ENG5325 Robotics Team Design Project

RoboCup Simulation in MATLAB

Team 1
Oscar Meunier: 2391076M
Yusuf, Abubakar Sadiq: 2359449Y
Xing, Shiyi: 2378951X
Almeida Arencibia, Alvaro: 2395481A
Lam, San Bok: 2426734L
Shi, Jiwei: 2429342S

Abstract

This report describes the development of a MATLAB-based simulation of RoboCup with humanoid robots, adhering to the guidelines and regulations of the Kid-size Humanoid Soccer League and utilising the NAO6 robot design. The report starts with an introduction to the significance of robotic competitions and provides an overview of RoboCup and the specifications of the virtual environment for the simulation.

The methods section explains the model representation of the NAO robot and analyses its kinematics and actuators. Additionally, the section covers the sensors for ball and robot detection. The results of the simulation include ball dynamics, shooting, dribbling, passing, and game state determination, as well as path planning techniques such as Rapidly exploring Random Trees (RRT) and Target-Interceptor will be discussed and evaluated.

The decision-making process is achieved through State flow, while boundary conditions for the court and robot roles are set using specific conditions. The report also presents the validation of the simulation, including visualisation and metrics like the number of goals and outs.

The discussion section addresses the implications and limitations of the simulation. Lastly, this report concludes with future work and recommendations for enhancement of the simulation. Overall, this simulation provides an efficient tool for analysing and testing various strategies and tactics for simulating the RoboCup competition.

1. Introduction

The field of robotics has witnessed significant advancements in recent years, in particular the growth in the domain of walking robots has meant humanoid robots such as the Nao6 have become increasingly capable of performing complex tasks, demonstrating increasing balance and agility. The RoboCup Humanoid League serves as a platform for researchers and developers to investigate various aspects of humanoid robotics, such as dynamic walking, visual perception, self-localization, and team play. In this study, we focus on designing and developing a basic 2D simulation for a humanoid football team based on the Nao6 humanoid robot.

This report presents an account of the methodologies used to create a realistic and accurate simulation environment using information compiled from Nao6 datasheets and footage available online. It focuses on behavioural and algorithmic aspects of the robotic team which includes trajectory planning and decision-making processes. The environment is developed, and robot model representation constructed. Through the simulation of a humanoid football team which consists of four autonomous robots, each with the role of Defender, Attacker and Goalkeeper this report aims to explore their ability to perform various football-related tasks such as kicking, dribbling, passing, running, and intercepting as well as differentiate behaviour based on the role of the robot. Using metrics and validation, control techniques are compared and used to benchmark teams.

One of the main advantages of 2D simulation is that it reduces computational costs compared to more complex 3D simulations and facilitates visualisation for better visual validation. This allows for much faster development as simulations can easily be accelerated and errors can be detected visually. Although the current simulation does not include the kinematics of the Nao robot, it could be extended in future to incorporate a kinematic model that would follow the trajectory described by this simulation, thus providing a more accurate representation of the dynamics of the robot.

The report begins with the background of the RoboCup Humanoid League and the relevant kinematics associated with the Nao6 robot. The methods section discusses the model representation of the Nao as well as ball dynamics, path planning, decision making, and the validation process. Analysis and results are presented in each of these subcomponent methods. This is followed by a Discussion section that critically analyses the findings and an overview of the outcomes of the simulation at the end and suggests potential improvements. Finally, the Conclusion and Future Work section summarises the key findings and outlines possible avenues for further research.

2. Background

RoboCup is a well-known and widely participated competition in the field of robotics. Its primary objective is to simulate a football match by utilising various types of robots (RoboCup, 2023). The main goal of the competition is to create a realistic environment where robots can play football while enabling the implementation of various football strategies and tactics (RoboCup, n.d.). The robots used in the competition are either wheeled or humanoid, providing a diverse range of approaches to the game (RoboCup, n.d.).

This project focuses on developing virtual soccer teams and a virtual environment for NAO robots to play in (RoboCup, n.d.). The humanoid robot players in this project are modelled after the NAO6 robot and are designed to play according to the rules and guidelines of the Kid-Size Humanoid Soccer League. Each team consists of four robots with specific behaviours assigned to them based on their role within the team and the state of play. These behaviours include Striker, Defender, and Goalkeeper, and all teams must follow the general rules of soccer as outlined by RoboCup. The robots' control, guidance, and navigation systems are fully autonomous, with no external operator input. The pitch used for the competition is similar to a standard football field but is smaller in size and has specific markings and dimensions, including two goals at each end of the field and a goal area in front of each goal (SSL.RoboCup, n.d.). The ball used in the game is similar in size to a FIFA Ball Size 1.

3. Methods

3.1 Model representation

3.1.1 NAO

Analysis/research of Nao

The NAO6 robot is a flexible and adaptable platform for robotics research and development (Robopreneur, 2018). Its humanoid shape makes it suitable for a wide range of educational and entertainment purposes, including creating a self-governing robot that plays football. To assess the specification of the NAO robot, we will investigate its physical dimensions, sensors, and actuators to determine its full range of capabilities and potential.

Physical dimension

To play football with NAO robots, it is important to understand the physical properties of the NAO, and its ability to move and interact with the football and other robots. The NAO robot has a width of 31 cm, and a length of 28 cm, and it weighs 5.4 kg (Generationrobots, 2018). It has a head, torso, two arms, and two legs, which are all articulated and allow the robot to move in a human-like manner (Generationrobots, 2018). Since the width and the length of the robots are similar, we can use a circle to model the robot in order to simplify the representation.

Sensors

The NAO robot is equipped with a variety of sensors that allow it to perceive and interact with its environment. One of the most important sensors on the NAO robot is the camera, which provides a wide field of vision for the robot. The NAO robot has two cameras, one located in its head, and another located in its chest. These cameras are capable of capturing both still images and video, allowing the robot to perceive and track objects in its field of vision (Robopreneur, 2018). To incorporate the camera into our simulation, we utilised the maximum field of vision potential of the NAO6 robot, as documented in its datasheet. Since our simulation is in 2-dimension, we can ignore the vertical factor and only model the horizontal field of view in MATLAB. In addition to cameras, the NAO robot is also equipped with a sonar sensor. This enables the robot to detect close objects and allows us to implement additional features like obstacle avoidance in our path-planning algorithm. Another important sensor on the NAO robot is the gyroscope, which helps the robot to maintain balance and detect whether it has fallen. To simulate the fall and recovery time of the NAO, we studied RoboCup videos and counted the time needed for the robot to fall, adjust the pose, and get up (Robots, 2018).

Actuators

The NAO robot's actuators are a critical component for the robot to move and interact with the environment. The robot has a total of 25 actuators, which control its joints and enable it to perform a wide range of movements (Generationrobots, 2018). To simplify the calculation, we assess the performance of the NAO robot's actuators to calculate the velocity of the robot based on its maximum step specification and the time taken to complete a step (Aldebaran, 2018; Cristiano, et al., 2011). This allows us to quantify the robot's movement capabilities and implement it in MATLAB. In addition to linear movement, the NAO robot's actuators also control its angular rotation. By analysing the robot's specifications and calculating the rate of angular rotation, we can obtain the angular velocity of the robot.

Overall, we used the datasheet, Softbank's NAO specifications and RoboCup videos to simulate and model the use of sensors in our simulation in MATLAB. This provides essential references, and it helps to model the performance of the NAO when conducting different tasks and applications. The simplified data is presented in the following table, while the detailed version containing calculations for all parameters will be provided in the Appendix.

Parameter	Value	Unit
Height of nao6 robot	0.58	m
Width of nao6 robot	0.31	m
Horizontal Field of View	57.2	deg
Maximum velocity	0.133	m/s
Maximum angular velocity	50.03	Deg/s
Time taken to fall down	0.55	s
Time taken to get up after the fall	3.90	s

Table 1. Nao specifications

Kinematics

The Nao can be divided into 5 individual kinematic chains with the midpoint of the torso taken as the centre and the edge of each limb as the end-effector.

There is however, a joint in the hip which is coupled in two axes which would make it difficult to use a conventional denavit-Hartenberg representation. This was overcome by simplifying it so that each limb only has two links which would then simplify the inverse kinematics for the robot.

A walking gait would then be generated using the Zero-moment pendulum method (ZMP). To simplify the model a differential drive with 2 wheels was used in place of the kinematic model with the goal of using the more complex model once the simulation itself was considered to be giving acceptable results. The governing equations for a differential drive are as follows:

For linear velocity:

$$v = \frac{r}{2} * (v_r + v_l)$$

Angular velocity is calculated by,

$$\omega = \frac{r}{2d}(v_r - v_l)$$

In both these equations r is the radius of the wheels, d is the distance between the wheels and v_r and v_l are the right and left linear velocities of the wheels, respectively.

The state-space model representation of the system can be expressed as follows:

$$\dot{X} = AX + BU$$

Where the state vector X and input vector U are defined as:

$$X = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \quad U = \begin{bmatrix} v_r \\ v_l \end{bmatrix}$$

A is simply an 3x3 Identity matrix and B is:

$$B = \begin{bmatrix} \frac{r}{2}\cos(\theta) & \frac{r}{2}\cos(\theta) \\ \frac{r}{2}\sin(\theta) & \frac{r}{2}\sin(\theta) \\ -\frac{r}{2d} & \frac{r}{2d} \end{bmatrix}$$

Giving:

$$\dot{X} = AX + BU = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} X + \begin{bmatrix} \frac{r}{2}\cos(\theta) & \frac{r}{2}\cos(\theta) \\ \frac{r}{2}\sin(\theta) & \frac{r}{2}\sin(\theta) \\ -\frac{r}{2d} & \frac{r}{2d} \end{bmatrix} U$$

Ball and robot searching Algorithm.

The ball searching function (‘searchBall(obj, ball_pose)’) is designed to search for a ball in front of an NAO robot, given its pose and the pose of the ball. The function takes in two inputs, the object representing the NAO robot and an array representing the position of the ball. It then uses the position and orientation of the robot, along with the position of the ball, to determine if the ball is within the robot's field of view. The function first obtains the current position and orientation angle of the robot, as well as the position of the ball. It then calculates the minimum and maximum angle of the sector within the robot's field of view. After that, the distance between the robot and the ball is calculated by using the Euclidean distance formula. The function establishes a condition where the robot's orientation must be greater than the minimum angle of the sector and smaller than the maximum angle of the sector, while the distance between the robot and the ball must be within the sensor radius. If this condition is met, the function will return true, indicating that the robot has successfully found the ball.

The robot searching function (‘searchRobot(obj,robot_idx,totalNumRobot, robots)’) searches for a target robot within the field of view of the NAO robot. Unlike the ball searching function, the robot searching function utilises a loop to check the position of each robot in the array against the position of the NAO robot. It uses a similar algorithm in the ball searching function to calculate the distance between the NAO robot and the target robot, and checks if the target robot falls within the sector of the NAO robot's field of view. It then returns a Boolean variable to indicate whether a robot is found.

In summary, both functions involve checking the position of an object in the NAO robot's field of view. The ball-searching function is used to locate the position of the ball on the football field and then pass the result to the path-planning algorithm. The purpose of the robot searching function is to detect the presence of any obstructing robots in the path of the target robot, making it useful in obstacle avoidance scenarios.

Falling conditions

Given the inherent instability of humanoid robots, it is common for Nao robots to fall during the game as they are exposed to various unexpected disturbances throughout. As such, it is crucial to incorporate a collision check into the simulation to account for this scenario as well as a function that represents getting back up.

In the current 2D simulation, a simple collision check determines, based on the radius of the Nao robot's representation if the robots have touched. If a collision is detected, the Nao robot is considered to have fallen and will subsequently take a certain amount of time to get back up, this is based on the average time of Nao robots' recovery time determined in prior research found in table 1.

While this approach provides a basic representation of collision detection and delay caused by falling, it does not account for the direction of forces and masses involved in a collision or of the colliding objects. To enhance the realism and accuracy of the simulation, future improvements could involve incorporating a more comprehensive collision model. This might consider the force required to knock a Nao robot over, as well as the direction and mass of the colliding objects. This layer of complexity would provide a more accurate representation of disturbances faced by the nao in a football match, ultimately leading to better strategies and decision-making algorithms that can account for the potential instability of the robots.

3.1.2 Ball

Ball dynamics and representation

The process of simulating a soccer ball involved the creation of an object within the simulation that could replicate the behaviour of a ball. In order to generate an accurate representation, the ball needed to fulfil specific requirements, such as being visible and able to be picked up by players, and possessing a force model which would gradually reduce its velocity to zero due to friction with the field. To accomplish this task in a structured manner, a class was implemented within the program, which facilitates the retrieval of the ball's properties and functions in a convenient manner.

The initial step involved determining a suitable approach for representing the ball within the simulation. This was achieved by developing a plotting function that depicted a black dot within a grid, corresponding to the ball's location. Additionally, the dot's size was calibrated to have a circumference of 45 cm as stipulated in the project specifications. However, it should be emphasised that during the majority of the testing, the ball's radius was increased to 50 cm for improved visualisation during validation procedures. Moreover, to assist with testing the dribbling functions, an orientation parameter was assigned to the ball, although it was not immediately apparent in the ball's representation Figure 1.

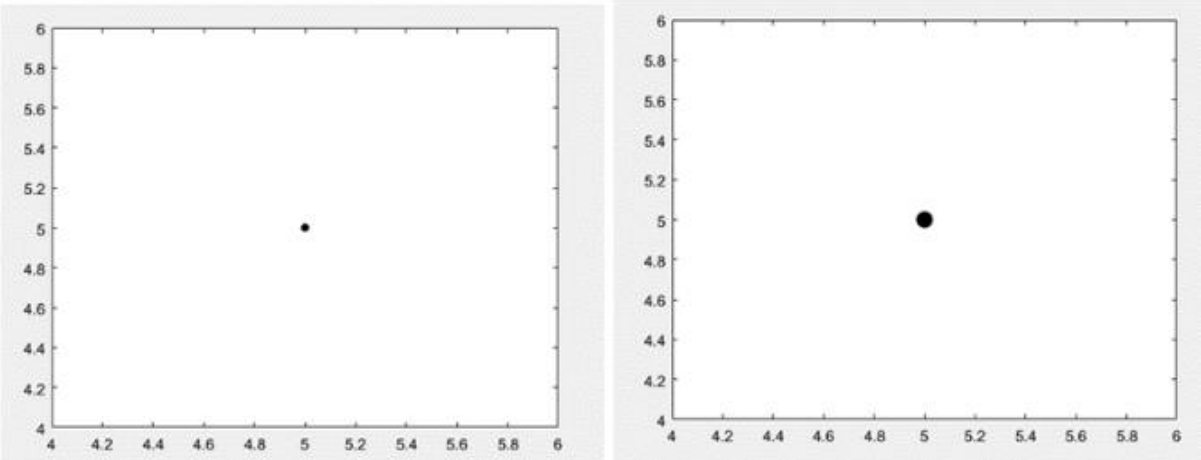


Figure 1. Simulated ball in comparison with the ball used for testing and validation procedures..

The following step was to identify a suitable mathematical model that could replicate the ball's behaviour, accurately calculating its position and velocity. After conducting extensive research, it was decided to incorporate a damping factor into the velocity equation during each iteration [1]. This mathematical model was then adapted for implementation in MATLAB, using the equations outlined in [Equation 1] and [Equation 2].

$$Position = Position + (Velocity * dt) \; [Equation\ 1]$$
$$Velocity = Velocity - (c * Velocity) \; [Equation\ 2]$$

Both the position and velocity of the ball were represented as 1x2 matrices, denoting the x and y positions [x, y]. The damping coefficient, denoted by c, was used to adjust the velocity based on the current velocity of the ball, resulting in high changes in velocity for a fast-moving ball. This replicated the friction between the air, soccer pitch, and soccer ball. There were numerous values for c that could approximate the behaviour of a real ball. Extensive research was conducted to determine the optimal value for c, with the damping coefficient being set to c = 0.2 based on a series of trials testing the passing and shooting functions.

Once the ball had functions for being plotted and updating its position, the next step was the creation of additional functions which could vary the input velocity given shooting, passing or dribbling procedures.

Shooting

The conceptual framework for designing the shooting algorithm was predicated on establishing specific conditions that would prompt the robot holding the ball to execute a shot. After conducting further research on the topic, it was determined that two essential conditions must be fulfilled for shooting to occur: the robot must be in possession of the ball and positioned near the opposing team's goal [2][3]. Consequently, functions were devised to establish these requisite conditions.

The first step involved creating a property within the Nao class to determine whether the ball was in the robot's possession. As previously demonstrated in the dribbling function, the velocity assigned to the ball during dribbling was equivalent to the Nao robot's velocity at the time of charging the ball. This function utilised the Nao's identification number to ascertain whether any other robots had the same ID number. If another robot shared the same ID number, the "holdball" variable was set to 1; otherwise, it was set to 0.
Subsequently, a function was developed to determine whether the Nao robot was positioned near the goal. This necessitated the creation of specific regions on the field where the robot could reasonably be expected to score a goal. These areas of increased scoring potential were designated as the red squares in Figure 2.

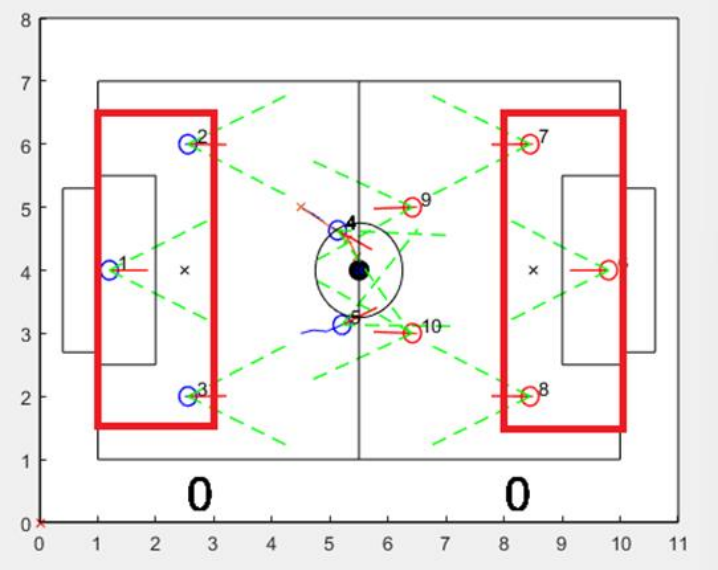


Figure 2. Critical areas for shooting.

As evident from Figure 2, the red rectangles represent optimal positions from which a professional football player would aim to score. It should be noted that verifying the correct position necessitated determining the team the robot was assigned to. If the robot belonged to team 1, it had to be situated in the left rectangle to satisfy the position requirement. Conversely, if the robot was a member of team 0, it had to be located in the right rectangle to be suitably positioned for shooting. The "readytoshoot" function is depicted in Figure 3. The block diagrams were created using the online tool "Lucidchart" [4].

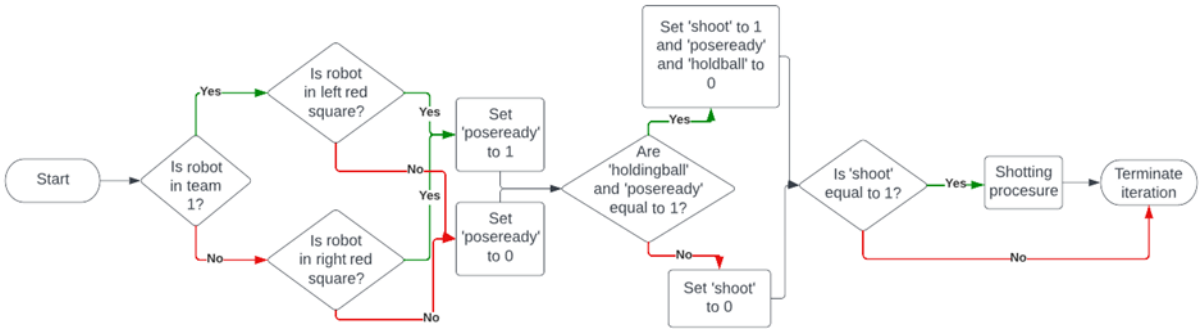


Figure 3. 'readytoshoot' function represented in the block diagram.

As shown in the diagram, the function operated by first checking whether the robot was in possession of the ball and properly positioned. If both conditions were met, the previous variables were reset to 0, and a new variable "shoot" was set to 1. Finally, when "shoot" was equal to 1, the shooting action was executed. This function was integrated into the simulation loop, verifying these conditions with each iteration.

The shooting procedure entailed the Nao robot executing a kick with its maximum force, which is approximately 25 Newtons according to the robot specifications [5]. The conversion from output force to input velocity was determined by the kinetic energy. This process is described by the following equations [6]:

$$W = F * d = \Delta K; \qquad K = 1/2 * m * V\text{^}2$$
$$W = K\_final - \langle\!\langle K \rangle\!\rangle\_initial; \qquad \langle\!\langle K \rangle\!\rangle\_initial = 0 \; as \; V\_initial = 0$$
$$W = K\_final = F * d; \qquad 1/2 * m * V\text{^}2 = F * d$$
$$V = \sqrt{((2 * (F * d))/m}; \qquad [Equation\ 3]$$

Where W represents work done, K denotes kinetic energy, F indicates force, d refers to the distance moved, m represents the mass of the ball, and V represents the velocity. As evident from the equations, the work done is equivalent to the difference between kinetic energy, and since the initial position is 0, the work done can be equated to the final kinetic energy. After several mathematical manipulations, the input velocity for the ball was determined [Equation 3].

Furthermore, it is worth noting that as the final step in the shooting procedure, the "shoot" parameter is reset to 0.

Passing

The cognitive process for the passing procedure was analogous to that for shooting, only happening under specific conditions. Although a broad range of conditions could have been established for passing, it was determined that a simple algorithm should be created initially. Consequently, the conditions for passing were decided to be when the robot had possession of the ball and two or more players were in close proximity to the ball.

The first condition was straightforward to represent as a function had already been built to verify whether robots had possession of the ball. Conversely, checking whether the opposing team's robots were near the ball required examining whether the robots were within the field of view. By combining these two conditions, a "needpassing" function was created. However, additional functions were required to determine which player to pass the ball to and to execute the passing procedure.

The robot to which the ball was to be passed was determined based on its location. To achieve an effective game strategy, it was decided that passing should always be made to the robot that was closer to the goal where scoring was needed. To accomplish this, a team-based function was created to assess which robot on each team was closer to the goal. This function was named "robottopass" and is shown in Figure 4.

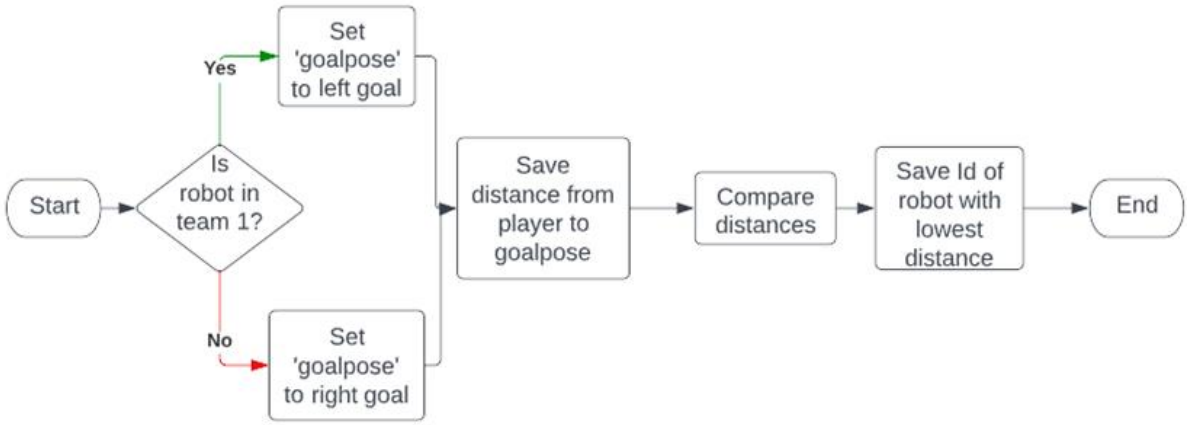


Figure 4. "robottopass" function is illustrated in a block diagram.

Once the robot to pass to was established, the next step was to pass the ball to an area near the position of this robot. Initially, the idea was to pass the ball directly to the player's position, but this would likely have been inefficient as the other team's robots could have detected the ball being passed and attempted to intercept it. Therefore, a more effective approach was to add a small distance towards the goal position to the robot's position, making it harder to estimate the passing position. Additionally, small advances were made in reaching the critical areas where shooting could be executed. This function was labelled "desiredposed" and can be seen in Figure 5.

Figure 5. "desiredposed" function illustrated in a block diagram.

Subsequently, velocity was calculated using the work and kinetic energy equations, similarly as it was done for shooting. However, for passing, a specific force was required so that the ball passed through the passing point at a speed that could be intercepted by other robots. This was easily calculated using a multiplication factor added to the distance [Equation 4].

$$F = a * d \quad \text{[Equation 4]}$$

Where a is the multiplication factor. After extensive research and testing, table 1 was obtained.

Force Multiplier	Damping coefficient
0.0565	0.5
0.0365	0.4
0.012	0.2
0.006	0.1

Table 2.

At this stage of the project, a damping coefficient had not been established, so testing for an accurate force multiplier was conducted. Figure 6 depicts the entire passing procedure in a block diagram.

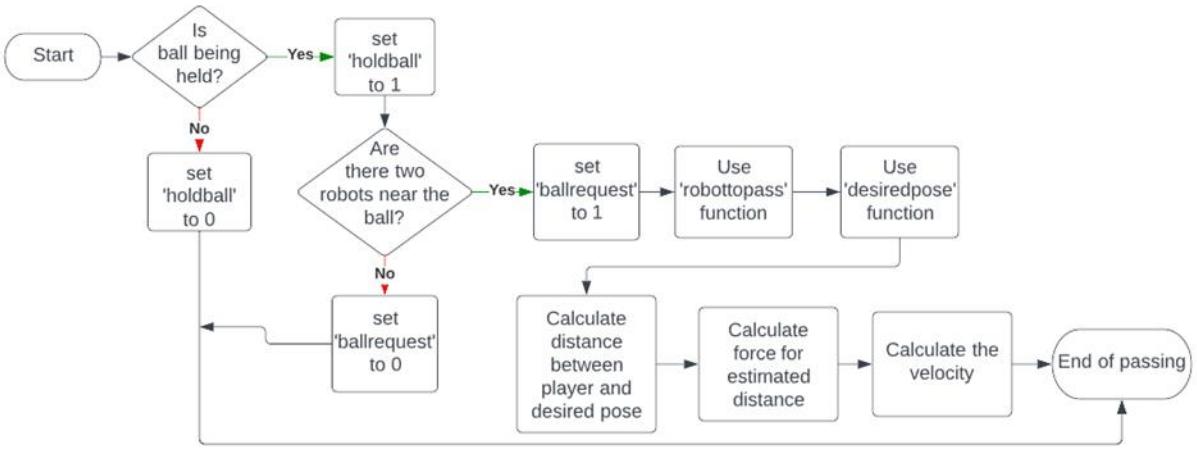


Figure 6. Passing procedure in a block diagram.

It must be noted how certain variables such as “holdball” or “ballrequest” are reset to 0 in this process.

Dribbling

The dribble function is implemented similarly to the pass and shoot functions, which are only activated under certain conditions. First, a simple algorithm needs to be built to trigger the dribble logic at the appropriate time. Therefore, the dribbling condition is determined to be when the robot is near the ball and no other robots are dribbling.

To achieve this condition, the distance between the robot and the ball needs to be checked first, which can be achieved by computing the Euclidean norm. At the same time, it also needs to check whether other robots are dribbling. By combining these two conditions, a function called robotDribble is created.

When the dribble condition is met, the robotDribble function will calculate the velocity vector of the robot based on its direction of motion. Then, set the velocity of the ball to be the velocity of the robot, and set the position of the ball to be 0.5 units away from the robot. The function also updates the dribble robot ID and the direction of the ball.

During the dribble process, it must be ensured that the ball is always close to the front of the robot, so the position and direction of the ball need to be updated according to the position and direction of the dribbling robot. If there is no robot dribble, the robotDribble method will update the ball's position and orientation based on the ball's velocity.

Some challenges were also encountered during the implementation of the robotDribble method. Keeping the ball close to the front of the robot is also an issue when implementing the dribble logic. However, these problems were resolved after precise dribbling function debugging, so that the dribbling function was realised. Throughout the implementation process, it is necessary to pay attention to the update of some variables (such as the position of the ball, the ID of the dribbling robot, etc.) to ensure the correct dribbling logic.

The results of this section must be divided into the evaluation of four sections: ball dynamics, dribbling, shooting, and passing.

When the plotting and update functions of the ball were added to the simulation code, they worked as expected. As mentioned earlier, several parameters had to be adjusted in the testing section to achieve better visualisation, but no significant issues were found with this section.

Dribbling proved to be challenging as it did not initially work with the rest of the code. Furthermore, once this issue was resolved, the ball was not held on the nose of the robots, leading to additional problems. However, this was debugged with an accurate dribbling function.

Both shooting and passing presented difficulties. Due to the large number of functions that had to work together, this section demonstrated high complexity and unexpected errors. Some of these errors included variables not being communicated between functions, resulting in program crashes. However, these issues were eventually resolved, and both passing and shooting were achieved as independent procedures. When added to the rest of the code, however, the functions were unable to perform their tasks.

3.1.3 Gamestate

When the game enters the GameState, the BallTracker class plays a key role in the simulated football game, which is used to monitor the position of the ball in real time, judge the state of the ball (crossing the line, scoring, etc.) and update the team's score. In the goal state, a series of methods of BallTracker will work together to ensure that the game progress is tracked correctly, and the score is updated in real time.

First, when the game starts or the position of the ball changes, the updateBallPos function will be called to update the position and score of the ball. This function first assigns the new position of the ball to the ballPos attribute, and then calls the functions of isBallOutOfBounds, isBallInLeftGoal and isBallInRightGoal according to the new position of the ball to check whether the ball crosses the line or enters a goal.If the isBallInLeftGoal or isBallInRightGoal function returns true, indicating that the ball has entered the left or right goal, a red "Left goal!" will be displayed at a specific position. Or "Right goal!" Text information. At this time, according to the goal, the score of the corresponding team will increase by 1 point. After the score is updated, the game will be suspended, and the referee will serve again in the middle circle to resume the game.

In a word, BallTracker class plays a significant role when the game enters the goal state. By using this kind of methods, we can accurately monitor the state of the ball, track the position change of the ball, and update the team's score in real time according to the actual game situation. This helps to provide a better game experience for the audience and analysts, while ensuring the fairness and accuracy of the game.

3.2 Path planning

Path planning is an essential component of robotic soccer, as it enables robots to navigate the field efficiently while avoiding obstacles and achieving their objectives. In our simulation, we employ two different path planning algorithms: Rapidly-exploring Random Trees (RRT) and Target-Interceptor.

3.2.1 Rapidly-exploring Random Trees (RRT)

RRT is utilised when the robot needs to navigate towards a static location, such as when an attacker has possession of the ball and wants to reach the goal line. This algorithm provides obstacle avoidance by using an occupation map that is updated every second such that:

$$n = \frac{1}{\Delta t}$$

Where n is the number of timesteps taken before updating. While updating the occupation map at every timestep would yield more accurate results, it would also be computationally expensive, making the current approach a reasonable compromise.

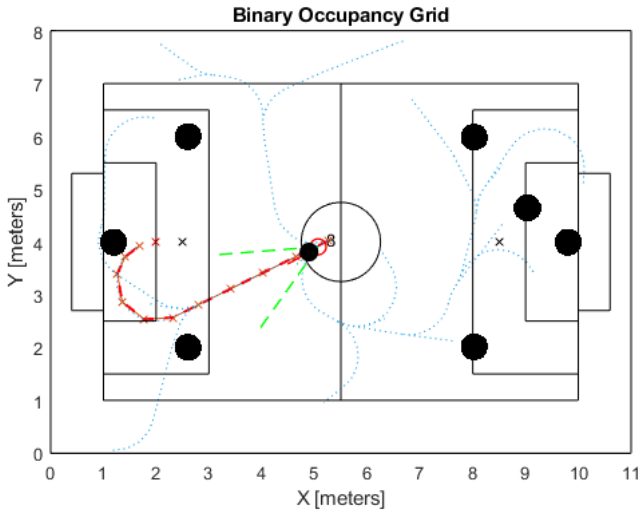


Figure 7. RRT search algorithm represented with path found

The benefits of RRT are that it provides efficient exploration as it is capable of quickly exploring large search spaces and converging to a solution, making it suitable for real-time applications like robotic soccer. It also offers the advantage of probabilistic completeness, which means that the probability of finding a solution approaches one as the number of iterations increases, provided a solution exists as well as it being able to handle complex environments where it can efficiently deal with high-dimensional and complex environments containing obstacles.

It does however come with some disadvantages. RRT often generates suboptimal paths, as it prioritises exploration over path optimality. It is also sensitive to parameters such as the step size and expansion factor, which can significantly impact the quality of the generated paths.

To improve upon this path planning algorithm the RRT* could be implemented which is an extension of the previous algorithm, RRT* can be used to iteratively refine the generated paths, improving their optimality over time ensuring a reasonable path is found. To further improve on this the implementation of informed RRT* would improve convergence time as this incorporates heuristic information such as euclidean distance desired heading. This can guide the search process towards the goal more effectively, resulting in faster convergence and more optimal paths.

3.2.2 Target-Interceptor

The Target-Interceptor algorithm is commonly used in scenarios where a robot aims to intercept a moving object. Examples of its application include missile guidance systems, autonomous vehicles tracking moving targets, and aerial drones intercepting other drones. In the context of robotic soccer, this could involve a defender attempting to intercept an attacker with the ball or a robot intercepting a pass.

The formulas used in the Target-Interceptor algorithm used with Proportional Navigation (PN) guidance, which is applicable for any Line-of-Sight (LoS) guidance problem and can be used for any type of vehicle. The LoS rate is calculated as:

$$\dot{R} = (V_x^r R_y - V_y^r * R_x)/(R_x^2 + R_y^2)$$

The acceleration command is then calculated as:

$$n = N\dot{R}$$

So that acceleration is then:

$$a_x = nR_y$$
$$a_y = -nR_y$$

The benefits of using real-time a target-Interceptor algorithm is that it provides real-time guidance for intercepting moving targets, making it well suited for dynamic scenarios such as the interception of a player and ball during a game. It does however require knowing the state of the target and interceptor. The current implementation uses a simplified model of the ball as it assumes that its speed of the ball is constant.

To be used in real world Robocup it would need to have a more accurate representation of the ball as well as the robot dynamics. To improve on this, integrating adaptive control techniques into the target-interceptor algorithm could compensate for uncertainties and non-linearities in the system caused by the other players. Machine learning algorithms, such as deep reinforcement learning, could also be employed to learn optimal interception strategies based on simulation as well as real-world data.

The incorporation of the improvements for these path planning techniques the system could effectively model the navigation.

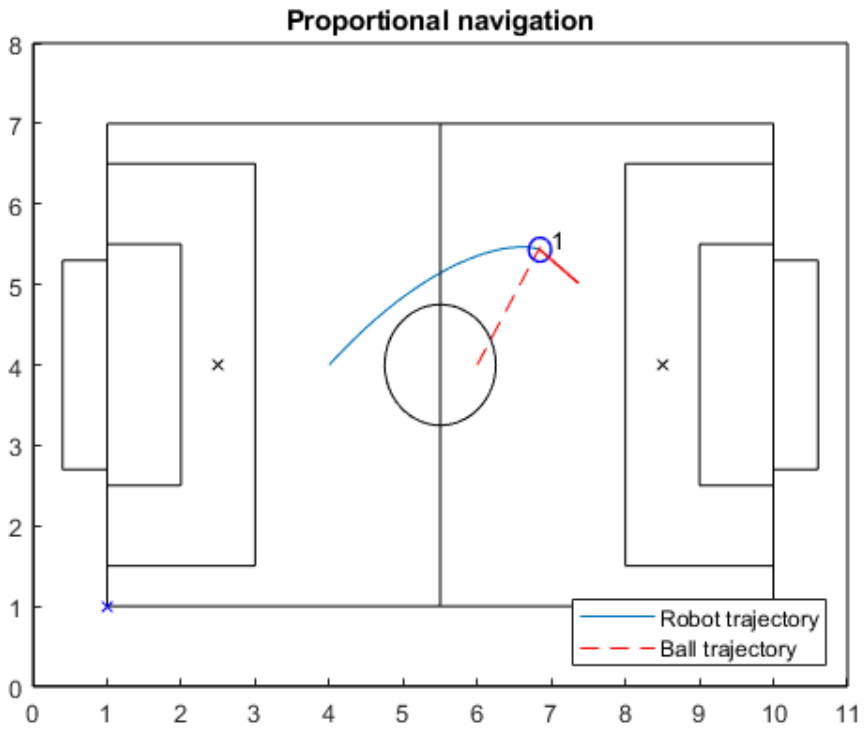


Figure 9.

3.3 Decision Making

3.3.1 Decision-Making Algorithm

There are various ways of implementing a system for each player to make decisions. This could be through the use of finite state machines (Aleluya et al,2018) or through the use of a Decision Tree(Huang and Liang,2002)

Finite State Machine: A mathematical model used to represent the behaviour of a system by defining a finite number of operating modes(states) and the conditions under which the system switches from one state to the other(transitions).(Mathworks,n.d) For this system there were to be three different states: Attacking, defending and keeping with action decisions within each of them. The transitions could then be modified in order to implement different strategies.

Decision Trees: A hierarchical model used for decision making, it consists of nodes, each representing a point at which a decision is to be made. Each outcome from a node is represented as a branch(IBM, n.d).

It was decided that decision trees should be used as they are more straightforward to implement in MATLAB using a series of If-Else statements. This allowed for the code to be more easily explainable and easier to troubleshoot.

An algorithm was created to control the decision-making of the robot. It is based on a Decision tree.
At the highest level, the behaviour is based on teams which allow for different strategies to be implemented for different teams.

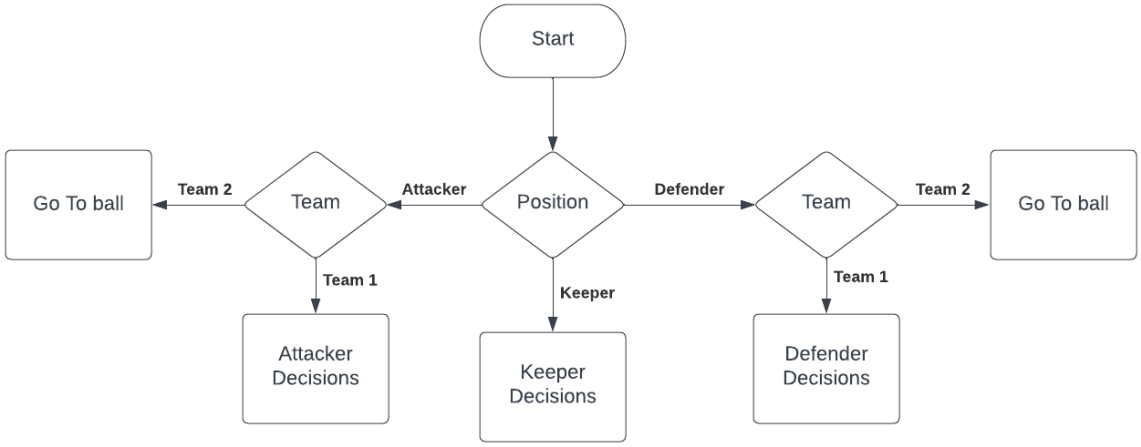


Figure 10.

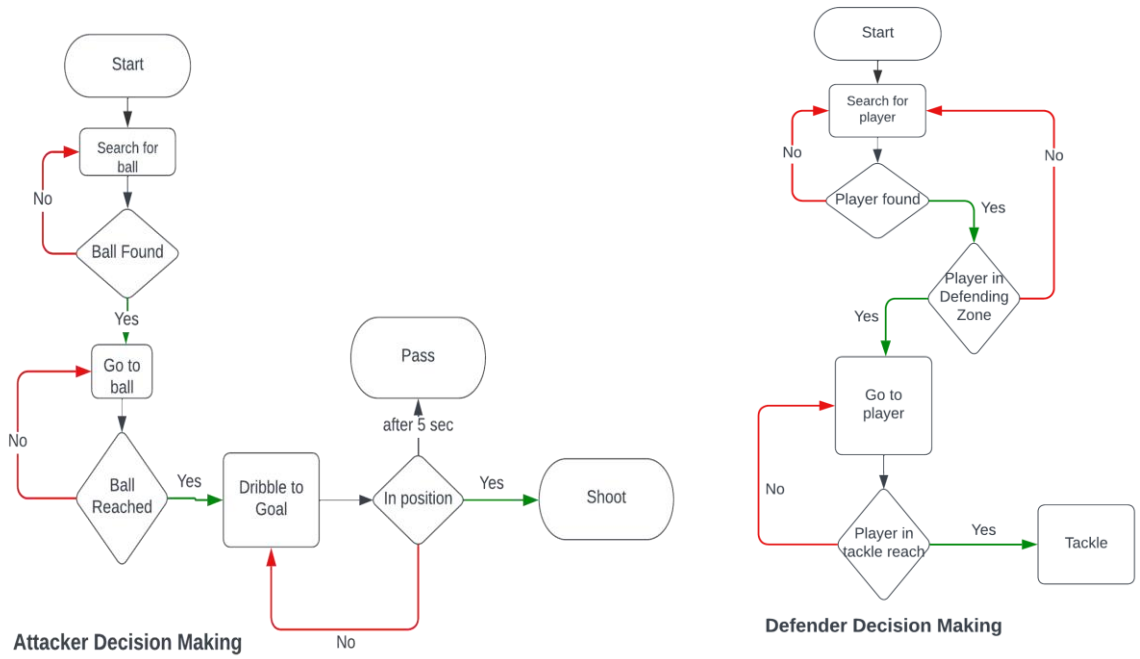


Figure 11. Decision Making Flow-Charts

For the Attackers; whenever the attacker is without the ball, it searches for whether the ball is within the robot's field of view. The robot then uses Proportional Navigation to plot a suitable trajectory towards the ball. Once it is within reach of the ball, the robot then plots a trajectory towards the opposing side's critical shooting area using RRT. At this point, the robot attempts to shoot straight ahead from its heading direction or passes if there is no

clear line of sight for shooting. The RRT is useful for avoiding obstacles, and it is updated every ten (10) time steps to account for the movement of obstacles in that time period.

For the defenders; the players search for whether there is an opposing player within the field of view which is within the zone of the defender. It then uses Proportional Navigation to plot a trajectory towards the player. Once it is within reach of the player to be tackled, it attempts to take the ball away from the player. If successful, it then continues forward in the heading from which it took the ball.

For the Goalkeepers, the player uses the ball sensor to find the ball, then tries to keep the ball at the centre of the player's field-of-view. This is to try to ensure that the ball would be headed straight for the centre mass of the keeper, where it can be stopped.

3.3.2 Boundary Conditions - Court

In the simulation, the court is divided into three different areas, goal, infield and outfield. In order to realise the boundary restriction of the ball on the court, the BallTracker class uses the isBallOutOfBounds method to check whether the position of the ball is within the court boundary. This method will check the position of the ball according to the minimum and maximum X and Y coordinates of the court boundary. If the ball is within the boundary, the method returns true, otherwise it returns false. Then by updating the position, when the ball is out of bounds, the screen will prompt that the ball has been out of bounds. In the BallTracker class, there is an attribute representing the boundary of the court, fieldPos, which is a vector used to represent the boundary of the court, including left, bottom, right and top. In a word, in this simulation, by checking the position of the ball, it can be ensured that the ball always stays within the boundary of the stadium, thus simulating the real game scene.

3.3.3 Boundary Conditions - Robot roles

Our simulation divides all robots into three roles: Attacker, Defender, and Goalkeeper. Each of these roles has its designated field area where they are expected to perform their tasks. The goalkeeper is responsible for guarding the goalpost and therefore is given a smaller area called the penalty area to operate. The defenders operate in the half-side of the field and are responsible for stopping the opponents from scoring. On the other hand, the attackers do not have a designated boundary and can move freely across the field to score goals.

To simulate the boundaries of these roles, we can design and implement the boundary as follows: for the goalkeeper, the boundary can be defined as the penalty area. For the defenders, the boundary can be defined as the half-side of the field. As for the attackers, there is no boundary, so they can move freely across the field. To implement these boundaries in MATLAB, we can use the two functions: checkSelfBoundary() and checkBallBoundary(). These two functions use the same searching algorithm and they both return a Boolean value indicating whether the robot or the ball is within its boundary.

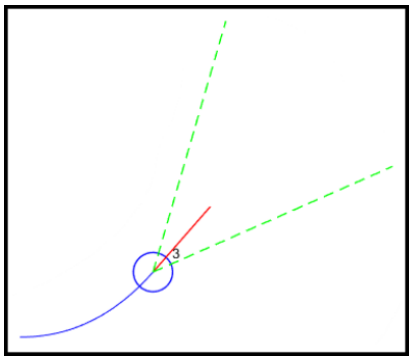
The first function involves checking whether the robot's current position is within the range of the x and y coordinates specified by the boundary. The robot class stores the boundary

coordinate using four variables, namely x1, y1 and x2, y2. These variables represent the coordinates of the top-left and bottom-right corners of the boundary, respectively. The function checks the x-coordinate of the robot's position against the minimum and maximum x-coordinates specified by the boundary. Similarly, the function checks the y-coordinate of the robot's position against the minimum and maximum y-coordinates specified by the boundary. If both checks return true, then the robot is within the boundary, and the function will return true. The ball boundary-checking function verifies whether the position of the ball is within the specified boundary of the robot. It takes in the object representing the robot and the position of the ball as input. This function is used to check if the ball remains within the target robot's respective boundaries.

3.4 Validation

An important aspect of our simulation is ensuring its accuracy and effectiveness in replicating real-world robotic soccer scenarios. To achieve this, we employ various validation techniques, including visualisation, metrics, and results analysis.

3.4.1 Visualisation



Visualising the simulation environment and the interactions between the robots and the ball allows for the understanding of the behaviour of the system. It allows developers to ensure that the desired behaviours are being implemented correctly as well as identify areas of improvement.

As seen in the figure each robot is represented by circles with a radius corresponding to the width of the NAO, see appendix A.1. The orientation of the robot is shown by the red line, along with the field of view of its sensors by the green dotted lines. This helps provide intuition about the robot's behaviour and assists with debugging and validation. The ball is represented using a plain design, making it easy to track its movement and interactions with the robots and the environment. The football pitch is drawn according to the specifications of the RoboCup Humanoid League, ensuring a realistic playing environment, see appendix A.2. Other optional visualisations of specific elements were added for further validation and understanding of the underlying code, such as the waypoints generated by the RRT algorithm, and goal pose trajectory which can be seen in figure 12 by the blue line.

By incorporating these visualisation techniques, development is improved as one can gain valuable insights into the performance of our simulation and identify areas that may require further improvement or optimization.

3.4.2 Metrics

To establish benchmarks for evaluating a robot team's aggressiveness or defensiveness, data or statistics from historical matches should be searched. In the project, the metrics for benchmarking used to evaluate the performance of the robot teams are the number of goals and ball out of pitch boundary, which can indicate the play styles and strategies of the robot teams. If the number of goals is higher than it of benchmarks, the robot team plays in an offensive style, while the robot teams plays in a defensive style if the number of outs is higher than it of benchmarks.

To record the number of goals and ball out of pitch boundary, the game state will be recorded in a short sample time period during the game, in the main file, a CSV file will be opened for writing data into it. To gain stable data of the game state, the main file should be run for several times to calculate the average performance of two robot teams. In order to present the game state data, a new script containing a loop is created. By reading the CSV file produced by running the main file, the newly created script will record the time and the number of goals and ball out of pitch boundary in every football match from the last row of pulled data in the CSV file. To generate an intuitive presentation of reading data, the script plots the bar charts for each robot team. The first figure consists of two side-by-side bar charts that display how many goals each team scored in each game, while the second figure is a bar chart that displays how many outs each game had, and a horizontal line showing the average value for each dataset appears in both figures.

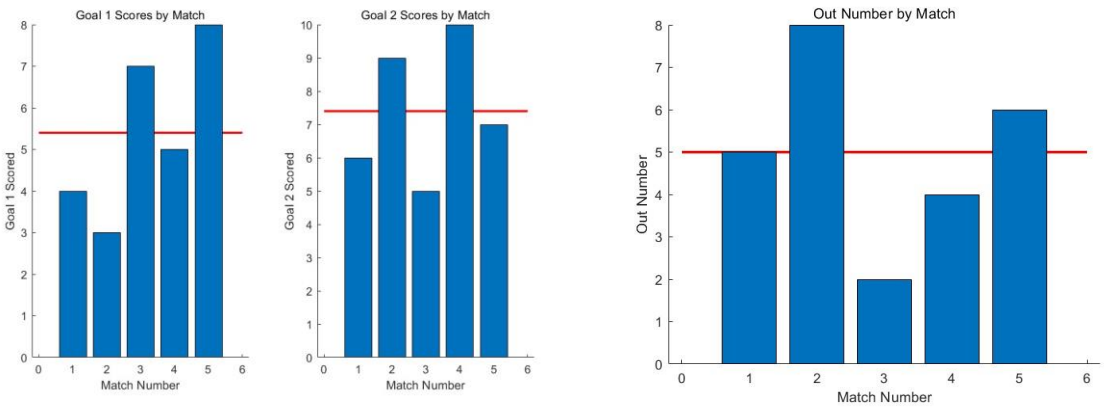


Figure 13. Example figure for recent 5 games record

Overall, metrics offer a straightforward method to examine the game states from numerous football matches and show the outcomes using bar charts and horizontal lines. Through comparing the average numbers with historical benchmarks, the robot team play style can be acquired, allowing further improvements on robot team simulation to be made.

4. Team Performance

	Oscar	Alvaro	Abuakar	Sanbok	Shiyi	Jiwei	Total
Time (hours)	113.25	97	93.5	96.5	75	74.5	529.75

Costs			
Staff costs	Expert costs	Total costs	Predicted costs
£54, 975	£12, 000	£66, 975	£125, 000

Table 3. Total costs and predicted costs in project

To analyse the team performance, the total working time of staff should be calculated firstly. According to the time allocation table, the total working time was 549.75 hours, for a staff cost of £100 per hour, the staff cost is calculated as £54,975 in total. Team meetings with our supervisor counted as expert assistants billed at a rate of £1000 an hour. This accounted for £12,000 of the final budget. As a result, the total costs come to £66, 975. This is lower than the predicted cost, with a gap of £58,025. With this additional budget this could be reinvested in improving the simulation further and implementing the suggested improvements.

5. Discussion

In terms of model representation, the utilisation of a vast number of functions responsible for ball behaviour resulted in unexpected errors when used collectively. Some of these errors were attributed to non-uniform code standards, such as the use of varying sizes of matrices during the recall of position and velocity parameters. Improvements in the documentation of codes between team members could have mitigated this issue. Additionally, further passing and conditions could have been incorporated to enhance the game strategy. It should be acknowledged that despite the possible enhancements, the overall ball functionality was successful.

As for path planning the report explores two path planning algorithms: Rapidly-exploring Random Trees (RRT) and Target-Interceptor. Using RRT for navigating towards static locations offers efficient exploration as well as obstacle avoidance. However, it generates suboptimal paths and is sensitive to certain parameters. Improvements can be made by implementing RRT* and informed RRT* for more optimal paths and faster convergence.

The other method explored is the Target-Interceptor algorithm focuses on intercepting moving objects and using Proportional Navigation (PN) guidance. It provides real-time guidance for moving target interception but requires knowledge of the target and interceptor states. The current implementation assumes constant ball speed and needs a more accurate representation of ball and robot dynamics for real-world application. Incorporating adaptive control techniques and machine learning algorithms can compensate for uncertainties and enhance interception strategies. By refining these path planning algorithms, the system can effectively model navigation for robotic soccer players in complex environments.

The chosen decision making algorithm worked for the task it was meant for. However, there were issues particularly to do with the attacking players which due to RRT were often able to dribble past the defenders before the defenders could update their path to intercept them. This was to some extent balanced out when the defenders were able to get to an interception point before the attacker's controller was updated. The rigidity of the decision tree used also meant that it was harder to change strategies on the fly. The rigidity of the algorithm also meant that once one side had established dominance, it was kept that way for the rest of the match.

From the perspective of visualisation in validation, it provides intuitive presentations on robots, ball and pitch, which allows the simulation to be monitored, and arising problems in the simulation can be tackled in time. Other elements can be added to the visualisation to improve the simulation because of the easy accessibility of the visualisation. As for the metrics part in validation, though comparing the number of goals and outs with benchmarks can easily help to judge the play style and strategies applied by robot teams, the realistic football match will consider more factors like passing and interception to evaluate a football team. Recording and comparing more factors during matches will help to evaluate the robot teams to improve the simulation, while large numbers of matches or longer match time should be applied to assure the accuracy of themetrics.

6. Conclusion and Future Work

6.1 Future work

The simulation's overall performance was deemed average due to several underlying factors. However, a comprehensive analysis was conducted to explore potential enhancements that could improve the simulation's performance. This section of the report focuses on discussing these improvements and their implementation.

Regarding the overall simulation, a 3D implementation could have been implemented. This would entail various modifications to the existing code. Firstly, a 3D model representing the NAO robot would need to be implemented, requiring the use of a kinematic model instead of a differential drive. Inputs and outputs would need to be added to the robot, such as the ability to change joint angles using input voltages. Secondly, the ball would need to have bouncing properties to simulate its behaviour more accurately. Additionally, incorporating spinning mechanics to allow the ball to rotate would further benefit the simulation.

Furthermore, certain improvements were required in the path planning aspect of the simulation. It is worth noting that the current implementation of the Rapidly-exploring Random Tree (RRT) resulted in collisions between robots since the RRT was updated every 10 iterations. An improved version of this system would have been to use RRT*, which refines the generated paths and improves their optimality. Additionally, the generation of paths took a longer time due to a lack of information, so an informed RRT utilising the Euclidean distance, potential fields, and learned heuristics could have decreased the computation time significantly. Another alternative approach that could have been more efficient is utilising the field of view of the robots for path planning. This simpler system could have updated itself on every iteration, resulting in improved performance.

Moreover, it would have been beneficial to add further inter-robot communication. Several parameters such as player positions of the inner and opposing teams, ball position, and potential passing opportunities could have been exchanged, which would have allowed for the creation of more precise and complex algorithms. This enhanced communication system would have also enabled the use of neural networks for weighted decisions. Although training such a network would have increased the computational time significantly, it would have greatly benefitted the attacker, defender, and goalkeeper algorithms.

Finally, the team's performance could have been improved by defining coding standards. As previously mentioned, when adding object classes together, crashes occurred between functions due to different matrix shapes and variables. Additionally, further documentation of individual codes would have aided in understanding the logic of team members. Shorter group meetings could have helped in keeping the team updated on code changes. Miscommunication between the team resulted in certain steps taking longer than expected, which could have been resolved by confirming task development and completion.

6.2 Conclusion

The objective of this project was to create a soccer match simulation utilising NAO robots as players, as stipulated by the project specifications. The initial phase involved extensive research on multiple platforms suitable for running the simulation. While Weebots was the original platform of choice, it was deemed suboptimal due to limited computational resources. Consequently, a decision was made to develop a 2D simulation using Matlab.

The team's tasks were divided into two distinct sections: the development of the kinematic model and the implementation of behavioural algorithms. Although the simulation was conducted in 2D, kinematic calculations for NAO robots were carried out to enable all required movements. Conversely, ball dynamics were created for the purpose of plotting and mathematical modelling. It is noteworthy that ball movement was achieved using a velocity-damping model, which was simplistic yet maintained a high level of accuracy for ball position. Additionally, an NAO class was developed to facilitate the implementation of robots in the simulation. This class included parameters for all robot properties, such as position, velocity, and orientation. Furthermore, functions were created for passing, shooting, and dribbling the ball. These functions were linked to the player's role, such as goalkeeper, attacker, and defender. The algorithms for these roles were based on the ball's position and the player holding the ball. Defenders were responsible for tackling attackers, while attackers aimed to take control of the ball and score a goal. Moreover, an RRT was implemented to generate path planning for the robots. Finally, further visualisation was achieved by creating a pitch plotting function and a score tracker.

Overall, the simulation's performance was satisfactory as it succeeded in completing most of the proposed tasks. However, it was unable to achieve a fully functional simulation. From the team performance we found that the predicted time

7. Bibliography

Aldebaran, 2018. *Locomotion control.* [Online]
Available at: http://doc.aldebaran.com/2-5/naoqi/motion/control-walk.html
[Accessed 14 Apr 2023].

Aldebaran documentation (no date) NAO - Technical overview - Aldebaran 2.1.4.13 documentation. Available at: http://doc.aldebaran.com/2-1/family/robots/index_robots.html (Accessed: April 18, 2023).

Aleluya, E.R.M., Zamayla, A.D. and Tamula, S.L.M. (2018) 'Decision-making system of soccer-playing robots using finite state machine based on skill hierarchy and path planning through Bezier polynomials', Procedia Computer Science, 135, pp. 230–237. Available at: https://doi.org/10.1016/j.procs.2018.08.170.

Cristiano, J., Puig, D. & García, M. A., 2011. *On the maximum walking speed of NAO humanoid robots.* Albacete - Spain: XII Workshop of Physical Agents.

Daniel Stephen (2005) " Advanced modelling of soccer balls."

Generationrobots, 2018. *Programmable Humanoid Robot NAO V6.* [Online]
Available at: https://www.generationrobots.com/en/403100-programmable-humanoid-robot-nao-v6.html
[Accessed 14 Apr 2023].

Huang, H.-P. and Liang, C.-C. (2002) 'Strategy-based decision making of a soccer robot system using a real-time self-organizing fuzzy decision tree', Fuzzy Sets and Systems, 127(1), pp. 49–64. Available at: https://doi.org/10.1016/S0165-0114(01)00152-X.

Intelligent diagramming (no date) Lucidchart. Available at: https://www.lucidchart.com/pages/landing?utm_source=google&utm_medium=cpc&utm_campaign=_chart_en_tier1_mixed_search_brand_exact_&km_CPC_CampaignId=1490375427&km_CPC_AdGroupID=55688909257&km_CPC_Keyword=lucidchart&km_CPC_MatchType=e&km_CPC_ExtensionID=&km_CPC_Network=g&km_CPC_AdPosition=&km_CPC_Creative=354596043016&km_CPC_TargetID=kwd-33511936169&km_CPC_Country=1007336&km_CPC_Device=c&km_CPC_placement=&km_CPC_target=&gclid=Cj0KCQjwlumhBhClARIsABO6p-w3F-teAO1O4tlP88ua2RmCJ8Svx3iMKFO0CdiPSM4y-klgFTHyd8EaAsBFEALw_wcB (Accessed: April 18, 2023).

Li, Y., Song, Y. and Rezaeipanah, A. (2021) "Generation a shooting on the walking for soccer simulation 3D league using Q-Learning algorithm," Journal of Ambient Intelligence and Humanized Computing [Preprint]. Available at: https://doi.org/10.1007/s12652-021-03551-9.

Model Finite State Machines - MATLAB & Simulink - MathWorks United Kingdom (no date). Available at: https://uk.mathworks.com/help/stateflow/gs/finite-state-machines.html (Accessed: 18 April 2023).

Pratomo (2010) "Position and obstacle avoidance algorithm in Robot Soccer," Journal of Computer Science, 6(2), pp. 173–179. Available at: https://doi.org/10.3844/jcssp.2010.173.179.

RoboCup, 2023. *A Brief History of RoboCup.* [Online]
Available at: https://www.robocup.org/a_brief_history_of_robocup
[Accessed 13 Apr 2023].

RoboCup, n.d. *Objective.* [Online]
Available at: https://www.robocup.org/objective
[Accessed 12 Apr 2023].

Robopreneur, 2018. *NAO V6 The 6th Generation Humanoid Robot.* [Online]
Available at: https://www.robopreneur.com/nao-v6
[Accessed 14 Apr 2023].

Robots, H., 2018. *Robocup 2018 SPL Finals: Nao-Team HTWK vs. B-Human.* [Online]
Available at: https://www.youtube.com/watch?v=H8xc6LpiNVs
[Accessed 14 Apr 2023].

SSL.RoboCup, n.d. *RULES.* [Online]
Available at: https://ssl.robocup.org/rules/
[Accessed 13 Apr 2023].

Tefft, B.J. and Tefft, J.A. (2007) “Galilean relativity and the work-kinetic energy theorem,” *The Physics Teacher*, 45(4), pp. 218–220. Available at: https://doi.org/10.1119/1.2715417.

UMich-CURLY-teaching. (n.d.). UMich 500-Level Mobile Robotics Course [GitHub repository]. Umich-Curly-Teaching/Umich-Rob-530-Public. https://github.com/UMich-CURLY-teaching/UMich-ROB-530-public

What is a Decision Tree | IBM (no date). Available at: https://www.ibm.com/uk-en/topics/decision-trees (Accessed: 18 April 2023).

8. Appendix

Appendix A: Model representation

Appendix A.1 - Nao specs

Parameter	Value	Unit
Height of nao6 robot	0.58	m
Width of nao6 robot	0.31	m
Weight of nao6 robot	5.48	kg
Display Field of View	68.2	deg
Horizontal Field of View	57.2	deg
Vertical Field of View	44.3	deg
Maximum forward translation along X	0.080	m
Default forward translation along X	0.040	m
Minimum forward translation along X	0.010	m
Absolute maximum rotation around Z	0.524	radian
Absolute default rotation around Z	0.349	radian
Absolute minimum rotation around Z	0.001	radian
Maximum rotation (degrees)	maxStepTheta*360/(2*pi)	Deg
Default rotation (degrees)	defaultStepTheta*360/(2*pi)	Deg
Minimum rotation (degrees)	minStepTheta*360/(2*pi)	Deg
Maximum step duration	0.6	s
Default step duration	0.5	s

Minimum step duration	0.42	s
Maximum velocity	maxStepX / maxStepPeriod	m/s
Normal velocity	defaultStepX / defaultStepPeriod	m/s
Minimum velocity	minStepX / minStepPeriod	m/s
Maximum angular velocity	maxStepTheta_deg / maxStepPeriod	Deg/s
Default angular velocity	(defaultStepTheta_deg / defaultStepPeriod)(2pi/(360))	rad/s
Minimum angular velocity	minStepTheta_deg / minStepPeriod	Deg/s
Time taken to fall down	0.55	s
Time taken to get up after the fall	3.90	s
Total time from falling to getting up	t_fall + t_getUp	s

Table 5. Full Detailed Nao Specification

Appendix A.2 - Robocup specs

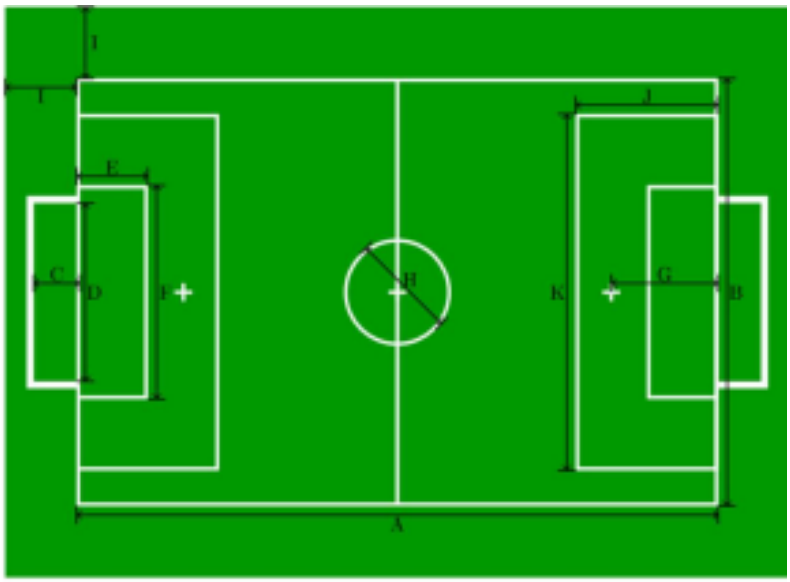


Figure 3: Pitch Markings

Figure 12.

Code	Description	Measurement
A	Pitch length	9m
B	Pitch width	6m
C	Goal depth	0.6m
D	Goal width	2.6m
	Goal height	1.2m
E	Goal area length	1m
F	Goal area width	3m
G	Penalty mark distance	1.5m
H	Centre circle diameter	1.5m
I	Border strip width	1m
J	Penalty area length	2m
K	Penalty area width	5m

Figure 13.

Appendix B: Path planning

Appendix B.1 - RRT

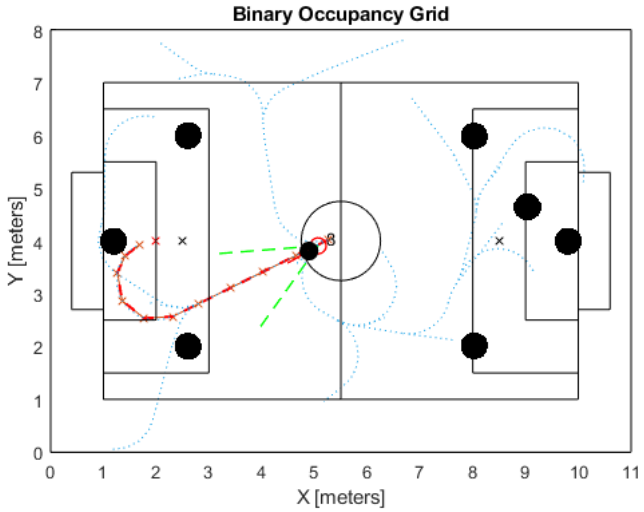


Figure 14.

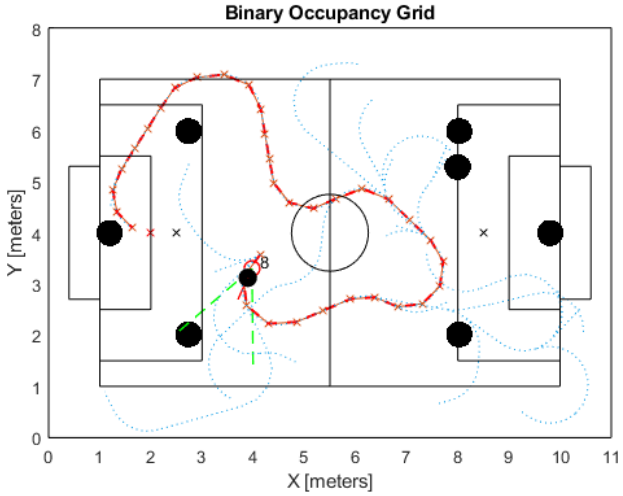


Figure 15.

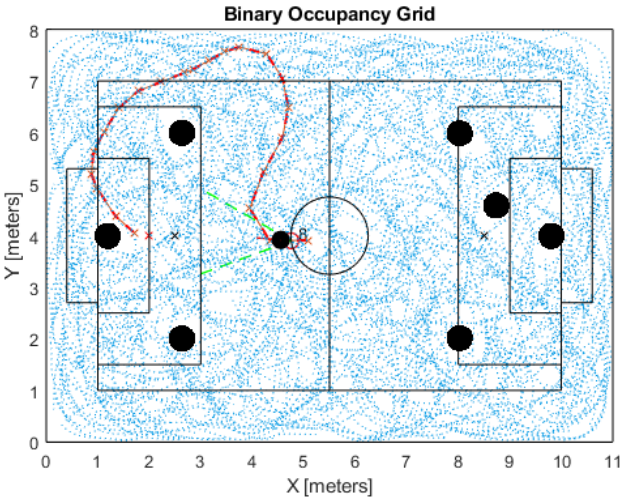


Figure 16.

Appendix B.2 - Interception

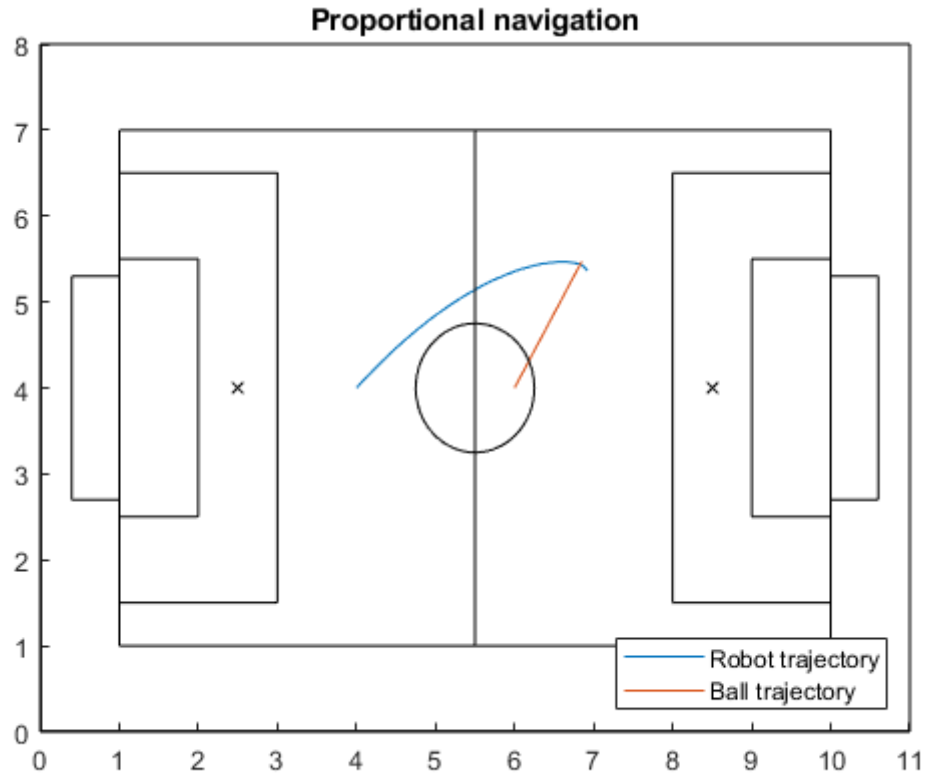


Figure 17.

Appendix C - Project management

Appendix C.1 - Project Time Schedule:

Table 6.

Task	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10	Week 11
Specification	✓	✓									
Requirements		✓									
Modelling		✓	✓								
Control and Guidance			✓	✓	✓	✓	✓	✓			
Algorithm Development			✓	✓	✓	✓	✓	✓			
Enviroment Development						✓	✓	✓	✓		

Operation								✓	✓		
Final Report								✓	✓	✓	✓

Table 7.

Appendix C.2 - Chart of team performance:

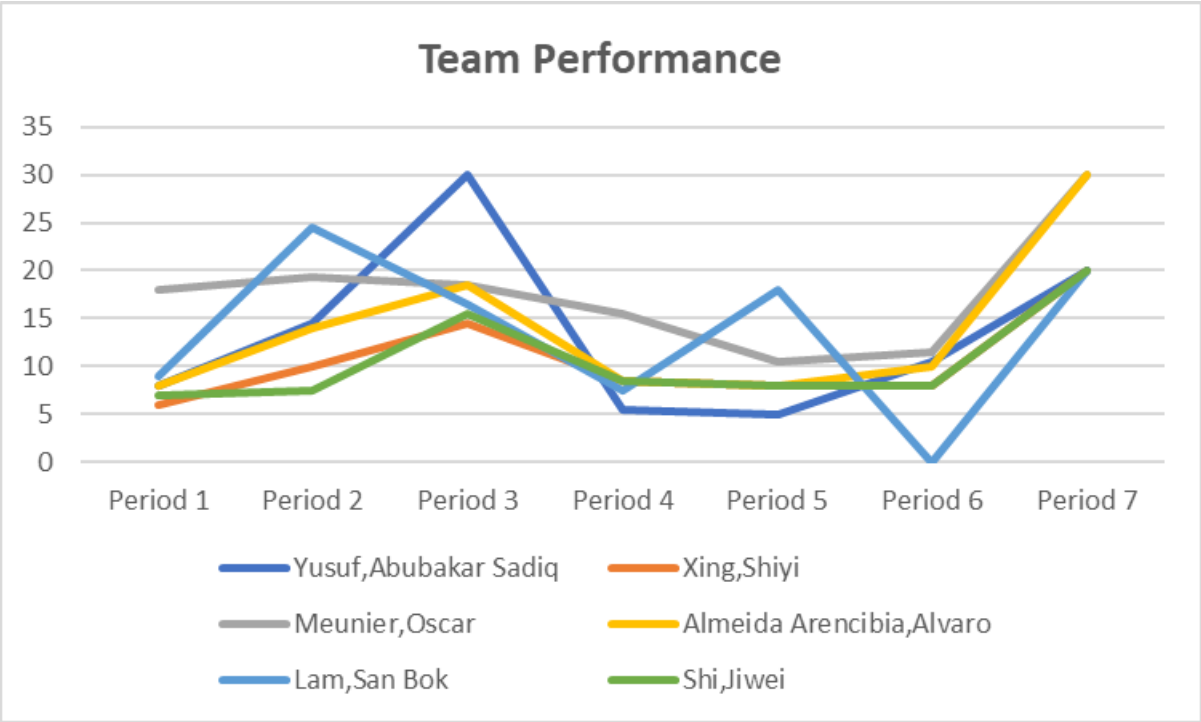


Figure 18.