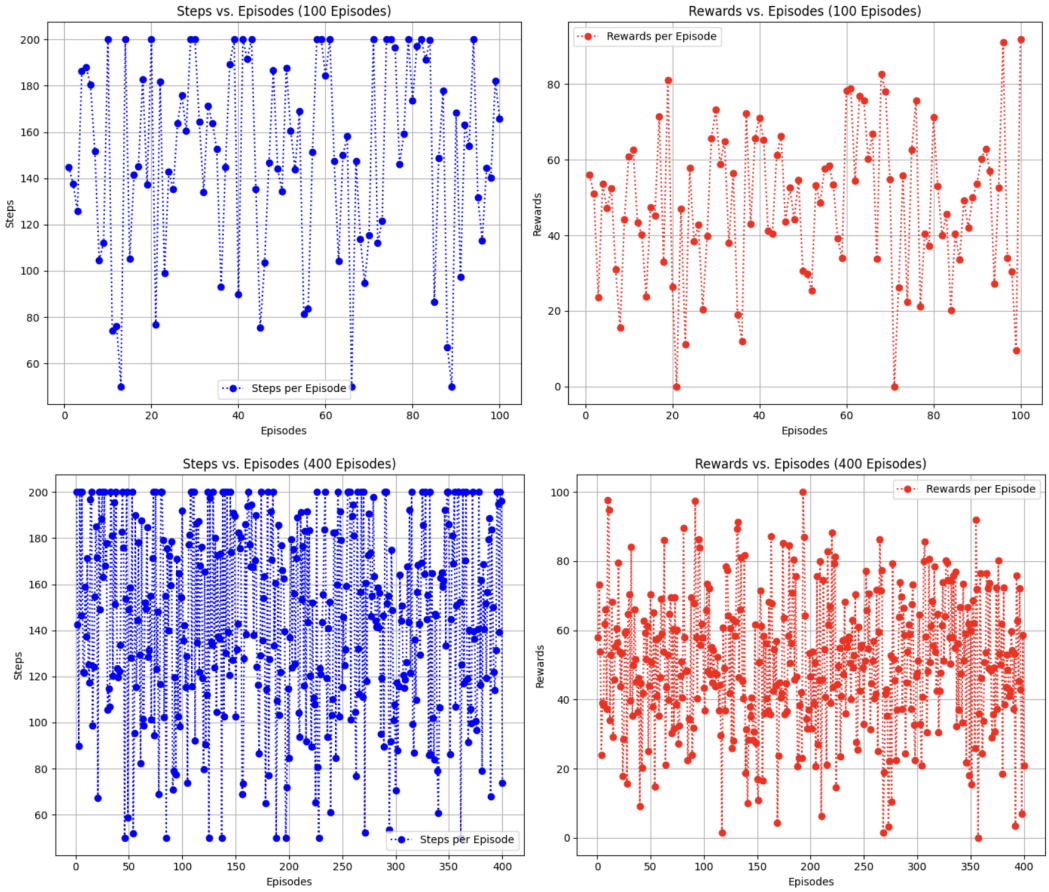
## 3.2 Parameter tuning

Tuning the parameters of a policy gradient model involves a careful balance of several key variables:

- Learning Rate: Controls the speed at which the policy is updated, set to moderate values to ensure stable learning without overshooting the optimal policy. (env_fast = Env(speed= 1., speed=.75*pi, verbose=False) ) , where the speed has been increased for accelerated training.
- Discount Factor (Gamma): Set between 0.9 and 0.99, reflecting the importance of future rewards and influencing the robot's strategic planning horizon.
- Batch Size: Used in updates to balance between learning stability and responsiveness, typically adjusted based on empirical performance and computational limitations.

Key parameters to tune includes the defining of the learning rate which controls the speed of policy updates. It also depicts the moderate value for defining a value of 0.001 that ensures stable learning. Similarly, the gamma has depicted the discount factor which is between 0.9 and 0.99. Moreover, buffer size is defined for experience replay, larger buffers provide more diverse experiences but require more memory. In this way, batch size has determined how many samples are used to update the policy. The range of this value is 32 to 128.

## 3.3 Model Performance




**100 Episodes:** In the rewards chart for 100 episodes, there is significant variability in the rewards per episode. There is no clear upward trend, which could indicate challenges in learning an optimal policy.

**400 Episodes** The rewards per episode show a more condensed range of values compared to the 100 episodes scenario, but still with considerable variability. This could indicate that the model has started to stabilize in terms of the rewards it can achieve.