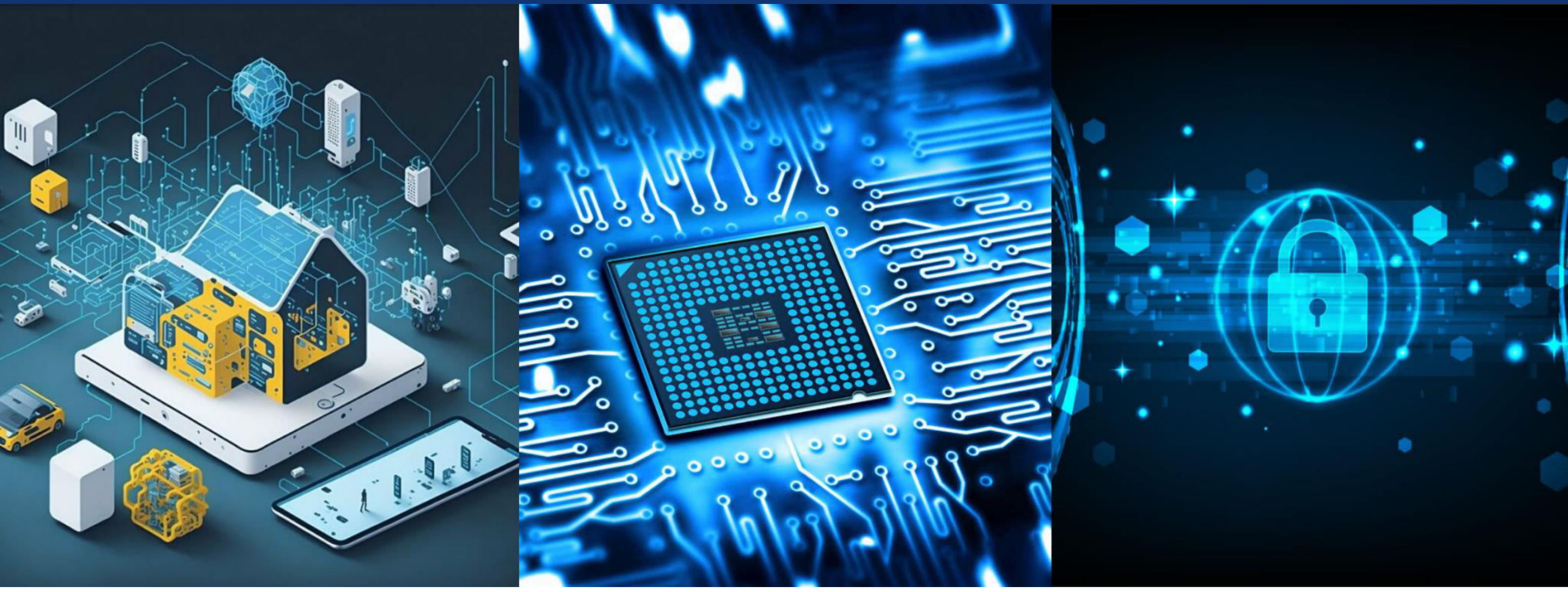
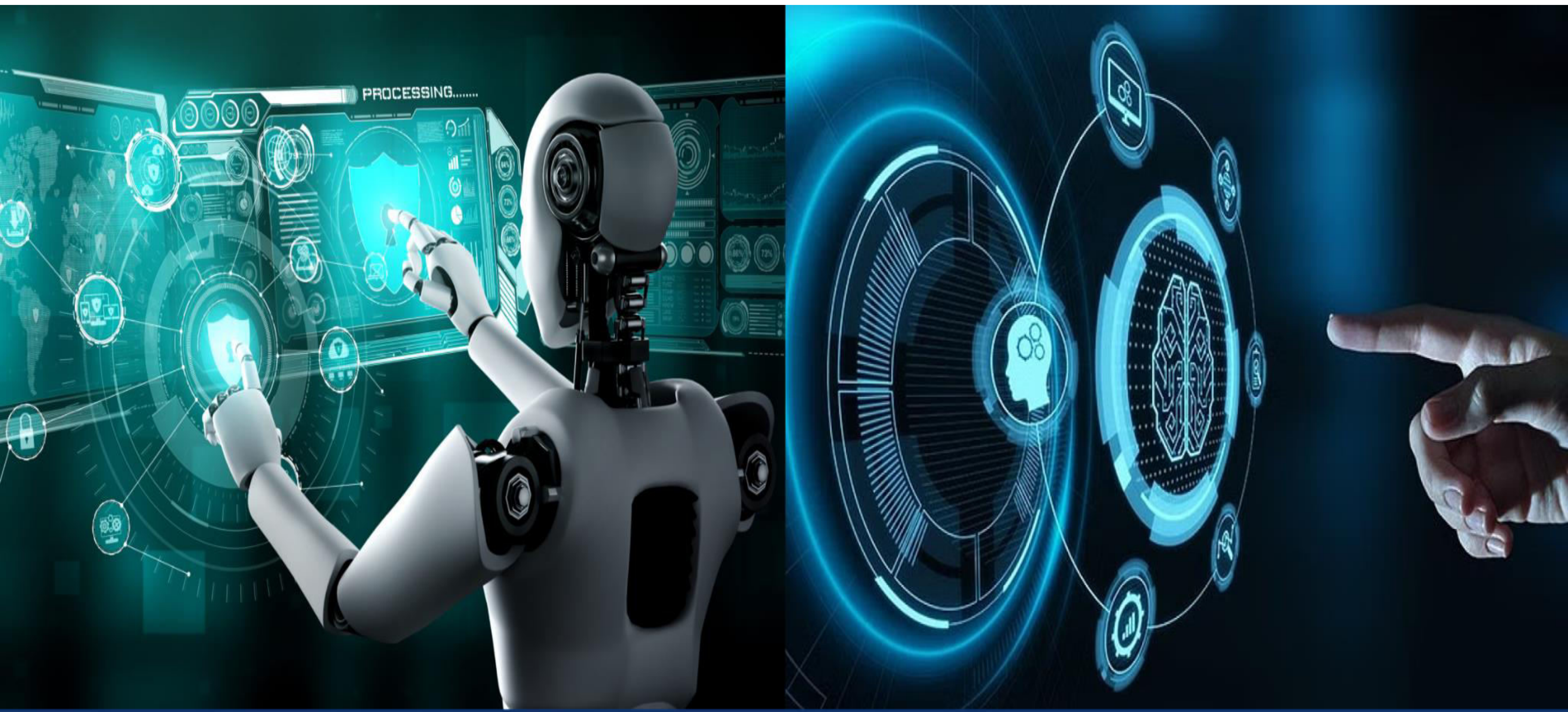
ISSN(O): 2320-9801

ISSN(P): 2320-9798



International Journal of Innovative Research in Computer and Communication Engineering

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)



Impact Factor: 8.771

Volume 13, Issue 3, March 2025

© 2025 IJIRCCE | Volume 13, Issue 3, March 2025| DOI: 10.15680/IJIRCCE.2025.1303125

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 8.771| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

Soccer Match Strategy Scheduling Based on NAO Robots

Aya Taouririte*, Li-Hong Juang

School of Artificial Intelligence, Nanjing University of Information Science and Technology, Nanjing, China

ABSTRACT: Intelligent robots especially NAO robots are expected to play an important role in various vertical industries and robot soccer match is an effective method for the research of multi-agent systems (MAS). Due to the highly dynamic and complex environment of the football field and the requirements for the real-time action of robot players, the optimization of robot strategy scheduling is recognized as an essential component of robot soccer match. Nevertheless, state-of-the-art decision-making on robot action scheduling can suffer from the processing of huge amount of date and the interaction between robots. To handle this issue, this paper proposes a robot soccer match strategy scheduling scheme, based on reinforcement learning (RL). We propose to formulate the decision-making problem of robot action scheduling as a Markov decision process (MDP), i.e. the basis of developing algorithms of deep Q-network (DQN) and proximal policy optimization (PPO). In order to strengthen the cooperation between robot players, we apply client-server (C-S) architecture to achieve efficient communication and joint scheduling between robot players. The simulation results confirm that the proposed algorithm is cable of learning a scheduling policy in the presence of a robot soccer match and significantly optimizing the robot action decision-making problem.

I.INTRODUCTION

The rapid development of intelligent robots has witnessed the success of applying intelligent robots in various vertical industries such as industrial automation [1], [2], medical services [3], [4] and intelligent agriculture [5]. Particularly, the NAO robot (a humanoid intelligent robot), designed and developed by Aldebaran Robotics, integrates sensors, a vision system, and a control system into a single platform. It is the most widely used fully autonomous humanoid robot in academic research to date [6]. In recent years, scholars have conducted research on multi-agent collaboration and planning through robot soccer. The NAO robots play a crucial role as one of the most commonly used robots for studying robot soccer [7], [8].

In robot soccer matches, team strategies are one of the key factors influencing the outcome of the game. Consequently, many researchers focus on optimizing team strategies and robot path planning. Ref. [9] focused on team strategies by optimizing underlying motions and action chains to enhance the smooth execution of actions by robot players during matches. Ref. [10] utilized artificial neural networks (ANNs) to optimize the offensive and defensive strategies of robot player agents while incorporating RL algorithm models into the behavioural decision-making modules of the robot player agents. Ref. [11] employed model-based collaborative filtering techniques to determine the optimal team strategies and used feature selection algorithms to identify the strategies with the greatest impact on the final match outcomes. Applying these methods to robot soccer matches improved team performance by over 35%.

However, robot soccer matches take place in a dynamic and uncertain field environment, making it challenging for team strategy optimization to cope with highly dynamic conditions. Consequently, robot player agents must be capable of making appropriate decisions in real time and at high speed. Therefore, how players make action decisions has become one of the key focuses of research. Ref. [12]–[14] utilized genetic algorithms to improve the evaluation functions for players' action execution behaviours, significantly enhancing their ability to perform actions. Ref. [15] applied unsupervised learning algorithms, such as clustering, and supervised learning algorithms to the design of robot players' underlying actions. Meanwhile, ref.

[16], [17] utilized reinforcement learning algorithm models to optimize robot players' shooting actions and predict the overall state behaviour of ball-holding players. These methods were further applied to improve players' field of vision selection.

This paper investigates a robot soccer match, as shown in Fig.1. Multiple robot player agents interact with each other in a highly dynamic field environment, forming a complex dynamic Nao robots action scheduling decision-making problem. As a countermeasure, we exploit RL to solve the robot players decision-making problem. We utilize the deep Q-Network (DQN) architecture for action determination and employ deep neural networks for parameter learning. Simultaneously, we propose a proximal policy optimization (PPO) algorithm based on the actor-critic (AC) architecture, to address the limitations of DQN in terms of convergence speed and stability. This helps mitigate performance degradation of DQN in continuous and highly dynamic environments. Meanwhile, we utilize client-server (C-S) architecture to enhance communication and collaboration among robot players. Simulation results confirm that proposed algorithm can converge and enable the robot to perform well in the competition. The comparison between AC approach with PPO and the DQN approach indicates that PPO theory can benefit the dynamic scheduling in the presence of many agents and complex environment.

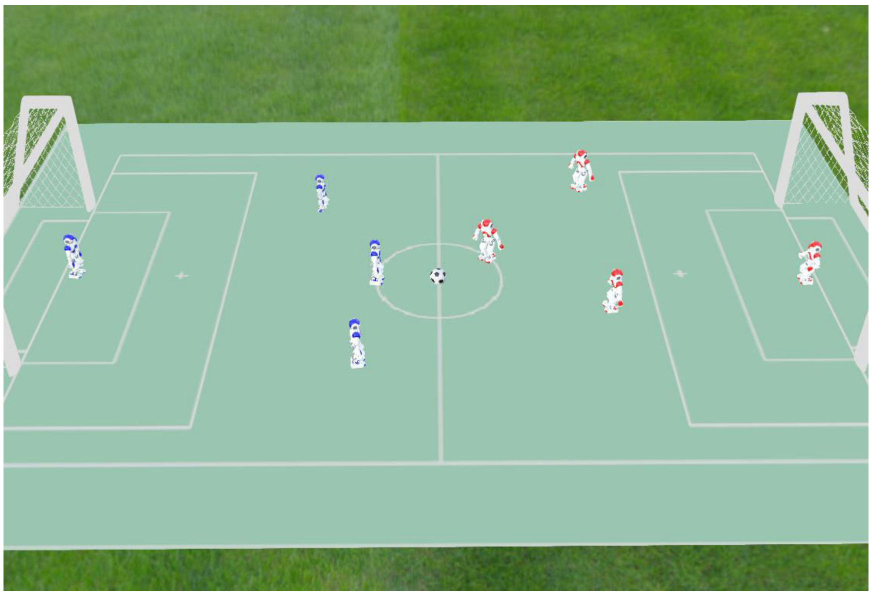


Fig. 1. 4v4 NAO Robots Soccer Match.

II. PRELIMINARY

This system controls a group of Nao robots to autonomously make decisions in a football field, completing a 4v4 football match. In this system, we employ reinforcement learning algorithms to assist the robots in playing the football game. Additionally, robots within the same team communicate with a server, which makes decisions based on the current state of the game. These decisions specify the actions that each robot should take in the next time step, facilitating teamwork among the robots of the same team.

A. Robot Behaviour Determination

In this system, there are two teams: the Blue team and the Red team. The robots in each team are denoted by n, where n ranges from 0 to 4. We define the position of the robot as $p_n = (x_n, y_n, z_n)$, the position of the ball as $p_{\text{goal}} = (x_{\text{goal}}, y_{\text{goal}})$, and the position of the goal as $p_{\text{target}} = (x_{\text{target}}, y_{\text{target}})$. The Nao robot determines its orientation θ_n using its own inertial measurement unit (IMU). The orientation between the robot and the ball is calculated as $\theta_n^{\text{goal}} = \arctan\left(\frac{y_{\text{goal}} - y_n}{x_{\text{goal}} - x_n}\right)$ based on the coordinates of the two points. We need to calculate the deviation between the robot's orientation and the orientation between the robot and the ball to determine whether the robot needs to rotate. The orientation deviation between the two can be defined as $\Delta\theta_n = \theta_n^{\text{goal}} - \theta_n$. If $|\Delta\theta_n| < b_{\text{rad}}$, the robot is considered to be facing the ball; otherwise, the robot needs to rotate, where b_{rad} is the orientation deviation threshold. At the same time, the robot needs to determine whether it has control of the ball through the distance between the robot and the ball, as well as the orientation deviation. The result of this decision $if\_c = [c_1, c_2, c_3, c_4]$, will serve as a state variable that influences the decisions of other robots on the same team. We calculate the distance between the ball and the robot $d_n = \sqrt{(x_{\text{goal}} - x_n)^2 + (y_{\text{goal}} - y_n)^2}$. If $d_n < b_d$ where b_d is the distance threshold and $|\Delta\theta_n| < b_{\text{rad}}$, robot n has control of the ball, and set $c_n = 1$, otherwise $c_n = 0$. The robot n also needs to perform a shooting judgment. We define $goal\_x\_width$ and $goal\_dis$ as the width of the goal and the distance from the goal to the field, respectively. We simultaneously define $x_{\text{goal}} < \dfrac{goal\_x\_width}{2}$ and $(goal\_dis - 1) < y_{\text{goal}} < goal\_dis$ as the shooting distance condition, meaning that the ball is within the goal range and sufficiently close to the goal. Under these conditions, the shooting distance requirement is satisfied; Meanwhile, a shooting angle condition needs to be set. When robot n has control of the ball, the angle between the robot and the goal $\theta_n^{\text{target}} = \arctan\left(\dfrac{y_{\text{target}} - y_n}{x_{\text{target}} - x_n}\right)$, as well as the angle between the ball and the goal $\theta_{\text{goal}}^{\text{target}} = \arctan\left(\dfrac{y_{\text{target}} - y_{\text{goal}}}{x_{\text{target}} - x_{\text{goal}}}\right)$, are calculated, and the deviation between the two is $\Delta\theta_n^{\text{target}} = \theta_n^{\text{target}} - \theta_{\text{goal}}^{\text{target}}$. Thus, the shooting angle condition is $\Delta\theta_n^{\text{target}} < b_{\text{shoot}}$, where b_{shoot} is the shooting angle threshold, meaning that the robot, the ball, and the goal are approximately aligned along a straight line. When both the shooting distance condition and the shooting angle condition are satisfied, the robot will perform the shooting action.

B. Gait Planning and Error Compensation for Nao Robots

When performing gait planning for the Nao robot, after determining the position of the ball, the robot should be able to walk to a position close to the ball. Typically, we use the $MoveTo(x, y, theta)$ function, setting the X and Y coordinates, as well as the robot's angle around the Z-axis, to make the robot start walking. However, experiments have shown that with this walking function, the robot's gait is unstable, and when walking a long distance, there can be significant deviation from the target position. To address this issue, we control the gait parameters of the Nao robot to reduce errors during walking. We use the $setFootSteps$ function to customize the robot's gait parameters, adjusting the step distance to stabilize the robot's walk. The function's input parameters are $[x, y, \theta]$, which represent the distance in the X and Y directions between the left and right foot after a step, as well as the rotational angle around the Z-axis. Experiments showed that setting the gait parameters to $[0.15, 0.1, 0]$ results in a step length of approximately $0.1m$ per step. When the step distance is greater than $0.1m$, the robot uses the $setFootSteps$ function to walk; when the step distance is less than $0.1m$, the robot uses the $MoveTo$ function. The error compensation strategy ensures that the Nao robot can move towards the target position with minimal error.

C. Communication Architecture and Team Collaboration

To facilitate efficient decision-making and enhance team collaboration among robot players, we employ a Client-Server (C-S) architecture as shown in Fig.2 in our robot soccer match system. This design enables seamless communication between individual robots (clients) and a centralized server. Each robot observes the environment to extract its state, which is then transmitted to the server.

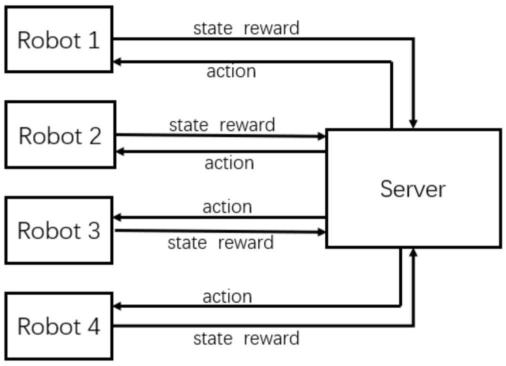


Fig. 2. C-S architecture.

On the server side, RL algorithms are executed to process the received state data. The server evaluates the state and reward sent back by robot players, generates optimal decisions for each robot, and transmits these decisions back to the respective robots. The robots subsequently execute the actions, completing their tasks in the soccer match.

This architecture strengthens communication between robots by leveraging the server as a centralized decision-making hub, enabling improved coordination among teammates. The C-S architecture ensures that robots operate cohesively, optimizing overall team performance in the dynamic soccer environment.

D. Robot Soccer Match Control Logic

The core goal of the match control logic in this study is to intelligently manage and control the robot football match, including match time, score, player states, ball control, robot recovery, and field boundary detection. We implement match management by inheriting the Supervisor class in Webots, which includes functions for robot movement, score determination, and collision detection. Before the match begins, a series of initialization operations are carried out, including setting the initial score, initializing robot positions, and resetting the ball's initial position. The positions of the robots and the ball are stored in global variables, which are updated during the match. The current position of the ball and its distance to the goal area are used to determine whether a goal is scored. If the ball is within the goal area and sufficiently close to the goal, a scoring event is triggered. Each time a goal is scored, the score is updated depending on which team scored, and the $ballinitial$ method is called to reset the ball to the center of the field for the next round.

During the match, the robots' states are monitored in real time, including whether the robot has fallen or needs to stand up. The robot's Z-axis height is used to determine if it has fallen. If the robot falls (i.e., the Z-axis is below a set threshold), a "get up" operation is triggered. This operation is implemented through the $getup$ method, which repositions the robot to a standing position. If the robot has fallen and the set recovery time (defined by $stand\_up\_cost$) has passed, the robot will automatically perform the $getup$ operation and return to a standing state. This process includes adjusting the robot's position and posture to ensure it can resume movement during the match. After each ball control, the $check\_belong$ method is used to determine which team controls the ball, ensuring that both teams do not have control of the ball simultaneously. Additionally, if a robot goes out of bounds, it will be reset to a predetermined position. To ensure the fairness of the match, the robot's movement range must be monitored, and the ball must remain within the field's boundaries. If the ball or robot goes out of bounds, the $check\_out\_of\_bounds$ and $check\_out\_of\_pitch$ methods are triggered, and the system will automatically adjust the positions of the robot and the ball to keep them within the designated area. Throughout the match, the system periodically updates the match information, including the current score and which team has control. After the match ends, the system will display a message on the screen using the $setLabel$ method, indicating that the match is over. The match ends when the pre-set time limit (e.g., 10 minutes) is reached. When the match time expires, $robot.step(timestep)$ enters the exit state, indicating that the match is over. To ensure coordination and information synchronization among multiple robots, the system uses the Emitter class for data transmission. Whenever location or state information needs to be transmitted, the *sendmessage* method sends the position data of all robots and the ball to the server using the Emitter class, ensuring that all robots and the match state remain synchronized during the match. As shown in Figure 3, the match logic control flow is illustrated.

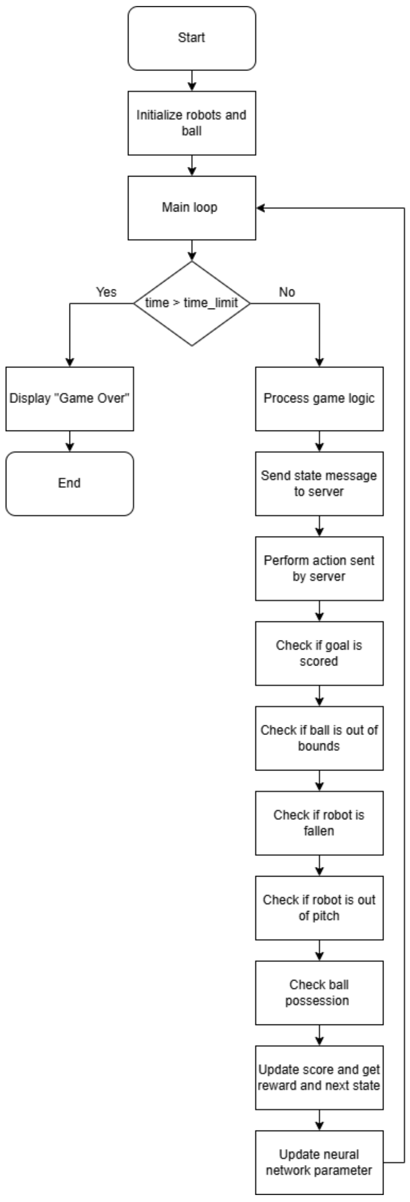

Start
Initialize robots and ball
Main loop
time > time_limit
Yes
No
Display "Game Over"
End
Process game logic
Send state message to server
Perform action sent by server
Check if goal is scored
Check if ball is out of bounds
Check if robot is fallen
Check if robot is out of pitch
Check ball possession
Update score and get reward and next state
Update neural network parameter

Fig. 3. Long-term average reward as a function of timeslots

E. Problem Formulation

Aiming at optimizing the actions taken by the NAO robots, this study addresses the problem of optimizing the robot's decision-making strategy for a football match. Intuitively, the decision-making on action scheduling turns out to be an MDP with one objective, where the immediate reward can be therefore formulated as $r_t = -\log\left(r_t^{\text{goal}} + r_t^{\text{close}} + r_t^{\text{control}}\right)$, where r_t^{goal}, r_t^{close} and r_t^{control} represent the goal reward, the proximity to the ball reward, and the ball control reward, respectively. Applying a logarithm to the reward helps smooth out large variations in reward values, preventing overly large rewards from dominating the learning process. It ensures more balanced learning by compressing extreme values, promoting stability and improving the efficiency of decision-making, especially in multi-objective tasks. For such an MDP, the state in timeslot t can be formulated as $s_t = [P, p_{\text{goal}}, if_c, P^{\text{oppo}}] \in \mathrm{S}$, where $P = [p_1, p_2, p_3, p_4]$, $P^{\text{oppo}} = [p_1^{\text{oppo}}, p_2^{\text{oppo}}, p_3^{\text{oppo}}, p_4^{\text{oppo}}]$ and S represent the positions of the robots on the same team and the opposing team and state space, respectively. The action scheduling action performed by robot $n$ can be obtained as $a_{n,t} \in \mathrm{A}_n$, where A_n is the action space w.r.t. robot $n$. The actions robot $n$ can take in a timeslot include moving forward, turning left, turning right, shooting, and staying. By defining $\mathrm{A} = \prod_{n=1}^{4} \mathrm{A}_n$, the action scheduling policy can be expressed as $\pi : \mathrm{S} \to \mathrm{A}$.

In order to maximize the long-term average reward, an objective featuring discounted cost can be written as $\bar{R} = \mathrm{E}\left\{\sum_{t=0}^{\infty} \gamma^t r_t\right\}$, where γ represents the discount factor. Hence, the optimization of π can be formulated as

$$\pi = *argmax_{\pi} \bar{R} \quad (1)$$

III. NAO ROBOT DECISION-MAKING OPTIMIZATION ALGORITHM

A. Action Scheduling with DQN

In multi-robot systems, particularly in task execution within dynamic environments, traditional rule-based or model-based approaches may not cope with the complexity and uncertainty of the environment. Therefore, it is crucial to use reinforcement learning methods to optimize the robots' decision-making. Deep Q-Network (DQN) is an algorithm that combines deep learning with Q-learning, using deep neural networks to approximate the Q-function and overcoming the limitations of traditional Q-learning when the state and action spaces are too large. In this paper, we employ DQN to train the robot agents, enabling them to optimize decision-making through experience learning in complex environments.

DQN uses a multilayer perceptron (MLP) architecture, where the input is the current state and the output is the Q-value for each possible action. The function $\_\_init\_\_$ initializes 5 fully connected layers, each created using $nn.Liner$, which represents a linear transformation between the neurons in each layer. After each hidden layer, the ReLU activation function $torch.relu$ is applied. Finally, the output layer $fc5$ does not use an activation function. The result of the output is the Q-value for each action, which represents the expected return for each action taken in the current state.

During training, DQN uses Experience Replay and Target Network mechanisms to improve the stability of training. To avoid instability in the model due to the correlation between consecutive state-action pairs (s, a), DQN introduces the experience replay mechanism. The experience replay pool stores the experience data collected from the agent's interaction with the environment (state, action, reward, next state, and whether the episode is done). During training, small batches of data are randomly sampled from the experience replay pool, breaking the correlation between the data and improving the efficiency and stability of training. DQN also introduces the target network to calculate the target Q-value, which ensures the stability of Q-value updates.

We set the parameters of the target network as θ_n^-. Parameter θ_n^- is updated to the current parameters of the Q network θ_n, after a fixed number of steps. This helps to reduce the fluctuation of the target during training and improves the stability of Q-value estimation. The goal of DQN is to minimize the mean squared error (MSE) between the Q-value output by the Q network and the target Q-value. Let the current state be s_t, the action taken be $a_{n,t}$, the reward be r_t, the next state be s_{t+1} and the done flag be $done$.The target Q-value is

$$Q_{\text{target}} = r_t + \gamma \cdot \max_{a_{n,t}} Q(s_{t+1}, a_{n,t}; \theta_n) \cdot (1 - done), \tag{2}$$

where γ is the discount factor, which represents the degree of influence of future rewards. So the loss function is

$$\mathrm{L}(\theta_n) = \mathrm{E}_{s_t, a_{n,t}, r_t, s_{t+1}}[(Q_{\text{network}}(s_t, a_{n,t}; \theta_n) - (r_t + \gamma \cdot \max_{a_{n,t}} Q(s_{t+1}, a_{n,t}; \theta_n^-)))^2].$$

(3)

We update the Q network parameters θ^n by calculating the gradient of the loss function

$$\theta_n \leftarrow \theta_n - \alpha \nabla_{\theta_n} \mathrm{L}(\theta_n), \tag{4}$$

where α is learning rate. The pseudocode of the proposed DQN algorithm is summarized in Algorithm 1.

Algorithm 1 Action Scheduling with DQN

```
1:  Initialize: $t = 1$; the learning rates $\alpha$; initial state $\mathbf{s}$; neural network parameters $\theta_n$; experience replay buffer memory;
2:  repeat
3:      Robots observe the state $s_t$ and transmit $s_t$ to the server;
4:      Server uses epsilon-greedy strategy to select actions for each robot;
5:      For each robot
6:      repeat
7:          Robot $n$ executes the action $a_{n,t}$, yield rewards $r_t$, and observe the new state $s_{t+1}$ and $done$ flag;
8:          Store the $s_t$, $a_{n,t}$, $r_t$, $s_{t+1}$ and $done$ in the memory;
9:          The robot action scheduling agents perform $Q_{\text{target}} = r_t + \gamma \cdot \max_{a_{n,t}} Q(s_{t+1}, a_{n,t}; \theta_n) \cdot (1 - done)$;
10:         Robot $n$ computes the loss function $\mathrm{L}(\theta_n) = \mathrm{E}_{s_t, a_{n,t}, r_t, s_{t+1}}[(Q_{\text{network}}(s_t, a_{n,t}; \theta_n) - (r_t + \gamma \cdot \max_{a_{n,t}} Q(s_{t+1}, a_{n,t}; \theta_n^-)))^2]$;
11:         Update Q-network parameters $\theta^n$ using gradient descent $\theta_n \leftarrow \theta_n - \alpha \nabla_{\theta_n} \mathrm{L}(\theta_n)$;
12:     until All the robots has updated the Q-network parameters
13:         $t \leftarrow t + 1$;
14: until Stopping criteria
```

B. Action Scheduling Optimization with PPO based on AC

Although DQN performs exceptionally well in tasks with discrete action spaces, it has limitations in terms of stability in continuous action spaces and policy optimization. In comparison, the PPO algorithm introduces a "clipping" mechanism to effectively control policy updates, preventing excessively large parameter updates. This results in higher stability and adaptability, particularly in dynamic tasks such as robot soccer games. This paper proposes a robot decision-making optimization algorithm based on the AC (Actor-Critic) architecture, and improves the original AC framework by incorporating the PPO algorithm.

First, PPO uses the data collected from the environment, such as states, actions, and rewards, to calculate the discounted rewards at each time step

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k}. \tag{5}$$

Specifically, the algorithm reverses the reward sequence and then uses the discount factor γ to iteratively compute the return at each time step, so that the reward at each step takes into account future rewards. Next, PPO normalizes the discounted rewards

$$R_t^{\text{norm}} = \frac{R_t - \mu}{\sigma}, \tag{6}$$

where μ and σ represent the mean and standard deviation of all the discounted rewards, respectively. Normalization is performed to ensure that the variance of the rewards does not become too large, preventing numerical instability during training.

As the next step, PPO observes the state value $V(s_t)$ from the critic network, which represents the expected return when following the current policy from state s_t. The output of the critic network is an estimate of the state value. The advantage function can be formulated as

$$A_t = R_t^{\text{norm}} - V(s_t), \tag{7}$$

which means how good the current action is relative to the state value estimated by the critic network. The larger the advantage function, the better the current action is compared to other actions. When updating the policy, PPO computes the probability ratio between the new and old policies

$$r_t(\theta) = \frac{\pi_\theta(a_{n,t} \mid s_t)}{\pi_{\theta_{old}}(a_{n,t} \mid s_t)} = \exp(\log prob_t - \log prob_{t,old}), \tag{8}$$

where $\pi_{\theta_{old}}$, π_θ represent old and current policy, and $\log prob_t$ and $\log prob_{t,old}$ represent the logarithmic probabilities of action $a_{n,t}$ under the current and old policies, respectively.

To ensure that the policy updates do not deviate too far and to prevent large updates from causing instability in training, PPO uses a surrogate loss function

$$L^{Clip}(\theta) = E_t[min(r_t(\theta)A_t, clip(r_t(\theta), (1-\in, 1+\in)A_t] \tag{9}$$

where $Clip(r_t(\theta), 1-\in, 1+\in)$ constraines the probability ratio within the interval $[1-\in, 1+\in)$]. This means that when the probability ratio exceeds this range, PPO will use the clipped ratio to compute the loss. The clipped surrogate loss restricts the update magnitude between $1-\in$ *and* $1+\in$, thereby preventing excessively large policy updates. ò is the clipping parameter, which is typically set to 0.20. To further optimize the policy, PPO's loss function also includes a value loss, which is calculated by minimizing the mean squared error between the state values estimated by the critic network and the actual discounted rewards

$$L^V(\theta) = \mathrm{E}_t[(R_t^{\text{norm}} - V(s_t))^2]. \tag{10}$$

In addition, to encourage exploration of the policy, PPO also includes an entropy loss, which can be written as

$$H(\pi_\theta) = -\sum_{a_{n,t}} \pi_\theta(a_{n,t} \mid s_t) \log \pi_\theta(a_{n,t} \mid s_t). \tag{11}$$

This entropy loss term encourages the policy network's distribution to maintain a certain level of randomness, preventing the policy from becoming overly deterministic, thereby reducing the risk of premature convergence. Therefore, PPO loss function can be formulated as

$$L(\theta) = L^{\text{CLIP}}(\theta) + c_1 L^V(\theta) - c_2 H(\pi_\theta), \tag{12}$$

where c_1 and c_2 are the hyperparameters for the weights of the value loss and entropy loss, respectively. Through this loss function, PPO can balance the policy update, the accuracy of the value function, and the exploration of the policy. During training, gradients are calculated through backpropagation to minimize this loss function, and the parameters of the policy network and critic network are updated using the Adam optimizer. The pseudocode of the proposed PPO algorithm is summarized in Algorithm 2.

Algorithm 2 Action Scheduling Optimization with PPO based on AC

```
1:  Initialize: $t=1$; the learning rates $lr\_actor$, $lr\_critic$; initial state $\mathbf{s}$; neural network
    parameters $w_n$, $\theta_n$; clipping parameter $\epsilon_{clip}$
2:  repeat
3:      repeat
4:          For each robot;
5:          Robot $n$ observe the state $s_t$;
6:          For robot $n$, the action scheduling agents sample $a_{n,t} \sim \pi_n(\cdot\,|\,s_t;\theta_n)$;
7:          In timeslot $t$, robots execute the actions $a_{n,t}$;
8:          The robots yield rewards $r_t$ and observe the new state $s_{t+1}$ and $V(s_t)$;
9:          The action scheduling agents perform
    $L^{CLIP}(\theta) = E_t[min(r_t(\theta)A_t, clip(r_t(\theta), 1-\varepsilon, 1+\varepsilon)A_t)]$ to calculate surrogate loss;
10:         The action scheduling agents calculate mean squared error between the state values estimated
    by the critic network and the actual discounted rewards $L^V(\theta_n) = \mathrm{E}_t[(R_t^{\mathrm{norm}} - V(s_t))^2]$;
11:         The action scheduling agents calculate entropy loss
    $H(\pi_{\theta_n}) = -\sum_{a_{n,t}} \pi_{\theta_n}(a_{n,t}\,|\,s_t)\log\pi_{\theta_n}(a_{n,t}\,|\,s_t)$;
12:         The action scheduling agents calculate loss function
    $L(\theta_n) = L^{\mathrm{CLIP}}(\theta_n) + c_1 L^V(\theta_n) - c_2 H(\pi_{\theta_n})$;
13:         Robot $n$ calculates gradients through backpropagation to minimize this loss function;
14:         Updated the parameters of the policy network and critic network;
15:     until All the robots has updated the parameters
16:     $t \leftarrow t+1$;
17: until Stopping criteria
```

IV.SIMULATION RESULTS

In the simulations, each team has 4 NAO robots, the size of the football field is 6×4.5, and the goal size is 2.6×1. The learning rate for the DQN network is $\alpha = 1e-3$, and the exploration rate ϵ= 0.3. In the PPO algorithm, the clipping parameter is ϵ_{clip}= 0.2, discount factor $\gamma = 0.99$. The learning rate of policy network is $lr\_actor = 3e-4$, and the value network is $lr\_critic = 1e-3$. We conducted simulations in Webots R2023b.

In the simulations, we implemented DQN and PPO methods, to optimize the decision-making process of robot players. The performance of both methods was evaluated in a simulated robot soccer environment. Fig. 4 illustrates the long-term average reward. The plot shows that both the average rewards achieved by two RL methods gradually grow and finally saturate as the number of timeslots increases. This observation confirms confirms that PPO converges to the optimal strategy faster than DQN, and PPO achieves a higher final long-term average reward value, indicating that PPO has higher training efficiency, better decision-making ability and more effective environmental learning. The faster convergence speed of PPO can be attributed to its policy gradient based architecture, which allows for smoother updates and more stable training in continuous and dynamic environments. In contrast, although DQN is effective in discrete action spaces, its convergence speed is slower and performance is lower in complex football environments. The comparison highlights the applicability of PPO in highly dynamic multi-agent scenarios such as robot soccer, where rapid adaptation and precise tasks are required.

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 8.771| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

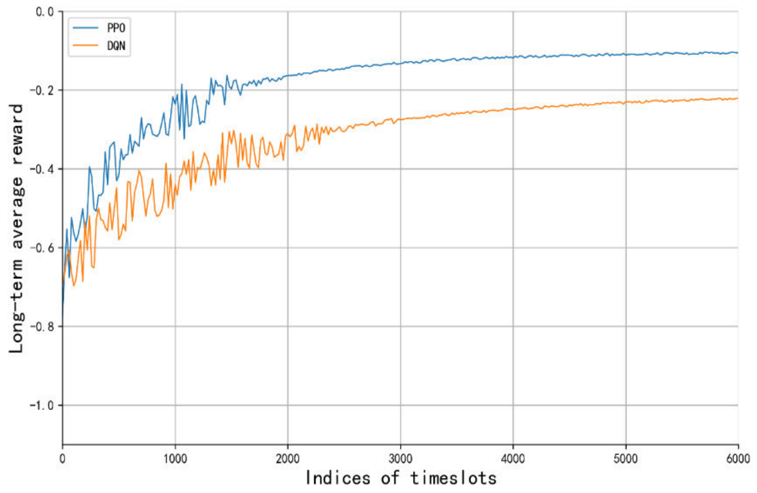


Fig. 4. Long-term average reward as a function of timeslots.

Fig. 5 and Fig. 6 respectively represent the long-term average rewards of four robots in the same team with DQN and PPO. Like Fig. 4, these two figures also show that PPO has a faster convergence speed and can converge to higher values compared to DQN. These two plots also illustrate that Robot R3 can converge to the highest value, Robots R2 and R1 converge to similar values, and Robot R4 converges to the lowest value. The different convergence values of the four robots confirm the optimization of the cooperation of robot players. Due to Robot 1 being the main offensive player with long-term ball control and shooting, it is able to converge to the highest value; Robots 2 and 3, as offensive players who spend most of their time in the frontcourt, tend to converge to slightly lower values; Robot 4 plays a defensive role in the backcourt for a long time, with less time for ball control and approach, so it will converge to the lowest value.

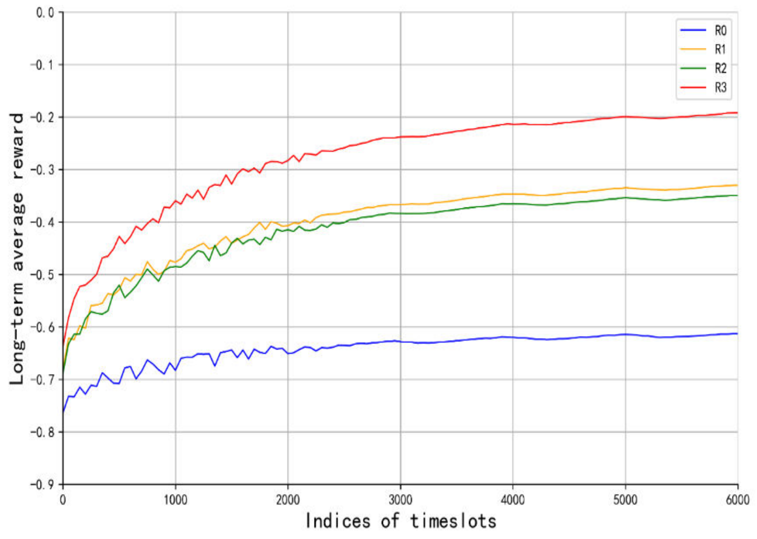


Fig. 5. Long-term average reward for each robot with DQN

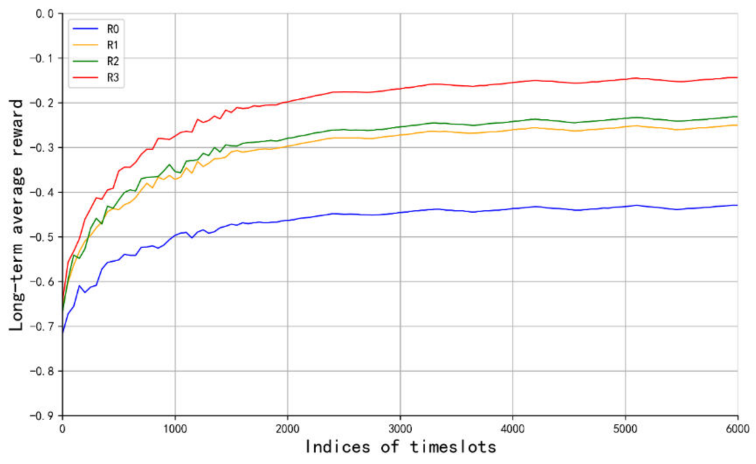


Fig. 6. Long-term average reward for each robot with PPO

We conducted 50 games between two teams optimized with DQN, and then conducted 50 matches between two teams optimized with PPO, with each game lasting 10 minutes. Table 1 shows the statistics of simulation test results. Table 1 demonstrates that using PPO to optimize robot decision-making can achieve better goal performance and more stable ball control compared to DQN. And more passes can also indicate that using the PPO algorithm can promote cooperation between robots.

TABLE I
MATCHES STATISTICS

	Average goals	Ball control rate	Average number of passes
PPO	4.32	82.9%	47.26
DQN	2.96	74.3%	22.34

Therefore, from the simulation results, whether from the perspective of long-term average reward convergence speed and reward value, or from the optimization of player offensive performance and team collaboration in actual matches, PPO algorithm can not only achieve decision-making for robots in highly dynamic environments, but also achieve better optimization results compared to DQN.

V. CONCLUSIONS

This paper has proposed decision-making methods based on DQN and PPO to address the complex decision-making problem of NAO robot players action scheduling. Simulation results shows that the proposed algorithms converge. Moreover, in the presence of a highly dynamic environment, the comparison between the proposed algorithms illustrates that the applying PPO in policy optimization is beneficial to dynamic scheduling in the robot soccer matches. PPO method has faster convergence speed and can converge to a higher value compared with DQN method,, which means that PPO method has better performance in optimizing robot player action decision-making.

REFERENCES

[1] M. Diprasetya, S. Yuwono, M. L ppenberg, and A. Schwung, "Integration of abb robot manipulators and robot operating system for industrial automation," in 2023 IEEE 21st International Conference on Industrial Informatics (INDIN), 2023, pp. 1–7.

[2] X. Wang, S. Lv, H. Liu, and M. Jia, "Research and design of an intelligent transfer system based on industrial robot," in 2024 IEEE 6th Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC), vol. 6, 2024, pp. 1348–1352.

[3] Y. Xia, Q. Li, R. Huang, and X. Zhao, "Design of intelligent medical service robot based on raspberry pi and stm32," in 2022 IEEE 10th Joint International Information Technology and Artificial Intelligence Conference (ITAIC), vol. 10, 2022, pp. 1577–1581.
[4] W. Wang, X. Liu, and C. Xu, "Design and implementation of medical service robot," in 2023 IEEE 3rd International Conference on Information Technology, Big Data and Artificial Intelligence (ICIBA), vol. 3, 2023, pp. 1219–1222.
[5] C. Zhai, T. Wang, Q. Xu, P. Zhang, W. Song, and C. Xie, "Research on the construction of intelligent agriculture based on agricultural robots," in 2023 6th International Conference on Mechatronics, Robotics and Automation (ICMRA)(, 2023, pp. 71–75.
[6] A. Balmik, M. Jha, and A. Nandy, "Nao robot teleoperation with human motion recognition," Arabian Journal for Science and Engineering, vol. 47, no. 2, pp. 1137–1146, 2022.
[7] Z. Wang, Y. Zeng, Y. Yuan, and Y. Guo, "Refining co-operative competition of robocup soccer with reinforcement learning," in 2020 IEEE Fifth International Conference on Data Science in Cyberspace (DSC), 2020, pp. 279–283.
[8] R. Kuga, Y. Suzuki, and T. Nakashima, "An automatic team evaluation system for robocup soccer simulation 2d," in 2020 Joint 11th Interna- tional Conference on Soft Computing and Intelligent Systems and 21st International Symposium on Advanced Intelligent Systems (SCIS-ISIS), 2020, pp. 1–4.
[9] H. Akiyama and T. Nakashima, "Helios base: An open source package for the robocup soccer 2d simulation," in RoboCup 2013: Robot World Cup XVII 17. Springer, 2014, pp. 528–535.
[10] T. Gabel and C. Roser, "Fra-united—team description 2018," in RoboCup 2019 Symposium and Competitions: Team Description Papers. Sydney, Australia, 2019.
[11] P. H. Abreu, D. C. Silva, J. Portela, J. Mendes-Moreira, and L. P. Reis, "Using model-based collaborative filtering techniques to recommend the expected best strategy to defeat a simulated soccer opponent," Intelligent Data Analysis, vol. 18, no. 5, pp. 973–991, 2014.
[12] M. Prokopenko, P. Wang, and O. Obst, "Gliders2014: Dynamic tactics with voronoi diagrams," 2014. [Online]. Available: https://api.semanticscholar.org/CorpusID:112517473
[13] "Gliders 2015 : Opponent avoidance with bio- inspired flocking behaviour," 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:8450405
[14] M. Prokopenko, P. Wang, O. Obst, and V. Jauregui, "Gliders 2016 :
Integrating multi-agent approaches to tactical diversity," 2016. [Online]. Available: https://api.semanticscholar.org/CorpusID:42354735
[15] T. Nakashima, H. Akiyama, Y. Suzuki, A. Ohori, and T. Fukushima, "Helios 2018 : Team description paper," 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:198184635
[16] N. Zare, M. Sarvmaili, O. Mehrabian, A. Nikanjam, S. H. Khasteh, A. Sayareh, O. Amini, B. Barahimi, and A. Majidi, "Cyrus 2d simulation 2019 team description paper," 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:232425078
[17] N. Zare, A. Sayareh, M. Sarvmaili, O. Amini, A. Soares, and S. Matwin, "Cyrus soccer simulation 2d team description paper 2021," arXiv preprint arXiv:2206.02310, 2022.