MIE1075 Artificial Intelligence for Robotics I

Soccer Playing Robot

Team Name	**Soccer Playing Robot**
Team Members	**Student Number**
Pengsong Zhang	1008701460
Zixuan Wan	1005016199
Zhuolun Song	1003943236
Olung Siu	1009745304

1. Problem Definition

1.1 Introduction

Artificial intelligence (AI) development has been increasing significantly in recent years, driven by advances in machine learning algorithms and hardware capabilities. While there are many different possible applications for AI, our team is particularly interested in implementing it in soccer. Sport is an important part of human culture, where we learn to test our physical limits and work as a team.

In soccer, the arrangement of 11 players on the field, known as the formation, is developed over years of playing and practicing. Formations like 4-4-2 (4 defenders, 4 midfielders, and 2 forwards) and 4-3-3 are chosen based on a team's strengths, weaknesses, and tactical objectives, as well as the strengths and weaknesses of their opponent. Deciding on the best strategy for each game is extremely challenging for coaches, as there is no formula for success. However, with machine learning, AI could teach themselves to play soccer. We believe that in future AI with machine learning configured with data on a team's and its opponents' strengths and weaknesses could provide effective statistical assistance to coaches by running thousands or even millions of simulated matches in the background.

While there is rapid growth in developing AI with different machine learning algorithms in sport. In this project we are going to dive into implementing machine learning on robot soccer playing, and simulate a real-world competitive soccer match.

1.2 Objective

The main task of the project is to design and develop an AI self-teaching robot that is able to learn to play soccer in a virtual environment, and to create a virtual environment for training and simulation that accurately replicates the rules and dynamics of a real-world soccer match. To train the AI self-teaching robot using reinforcement learning or another suitable machine learning algorithm. lastly to enable the AI self-teaching robot to work as a team with other robots to simulate a competitive soccer match in the virtual environment.

While the sub task is to identify the most effective training method by experimenting with different algorithms and settings, based on the performance of the robots in the virtual environment. and to demonstrate the capabilities of the AI self-teaching robot and the effectiveness of the training method through simulations of real-world soccer matches in the virtual environment.

1.3 Proposed Solution

Our proposed solution for a robot in a soccer game is to use visual feedback and planning and control systems to enable the robot to interact with and understand its environment, additionally a virtual physical environment to allow them to perform visual soccer matches.

The visual feedback system for our robot will use image processing and classification for visual navigation, allowing the robot to avoid obstacles and stay within bounds. It will also use object detection to recognize and track objects such as the ball, teammates, and opponents, enabling the robot to perform tasks related to these objects. Additionally, we will use localization and Kalman filter to determine the position of objects and estimate the robot's location in the map over time.

To enable the robot to make decisions and take actions based on the data provided by the visual feedback system, we will need to use path planning and motion control methods. To perform self-teaching tasks, our proposed method is to use machine learning to train the robot to make decisions and take actions based on data, additionally reinforcement learning algorithms to adapt to changing environments and goals.

To perform a soccer match simulation, we will need to build a virtual physical environment and model. The environment should include models of a soccer pitch with two goals at each end, football and the soccer playing robot. In addition, to build the intelligent systems and behaviors for the players into the simulation, we will need an AI platform with AI-related features and tools. This platform should provide tools for developing and training the robot, as well as features for building and running simulations.

By combining the above approaches, our proposed solution will enable a robot to navigate and interact with its environment effectively in a soccer game, allowing it to perform training in simulation to achieve its goals.

Assumption

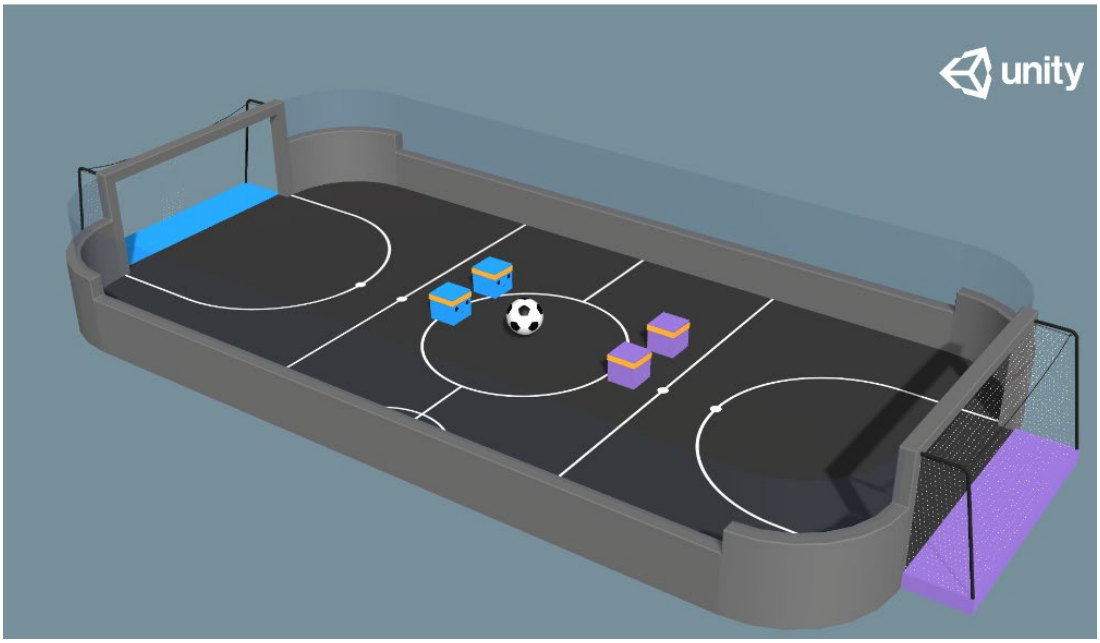
The following assumptions have been made in order to simplify the model and make it more manageable:

- The player and the ball are rigid bodies.
- The sense of vision and hearing is simulated by rays emitting from the body.
- The player and ball motion and movement is limited to forward, backward, and rotation only, and is constrained to a 2D horizontal plane.
- No offside, penalties or concern
- There are walls to limit the ball and robot to stay in the field
- There is no effect of wind or air resistance in the environment.

Additionally, we assume that the project has the necessary resources and computer power to develop the AI self-teaching robot and create the virtual environment for training and simulation.

Basic simulation platform

Throughout the design process, we have identified an open-source project called ML-Agents[1] as a suitable starting point for our soccer project, which uses reinforcement learning algorithms to achieve self-teaching soccer playing. The project utilizes the Unity game engine to create a visual simulation and the Unity ML-Agents toolkit and PyTorch library to develop the model and reinforcement learning algorithm.



We believe that this existing project, which provides a basic 2v2 soccer game, is well-suited to our needs and can serve as the foundation for our goal. Our approach involves building upon this 2v2 learning environment and experimenting with our own learning algorithms, with the ultimate goal of expanding the game to a more complex 5v5 model featuring two teams of 5 robots trained by different algorithms, In addition, we plan to enhance the performance of the robots by assigning them positions such as goalie, attacker, and defender.

2. Literature Review

2.1 Image Processing

2.1.1 Image classification

Image classification is a task in computer vision, which involves assigning labels or categories to input images according to their contents. There are many different methods for image classification, including traditional machine learning methods and deep learning methods.

Traditional machine learning methods for image classification usually involve extracting features from input images, and then using these features as input to classifiers (such as support vector machines or decision trees). These methods may be effective, but they often require manual feature engineering, which may take a long time and may not always produce good results.

Deep learning methods can automatically learn features from input data. These methods usually use convolutional neural networks (CNN) or other types of neural networks to classify images. Neural networks are particularly suitable for image classification tasks because they can capture the spatial relationship between pixels in an image and learn features of multiple scales. Many CNN architectures have been developed for image classification, including VGG[2], ResNet[3], and DenseNet[4]. These architectures differ in the number and type of layers used and the way input data is processed. Some architectures (such as ResNet) use jump connections or other technologies to enable the network to learn deeper representations, while other architectures (such as DenseNet) use dense connections to enable the network to learn more effectively.

In general, the depth learning method has achieved the most advanced results in many image classification tasks and is now the choice of many researchers and practitioners. However, it is important to carefully select the appropriate model architecture and training technology for a given task, because different architectures may have different trade-offs in terms of accuracy, efficiency and versatility.

2.1.2 Object detection

Object detection is a computer vision task, which involves identifying and locating objects in images or videos. Several algorithms for object detection have been developed, each of which has its own advantages and disadvantages. There are several object detection algorithms that have been developed in recent years, including Fast RCNN[5], Faster RCNN[6], Mask RCNN[7], SSD[8], YOLO[9], YOLOv3[10], YOLOv7[11], and PP-YOLO[12]. These algorithms use convolutional neural networks (CNNs) to analyze images and identify objects within them. Fast RCNN and Faster RCNN both involve stages of region proposal and classification/regression to identify and refine bounding boxes around objects. Mask RCNN is an extension of Faster RCNN that also

generates pixel-level masks for each identified object. SSD is a single-stage object detection algorithm that directly predicts bounding boxes and class probabilities from the input image using a CNN. YOLO and its later versions (YOLOv3, YOLOv7) are single-stage object detection algorithms that predict bounding boxes and class probabilities directly from the input image using a CNN and divide the image into a grid for analysis. PP-YOLO is based on the YOLO architecture and uses a pyramid network to improve the accuracy and speed of object recognition. Here some details of some algorithms:

2.1.2.1 Faster RCNN (Regions with Convolutional Neural Networks)

Shaqing Ren et al. created a faster RCNN in 2015. Faster and more effective region suggestion makes it an improvement over Fast RCNN. Instead of using separate techniques like selective search, faster RCNN uses CNNs to generate region recommendations directly from input photos. This makes the method faster than Fast RCNN because it can complete the region proposal and classification/regression stages in a single pass. Similar to Fast RCNN, Fast RCNN classifies and improves bounding boxes using a CNN.

2.1.2.2 YOLOv3 (You Only Look Once, version 3)

This is the third version of the YOLO algorithm, which was created in 2018 by Joseph Redmon et al. It is more precise and quicker than its predecessors. YOLOv3 predicts bounding boxes and class probabilities directly from input photos using a sequence of neural layers. To boost accuracy, it also introduces novel network design, anchor boxes, and a feature pyramid network.

2.1.2.3 PP-YOLO (Efficient Detector with Progressive Pyramid Networks)

The creators of this algorithm are Qi Han et al., 2020. While using a pyramid network to steadily increase the accuracy and speed of object recognition, it is based on the YOLO architecture. An input image is processed at various scales by each of the numerous sub-networks that make up a pyramid network. The lateral connections between the sub-networks enable them to communicate features and raise object detection precision. High accuracy is possible with PP-YOLO at a comparatively cheap computational cost.

2.2 Localization

Location refers to the process of determining the position of an object in space. This can be achieved through various methods such as GPS, triangulation, and dead reckoning. SLAM (simultaneous localization and mapping) is a problem in robotics and computer vision that involves building a map of an environment while simultaneously determining the location of a robot or camera within that environment. SLAM algorithms are used in a range of applications including autonomous robotics, augmented reality, and mobile devices. The main difference between localization and SLAM is that while localization algorithms usually only focus on finding the location of objects, SLAM algorithms also construct a map of the environment and utilize it to improve the accuracy of position estimates. SLAM algorithms can also use this map to correct

errors in position estimates and resolve ambiguities between different potential positions. In summary, location is the process of determining the position of an object, while SLAM involves building a map of the environment and concurrently determining the location of an object within that environment. SLAM is a problem in robotics and computer vision that involves building a map of an environment while determining the location of a robot or camera within that environment. SLAM algorithms have a variety of applications including autonomous robotics, augmented reality, and mobile devices. Many different algorithms have been developed for SLAM, each with its own strengths and weaknesses. There are several approaches to simultaneous localization and mapping (SLAM), including Kalman filter-based SLAM and extended Kalman filter (EKF) SLAM, graph-based SLAM, visual SLAM, LIDAR SLAM, and multi-sensor SLAM. Kalman filter-based SLAM[13] uses a Kalman filter to estimate the pose of a robot and map its environment. EKF SLAM is similar but is able to handle nonlinearities in the system dynamics and measurement models. Graph-based SLAM[14] represents the map and robot pose as a graph, which is iteratively updated with measurements from sensors such as laser rangefinders or cameras. Visual SLAM[15] algorithms use visual data from cameras or other vision sensors to estimate the pose of a robot and build a map of its environment. LIDAR SLAM[16] algorithms use LIDAR data to determine a robot's position and create a map of its environment. Multi-sensor SLAM algorithms use data from multiple sensors to estimate the pose of a robot and build a map of its environment. These approaches can be used in a variety of applications and have been implemented on various robot platforms. Here are some most common algorithms:

2.2.1 Kalman filter-based SLAM

These techniques employ Kalman filters to estimate the robot's pose (position and orientation) as well as a map of the environment. The Kalman filter is a recursive algorithm that estimates the state of a system using a predictive-correction method. In the Kalman filter-based SLAM, the system state comprises the robot's pose and map. Typically, the algorithm works by anticipating the robot's pose and map based on prior states and the robot's movements, and then correcting those predictions based on measurements from sensors such as laser rangefinders or cameras. To execute the prediction and correction processes, the Kalman filter employs a linearized model of the system dynamics and a measurement model. SLAM techniques based on Kalman filters are generally simple and computationally efficient, although they may perform poorly in noisy or nonlinear settings.

2.2.2 Visual SLAM

Visual SLAM algorithms use visual data from cameras or other vision sensors to estimate the pose of a robot and build a map of its environment. There are several different approaches to visual SLAM, including feature-based methods, direct methods, and dense methods. Feature-based methods identify unique features in the image, such as corners or edges, to estimate the pose and map of the robot. These methods often use computer vision techniques like feature matching and

motion structure. Direct methods, on the other hand, use raw pixel intensities from the image to estimate the robot's pose and map. These methods often utilize machine learning techniques such as convolutional neural networks. Dense methods use raw pixel intensities to create a dense reconstruction of the environment. These methods often utilize computer vision techniques like multi-view stereo. Visual SLAM algorithms can take advantage of rich visual information, but may be sensitive to lighting conditions and require more computation than other types of SLAM algorithms.

2.2.3 LIDAR SLAM

Lidar SLAM (light detection and ranging SLAM) is an algorithm that uses lidar data to determine a robot's position and create a map of its environment. Lidar is a technology that uses laser light to measure distance and create a 3D point cloud representation of the environment. LiDAR SLAM algorithms can use this 3D point cloud to estimate the robot's position and map. Like other SLAM algorithms, lidar SLAM can use a variety of techniques, including feature-based methods, direct methods, and graph-based methods. Feature-based methods use unique features in lidar point clouds, such as corners or edges, to estimate the robot's position and map. These methods often rely on computer vision techniques such as feature matching and structure from motion. Direct methods use the raw 3D points in the lidar point cloud to estimate the robot's position and map. These methods often rely on machine learning techniques such as convolutional neural networks. Graph-based methods represent the robot's map and location as a graph, and estimate the map and location by iteratively updating the graph with measurements from lidar sensors. Lidar SLAM algorithms have the advantage of using precise 3D measurements, but can be sensitive to occlusions and may require more computational resources than other types of SLAM algorithms.

2.3 Pose Estimation

Pose estimation is the problem of identifying an object's position and orientation (i.e., pose) in space. Pose estimation techniques are utilized in many different applications, including as robots, augmented reality, and computer vision. There are numerous algorithms for pose estimation, each with its own set of strengths and disadvantages. Following are some examples of pose estimation algorithms:

2.3.1 Perspective-n-Point (PnP) algorithms

PnP techniques[17] use known 3D coordinates and their matching 2D projections to estimate an object's posture within an image. This method relies on a perspective projection model where it is assumed that a single pinhole camera would be used to project 3D points onto an image plane. You require a set of 3D points and the matching 2D projections in the image in order to apply the PnP algorithm. The algorithm then determines the camera posture that best fits the observed 2D

projection to estimate the pose of the object. There are two types of PnP algorithms: direct and iterative. Direct PnP algorithms solve the pose directly using the perspective projection model and typically use closed-form solutions, meaning they can compute the pose in one step rather than an iterative optimization process. However, direct PnP algorithms may not be as accurate as iterative PnP algorithms, especially when the perspective projection model is inaccurate. Iterative PnP algorithms use an iterative optimization process to refine the pose estimate. They start with an initial pose estimate and then adjust the pose iteratively until it best fits the observed 2D projections. Iterative PnP algorithms may be more accurate than direct PnP algorithms, but may require more computation.

2.3.2 Structure from Motion (SfM) algorithms

The SfM algorithm[18] uses a sequence of images to reconstruct the 3D structure of an object and estimate its pose. The SfM algorithm is based on the idea of triangulating the 3D position of a point in the world based on the 2D projection of the point in the image. To use the SfM algorithm, a sequence of images of objects or scenes is required. The algorithm then estimates the 3D structure of the object or scene and the camera pose for each image by triangulating points in the world based on the 2D projections in the images. SfM algorithms can be monocular (using a single camera) or multi-view (using multiple cameras). Monocular SfM algorithms use a single camera to estimate the 3D structure and pose of an object or scene. These algorithms typically rely on camera motion to provide depth information and disambiguate between different possible 3D structures. Monocular SfM algorithms may be more error-prone due to lack of depth information, but may be simpler and require less computation than multi-view SfM. Multi-view SfM algorithms use multiple cameras to estimate the 3D structure and pose of an object or scene. These algorithms can use additional views to provide more accurate depth information and disambiguate between different possible 3D structures. Multi-view SfM algorithms may be more accurate than monocular SfM, but may require more computation.

2.3.3 Visual Odometry (VO) algorithms

The VO algorithm[19] uses a sequence of images from a single camera to estimate a subject's posture by tracking the movement of points or features in the image. VO algorithms are often based on the idea of tracking the movement of points or objects in an image and using that movement to estimate camera posture. VO algorithms can be feature-based or simple. VO based algorithms track the movement of various features in an image, such as corners or edges. These algorithms typically use computer vision techniques, such as motion-word structure and feature matching, to track feature motion and estimate camera pose. Feature-based VO algorithms may be more accurate than direct VO algorithms, but may be sensitive to the presence of repetitive or non-textured regions in the image. The live VO algorithm uses the raw pixel intensities of the image to estimate the camera pose. These algorithms often use machine learning techniques, such as convolutional neural networks (CNN), to learn camera movement directly from images. Direct VO

algorithms may not be as accurate as feature-based VO algorithms, but may be less sensitive to the presence of repetitive or non-textured regions in the image.

2.3.4 Visual Inertial Odometry (VIO) algorithms

The VIO algorithm[20] combines visual data from the camera with data from an inertial measurement unit (IMU) to estimate the subject's posture. The IMU is a sensor that measures an object's acceleration and angular velocity, which can be used to estimate an object's acceleration and angular velocity over time. The VIO algorithm uses IMU data to provide additional information about the subject's movement, which improves the accuracy of posture estimation. VIO algorithms can be loosely coupled or tightly coupled. Loosely coupled VIO algorithms process vision data and IMU data separately, and then combine the results. These algorithms often merge vision and IMU data using techniques such as Kalman filters or extended Kalman filters. Loosely coupled VIO algorithms may be simpler and require less computation than tightly coupled VIO algorithms, but may be less precise. VIO algorithms are tightly coupled together to process vision and IMU data. These algorithms often merge vision and IMU data using techniques such as non-linear optimization or maximum likelihood estimation. Tightly coupled VIO algorithms can be more precise than loosely coupled VIO algorithms, but may require more computation.

2.4 Path planning and Motion control methods

Path planning and motion control are two closely related but distinct problems in robotics and control engineering. Path planning involves finding a path for a robot or vehicle from an origin to a goal point while avoiding obstacles and satisfying a set of constraints. Path planning algorithms are used to generate a plan, or set of instructions, specifying the sequence of actions that a robot or vehicle should take to achieve a goal. Motion control, on the other hand, involves controlling the movement of a robot or vehicle along a given path. Motion control algorithms are used to generate control signals (eg, motor commands, actuator positions) to drive a robot or vehicle along a path. In other words, path planning involves generating a plan or set of instructions for a robot or vehicle, while motion control involves executing that plan or set of instructions. Path planning and motion control are often used together in robotics and control systems, with the output of the path planning algorithm used as input to the motion control algorithm. For example, a path planning algorithm can generate a set of waypoints for a robot to follow, and then a motion control algorithm will control the robot's motion to follow the waypoints.

Finally, this section will describe how Path Planning and Motion Control are achieved by discussing the algorithms that the team used in Reinforcement Learning and reward functions, including Multi-Agent DDPG, POCA, as well as PPO. Moreover, the ELO rating system is discussed and used to evaluate the performance of the Soccer-Playing agents in the environment.

2.4.1 Path Planning

Path planning is the problem of finding a path for a robot or vehicle from an origin to a goal point while avoiding obstacles and satisfying a set of constraints. Path planning algorithms are used in a variety of applications, including robotics, autonomous vehicles, and manufacturing. Many different path planning algorithms have been developed, each with its own strengths and weaknesses. There are several approaches to path planning for robotics, including graph search algorithms[21], potential field algorithms[22], motion planning algorithms, hybrid A* algorithms[23], sampling-based algorithms[24], decision-theoretic planning algorithms[25], game-theoretic planning algorithms[26], and evolutionary algorithms[27]. Graph search algorithms represent the environment as a graph and use algorithms such as depth-first search, breadth-first search, or A* search to find a path from the start state to the end state. Potential field algorithms use a scalar field to guide the robot's motion towards the goal or away from obstacles by moving in the direction of the field's gradient. Motion planning algorithms generate continuous paths through the environment and can be deterministic or probabilistic. Examples include the Rapid Exploration Random Tree (RRT) and Probabilistic Roadmap (PRM) algorithms. Hybrid A* algorithms combine graph search and motion planning by using a graph representation of the environment and A* search to find paths, but also generating continuous paths by interpolating between graph vertices. Sampling-based algorithms sample the environment to create a representation of free space and use this representation to plan paths. These algorithms can also be deterministic or probabilistic, with examples including RRT and PRM. Decision-theoretic planning algorithms model the environment as a Markov decision process (MDP) and use techniques such as dynamic programming or reinforcement learning to find optimal actions at each state. Game-theoretic planning algorithms model the environment as a game and use techniques such as minimax or Nash equilibria to find strategies for the robot. Evolutionary algorithms use techniques such as selection, crossover, and mutation to evolve a population of possible paths and optimize an objective function. The following are some details of some algorithms:

2.4.1.1 Potential field algorithms

These algorithms use potential fields to guide the robot's motion. The potential field is a scalar field that represents the attraction or repulsion of the robot to different parts of the environment. The robot moves in the direction of the gradient of the potential field, which guides the robot towards the goal or away from the obstacle.

2.4.1.2 Hybrid A* algorithm

The algorithm is a combination of graph search and motion planning. It uses a graph to represent the environment and uses A* search to find paths through that graph, but it also generates continuous paths through the environment by interpolating between the vertices of the graph.

2.4.1.3 Sampling-based algorithms

These algorithms sample the environment to create a representation of free space, and then use this representation to plan paths. Sampling-based algorithms can be either deterministic or probabilistic. Algorithms based on deterministic sampling generate a single path through the environment, while algorithms based on probabilistic sampling generate a set of possible paths and assign a probability to each path. Examples of sampling-based algorithms include the Rapid Exploration Random Tree (RRT) algorithm and the Probabilistic Roadmap (PRM) algorithm.

2.4.2 Motion Control

2.4.2.1 PID (Proportional-Integral-Derivative) control

PID control[28] is a widely used control algorithm that uses feedback to regulate the output of a system to follow a reference signal. It consists of three terms: a proportional term, an integral term, and a derivative term. The proportional term adjusts the output of the system based on the current error between the reference signal and the output. The integral term adjusts the output based on the error accumulated over time. The derivative term adjusts the output based on the rate of change of the error.

2.4.2.2 Model predictive control (MPC)

MPC[29] is a control algorithm that predicts the future behavior of a system and uses that prediction to optimize control signals. The MPC algorithm uses a model of the system to predict the future output of the system based on the current control signals and the current state of the system. They then optimize the control signal to minimize a cost function that takes into account the reference signal, predicted output, and system constraints.

2.4.2.3 Linear quadratic regulator (LQR)

LQR[30] is a control algorithm that optimizes control signals to minimize cost in constrained linear systems. The LQR algorithm uses dynamic programming to find the optimal control signal that minimizes a cost function within a finite range.

2.4.2.4 Sliding mode control (SMC)

SMC[31] is a control algorithm that uses discontinuous control signals to achieve precise control of the system. SMC algorithms use a sliding mode surface to compare a reference signal and the output of the system, and they generate control signals that drive the system onto the sliding mode surface.

2.4.2.5 Adaptive control

Adaptive control algorithms[32] adjust the control signal according to the changing characteristics of the system. These algorithms use adaptive elements to learn the changing characteristics of the system and update the control signals accordingly.

2.4.3 Algorithms for Reinforcement Learning

2.4.3.1 MADDPG

MADDPG (Multi-Agent Deep Deterministic Policy Gradient) is a reinforcement learning algorithm that is used to train multiple agents to interact and cooperate with each other in a multi-agent environment[33]. It is an extension of the DDPG (Deep Deterministic Policy Gradient) algorithm, which is a model-free, off-policy algorithm that uses function approximation to learn the optimal policy for a given task.

In MADDPG, each agent has its own policy and learns to optimize its own actions independently of the other agents. However, the agents also need to consider the actions of the other agents in the environment, as they can affect the overall reward. To address this, MADDPG uses a centralized training process, where a central Critic network is used to estimate the value function for each agent. The Critic network receives information about the actions and observations of all the agents in the environment, and uses this information to learn the optimal value function for each agent.

During training, the agents collect experiences in the form of transitions (observations, actions, rewards, and next observations) and store them in a replay buffer. The transitions are then sampled from the replay buffer and used to update the policies and value functions of the agents using gradient descent.

2.4.3.2 POCA

The (MA-POCA) Multi-Agent POsthumous Credit Assignment algorithm is a machine learning approach that allows multiple agents to learn from each other's experiences in a collaborative manner. It is based on the concept of posthumous credit assignment, which refers to the ability of an agent to assign credit to other agents for their contributions to a task[34]. Each agent has its own set of learning parameters and experiences. When an agent completes a task, it can assign credit to other agents for their contributions. This credit assignment process is done through a credit assignment matrix, which is updated based on the agent's experiences and the outcomes of the task. This is particularly useful in complex, dynamic environments where the agents need to be able to adapt quickly and effectively to changing conditions. An illustration of MA-POCA, introduced by Unity is shown below.

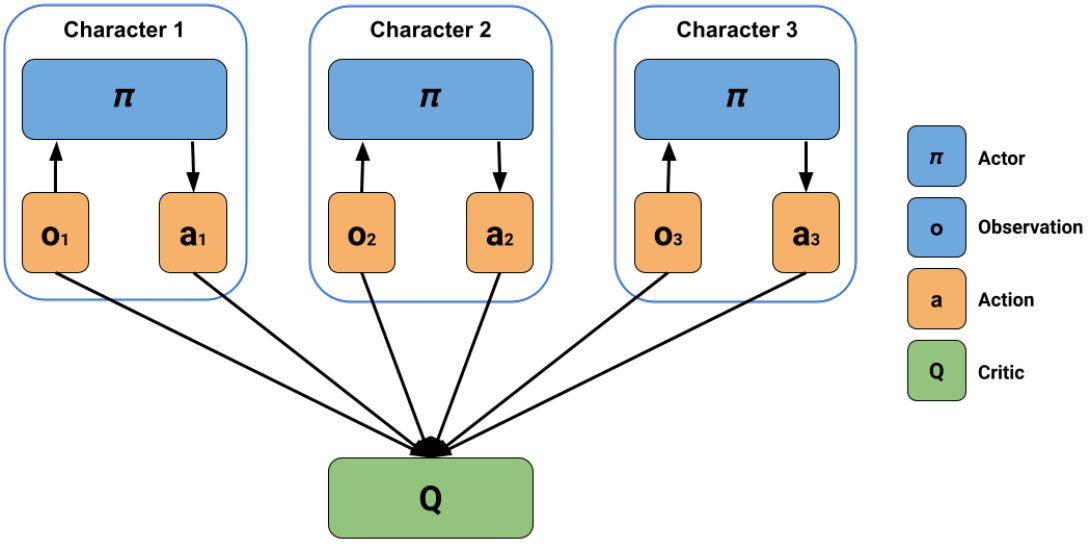


Figure 2.4.3.2. Illustration of MA-POCA algorithm

The MA-POCA algorithm allows for more efficient learning in multi-agent systems, as it allows each agent to learn from the experiences of other agents. It can be applied in a variety of settings, including multi-agent control systems, social networks, and games. Overall, the MA-POCA algorithm is a useful tool for enabling collaborative learning in multi-agent systems. It allows agents to learn from each other's experiences and improve their performance over time, leading to more effective and efficient outcomes.

2.4.3.3 PPO

Proximal Policy Optimization (PPO) is a reinforcement learning algorithm that can be used to train agents to take actions in an environment in order to maximize a reward signal[35]. The PPO algorithm is a new type of Policy Gradient algorithm that proposes a new objective function that can update small batches in multiple training steps, which solves the problem that the step size in the Policy Gradient algorithm is difficult to determine. The basic idea behind PPO is to find a policy that is "close" to the current policy, but that still improves performance. To do this, PPO uses a "proximal" objective function that penalizes large changes to the policy, while still allowing for some improvement. This helps to ensure that the updated policy is not too different from the current policy, which can help to reduce the risk of destabilizing the learning process[35]. It is also worth noting that the PPO algorithm has been adopted by OpenAI as the first choice in its algorithm for reinforcement learning research.

2.4.4 Reward Function

A reward function is a mathematical function that is used in reinforcement learning to assess the quality of an action taken by an agent in the environment[36]. It is used to guide the agent, or the learning system, in choosing the most beneficial actions to take in order to achieve a desired goal. It is defined based on the objectives and goals of the agent by the users and is used to evaluate the

success or failure of the agent's actions. The reward function assigns a numeric value to each action or state, with higher values indicating more favorable outcomes.

The reward function is an essential component of reinforcement learning because it provides the agent with a way to measure the success or failure of its actions. It helps the agent to understand what actions are likely to lead to positive or negative outcomes. The reward function is also adjustable, allowing the agent to fine-tune its behavior over time as it learns more about the environment and its own capabilities. The utilization and modification of reward functions for our project will be discussed in detail later in section 3.4.2.

2.4.5 ELO Rating System

ELO, which is short for ELO rating system, is a method for calculating the relative skill levels of players in two-player games[37]. It was originally developed by Arpad Elo, a Hungarian-American physicist and chess grandmaster, to rank chess players based on their performance in tournaments and the demonstration of rating scores are shown in the figure below[38]. In machine learning, ELO is often used as a ranking system for evaluating the performance of different algorithms or models. It works by assigning each algorithm or model a rating based on its past performance, with higher ratings indicating better performance. When two algorithms or models are compared, the one with the higher rating is expected to win, and the difference in ratings is used to calculate the probability of a win or loss.

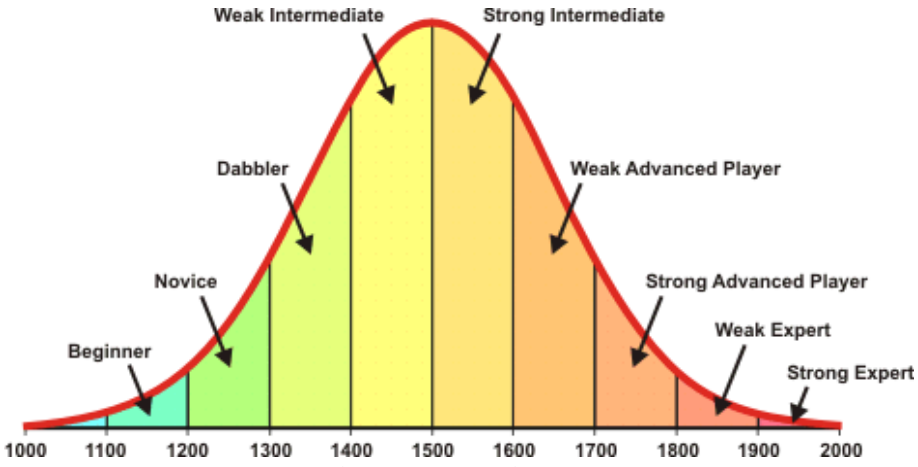


Figure 2.4.3. ELO rating score

ELO rating systems are commonly used in machine learning competitions and tournaments, as they allow for fair and accurate comparisons of different algorithms or models. One of the main advantages of ELO is that it is able to adjust ratings based on the relative strength of an opponent, meaning that an algorithm or model that performs well against strong opponents will receive a higher rating than one that performs poorly, which will result in a more accurate representation of an algorithm or model's true performance. Moreover, if a model has a longer training time and is able to learn from more data, it may have a higher Elo rating and be considered a more skilled model. However, it is important to note that training time is not the only factor that determines a model's Elo rating, as the quality and relevance of the data also play a significant role.

2.5 Unity Environment

In order to develop the AI simulation, and software infrastructure that is designed to support the development, deployment, and management of AI applications and models is required, and they generally provide a range of tools and services that enable developers to build, train, and deploy machine learning models. While there are different platforms available, Unity was selected for building our AI environment.

2.5.1 Unity AI developing artificial intelligence (AI) Platform

Although Unity is not specifically designed as an artificial intelligence (AI) platform, it does include a range of AI-related features and tools that we are able to use to build intelligent systems and behaviors into a visual simulation sense. And in recent years, there has been a growing body of research on the use of Unity for AI development, covering a wide range of topics and applications. The paper “Unity: A General Platform for Intelligent Agents"[] presents us with an overview of the use of Unity as a platform for developing intelligent agents and discusses the capabilities and performance of Unity. The various features and tools provided by Unity that are relevant for AI development include support for a variety of programming languages, an extensive API, and tools for visual scripting and debugging. While a series of experiments to evaluate the performance of Unity is conducted for AI development. It is found out that Unity could provide good performance for AI development and was able to scale to larger and more complex environments.

2.5.2 Unity ML-Agents tool kit

The paper, “Deep Reinforcement Learning in Agents’ Training: Unity ML-Agents”,[] presents ML-Agents, a Unity plugin that provides a framework for training intelligent agents using reinforcement learning. It includes a range of pre-built environments and tools for training and evaluating agents and is particularly well-suited for use with deep reinforcement learning algorithms. The various features and capabilities of ML-Agents, and how it can be used to train

intelligent agents for a variety of applications, including games, simulations, and robotics is described. After all these papers influence how, we decide to apply them to extend and build up our project sense.

2.5.3 Set up the simulation sense

The environment is created with following steps:

1. Start up a new Unity Project with the Unity editor.
2. Install the ML-Agents toolkit.
3. Design the game: configuration such as number of players, size of the field, rule.
4. Set up an ML-Agents environment: creating a scene that serves as a "sandbox" for training machine learning models.

3. Methodology

Here, we use the deep learning method to develop the algorithm of object detection, pose estimation, reinforcement learning. And, we also developed the path planning algorithm. Most importantly, we try to use reinforcement learning algorithms to develop the End to End model to train the player to play soccer by themselves.

3.1 Image Processing

As shown in Fig 3.1.1, we use the object detection algorithm PPYOLO-V2[39] as our basic model to develop our object detection algorithm. PP-YOLOv2 divides the input image into smaller patches and processes each patch in parallel on multiple CPU cores, and uses a modified version of the YOLOv2 network architecture that is optimized for parallel processing. It is a fast and accurate object detection algorithm that has been used in a variety of applications, including image and video analysis, robotics, and security systems. PP-YOLOv2 is a variant of the YOLO object detection algorithm that was designed to be a practical and efficient object detector for real-time applications. It incorporates techniques such as the Path Aggregation Network, the Mish activation function, larger input sizes, and an IoU aware branch to improve its accuracy and robustness. These features make PP-YOLOv2 a strong object detection algorithm that can be used for a variety of practical applications.

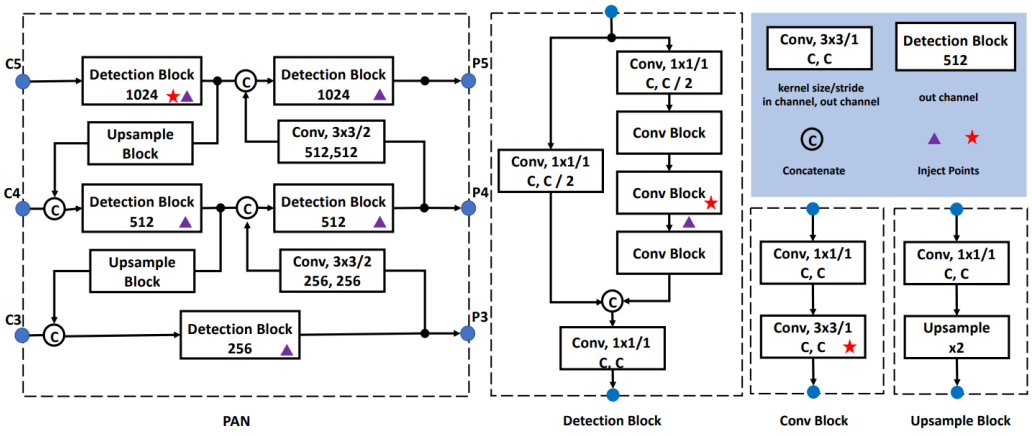


Fig 3.1.1: The architecture of PP-YOLOv2's detection neck

3.2 Localization

We know that the object detection algorithm in image processing can identify the position of the target in the field of view. However, the field of view observed by the agent is generally a local field of view, so we need to reconstruct the map. By extracting the features of the local field of view frame by frame, and then performing image registration, after the registration is completed, fusion is performed to form a global map, and finally Realize the positioning of the target on the global map. The main process contains: 1,collect partial field of view pictures with overlapping

fields of view. 2, match with sift features or any features you are familiar with. (with feature map). 3, use the RANSAC[40] method to remove mismatched point pairs. (attached feature point map after screening). 4, give the transformation matrix (either affine model or projective model is acceptable, please specify), complete the splicing (forward mapping or reverse mapping is acceptable, please specify), and give the spliced image. 5, fusion the spliced images (any fusion method you like), and attach the fused images.

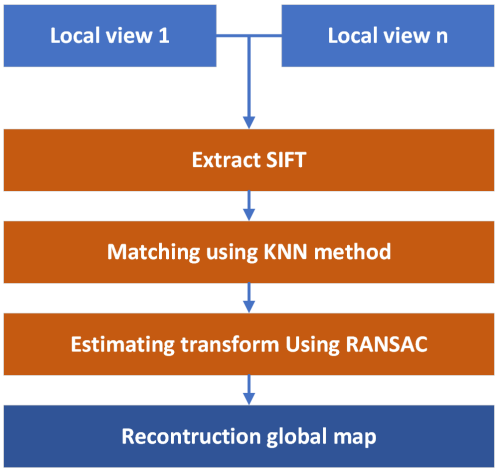


Fig 3.2.1: The architecture of global image reconstruction

After establishing the global map, we only need to map the result of the object detection in the previous step back to the global map to get the global position of the target in the global map.

3.3 Pose Estimation

Although the environment is 3D, because simulating real athletes consumes too much computer power, and we do not have high-performance computers, our athletes can only move in two dimensions in a 3D environment. Therefore, for the posture of athletes, we only detect their azimuth. Here, we use the network according to the paper "PP-LCNet: A Lightweight CPU Convolutional Neural Network"[41]. The angle setting totals 12 classes: 180, 150, 120, 90, 60, 30, 0, -30, -60, -90, -120, -150. As shown in the figure below, we just only modify the output layer from 1000 to 12.

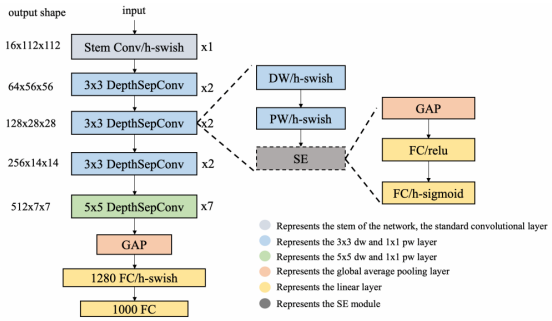


Fig 3.3.1: The architecture of PP-LCNET detection neck

3.4 Path planning and Motion control methods

As we mentioned before in the literature review section, the path planning and motion control of the agents are solved by directly setting up reinforcement learning methods and reward functions. According to 3 reinforcement learning algorithms and 2 reward functions we designed, we came up with a series of 5 designs throughout the project. Which will be explained by the following sections

3.4.1 Traditional planning and control

3.4.1.1 RRT-Connect

We use the RRT-connect algorithm[42] here to generate the path. The basic idea behind the RRT Connect algorithm is to construct two trees, one rooted at the start state and the other at the goal state. The algorithm then repeatedly generates random states and tries to extend both trees towards these random states. If the distance between the two new nodes added to the trees is within a certain threshold, it means that a connection has been found and the algorithm can return a path from the start state to the goal state.

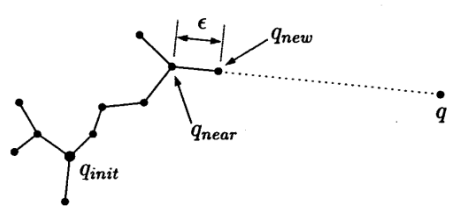


Fig 3.4.1.1.1: The EXTEND operation

3.4.1.2 PID

We used a PID controller to control the angle and position of robots. As shown in Figure below, the controller mainly contains position loop and angle loop. The PID controller calculates an error signal by comparing the desired setpoint to the actual output of the system. It then adjusts the output of the system by applying a control signal based on three components: Proportional (P) control: This component adjusts the output based on the current error signal. It is designed to bring the output closer to the setpoint as the error gets larger. Integral (I) control: This component helps to eliminate steady-state error by taking into account the accumulated error over time. It helps to ensure that the output reaches and stays at the setpoint. Derivative (D) control: This component helps to stabilize the system by predicting future errors based on the rate of change of the error signal. It helps to dampen oscillations and reduce overshoot.

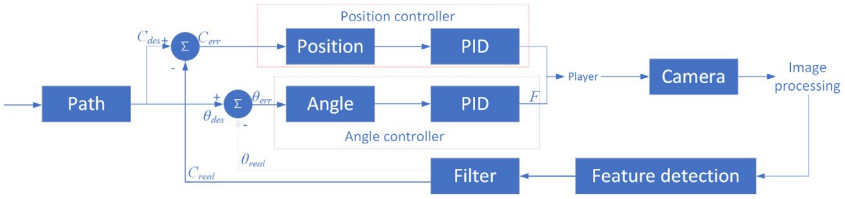


Fig 3.4.1.2.1: Schematic diagram of position loop and angle change controller

3.4.2 Reinforcement learning algorithms

Here, we want to design an End-to-End reinforcement learning algorithm to make the player learn to play soccer by themselves. Unlike other traditional sensing, planning, control algorithms, reinforcement learning can do all these tasks, it has the abilities to sense, path plan and control. First of all, we set the action type and formed the action space.

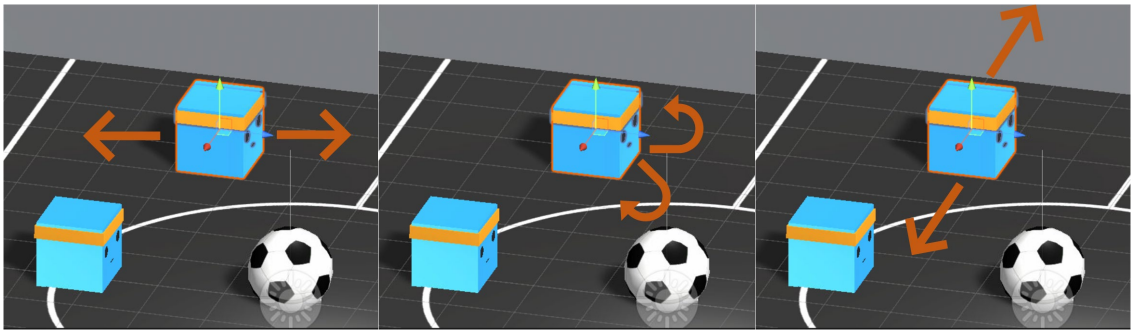


Fig 3.4.2.1: Action design

Discrete actions:

- [1, 1, 1]: Forward, stop, backward
- [1, 1, 1]: Left, stop, right
- [1, 1, 1]: Clockwise, stop, counterclockwise

We originally wanted to use the output results of the deep learning target detection algorithm, map reconstruction algorithm, visual pose estimation and other models as the input of the reinforcement learning model, but due to our limited computing power, we only have ordinary computers without high performance. GPU, so we use the built-in sensor of unity to replace the function of the above algorithm in the observation space.

3.4.2.1 MADDPG

We tried to use the MADDPG reinforcement learning algorithm[33] to train the player to play soccer. As shown in figure below, the model input is the observation space and the output is action space. MADDPG is an algorithm for training decentralized multi-agent systems, in which each agent has its own policy. It is based on the Deep Deterministic Policy Gradient (DDPG) algorithm and is designed to learn continuous control policies in reinforcement learning. MADDPG uses a centralized critic that learns a joint action-value function for all agents, based on the observations and actions of all agents.

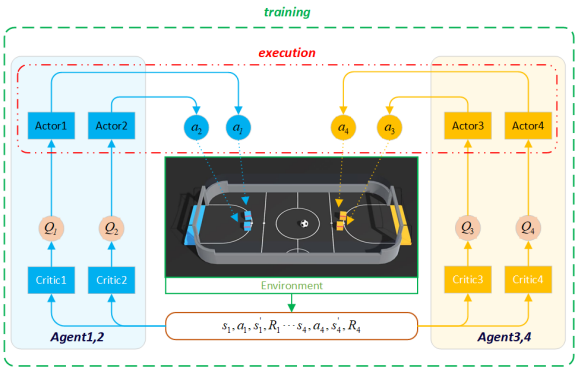


Fig 3.4.2.1.1: Schematic diagram of MADDPG applied on soccer playing

3.4.2.2 POCA

We also tried to use the POCA[34] algorithm to train the player agents to play soccer, the output of the model is a defined action space, and the observation space is obtained by using the ray sensor in unity. The overall structure diagram is as follows:

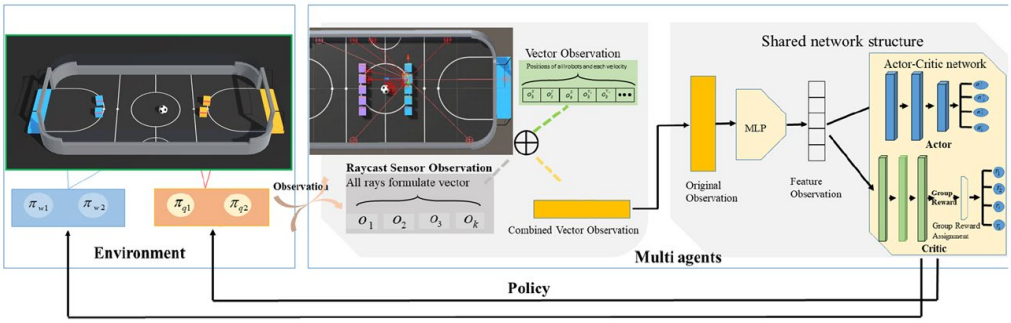


Fig 3.4.2.2.1: Schematic diagram of POCA applied on soccer playing

3.4.2.3 PPO

Here, we also tried to use the PPO[35] algorithm to train the player agents to play soccer, the output of the model is a defined action space, and the observation space is obtained by using the ray sensor in unity. We did not change much of the PPO model, only the action and observation setting was changed to be adapted to the soccer environment.

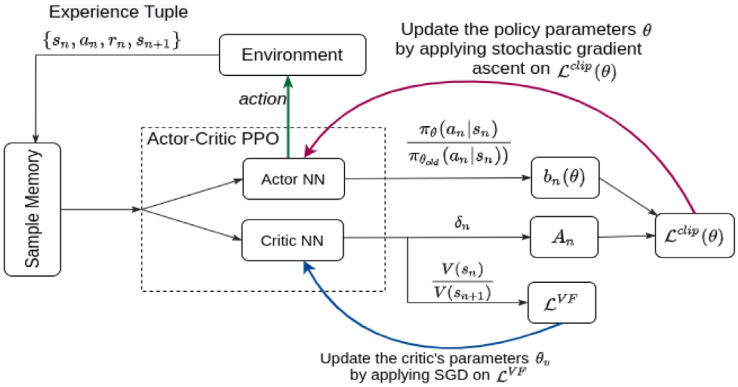


Fig 3.4.2.3.1: Schematic diagram of PPO applied on soccer playing

3.4.3 Reward Function Designs

There are two sets of reward functions we used in this project. We first test on the basic version of the reward function, which is open-sourced on the Unity Soccer_twos environment. Then we modified the reward function and tried to enhance it.

1. **Basic Reward Function**

All players share one reward function. That is, when the ball enters Opponent's goal, it will reward its own group and punish the opponent's Group. Vise versa, when the ball enters

the agent's own goal, it will punish the agent's own group and reward the opponent's group. We can see that the agents are rewarded or punished only when the ball actually enters any goal, and it affects the entire group.

2. Enhanced Reward Function

To improve the reward function, we decided to add a player's self reward on top of the group reward to make agents learn more efficiently. We are building two self rewards for two kinds of agents: Striker and Goalie.

a. Strikers are encouraged to play more offensively, therefore they will be rewarded if the ball is near the enemy's goal and punished if the ball is near his own goal. Besides that, strikers are encouraged to compete with other players and try to get the balls, therefore there will be extra punishments if they didn't reach ball for a long time
b. Goalies are encouraged to play around his own goal and keep the ball away from the goal. Therefore, goalies will be punished for being too far away from his own goal or if the ball is too close to his own goal. On the other hand, goalies will be rewarded if the ball is far away from his own goal.
c. As for the group reward, besides the basic reward function, we focus on the ball's movement as well. That is, if the ball is moving to the opponent's side, we reward the entire group, and vice versa.

3.4.4 Design Process with RL and Reward function

With the above 3 algorithms and 2 reward functions, our design processes are listed below. For each of the following approaches, we modify the training parameters in the .yaml file. The training algorithms are modified by the *"trainer_type"* parameter, the *"learning_rate_schedule"* is set to be constant, and maximum training steps are set to be 50,000,000 steps under the *"max_steps"* variable.

1. 2v2 Soccer playing 1: with algorithm MADDPG and Basic Reward Function

First, we tried the regular 2v2 Soccer playing with the MADDPG algorithm and Basic Reward Function. The model was trained for more than 12 hours and 200,000 steps. However, the ELO score continues to drop and is always below 1200. While in the training process, we noticed that the agents rarely touched the ball. Therefore we concluded that this approach is not suitable and failed.

We believe the reason for this failure is because the MADDPG algorithm uses Deep Deterministic Policy Gradient (DDPG) algorithm, which is known to be effective for learning continuous action spaces[33], while our environment is discrete action space. Although MADDPG would still be used for discrete environments, there might be some details that are not handled well that lead to difficult convergence of the model.

2. 2v2 Soccer playing 2: with algorithm POCA and Basic Reward Function

In the second approach, we tried a new algorithm POCA with the Basic Reward Function to see if there's any difference. It turns out that this approach succeeded, the Self Play ELO score had significant increases, which means the model is training and the agents are learning.

3. **2v2 Soccer playing 3: with algorithm POCA and Enhanced Reward Function**

Then, we try to enhance the reward function to see if the approach 2 can be further improved. We used the design principles for Enhanced Reward Function mentioned in 3.4.2, where for 2v2, all agents follow the striker's personal reward + group reward. As a result, we figured out that the training ELO score for this approach is slightly better than the approach 2.

We then draw a quick conclusion, that POCA is a suitable RL algorithm for training the soccer playing environment, and the Enhanced Reward Function has better performance than the basic version.

We wanted to further extend the game to a more real-life environment, therefore we tried to perform 5v5 soccer playing with the above choices. We not only increased the agent's number, but also added different roles for certain agents: Goalies and Strikers. For 2v2 Soccer playing, every agent is considered as strikers. As for 5v5, we have 3 strikers and 2 goalies.

4. **5v5 Soccer playing 1: with algorithm POCA and Enhanced Reward Function**

Our first approach with 5v5 soccer playing was to use POCA and Enhanced Reward Function. As for 5v5 there are two roles and they both have their own defined reward function, we eventually trained 2 models out: Striker and Goalie.

5. **5v5 Soccer playing 2: with algorithm PPO and Enhanced Reward Function**

Our second approach for 5v5 soccer playing was to use a new algorithm PPO with the Enhanced Reward Function to see if the training results are better. However, the results showed that both goalie and striker didn't perform as well as the first approach.

The detailed simulation results for each design process can be found in Experiment and Results section 4.5: Reinforcement Learning Simulation Results, and we would draw our conclusion there as well.

4. Experiment and Results

In this section, we mainly conduct various experiments, ranging from object detection, map reconstruction, path planning, motion control, and end-to-end models based on reinforcement learning. Due to the limitation of computer hardware resources, although we have done experiments in image processing, pose estimation, map reconstruction, etc., we did not use these models in the final experimental design, because these models will consume a lot of computing power and hardware. Because our resources are limited, we used Ray sensor instead of these algorithms to obtain feedback, and finally applied it to the reinforcement learning model, realized the football model based on the neural network reinforcement learning model, and compared various reinforcement learning models A variety of football game experiments were carried out to obtain the most effective reinforcement learning model.

4.1 Experiment setup

As shown in Fig 4.1.1, the environment was built in the software UNITY, and the computer we used is a laptop (Microsoft Laptop3, I7 cpu, **No GPU**).

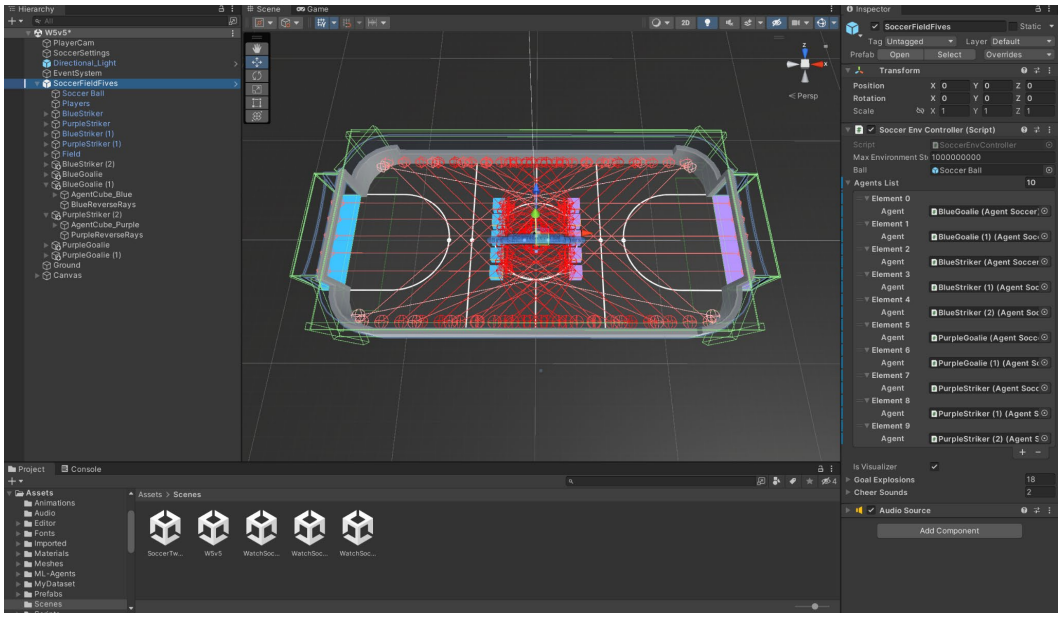


Fig 4.1.1: Use unity to create the 2v2 and 5v5 competition scene

4.2 Object detection

The following is the parameter change curve and result display of the training process of target detection. We can see that the mAP reached 0.60.

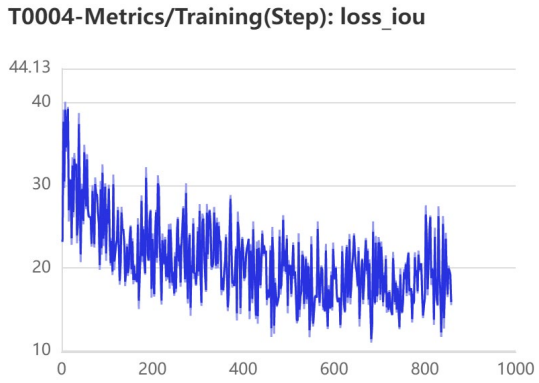


Fig 4.2.1: The loss_iou of training process PPYOLO-V2

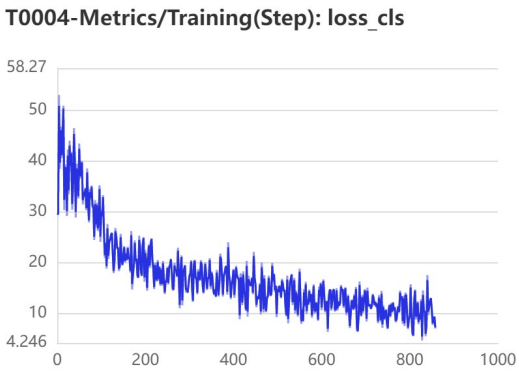


Fig 4.2.2: The loss_class of training process PPYOLO-V2

Fig 4.2.3: The loss of training process PPYOLO-V2

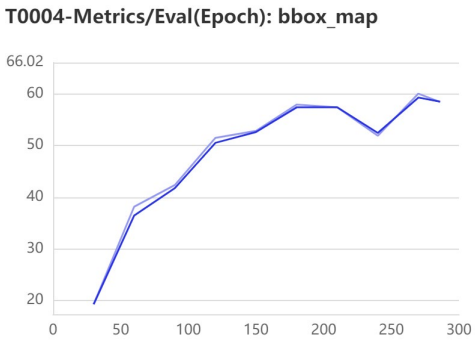


Fig 4.2.4: The bbox mAP of training process PPYOLO-V2

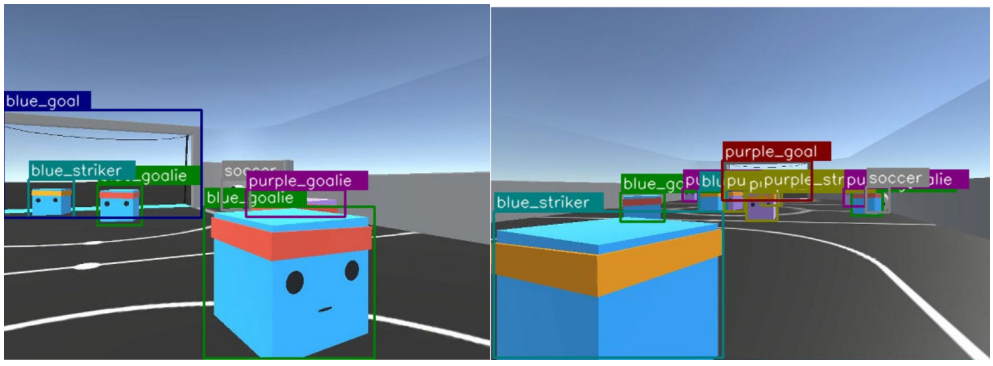


Fig 4.2.5: The detection results of first person view

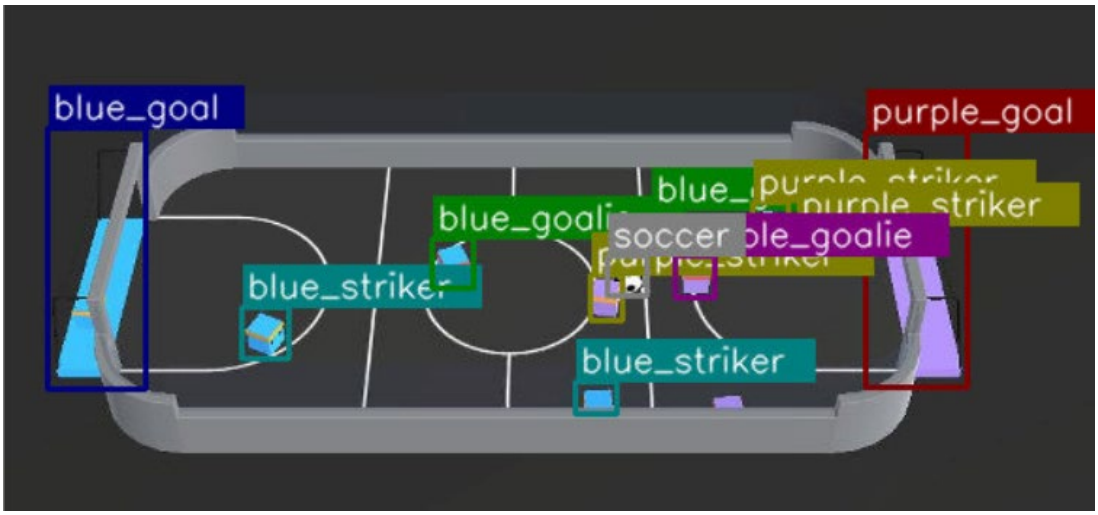


Fig 4.2.6: The detection results of third person view

4.3 Location and pose estimation

4.3.1 Map Reconstruction

As shown in Fig 4.3.1.1, the four small images are the different local views. What we want to do is to restore the global scene according to their characteristics .

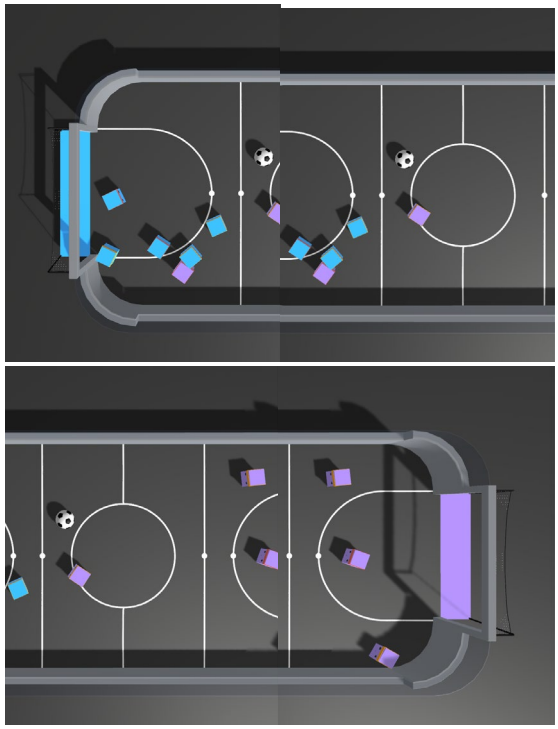


Fig 4.3.1.1: The local view of different location

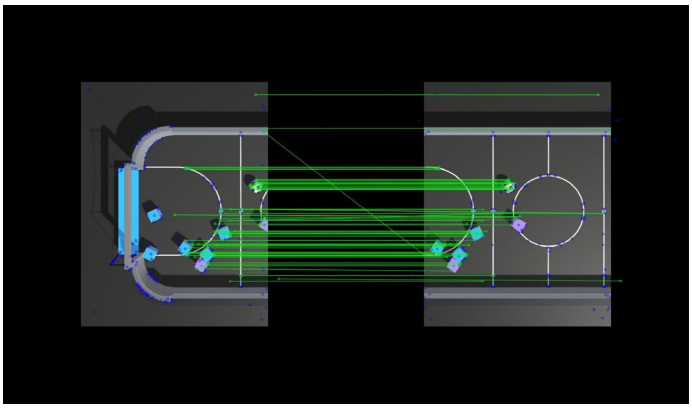


Fig 4.3.1.2: The matching of the first two view images

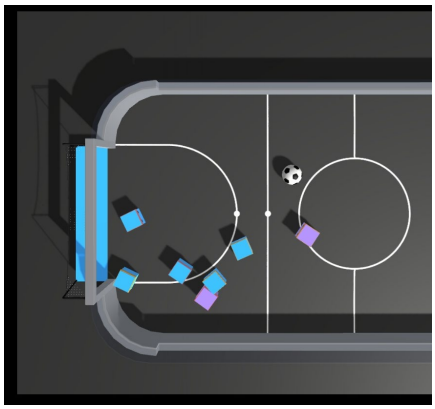


Fig 4.3.1.3: The merging image of the first two view

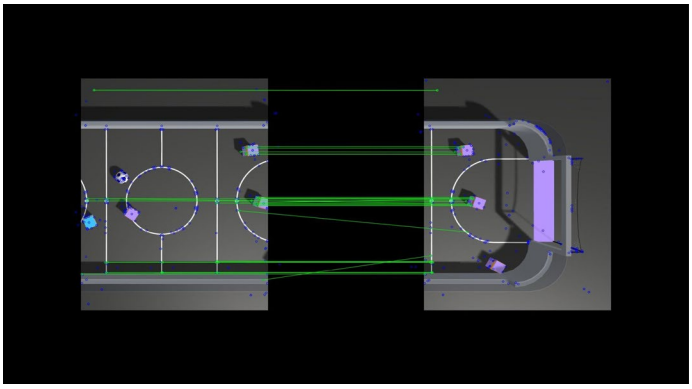


Fig 4.3.1.4: The matching of the last two view images

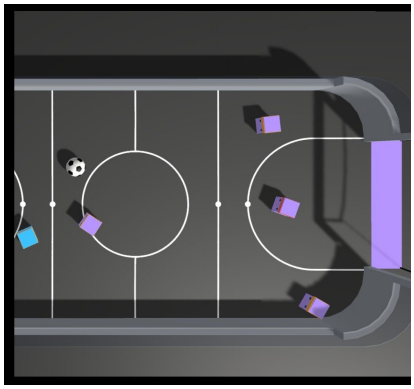


Fig 4.3.1.5: The merging image of the first two view

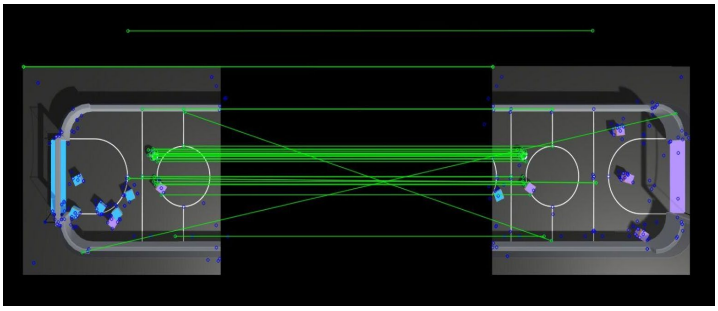


Fig 4.3.1.6: The final matching of the two merge images construction

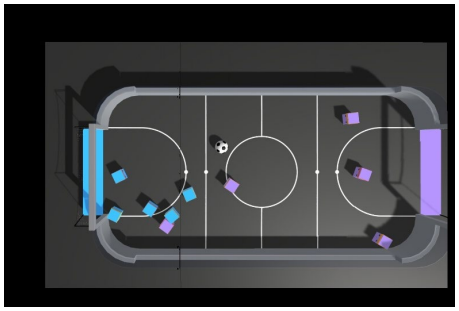


Fig 4.3.1.7: The final merging image - map

Based on the global map, we can get the position of each agent. From Fig 4.3.1.2 to Fig 4.3.1.7, We can see that the model can extract features of the local field of view very well, match the feature points, and finally iterate and register different fields of view step by step, and finally reconstruct the global map.

4.3.2 Pose estimation

We divided the athlete's azimuth angle into 12 classes, and each interval of 30 degrees is a posture. The posture estimation uses a deep learning model PP-LCNET for classification and recognition. As shown in the figure below, the model can recognize the posture very well.

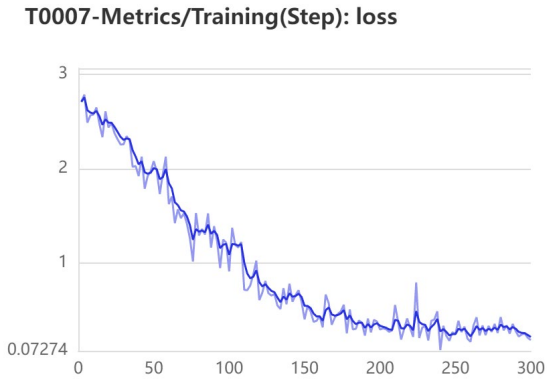


Fig 4.3.2.1: The loss of training process using PP-LCNET

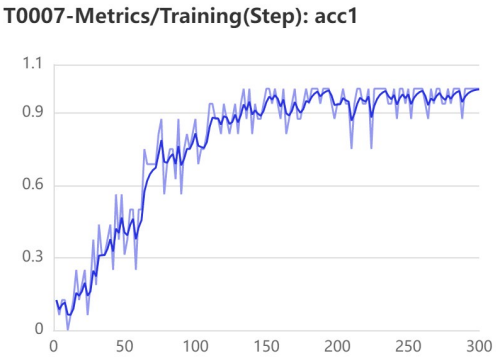


Fig 4.3.2.2: The accuracy of training process using PP-LCNET

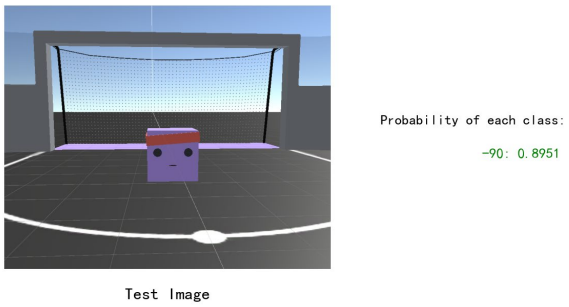


Fig 4.3.2.3: Infer the -90 degree

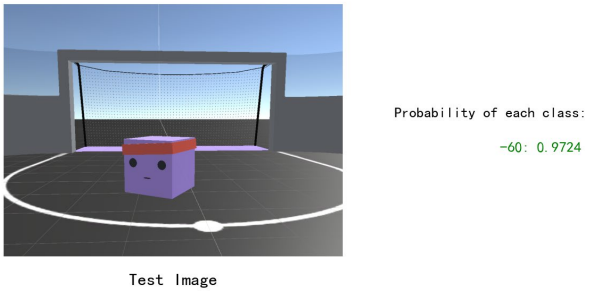


Fig 4.3.2.4: Infer the -60 degree

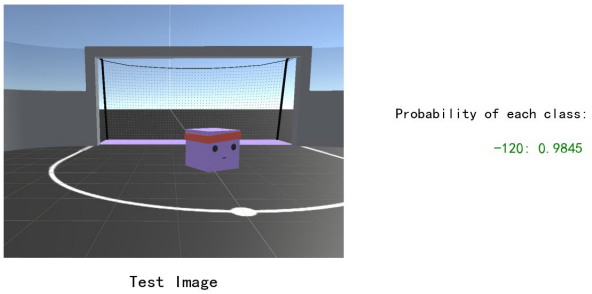


Fig 4.3.2.5: Infer the -120 degree

4.4 Path planning and motion control

We tried to use traditional methods for path planning and motion control. The RRT-Connect algorithm was mainly used for path planning, and the PID algorithm was mainly used for control.

4.4.1 RRT-Connect path planning

The figures below show the result of path planning using the RRT-Connect algorithm.We selected several players to generate the route to the football, and the results show that the generated route can reach the position of the football well and avoid other players.

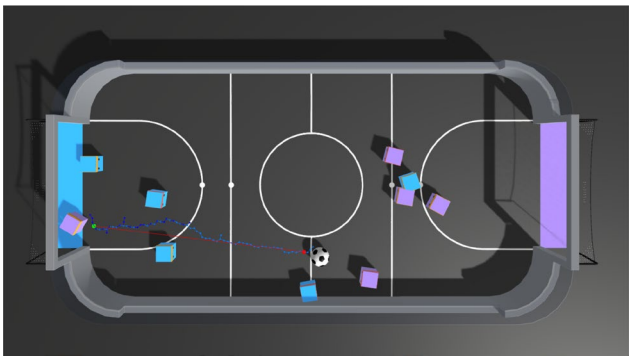


Fig 4.4.1.1: The path 1 generated by RRT-Connect

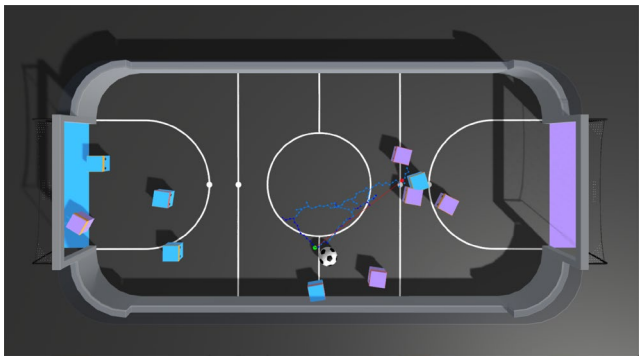


Fig 4.4.1.2: The path 2 generated by RRT-Connect

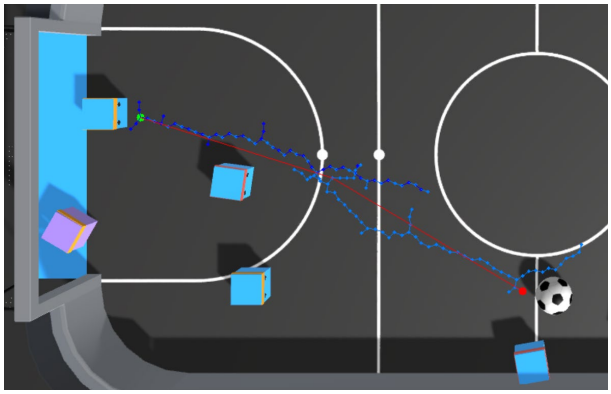


Fig 4.4.1.3: The path 3 generated by RRT-Connect

4.4.2 Motion controller

Since we adopted the method of reinforcement learning to realize End-to-End football training in the later period, we gave up the traditional method of path planning combined with the controller, but we still tried the simulation of the PID model. As shown below, when the kp=10, there are some overshoot in the step response, the performance is relatively good when kp=2.

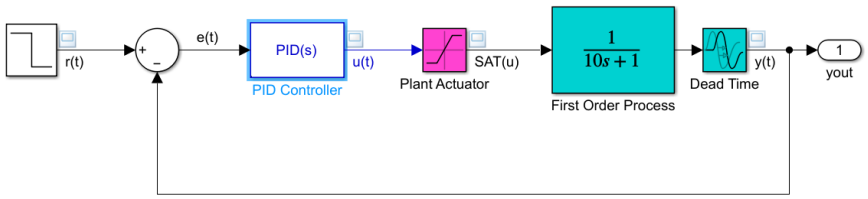


Fig 4.4.2.1: Schematic diagram of PID simulation

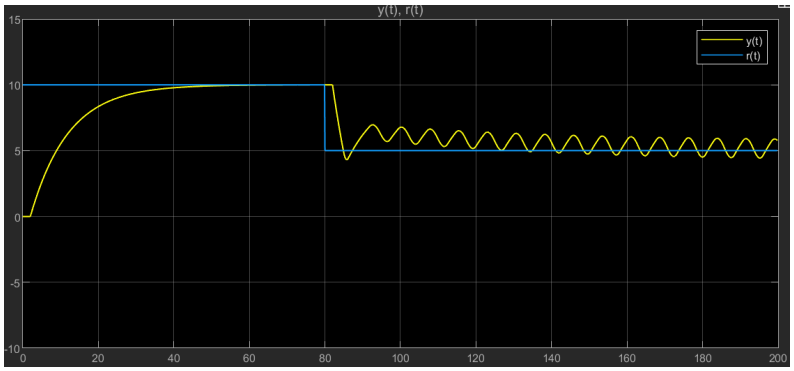


Fig 4.4.2.2: Unit Step Response with Kp=10

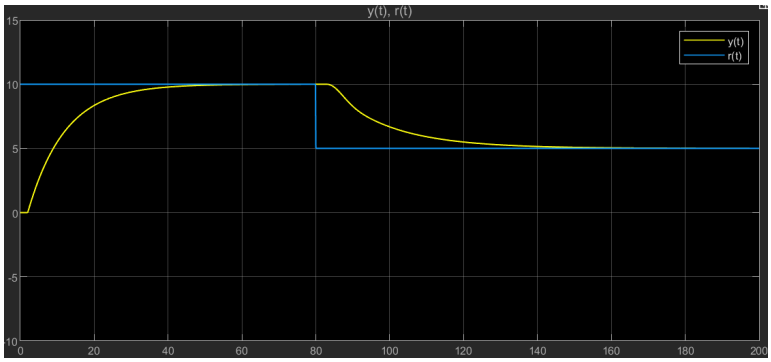


Fig 4.4.2.3: Unit Step Response with Kp=2

4.5 Reinforcement Learning Results

According to section *"3.4.2 Path planning and Motion control methods"*, the path planning was conducted through reinforcement learning methods, we trained and obtained statistical results for each approach during the training process. These statistical results are generated in the Tensor

Board Statistics as a form of linear diagrams. Despite there are so many factors, we conclude that there are only 2 key parameters that are important:

- Self-Play / ELO
  Self-play and ELO score has already been discussed in Section *"2.4.3 ELO"*. It basically shows how well the model is trained and performed, and it is the most important factor we are considering while comparing the performance of the models.
- Policy Statistics / Entropy
  It represents how random the decisions of the model are. For a successful training process, it should slowly decrease.

Statistical Results for each design approach is shown in the below sequence:

1. **2v2 Soccer playing 1: with algorithm MADDPG and Basic Reward Function**

As we mentioned above, this approach failed and was discarded, thus no statistical results

2. **2v2 Soccer playing 2: with algorithm POCA and Basic Reward Function**

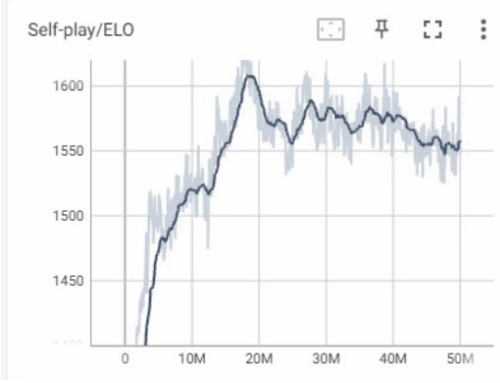


Fig 4.5.1: Self-play ELO score of the 2v2 Soccer playing 2

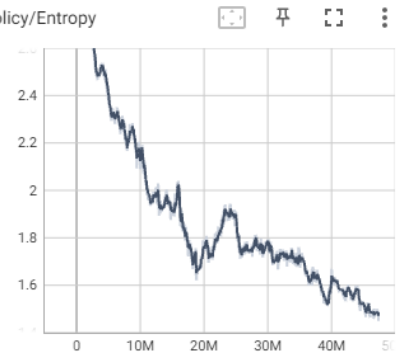


Fig 4.5.2: Policy Entropy score of the 2v2 Soccer playing 2

3. **2v2 Soccer playing 3: with algorithm POCA and Enhanced Reward Function**

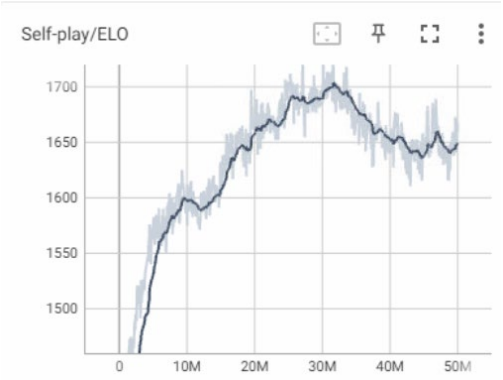


Fig 4.5.3: Self-play ELO score of the 2v2 Soccer playing 3

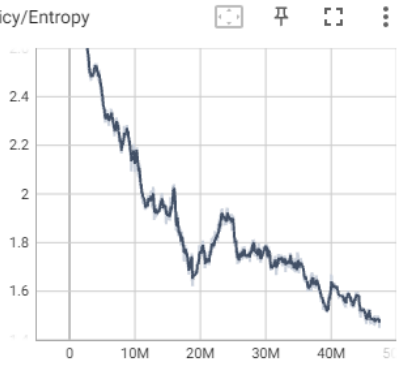


Fig 4.5.4: Policy Entropy score of the 2v2 Soccer playing 3

For the 2v2 Soccer playing 2, the ELO score reaches its maximum of 1634 in 20M steps, then it drops slowly and stabilizes around 1550; however for 2v2 Soccer playing 3, the ELO score reaches 1720 in 30M steps and then fell and stabilized around 1650. By comparing the ELO scores of these

2 approaches, we can see that the performance of the Enhanced Reward Function is strictly better than the one with Basic Reward Function.

As for the Entropy score, both approaches have their entropy dropped from 2.5 to 1.6 in 40M steps. As we mentioned before, for a successful training process, the entropy score should decrease slowly. However if the entropy graph drops quickly, it generally means that the agent is becoming increasingly certain about which actions to take and is focusing more on exploiting known good actions rather than exploring new ones. This can be a sign that the agent has learned a good policy and is able to consistently take actions that lead to positive outcomes.

4. 5v5 Soccer playing 1: with algorithm POCA and Enhanced Reward Function

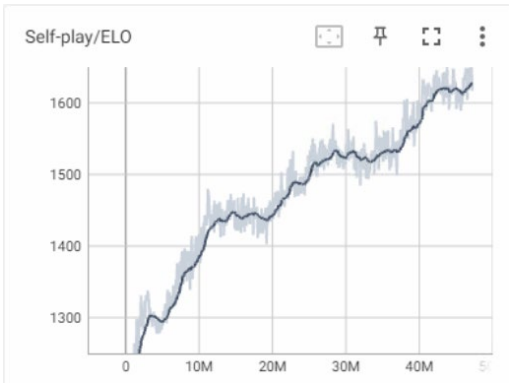


Fig 4.5.5: ELO score of 5v5 Soccer playing 1, Goalie

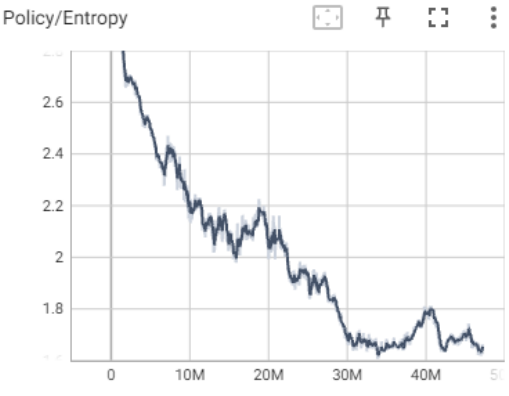


Fig 4.5.6: Entropy score of 5v5 Soccer playing 1, Goalie

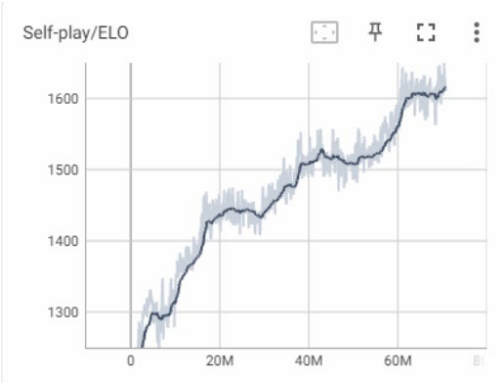


Fig 4.5.7: ELO score of 5v5 Soccer playing 1, Striker

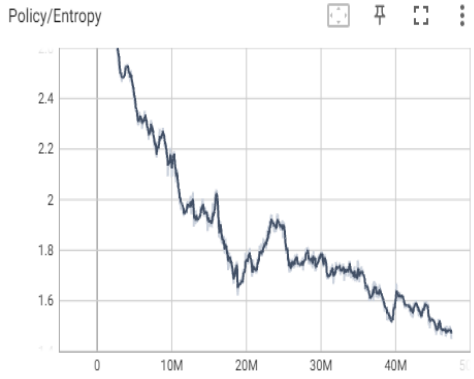


Fig 4.5.8: Entropy score of 5v5 Soccer playing 1, Striker

5. 5v5 Soccer playing 2: with algorithm PPO and Enhanced Reward Function

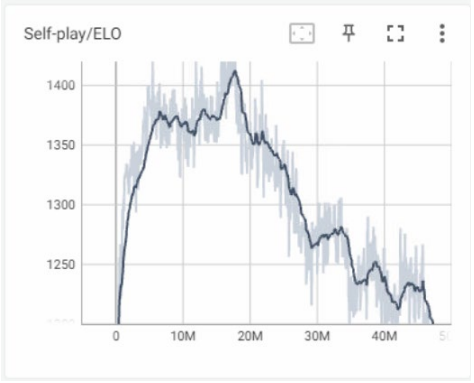


Fig 4.5.9: ELO score of 5v5 Soccer playing 2, Goalie

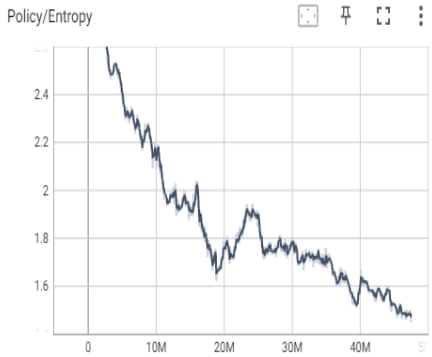


Fig 4.5.10: Entropy score of 5v5 Soccer playing 2, Goalie

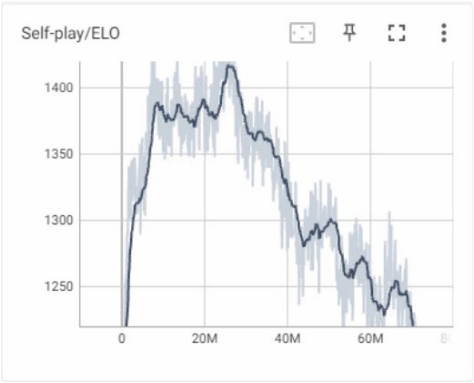


Fig 4.5.11: ELO score of 5v5 Soccer playing 2, Striker

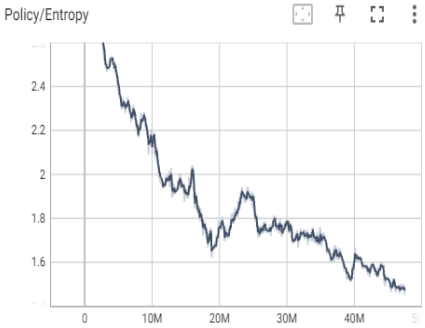


Fig 4.5.12: Entropy score of 5v5 Soccer playing 2, Striker

Generally with the same approach, the Goalie and Striker share a similar trend in diagrams of ELO score and Entropy scores. Which means they are trained with similar performances.

For the ELO score, the approach 1 with the POCA algorithm performs well, with the highest ELO score around 1600. We can also see that this is not the climax yet, which means this approach still has the potential to increase the ELO score and perform better if we increase the training step size. However, for the second approach with the PPO algorithm, the ELO scores reach a maximum of 1450 around 20M steps and then start to fall. Which means that this approach had their best performance when they are trained at 20M steps, and all the training after that is worthless and had no significant effects at all.

As for the Entropy score, both approaches have their entropy score dropped significantly and quickly, even quicker than the 2v2 approaches. They went from 2.6 all the way down to 1.6 in 40M steps. Although we mentioned before that dropping is a good sign that the agents are learning. However, for an entropy score that dropped too much, this can also be an indication that the agent is getting stuck in a suboptimal policy and is not exploring enough to find better actions. In this case, it may be necessary to adjust the learning algorithm or the environment to encourage the agent to continue exploring and learning.

The designed training steps for these approaches are 50,000,000 steps and it took us roughly 100 hours to complete the training for each 2v2 models and 150 hours for each 5v5 models. For 5v5 soccer playing, we find out that the striker model trained faster than the goalie model.

The figure below is a prompt output during our training process for 5v5 Soccer playing with PPO, we can see that the striker model finished training roughly 1.2 times earlier than the goalie model, after it reaches 50,000,000 steps, it is not trained anymore.

```
[INFO] Goalie. Step: 38720000. Time Elapsed: 470597.801 s. Mean Reward: -0.015. Std of Reward: 0.309. Training. ELO: 1254.063.
[INFO] Striker. Step: 58080000. Time Elapsed: 470597.834 s. Mean Reward: -0.004. Std of Reward: 0.283. Not Training. ELO: 1278.649.
[INFO] Striker. Step: 58090000. Time Elapsed: 470672.888 s. Mean Reward: -0.004. Std of Reward: 0.276. Not Training. ELO: 1259.143.
[INFO] Goalie. Step: 38730000. Time Elapsed: 470708.681 s. Mean Reward: -0.031. Std of Reward: 0.347. Training. ELO: 1239.048.
```

Fig 4.5.13: A sample output during the training process, indicating that Striker finished training first

4.6 Competition between Reinforcement Learning Methods

In order to figure out which reinforcement learning method has the most significant effect on this soccer playing environment, we decided to perform competitions between different algorithms and reward functions for 2v2 as well as 5v5 game plays. Each team represents a method and are set to perform a 30 min match under the same environment and conditions. The sequence and results of the game plays are shown below:

1. **Competition 1: 2v2 Soccer Playing**

In this competition, the environment is set to be a 2v2 match with both strikers at each side. Team 1 with the blue agents is using the POCA algorithm with the basic reward function, while team 2 with purple agents is using the POCA algorithm as well but with the enhanced reward function. Therefore this is a competition between reward functions for 2v2 game play. The following figure is a snapshot of the match indicating the results:

Fig 4.6.1: Snapshot of Competition 1 results

We can see that the final result is 121:157, and team 2 wins. Which means for 2v2 game play with the POCA algorithm, the enhanced reward function is better than the basic reward function. Which also matched the best ELO score comparison. (1634 vs 1720)

2. Competition 2: 5v5 Soccer Playing

In this competition, the environment is set to be a 5v5 match with 2 goalies and 3 strikers for both sides. Team 1 with the blue agents is using the PPO algorithm with the enhanced reward function, while team 2 with the purple agents is using the POCA algorithm with also the enhanced reward function. Therefore this is a competition between POCA and PPO algorithms. The snapshot of the competition result is shown in the figure below:

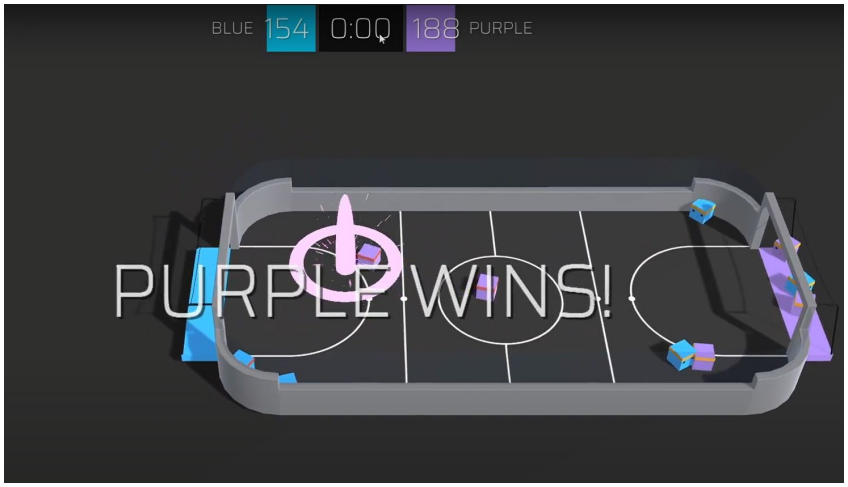


Fig 4.6.2: Snapshot of Competition 2 results

We can see that the final result is 154:188, and team 2 wins as well. Which means for 5v5 game play with the enhanced reward function, the POCA algorithm performs better than the PPO algorithm, which also matches the best ELO score comparison. (1600 vs 1450)

3. Competition 3: 5v5 Soccer Playing

The third competition is a bit different. The environment is set to be a 5v5 match. However, each side's training model is selected from the winners of the previous 2 competitions. For team 1 with the blue agents, the models are selected from the best 2v2 model from competition 1: the POCA algorithm with the enhanced reward function (Striker Only). As for team 2, we are using the best 5v5 model from competition 2: the POCA algorithm with the enhanced reward function. (3 Strikers and 2 Goalies)

As it is strictly unfair to perform a 2v5 game match, therefore, team 1 is filled with 5 strikers, and team 2 is filled with 3 strikers and 2 goaliers. Therefore in this competition, we are competing between the training methods, whether it is better to have the players all trained as strikers or with different roles. Final competition result is shown below:

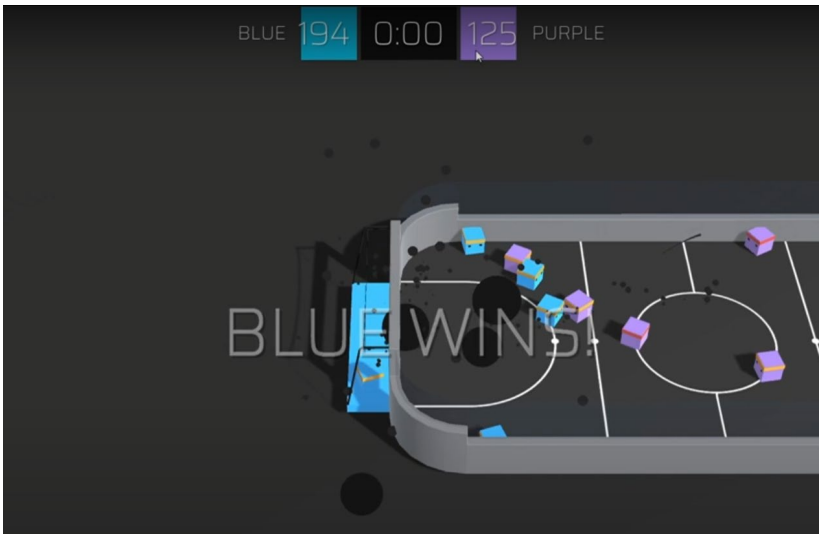


Fig 4.6.3: Snapshot of Competition 3 results

We can see that the final result is 194:125, team 1 with blue agents wins. Which means agents trained from the 2v2 all strikers model performs better than the 5v5 model that is trained with 3 strikers and 2 goalies. However, this doesn't strictly mean that the models are trained better when they only have the striker roles. If we go back to section 4.4 and check the training simulation results of the ELO score. We will notice that for the best 2v2 model (which is in 2v2 soccer playing 3), the ELO score reaches a maximum of 1720 around 30M steps, and then falls to stabilize around 1650. Which means the model can only perform with a maximum of 1720 ELO score. However for the best 5v5 model (which is in 5v5 soccer playing 1), both goalies and strikers have the ELO score reaching 1650 around 70M steps, however the diagram shows that the ELO score is still in a ascending pattern and would still increase if we increase the training step size furthermore. We believe that although for now, the best 2v2 model out performs the best 5v5 models, the 5v5 models with the roles splitted will perform better with longer training time and training step size.

5. Conclusion

In summary, we have simulated and designed two soccer playing environments, a 2v2 soccer playing as well as a 5v5 soccer playing under the Unity Environment. We tried to use object detection PP-YOLOV2, pose estimation PP-LCNET as our sensing method to got the observation of agents, soccer and goal. And we also tried to use the SIFT and RANSAC method to reconstruct the global map, and use the RRT-Connect to generate the path of each agent. However, due to the limited computer hardware resources, it will cost us maybe serval months if all deep learning method was applied on our task. After these algorithms was verified, we use the ray sensor to replace the function of these methods to accelerate our training time. The concept of reinforcement learning was used and we have trained several player models with different algorithms and have them compete for the final model with the best outcome. During the design process, we decided to set the hyperparameters of the training models on reinforcement learning algorithms and reward functions.

At the end, we successfully trained out 4 sets of models and compared their performance by performing 3 competitions between them. After the competition was done and analysis were made, we concluded the following for this soccer playing project:

1. With the two reward functions we used in this project, the enhanced reward function with both personal and group rewards performs better than the basic reward function with only group reward, having a more detailed reward function would improve the performance.

2. For this soccer playing project, the POCA algorithm performs best, followed by the PPO algorithm and the MADDPG algorithm performs worst and fails.

3. During the final competition, the team with models trained from 2v2 models with only strikers wins the team with models trained from 5v5 models with 3 strikers and 2 goalies. However, we didn't conclude that the 2v2 environment trains better models than the 5v5 environment with more roles. By comparing the ELO score plots and predictions, we believe that as the training time and training step increases, the performance of the 5v5 soccer playing would eventually be better than the 2v2 model. We conclude that having more roles would enhance the performance of the training only if we have enough training time, the time complexity and computational cost would be higher for a training with more roles, but the results would be better

6. Reference

[1] A. Juliani *et al.*, "Unity: A General Platform for Intelligent Agents," pp. 1–28, 2018, [Online]. Available: http://arxiv.org/abs/1809.02627

[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *3rd Int. Conf. Learn. Represent. ICLR 2015 - Conf. Track Proc.*, pp. 1–14, 2015.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2016-December, pp. 770–778, 2016, doi: 10.1109/CVPR.2016.90.

[4] M. Steen, S. Downe, N. Bamford, and L. Edozien, "DenseNet:Densely Connected Convolutional Networks arXiv:1608.06993v5," *Arxiv*, vol. 28, no. 4, pp. 362–371, 2018.

[5] R. Girshick, "Fast R-CNN," *Proc. IEEE Int. Conf. Comput. Vis.*, vol. 2015 International Conference on Computer Vision, ICCV 2015, pp. 1440–1448, 2015, doi: 10.1109/ICCV.2015.169.

[6] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 6, pp. 1137–1149, 2017, doi: 10.1109/TPAMI.2016.2577031.

[7] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 2, pp. 386–397, 2020, doi: 10.1109/TPAMI.2018.2844175.

[8] W. Liu *et al.*, "SSD: Single shot multibox detector," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 9905 LNCS, pp. 21–37, 2016, doi: 10.1007/978-3-319-46448-0_2.

[9] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2016-December, pp. 779–788, 2016, doi: 10.1109/CVPR.2016.91.

[10] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," 2018, [Online]. Available: http://arxiv.org/abs/1804.02767

[11] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors," pp. 1–15, 2022, [Online]. Available: http://arxiv.org/abs/2207.02696

[12] X. Long *et al.*, "PP-YOLO: An Effective and Efficient Implementation of Object Detector," 2020, [Online]. Available: http://arxiv.org/abs/2007.12099

[13] S. Huang and G. Dissanayake, "Convergence and consistency analysis for extended Kalman filter based SLAM," *IEEE Trans. Robot.*, vol. 23, no. 5, pp. 1036–1049, 2007, doi: 10.1109/TRO.2007.903811.

[14] G. Grisetti, R. Kümmerle, C. Stachniss, and I. Introduction, "Giorgio Grisetti, Rainer Kümmerle, Cyrill Stachniss, and Wolfram Burgard," *Aerospace*, pp. 31–43, 2010.

[15] T. Taketomi, H. Uchiyama, and S. Ikeda, "Visual SLAM algorithms: A survey from 2010 to 2016," *IPSJ Trans. Comput. Vis. Appl.*, vol. 9, 2017, doi: 10.1186/s41074-017-0027-2.

[16] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-time loop closure in 2D LIDAR SLAM," *Proc. - IEEE Int. Conf. Robot. Autom.*, vol. 2016-June, pp. 1271–1278, 2016, doi: 10.1109/ICRA.2016.7487258.

[17] X. X. Lu, "A Review of Solutions for Perspective-n-Point Problem in Camera Pose Estimation," *J. Phys. Conf. Ser.*, vol. 1087, no. 5, 2018, doi: 10.1088/1742-6596/1087/5/052009.

[18] A. Lucieer, S. M. d. Jong, and D. Turner, “Mapping landslide displacements using Structure from Motion (SfM) and image correlation of multi-temporal UAV photography,” *Prog. Phys. Geogr.*, vol. 38, no. 1, pp. 97–116, 2014, doi: 10.1177/0309133313515293.

[19] M. O. A. Aqel, M. H. Marhaban, M. I. Saripan, and N. B. Ismail, “Review of visual odometry: types, approaches, challenges, and applications,” *Springerplus*, vol. 5, no. 1, 2016, doi: 10.1186/s40064-016-3573-7.

[20] Z. Huai and G. Huang, “Robocentric visual–inertial odometry,” *Int. J. Rob. Res.*, vol. 41, no. 7, pp. 667–689, 2022, doi: 10.1177/0278364919853361.

[21] S. K. Debnath *et al.*, “A review on graph search algorithms for optimal energy efficient path planning for an unmanned air vehicle,” *Indones. J. Electr. Eng. Comput. Sci.*, vol. 15, no. 2, pp. 743–750, 2019, doi: 10.11591/ijeecs.v15.i2.pp743-749.

[22] Y. K. Hwang and N. Ahuja, “A Potential Field Approach to Path Planning,” *IEEE Trans. Robot. Autom.*, vol. 8, no. 1, pp. 23–32, 1992, doi: 10.1109/70.127236.

[23] S. Sedighi, D. Van Nguyen, and K. D. Kuhnert, “Guided Hybrid A-star Path Planning Algorithm for Valet Parking Applications,” *2019 5th Int. Conf. Control. Autom. Robot. ICCAR 2019*, pp. 570–575, 2019, doi: 10.1109/ICCAR.2019.8813752.

[24] S. Karaman and E. Frazzoli, “Sampling-based algorithms for optimal motion planning,” *Int. J. Rob. Res.*, vol. 30, no. 7, pp. 846–894, 2011, doi: 10.1177/0278364911406761.

[25] C. Boutilier, T. Dean, and S. Hanks, “Decision-Theoretic Planning: Structural Assumptions and Computational Leverage,” *J. Artif. Intell. Res.*, vol. 11, pp. 1–94, 1999, doi: 10.1613/jair.575.

[26] J. F. Fisac, E. Bronstein, E. Stefansson, D. Sadigh, S. S. Sastry, and A. D. Dragan, “Hierarchical game-theoretic planning for autonomous vehicles,” *Proc. - IEEE Int. Conf. Robot. Autom.*, vol. 2019-May, pp. 9590–9596, 2019, doi: 10.1109/ICRA.2019.8794007.

[27] A. Slowik and H. Kwasnicka, “Evolutionary algorithms and their applications to engineering problems,” *Neural Comput. Appl.*, vol. 32, no. 16, pp. 12363–12379, 2020, doi: 10.1007/s00521-020-04832-8.

[28] S. Bennett, “Development of the PID Controller,” *IEEE Control Syst.*, vol. 13, no. 6, pp. 58–62, 1993, doi: 10.1109/37.248006.

[29] J. H. Lee, “Model predictive control: Review of the three decades of development,” *Int. J. Control. Autom. Syst.*, vol. 9, no. 3, pp. 415–424, 2011, doi: 10.1007/s12555-011-0300-6.

[30] R. Mohammadi Asl, A. Mahdoudi, E. Pourabdollah, and G. Klančar, “Combined PID and LQR controller using optimized fuzzy rules,” *Soft Comput.*, vol. 23, no. 13, pp. 5143–5155, 2019, doi: 10.1007/s00500-018-3180-3.

[31] M. Hassler, I. A. Shaltiel, and C. H. Haering, “Towards a Unified Model of SMC Complex Function,” *Curr. Biol.*, vol. 28, no. 21, pp. R1266–R1281, 2018, doi: 10.1016/j.cub.2018.08.034.

[32] A. Shekhar and A. Sharma, “Review of Model Reference Adaptive Control,” *2018 Int. Conf. Information, Commun. Eng. Technol. ICICET 2018*, pp. 1–5, 2018, doi: 10.1109/ICICET.2018.8533713.

[33] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, “Multi-agent actor-critic for mixed cooperative-competitive environments,” *Adv. Neural Inf. Process. Syst.*, vol. 2017-Decem, pp. 6380–6391, 2017.

[34] A. Cohen *et al.*, “On the Use and Misuse of Absorbing States in Multi-agent Reinforcement Learning,” 2021, [Online]. Available: http://arxiv.org/abs/2111.05992

[35] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal Policy Optimization Algorithms,” pp. 1–12, 2017, [Online]. Available: http://arxiv.org/abs/1707.06347

[36] F. Otto, *Model-Free Deep Reinforcement Learning—Algorithms and Applications*, vol. 883. 2021. doi: 10.1007/978-3-030-41188-6_10.

[37] B. Düring, M. Fischer, and M. T. Wolfram, “An Elo-type rating model for players and teams of variable strength,” *Philos. Trans. R. Soc. A Math. Phys. Eng. Sci.*, vol. 380, no. 2224, pp. 1–19, 2022, doi: 10.1098/rsta.2021.0155.

[38] พวงผกา มะเสนา และประณต นันทิยะกุล, “No Titleการบริหารจัดการการบริการที่มีคุณภาพในโรงพยาบาลสังกัดกระทรวงสาธารณสุข,” *วารสารวิชาการมหาวิทยาลัยอีสเทิร์นเอเชีย*, vol. 4, no. 1, pp. 88–100, 2557.

[39] X. Huang *et al.*, “PP-YOLOv2: A Practical Object Detector,” pp. 1–7, 2021, [Online]. Available: http://arxiv.org/abs/2104.10419

[40] M. A. Fischler and R. C. Bolles, “Random sample consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography,” *Commun. ACM*, vol. 24, no. 6, pp. 381–395, 1981, doi: 10.1145/358669.358692.

[41] C. Cui *et al.*, “PP-LCNet: A Lightweight CPU Convolutional Neural Network,” 2021, [Online]. Available: http://arxiv.org/abs/2109.15099

[42] J. J. Kuffner and S. M. La Valle, “RRT-connect: an efficient approach to single-query path planning,” *Proc. - IEEE Int. Conf. Robot. Autom.*, vol. 2, no. Icra, pp. 995–1001, 2000, doi: 10.1109/robot.2000.844730.

7. Appendix

7.1 Part of the code of Image processing

```
import paddlex as pdx
from paddlex import transforms as T

train_transforms = T.Compose([
    T.RandomResizeByShort(
        short_sizes=[640, 672, 704, 736, 768, 800],
        max_size=1333,
        interp='CUBIC'), T.RandomHorizontalFlip(), T.Normalize(
            mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

eval_transforms = T.Compose([
    T.ResizeByShort(
        short_size=800, max_size=1333, interp='CUBIC'), T.Normalize(
            mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])


train_dataset = pdx.datasets.VOCDetection(
    data_dir='data/dataset',
    file_list='data/dataset/train_list.txt',
    label_list='data/dataset/labels.txt',
    transforms=train_transforms,
    shuffle=True)

eval_dataset = pdx.datasets.VOCDetection(
    data_dir='data/dataset',
    file_list='data/dataset/val_list.txt',
    label_list='data/dataset/labels.txt',
    transforms=eval_transforms,
    shuffle=False)

num_classes = len(train_dataset.labels)
model = pdx.det.FasterRCNN(
    num_classes=num_classes, backbone='ResNet50', with_fpn=True)


model.train(
    num_epochs=12,
    train_dataset=train_dataset,
    train_batch_size=2,
    eval_dataset=eval_dataset,
    learning_rate=0.0025,
    lr_decay_epochs=[8, 11],
    warmup_steps=5,
    warmup_start_lr=0.00025,
    save_dir='output/faster_rcnn_r50_fpn',
    use_vdl=True)
```

7.2 Part of the code of Location (Global Map)

```python
def match_feather_point(img1, img2):

    sift = cv.SIFT_create()
    kp1, des1 = sift.detectAndCompute(img1, None)
    kp2, des2 = sift.detectAndCompute(img2, None)

    FLANN_INDEX_KDTREE = 1
    index_params = dict(algorithm=FLANN_INDEX_KDTREE, trees=5)
    search_params = dict(checks=50)
    flann = cv.FlannBasedMatcher(index_params, search_params)
    matches = flann.knnMatch(des1, des2, k=2)

    return kp1, kp2, matches


def get_good_match(img1, img2, kp1, kp2, matches):

    # TODO 使用RANSAC方法去除错误匹配点对
    matches_mask = [[0, 0] for _ in range(len(matches))]
    good_match = []
    for i, (m, n) in enumerate(matches):
        if m.distance < 0.5 * n.distance:
            good_match.append(m)
            matches_mask[i] = [1, 0]
      draw_params = dict(matchColor=(0, 255, 0),
                         singlePointColor=(255, 0, 0),
                         matchesMask=matches_mask,
                         flags=0)
    img3 = cv.drawMatchesKnn(img1, kp1, img2, kp2, matches, None, **draw_params)

    MIN_MATCH_COUNT = 10
    if len(good_match) < MIN_MATCH_COUNT:
        print("Not enough matches are found - {}/{}".format(len(good_match), MIN_MATCH_COUNT))
        return None

    src_pts = np.float32([kp1[m.queryIdx].pt for m in good_match]).reshape(-1, 1, 2)
    dst_pts = np.float32([kp2[m.trainIdx].pt for m in good_match]).reshape(-1, 1, 2)

    M, mask = cv.findHomography(src_pts, dst_pts, cv.RANSAC, 5.0)
    img2 = cv.warpPerspective(img2, np.array(M), (img2.shape[1], img2.shape[0]),
                              flags=cv.WARP_INVERSE_MAP)

    dst = img1.copy()
    for i in range(img2.shape[0]):
        for j in range(img2.shape[1]):
            pix = img2[i, j]
            if pix.any():
                dst[i, j] = pix
```

7.3 Part of the code of Pose estimation

```python
from paddle.nn import Conv2D, Identity

from ..legendary_models.pp_lcnet_v2 import MODEL_URLS, PPLCNetV2_base, RepDepthwiseSeparable, _load_pretrained

__all__ = ["PPLCNetV2_base_ShiTu"]


def PPLCNetV2_base_ShiTu(pretrained=False, use_ssld=False, **kwargs):
    """
    An variant network of PPLCNetV2_base
    1. remove ReLU layer after last_conv
    2. add bias to last_conv
    3. change stride to 1 in last two RepDepthwiseSeparable Block
    """
    model = PPLCNetV2_base(pretrained=False, use_ssld=use_ssld, **kwargs)

    def remove_ReLU_function(conv, pattern):
        new_conv = Identity()
        return new_conv

    def add_bias_last_conv(conv, pattern):
        new_conv = Conv2D(
            in_channels=conv._in_channels,
            out_channels=conv._out_channels,
            kernel_size=conv._kernel_size,
            stride=conv._stride,
            padding=conv._padding,
            groups=conv._groups,
            bias_attr=True)
        return new_conv

    def last_stride_function(rep_block, pattern):
        new_conv = RepDepthwiseSeparable(
            in_channels=rep_block.in_channels,
            out_channels=rep_block.out_channels,
            stride=1,
            dw_size=rep_block.dw_size,
            split_pw=rep_block.split_pw,
            use_rep=rep_block.use_rep,
            use_se=rep_block.use_se,
            use_shortcut=rep_block.use_shortcut)
```

7.4 Part of the code of Path Planning (RRT-Connect)

```
# - Vertex class:
class Vertex:
  def __init__(self,pos,parent):
    self.pos = pos
    self.parent = parent

# - RRT class :
class RRT :

    def __init__(self,img_map):
        print("Initializing RRT")
        self._pgm_format = ''
        self._initPos = (10,10)
        self._targetPos = (90,80)
        self._map = img_map.copy()
        self._map[self._map==255] = 1
        self._height = img_map.shape[0]
        self._width = img_map.shape[1]
        self._path=[]
        self.click=0

    # - Load the current position of the robot as the initial one and the target position from topics ROS
    def loadPos(self): #TODO
    # charge pos of the robot int the map from image_processing
    # charge final pos of the robot from rviz
        return


    def start(self,img_map=None):
        """
        RRT based algorithm
        """

        goal = False
        self._path=[]
        newvertex = Vertex(self._initPos,None)
        vertices = [newvertex]
        lin_dist = dist(self._initPos,self._targetPos)

        # main loop
        while not goal:
            # create random point
            newpoint = (rand(self._width),rand(self._height))
            nearest_dist = float('inf')
            # look for the nearest point in the tree
            for v in vertices:
                currdist = dist(newpoint,v.pos)
                if currdist < nearest_dist:
                    nearest = v
```

7.5 Part of the code of Reinforcement learning

MADDPG

```python
def main():

    env = TransUnity2Gym(origin_env)

    agents = []
    agents_sum = 0
    behavior_agents_num = 0

    multi_action_space = [[3,3,3],[3,3,3],[3,3,3],[3,3,3],[3,3,3],[3,3,3],[3,3,3],[3,3,3],[3,3,3],[3,3,3]]
    multi_obs_shape_n = [336, 336, 336, 336, 336, 336, 336, 336, 336, 336]
    multi_act_shape_n = [27,27,27,27,27,27,27,27,27,27]
    critic_in_dim = sum(multi_obs_shape_n) + sum(multi_act_shape_n)

    # build agents

    for i in range(env.n):
        model = MAModel(multi_obs_shape_n[i], multi_act_shape_n[i], critic_in_dim, False)
        algorithm = MADDPG(
            model,
            agent_index=i,
            act_space=multi_action_space,
            gamma=GAMMA,
            tau=TAU,
            critic_lr=CRITIC_LR,
            actor_lr=ACTOR_LR)
        agent = MAAgent(
            algorithm,
            agent_index=i,
            obs_dim_n=multi_obs_shape_n,
            act_dim_n=multi_act_shape_n,
            batch_size=BATCH_SIZE)
        agents.append(agent)




    if False:#args.restore:
        # restore modle
        for i in range(len(agents)):
            model_file = args.model_dir + '/agent_' + str(i)
            if not os.path.exists(model_file):
                raise Exception(
                    'model file {} does not exits'.format(model_file))
            agents[i].restore(model_file)
```

PPO

```yaml
behaviors:
  Goalie:
    trainer_type: ppo
    hyperparameters:
      batch_size: 2048
      buffer_size: 20480
      learning_rate: 0.0003
      beta: 0.005
      epsilon: 0.2
      lambd: 0.95
      num_epoch: 3
      learning_rate_schedule: constant
    network_settings:
      normalize: false
      hidden_units: 512
      num_layers: 2
      vis_encode_type: simple
    reward_signals:
      extrinsic:
        gamma: 0.99
        strength: 1.0
    keep_checkpoints: 5
    max_steps: 60000000
    time_horizon: 1000
    summary_freq: 10000
    self_play:
      save_steps: 50000
      team_change: 200000
      swap_steps: 2000
      window: 10
      play_against_latest_model_ratio: 0.5
      initial_elo: 1200.0
  Striker:
    trainer_type: ppo
    hyperparameters:
      batch_size: 2048
      buffer_size: 20480
      learning_rate: 0.0003
      beta: 0.005
      epsilon: 0.2
      lambd: 0.95
      num_epoch: 3
      learning_rate_schedule: constant
    network_settings:
      normalize: false
      hidden_units: 512
      num_layers: 2
      vis_encode_type: simple
```

POCA

```yaml
behaviors:
  Goalie:
    trainer_type: poca
    hyperparameters:
      batch_size: 2048
      buffer_size: 20480
      learning_rate: 0.0003
      beta: 0.005
      epsilon: 0.2
      lambd: 0.95
      num_epoch: 3
      learning_rate_schedule: constant
    network_settings:
      normalize: false
      hidden_units: 512
      num_layers: 2
      vis_encode_type: simple
    reward_signals:
      extrinsic:
        gamma: 0.99
        strength: 1.0
    keep_checkpoints: 5
    max_steps: 50000000
    time_horizon: 1000
    summary_freq: 10000
    self_play:
      save_steps: 50000
      team_change: 200000
      swap_steps: 2000
      window: 10
      play_against_latest_model_ratio: 0.5
      initial_elo: 1200.0
  Striker:
    trainer_type: poca
    hyperparameters:
      batch_size: 2048
      buffer_size: 20480
      learning_rate: 0.0003
      beta: 0.005
```

7.6 Part of the code of Reward Functions

```
   void GiveReward()
    {
        float halfRewardDistance = 16;
        float goalieRewardDstance = 7;

        switch (position)
        {
            case Position.Striker:
                if (team == Team.Blue && Vector3.Distance(GoalPurple.transform.position, ball.transform.position) <
halfRewardDistance)
                {
                    AddReward(0.002f);
                }
                else if (team == Team.Purple && Vector3.Distance(GoalBlue.transform.position, ball.transform.position)
< halfRewardDistance)
                {
                    AddReward(0.002f);
                }
                else if (team == Team.Blue && Vector3.Distance(GoalBlue.transform.position, ball.transform.position) <
halfRewardDistance)
                {
                    AddReward(-0.002f);
                }
                else if (team == Team.Purple && Vector3.Distance(GoalPurple.transform.position,
ball.transform.position) < halfRewardDistance)
                {
                    AddReward(-0.002f);
                }
                break;
            case Position.Goalie:
                if(team == Team.Blue)
                {
                    if (Vector3.Distance(transform.position, GoalBlue.transform.position) >= goalieRewardDstance)
                    {
                        AddReward(-(Vector3.Distance(transform.position, GoalBlue.transform.position) -
goalieRewardDstance) / 1000);
                    }
                    else if (Vector3.Distance(ball.transform.position, GoalBlue.transform.position) <=
goalieRewardDstance * 0.7)
                    {
                        Vector3 blueGoalToBall = ball.transform.position - GoalBlue.transform.position;
                        blueGoalToBall.y = 0;
                        var r = Vector3.Dot(ballRigdbody.velocity, blueGoalToBall) / 1000;
                        AddReward(r);
                    }

                }
                else if (team == Team.Purple)
                {
                    if (Vector3.Distance(transform.position, GoalPurple.transform.position) >= goalieRewardDstance)
                    {
                        AddReward(-(Vector3.Distance(transform.position, GoalPurple.transform.position) -
goalieRewardDstance) / 1000);
                    }
                    else if (Vector3.Distance(ball.transform.position, GoalPurple.transform.position) <=
goalieRewardDstance * 0.7)
                    {
                        Vector3 blueGoalToBall = ball.transform.position - GoalPurple.transform.position;
                        blueGoalToBall.y = 0;
                        var r = Vector3.Dot(ballRigdbody.velocity, blueGoalToBall) / 1000;
                        if (r < 0)
                        {
```

```
                        r = r * 0.7f;
                    }
                    AddReward(r);
                }

            }
            break;
        case Position.Generic:
            break;
    }
}


public override void OnActionReceived(ActionBuffers actionBuffers)

{

    if (position == Position.Goalie)
    {
        // Existential bonus for Goalies.
        AddReward(m_Existential);
    }
    else if (position == Position.Striker)
    {
        // Existential penalty for Strikers
        AddReward(-m_Existential);
    }
    MoveAgent(actionBuffers.DiscreteActions);
    //GiveReward();
}
void OnCollisionEnter(Collision col)
{
    if (col.gameObject.CompareTag(purpleGoalTag)) //ball touched purple goal
    {
        if (previous_agent_touch_ball.team == Team.Blue) //蓝队踢进了紫队球门
        {
            previous_agent_touch_ball.AddReward(1.0f); //踢进去的球员单独再+一分
        }
        if (previous_agent_touch_ball.team == Team.Purple) //紫队踢进了紫队球门，或者没防守住
        {
            previous_agent_touch_ball.AddReward(-1.0f);//踢进自己的球门，该球员单独再-1 分
        }
        envController.GoalTouched(Team.Blue);
    }
    if (col.gameObject.CompareTag(blueGoalTag)) //ball touched blue goal
    {
        if (previous_agent_touch_ball.team == Team.Purple) //紫队踢进了蓝队球门
        {
            previous_agent_touch_ball.AddReward(1.0f);
        }
        if (previous_agent_touch_ball.team == Team.Blue) //蓝队踢进了蓝队球门，或者没防守住
        {
            previous_agent_touch_ball.AddReward(-1.0f);
        }
        envController.GoalTouched(Team.Purple);
    }
}
void GiveReward()
{
    var ballVelocity = ballRb.velocity;
    ballVelocity.y = 0;
    var ballToBlueGoalDir = GoalBlue.transform.position - ball.transform.position;
    var ballToPurpleGoalDir = GoalPurple.transform.position - ball.transform.position;
    ballToBlueGoalDir.y = 0;
```

```
        ballToPurpleGoalDir.y = 0;
        ballToBlueGoalDir = ballToBlueGoalDir.normalized;
        ballToPurpleGoalDir = ballToPurpleGoalDir.normalized;
        m_PurpleAgentGroup.AddGroupReward(Vector3.Dot(ballVelocity, ballToBlueGoalDir) / 1000);
        m_BlueAgentGroup.AddGroupReward(Vector3.Dot(ballVelocity, ballToPurpleGoalDir) / 1000);
    }
```

7.6 Github: https://github.com/universea/MIE1075_Soccer