Deep Reinforcement Learning for Leader-Follower Mobile Robot Navigation with LSTM-Based Obstacle Encoding

Thesis

Submitted in partial fulfillment of the requirements for the degree

Master of Engineering, Mechanical

At

The City College of the City University of New York

By

Ishvar Sitaldin

May 2026

Approved:

Prof. Bo Wang, Thesis Advisor

Prof. Feridun Delale, Chair Dept. of Mechanical Engineering

Deep Reinforcement Learning for Leader-Follower Mobile Robot Navigation with LSTM-Based Obstacle Encoding

Ishvar Sitaldin
Dept. of Mechanical Engineering
Thesis Advisor: Prof. Bo Wang

Abstract

Autonomous mobile robot navigation in dynamic environments requires a control framework that can balance goal-directed motion, obstacle avoidance, and accurate tracking in multi-robot settings. This thesis develops a simulation-based leader-follower navigation framework that combines Soft Actor-Critic (SAC) reinforcement learning with a pre-trained Long Short-Term Memory (LSTM) obstacle encoder. The LSTM encoder converts ordered obstacle-distance information into a fixed-length latent representation, enabling the SAC policy to handle varying obstacle configurations while maintaining a fixed-dimensional observation space.

The proposed framework is developed in three stages. First, Deep Deterministic Policy Gradient (DDPG), Twin Delayed Deep Deterministic Policy Gradient (TD3), and SAC are compared for baseline mobile robot navigation. SAC is selected as the primary controller due to its superior convergence behavior and more consistent performance in randomized simulation trials. Second, an LSTM-based obstacle encoder is trained using a supervised collision-risk classification dataset containing 20,000 randomized obstacle configurations. After training, the classification layers are removed, and the remaining encoder is integrated with the SAC controller as a frozen feature-extraction module. Third, the resulting SAC-LSTM controller is evaluated for leader navigation and leader–follower tracking in environments containing static and dynamic obstacles.

In randomized single-leader single-follower simulations, the integrated controller achieves a 100% navigation and tracking success rate, with the follower maintaining an average tracking error of 0.054 m. These results demonstrate that the proposed SAC-LSTM architecture can support effective obstacle-aware navigation and accurate leader–follower tracking in simulation. A preliminary two-follower triangle formation stress test is also conducted to examine scalability. Although the agents show small post-convergence tracking errors, the test reveals safety and initialization limitations when the framework is directly extended to larger formations. Overall, this thesis establishes a useful foundation for SAC-LSTM-based mobile robot navigation and identifies key directions for future work in robust multi-robot deployment.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and Motivation

In recent years, autonomous mobile robots have become increasingly important across various industrial and commercial sectors. Applications ranging from automated warehouse logistics and delivery systems to search-and-rescue operations rely heavily on the ability of mobile robots to navigate environments safely and efficiently. While a single robot can complete basic tasks, multi-agent robotic systems offer significant advantages in terms of efficiency, redundancy, and scalability [1], [2].

Among the various coordination strategies for multi-agent systems, the leader-follower framework is one of the most widely implemented. In this setup, a designated leader robot is responsible for navigating toward a global goal position, while one or more follower robots track the leader's position to maintain a specific geometric formation.

However, the success of a leader-follower system depends heavily on the real-time decision-making capabilities of both the leader and follower agents. The leader must not only find an efficient path to the target destination but must also adapt the trajectory to ensure safety. The follower must ensure safe path planning in addition to tracking a moving target, the leader. This introduces additional complexity to its control policy, in the form of velocity matching, relative heading matching, and tracking accuracy. In real-world environments, this navigation task is further complicated by the presence of both static and dynamic obstacles, such as human workers, other vehicles, or moving machinery [3], [4]. Developing a robust control framework for the leader agent that balances goal-seeking behavior with real-time obstacle avoidance remains a critical area of research in robotics and control systems

engineering [5].

1.2 Problem Statement

Traditional path planning and obstacle avoidance methods, such as the Artificial Potential Field (APF) method or the Dynamic Window Approach (DWA), have been successfully implemented in static environments. However, these classical approaches often struggle when deployed in complex workspaces containing dynamic obstacles with time-varying trajectories. A common limitation of reactive methods like APF is their susceptibility to getting trapped in local minima, where the attractive force of the goal and the repulsive forces of the obstacles cancel each other out, causing the robot to stall before reaching its destination. Furthermore, these methods typically lack predictive capabilities, treating moving obstacles as static snapshots at each time step.

Deep Reinforcement Learning (DRL) has emerged as a powerful alternative for autonomous navigation. By allowing an agent to learn optimal control policies through trial-and-error interactions with a simulated environment, DRL can find solutions to non-linear control problems without requiring an exact mathematical model of the workspace. Despite their potential, standard continuous action-space DRL algorithms face distinct challenges:

- **Deep Deterministic Policy Gradient (DDPG):** Frequently suffers from critic overestimation bias, leading to unstable training profiles and suboptimal policies that can fail when deployed in unseen environments.
- **Twin Delayed DDPG (TD3):** Addresses overestimation bias using a twin-critic architecture, but can still require extensive training time and fail to explore continuous action spaces efficiently in highly constrained environments.
- **Soft Actor-Critic (SAC):** Utilizes maximum entropy regularization to encourage thorough exploration, but scaling it to handle high-dimensional, time-varying obstacle data directly can destabilize the policy or drastically increase computational overhead.

Additionally, standard DRL architectures often struggle to process the temporal sequences of moving obstacles. Without a mechanism to retain past trajectory data, the agent cannot accurately predict the future positions of dynamic bottlenecks, increasing the risk of collisions in crowded workspaces.

1.3 Research Objectives

The primary objective of this thesis is to design, implement, and evaluate a robust continuous control framework for a leader-follower mobile robot system navigating environments with static and dynamic obstacles. To address the limitations of traditional navigation algorithms, this work presents a decoupled control architecture that combines a recurrent neural network with a continuous deep reinforcement learning policy.

The specific objectives of this research are as follows:

1. **Evaluate traditional DRL methods:** Benchmark the performance of baseline DRL architectures (DDPG, TD3, SAC) across randomized starting configurations to validate their operational efficiency, safety metrics, and success rate.
2. **Develop an LSTM Obstacle Encoder:** Design and pre-train a Long Short-Term Memory (LSTM) neural network to process time-series positional data of static and dynamic obstacles. The network is optimized to classify collision risks and compress that information into a fixed-length vector representation.
3. **Optimize the SAC Leader Agent Framework:** Integrate the trained LSTM encoder into a Soft Actor-Critic reinforcement learning framework. Design an optimized reward function that incorporates goal-seeking incentives, dynamic obstacle penalties, and velocity-damping components to ensure smooth, stable convergence to the target.
4. **Implement Multi-Agent Tracking:** Verify the foundational policy by deploying the leader agent within a complete multi-agent leader-follower control loop, ensuring that the follower agents can safely maintain formation along the generated trajectory.

1.4 Thesis Outline

The remainder of this thesis is organized into the following chapters:

- **Chapter 2: Background and Related Work**, provides an overview of existing research in autonomous mobile robot navigation, classical obstacle avoidance techniques, multi-agent formation control, and the application of deep reinforcement learning in continuous action spaces.
- **Chapter 3: Modeling and SAC-LSTM Control Architecture**, outlines the theoretical foundations of the core components used in this work, including the kinematics of the mobile robot platform, the LSTM network structure, and the mathematical frameworks behind DDPG, TD3, and SAC.
- **Chapter 4: Simulation Setup and Training Implementation**, details the simulation environment setup, the generation of the pre-training dataset for the obstacle encoder, and the design of the state space, action space, and customized reward functions.
- **Chapter 5: Simulation Results and Performance Evaluation**, presents the qualitative and quantitative evaluation of the standalone DRL algorithms, the pre-trained LSTM encoder, and the integrated multi-agent system. This chapter begins by benchmarking baseline DRL architectures to validate the selection of the core controller, followed by performance assessments of the LSTM tracking network and the subsequent trajectory profiles, tracking errors, and constraints observed during leader-follower formation control.
- **Chapter 6: Conclusions, Limitations, and Future Work**, summarizes the major contributions of this research and outlines potential directions for future improvements, including physical hardware deployment and alternative reward formulations.

Chapter 2

Background and Related Work

2.1 Classical Navigation and Obstacle Avoidance Methods

Autonomous mobile robot navigation has been a core research topic in robotics for decades. Early classical methods focused primarily on finding optimal paths in static environments where the workspace geometry is known beforehand. Algorithms such as A* [6] and Dijkstra's algorithm [7] are widely used for global path planning because they guarantee finding the shortest path if one exists. However, these methods are computationally expensive to recalculate in real-time when the environment changes or when obstacles move.

To handle real-time constraints, reactive obstacle avoidance methods were developed. The Artificial Potential Field (APF) method treats the robot as a particle moving under the influence of an attractive force from the goal and repulsive forces from obstacles. While APF is computationally efficient and easy to implement, it suffers from the local minima problem, where forces cancel out and stall the robot before it reaches the goal. Traditional reactive approaches also struggle to handle complex or dynamic workspaces smoothly because they lack predictive capabilities, often treating moving obstacles as static snapshots at each time step [8]. They cannot accurately anticipate where a dynamic bottleneck will occur, leading to jerky corrections or collisions in crowded environments.

2.2 Leader-Follower Formation Control for Multi-Robot Systems

Multi-agent robotic systems extend the capabilities of single robots by allowing multiple platforms to collaborate on a shared task. Research in multi-agent coordination typically focuses on formation control, where the goal is to maintain a specific geometric arrangement among the robots while navigating through a workspace.

The leader-follower framework is one of the most common coordination strategies because of its simplicity and scalability [9]. In this setup, one robot is designated as the leader and is responsible for global path planning and target seeking, while the follower robots track the leader's relative position to maintain formation. This structure significantly reduces the computational overhead on the followers since they do not need to solve the global navigation problem themselves. However, the main vulnerability of this framework is its absolute dependence on the leader agent. If the leader fails to plan a safe path, hits an obstacle, or makes erratic tracking adjustments, the entire formation breaks down. Therefore, ensuring the leader can generate stable and robust trajectories around dynamic obstacles is a critical prerequisite for the whole system.

2.3 Deep Reinforcement Learning in Continuous Control

To overcome the limitations of classical reactive methods, recent research has shifted toward Deep Reinforcement Learning (DRL) for continuous robotic control [10], [11], [12]. Unlike traditional controllers that require precise mathematical models, DRL agents learn optimal policies through trial-and-error interactions to maximize a cumulative reward function [13], [14].

Early continuous control applications relied heavily on the Deep Deterministic Policy Gradient (DDPG) algorithm [15], [16]. However, standard DDPG frequently suffers from critic overestimation bias, leading to unstable training profiles and suboptimal policies that can fail when deployed in unseen navigation environments [17], [18], [19]. To address this, the Twin Delayed DDPG (TD3) algorithm introduced a twin-critic architecture to suppress overestimation bias, which stabilizes training but can still require extensive training time and struggle to explore continuous action spaces efficiently in highly constrained environments [20], [21], [22].

The Soft Actor-Critic (SAC) algorithm resolves these exploration issues by maximizing both

the expected reward and policy entropy [23]. This maximum entropy formulation encourages the agent to explore the continuous action space more thoroughly during training, preventing the policy from getting trapped in suboptimal local strategies. Because of this, SAC has shown excellent sample efficiency, smooth path generation, and strong tracking performance in complex continuous control tasks like autonomous driving and robot navigation [24], [25].

2.4 Temporal Representation and Sequence Modeling in DRL

A major limitation of standard DRL algorithms in dynamic environments is the assumption of the Markov property, which implies that the current state observation contains all necessary information to choose an optimal action. In a workspace with moving obstacles, a single spatial snapshot (like a single sensor scan) only provides the current coordinates of the obstacles. It does not provide their velocities, headings, or moving trends over time.

To capture the time-varying nature of dynamic environments, recent studies have integrated recurrent neural networks, specifically Long Short-Term Memory (LSTM) networks, into the navigation pipeline [26], [27]. LSTMs use hidden state mechanisms to retain information over past sequences, allowing the model to learn the underlying movement profiles of dynamic obstacles.

Instead of training a large end-to-end recurrent DRL agent, which often causes unstable gradients and long training times, a decoupled approach can be used. Pre-training an LSTM network as a dedicated obstacle encoder allows the system to compress temporal tracking data into a fixed-length state representation before passing it to the DRL network. This modular structure significantly improves training efficiency and policy stability when handling moving obstacles [26].

2.5 Summary of Research Gap

While modern DRL architectures like SAC offer excellent continuous control characteristics, directly feeding raw, high-dimensional temporal obstacle sequences into the policy network often leads to computational bottlenecks or training divergence. Most existing literature focuses either on obstacle avoidance using standard DRL or simplifies dynamic scenarios by giving the agent perfect, omniscient knowledge of the obstacles' exact future paths.

There is a distinct gap in research regarding the integration of a decoupled temporal encoder with an entropy-regularized continuous control algorithm specifically optimized for the agents in a multi-agent framework. This thesis addresses this gap by combining a pre-trained LSTM obstacle encoder with an optimized SAC control policy, providing the leader and follower agents with predictive obstacle avoidance capabilities while maintaining a safe navigation policy.

Chapter 3

Modeling and SAC-LSTM Control Architecture

The goal of this research is to develop a robust navigation controller for a differential drive mobile robot capable of obstacle avoidance. This chapter presents the methodology employed to develop and evaluate the reinforcement learning-based navigation system. The discussion begins with the kinematic modeling of the mobile robot, followed by the mathematical formulation of the navigation task as a Markov Decision Process (MDP). Finally, the architecture of the Soft Actor-Critic (SAC) agent, integrated with Long Short-Term Memory (LSTM) layers, is detailed.

3.1 Mobile Robot Kinematic and Dynamic Model

The mobile robot is modeled as a unicycle model for simplicity, as this can be transformed into a differential drive controller afterwards. The Kinematic and Dynamic equations for the unicycle model are as follows:

$$\begin{cases} \dot{x} = v\cos\theta \\ \dot{y} = v\sin\theta \\ \dot{\theta} = \omega \end{cases} \tag{3.1}$$

Where $\dot{x}$ represents the horizontal velocity component and $\dot{y}$ represents the vertical velocity component, both determined by the model orientation, θ, and the model velocity, v. The rate of change of orientation is given by $\dot{\theta}$, which is equal to the angular velocity, ω.

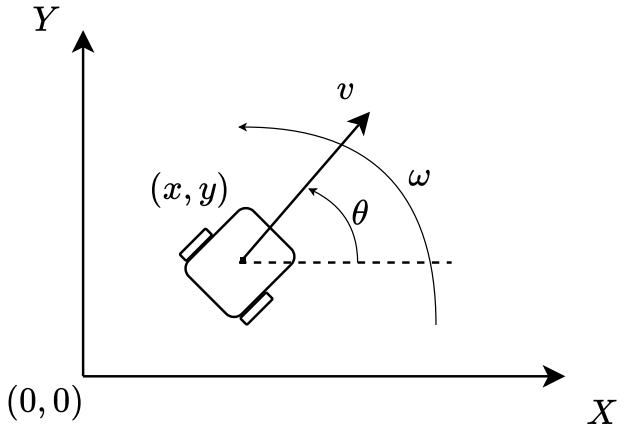

Figure 3.1: *Kinematic representation of the unicycle model, illustrating the state-space variables* (x, y, θ) *and the resultant linear and angular velocities* (v, ω) *in the global reference frame.*

Furthermore, to account for the acceleration limits and the conversion to control inputs u_1 and u_2, the following second-order relationships are utilized:

$$\begin{cases} \dot{v} = u_1 \\ \dot{\omega} = u_2 \end{cases} \tag{3.2}$$

To more accurately simulate existing mobile robots in our laboratory, we opted to implement Force-Balance equations based on the target robot geometry. This will allow us to utilize velocity, v, and angular velocity, ω, as controller inputs, and will also help us to better visualize our controller performance. The equations utilized are shown below.

$$\begin{cases} \dot{v} = \frac{F}{m} \\ \dot{\omega} = \frac{T}{I} \end{cases} \tag{3.3}$$

Where F is the Force applied to the model, T is the applied Torque, both due to the motors on the robot. The mass is given by m, and I represents the robot's inertia.

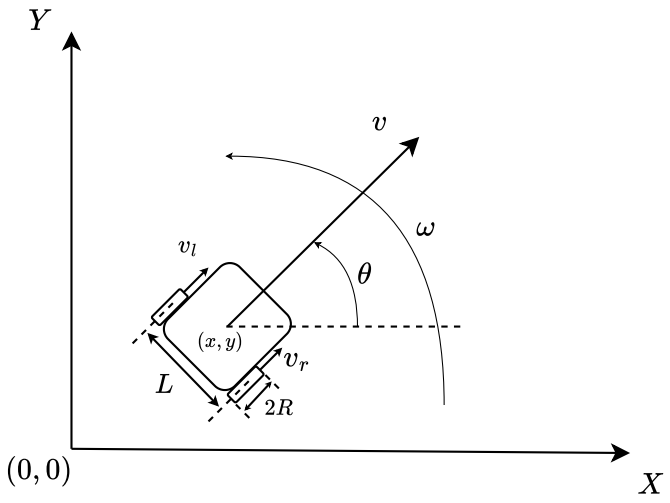

Figure 3.2: *Geometric parameters and velocity components of the differential drive system, defining the track width L and wheel diameter $2R$ in relation to the individual wheel velocities v_l and v_r.*

To transform our results from the unicycle model to the differential drive system, we can use the equations:

$$\begin{cases} v_r = \frac{2v+\omega L}{2R} \\ v_l = \frac{2v-\omega L}{2R} \end{cases} \tag{3.4}$$

However, for the remainder of this project, we will not utilize these transformation equations and will continue to simulate for the unicycle model. In our simulations, the model assumes no-slip conditions, neglects gravity, and assumes a perfectly rigid body.

3.2 Markov Decision Process Formulation

To achieve robust navigation and safe obstacle avoidance, a Deep Reinforcement Learning (DRL) Agent is implemented on the aforementioned unicycle model. The navigation task is formulated as a Markov Decision Process (MDP), which provides a mathematical framework for modeling decision-making in situations where outcomes depend on both the agent's control and environmental transitions. In this context, the robot acts as the agent, interacting with a simulated environment to reach a target coordinate while potentially navigating around static and dynamic obstacles.

The MDP is formally defined by the five-tuple (S, A, P, R, γ), where:

- S_t is the set of states representing the robot's physical configuration.
- A_t is the set of continuous actions.
- $P(s_{t+1}|s_t, a_t)$ represents the state transition dynamics of the unicycle model.
- $R(s_t, a_t)$ is the reward function used to shape the navigation behavior.
- $\gamma \in [0, 1]$ is the discount factor that weighs the importance of future returns.

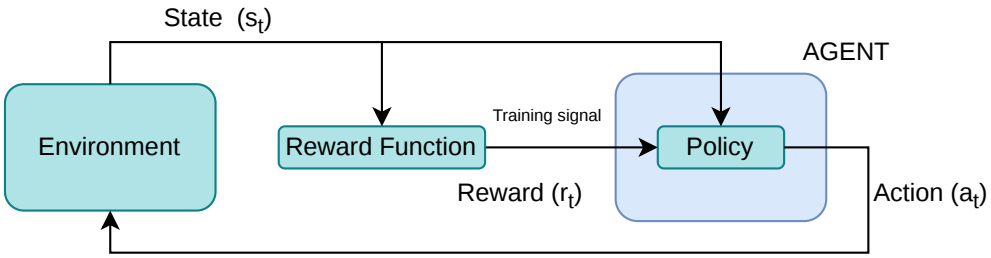

Figure 3.3: *Schematic representation of the Markov Decision Process (MDP).*

Conceptually, the MDP follows a cyclical process as shown in Figure 3.3. The agent observes the current **state** (s_t) from the environment, which serves as the input to the agent's policy to determine the resulting **action** (a_t). This action transitions the environment into a new state (s_{t+1}), and the agent receives a **reward** or **penalty** (r_t) based on the quality of that transition. Through repeated interactions, the agent learns an optimal policy that maximizes the expected cumulative discounted return.

The state space S of the robot is defined by its global position (X, Y), linear and angular velocities (v, ω), and orientation expressed via $\cos\theta$ and $\sin\theta$. Utilizing a trigonometric representation for orientation ensures a continuous input range and prevents calculation errors during the transition between π and $-\pi$ radians.

$$S_t = [X, Y, v, \omega, cos\theta, sin\theta]^T \tag{3.5}$$

While this initial state vector is used for algorithm benchmarking, it is expanded in later chapters with vector $\mathbf{z}$, to include encoded data for obstacle avoidance.

$$S_t = [X, Y, v, \omega, cos\theta, sin\theta, \mathbf{z}^T]^T \tag{3.6}$$

The action space A is continuous, consisting of the normalized Force (F) and Torque (T) defined in Eq. Equation (3.3). Actions are constrained to the range $[-1, 1]$ to ensure consistent gradient updates and to prevent saturation of the actor network's activation functions. The reward function based on these actions and states will be discussed in the following sections.

$$A_t = [F, T]^T \tag{3.7}$$

3.3 Candidate DRL Algorithms for Continuous Control

As we have seen in the previous chapter, we have several options to choose from for our DRL Agent. The agent types we are considering are:

- Deep Deterministic Policy Gradient (DDPG)
- Twin Delayed Deterministic Policy Gradient (TD3)
- Soft Actor-Critic (SAC)

In this section, we will briefly discuss the theoretical details of the three algorithms. Each algorithm was selected for its ability to handle the continuous action spaces for smooth robot control.

3.3.1 Deep Deterministic Policy Gradient

The DDPG algorithm is an off-policy actor-critic method. It utilizes a deterministic policy $\mu(s|\theta^\mu)$ which maps states directly to actions. To ensure exploration in a continuous space, an additive noise process $\mathcal{N}$ (typically Ornstein-Uhlenbeck noise) is added to the action output during training to encourage exploration in continuous action spaces [16]:

$$a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t \tag{3.8}$$

While foundational, DDPG is sensitive to hyperparameter tuning and suffers from overestimation bias in the critic network, which can lead to unstable learning and performance degradation in complex environments. These limitations motivated the development of improved variants such as TD3, which address instability and overestimation issues.

3.3.2 Twin Delayed Deep Deterministic Policy Gradient

To address the instability and overestimation bias observed in DDPG, the TD3 algorithm introduces three key modifications to the actor-critic framework [20]:

- **Clipped Double-Q Learning:** Uses the minimum value of two independent critic networks to reduce overestimation bias in Q-value estimation.
- **Target Policy Smoothing:** Adds clipped noise to target actions to reduce sensitivity to estimation errors and improve policy smoothness.
- **Delayed Policy Updates:** Updates the actor network less frequently than the critic to allow the value function to stabilize before policy optimization.

The target value y in TD3 is calculated as:

$$y = r + \gamma \min(Q'_1(s', a'), Q'_2(s', a')) \tag{3.9}$$

These modifications make TD3 more stable than DDPG and significantly reduce overestimation errors in continuous control tasks.

3.3.3 Soft Actor-Critic

The SAC algorithm represents a shift from deterministic to stochastic policies. It is based on the maximum entropy reinforcement learning framework, where the agent aims to maximize both the expected reward and the entropy of the policy [23]. The objective function J is defined by:

$$J(\pi) = \sum_t \mathbb{E}[r(s_t, a_t) + \alpha \mathcal{H}(\pi(\cdot|s_t))] \tag{3.10}$$

Where $\mathcal{H}$ is the entropy, and α is the temperature parameter controlling the exploration-exploitation trade-off. By maximizing entropy, SAC prevents the agent from prematurely converging to a local optimum and ensures a more robust exploration of the obstacle-filled workspace.

We have seen in [22], where DDPG and TD3 were compared against each other, that TD3 agents consistently outperform DDPG agents at the cost of training time. In that paper, there was no comparison against SAC agents, and given that training time is highly dependent on the simulation environment and the reward structure, we decided to conduct our own experiments comparing these three agents. For these experiments, we utilized the same training environment, including reward structure, for all agent types, only adjusting

agent-specific parameters given the architectural differences.

3.4 LSTM-Based Obstacle Representation

A significant limitation of the implementation of DRL Agents in robotics is that the state-space must be a fixed-sized input to the agent. This can make it more difficult for the agent to deal with complex environments with a varying number of obstacles. If the agent was trained to consider only one or a limited number of obstacles, this would result in a major shortfall when deployed in real-world environments where the number of obstacles at any instance is uncertain. As proposed in [26], implementing a Long Short-Term Memory Recurrent Neural Network, in addition to a DRL Agent, can help prevent this shortcoming.

Recurrent Neural Networks (RNN), like LSTM, are a type of neural network that are very equipped to deal with prediction-based problems and also do not require a fixed number of inputs. These networks are able to output a fixed-sized output vector, containing encoded information on the variable-sized inputs. We utilize this characteristic to have the LSTM encode the varying number of obstacle data into a vector $\mathbf{z} \in \mathbb{R}^{16\times 1}$, which we concatenate with the other state-space parameters, to create the expanded state-space vector S, shown in Equation (3.6), which is fed to the DRL Agent.

By using this method, we ensure that the number of obstacles is not a limitation of our DRL Agent's performance. Just like in [26], we will also implement a safe processing rule to order the obstacle information in descending order, meaning the closest obstacle will be processed last by the LSTM. The reason for this is due to the LSTM being more biased towards the last input, which is a characteristic of RNNs. However, our implementation of the LSTM will differ slightly from [26], given that we are limited in implementing the LSTM layer within MATLAB's actor/critic layers of the DRL agents. In the following sections, we will explain in more detail how we plan to utilize the LSTM as an encoder for our DRL Agents.

3.4.1 Obstacle Encoder Architecture

As we mentioned in the previous section, our plan is to include the LSTM as a supplemental component to our DRL Agent. The LSTM will analyze the obstacles near the agent as shown in Figure 3.4. The input of the LSTM will be a matrix containing distance information about the obstacles. Each obstacle will be represented by a three-element row of matrix $\mathbf{z}$ defined in the robot's local frame.

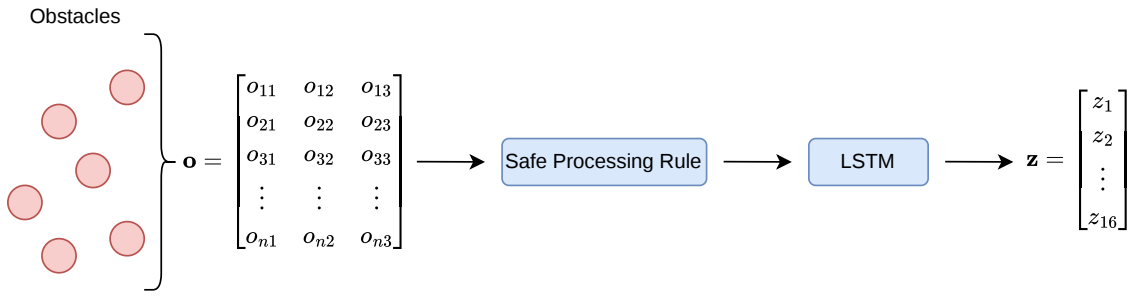

Figure 3.4: *Schematic of the LSTM-based obstacle encoding architecture, transforming a variable-sized input matrix* $\mathbf{o}$ *into a fixed-size latent context vector* $\mathbf{z} \in \mathbb{R}^{16\times 1}$.

The information fed to the LSTM for each obstacle will be the following:

$$\begin{bmatrix} dx_{edge} & dy_{edge} & d_{edge} \end{bmatrix} \tag{3.11}$$

where dx_{edge} and dy_{edge} are the relative horizontal and vertical distances to the obstacle boundary, and d_{edge} is the Euclidean distance to the nearest point on that boundary. This representation mimics the data produced by LiDAR or proximity sensors, enhancing the model's physical realism.

To maintain a constant input size for the LSTM, a maximum obstacle count of $N_{max} = 10$ is defined. The input is structured as a 10×3 matrix $\mathbf{o}$. If fewer than 10 obstacles are present, the remaining rows are padded with dummy values. One might suggest that this implementation does not solve the varying input problem, but in the following section, we will further explain the benefits of this methodology.

3.4.2 Supervised Pre-training and Collision Labeling

One of the drawbacks of DRL Agents is the time necessary for training robust agents. By implementing an LSTM Encoder, we can reduce most of this training time by having the LSTM do the 'heavy-lifting' of this combined controller. Given that the LSTM is capable of outputting a fixed-sized vector, we can use this to our advantage. In theory, once we train a DRL Agent with the combined state-space vector, in our case $\mathbf{S} \in \mathbb{R}^{22\times 1}$, we only need to retrain the LSTM network if we want to increase the number of obstacles the robot should be able to handle.

To train the LSTM and make sure it learns meaningful spatial features, we need to generate a large dataset consisting of random robot poses and obstacle configurations. We then generate labels for this dataset using a simplified motion prediction model. The robot is assumed to maintain a constant linear velocity v and zero angular velocity ω over a finite prediction horizon. A "collision" label ($y = 1$) is assigned if the robot's predicted trajectory intersects with an obstacle's safety radius:

$$d(t) \leq d_{safe} + r_{obs} \tag{3.12}$$

Otherwise, the sample is labeled "safe" ($y = 0$). This created a binary classification dataset representing whether the current obstacle configuration would lead to a future collision under straight-line motion. By training the LSTM to predict future collisions under these conditions, the network learns to extract high-level features related to proximity, obstacle density, and navigable free space.

After supervised training was completed, the classification layers of the LSTM network were removed, and the remaining network was retained as a latent feature encoder. This encoder was then integrated into the Simulink reinforcement learning environment alongside the DRL agent and operated in inference mode during training and deployment.

3.5 Leader–Follower Tracking Formulation

By implementing a Leader-Follower system, we can have multiple DRL Agents work together in a single environment to achieve certain goals, such as collaborative transport or maintaining a specific geometric configuration while traversing cluttered environments. We can use the methods we have previously discussed to develop a Leader-Follower system capable of autonomous navigation with robust obstacle avoidance using the LSTM. To accomplish this goal, we will be using our most efficient DRL Agent and combining it with an LSTM network.

To achieve this, we will need to adjust our reward function for the follower to now be capable of tracking a moving target as opposed to the origin our leader is navigating towards. The most optimal way is by adding more reward components which will be further explained in Section 3.6. In addition to these new components, we must also transform the global error towards this moving goal to the local coordinates of the follower. This allows us to define the follower's goal more clearly and efficiently. To do this transformation and define the moving target for the Follower in its local frame, we will utilize the following steps.

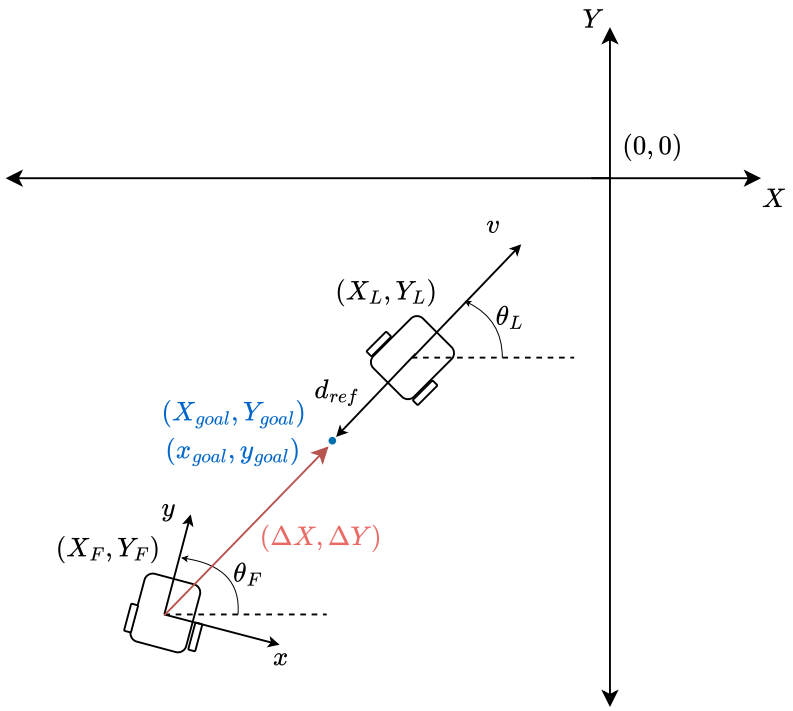

Figure 3.5: *Coordinate transformation schematic for leader-follower formation control, depicting the projection of the global error vector* $(\Delta X, \Delta Y)$ *onto the follower's local reference frame using the rotation matrix* $R(\theta_F)^T$ *to derive the relative tracking states.*

1. **Define Follower Goal Point in the Global Frame:**
   The desired goal position $(X_{\text{goal}}, Y_{\text{goal}})$ is calculated relative to the leader's position (X_L, Y_L) and orientation θ_L using a reference distance d_{ref}. This distance is chosen to have the goal slightly behind the Leader to avoid any collisions:

$$\begin{cases} X_{\text{goal}} = X_L - d_{\text{ref}} \cos(\theta_L) \\ Y_{\text{goal}} = Y_L - d_{\text{ref}} \sin(\theta_L) \end{cases} \tag{3.13}$$

2. **Define Global Error:**
   The error vector in the global frame is the difference between the goal point and the follower's current position (X_F, Y_F):

$$\begin{bmatrix} \Delta X \\ \Delta Y \end{bmatrix} = \begin{bmatrix} X_{\text{goal}} - X_F \\ Y_{\text{goal}} - Y_F \end{bmatrix} \tag{3.14}$$

3. **Transform Global Error to Follower Local Frame:**
   The rotation matrix $R(\theta_F)$ and its transpose (which represents the inverse rotation for orthogonal matrices) are used to map the global error into the follower's local coordinates $(x_{\text{goal}}, y_{\text{goal}})$:

$$R(\theta_F) = \begin{bmatrix} \cos(\theta_F) & -\sin(\theta_F) \\ \sin(\theta_F) & \cos(\theta_F) \end{bmatrix} \tag{3.15}$$

$$\begin{bmatrix} x_{\text{goal}} \\ y_{\text{goal}} \end{bmatrix} = \underbrace{\begin{bmatrix} \cos(\theta_F) & \sin(\theta_F) \\ -\sin(\theta_F) & \cos(\theta_F) \end{bmatrix}}_{R(\theta_F)^T} \begin{bmatrix} \Delta X \\ \Delta Y \end{bmatrix} \tag{3.16}$$

Now we can use these local coordinates of the goal to determine the state-space of the Follower agent. As previously mentioned, we also need to adjust the reward function to include new reward components for velocity and orientation matching. Once this has been done and the DRL Follower Agent has been successfully trained, we can then create a new environment, creating any formation we like, since the follower agent can be utilized for multiple followers in a system.

3.6 Reward Function Design

Designing a reward function to work with the DRL Agents is one of the most important steps when training your agents. Reward functions are 'points' given to the agent based on its actions and the resulting state in the environment. These points are then used by the agent to update its policy to learn what the most rewarding actions are. The points can also be negative, in the form of penalties, if the agent takes an undesirable action. By balancing a reward function with penalties, the agent can learn which actions lead to desirable states and which do not. It is also important to note that these agents can be trained to not prioritize only the reward gained by the next action, but it should also to prioritize the overall reward of a training episode. This is accomplished by implementing a discount factor, $\gamma \in [0, 1]$, as shown in the equation below.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{3.17}$$

If $\gamma = 0$, the agent can be considered short-sighted and only cares about the next reward. In contrast, if $\gamma \to 1$, the agent is far-sighted and the reward after n steps is almost as important as the next reward.

Apart from the discount factor, there are several other aspects to consider when designing a reward function, depending on the goal you want to achieve with the agent. One of these aspects is the difference between **Dense** and **Sparse** rewards. When a goal is defined for an agent, as in our case, reaching the origin $(0, 0)$, a reward structure needs to be in place to guide the agent towards that goal. There are several ways this can be accomplished, but an important consideration must be made regarding dense and sparse Rewards.

Rewards are considered **dense** when the agent is given a reward or penalty at nearly every time-step t. This type of reward is usually calculated based on the agent's current state and progress towards the goal. Conversely, a reward is considered **sparse** when the agent is only rewarded or penalized at a single state. These types of rewards are difficult for agents to learn from as they provide little to no guidance on how to improve. This is why **dense** rewards are considered to be essential for reward function design.

In this project, we have utilized both Dense and Sparse Rewards to fine-tune agent training. Some examples of these rewards are:

- **Euclidean Distance Reward (r_{dist}):** To provide a dense reward signal, the agent is rewarded based on the reduction in Euclidean distance to the goal between successive time steps.

  $$r_{\text{dist}} = d_{t-1} - d_t \tag{3.18}$$

  where d_t is the current distance to the target. This ensures the agent is incentivized to move directly toward the goal at every time step.

- **Approach Velocity Penalty:** A penalty is applied when the robot has a velocity greater than zero when arriving near the goal. This is implemented to encourage the robot to stop when it reaches the goal, and increases as the distance to the goal decreases.

  $$r_v = -v_t^2 \exp(-0.5 d_t^2) \tag{3.19}$$

  where v_t is the velocity of the robot at time step t and d_t is the Euclidean distance to the target.

- **Goal Arrival Reward (r_{reach}):** A large sparse positive reward is granted when the robot enters the predefined tolerance zone. This signal serves as the primary

reinforcement for completing the task.

$$r_{\text{reach}} = +500 \tag{3.20}$$

- **Collision Penalty (r_{coll}):** A significant negative penalty is applied if the robot's physical radius or its Lidar sensors detect an obstacle within a critical safety threshold. This penalty is typically large enough to outweigh any potential distance rewards, teaching the agent that collisions are the least desirable outcome.

$$r_{\text{coll}} = -1000 \tag{3.21}$$

  This component is usually penalized more heavily than reaching the goal, as collision avoidance is weighed more highly by us than simply reaching the goal.

These examples also highlight the other important aspects of Reward Function Design, **Multi-Objective Balancing** and **Smooth Termination Logic**.

Multi-Objective Balancing allows us to define multiple goals for the agent, which can be taught in a single training session, and thus nullifies the need to retrain agents to accomplish these objectives. This is implemented by dividing the Reward Function into several components and assigning weights to signify the importance of each objective. As we have seen in the list above, all of these components are part of the same Reward Function, but each signifies a different goal we are trying to teach the agent.

Smooth Termination Logic allows us to teach the agent to behave in a more natural way, such as slowing down when reaching a positional target or being able to navigate between obstacles without making sudden, sharp movements. All of these aspects are essential when considering Reward Function Design and will be used in the following chapters to design our own functions.

Chapter 4

Simulation Setup and Training Implementation

In this chapter, we will be discussing the implementation of the topics discussed in Chapter 3. We will mention parameters that were used across all simulations, and where necessary, we will note any exceptions to these. Additionally, we will also discuss the implementation of every component and explain the framework for the LSTM Encoder combined with the DRL Agent to create a Leader-Follower system capable of navigation and obstacle avoidance.

4.1 Computational Platform and Software Environment

The training of the Long Short-Term Memory (LSTM) encoder and the reinforcement learning simulations were conducted on a high-performance workstation. The hardware specifications used for all computational tasks are summarized in Table 4.1.

Table 4.1: *Hardware and software configuration for the simulation and training environment.*

Component	**Specification**
Processor (CPU)	Intel Core i9-12900h
Graphics Card (GPU)	NVIDIA GeForce RTX 3070Ti
Memory (RAM)	40 GB DDR5
Operating System	Windows 11

The GPU was primarily utilized for the supervised pre-training of the LSTM feature encoder,

while the CPU handled the kinematic simulation and the calculation of the reward functions at a 10 Hz sampling frequency.

To conduct the training of the LSTM and the reinforcement learning agents, MATLAB 2025a was utilized with the add-on toolboxes:

- Control System Toolbox
- Deep Learning Toolbox
- Parallel Computing Toolbox
- Reinforcement Learning Toolbox
- Simulink

4.2 Simulation Environment and Robot Parameters

The reinforcement learning agents were trained and evaluated in a controlled 2D kinematic simulation. To ensure physical consistency, the robot model was assigned mass and inertia properties typical of a small-scale mobile robot.

4.2.1 Physical Constants

The unicycle model's dynamics are governed by the force-balance equations defined in Section 3.1. The specific mass and inertial constants used across all training episodes are summarized in Table 4.2.

Table 4.2: *Simulation and Physical Parameters*

Parameter	**Symbol**	**Value**	**Unit**
Robot Mass	m	1.0	kg
Moment of Inertia	I	0.005	kg·m^2
Workspace Size	-	20×20	m
Terminal Boundaries	x_{lim}, y_{lim}	± 10.0	m
Simulation Time Step	Δt	0.1	s

Note: For the comparison of the DRL Agent types, a smaller Workspace Size of 5×5 m was utilized for the simulations to save on computing time. Consequently, the Terminal Boundaries also decreased and were ± 5.0 m.

4.2.2 Environment Logic

The workspace is defined as a 20×20 m square with the origin $(0, 0)$ located at the geometric center. Terminal boundaries were established at $|x| \geq 10$ and $|y| \geq 10$. During training, the robot's initial pose (x_0, y_0, θ_0) and the obstacle configurations were randomized at the start of each episode to ensure the agent developed a generalized navigation policy rather than memorizing a specific path. This was implemented by incorporating a custom reset function, which is executed at the start of each episode. This function implements three important stability measures:

- **Boundary-Aware Initialization:**
  While the workspace boundaries are set at ± 10, the robot's initial position is constrained to a 9 m radius. This 1 m buffer ensures that the agent does not trigger a terminal boundary state immediately upon initialization, providing a window for the control policy to stabilize.

- **Zero-Velocity Constraints:**
  All initial translational and angular velocities are set to zero. This static start ensures the agent's performance is a direct result of its learned policy rather than residual momentum from a previous state.

- **Discrete-Time Synchronization:**
  The initial conditions of any implemented unit delay blocks in Simulink are synchronized with the randomized starting coordinates. This prevents any errors during calculations in the first simulation time step, such as for certain reward function components and any observation vectors.

4.2.3 Simulink Environment

The Simulink Environment is an important aspect of training DRL Agents and is an easy way to visualize the reinforcement learning framework. In the figure below, a simplified version of the Simulink training environment is shown.

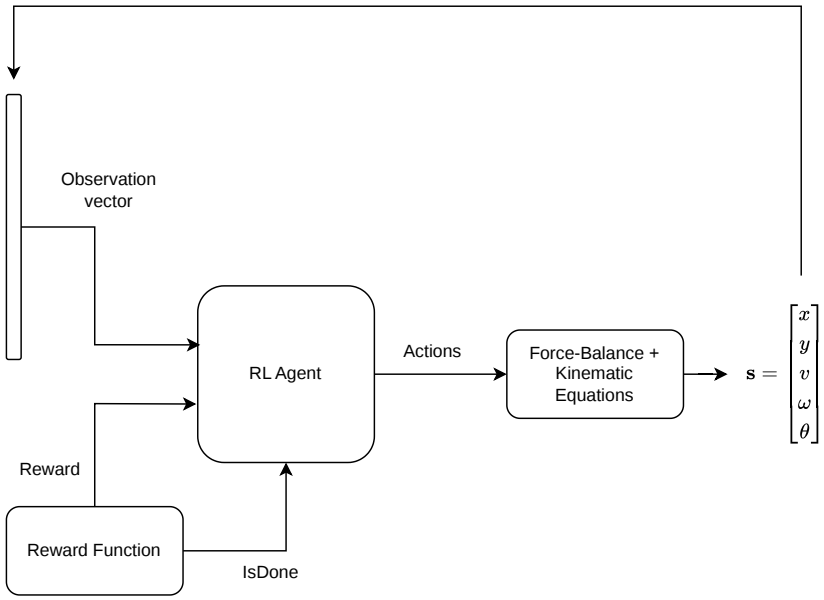

Figure 4.1: *Closed-loop control architecture for the deep reinforcement learning navigation policy. The reinforcement learning agent maps the localized environmental observation vector and step reward to continuous control actions, which update the robot's physical plant via kinematic equations to yield the updated state-space vector* **s**.

Figure 4.1 depicts the RL Agent block with the Observation vector, the Reward Function, and an IsDone signal as its inputs. This signal notifies the agent when a training episode ends, either through a collision, a timeout, or if the agent reaches its target. The outputs of the RL Agent block are the **Actions** of the agent; in this case, they are split into the Force and Torque. These are then used together with the Force-Balance equations, Equation (3.3), and the Kinematic equations, Equation (3.1), to generate the **State-space** variables: x, y, v, w, θ. These variables are then used to make up the **Observations** of the agent, and also used to determine the time step reward in the Reward Function block, where we implemented our reward components as mentioned in Section 3.6.

4.3 Baseline DRL Benchmark Setup

In this section, we will discuss the training parameters and also the different DRL Agent hyperparameters used in this project. We will start by noting the parameters used for the Learning Algorithm Selection, where we compared the DDPG, TD3, and SAC agents. Afterwards, we will discuss the refined parameters used for training the chosen agent to be used in the Leader-Follower system with the LSTM Encoder.

4.3.1 Baseline Training and Hyperparameter Settings

Before we begin training the DRL Agents to compare them, we first need to define some important parameters regarding the agents. These can heavily influence the training results, and some DRL Agents have architecture-specific parameters that can be tuned separately. It is important to note, however, that all these agents will be tested in the same environment, meaning that they all have the same State and Action spaces as well as the same Reward function. This is done to create a fair playing field, which will allow us to compare the agents based on their performance alone. The state-space we will utilize was shown in Equation (3.5), and the action-space was shown in Equation (3.7). The tables below will showcase the different hyperparameters for each DRL Agent in addition to the training parameters.

Table 4.3: *Comprehensive Hyperparameter Comparison for Baseline DRL Agents*

Hyperparameter	**DDPG**	**TD3**	**SAC**
Architecture			
Number of Critics	1	2 (Twin)	2 (Twin)
Actor Network Type	Deterministic	Deterministic	Gaussian (Stochastic)
Hidden Layers	3×128	3×128	3×128
Activation Function	ReLU	ReLU	ReLU
Optimization			
Actor Learning Rate	1×10^{-4}	1×10^{-4}	1×10^{-4}
Critic Learning Rate	1×10^{-3}	1×10^{-3}	1×10^{-3}
Mini-Batch Size	512	512	512
Experience Buffer	1×10^{6}	1×10^{6}	1×10^{6}
Gradient Threshold	5	5	5
Exploration Strategy			
Primary Method	Gaussian Noise	OU Noise	Automated Entropy
Init. Noise / Alpha	0.1	0.1	Dynamic
Entropy Learn Rate	N/A	N/A	1×10^{-4}
Algorithm Specifics			
Target Smooth Factor (τ)	5×10^{-3}	5×10^{-3}	5×10^{-3}
Policy Update Freq.	1	2 (Delayed)	1
Target Policy Noise (σ)	N/A	0.2	N/A
Target Entropy	N/A	N/A	-2.0

As we can see from Table 4.3, the agents were trained using extremely similar hyperparameters. This was by design to ensure a fair comparison between the different agent architectures. Most of the differences are noted in the *Algorithm Specifics* section of the table and are made simply due to the different capabilities of each agent. Hyperparameters for these agents, such as learning rates and optimizer selection, were chosen based on established empirical standards in reinforcement learning literature.

Table 4.4: *Global Training and Evaluation Configuration*

Parameter	**Symbol**	**DDPG / TD3**	**SAC**
Maximum Episodes	E_{max}	10,000	10,000
Maximum Steps per Episode	T_{max}	300	300
Simulation Duration	T_f	30 s	30 s
Sample Time	Δt	0.1 s	0.1 s
Averaging Window Length	-	50 episodes	100 episodes
Convergence Threshold	-	495	505
Evaluator Settings			
Evaluation Frequency	-	50 episodes	50 episodes
Episodes per Evaluation	-	10	10

In Table 4.4, we can see that exactly the same training and evaluation configuration was used for the DDPG and TD3 agents. The parameters were fairly similar for the SAC agent, with only the averaging window length and convergence threshold being adjusted. These adjustments were made after the SAC agent outperformed the DDPG and TD3 agents during initial tests. By increasing the convergence threshold, stricter convergence criteria were imposed on the SAC agent, resulting in a more rigorous evaluation compared to the DDPG and TD3 agents. The value for this parameter was decided based on the maximum obtainable reward for a successful episode. The reward structure will be explained in the following section. The averaging window length was chosen to be 50 and 100 to visualize the agent's learning behavior during training.

To evaluate the true performance of the DRL agents independently of training noise, an evaluation statistic managed by the MATLAB rlEvaluator object was incorporated into the comparison framework. During standard training episodes, random exploration causes the raw cumulative rewards to fluctuate significantly as the agent experiments within the workspace. An evaluation cycle addresses this volatility by temporarily pausing training,

disabling all random exploration, and forcing the agent to execute its deterministic policy over a series of clean test trials to calculate a true snapshot of its generalization skills. This diagnostic process is governed by two parameters: the evaluation frequency, which specifies the interval of training episodes that must pass between consecutive evaluation cycles, and the number of episodes, which dictates the quantity of consecutive test runs executed per cycle to ensure that stochastic variations in initial starting configurations do not skew the resulting mean episode reward metric.

4.3.2 Baseline Navigation Reward Function

For the baseline comparison between DDPG, TD3, and SAC, the same reward function was applied for all DRL Agents to ensure the performance differences were only due to the algorithm itself. As previously noted, the workspace for these trials was constrained to 5×5 m, with terminal boundaries at ± 2.5 m. The total reward R is defined by:

$$R = r_s + r_c + r_p + r_d + r_v + r_w$$

The components and their specific weighting factors are summarized below.

- **Success Reward**

$$r_s = \begin{cases} 500 & d_t \leq 0.1 \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

- **Boundary Penalty**

$$r_c = \begin{cases} -100 & |x| \geq 2.5 \text{ or } |y| \geq 2.5 \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

- **Progress Reward**

$$r_p = 20(d_{t-1} - d_t) \tag{4.3}$$

- **Distance-to-Goal Penalty**

$$r_d = -0.1 \cdot d_t \tag{4.4}$$

- **Approach Velocity Penalty**

$$r_v = -v_t^2 \exp(-0.5 d_t^2) \tag{4.5}$$

- **Angular Stability Penalty**

$$r_w = -0.1 \cdot \omega_t^2 \tag{4.6}$$

With,

$$\begin{aligned} d_t &= \sqrt{x_t^2 + y_t^2} \\ d_{t-1} &= \sqrt{x_{t-1}^2 + y_{t-1}^2} \end{aligned} \tag{4.7}$$

These variables calculate the Euclidean distance of the agent to the goal, $(0, 0)$. The reward structure was designed to prioritize arrival at the $(0, 0)$ target while ensuring kinematic stability. The highest single instance reward is given by the **Success Reward** r_s, which purposefully outweighs the **Boundary Penalty** r_c, to serve as the primary terminal objective. To provide a continuous gradient for the agent, the **Progress Reward** (r_p) is heavily incentivized with a gain of 20, representing the second-highest weighting in the function. This component also ensures that any movement away from the target is immediately penalized. The two smallest components of this reward function are the **Distance-to-goal Penalty** r_d and the **Angular Stability Penalty** r_w, with a gain of 0.1. The purpose of these components was to refine the movement of the agent towards the goal, and they are critical for suppressing the circling behavior often observed in autonomous agents. The final component is the **Approach Velocity Penalty** r_v, whose exponential term increases in magnitude as the distance to the target (d_t) decreases, which discourages high linear velocity, especially near the target. By not using a single sparse velocity reward at the goal, we simplify the training process for the agent and significantly reduce training time.

4.4 SAC-LSTM Leader–Follower Control Implementation

Following the results of the comparison experiment between the DRL Agents, the best performing agent was used to set up the Leader-Follower system in combination with the LSTM Encoder for navigation with obstacle avoidance. This section is divided into three main parts, focusing on the implementation of the LSTM Encoder, the Leader Agent, and the Follower Agent.

4.4.1 Obstacle Encoder Dataset and Training Procedure

Just like the DRL Agent, the LSTM encoder needs to be trained as well. However, unlike the DRL Agent, this encoder will be pre-trained and implemented into the Leader-Follower environment. The parameters used to train the LSTM Encoder are shown below.

Hyperparameters for LSTM Training

Table 4.5: *LSTM Encoder Training and Dataset Generation Parameters*

Parameter	**Value**	**Description**
Architecture & Training		
LSTM Hidden Units	32	Recurrent state size
Latent Dimension	16×1	Bottleneck feature vector size
Optimizer	Adam	Adaptive Moment Estimation
Initial Learning Rate	1×10^{-3}	Step size for weight updates
Mini-Batch Size	128	Samples per gradient update
Training Epochs	20	Full passes through the dataset
Dataset Generation		
Total Samples	20,000	Randomized configurations
Input Sequence Size	3×10	$[dx, dy, d_{edge}]$ for 10 obstacles
Prediction Horizon (T_{horizon})	2.0 s	Straight-line projection time
Time Step (Δt)	0.05 s	Simulation resolution
Simulated Velocity (v_{sim})	0.5 m/s	Straight-line velocity
Safety Margin (d_{safe})	0.25 m	Robot radius and clearance

The LSTM Encoder was trained using a supervised classification approach to provide the DRL Agents with a latent representation of the surrounding obstacles. The architecture consisted of 32 LSTM hidden units and a latent dimension of 16. This configuration was chosen to balance computational efficiency with the obstacle representation capability. To ensure the encoder learned obstacle representations, a dataset of 20,000 randomized configurations of robot states and obstacle locations was generated in a 10×10 m workspace. The labeling process used a constant simulated velocity (v_{sim}) of 0.5 m/s and a prediction horizon (T_{horizon}) of 2.0 s. To account for the physical dimensions of the robot, a safety margin (d_{safe}) was implemented. These values were chosen to mimic safe operating states of our laboratory's differential drive robots. The combination of these parameters was used to train the encoder to anticipate future collisions over a 2-second straight-line rollout.

As mentioned in Section 3.4.1, the input sequence of the encoder consisted of a 3×10 matrix representing up to ten obstacles. This matrix was sorted in descending order to emphasize nearby obstacles appearing later in the sequence. Training was conducted using the Adam optimizer with an initial learning rate of 1×10^{-3} and a mini-batch size of 128.

These standard hyperparameters facilitated stable convergence over 20 epochs, after which the classification layers were removed and the remaining network was utilized as a frozen latent feature encoder producing a latent vector ($\mathbf{z} \in \mathbb{R}^{16\times1}$).

4.4.2 SAC Leader Training

The next step in our implementation is to set up the training parameters for the SAC Leader agent. Many of these parameters were chosen similarly to the comparison experiment, with some slight modifications to account for the increase in complexity of the environment. It is important to note that the SAC Leader was trained as a standalone agent in an environment, without the presence of the follower agent, which will be trained later on.

To maintain consistency with the baseline evaluation, the foundational hyperparameters remain identical to those discussed in Section 4.3.1 and defined in Tables 4.3 and 4.4. However, targeted modifications were introduced to the network architecture to integrate the LSTM encoder interface described in Section 3.4.1. Specifically, the observation vector was expanded to incorporate the LSTM's latent context vector $\mathbf{z} \in \mathbb{R}^{16\times1}$, previously defined in Equation (3.5). In contrast, the continuous action space for the SAC agent remained completely unchanged from the baseline configurations.

Hyperparameters

Table 4.6: *Hyperparameters and Training Configuration for the SAC Leader Agent (with LSTM Encoder)*

Parameter	**Value**	**Description**
Architecture		
Observation Dimension	22×1	Includes 16 latent features from LSTM
Action Dimension	2	$[Force, Torque]$
Hidden Layers	3×128	Fully connected layers
Activation Function	ReLU	Hidden layer non-linearity
Actor Type	Gaussian	Stochastic policy (Mean/Std Heads)
Optimization		
Actor Learning Rate	1×10^{-4}	Step size for policy optimization
Critic Learning Rate	1×10^{-3}	Step size for Q-value estimation
Mini-Batch Size	512	Samples per gradient update
Experience Buffer	1×10^{6}	Capacity for transition memory
Gradient Threshold	5	Protection against exploding gradients
SAC Specifics		
Target Smooth Factor (τ)	5×10^{-3}	Soft update rate for target networks
Target Entropy	-2.0	$-dim(\mathcal{A})$ to maintain exploration
Entropy Learning Rate	1×10^{-4}	Dynamic alpha adjustment rate
Training Management		
Maximum Episodes	15,000	Total allowed training trials
Max Steps per Episode	300	Total steps over the simulation
Simulation Duration (T_f)	30 s	Total simulation time
Sample Time (Δt)	0.1 s	Time increments during simulation
Averaging Window Length	100	Running Average Window Size
Convergence Threshold	1,600	Evaluation score required to stop training
Evaluator Settings		
Evaluation Frequency	50 episodes	How often evaluation cycle runs
Evaluation number	10 episodes	Number of test episodes

The training parameters did see major changes compared to those previously defined. The number of maximum episodes was increased to 15,000 to prepare for a more complex environment that could require longer training. The convergence threshold also increased as the reward function for this training was adjusted as well. The updated reward function with some new reward components will be discussed in the next section.

Reward Function

The reward function for the SAC Leader needed refinement as the goal for this agent became more complex compared to the DRL comparison experiment. The biggest change in the training environment is the introduction of obstacles, which the SAC agent must learn to avoid. The updated reward function, consisting of several reward components, is discussed below.

The comprehensive reward R_t is the summation of all reward components:

$$R_t = r_s + r_c + r_p + r_d + r_v + r_w + r_e + r_{\text{prox}} + r_{\text{step}} \tag{4.8}$$

The components are defined as:

- **Stationary Parking Reward**

  The success reward ensures the agent reaches the goal with near-zero translational and angular velocity.

$$r_s = \begin{cases} 2000 & d_t \leq 0.1,\ |v_t| < 0.07,\ |\omega_t| < 0.1 \\ 2 & d_t \leq 0.1,\ \text{otherwise} \\ 0 & \text{otherwise} \end{cases} \tag{4.9}$$

- **Boundary and Collision Penalty**

  A terminal penalty is applied for workspace boundary violations or if the robot's distance to the closest obstacle edge (d_{min}) falls below the safety radius.

$$r_c = \begin{cases} -1000 & |x_t| \geq 10 \text{ or } |y_t| \geq 10 \text{ or } d_{\min} \leq 0.10 \\ 0 & \text{otherwise} \end{cases} \tag{4.10}$$

- **Navigation Shaping Components**

  Continuous feedback terms are used to guide the agent toward the goal and penalize erratic behavior:

  - Progress Reward

$$r_p = 40(d_{t-1} - d_t) \tag{4.11}$$

  - Distance Penalty

$$r_d = -1.0 \cdot d_t \tag{4.12}$$

  - Approach Velocity Penalty

$$r_v = -v_t^2 \exp(-0.5 d_t^2) \tag{4.13}$$

  - Angular Stability Penalty

$$r_w = -0.1 \cdot \omega_t^2 \tag{4.14}$$

  - Control Effort Penalty

$$r_e = -0.1(F_{t-1}^2 + T_{t-1}^2) \tag{4.15}$$

- **Gaussian Obstacle Repulsion**

  A soft-constraint penalty is applied within a 1.5 m safety buffer of any obstacle.

$$r_{prox} = \begin{cases} -100 \exp\left(-\frac{d_{\min}^2}{2\sigma^2}\right) & d_{\min} \leq 1.5 \\ 0 & \text{otherwise} \end{cases} \tag{4.16}$$

  where $\sigma = 0.5$.

- **Time Step Penalty**

  To encourage time-optimal trajectories:

$$r_{\text{step}} = -0.5 \tag{4.17}$$

The first major refinement of the reward function is the introduction of the **Stationary Parking Reward** (r_s), which rewards the agent for reaching near-zero linear and angular

velocity near the goal. This reward combines the goal-reaching reward discussed in Section 4.3.2, with a stationary stopping criterion and functions as the main termination condition of the agent. The **Boundary and Collision Penalty** (r_c) has also been updated; it now depends on a larger boundary area of ± 10 m and includes the term $d_{\min}$, which is defined as the Euclidean distance between the agent and the closest obstacle. The limit of $d_{\min}$ is set at 0.1 m, which teaches the agent to stay at least 10 cm away from any obstacle. This reward component is weighted as the second highest, highlighting the importance of this component for the training of the agent.

The next components of the reward function are focused on the navigation of the agent. Many of these components remained the same as those previously discussed, with a slight increase to the weight of the **Progress Reward** (r_p) to entice the agent to continue moving towards the goal even with obstacles in the way. An addition to the navigation components is the **Control Effort Penalty** (r_e), which penalizes the agent for the actions taken in the previous time-step regardless of the outcome to encourage smooth control transitions, as well as to make the agent reach the defined goals with some urgency. Another component focused on reaching the goals more quickly is the **Time Step Penalty** (r_{step}), which is also applied every time step until the agent reaches its stopping criterion.

The addition of obstacles into the training environment greatly increased the complexity, requiring us to include a dedicated component focused on obstacle avoidance. Our implementation of the **Gaussian Obstacle Repulsion Penalty** (r_{prox}) teaches the agent to keep its distance from obstacles. This is a dense penalty, meaning that the magnitude of the penalty constantly changes and depends on the distance between the obstacle and the agent. While this type of penalty could be achieved using other algebraic forms, we opted to utilize the Gaussian distribution function to control the magnitude of the penalty.

Simulation Environment

Before we can continue with training the SAC leader agent, the training environment must be defined. While much of this stays similar to the environment used to compare the DRL agents, we must incorporate one major aspect: the obstacles. Defining the obstacles is very simple; we need to assign the center point of the obstacle, and for simplicity, we will be using circular obstacles. This means we need to define the radius of the obstacles, which will help us calculate the distance from the robot to the edge surface of the obstacle, which is what the LSTM encoder needs to derive its latent vector. The obstacles used for SAC leader training are detailed below.

Table 4.7: *Geometric and Kinematic Properties of Training Obstacles for the SAC-LSTM Leader Agent*

Obstacle ID	**Initial Position** (x_0, y_0) **[m]**	**Radius** r **[m]**	**Motion Type**	**Velocity Vector v [m/s]**
1	$(-4.0, 2.0)$	0.75	Dynamic (Horizontal)	$[0.50, 0.00]^T$
2	$(2.0, -4.0)$	0.50	Dynamic (Vertical)	$[0.00, 0.25]^T$
3	$(5.0, 5.0)$	0.35	Static	$[0.00, 0.00]^T$
4	$(-4.0, -2.0)$	0.20	Static	$[0.00, 0.00]^T$
5	$(3.0, -3.0)$	0.80	Static	$[0.00, 0.00]^T$

4.4.3 SAC Follower Training

After training the SAC Leader Agent, the next step is to train the SAC Follower Agent. For this section of the study, we will be using mostly the same hyperparameters as the leader. The only differences will be with the observation vector and the simulation duration. Additionally, we must also transform the global error relative to the moving goal behind the leader into the local coordinates of the follower, as discussed in Section 3.5. It is important to note that the leader and the follower are not trained at the same time. The environment of the follower includes a pre-trained leader policy running in inference mode to generate the required state-space variables for the follower to observe.

Hyperparameters

Table 4.8: *Hyperparameters and Training Configuration for the SAC Follower Agent*

Parameter	**Value**	**Description**
Architecture		
Observation Dimension	26×1	Relative state to leader/formation goal
Action Dimension	2	$[Force, Torque]$
Hidden Layers	3×128	Fully connected layers
Activation Function	ReLU	Hidden layer non-linearity
Actor Type	Gaussian	Stochastic policy for tracking
Optimization		
Actor Learning Rate (L_{ra})	1×10^{-4}	Step size for policy optimization
Critic Learning Rate (L_{rc})	1×10^{-3}	Step size for Q-value estimation
Mini-Batch Size	512	Samples per gradient update
Experience Buffer	1×10^{6}	Capacity for transition memory
Gradient Threshold	5	Protection against exploding gradients
SAC Specifics		
Target Smooth Factor (τ)	5×10^{-3}	Soft update rate for target networks
Target Entropy	-2.0	$-dim(\mathcal{A})$ to maintain exploration
Entropy Learning Rate	1×10^{-4}	Dynamic alpha adjustment rate
Training Management		
Maximum Episodes	15,000	Total allowed training trials
Max Steps per Episode	600	Total steps over the simulation
Simulation Duration (T_f)	60 s	Total simulation time
Sample Time (Δt)	0.1 s	Time increments during simulation
Averaging Window Length	100	Running Average Window Size
Convergence Threshold	4,200	Evaluation score required to stop training
Evaluator Settings		
Evaluation Frequency	50 episodes	How often evaluation cycle runs
Evaluation number	10 episodes	Number of test episodes

The observation vector for the follower must now include information on the state of the leader, as these parameters are essential for designing a robust reward function. The observation vector for the follower was expanded to a 26×1 vector in the following form:

$$\mathbf{O}_{F,t} = \begin{bmatrix} x_{g,t-1} & y_{g,t-1} & v_F & \omega_F & \sin\theta_{\text{rel}} & \cos\theta_{\text{rel}} & X_L & Y_L & v_L & \omega_L & \mathbf{z}^T \end{bmatrix}^T \tag{4.18}$$

The first two components, $x_{g,t-1}$ and $y_{g,t-1}$, represent the relative tracking goal coordinates from the previous time-step. The follower's own motion is captured by its current linear velocity v_F and angular velocity ω_F, while the relative orientation between the two robots is included using the trigonometric terms $\sin\theta_{\text{rel}}$ and $\cos\theta_{\text{rel}}$. To ensure accurate coordination, the leader's states are also included in the vector; these consist of the leader's global position coordinates (X_L, Y_L) along with its linear and angular velocities (v_L, ω_L). Finally, the vector is appended with the latent vector $\mathbf{z}^T$ from the frozen LSTM encoder.

Additionally, the simulation time (T_f) was increased to 60 s to account for the increase in complexity of the environment. Compared to the leader, this agent now has a moving goal while avoiding obstacles as well. This adjustment was implemented as a preventive measure to ensure the agent had enough time to effectively learn an optimal policy.

Reward Function

As mentioned at the end of the previous section, the reward function now needs to be adjusted to accommodate a moving end goal. This brings additional complexity, as the goal has its own velocity until the leader comes to a stop at the origin. We must also ensure that the episode ends when both the leader and follower reach their respective goals, as the follower can achieve its target position before the leader reaches the origin, depending on the initial starting configurations. The new reward function, consisting of several reward components, is discussed below.

The total follower reward R_F is calculated as the summation of all reward components:

$$R_F = r_t + r_c + r_g + r_p + r_d + r_v + r_w + r_o + r_s + r_{vm} + r_{\text{prox}} \tag{4.19}$$

The components are defined as:

- **Joint Success Reward**

  A high-magnitude terminal bonus is awarded only when the leader reaches its goal and the follower is precisely within its target relative formation zone.

$$r_g = \begin{cases} 5000 & d_{L,\text{goal}} \leq 0.1 \text{ and } d_t \leq 0.05 \\ 0 & \text{otherwise} \end{cases} \tag{4.20}$$

- **Boundary and Obstacle Collision Penalties**

  Terminal penalties are applied if the relative formation error exceeds the workspace limits or if the follower's distance to the closest obstacle edge (d_{min}) falls below the safety threshold.

$$r_c = \begin{cases} -2000 & |x_{\text{rel}}| \geq 5 \text{ or } |y_{\text{rel}}| \geq 5 \\ -2500 & d_{\min} \leq 0.10 \\ 0 & \text{otherwise} \end{cases} \tag{4.21}$$

- **Velocity Matching (Tracking Stability)**

  To prevent oscillations and ensure a stable platoon, the follower is penalized for the squared difference between its velocities and those of the leader.

$$r_{vm} = -2.0(v_F - v_L)^2 - 0.5(\omega_F - \omega_L)^2 \tag{4.22}$$

- **Formation Shaping and Vertical Deviation Penalty**

  Continuous rewards drive the follower toward the leader while enforcing a penalty to prevent the follower from drifting away from the straight-line path behind the leader.

  – Progress reward:

$$r_p = 10.0(d_{t-1} - d_t) \tag{4.23}$$

  – Vertical Deviation:

$$r_d = -1.0 \cdot d_t - 5.0(y_{\text{rel}}^2) \tag{4.24}$$

- **Success Zone Attraction Reward**

  Small rewards triggered within the target zone to provide an "attracting" force for steady-state positioning.

$$r_t = \begin{cases} 2.0 & d_t \leq 0.05 \\ 0 & \text{otherwise} \end{cases}, \quad r_s = \begin{cases} 1.0 & d_t \leq 0.05 \text{ and } |y_{\text{rel}}| \leq 0.02 \\ 0 & \text{otherwise} \end{cases} \tag{4.25}$$

- **Gaussian Obstacle Repulsion**

  A soft-constraint penalty applied within a 1.5 m safety buffer of any obstacle to encourage safe navigation.

$$r_{\text{prox}} = \begin{cases} -100 \exp\left(-\frac{d_{\min}^2}{2\sigma^2}\right) & d_{\min} \leq 1.5 \\ 0 & \text{otherwise} \end{cases} \tag{4.26}$$

  where $\sigma = 0.5$.

- **Orientation and Damping**

  Low-gain terms used to penalize relative heading misalignment and minimize high-frequency control chattering.

  – Relative orientation:

$$r_o = -2.0(1 - \cos\theta_{\text{rel}}) \tag{4.27}$$

  – Velocity damping:

$$r_v = -0.1 \cdot v_F^2 \tag{4.28}$$

  – Angular damping:

$$r_w = -0.1 \cdot \omega_F^2 \tag{4.29}$$

Following the increased complexity and the inclusion of the SAC leader policy running in inference, the stopping criterion for the training episodes needs to be adjusted. The new **Joint Success Reward** (r_g) does exactly this, by awarding a large reward when the distance of the leader to the origin is less than 0.1 m, and the distance of the follower to the goal behind the leader is 0.05 m. The **Boundary and Collision Penalty** (r_c) was adjusted slightly, the boundary itself was decreased to ± 5 m, but the obstacle distance limit $d_{\min}$ is kept at 0.1 m. Another difference in this reward component is that it has been split into 2 sub-components, as the boundary limit was deemed a more important task for this agent. We need to ensure that the follower agent stays close to the leader at all times during the simulation.

A new reward component called **Velocity Matching** r_{vm} was added to reduce any "banding" movement from the follower to the moving target. The goal is to have the follower match the speed of the leader. This component considers both the linear and angular velocity and penalizes any difference in magnitude between the leader and follower.

The **Progress Reward** r_p remains unchanged, with only the weight reduced to 10.0. There is a slight adjustment to the distance error previously defined for the leader; this component now includes a term penalizing vertical distance error toward the leader in an effort to close the gap more quickly. Another new reward component is the **Success Zone Attraction Reward** (r_t, r_s). This component is designed to give the follower rewards when being close to the moving target.

The **Gaussian Obstacle Repulsion Penalty** r_{prox} remained unchanged from the leader's reward function; the same can be said for the **Velocity and Angular Damping components** (r_v, r_w). The last new reward component is the **Relative Orientation Penalty** r_o, which penalizes the follower for having a different orientation than the leader. this in an effort to match the leader's orientation, ensuring that both agents are moving in the same direction.

Chapter 5

Simulation Results and Performance Evaluation

5.1 Baseline DRL Benchmark Results

In this section we will be discussing the DRL Agent comparison experiment. We will first analyze the training progress of each agent and then compare the performance of the agents against each other to decide which agent will be used for the Leader-Follower system with the LSTM Encoder. We will be training the agents configured the the hyperparameters and reward function discussed in Section 4.3. We will start our discussion with the training progress of the DDPG agent.

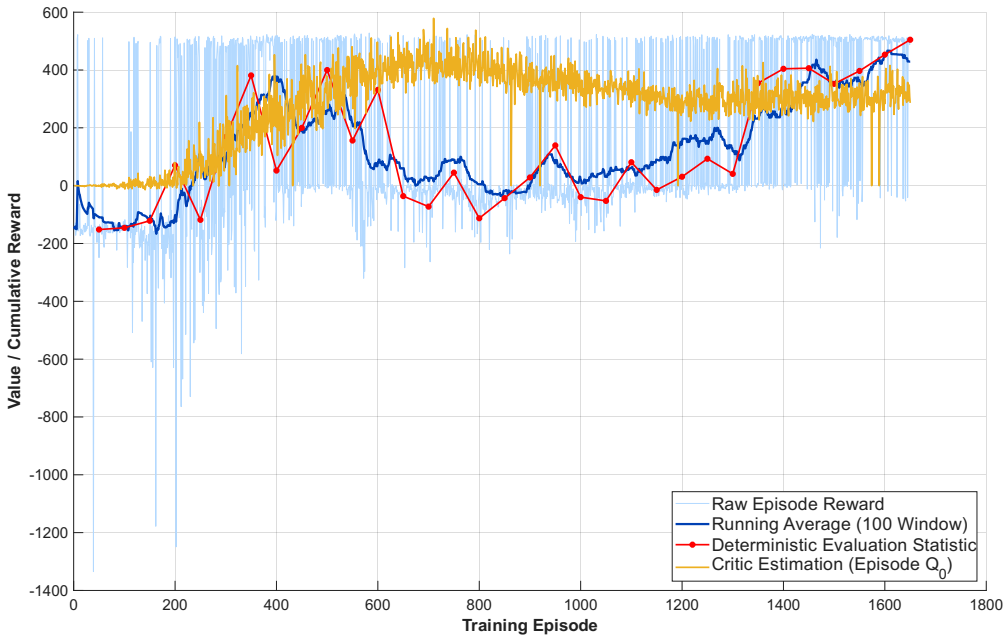

Figure 5.1: *Training plot for the DDPG Agent shows raw episode rewards (light blue) alongside the 100-episode running average (dark blue) and deterministic evaluation scores (red markers). The yellow line tracks the critic's initial value estimation (Q_0).*

The training profile of the DDPG agent is illustrated in Figure 5.1. The policy successfully satisfied the training convergence criterion after 1650 episodes, concluding immediately after the deterministic evaluation statistic achieved an average cumulative reward of 504.6 over 10 independent episodes. As detailed in Section 4.3.1, this evaluation metric gauges the agent's performance without exploration noise over a specified episode window. If this independent evaluation reward equals or exceeds the predefined threshold, training terminates automatically.

Crucially, the training history exposes an architectural vulnerability of standard DDPG: critic overestimation bias. Between episodes 500 and 1300, a distinct divergence occurs where the critic's initial value estimation (Q_0, yellow line) remains consistently inflated, while the true running average reward (dark blue line) and evaluation statistics show a severe performance drop. This divergence demonstrates that if the training termination criterion had depended on the critic's internal value estimation rather than an independent deterministic baseline, the training loop would have ended prematurely. This would have yielded a suboptimal, un-converged policy with poor deployment robustness. Ultimately, the policy recovered from this overestimation phase, achieved an acceptable evaluation statistic reward value, and completed training in a total duration of 51 minutes.

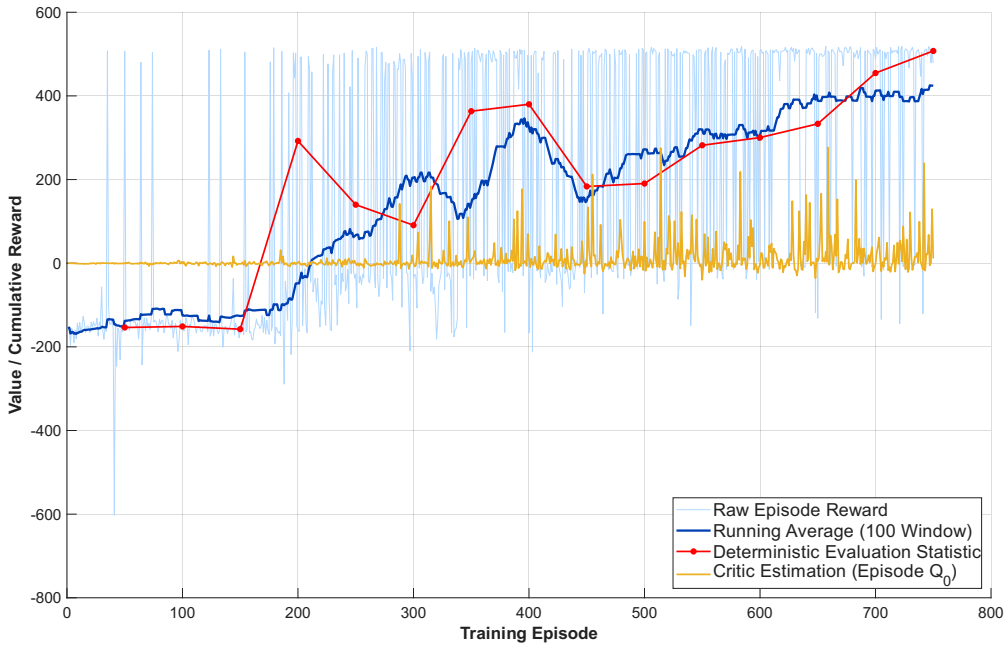

Figure 5.2: *Training plot for the TD3 Agent shows raw episode rewards (light blue) alongside the 100-episode running average (dark blue) and deterministic evaluation scores (red markers). The yellow line tracks the critic's initial value estimation (Q_0).*

Figure 5.2 illustrates the training profile of the TD3 agent, which needed 750 episodes to reach the required stopping criterion. The deterministic evaluation statistic achieved an average cumulative reward of 507.5. This conservative profile explicitly demonstrates the impact of the twin-critic architecture (Section 3.3.2), where the clipped double Q-learning mechanism successfully suppresses overestimation bias by bounding the target value estimation. This training of the TD3 agent completed training in a total of 30 minutes, which represents a reduction of roughly 40% in total training time compared to the DDPG agent, also demonstrating greater computational efficiency.

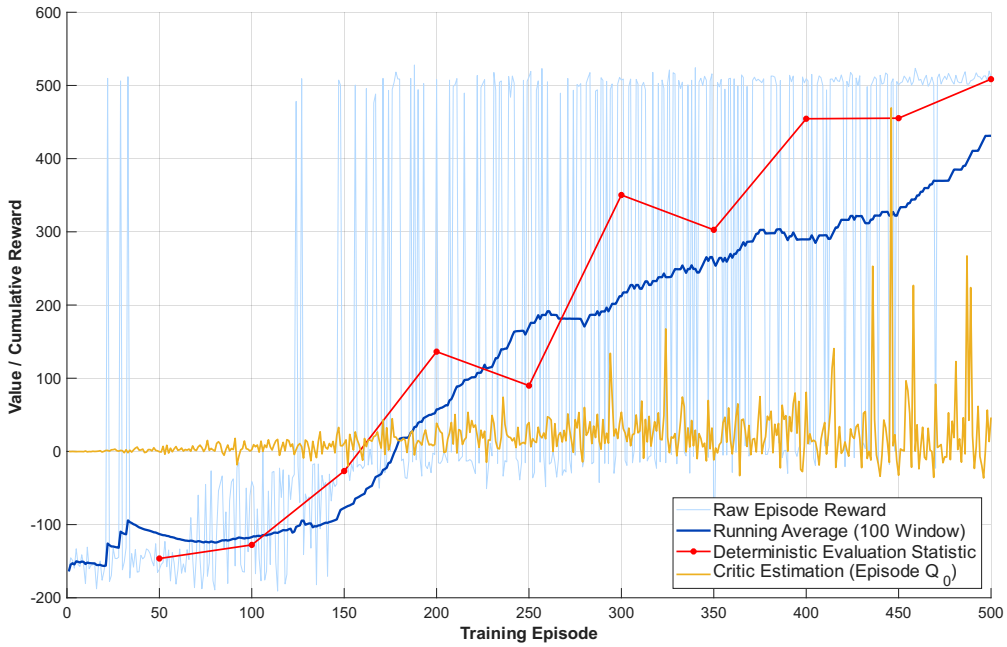

Figure 5.3: *Training plot for the SAC Agent shows raw episode rewards (light blue) alongside the 100-episode running average (dark blue) and deterministic evaluation scores (red markers). The yellow line tracks the critic's initial value estimation (Q_0).*

The training profile of the SAC agent is depicted in Figure 5.3, showcasing how the clipped double Q-learning mechanism operates in tandem with entropy regularization to keep value estimations grounded. This agent's deterministic evaluation statistic achieved an average cumulative reward of 508.6 after 500 episodes. These results validate the advantages of continuous action space exploration via stochastic actions, which optimized the policy layout faster than the previously discussed DRL agents and completed execution in 15 minutes. This represents a 70% reduction in training time compared to DDPG and a 50.0% reduction compared to TD3. Next, we will discuss the performance of these agents when deployed in unseen starting configurations.

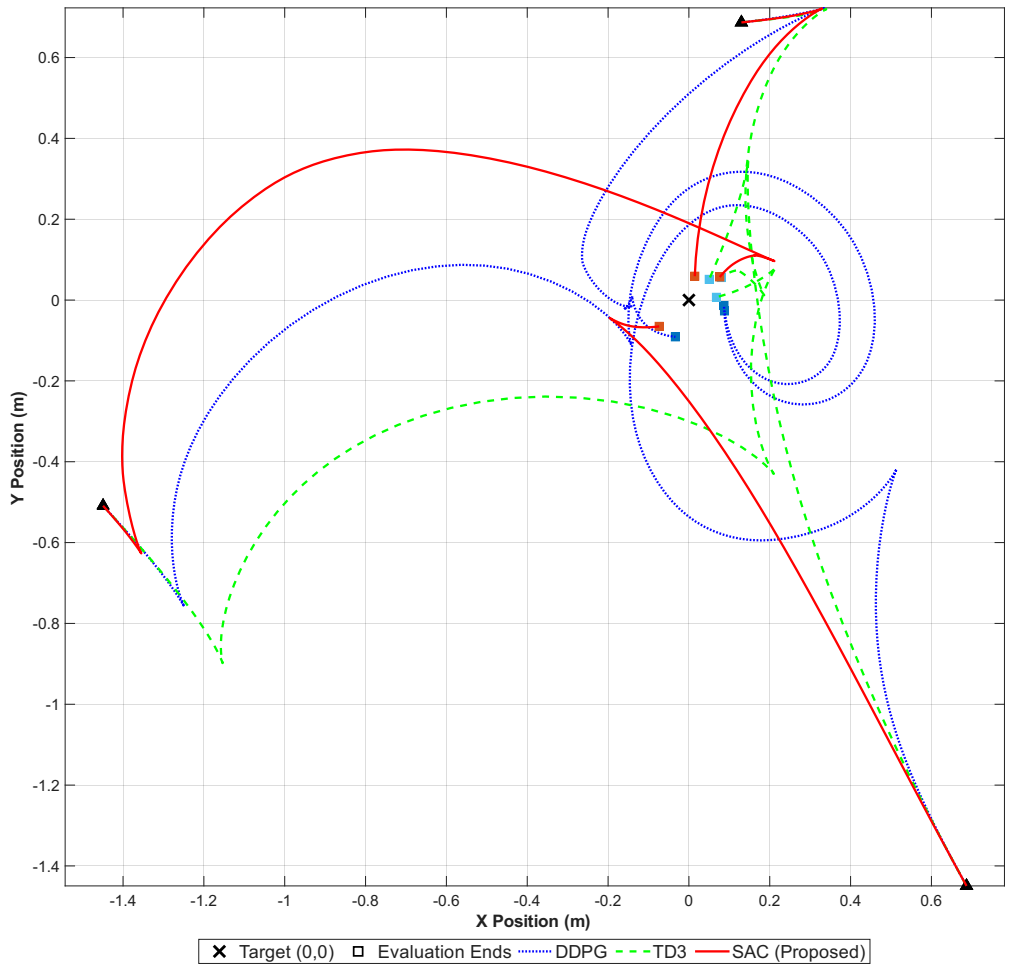

Figure 5.4: *The spatial trajectory comparison of baseline DRL architectures evaluated from three unseen initialization positions toward a stationary target at the origin* $(0, 0)$*. The trajectory of the DDPG agent is shown by the blue dotted line, the TD3 agent is shown by the green dashed line, and the SAC agent is shown by the red line.*

The spatial trajectories of each agent from three unseen starting configurations are illustrated in Figure 5.4. The visual paths directly validate the quantitative trends recorded in Table 5.1. The Soft Actor-Critic (SAC) agent consistently generates smooth, minimal-length trajectories that converge directly to the target at the origin without erratic corrections.

In contrast, the other architectures display severe control deficiencies as they approach the goal state. The DDPG policy produces an orbital spiraling behavior, forcing the robot to repeatedly loop around the origin before coming to a stop short of the target coordinate. This orbital path significantly increases the total distance traveled and delays the settling time. Meanwhile, the TD3 agent experiences severe trajectory chattering and requires correctional movements near the origin. This highly aggressive steering behavior demonstrates

TD3's inability to calculate a smooth deceleration profile in the continuous action space under these initialization constraints. Ultimately, this spatial comparison visually confirms that SAC provides the stable, smooth, and predictable trajectory generation required for a reliable leader-follower framework.

Table 5.1: *Quantitative performance benchmark of baseline DRL architectures evaluated over 10 independent trials.*

Algorithm	**Success Rate (%)**	**Avg. Reward**	**Reward Std. Dev.**	**Avg. Settling Time (s)**	**Avg. Final Distance (m)**
DDPG	80.0	401.56	223.68	11.50	0.0966
TD3	80.0	403.91	217.57	14.99	0.1261
SAC	100.0	510.94	5.26	6.32	0.0895

To validate the reliability, robustness, and generalization capabilities of the trained agents, each one was subjected to 10 independent evaluation trials. Crucially, while all three architectures were evaluated under identical environmental conditions and starting positions to ensure a fair benchmark, these initial states were selected from a completely unseen set of starting configurations different from those encountered during the training phase. The results of this benchmark are summarized in Table 5.1. The Soft Actor-Critic (SAC) framework demonstrated clear superiority across all evaluation criteria, achieving a 100% success rate alongside the highest average return ($\mu = 510.94$) and the closest proximity to the target coordinate (0.0895 m).

The primary distinction between the architectures lies in their stability and operational efficiency, as illustrated in the performance attributes of Figure 5.5. The SAC agent exhibits an exceptionally low reward standard deviation of only 5.26, confirming that the policy converges to a highly consistent and reliable control strategy. In contrast, both DDPG and TD3 demonstrate large variance, with standard deviations exceeding 217. This instability is visually apparent in individual testing trials; for instance, while TD3 successfully tracks the goal in the majority of episodes, it experiences catastrophic tracking degradation in Run 2 (0.2738 m) and Run 4 (0.2923 m). These localized failures heavily penalize its average settling time (14.99 s) and increase its mean final tracking distance to 0.1261 m.

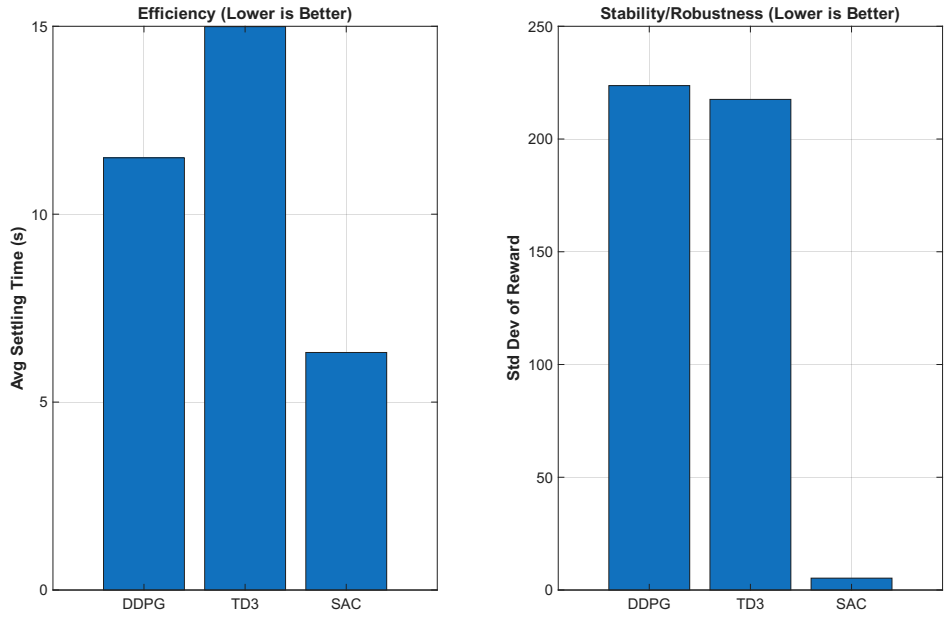

Figure 5.5: *Visual comparison of baseline DRL performance attributes derived from the 10-run evaluation data in Table 5.1: (a) operational efficiency measured by average settling time, and (b) policy stability indicated by the standard deviation of cumulative rewards.*

Furthermore, SAC displays excellent transient behavior, isolating and settling at the goal destination in an average of 6.32 s, nearly halving the time required by DDPG (11.50 s) and outperforming TD3 by 57.8%. This significant margin in sample efficiency and robustness is attributed to SAC's maximum entropy formulation, which encourages thorough exploration of the continuous action space during training and prevents the policy from trapping itself in suboptimal local minima. Thus, the benchmarking data confirms that the SAC architecture provides the ideal foundational policy framework for the multi-agent leader-follower system.

5.2 SAC-LSTM Leader Navigation Results

In this section, the training and operational performance of an optimized SAC agent acting as the system's leader is evaluated. This agent incorporates the latent vector output from the pre-trained LSTM encoder to yield a robust policy capable of dynamic obstacle avoidance.

5.2.1 Obstacle Encoder Evaluation

As discussed in Section 3.4.1, the LSTM encoder is pre-trained via supervised learning on a synthesized tracking dataset to classify whether the current trajectory could lead to a collision. The training progress of this network is shown below.

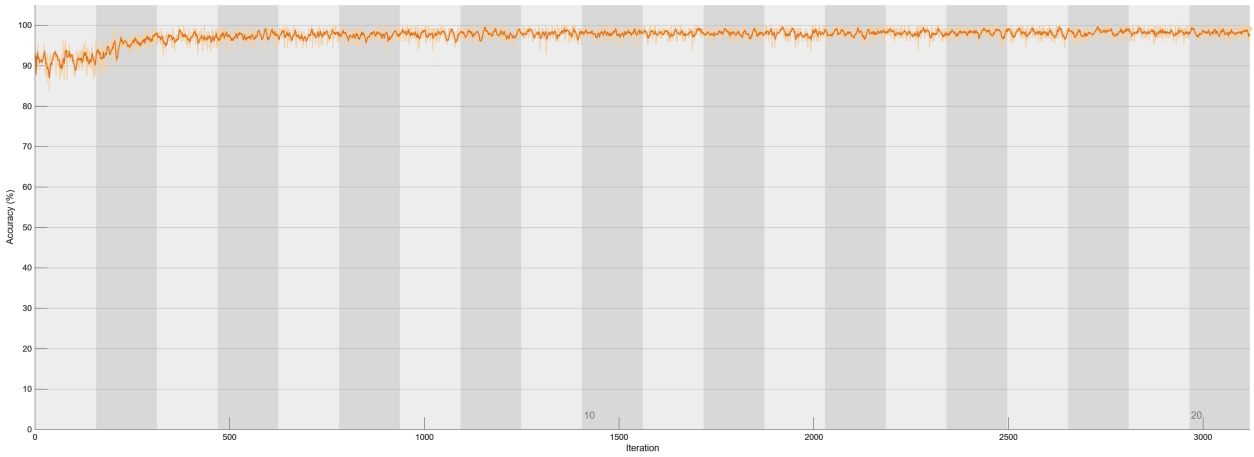

Figure 5.6: *The LSTM training profile demonstrates highly efficient feature extraction, with smoothed classification accuracy (dark orange) converging above 98% by iteration 500. Steady tracking across Epoch 10 (iteration ∼1450) and Epoch 20 (iteration ∼2900) with minimal residual stochastic variance (light orange) indicates strong optimization stability and asymptotic state convergence.*

The figure illustrates the classification accuracy of the LSTM network during its supervised pre-training phase across 3,000 iterations. The network exhibits rapid feature extraction, with the smoothed accuracy curve (dark orange) climbing steeply and stabilizing above 98% within the first 500 iterations. The minimal variance between the raw data (light orange) and the smoothed trend along the extended plateau highlights a highly stable gradient descent process across Epoch 10 and Epoch 20. This robust convergence demonstrates that the LSTM network successfully learned to compress the temporal obstacle configurations into a reliable state representation before being integrated into the reinforcement learning pipeline. We can now implement this encoder into the training environment of the SAC leader agent.

5.2.2 Leader Navigation and Obstacle-Avoidance Performance

Following the successful training of the obstacle encoder, the module is integrated into the SAC leader's environment to train the agent to optimize the objectives defined by the reward function described in Section 4.4.2. The resulting training profile is illustrated in Figure 5.7.

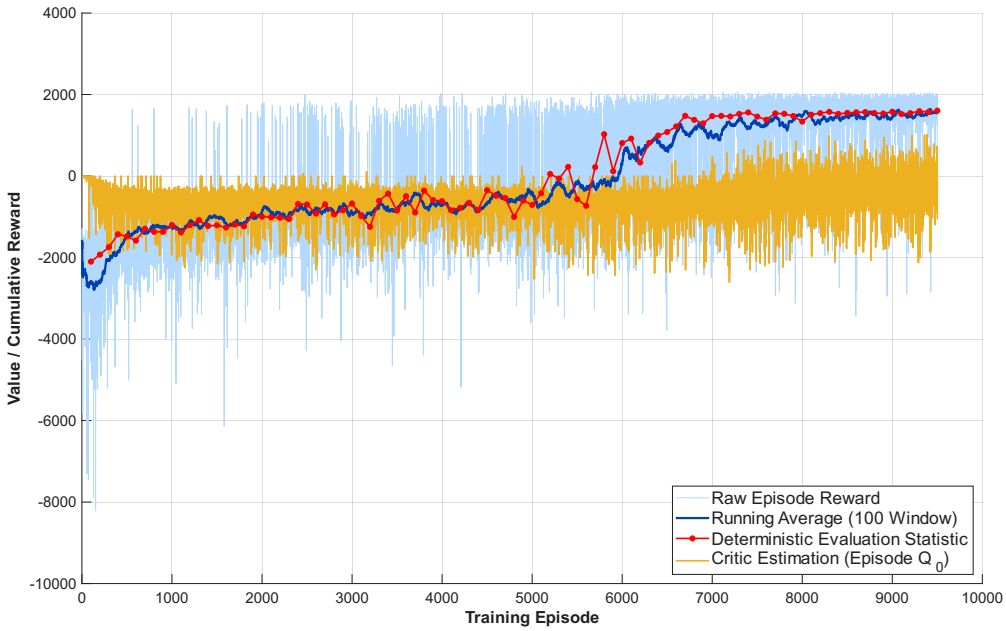

Figure 5.7: *Training plot for the SAC Leader Agent shows raw episode rewards (light blue) alongside the 100-episode running average (dark blue) and deterministic evaluation scores (red markers). The yellow line tracks the critic's initial value estimation (Q_0).*

As indicated in Figure 5.7, the deterministic evaluation statistic of the agent reached an average cumulative reward of 1603.3, successfully triggering the termination threshold preset at 1600. This convergence concluded the training phase at episode 9500, requiring a total execution time of approximately 13.5 hours. As the agent has successfully been trained, we can now analyze its performance when deployed.

To test the performance of the agent, we will deploy it in a slightly modified version of the training environment. We will increase the velocity of one of the moving obstacles and remove one static obstacle to balance it out. Information regarding the obstacles was discussed in Table 4.7.

Now that the environment has been defined, the trained SAC leader is deployed into the environment, which yields the following results.

Table 5.2: *Quantitative Performance Evaluation Metrics for the SAC Leader Agent over 20 Randomized Trials.*

Performance Metric	**Evaluation Value**
Total Test Simulations	20
Success Rate	100.0%
Average Total Reward	1743.20
Reward Standard Deviation (σ)	191.69
Average Settling Time	11.30 s
Average Final Distance to Goal	0.055 m

Table 5.2 validates the robust policy of the trained leader agent, demonstrating a flawless 100% success rate over 20 randomized evaluation episodes. The mean accumulated reward over these trials exceeds the training termination threshold, confirming policy stability post-training. While a noticeable standard deviation ($\sigma = 191.69$) exists in the cumulative reward metrics, this variance is a result of the randomized initial spatial configurations of the obstacles rather than unstable control, as zero collisions occurred across all evaluation windows. Furthermore, the average settling time of 11.30 s outperforms both the baseline DDPG and TD3 architectures outlined in Table 5.1, representing a significant milestone given the increased complexity of the workspace. The average final distance to the goal (0.055 m) marks a major improvement over unoptimized baselines, showcasing the effectiveness of the target-seeking reward component defined in Section 4.4.2. From these 20 episodes, the following 5 simulations were chosen to depict the SAC leader agent performance.

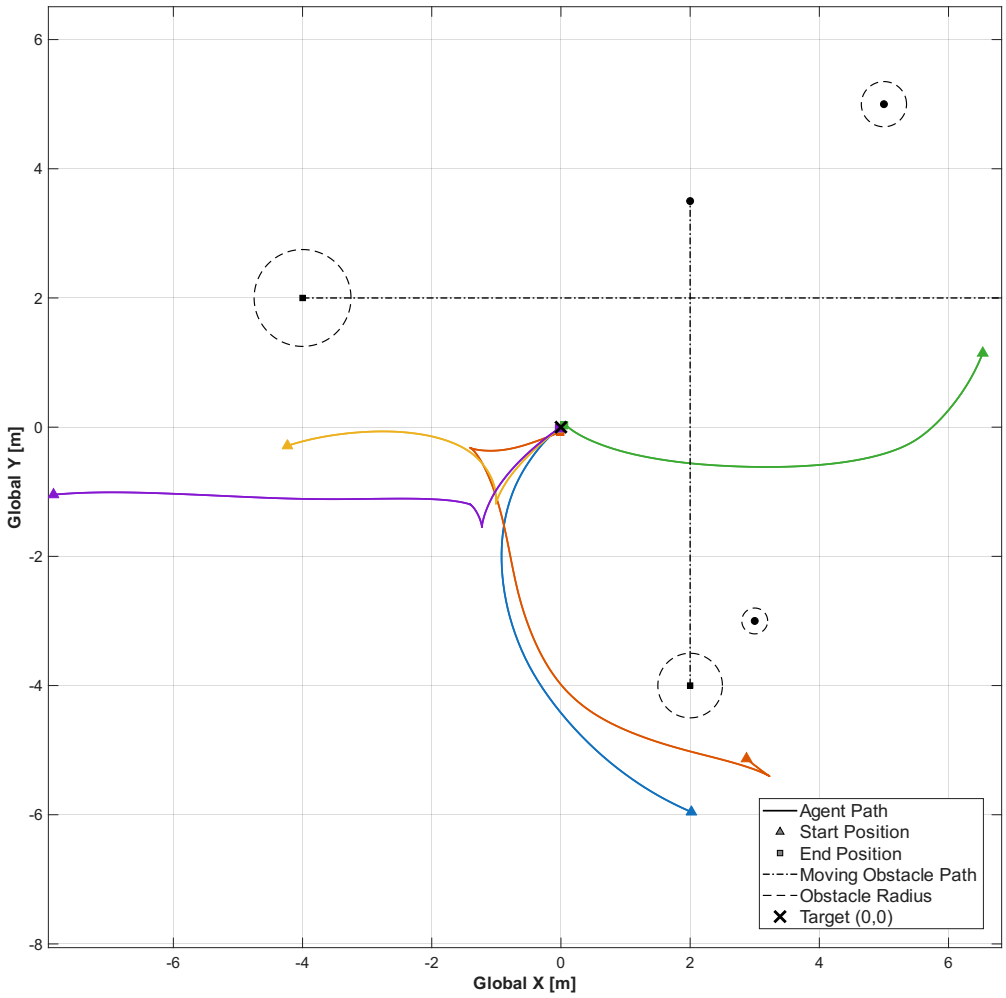

Figure 5.8: *The Agent trajectory plot for the 5 chosen SAC Leader Agents simulated in Table 5.2. The figure shows agent position during the simulation, with each simulation highlighted with a different color. The figure also shows two static components along with two dynamic obstacles, with their paths depicted by dashed dotted lines. All obstacles are shown as circles with a dashed edge.*

Figure 5.8 illustrates the performance of the SAC leader agent through safe and smooth trajectories. The figure clearly shows the agent avoiding obstacles, both static and dynamic, through its curved path towards the goal. Each simulation also ended very close to the goal marked by a cross, which further enforces the results shown in Table 5.2 for the average distance error from the goal.

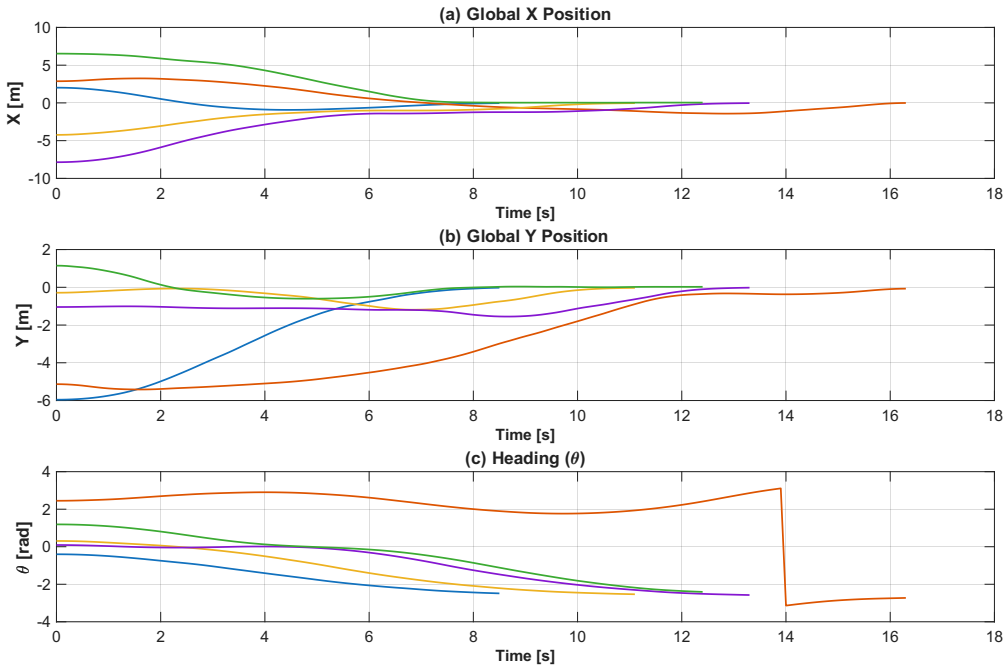

Figure 5.9: *The global coordinates and heading of the simulated agents over time. Figure (a) The global position of the agent in the X dimension. Figure (b) shows the global Y position of the agent, and Figure (c) shows the heading of the agent in the global coordinate frame.*

The positional data in Figure 5.9 shows the smooth movement of the agent in both the X and Y directions. The agent does not have any sudden oscillations or chattering movements. This figure also illustrates the convergence of the agent to the goal with global coordinates $(0, 0)$.

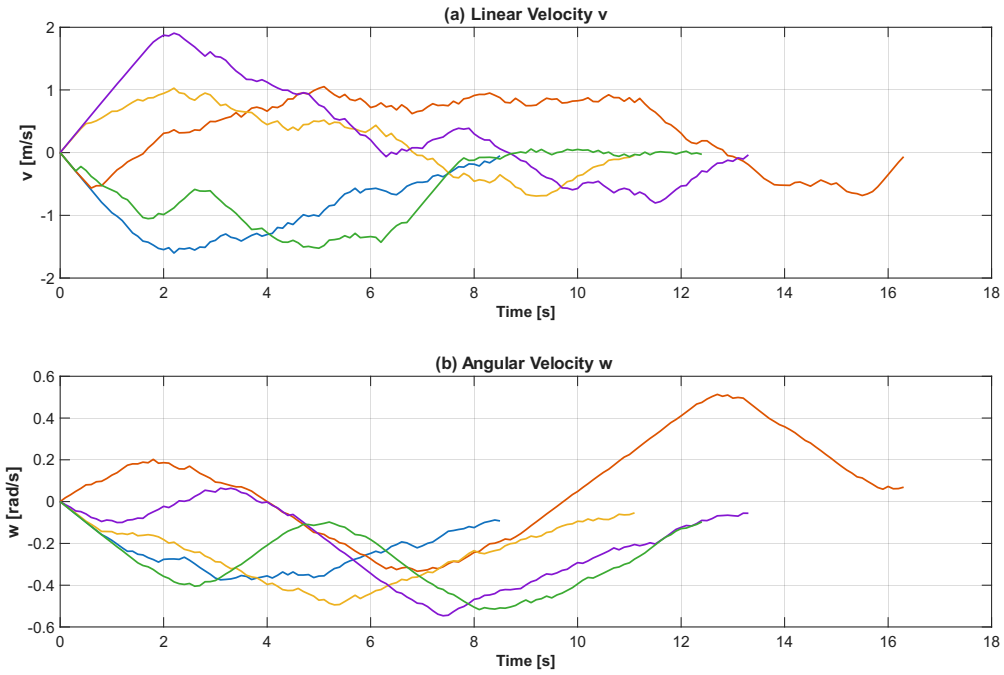

Figure 5.10: *Velocity over time for the SAC Leader Agent. Figure (a) illustrates the linear velocity v, (b) illustrates the angular velocity ω.*

Figure 5.10 plots the velocity change over time for the agent. From this plot, we can conclude that the peak velocity reached over the simulated agents has a magnitude of 2 m/s, which is in a completely acceptable range of velocity for a small mobile robot. While the plots do show some noise, there are no sudden spikes in magnitude, which tells us that the obstacle avoidance is done in a smooth manner, and that no sudden corrections are needed. These figures also showcase the effectiveness of the stopping reward component defined in Section 4.4.2 as both the linear and angular velocity converge to zero as the simulation ends. The angular velocity plot also indicates that there are no unnecessary angle adjustments being made by the agent, which illustrates good trajectory shaping being done by the agent. There are, however, some aspects that can be improved on, as for two of the simulations, the agent is moving in reverse for a large portion of time. While this allows the agent to reach the goal faster without wasting time turning around, it could be overcome in the future by including a reward component focused on keeping the heading of the agent continuously angled toward the goal.

5.3 Single-Follower Tracking Results

Next, the performance of the SAC follower will be discussed. This agent was trained using the parameters and reward function described in Section 4.4.3. Similar to the leader agent, the trajectory plots describing the movement of both the leader and follower are shown below. However, before that, we will show the training profile of the follower agent.

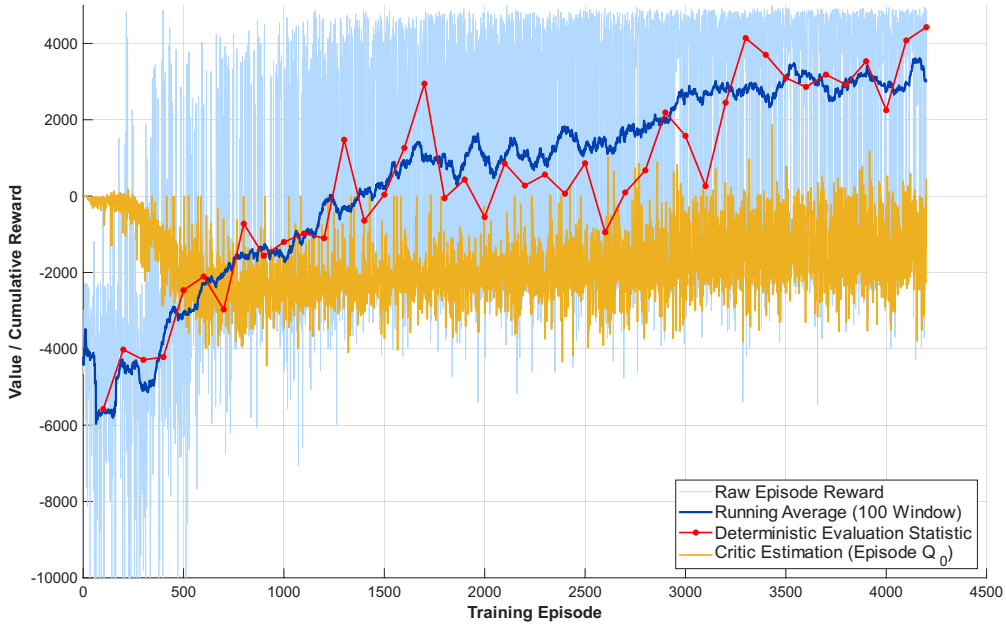

Figure 5.11: *Training plot for the SAC Follower agent shows the raw episode rewards (light blue) alongside the 100-episode running average (dark blue) and deterministic evaluation scores (red markers). The yellow line tracks the critic's initial value estimation (Q_0).*

Similar to the training profile of the leader agent(Figure 5.7), the follower agent also showcases the pessimistic characteristic of the twin-critic setup. The training of the follower agent concluded after the deterministic evaluation statistic averaged a cumulative reward of 4426.5, which triggered the stopping criterion of 4,200. Coincidentally, the training ended after 4200 episodes and needed a total of 3.5 hours to train. The trained agent was then deployed in a slightly modified environment, with changes to one of the moving obstacles and a decrease in the radius of a static obstacle. This change was implemented to observe the agent's robustness when being deployed in an unfamiliar environment.

Table 5.3: *Quantitative Performance Evaluation Metrics for the Follower Agents over 20 Randomized Trials.*

Performance Metric	**Evaluation Value**
Total Test Simulations	20
Formation Success Rate	100.0%
Average Total Reward	4315.67
Reward Standard Deviation (σ)	427.21
Average Formation Time	11.94 s
Average Tracking Error	0.054 m

Table 5.3 shows the follower agent's performance in the modified environment across 20 randomized runs. Despite the changes made, the agent still achieved a 100% success rate. The average tracking error was only 0.054 m, proving that the follower can maintain the tracking goal accurately. Additionally, the average formation time of 11.94 s closely aligns with the leader's settling time, showing that the entire system moves together efficiently. From our results, the follower agent needed only an average of 0.64 s to converge to the goal. The standard deviation in the reward is somewhat high, but because every run ended successfully without a collision, this variance just reflects the different random distances the robots had to travel.

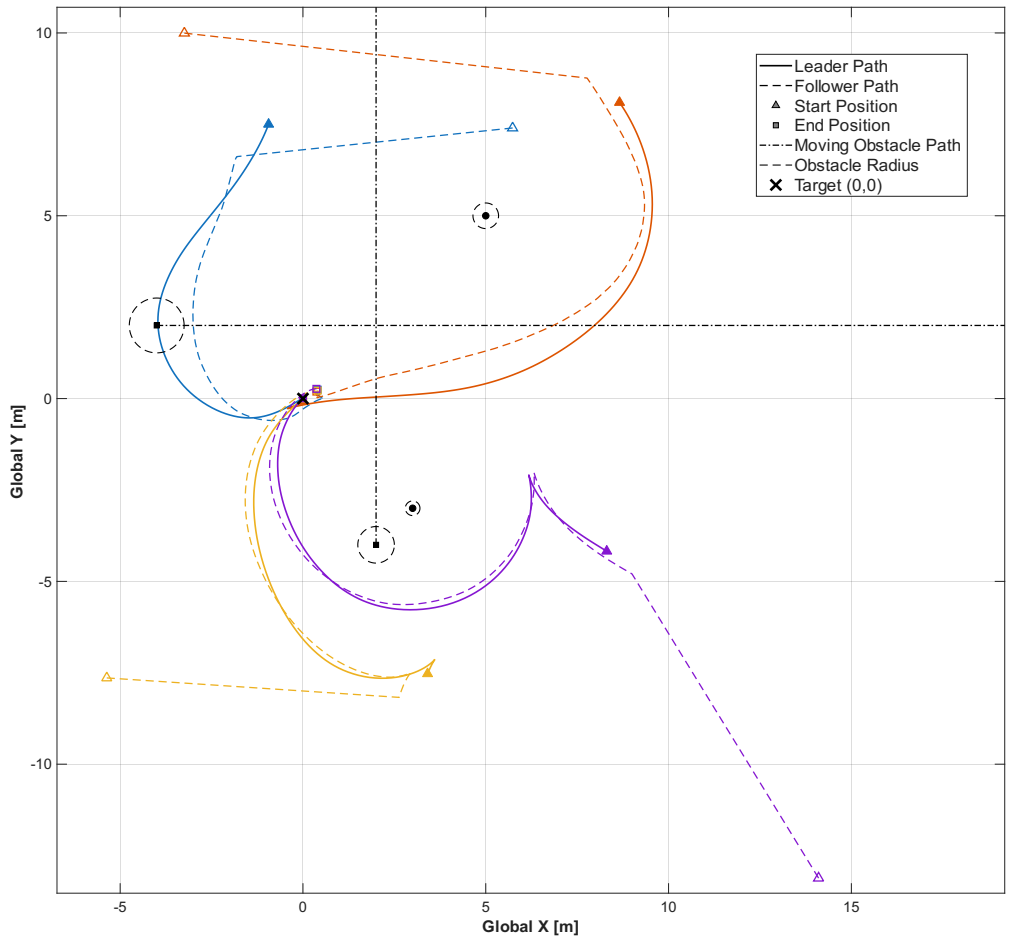

Figure 5.12: *The Agent trajectory plot for the 5 chosen SAC Leader Agents simulated in Table 5.2. The figure shows agent's position during the simulation, with each simulation highlighted with a different color. The figure also shows two static components along with two dynamic obstacles, with their paths depicted by dashed dotted lines. All obstacles are shown as circles with a dashed edge.*

Figure 5.12 illustrates the performance of the SAC leader-follower system through safe and smooth trajectories. The figure clearly shows both the leader and follower agents avoiding obstacles through their curved paths toward the goal. Each simulation also ended very close to the goal marked by a cross, which further supports the results shown in Table 5.3 for the average tracking error. We would like to highlight the purple path generated by the agents, as this demonstrates the effectiveness of the LSTM encoder. Initially, the leader was headed straight to the goal while accounting for the small static obstacle on its left; however, it realized that the vertically moving obstacle would cut off that path and decided to adjust its heading. This decision shows that the agent is able to consider multiple obstacles at each time step and make safe decisions accordingly.

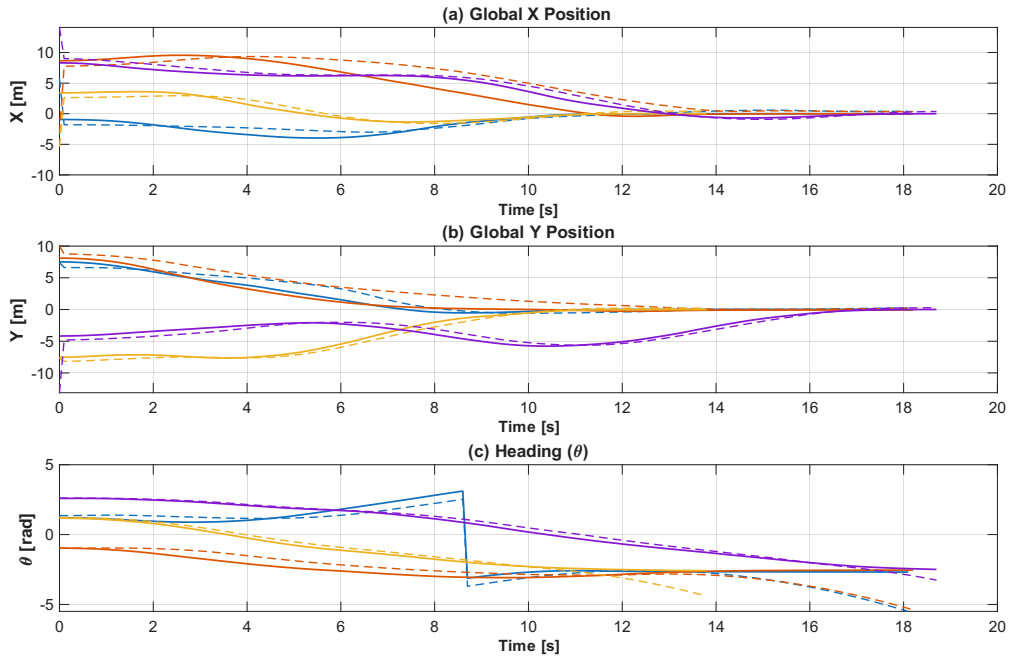

Figure 5.13: *The global coordinates and heading of the simulated agents over time. Figure (a) The global position of the agent in the X dimension. Figure (b) shows the global Y position of the agent and Figure (c) shows the heading of the agent in the global coordinate frame.*

The positional data in Figure 5.13 shows the smooth movement of the leader-follower system in both the X and Y directions. The agent does not have any sudden oscillations or chattering movements. This figure also illustrates the convergence of the agent to the goal with global coordinates $(0, 0)$. The heading is also plotted, showcasing that the follower is able to accurately match the leader's orientation.

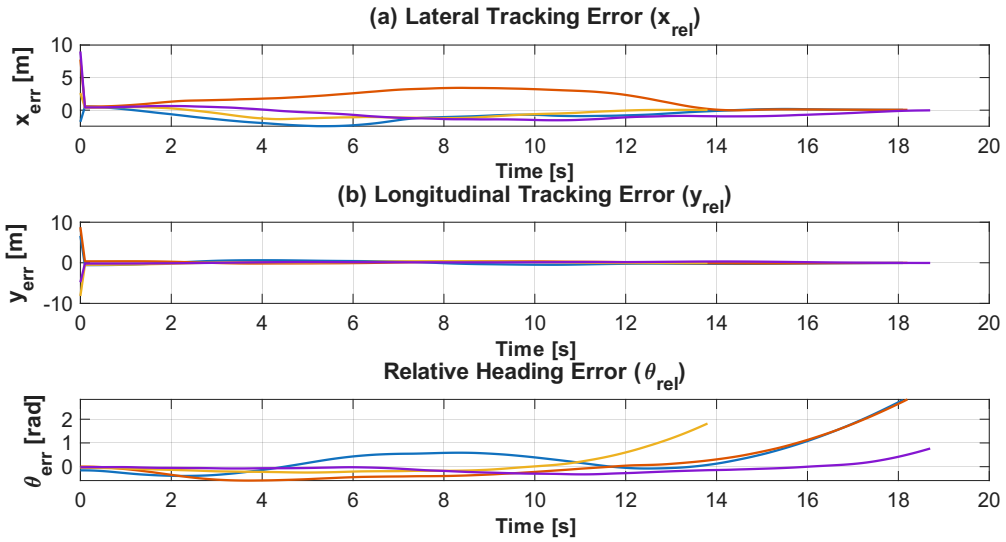

Figure 5.14: *Relative error over time for the SAC Follower Agent. Figure (a) illustrates the lateral tracking error x_{rel}, (b) illustrates the longitudinal tracking error y_{rel} (c) illustrates the relative heading error θ_{rel}.*

Figure 5.14 illustrates the relative tracking error of the follower agent over time relative to its goal behind the leader. As shown in Figures (a) and (b), the large initial tracking errors at the very start of the simulation are corrected almost immediately, with both longitudinal and lateral errors dropping close to zero within the first second.

Looking closer at the lateral tracking error (x_{rel}) in Figure (a), there is a slight deviation between 4 and 12 seconds where the error waves between roughly -2.5 m and 4 m. This temporary fluctuation directly corresponds to the moments when the leader agent is making curved maneuvers to dodge the dynamic obstacles. Once the leader clears the obstacles and returns to a steady heading, the follower quickly catches up, and the longitudinal error settles back to zero by the 14-second mark.

In contrast, the longitudinal tracking error (y_{rel}) in Figure (b) remains exceptionally flat and stays close to zero across all simulated runs. This proves that the follower agent is highly stable at staying locked onto the leader's lateral path, even while the entire formation is performing obstacle avoidance maneuvers.

Finally, Figure (c) tracks the relative heading error (θ_{rel}). The heading error remains low and bounded for the majority of the run, but shows a sharp upward climb at the very end of the simulations. This trend occurs because the agents are reaching the target and slowing down to a complete stop. Once the linear velocity drops to zero, minor final position adjustments cause the relative heading angle to mathematically drift. However, because the actual spatial tracking errors in Figures (a) and (b) are already perfectly minimized at less than 0.05 m, this terminal heading drift does not affect the physical safety or accuracy of the final formation layout.

5.4 Preliminary Triangle Formation Stress Test

To evaluate the scalability of the trained agents, a final stress test was conducted by deploying a three-agent fleet consisting of one leader and two follower agents configured to maintain a symmetric triangle formation. The performance metrics of the system for the simulation trial shown in the plots are compiled in Table 5.4.

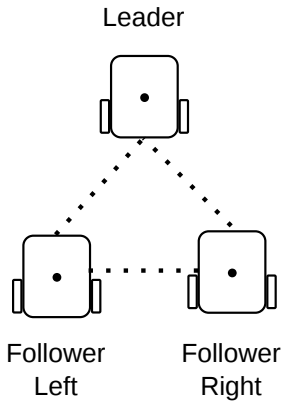

Figure 5.15: *The triangular formation geometry established between the leader and follower agents. The dashed lines represent the specified desired inter-agent relative distances that are maintained by the control architecture.*

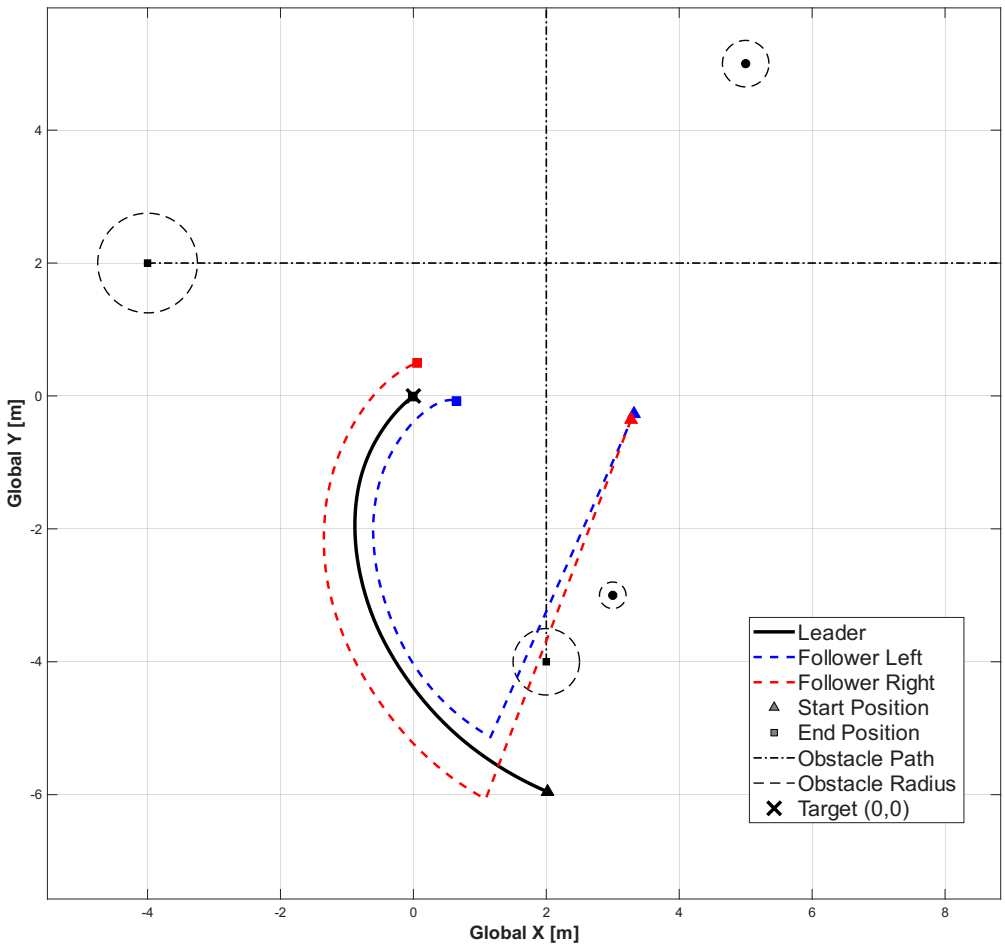

Figure 5.16: *Trajectory profiles for the three-agent triangle formation test. The leader agent successfully navigates to the target origin, while the left follower and right follower attempt to converge into formation from their initial spawning positions.*

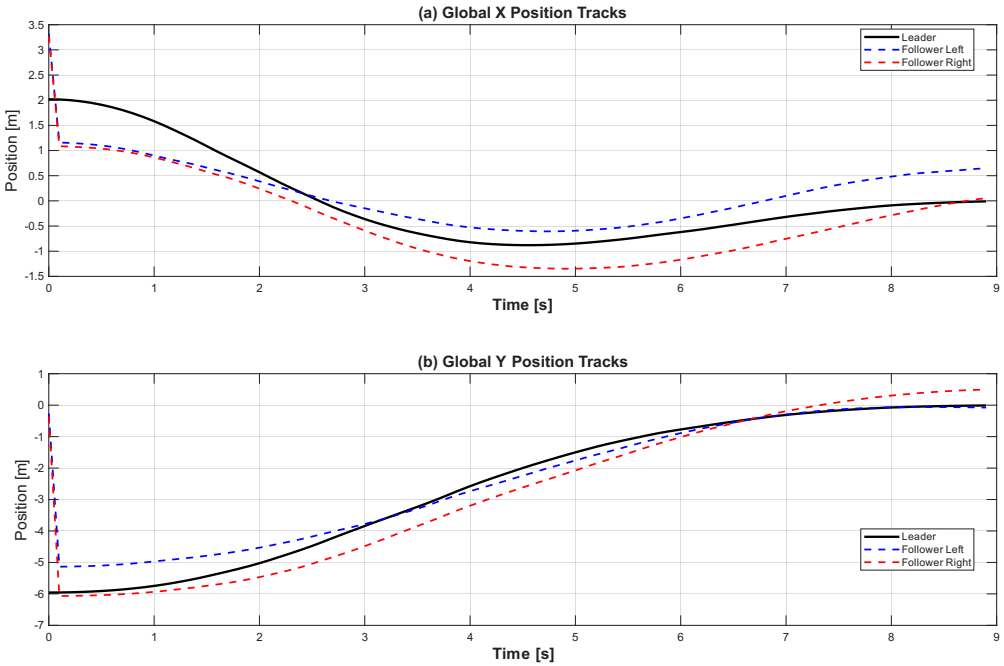

Figure 5.17: *Global position tracks over time for the triangle formation. Figure (a) illustrates the global x-position tracks and Figure (b) illustrates the global y-position tracks for the leader and both follower agents.*

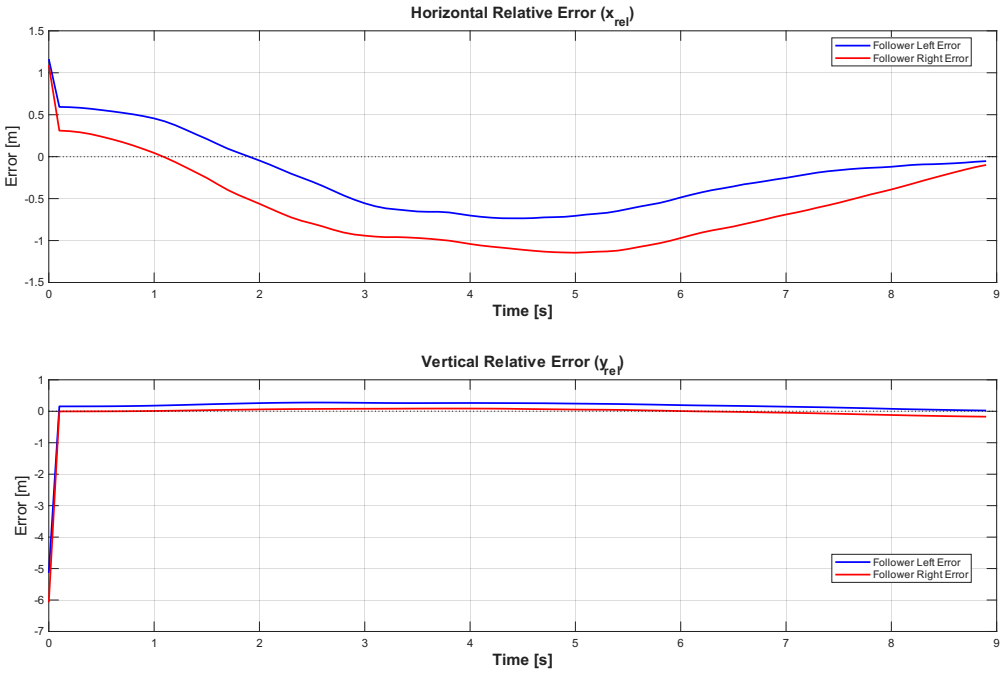

Figure 5.18: *Relative tracking error profiles for the triangle formation followers over time, showing (a) horizontal relative error (x_{rel}) and (b) vertical relative error (y_{rel}).*

Table 5.4: *Quantitative Performance Evaluation Metrics for the Representative Triangle Formation Trial.*

Performance Metric	**Evaluation Value**
Total Reward (Left Agent)	4719.36
Total Reward (Right Agent)	-390.14
Average Tracking Error (Left Agent)	0.058 m
Average Tracking Error (Right Agent)	0.198 m

As indicated in Table 5.4, the success rate for this run was 0.0%, meaning a collision stopping criterion was triggered or the followers did not converge within the specified tracking goal of 0.05 m. Looking at the trajectory profiles in Figure 5.16, this could be a result of the initial spawning setup rather than a failure of the tracking control. The follower agents were initialized close together, while the leader agent was spawned nearly 6 m further away. Because of this initial separation, the followers had to make aggressive corrections to catch up to the leader. On its way to the formation slot, the right follower cut too close to the obstacle's safety radius, triggering the early collision failure criterion.

This early collision explains the big difference in the results between the two agents. The right agent took a large collision penalty, resulting in a negative reward of -390.14 and a higher tracking error of 0.198 m. On the other hand, the left agent managed to avoid the obstacle, earning a high reward of 4719.36 and maintaining a very precise tracking error of only 0.058 m.

Despite the collision at the start, the time-series data shows that the underlying formation control loop works exactly as intended once the agents catch up. As illustrated in Figure 5.17 and Figure 5.18, within the first second of initialization, the tracking errors snap directly close to zero. For the remaining 8 seconds of the simulation, both followers track the leader's global x and y position movements perfectly all the way to the global goal $(0, 0)$. This demonstrates that the controller is fully capable of holding a stable triangle formation, and the early initialization issues can be fixed in the future by using safer spawning zones and more heavily penalizing collisions. Another aspect to note is that the followers are on the opposite sides of the leader, which is most likely due to the leader traveling in reverse. Which is something we have seen in Figure 5.10, this would explain the perceived phenomena.

To provide a more complete view of the system's performance, Table 5.5 summarizes the aggregate data across all 10 randomized simulation trials.

Table 5.5: *Aggregate Quantitative Performance Metrics for the Triangle Formation Across 10 Randomized Trials.*

Performance Metric	**Evaluation Value**
Total Test Simulations	10
Success Rate	0.0%
Average Reward (Left Agent)	1606.11
Average Reward (Right Agent)	8.79
Average Tracking Error (Left Agent)	1.535 m
Average Tracking Error (Right Agent)	1.289 m

As shown in Table 5.5, the success rate remains at 0.0% across the extensive testing suite. While the randomized initial spawning configurations consistently forced the agents into difficult positions during the catch-up phase, the training of the follower agents also contributed to these results. Because the policy aggressively prioritizes rushing to the formation slots at the start of a run, the agents tend to take riskier paths around obstacles. This indicates that the collision penalty in the reward function should be increased in future training to force the agents to prioritize safety over speed even during initialization.

The average tracking errors across all 10 runs are 1.535 m for the left follower and 1.289 m for the right follower. These higher error values are heavily skewed by the initial distance the robots had to travel to reach their formation slots at the start of every run. Additionally, the average reward for the left agent (1606.11) is significantly higher than the right agent (8.79), which shows that across the different randomized layouts, the right follower was more frequently caught in obstacle safety zones during initialization. However, the overall behavior across all 10 runs matched our representative trial, where the fleet successfully established and maintained a stable triangle formation for the remainder of the simulation once the initial phase was completed.

Chapter 6

Conclusions, Limitations, and Future Work

This thesis presented the design, implementation, and evaluation of a robust continuous control framework for autonomous mobile robot navigation and formation control, combining a Deep Reinforcement Learning (DRL) agent with a temporal Long Short-Term Memory (LSTM) obstacle encoder. By successfully integrating this combined controller in simulation environments different from its training environment, we demonstrated its high capability and robustness. Before we further discuss these results, we must first explain the decision to choose the SAC architecture as the base for our combined controller.

The first part of this thesis focused on comparing three widely used DRL agents, DDPG, TD3, and SAC, to determine the right agent for our controller. To ensure a fair comparison, these agents were trained using identical hyper- and training-parameters, as well as identical reward functions. The only differences between the agents were the architecture-specific differences and any consequent parameters. The training profiles of each agent clearly showcased some of these architectural characteristics. The DDPG agent's training showed signs of overestimation bias; the TD3 and SAC agents showcased the pessimistic learning of the critic caused by the twin critic implementation.

After training, these agents were compared across the same 10 simulation runs, ensuring a clean comparison between the agents. From these simulations, the SAC agent came out the clear winner, boasting a 100% success rate for converging to $(0, 0)$ within a 0.1 m radius, further supported by the lowest final distance observed to the origin of 0.0895 m. The SAC agent also demonstrated the lowest average settling time, nearly twice as fast as the DDPG

agent and roughly 2.5 times faster than the TD3 agent. Additionally, the SAC agent achieved the highest average reward of 510.94, with the lowest standard deviation among the tested agents of 5.26. This highlights the consistent performance of the SAC architecture across the simulations and cements our choice for the base of our combined controller.

Now that the SAC architecture was chosen as our base, we focused our efforts on training the LSTM encoder. This Recurrent Neural Network was trained on a generated dataset of 20,000 samples via a supervised classification approach to provide the SAC agent with a latent representation of the surrounding obstacles. The training of the LSTM was conducted with the goal of being able to predict any potential collisions based on a spatial representation of the obstacles, using a simulated constant velocity and fixed simulation time. Now that the LSTM encoder was pre-trained, it could be implemented with an SAC agent to achieve our goal of mobile robot navigation with obstacle avoidance.

The combined SAC and LSTM controller was trained in Simulink with a refined reward function, adding reward components focused on static and dynamic obstacle avoidance, control effort, and overall navigation stability. The trained controller was then deployed in a modified environment, with changes made in obstacle velocity and size. The policy of this agent demonstrated its robustness by delivering a flawless success rate over 20 randomized evaluation episodes. Across these simulations, the agent showcased an average settling time of 11.3 s and an average distance to the origin of 0.055 m. These results justified our decision to combine the base SAC agent with the LSTM encoder to control our simulated mobile robot. This trained agent demonstrated a strong policy that could function as the leader agent in our Leader-Follower system for formation control. There are still some improvements that can be made to this agent, especially regarding the orientation while traveling towards the origin. We noticed that the agent traveled backwards in a number of simulation episodes, which can be improved by implementing reward components specifically focused on the robot's heading.

Utilizing this SAC + LSTM Leader agent, we started developing our controller to be capable of formation control. The first step to achieve this is to train a single follower agent capable of tracking a moving target, the leader. To train this follower, we deployed the previously obtained leader in a simulation environment to generate the moving target for our follower. The follower was also chosen to be a combined controller consisting of the SAC agent with the LSTM encoder, in order to have obstacle avoidance capabilities. The trained follower was tested across 20 simulation runs to demonstrate its performance, and achieved a 100% success rate in tracking the leader without any collisions, with an average error of 0.054m

to the target. The average settling time of the follower was 11.94 s, which is comparable to the leader's performance. On average, the follower needed an additional 0.64 s to accurately track the moving target and converge with the leader near the origin.

Now that both the leader and follower agents have been trained, we have successfully set up a single-leader single-follower system, incorporating the LSTM encoder to be capable of obstacle avoidance. This achieves the main target of this thesis, to design a DRL controller combined with an LSTM encoder capable of obstacle avoidance, but we wanted to push the limits of our controllers and work on multi-agent systems. We attempted to implement a 2-follower system to construct a triangle formation, but were not able to achieve satisfactory results. Across a 10-simulation window, our agents were not able to achieve a successful run, with the main issue being collisions with obstacles in the simulation environment. We have shown that our setup is capable of achieving very strong tracking of the leader, demonstrated by very low tracking errors of 0.058 m for one follower and 0.198 m for the other. From our results, we concluded that the main issue lies with the training of the follower agent. We believe that more importance can be placed on the reward components focused on obstacle avoidance, providing the agent with a stricter policy towards that goal.

In conclusion, we can say that we have laid the groundwork for a combined controller consisting of a DRL agent (SAC) and a Recurrent Neural Network (LSTM) capable of obstacle avoidance in a single-leader single-follower system. Future works built on this framework should be focused on refining the reward functions of the leader-follower system to be capable of formation control in any desired shape or form. Our agents demonstrated strong safety and robustness across most simulations, leading us to believe that the addition of the LSTM encoder was fully warranted and that further refinement of this system can lead to real-world deployment.

References

[1] B. Wang, *Cooperative Control for Heterogeneous Underactuated Autonomous Vehicle Networks*. Villanova University, 2022.

[2] B. Wang, S. G. Nersesov, and H. Ashrafiuon, "Time-varying formation control for heterogeneous planar underactuated multivehicle systems," *Journal of Dynamic Systems, Measurement, and Control*, vol. 144, no. 4, p. 041 006, 2022.

[3] T. Han and B. Wang, "Safety-critical stabilization of force-controlled nonholonomic mobile robots," *IEEE Control Systems Letters*, vol. 8, pp. 2469–2474, 2024.

[4] B. Wang, T. Han, and G. Wang, "Further results on safety-critical stabilization of force-controlled nonholonomic mobile robots," *ASME Letters in Dynamic Systems and Control*, vol. 6, no. 2, p. 021 011, 2026.

[5] B. Wang, S. G. Nersesov, and H. Ashrafiuon, "Robust formation control and obstacle avoidance for heterogeneous underactuated surface vessel networks," *IEEE Transactions on Control of Network Systems*, vol. 9, no. 1, pp. 125–137, 2022.

[6] P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, Jul. 1968, ISSN: 2168-2887. DOI: 10.1109/TSSC.1968.300136.

[7] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, Dec. 1959, ISSN: 0945-3245. DOI: 10.1007/BF01386390.

[8] L.-S. Schneider, J. Peng, and A. Maier, "Robot movement planning for obstacle avoidance using reinforcement learning," *Scientific Reports*, vol. 15, no. 1, p. 32 506, Sep. 2025, ISSN: 2045-2322. DOI: 10.1038/s41598-025-17740-5.

[9] J. Desai, J. Ostrowski, and V. Kumar, "Modeling and control of formations of nonholonomic mobile robots," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 6, pp. 905–908, Dec. 2001, ISSN: 2374-958X. DOI: 10.1109/70.976023.

[10] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. The MIT Press, 2020, vol. 9, ISBN: 978-0-262-03924-6. DOI: 10.1109/TNN.1998.712192.

[11] L. C. Garaffa, M. Basso, A. A. Konzen, and E. P. De Freitas, "Reinforcement Learning for Mobile Robotics Exploration: A Survey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 34, no. 8, pp. 3796–3810, Aug. 2023, ISSN: 2162-237X, 2162-2388. DOI: 10.1109/TNNLS.2021.3124466.

[12] H. Jiang, H. Wang, W.-Y. Yau, and K.-W. Wan, "A Brief Survey: Deep Reinforcement Learning in Mobile Robot Navigation," in *2020 15th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, Kristiansand, Norway: IEEE, Nov. 2020, pp. 592–597, ISBN: 978-1-7281-5169-4. DOI: 10.1109/ICIEA48937.2020.9248288.

[13] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement Learning: A Survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, May 1996, ISSN: 1076-9757. DOI: 10.1613/jair.301.

[14] Y. Li, *Deep Reinforcement Learning*, Oct. 2018. DOI: 10.48550/arXiv.1810.06339. arXiv: 1810.06339 [cs].

[15] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic Policy Gradient Algorithms," in *Proceedings of the 31st International Conference on Machine Learning*, PMLR, Jan. 2014, pp. 387–395.

[16] T. P. Lillicrap et al., "Continuous control with deep reinforcement learning," in *4th International Conference on Learning Representations (ICLR)*, ICLR, 2016. DOI: 10.48550/arXiv.1509.02971. arXiv: 1509.02971 [cs].

[17] O. Bouhamed, H. Ghazzai, H. Besbes, and Y. Massoud, "Autonomous UAV Navigation: A DDPG-Based Deep Reinforcement Learning Approach," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, Oct. 2020, pp. 1–5. DOI: 10.1109/ISCAS45731.2020.9181245.

[18] J. C. Jesus, J. A. Bottega, M. A. S. L. Cuadros, and D. F. T. Gamarra, "Deep Deterministic Policy Gradient for Navigation of Mobile Robots in Simulated Environments," in *2019 19th International Conference on Advanced Robotics (ICAR)*, IEEE, Dec. 2019, pp. 362–367. DOI: 10.1109/ICAR46387.2019.8981638.

[19] M. Park, S. Y. Lee, J. S. Hong, and N. K. Kwon, "Deep Deterministic Policy Gradient-Based Autonomous Driving for Mobile Robots in Sparse Reward Environments," *Sensors*, vol. 22, no. 24, p. 9574, Jan. 2022, ISSN: 1424-8220. DOI: 10.3390/s22249574.

[20] S. Fujimoto, H. Hoof, and D. Meger, "Addressing Function Approximation Error in Actor-Critic Methods," in *Proceedings of the 35th International Conference on Machine Learning*, PMLR, Jul. 2018, pp. 1587–1596. DOI: 10.48550/arXiv.1802.09477. arXiv: 1802.09477 [cs].

[21] M. Abdallah, M. Lashin, A. El-Awamry, and W. I. Gabr, "Reinforcement Learning-Based Optimal Path Planning for Mobile Robot with Obstacles Avoidance," in *2024 12th International Japan-Africa Conference on Electronics, Communications, and Computations (JAC-ECC)*, Dec. 2024, pp. 1–6. DOI: 10.1109/JAC-ECC64419.2024.11061237.

[22] A. J. Soriano, "Deep Reinforcement Learning for Autonomous Mobile Robot Navigation: Comparing the Performance of DQN, DDPG, and TD3 Algorithms," in *2025 4th International Conference on Electronics Representation and Algorithm (ICERA)*, Jun. 2025, pp. 1–6. DOI: 10.1109/ICERA66156.2025.11087316.

[23] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor," in *Proceedings of the 35th International Conference on Machine Learning*, PMLR, Jul. 2018, pp. 1861–1870. arXiv: 1801.01290 [cs].

[24] J. Choi, G. Lee, and C. Lee, "Reinforcement learning-based dynamic obstacle avoidance and integration of path planning," *Intelligent Service Robotics*, vol. 14, no. 5, pp. 663–677, Nov. 2021, ISSN: 1861-2784. DOI: 10.1007/s11370-021-00387-2.

[25] Z. Zhang, Y. Luo, Y. Chen, H. Zhao, Z. Ma, and H. Liu, "Automated Parking Trajectory Generation Using Deep Reinforcement Learning," in *2025 6th International Conference on Computer Engineering and Application (ICCEA)*, Apr. 2025, pp. 516–520. DOI: 10.1109/ICCEA65460.2025.11103087.

[26] X. Gao, L. Yan, Z. Li, G. Wang, and I.-M. Chen, "Improved Deep Deterministic Policy Gradient for Dynamic Obstacle Avoidance of Mobile Robot," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 53, no. 6, pp. 3675–3682, Jun. 2023, ISSN: 2168-2232. DOI: 10.1109/TSMC.2022.3230666.

[27] R. S. Nair and P. Supriya, "Robotic Path Planning Using Recurrent Neural Networks," in *2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, Jul. 2020, pp. 1–5. DOI: 10.1109/ICCCNT49239.2020.9225479.