School of Computer Science and Engineering

Faculty of Engineering

The University of New South Wales

Evaluating Reinforcement Learning on Nao Robot Behavior Skills for RoboCUP SPL

by

Srideep Saha

Thesis submitted as a requirement for the degree of

Master of Information Technology

Submitted: August 15 2025

Supervisor: Prof. Naranjo, Prof. Sammut, Prof. Bamdad(Assessor)

Student ID: z5527420

Abstract

The document investigates single-agent and multi-agent reinforcement learning approaches for ball interception in a simulated RoboCUP environment with future plans of deploying trained behaviours on real Nao robots. Custom training setup were developed for single agent and multi-agent environments to ensure realistic, interceptible ball trajectories, with carefully designed rewards to balance speed, stability, and accuracy. Policies were trained using Proximal Policy Optimization (PPO) and evaluated on metrics such as success rate, episode length, and reward stability. Results show that both setups achieve high interception success, with multi-agent policies converging to complementary roles albeit with limitations like overshooting which can lead to reduced performance. Practical considerations for real Nao robot deployment, including computation limits, communication constraints, and methods to overcome limited visibility in real world setting are discussed.

Acknowledgements

This work has been inspired by the labours of numerous academics in the Faculty of Engineering at UNSW who have endeavoured, over the years, to encourage students to present beautiful concepts using beautiful typography.

Further inspiration has come from Donald Knuth who designed TEX, for typesetting technical (and non-technical) material with elegance and clarity; and from Leslie Lamport who contributed LATEX, which makes TEX usable by mortal engineers.

John Zaitseff, an honours student in CSE at the time, created the first version of the UNSW Thesis LATEX class and the author of the current version is indebted to his work.

The author also thanks the supervisors and the UNSW RoboCUP rUNSWift team for their guidance and opportunity to undertake this project.

Abbreviations

RL Reinforcement Learning

SB3 Stable Baselines3

PPO Proximal Policy Optimization

RLlib Raylib Reinforcement Learning Library

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

RoboCUP is an international research initiative that aims to advance the state of the art in intelligent autonomous robotics through soccer matches. Among its many leagues, the RoboCUP 3D Simulation League provides a realistic, physics-based environment powered by the SimSpark simulator while the RoboCUP Standard Platform League provides teams the opportunity to develop code and deploy it on real Nao robots in a competitive soccer environment.

Reinforcement learning (RL) has emerged as a powerful tool for training agents to perform complex tasks by trial and error, guided by reward signals. Policy gradient methods, such as Proximal Policy Optimization (PPO), have been particularly successful in continuous control problems such as locomotion. In the RoboCUP setting, RL enables agents to learn skills such as interception, dribbling, and shooting directly from their sensor observations. In addition, RL presents the possibility of a noticeable improvement over rule-based behaviours which have high precision and low recall as they do not work well in every situation. In a competitive environment where even the smallest advantage can mean the difference between victory and defeat, RL can prove very beneficial if utilized properly.

1.2 Problem Statement

This work focuses on the *midfielder interception* skill. In soccer, midfielders play both offensive and defensive roles as the situation demands. In this study, a simplified version of the midfielder skill is considered, where the robot moves laterally (left or right) to intercept a moving ball in both single-agent and multi-agent settings. There is no intentional forward or backward movement relative to the ball; however, drift can sometimes cause the robot to unintentionally move forward or backward.

In the single-agent environment, we train one robot to intercept the moving ball by sidestepping left or right. In the multi-agent environment, we train two robots to intercept a moving ball over a larger region by encouraging cooperation over competition.

In the simulation, actions are updated at every step without constraints, but when transferring to a real Nao robot (Sim2Real), challenges arise:

- **Hysteresis and Jitter:** Rapidly changing actions can cause physical instability in the robot's movement.
- **Partial Observability:** Real robots have limited field of view and cannot always see the ball.
- **Communication Limits:** RoboCUP rules restrict inter-robot communication, limiting policies requiring frequent sharing of observations and states between robots.
- **Multi-Agent Deadlocks:** Synchronization issues in SimSpark can cause the simulation to freeze if not handled correctly.

Specifically, the contributions of this work are:

- Implementation of PPO-based training for the interception skill in both single-agent and multi-agent settings.
- Resolution of synchronisation-related deadlocks in multi-agent SimSpark training.

- Development of deployment strategies to mitigate hysteresis, handle limited ball visibility, and comply with communication restrictions.
- Evaluation of learned policies over multiple iterations and analyzing edge cases where model fails to demonstrate intended behaviour.
- Analysis of differences between simulation and real-world feasibility.

The remainder of this thesis is organised as follows: Chapter 2 reviews the relevant literature in detail and outlines past work in RoboCUP simulation, RL, and Sim2Real transfer. Chapter 3 details the methodology, including environment setup, reward shaping, and training configurations. Chapter 4 presents the experimental results, followed by discussion. Chapter 5 concludes the thesis and outlines directions for future work.

Chapter 2

Literature Review

2.1 Previous work and Limitations

Previous work in this field has explored reinforcement learning (RL) for specific scenarios. One such example is cooperative multi-agent deep RL in a 2v2 free-kick task [ORR19] which showcases the difference in training between join-action learners (shared observations between agents) and independent learners (independent observations) by using the B-Human SimRobot simulator [R$^+$17]. Another example is the codebase we use for this project: FC Portugal RoboCUP 3D Simulation League Agent [ARL25], which uses RL to train low-level motions such as walking, dribbling, running, sprinting, and jumping. While these studies and other similar work demonstrate the potential of RL for targeted behaviours, they are often restricted in training low-level locomotion skills and do not address broader team-wide coordination or direct transfer to physical robots. This research instead focuses on training high-level behavioural skills, such as the midfielder interception role, to develop better team strategies and foster cooperation between teammates.

Currently, the RoboCUP SPL rUNSWift team's behaviour module relies on rule-based methods. These approaches determine actions based on predefined logic (IF-ELSE LOOPS) using the outputs from vision, localization, and other modules. While ef-

fective under normal conditions, rule-based systems are often inflexible, difficult to extend to every possible edge case, and can become overly complex when handling diverse scenarios. Adding more rules increases debugging difficulty without guaranteeing substantial performance gains.

In contrast, RL enables agents to learn policies that maximize long-term rewards and enables robots to respond to dynamic changing environments in a robust manner. However, existing RL applications in RoboCUP rarely address the challenges of sim-to-real transfer. Factors such as imperfect vision, localization drift, hardware faults, and noisy real-world conditions can significantly degrade performance when moving from simulation to real Nao robots. Addressing these gaps, improving adaptability, and ensuring robust transfer to real-world play remains an open challenge.

2.2 RoboCUP

RoboCUP is an international scientific initiative launched in 1997 with an ambitious goal to produce a team of fully autonomous robots capable of beating the reigning FIFA World Cup champions under official rules. [Rob97] RoboCUP consists of the following leagues which are relevant to the research work done in this paper.

2.2.1 RoboCUP Standard Platform League

RoboCUP Standard Platform League (RoboCUP SPL) is a soccer league where all teams participate using identical robot platform. Currently this is the Nao robot designed by Aldebaran.

All SPL matches are autonomous and there is no human teleoperation during gameplay. Robots run on team-developed software using the Robot Operating System (ROS 2) architecture to power different modules such as:

- **Computer Vision:** Detecting and tracking the ball, field lines and other robots

- **Localisation:** Identifying the position and orientation of robots relative to the field
- **Motion:** Generating efficient and collision free movements while walking, turning, kicking, sprinting, etc.
- **Communication:** Exchanging critical information like opponent positions, ball positions, strategy among teammates.
- **Behaviours:** Serves as high level decision making system of the robot which integrates information from all previous modules, interprets the current game situation and determines the actions of each robot.

2.2.2 RoboCUP Simulation 3D League

The RoboCUP Simulation 3D League is a virtual soccer competition where teams control fully autonomous robots in a realistic physics based environment. Matches are played in the SimSpark simulator which models the Aldebaran Nao platform and incorporates elements such as rigid body dynamics, joint constraints, collision and noise perception. Teams create agents that communicate with the SimSpark server by sending motor commands and receiving simulated sensor data such as vision, gyroscope and joint position feedback. Agents send SYN packets containing motor commands to the server and receive SYN/ACK packets containing simulation information. This league allows researchers to test and train behaviours and logic without involving any physical robots. Because the simulation closely mirrors real world physics it serves as an excellent platform to train complex behaviours and algorithms before transferring them to actual robots.

2.3 Reinforcement Learning

Reinforcement Learning (RL) is a paradigm of machine learning in which an agent learns to make sequential decisions by interacting with an environment and receiving

scalar rewards [SB18]. Unlike supervised learning, where the correct outputs are provided for each input, RL relies on trial-and-error search and delayed reward signals to guide learning. The interaction loop is typically formalized through a Markov Decision Process (MDP).

2.3.1 Markov Decision Process

An MDP [Put14] is defined by the tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$, where:

- $\mathcal{S}$: the set of possible states,
- $\mathcal{A}$: the set of possible actions,
- $P(s' \mid s, a)$: the transition probability function,
- $R(s, a)$: the expected reward for taking action a in state s,
- $\gamma \in [0, 1]$: the discount factor controlling the weight of future rewards.

At each timestep t, the agent observes state $s_t \in \mathcal{S}$ and based on that observation, selects an action $a_t \in \mathcal{A}$, receives a reward $r_t \in \mathbb{R}$, and transitions to a new state s_{t+1} sampled from $P(\cdot \mid s_t, a_t)$. The objective is to maximize the expected discounted return:

$$J(\pi) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t \right],$$

where $\pi(a|s)$ is the policy, i.e., the probability of selecting action a in state s. A key thing to note is that MDP only looks at the current state and observation to determine the action and not past states.

2.3.2 Policy Gradient Methods

Policy gradient methods parameterize the policy $\pi_\theta(a|s)$ with parameters θ (e.g., neural network weights) and adjust θ to maximize the objective function $J(\pi_\theta)$ [SMSM00]. The

gradient of the objective function with respect to θ is given by the formula:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a_t|s_t)\, \hat{A}_t\right],$$

where $\hat{A}_t$ is an estimator of the advantage function $A(s_t, a_t)$. $\hat{A}_t$ represents how much better action a_t is compared to the average action at s_t. This formulation serves as the foundation for actor–critic methods. Policy gradient methods are well-suited for high-dimensional and continuous action spaces such as the RoboCUP setting, as they allow bypassing the need to test every single action. In this paper, we use policy gradient method on discrete action space.

2.3.3 On-Policy vs. Off-Policy Learning

Reinforcement learning algorithms are often categorized as *on-policy* or *off-policy* [SB18]. In on-policy methods, the data used to update the policy is collected by following the *current* policy which may be slightly modified due to exploration noise. This ensures that updates are directly aligned with the policy's current behaviour, but the drawback is that previously collected data can become stale and cannot be reused for long.

In off-policy methods, the learning algorithm can leverage experience generated by a different behaviour policy from the one being optimized. This allows the current policy to learn from experience collected by the same or a different policy on past data which facilitates varied and diverse sampling. In this work, PPO — an on-policy method — was chosen for its stability in control and multi-agent settings, despite the higher sample requirements.

2.3.4 Actor–Critic Methods

Actor–critic algorithms combine the strengths of *policy-based* and *value-based* methods [KT00]. The **actor** is the policy $\pi_\theta(a|s)$ that selects actions given states, while the **critic** is a value function $V_w(s)$ (or action-value function $Q_w(s, a)$) that estimates expected returns under the current policy.

The critic's role is to provide a low-variance, learned baseline for the policy gradient update. The actor updates its parameters in the direction suggested by the critic's advantage estimates, while the critic updates its parameters to minimize the temporal-difference (TD) error:

$$\delta_t = r_t + \gamma V_w(s_{t+1}) - V_w(s_t).$$

This bidirectional interaction stabilizes training and accelerates convergence. In this project, PPO's clipped-actor–critic formulation is used, with a shared policy across agents in the multi-agent case.

2.3.5 Proximal Policy Optimization

Proximal Policy Optimization (PPO) [SWD+17] is a commonly used on-policy actor–critic method designed to improve the stability and efficiency of policy gradient updates. PPO introduces a clipped surrogate objective:

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right],$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the probability ratio between the new and old policies, and ϵ is a small hyperparameter (commonly 0.1–0.3). This objective discourages large, destabilizing updates by clipping the ratio and making sure the new policy does not deviate too far from the old one in a single step. PPO is efficient and easy to implement, making it a common default choice in modern RL applications.

2.4 Multi-Agent Reinforcement Learning

Multi-Agent Reinforcement Learning (MARL) extends the single agent RL framework to settings with multiple agents that may cooperate, compete, or both. Each agent i interacts with the environment according to its own policy π_i and receives (possibly distinct) observations and rewards. The joint system can be modeled as a stochastic game, generalizing the MDP to multiple decision-makers.

In the cooperative RoboCUP setting studied here, both agents play the role of midfielders on the same team. Their shared goal is to cooperate and showcase a team strategy where the closest robot can intercept the incoming ball and farthest agent stands in its original position to maintain defensive coverage in its region. The work employs a parameter sharing approach in which both agents are controlled by a single shared-policy PPO network. This shared policy promotes role differentiation into closest and farthest midfielders. It also ensures that each agent can take action solely based on its own local observation without having to rely on communication with its fellow teammate.

2.5 Tools and Software

2.5.1 SimSpark

SimSpark [HR06] is an open-source, physical multi-agent simulation framework, widely used for research in robotics and artificial intelligence. It provides a 3D physics environment based on the Open Dynamics Engine (ODE) and supports both rigid-body dynamics and articulated figures. SimSpark is the official simulation platform for the RoboCUP 3D Soccer Simulation League, offering network-based agent-server communication as well as enabling TCP connection with monitors for seamless visualization of the 3D environment. Some of the key features are realistic rigid-body dynamics and collision handling, extensible server-client architecture, and support for both single and multi-agent with synchronous and asynchronous simulation steps. In this paper, SimSpark is used to simulate the robots(agents) and the ball in a soccer setting equivalent to the RoboCUP 3D Soccer Simulation League.

2.5.2 RoboViz

RoboViz is a monitoring and visualization tool for the RoboCUP SPL. The tool is built by Java and offers a real time interactive 3D visualization of both agent and environment state. [SV12] The tools also enables programmable drawing which allows

teams to visualize by overlaying custom drawn lines and circles on the screen, helping greatly in the development process.

2.5.3 Stable Baselines3

Stable Baselines3 (SB3) [RHG+21] is an open-source `PyTorch` library providing reliable implementations of popular RL algorithms, including PPO, A2C, DDPG, TD3, and SAC. SB3 offers a unified API, reproducible training, and seamless integration with OpenAI Gym environments, making it widely adopted in both research and applied RL.

2.5.4 Raylib

Raylib (RLlib) [LLM+18] is a scalable RL library built on the `Ray` distributed computing framework [MNW+18], supporting single- and multi-agent systems with minimal code changes. It offers distributed rollouts, native multi-agent environments, and integration with `TensorFlow` and `PyTorch`. In this work, RLlib's multi-agent API is used to implement a multi-agent shared policy PPO with role-based reward shaping, leveraging Ray's native multi-agent setup and parallelism to deploy training environments across multiple SimSpark servers.

Chapter 3

Methodology

3.1 Simulation Setup

Two different types of simulations were considered for the purpose of training the midfielder robot. First was the FCP Portugal Codebase [ARL25] for RoboCUP 3D Simulation League while the second was Cyrus2D for RoboCUP 2D Simulation League [ZAS+22]. While both FCP Portugal Codebase and Cyrus2D codebase offered ways to train reinforcement learning on robots in simulation, the 3D architecture and the physics based environment of the FCP Codebase combined with accurate modeling of robot dynamics, collisions and similarity to the SPL team rUNSWift codebase made it the preferred choice to train RL behaviour skills.

3.2 FCP Agent Architecture

The FCP Agent architecture is organized into three modular layers integrating perception, decision making and motion control. The agent communicates with the SimSpark server via TCP. It receives noisy sensor updates at fixed intervals and sends corresponding action commands like joint torques, walking gait parameters or kick motions. The

perception layer allows the robot to receive and interpret a continuously updated world model comprising of ball position, robot position and robot posture among others.

The codebase comes with a pre-trained Walk skill which is utilized by robots to sidestep left or right during training. It also features a pre-built Stable Baselines3 wrapper with Proximal Policy Optimization(PPO) algorithm implementation. Some examples of Reinforcement Learning on low-level motor skills like walking, dribbling and running by training robot joints are also provided which served as a solid foundation to start training higher-level Midfielder behaviour for this research project.

The architecture of the setup is given in Figure 3.1:

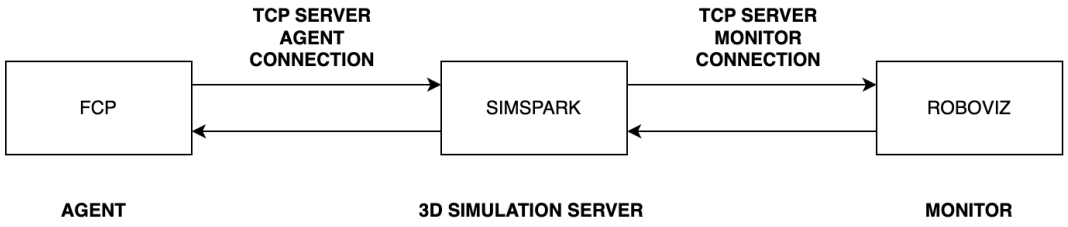

Figure 3.1: **Architecture of the FCP codebase**. The agent (FCP codebase) communicates with the 3D simulation server (SimSpark) via TCP, while Roboviz connects to the server port for real-time visualization and monitoring.

3.3 Server Settings

For RL training, the simulation server was configured with custom settings. Soccer rules and synchronous mode were enabled to ensure fair play and controlled agent–server communication. Cheats were turned on to allow direct access to agent and ball positions for training at all times. Noise was added to observations to mimic real-world data. Real-time mode was disabled to speed up the training process. The monitor was set to 25Hz for consistent visualization, and penalty shootout mode was turned off.

3.4 Single Agent Training Setup

The training is carried out by utilizing the built-in Stable-Baselines3 wrapper.

3.4.1 Environment Setup

Our custom environment class `InterceptionEnv` inherits from the Gym API and connects a single robot agent to a SimSpark server (FCP TCP) with RoboViz.

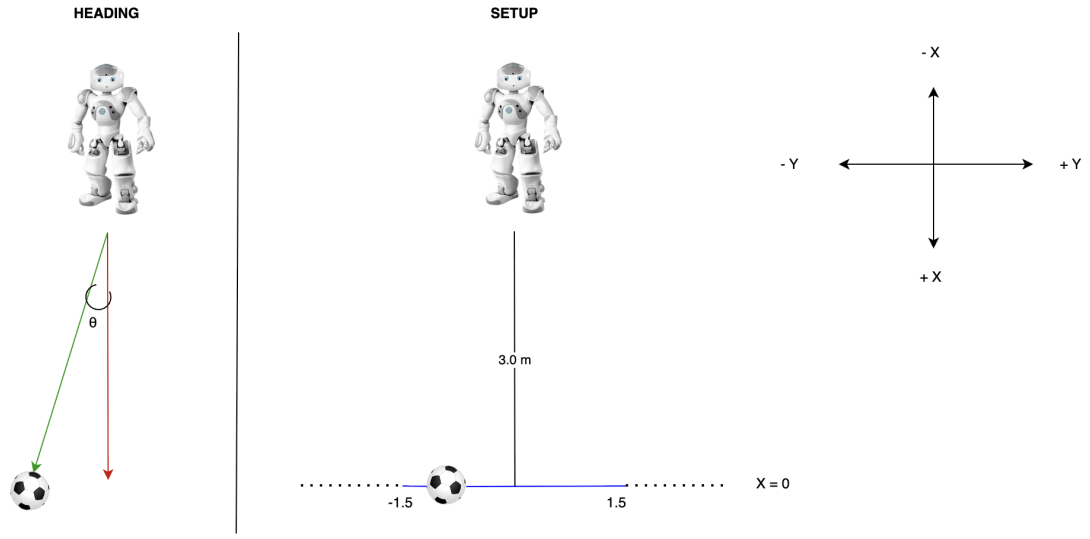

Figure 3.2: **Single-agent interception setup.** Left: Heading Offset Calculation Right: Episode start configuration

Teleport Routine.

At the start of each episode, the robot is positioned at $(x, y) = (-3.0,\ 0.0)$ in a stable posture. The ball is spawned at a random lateral position $y \in [-1.5,\ 1.5]$ along the center line ($x = 0$), as shown by the blue segment in Fig. 3.2. It is then launched toward the robot with

$$v_x = -4.0 \text{ m/s}, \qquad v_y \sim \mathcal{U}\big(-1.5 - y,\ 1.5 - y\big),$$

where v_x drives the ball towards the robot ($-x$ direction) and v_y controls its lateral motion. By making v_y depend on the ball spawn position y, the ball always drifts *inward* toward $y = 0$ when starting near the extremes ($y \approx -1.5$ or $y \approx 1.5$), improving the chance of interception. For example, if $y \approx -1.5$, then $v_y \in [0, 3]$; if $y \approx 1.5$, then $v_y \in [-3, 0]$. Without this dependency, the ball could drift outward, making realistic interception far less likely. This setup ensures the ball reliably approaches the robot along an interceptible path.

Heading Offset Calculation.

At every timestep, the heading offset θ measures the angular difference between the robot's forward axis (red arrow) and the vector from the robot to the ball (green arrow), as shown in Fig. 3.2. It is computed as

$$\theta = \text{atan2}\left(y_{\text{ball}} - y_{\text{robot}},\ x_{\text{ball}} - x_{\text{robot}}\right),$$

with the result wrapped to $(-\pi, \pi]$. Since this is expressed in the robot's relative coordinate frame, $x_{\text{robot}} = 0$ and $y_{\text{robot}} = 0$. The heading offset is a key component of the robot's observation space in both the single-agent and multi-agent setups.

Observation space.

A 6D vector:

$$[o_x,\ o_y,\ d,\ \theta_{\text{ball}},\ v_x^{\text{ball}},\ v_y^{\text{ball}}],$$

where (o_x, o_y) is the ball position relative to the torso, $d = \sqrt{o_x^2 + o_y^2}$ is the relative distance, $\theta_{\text{ball}} = \text{atan2}(o_y, o_x)$ is the ball heading in the robot's frame, and $(v_x^{\text{ball}}, v_y^{\text{ball}})$ is the ball's absolute planar velocity.

Action space.

A discrete set $\mathcal{A} = \{0, 1, 2, 3, 4\}$ mapped to a sidestep controller:

$$0 : \text{sidestep left (fast)} \quad [0,\ 5.0],\ \text{speed} = 0.5$$

$$1 : \text{sidestep right (fast)} \quad [0,\ -5.0],\ \text{speed} = 0.5$$

$$2 : \text{stand} \quad [0,\ 0.0],\ \text{speed} = 0.5$$

$$3 : \text{sidestep left (slow)} \quad [0,\ 5.0],\ \text{speed} = 0.1$$

$$4 : \text{sidestep right (slow)} \quad [0,\ -5.0],\ \text{speed} = 0.1$$

3.4.2 Synchronized Steps

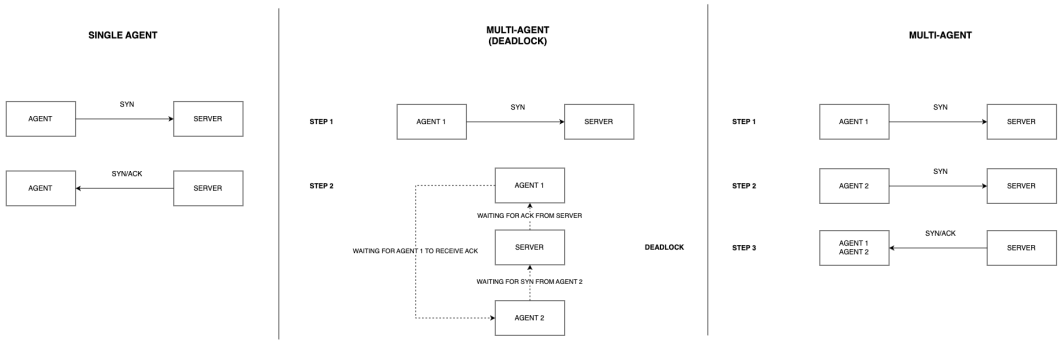

Figure 3.3: **Illustration of agent–server synchronization.** Left: **Single-agent** handshake sequence between an agent and the server. Middle: **Multi-agent deadlock** scenario where Agent 1 and Agent 2 block each other due to sequential waits for acknowledgments. Right: **Fixed multi-agent** handshake in which both agents synchronize with the server without causing deadlock.

Each step, the agent issues a motion command and then performs a single `commit_and_send` → `receive` cycle (`sync()` in code) as seen in Figure 3.3 This matches SimSpark's sync mode and prevents freezes by ensuring the server processes the robot's command before advancing physics.

3.4.3 Reward Function

The per-step reward uses dense shaping system to promote fast, stable interceptions and discourage aimless motion.

Core shaping.

Let d be the robot–ball distance and let θ be the heading offset. The base reward is

$$r_{\text{shape}} = 0.6 \cdot (-d) + 0.4 \cdot (-|\theta|).$$

Early motion restraint (first 25 steps).

If the action is not *stand* during the first 25 steps, add -0.2 (discourages immediate movement while the ball approaches to prevent overshoot which occurs when the agent crosses the optimal interception point and has to readjust itself).

Per-step time cost.

Add a constant -0.4 each step to encourage quick interception.

Progress bonus every 10 steps.

Every 10 steps compare d to the stored best distance to the ball:

$$\begin{cases} +0.3 & \text{if } d \leq \text{best\_so\_far (getting closer)}, \\ -0.25 & \text{if } d \text{ increased (moving away)}. \end{cases}$$

Near-ball lateral discipline.

If $d \leq 2.0$ and $|o_y| > 0.15$ where $|o_y|$ is the lateral offset of the ball, add -0.25 (penalize excessive lateral offset when close).

Terminal logic (success/failure).

- **Success:** If $d \leq 0.8$ and $|\theta| \leq 0.03$:
  - If the action is *stand* (i.e., stopped in front of the ball), add $+5.0$ and terminate.
  - Otherwise add -2.5 (arrived but still moving—encourages standing to complete the intercept).
- **Timeout:** If the step limit (500) is reached and $|\theta| > 0.03$, add -5.0 and terminate.

The timestep reward for the agent is:

$$R = \begin{cases} 0.6(-d) + 0.4(-|\theta|) & \text{base shaping reward,} \\ -0.2 & \text{if steps} < 25 \text{ and moving,} \\ -0.4 & \text{constant timestep penalty,} \\ +0.3 \text{ (or } -0.25) & \text{every 10 steps if d} < \text{last\_stored\_distance,} \\ -0.25 & \text{if } d \leq 2.0 \text{ and } |o_y| > 0.15, \\ \mathbf{+5.0} & \text{if } d \leq 0.8,\ |\theta| \leq 0.03, \text{ and action = stand (success),} \\ \mathbf{-2.5} & \text{if } d \leq 0.8,\ |\theta| \leq 0.03, \text{ and action} \neq \text{stand (arrived but moving),} \\ \mathbf{-5.0} & \text{if step limit (500) reached and } |\theta| > 0.03 \text{ (timeout failure).} \end{cases}$$

where d is the distance to the ball, o_y is the lateral offset from the ball, and rewards in **bold** are given only when the episode ends due to success or failure.

Hyperparameters

We train the PPO policy with SB3. Key settings are listed below:

Parameter	**Value**
Algorithm	PPO (Stable-Baselines3)
Policy	`MlpPolicy` ($[64, 64]$ for π and V)
Vectorized environments	$n_{\text{envs}} = 10$ (`SubprocVecEnv`)
Rollout length per env	$n_{\text{steps}} = 512 \Rightarrow$ rollout batch $= 5120$
Minibatch size	64
Epochs per update	10
Learning rate	3×10^{-4}
Discount γ	0.99
GAE λ	0.95
Clip range	0.20
Total timesteps	4.0×10^5

Table 3.1: PPO hyperparameters for the single-agent setup.

SB3 updates on fixed-length rollouts (not necessarily complete episodes), so every update uses exactly 5120 steps regardless of episode boundaries. Our custom callbacks add success-rate tracking and richer per-rollout statistics that include unfinished episodes.

3.5 Multi-Agent Training Setup

While the FCP Codebase's built-in Stable Baselines3 wrapper with some modifications provides a solid starting point to train single agent behaviour skills by PPO, it does not offer support for multi-agent policies which is a drawback as in RoboCUP coordination between agents is crucial and can be the difference between winning and losing.

3.5.1 RLlib Wrapper

To overcome the limitation mentioned above, the FCP environment was integrated with Ray RLlib, an industry-grade scalable reinforcement learning library which provides out of the box support for multi-agent reinforcement learning. The custom wrapper built from scratch was also optimized and scaled to run in parallel across multiple SimSpark servers to speed up the training process. The RLlib architecture is shown in Figure 3.4

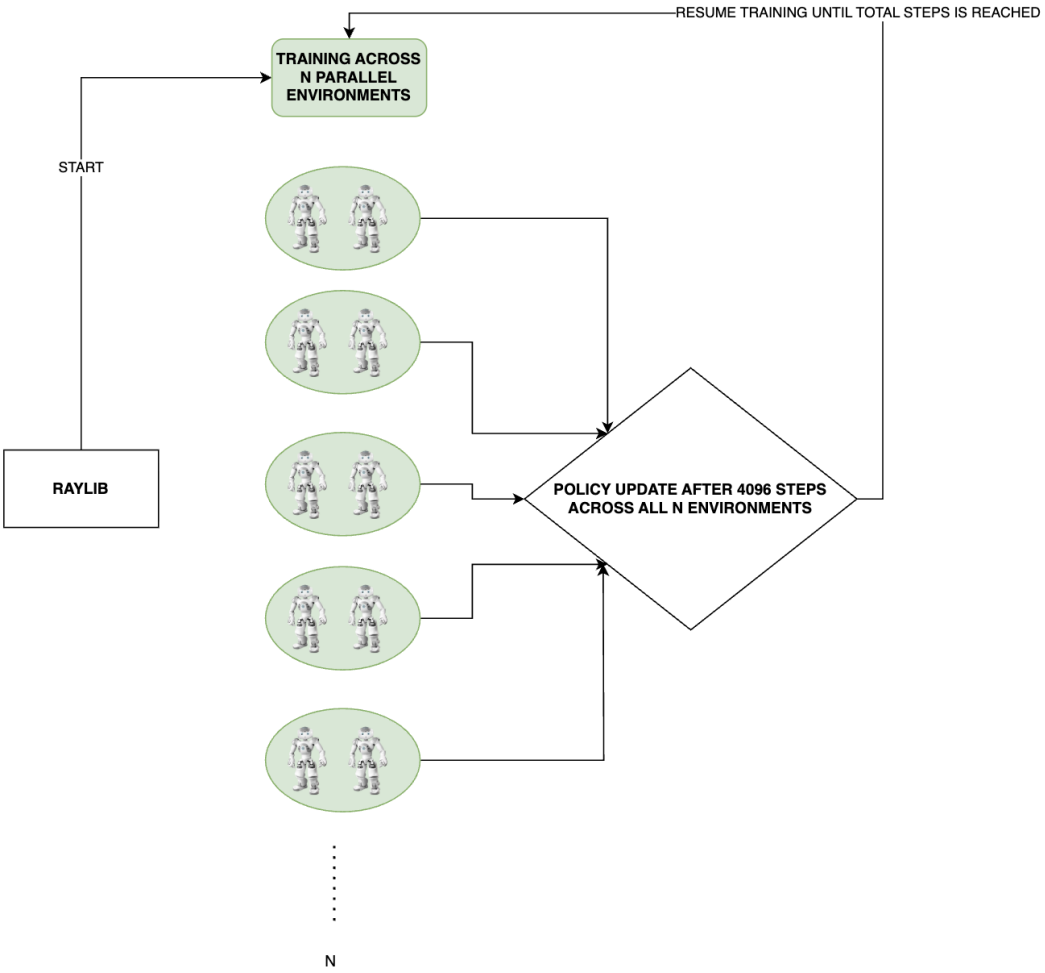

Figure 3.4: **RLlib Training Setup** with N parallel environments.

The following sections outline the features of this wrapper.

3.5.2 Environment Setup

Our custom environment class MultiAgentSharedWorld inherits from RLlib's deafult MultiAgentEnv and instantiates two robot agents. Both agents are bound to a single SimSpark server via the FCP TCP interface (agent port + RoboViz port). As seen in Figure 3.4, for parallel training, multiple environments similar to the above are run to speed up the training process.

Heading Offset (per agent)

Similar to the single-agent setup, the heading offset at every timestep is given by

$$\theta_i = \text{atan2}\big(y_{\text{ball}} - y_{\text{robot},i},\ x_{\text{ball}} - x_{\text{robot},i}\big),$$

wrapped to $(-\pi, \pi]$. Since positions are expressed in each robot's relative coordinate frame, $x_{\text{robot},i} = 0$ and $y_{\text{robot},i} = 0$. This value is part of each agent's observation.

Observation space (per agent)

A 3D vector:

$$[o_x^{(i)},\ o_y^{(i)},\ \theta_{\text{ball}}^{(i)}],$$

where $(o_x^{(i)}, o_y^{(i)})$ is the ball position relative to agent i's torso frame and $\theta_{\text{ball}}^{(i)} = \text{atan2}\big(o_y^{(i)}, o_x^{(i)}\big)$ is the ball heading in that frame.

Action space (per agent)

A discrete set $\mathcal{A} = \{0, 1, 2\}$ mapped to a sidestep controller:

$$0: \text{ sidestep right} \quad [0,\ -0.5],$$

$$1: \text{ sidestep left} \quad [0,\ +0.5],$$

$$2: \text{ stand} \quad [0,\ 0.0].$$

Policy and coordination.

A *shared* policy is used for both agents (same weights); coordination emerges from the closest/farthest shaping and penalties, without inter-agent communication.

3.5.3 Synchronized Steps

To match the SimSpark synchronization mode in the `step()` method of our wrapper, every agent sends its joint targets to the server and only then performs a single receive

pass over all the agents. If this is not done, a deadlock occurs where an agent waits for the server to send the "syn/ack" packet while the server waits for other agents to send their "syn" packets. This happens because, in synchronization mode, the server must receive all "syn" packets from all agents before sending back the acknowledgement. The situation is illustrated in Figure 3.3. A custom method integrated into the the wrapper allows the server to collect the "syn" packets from all agents before sending back the "syn/ack" packets, which keeps the simulation running smoothly without freezing.

3.5.4 Episode Resets

Much like the single agent setup, to maintain consistency, at the start of every episode, our wrapper teleports the agents to hover above the center of the field and finally beams(teleports) them to the ground. This prevents unstable robot behaviour and falling to the ground as soon as the episode starts.

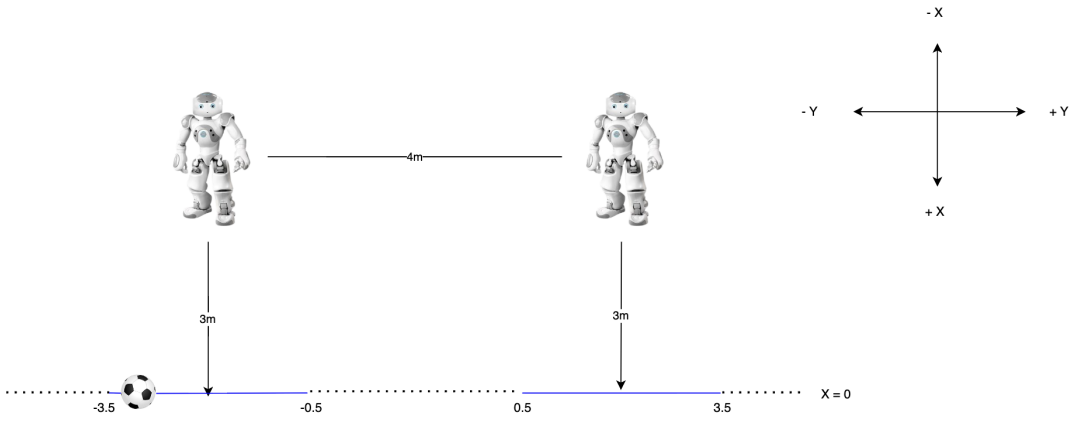

Figure 3.5: **Multi Agent Midfielder** Episode Start Configuration

Figure 3.5 illustrates the environment configuration at the start of each episode. The blue line segments on the center line defined by $x = 0$ represent the possible ball spawn zones. The two midfielder robots are positioned 3 m into our half ($x = -3.0$) and are laterally separated by 4 m to minimize the risk of unnecessary collisions.

At the start of each episode, the ball is randomly placed on the center line ($x = 0$)

within one of two **blue lateral bands** shown in Figure 3.5:

- **Right band:** $y \in [-3.5, -0.5]$
- **Left band:** $y \in [0.5, 3.5]$

In this image, the bands appear mirrored from our viewpoint. However, from the robot's perspective, since all observations are relative to it, the band on our left in Figure 3.5 is actually its right band, and has been classified above accordingly.

The ball is then launched toward the robots with an initial velocity vector (v_x, v_y). The velocity of the ball consists of the following components:

Longitudinal component (v_x) This component is fixed at $v_x = -4.0$ m/s, guaranteeing that the ball always travels from the center line toward the robots.

Lateral component (v_y) The lateral speed is determined by the ball's starting y-coordinate y and the band it is in:

- If y is in the **Right Band** $[-3.5, -0.5]$,

$$v_y \sim \mathcal{U}[-3.5 - y,\ -0.5 - y]$$

  For example, near the outer edge ($y \approx -3.5$), $v_y \in [0, 3]$, forcing the ball inward toward the gap between robots.

- If y is in the **Left Band** $[0.5, 3.5]$,

$$v_y \sim \mathcal{U}[0.5 - y,\ 3.5 - y]$$

  For example, near the outer edge ($y \approx 3.5$), $v_y \in [-3, 0]$, again biasing the ball inward.

Without these constraints, the ball could travel in a direction beyond the reach of either robot. The applied limitations ensure that the ball spawns within one of two designated

bands, with a random probability and an initial velocity directed toward the robots, giving the midfielders a realistic chance to intercept it.

3.5.5 Multi-Agent Reward Function Design

The reward function in our custom Ray RLlib wrapper is designed to train two cooperating midfield robots to intercept a moving ball without unnecessary movement or collisions. It combines **sparse terminal rewards** for clear success/failure signals and **dense shaping rewards** to guide agents during play.

Sparse Terminal Rewards

Terminal rewards are given only upon episode termination or success:

- **Successful Interception:** The "closest" agent (determined dynamically each timestep by smallest ball-heading offset) receives a +0.9 reward if it reaches the ball with both:

  $$\text{distance} < 0.8 \quad \text{and} \quad |\theta_i| < 0.03$$

  The "farthest" agent also receives a +0.9 reward if its teammate intercepts successfully, reinforcing cooperation.

- **Failure:** If the ball is not intercepted within the maximum episode length (1500 steps), the closest agent receives a -0.4 penalty if it has not aligned with the ball by the end ($|\theta_i| > 0.025$).

Dense Shaping Rewards

Dense shaping rewards are applied every timestep to speed up learning:

- **Closest-Agent Shaping:** The agent currently closest in heading θ_i to the ball receives:

$$r_{\text{closest}} = 0.3 \cdot (-\text{distance}) + 0.7 \cdot (-|\theta_i|)$$

  This encourages the active agent to move laterally to intercept the ball.

- **Farthest-Agent Penalty:** The farthest agent's shaping reward is zero by default, but if it moves laterally (action $\neq$ stand still) it is penalized by -0.5 to discourage unnecessary chasing.

- **Collision Penalty:** If the robots' torso positions are within 0.3 m, both agents receive a -0.3 penalty to avoid collisions.

This reward design ensures:

- Only one agent moves to intercept the ball.

- The second agent holds position, minimizing risk of defensive gaps or robot-robot collisions.

Final Reward Computation

The timestep reward for agent i is:

$$R_i = \begin{cases} \textbf{Closest agent:} & \\ \quad 0.3(-d_i) + 0.7(-|\theta_i|) & \text{shaping}, \\ \quad -0.3 & \text{collision}, \\ \quad +\mathbf{0.9} & d_i < 0.8,\ |\theta_i| < 0.03 \text{ (success)}, \\ \quad -\mathbf{0.4} & \text{step} = 1500,\ |\theta_i| > 0.025 \text{ (timeout)}, \\ \textbf{Farthest agent:} & \\ \quad -0.5 & a_i \neq \text{stand}, \\ \quad -0.3 & \text{collision}, \\ \quad +\mathbf{0.9} & \text{teammate succeeds (coop.)}. \end{cases}$$

where d_i is the distance to the ball, θ_i is the heading offset, a_i is the action performed by the robot and rewards in **bold** are the terminal rewards.

HyperParameters

We train a single shared PPO policy for both agents with RLlib. The key settings used in all reported experiments are listed below.

Parameter	Value
Algorithm	PPO (RLlib, `framework=torch`)
Optimizer batch size	`train_batch_size_per_learner` = 4096
SGD minibatch size	`minibatch_size` = 512
SGD epochs per iter	`num_epochs` = 10
Learning rate	`lr` = 3×10^{-4}
Entropy coefficient	`entropy_coeff` = 0.05
Shared policy	one policy for both agents (`policy_mapping_fn` → `"shared_policy"`)
Value of γ	0.99
Value of λ	1.0
Clipping parameter	0.3

Table 3.2: PPO hyperparameters for multi-agent setup.

The policy update should happen every 4096 steps across all the parallel environments. However, because of batch mode = "complete episodes", the runner ensures every episode is finished before policy update. As such, the minimum number of steps before policy update is 4096 as more steps might be completed to accommodate unfinished episodes. Our detailed custom callback class exposes metrics such as success rate, accumulated rewards, and individual episode rewards per iteration, which are key to identifying the success or failure of training.

3.5.6 Evaluation Methodology for Both Single and Multi Agent

During training, the policy was periodically evaluated, and the best-performing model was saved based on a combination of metrics: highest mean episode reward per iteration, lowest episode reward standard deviation per iteration, lowest mean episode length per iteration, and lowest episode length standard deviation per iteration. The saved model was then evaluated for 50 episodes in the simulation environment. Performance was visualized using:

- **Episode reward per episode** – total reward achieved in each evaluation episode.
- **Episode length per episode** – number of steps taken before termination in each evaluation episode.

The plots provide insight into both the effectiveness and stability of the learned policy.

Chapter 4

Evaluation

4.1 Results

The following details the results and analysis of the training and evaluation process for both single agent and multi-agent setups.

4.1.1 Single Agent Training Results

Figure 4.1 shows that learning is stable and monotonic after the first hundred thousand steps:

- **Accumulated rewards per iteration** (top-left) does not show an overall trend, however as our policy updates every 5120 steps and more episodes are finished in the latter part of our training, the Average reward might be a better metric to indicate overall improvement.
- **Average reward** (top-right) improves from roughly -800 at the start toward about -200, with the standard deviation narrowing over time.
- **Success rate** (middle-left) rises quickly and saturates near 1.0 by $\sim 10^5$ environment steps, remaining near-perfect thereafter. **Note** This applies to the *terminal*

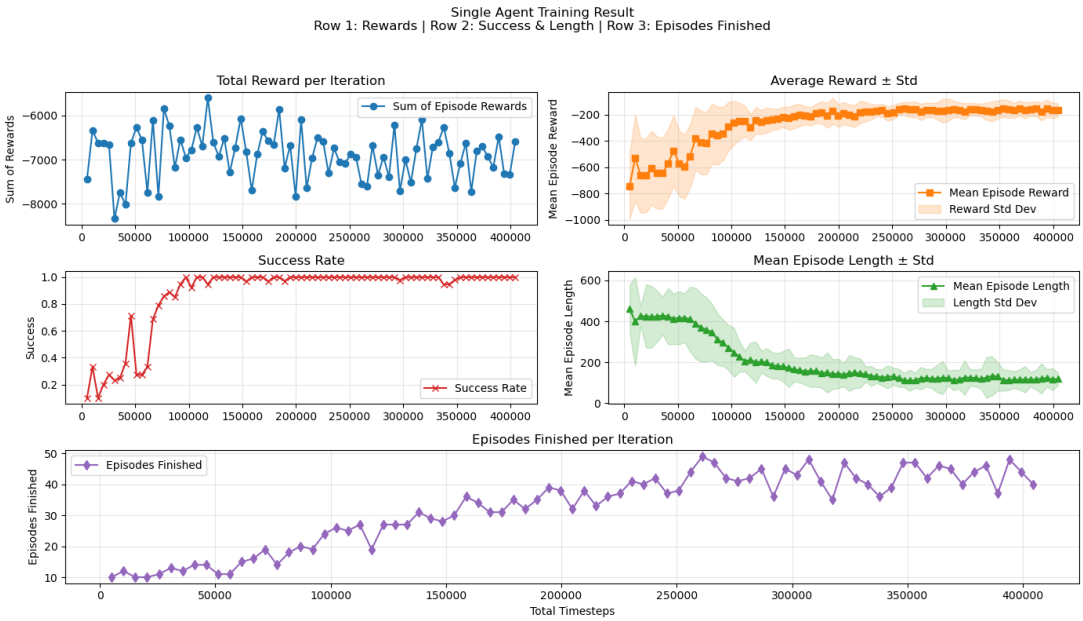

Figure 4.1: **Single-agent training curves.** Top-left: Sum of episode rewards per training iteration. Top-right: Mean episode reward with one–sigma band. Middle-left: Success rate. Middle-right: Mean episode length with one–sigma band. Bottom: Number of episodes finished per iteration.

success condition only.

- **Mean Episode length** (middle-right) drops from ∼400–500 steps to ∼100–150 steps, indicating faster interceptions.
- **Episodes finished/iteration** (bottom) steadily increases from ∼ 10 up to ∼ 40–50, consistent with shorter episodes as the policy improves.

4.1.2 Multi-Agent Training Results

Figure 4.2 shows the same qualitative progression as in the single-agent case, with coordination emerging quickly once the closest/farthest role split is learned by the shared policy:

- **Accumulated rewards per iteration** (top-left) start low and fluctuate heavily in the early iterations, then steadily improve and stabilize after around hundred

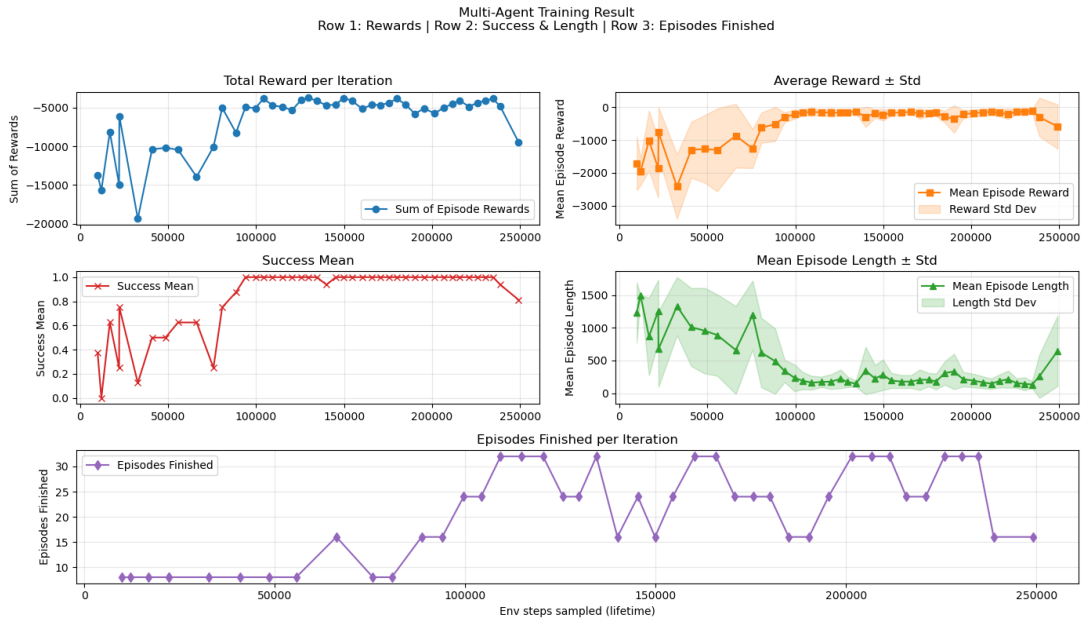

Figure 4.2: **Multi-agent training curves.** Top-left: Sum of episode rewards per training iteration. Top-right: Mean episode reward with one–sigma band. Middle-left: Success rate. Middle-right: Mean episode length with one–sigma band. Bottom: Number of episodes finished per iteration.

thousand environment steps, showing a clear improvement over time.

- **Average reward** (top-right) climbs from roughly -2000 toward ≈ -200, and the variance shrinks after 90K environment steps, indicating stabilization of the shared policy.

- **Success rate** (middle-left) rises from near zero to ≈ 1.0 by 90K steps and stays constant for the remainder of training (with a brief dip near the end). **Note** This applies to the *terminal success* condition only.

- **Mean Episode length** (middle-right) drops from $\sim$1000–1500 steps down to $\sim$100–200 steps, mirroring the success curve and reflecting faster interceptions once roles are established.

- **Episodes finished/iteration** (bottom) increases from about $\sim$8 to $\sim$24–32 as episodes shorten.

4.2 Evaluation

4.2.1 Single Agent Training Evaluation

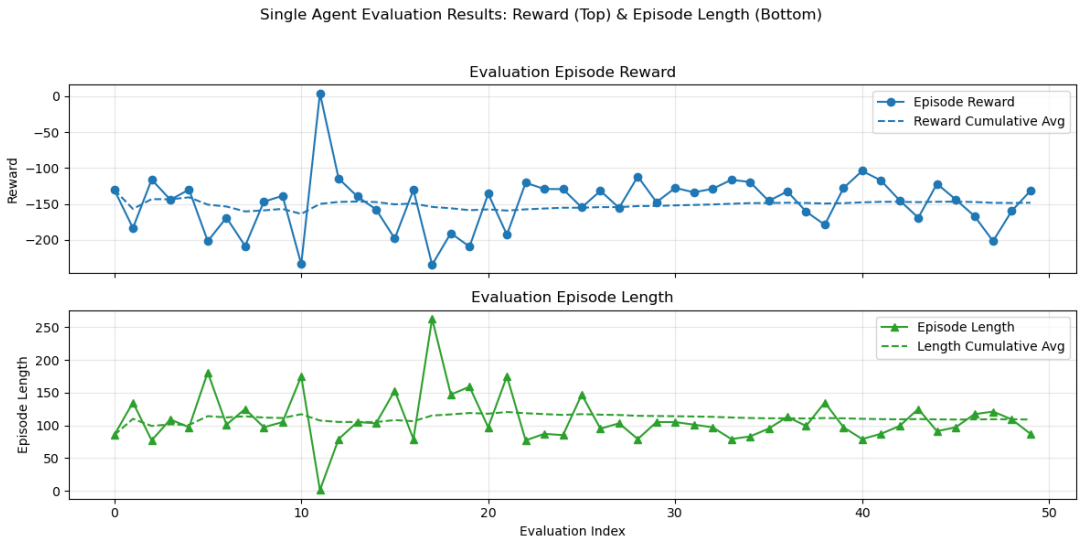

Figure 4.3: **Single-agent evaluation.** Top: per-episode reward with the cumulative average (dashed). Bottom: per-episode length with the cumulative average (dashed).

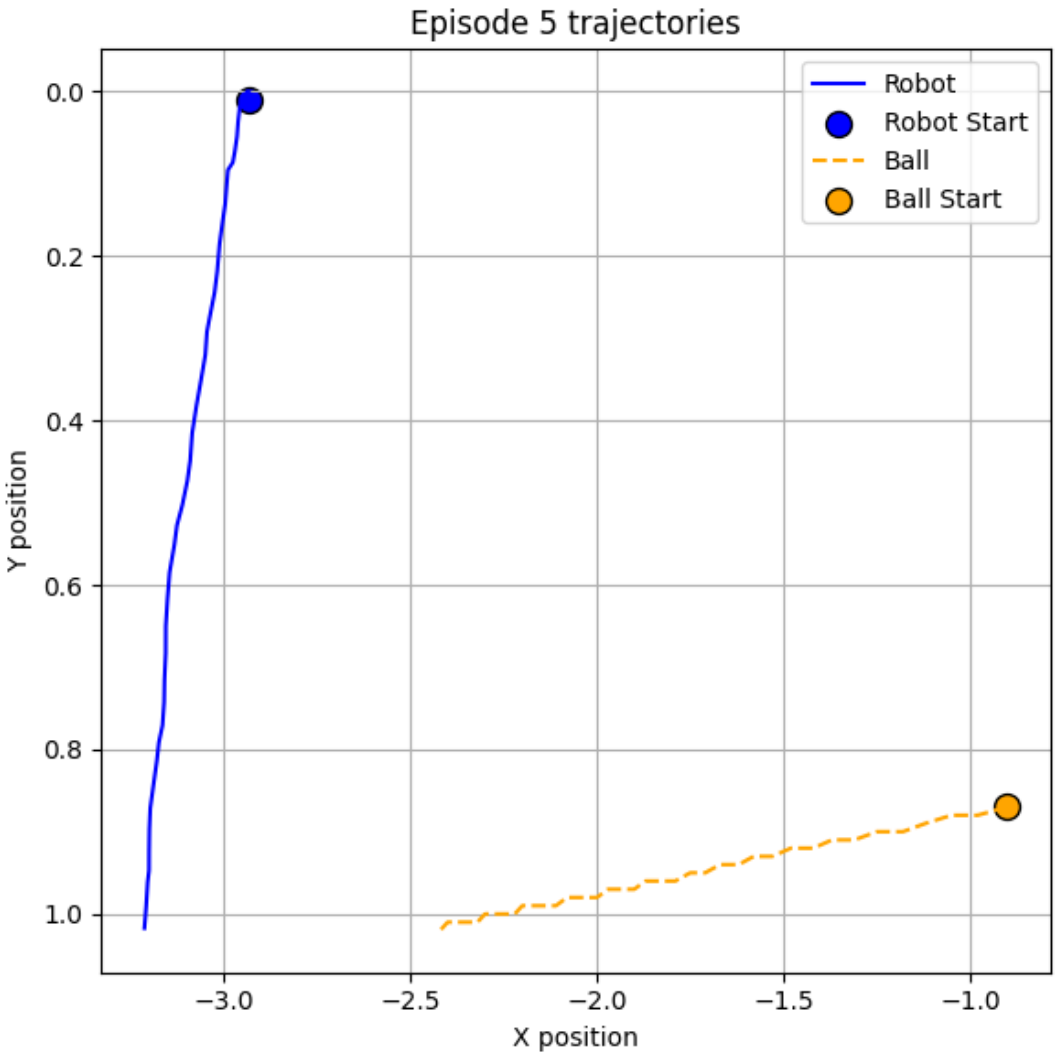

Figure 4.4: **Single-Agent Interception**

We evaluate the final policy for 50 episodes using the same reset/teleport protocol. Figure 4.3 shows that:

- **Rewards** cluster mostly between -200 and -100, with the running average stabilizing near ~ -150.
- **Episode length** varies around 80–120 steps (running average $\sim$100–110), with a few isolated outliers consisting of very short/very long episodes.

An example of successful single agent interception is given in Figure 4.4. Absolute rewards remain negative due to design-time per-step penalties (time penalty, early-motion penalty, periodic progress checks). Performance is therefore better reflected by the consistent episode lengths and the absence of strong deviation across rewards and episode lengths during evaluations.

4.2.2 Multi-Agent Training Evaluation

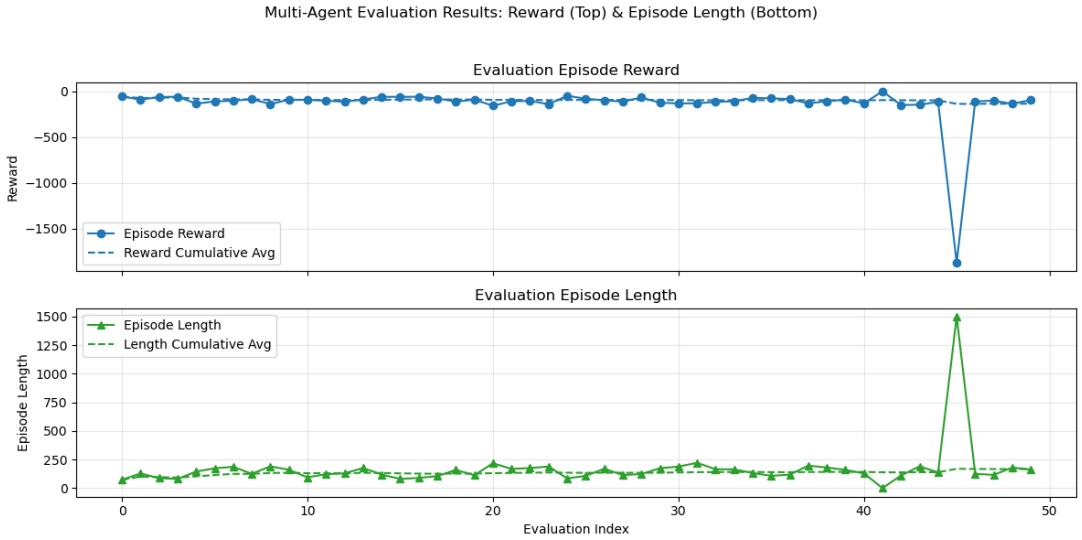

Figure 4.5: **Multi-agent evaluation.** Top: per-episode reward with cumulative average (dashed). Bottom: per-episode length with cumulative average (dashed).

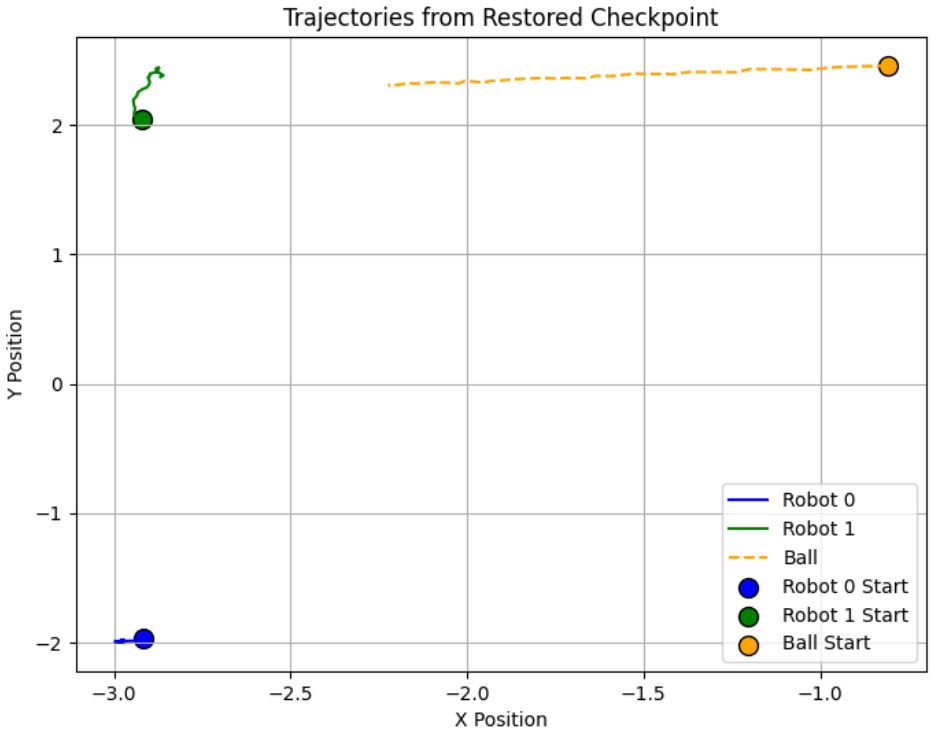

Figure 4.6: **Multi-Agent Interception**

We evaluate the final shared policy for 50 episodes using the same teleport/reset protocol and greedy action selection at test time. As shown in Figure 4.5:

- **Rewards** are tightly clustered between roughly -50 and -150 for almost all episodes; the cumulative average stabilizes near ~ -100. A single rare outlier drops to ≈ -1700, coinciding with a time-limit episode.
- **Episode length** concentrates around 80–180 steps, consistent with quick interceptions. The same outlier reaches the maximum of 1500 steps (time limit), and there is one very short episode (immediate termination), likely from a favorable initialization or a delay in initial teleportation of robots leading to one of them meeting the terminal success criteria in a very short time span.

An example of successful multi-agent interception is given in Figure 4.6.

4.3 Discussion

4.3.1 Single Agent Training Discussion

The results are consistent with the reward design and rollout configuration ($n_{\text{env}} = 10$, $n_{\text{steps}} = 512$, learning rate 3×10^{-4}). Even though the *sum* of rewards per iteration remains negative, the agent learns to intercept the moving ball. This is also indicated by the decreasing standard deviation bands for both average episode rewards and episode lengths, suggesting more uniform episodes in the latter part of training. This outcome is expected because of the reward design which penalizes wasting time and encourages successful interception by giving rewards based on current progress. The overall learning is more prominent in metrics like *success rate*, *mean episode length*, and *average reward* than in absolute reward totals.

4.3.2 Single Agent Evaluation Discussion

During the evaluation episodes, the learned policy maintains the same qualitative behaviour observed during training: quick interceptions (mean length near 100 steps) and low variance across average rewards and episode lengths.

4.3.3 Multi-Agent Training Discussion

The multi-agent training data reflects the result of a shared policy assigning the closest and farthest midfielder roles to each agent, leading to successful interception completion. Even though the *sum* of rewards per iteration remains negative, the policy converges to cooperative interception. This is consistent with the design which rewards the closest agent for actively trying to intercept the ball by minimizing both distance and heading error, while the farthest agent is penalized for unnecessary motion.

Consistent success rate in the latter part of training and low standard deviation across mean rewards and lengths signifies convergence to stable, complementary roles. Minor

late training fluctuations in success and episode length reflect overfitting; the model was saved before performance deteriorated.

4.3.4 Multi Agent Evaluation Perspective

During evaluation episodes, the cooperative strategy remains consistent: both agents act in accordance with their learned roles, and episodes are short and stable in length while the reward remains consistently high. The isolated long-duration, low-reward episode corresponds to a time limit being reached due to the model encountering an edge case, whereas the very short episode lengths indicate rapid interception, even though overshooting is observed in some episodes.

4.3.5 Limitations

Overshoot correction

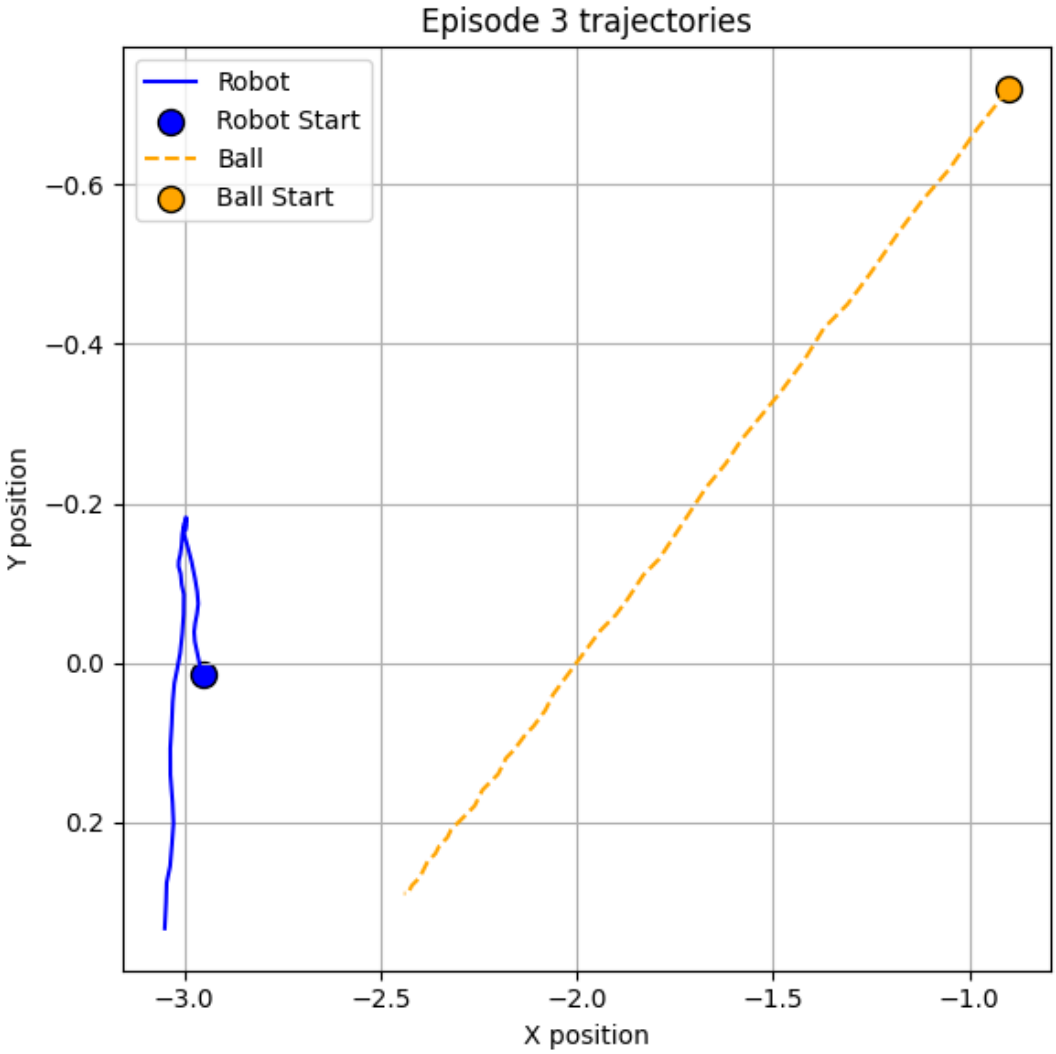

Figure 4.7: **Overshooting**

In both single-agent and multi-agent setups, the agent(s) occasionally sidestep past the optimal interception point and then adjust back toward the ball. While this does not prevent successful interceptions, it adds unnecessary motion and slightly increases episode length. This is an example of hysteresis, where delayed corrective actions cause oscillatory behaviour, as shown in Fig. 4.7 where the robot initially moves to its left

(+Y) before correcting its trajectory and moving back to its right (-Y). This results in unnecessary movement and time loss, leading to longer episodes and reduced rewards. To address this, the heading condition for terminating an episode successfully needs to be relaxed, as the current constraint can lead to overshooting. The heading parameter itself depends on the distance to the ball which needs to be considered. Tighter heading-alignment shaping or predictive positioning can reduce this effect without altering the overall policy.

Shared Policy Confusion

Shared policy allows agents to observe only their local relative observations during training. This can be a double-edged sword, as certain situations may arise where the ball is at an equal heading offset from each robot, in which case, the model may fail to make a definitive choice about which agent should attempt the interception.

Seeding and Edge Cases in Multi-Agent Scenario

The ball is teleported with a random probability onto one of the blue bands and assigned a random speed based on its initial spawn location. With different seeds, it is possible that some edge cases will not be encountered by the model, even if overall performance appears good. One such example is shown in Fig. 4.5, where an evaluation episode runs out of time at 1500 steps. This case occurred when the ball was teleported to the extreme boundary of a blue band and no robot moved to intercept it. Technically, this behaviour is reasonable, as a faster-moving ball in that position would be unreachable. To fix this issue, we can further decrease the width of the blue bands and narrow the interception zone, so that the robots can intercept the ball in a more realistic way. Another reason for this behaviour is the probabilistic way in which the ball is teleported and launched, meaning the robot may not have encountered this particular condition often during the training seed, leading to unintended behaviour. As such, training with more seeds should expose the model to a wider range of such edge cases.

4.3.6 Drift

Similar to the real-world setting, Nao robots inside the simulation can drift showing unnecessary backward and forward movement even though the action spaces only allow lateral movement. This can cause complications, especially with the episode terminal conditions if not handled properly.

Chapter 5

Conclusion

5.1 Conclusion

This thesis explored the application of reinforcement learning (RL) to develop and evaluate ball interception behaviours for the Nao robot in the RoboCUP 3D Simulation environment. Through both single-agent and multi-agent training, the intended midfielder behaviour was observed as robots moved laterally to intercept the moving ball. The experiments also demonstrated that RL based policies can learn cooperative strategies which result in high rewards and low episode durations i.e faster interceptions. In the multi-agent scenario, training also leads to successful differentiation of closest and farthest midfielder roles without relying on each other's observations. However, certain limitations such as overshooting, shared policy ambiguity in edge cases, and seed-dependent coverage gaps highlight the need for more robust training methodologies.

Overall, the work presented demonstrates the advantages and challenges of applying RL in both single agent and multi-agent robotic environments. By refining training methods and addressing hardware specific constraints on real robots, the learned behaviours can be improved further and made ready for transfer from simulation to real robots.

5.2 Future Work Simulation

Further work can be performed in the simulation to facilitate improved performance and fix overshooting.

5.2.1 Recurrent Policy

The next step will be to try out recurrent policy which allows the robot to make decisions on past history in addition to current observations to see if the overshooting problem can be fixed.

5.2.2 Curriculum Learning

Curriculum Learning is a strong candidate to improve performance in complicated environments with noise like the one studied here. The plan is to start with slower ball speed and gradually increase it to check for possibility of improved performance.

5.3 Future Work Deployment

There are some key things to consider while deploying the learned RL behaviour to real Nao robots.

5.3.1 Ball Visibility

All experiments in this paper have been performed under the assumption that the ball is visible to the robot(s) at all times, using the server cheat command to include the ball position in the observations. The real Nao robot, however, has limited visibility due to its vision capabilities. There are essentially two ways to address this issue:

- Simulate visibility constraints in the server and retrain the model. This method requires knowledge of the ball's past positions, which would necessitate the use of recurrent policies such as LSTM-PPO, making the model more complex and computationally expensive.
- Deploy the model as is on the real robot, and during periods when the ball is not visible, have the robot perform the "Stand" action until the ball becomes visible again.

5.3.2 Hysteresis

Hysteresis is the phenomenon where a robot overshoots or jitters, especially when performing different actions in rapid succession over a short time period. During training, the action command for the robot is updated at every step based on the current observation. However, on a real robot, if different action commands are issued every step, the robot can become unstable and jittery. One possible solution is to call the model and select an action only once every N steps, with the robot repeating that action continuously during that period to reduce jitter and improve stability.

5.3.3 Model Computation

The real Nao robot can overheat easily if the model is computationally expensive. RLlib offers a way to export the model in ONNX format, which provides a lightweight representation. In addition, the actions performed during simulation learning are mapped to the corresponding actions available on the real Nao robot. Because the observation space is relative, this approach offers the advantage of keeping the implementation simple.

5.3.4 Multi-Agent Communication

RoboCUP limits communication between agents during a match, and sending too many packets can result in penalties or disqualification. The use of a shared policy for multi-agent setup ensures that each agent is exposed only to its own relative observations during training. This is advantageous because, once the policy is exported to real Nao robots, no communication is required to run the model or test it.

Bibliography

[ARL25] Miguel Abreu, Luis Paulo Reis, and Nuno Lau. Designing a skilled soccer team for RoboCup: exploring skill-set-primitives through reinforcement learning. *Neural Computing and Applications*, 2025.

[HR06] Daniel Hein and Thomas Röfer. SimSpark–concepts and application in the RoboCup 3d soccer simulation league. In *RoboCup 2005: Robot Soccer World Cup IX*, pages 173–183. Springer, 2006.

[KT00] Vijay R. Konda and John N. Tsitsiklis. Actor-critic algorithms. *Advances in Neural Information Processing Systems*, 12:1008–1014, 2000.

[LLM+18] Eric Liang, Richard Liaw, Philipp Moritz, Robert Nishihara, Roy Fox, Ken Goldberg, Joseph E. Gonzalez, Michael I. Jordan, and Ion Stoica. RLlib: Abstractions for distributed reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *PMLR*, pages 3053–3062, Stockholm, Sweden, 2018. Available at `http://rllib.io`.

[MNW+18] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I. Jordan, and Ion Stoica. Ray: A distributed framework for emerging AI applications. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 561–577, 2018.

[ORR19] Jim Martin Catacora Ocaña, José L. Recio, and Luis P. Reis. Cooperative multi-agent deep reinforcement learning in soccer domains: The 2v2 free-kick task. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2019)*, pages 1865–1867, Montreal, Canada, 2019. IFAAMAS. Available at `https://www.ifaamas.org/Proceedings/aamas2019/pdfs/p1865.pdf`.

[Put14] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 2014.

[R+17] Thomas Röfer et al. B-human: team report and code release 2017. Technical report, Deutsches Forschungszentrum für Künstliche Intelligenz, Universität Bremen, 2017.

[RHG+21] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22:1–8, 2021. Available at `https://github.com/DLR-RM/stable-baselines3`.

[Rob97] RoboCup Federation. Robocup official website. https://www.robocup.org/, accessed 10/08/2025, 1997. RoboCup Federation.

[SB18] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2nd edition, 2018.

[SMSM00] Richard S. Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in Neural Information Processing Systems*, 12, 2000.

[SV12] Justin Stoecker and Ubbo Visser. Roboviz: Programmable visualization for simulated soccer. In Thomas Röfer, Noriaki Ando, James J. Kuffner, and Thomas M. Rückert, editors, *RoboCup 2011: Robot Soccer World Cup XV*, volume 7416 of *Lecture Notes in Computer Science*, pages 282–293, Berlin, Heidelberg, 2012. Springer-Verlag.

[SWD+17] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. In *arXiv preprint arXiv:1707.06347*, 2017.

[ZAS+22] Nader Zare, Omid Amini, Aref Sayareh, Mahtab Sarvmaili, Arad Firouzkouhi, Saba Ramezani Rad, Stan Matwin, and Amilcar Soares. Cyrus2d base: Source code base for robocup 2d soccer simulation league. In *RoboCup International Symposium*, Lecture Notes in Artificial Intelligence, 2022. Accepted for publication.