Ball Interception in Humanoid Robot Soccer

Author

Saman Khan

2019-RIME-00000319685

Supervisor

Dr. Sara Ali

Co-Supervisor

Dr. Yasar Ayaz

Department of Robotics and Artificial Intelligence

School of Mechanical and Manufacturing Engineering (SMME)

National University of Sciences and Technology (NUST)

Islamabad, Pakistan

December , 2022

Ball Interception in Humanoid Robot Soccer

Author

Saman Khan

2019-RIME-00000319685

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science in Robotics and Intelligent Machine Engineering

Supervisor

Dr. Sara Ali

Thesis Supervisor's Signature: _____

Department of Robotics and Artificial Intelligence

School of Mechanical and Manufacturing Engineering (SMME)

National University of Sciences and Technology (NUST)

Islamabad, Pakistan

December, 2022

Declaration

I certify that this research work titled "*Ball Interception in Humanoid Robot Soccer"* is my own work. The work has not been presented elsewhere for assessment. The material that has been used from other sources is properly acknowledged / referred.

_____

Saman Khan

2019-RIME-00000319685

Plagiarism Certificate (Turnitin Report)

This thesis has been checked for Plagiarism. Turnitin report endorsed by Supervisor is attached.

_____

Saman Khan

2019-RIME-00000319685

_____

Signature of Supervisor

Copyright Statement

- Copyright in text of this thesis rests with the student author. Copies (by any process) either in full, or of extracts, may be made only in accordance with instructions given by the author and lodged in the Library of NUST School of Mechanical & Manufacturing Engineering (SMME). Details may be obtained by the Librarian. This page must form part of any such copies made. Further copies (by any process) may not be made without the permission (in writing) of the author.
- The ownership of any intellectual property rights which may be described in this thesis is vested in NUST School of Mechanical & Manufacturing Engineering, subject to any prior agreement to the contrary, and may not be made available for use by third parties without the written permission of the SMME, which will prescribe the terms and conditions of any such agreement.
- Further information on the conditions under which disclosures and exploitation may take place is available from the Library of NUST School of Mechanical & Manufacturing Engineering, Islamabad.

Acknowledgements

First and foremost, I would like to humbly state praises to Allah Almighty for His blessings and guidance throughout the research process and in general, towards every aspect of life. The faith and belief in His blessings kept me motivated to complete my research.

I would like to express my sincere gratitude to my research supervisor Dr. Sara Ali, Assistant Professor, Department of Robotics and Artificial Intelligence(R&AI), National University of Sciences and Technology (NUST) for believing in me and giving me the opportunity to work under her supervision. She remained very patient and polite and kept me motivated during the entire research phase.

I would also like to acknowledge the support and guidance of my co-supervisor Dr.Yasar Ayaz , Professor, Department of Robotics and Artificial Intelligence(R&AI), National University of Sciences and Technology (NUST) and Dr. Khawaja Fahad Iqbal, Assistant Professor, Department of Robotics and Artificial Intelligence(R&AI), National University of Sciences and Technology (NUST) for their assistance and guidance.

Special thanks to my friends and family who supported me and kept me motivated throughout the research journey. I would like express my gratitude to my seniors, colleagues and friends,Miss Rameesha Murtaza ,Miss Maham Ehsan, Mrs Urva Shahzad and Mr. Muhammad Imran for providing valuable advices and emotional support during my research program.

I would also like to convey special praises to open- source platforms (and people involved in them) including B-Human Repository for providing amazing learning opportunities to young and enthusiastic individuals.

Finally, I would like to express my gratitude to all the individuals who have rendered valuable assistance to my study directly or indirectly.

Dedicated to my exceptional parents and adored siblings whose tremendous support and cooperation led me to this wonderful accomplishment.

Abstract

Humanoid Robots are widely used in various real-world applications which contributed in raising the interest of researchers to enhance Robot development and deployment in dynamic environments. Robocup Soccer league is a platform for qualitative development in the fields of Robot Kinematics and Dynamics, Motion Planning and Control, Navigation, Computer Vision and Machine Learning. New challenges are introduced each year for teams that aim to improve robot behaviors in real world scenarios. One of the most popular leagues in Robocup is Standard Platform League that majorly focuses on improving individual robot capabilities, team capabilities and coordinated strategies in dynamic soccer matches played by teams of Aldebaran Nao humanoid robots. The robots with capability to quickly gain ball possession from opponent by intercepting the moving ball path will contribute to a better defense strategy. The aim of this study is to improve individual robot and team skills during game scenarios where ball is moving using motion primitives and geometric parameters. The proposed strategy is used to compute the interception point using geometric equations given the ball position and velocity information. The robot response is dependent on factors including initial position and velocities of robot and ball. Furthermore, to demonstrate the feasibility of proposed approach, different scenarios with single and multi-agents are simulated using SimRobot. Simulations validate the effectiveness of proposed algorithm in moving ball interception scenario.

Key Words: *Humanoid, Nao Robot, Geometric and motion primitives, Soccer Robotics, Ball Interception*

Table of Contents

List of Figures

List of Tables

List of Acronym

DOF – Degree of freedom

SPL – Standard Platform League

CHAPTER 1: INTRODUCTION

Continuous evolution of humanoid robot platforms[1] in terms of better hardware and advanced software architecture have encouraged researchers to implement diversified techniques enabling better response of humanoids in dynamic environment. Behavior of robot in physical world adds complexity due to various factors including limited and error-prone perceptions. Along with this, the multi-agent systems add another dimension to the complexity as the robots need to collaborate towards achieving goal. Robocup, the International Robotics Competition, provides a platform promoting research in field of robotics.

1.1 RoboCup

Professor Alan Mackworth (University of British Columbia, Canada) floated the idea of soccer gameplay between multiple robots in 1992. Soon after this, J-league, a robotics soccer competition was organized in Japan. J-league gained international recognition and resulted in formation of Robocup. Robocup platform encourages researchers to carry out inter-disciplinary research in field of computer vision, motion planning, dynamic behaviors etc. Every year, new challenges and themes are introduced that highlights future research avenues. Robocup is a great platform for experimenting with new state of the art schemes and the application in real-world scenario aids to the progress of fully autonomous robots. Robocup started the soccer domain with three leagues and then included Junior, @Home, Rescue, and Industry leagues. Each league features a different platform with its own challenges (Figure 1). The aim behind is to tackle problems in different domains with varied approaches and tactics. The Robocup federation's challenging mission is that by 2050, a team of fully autonomous robots shall win soccer game against reigning human soccer world champions. The federation announces new challenges every year with the aim of bringing the performance of humanoid robots closer to real players.

Figure 1: Robocup Competitions with different robotic platforms[2]

1.2 Standard Platform League

Robocup SPL is one of the leagues of the competition in which a team of humanoid robots plays soccer against another team. The SPL League has given rise to the development of numerous techniques, including motion planning, role- assignment, navigation, adversarial strategies etc.

Both teams use the same hardware platform, NAO Robot from Aldebaran Robotics. Two teams consisting of 5 robots each play soccer match against each other (Figure 2) to compete for the title of world champions. With use of standard hardware platform, both teams possess same sensors and processing capabilities. The differentiator is majorly the software design. The team tactics, including high and low-level strategies differentiates the performance of team. In a soccer match, the dynamic environment is major source of added complexity. The robot communicates with outside world through a Game controller. The game controller is a computer that is used to control the state of the game.

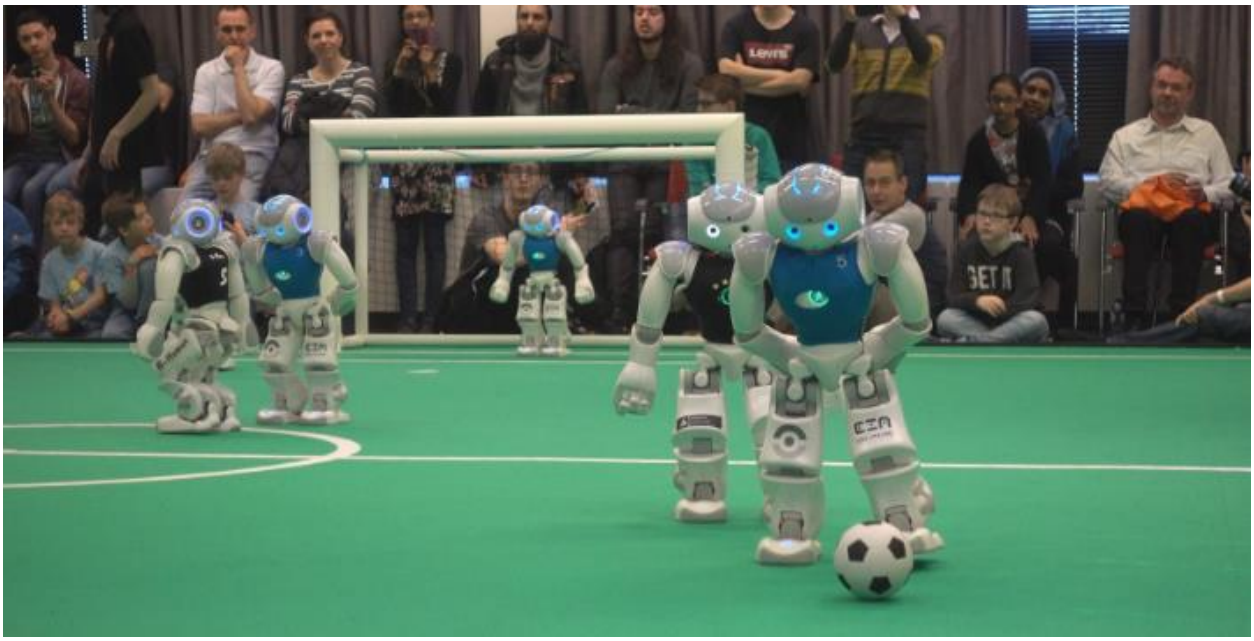


Figure 2 : SPL Soccer Match played by NAO Humanoid Robots

1.3 Problem Statement

During games, the strategies that enable team players to gain the ball possession quicky are of great importance as a quick and long ball possession among own teammates increase the probability of scoring a goal. Other than ball possession, strategies to block other players and passing to teammates are also useful during gameplay scenarios

This thesis focuses on the approach of evaluating the possibility of quick response of humanoid in a dynamic environment particularly during a game play. The problem to gain possession of moving ball in SPL league is particularly complex due to multiple reasons. The response of humanoids is relatively slow as compared to wheeled robots. Also, the assumption to intercept a ball is generally taken to be that the intercepting robot should have a speed faster than the speed of ball to gain possession. As humanoids, have slow walking speeds as compared to ball, tracking the ball is not a good idea. This is also because of the presence of multiple obstacles including own and opponent team players during a match. A humanoid also cannot make quick turns to change its orientation to approach the ball. Hence, the ball interception behavior for humanoid has multiple different constraints as compared to a wheeled platform. A humanoid cannot have changing accelerations and velocities like a wheeled platform. Generally, humanoids walk at a constant speed.

1.4 Objectives

The study aims to design a response behavior of humanoid for moving ball interception with the primary aim of achieving better response of humanoid in scenarios where ball is in motion and to present the application of proposed architecture in the domain of robot soccer. Furthermore, it provides the implementation of proposed strategy in framework of B-Human Team. The strategy is tested in simulation environment using SimRobot Simulator Platform. Team B-Human release their base code as part of their GitHub repository. The code release main aim is to encourage new researchers to work on high-level behavior and strategies. Geometric approach is used for calculating a feasible interception point and possibility to reach the interception point based on position and velocity parameters of ball and robot are evaluated.

Finally, the thesis demonstrates the effectiveness of proposed strategy in various scenarios. (a) A single robot in simulation environment with ball movement simulated by striker striking ball towards goal. (b) Simulation scenario with two active robots from own team and a striker from opponent team. A different approach is used where robot nearest to ball end position attempts to stop the ball. Results are compared with case (a) to demonstrate the effectiveness of proposed strategy. (c) Simulation scenario with robots as agents, goalkeeper and striker. The behavior selects the most suitable candidate for interception in multi-player scenario. Besides these, the functionality to apply forces to ball center of mass in simulation environment was added to study the robot response in case of varying ball speeds.

1.4 Thesis Outline

Chapter 2 consist of in-depth review about the techniques and methods used to solve the problem of moving ball interception. The review relates the previous work in the domain to the proposed approach. It also discusses the parameters that contribute towards the estimation of interception point. It highlights the geometric techniques and laws of physics that are involved.
Chapter 3 provides insight about the code architecture and simulation environment that used to evaluate the effectiveness of the proposed strategy. Chapter 4 discuss the methodology of single

and multi-agent behavior using flowcharts. Chapter 5 includes the results from different simulated scenarios followed by conclusion and future works in Chapter 6.

CHAPTER 2: Literature Review

This chapter aims to provide a detailed overview about moving ball interception techniques used in literature. While the ability of a robot to intercept a moving ball is highly advantageous during gameplay, only a few works can be found in literature that discuss this topic for the case of humanoids.

Gathering information about moving target is extremely important. In literature, variety of methods can be used to not only estimate the current position of moving object but also to predict the future locations. The authors of [3] use Kalman filters for trajectory prediction of falling ball. The use case is the robot trying to catch the ball. With the increasing use of neural networks these days, [4] presented LSTM based model for trajectory prediction of vehicles on highway. Kalman filter prediction is dependent on system state equation while neural network-based techniques require huge amount of data. The work[5] presented a model combined with k-nearest neighbor method for prediction based on previously observed data and auto-regression for carrying out prediction based on current ball path. Two different schemes were introduced mainly fixed and adaptive scheme highlighting the advantages of adaptive scheme in cases where external disturbances change the robot path.

The study [6] provides methods for predicting ball trajectory on ground. Online methods include double exponential smoothing (DES), autoregressive (AR) and quadratic prediction (QP) while self-perturbing recursive least squares (SPRLS) is offline method. Results showed poor performance of pre-defined models for cases where the friction deviates from presumed model. Performance of offline methods is promising but pre-training is required.

Ball interception is widely focused for the goalkeepers and many research works demonstrate various algorithms for goalkeeper strategy[7]. However, having the field players with ability to intercept moving balls can play a vital role in team defense strategies.

The following sub-sections will discuss literature relevant to the research.

2.1 Moving ball handling using mobile robots

The methods used for the task of moving ball interception depends on the environment and robot kinematic model. The degree of freedom also plays a major part in the decision. Moving ball interception using mobile robots is extensively discussed in literature. Interception is defined as the ability of approaching a moving object for collision. Interception is generally required in defensive scenarios where defending robots having the ability to intercept moving targets bring advantage to team behavior.

The study [8] focuses on problem of ball interception of simulated soccer agents in Robocup. The authors compare the performance of numerical methods, methods based on reinforcement learning and qualitative velocity approaches. The results showed the usefulness of numerical methods, but it also highlights its limitation as the knowledge of physical model describing movements is needed. The paper also highlights the advantage of qualitative interception method due to its robustness and portability. All methods have similar success ratio.

In [9], the authors focused on simulating dynamic object interception problem using wheeled mobile robot. Expected position of moving target is computed using polynomial fitting using observations of last 15 seconds. The waypoints of the path of wheeled mobile robot are found using A* search and optimized using gradient descent. The speed curve is formulated into piece wise Bezier curves optimization on ST graph. However, the polynomial fitting requires observation of long time before predicting the future location of moving object. Also, the kinematics of wheeled robot are mainly focused which are very different from the behavior of humanoid robot.

In [10], simulations were carried out for a wheeled mobile robot platform where a wheeled robot goalkeeper intercepts the moving ball using parallel navigation guidance law. In parallel navigation guidance law, the robot moves to maintain the line of sight parallel to the initial line of sight between ball and robot. The goalkeeper is moved for controlled linear velocity or controlled orientation angle.

In [11], simulations and experiments were carried out for non-holonomic wheeled mobile platform. The proposed interception method had a two-level structure, one for camera mounted on pan-tilt platform to keep the target near center of image plane. On higher level, control laws were proposed for trajectory planning of wheeled robot for interception. The approach does not include

prediction of ball movement and control law only use positions and velocities from visual data for trajectory planning.

The paper [12] discussed mainly the pass strategy where the receiving robot intercepts the ball and catches the pass. However, the study assumes that ball is coming directly towards the receiver and the receiver positions itself to catch it without the ball bouncing back.

In ,[13]the authors described two types of interception for wheeled robot. In passive interception, the robot reaches to shortest distance and wait for ball while in active interception, the robot calculates the point online of ball trajectory where the robot can reach before the ball.

A modified proportional navigation-based method [14]was introduced focusing on the interception velocity between the robot and target for smooth grasping. Experiments were carried out in simulations and for wheeled robot platform Qbot-2. The wheeled platform has a variable velocity. Simulations and experimentation showed promising results for cases including fixed goal, constant velocity goad and maneuvering goal situations

In [15], the problem of moving target interception is solved by presenting a moving target interception algorithm that is based on the new ant algorithm. Straight line interception points are calculated based on the current trajectory of robot and moving target. By monitoring the target trajectory, the intercept point is re-calculated and the robot re-plans the path. It is assumed that velocity of robot is greater than velocity of moving target to ensure successful interception attempts.

2.2 Moving Ball handling using Humanoid Robots

Moving ball interception using humanoids is a difficult task. This is mainly due to a slow response of humanoid as compared to ball velocity. However, techniques could be developed to get maximum benefits of interceptions in scenarios where it is possible. Currently, for majority of the teams, only a goalkeeper has the ability to walk towards a ball reaching its goal or to dive to save a goal scoring attempt of opponents. Other field players use a passive approach.

The work [16]highlights the advantage of using teammate observations for time-critical tasks such as interception of moving ball. Two stationary robots with overlapping field of view compute their relative transformation for translating team-mate raw sensor information according to the robot's field of view.

As part of technical challenge of Robocup, researchers are motivated to successfully carry out tasks such as push recovery, high jump, high kick etc. Technical challenges of 2017[17] include the task of direct pass from a moving ball. In this task, a ball is rolled towards a robot. The robot detects position, velocity and acceleration and attempts to kick the ball towards the goal. Authors of [18] participated in the challenge and claimed 80% success rate using developed T-Flow robot. The challenge, however, assumes the fact that the ball will move towards the robot and the robot will try intercepting it, for passing it to the target just by moving its foot while standing.

In BHuman team behavior, the playing ball role is assigned to the robot nearest to the ball. In cases of moving ball, a role is assigned to the robot based on expected end position of ball[19].

The paper[20] use a hybrid method that combines Heuristically Generating Possible Actions algorithm (HGPA) and Q-learning to score a moving ball. The paper is focused on attacker behavior in presence of five defenders. The HGPA will generate a set of possible actions and the learning algorithm will make the best decision according to chance of scoring.

The work in [21] demonstrated reactive full-body behavior of humanoid robot player to block incoming ball. However, the study assumes that ball is moving towards robot and the robot decides the key-frame based motion to block the incoming ball.

In simulation leagues, the team UT Austin Villa[22] assigns a play ball role to the player closest to the ball and formations of other players is decided based on ball offset of players on field using Delaunay triangulation. Neural networks are used to decide where to kick the ball and for a moving ball, the players move towards anticipated kick destination.

The paper[23] also worked in the context of simulation league and presented their interception strategy. Their interception strategy depends on rating of success probability. The success probability considers parameters such as ball average speed, player average speed and distance of players from ball. It also considers the angle between ball and goal. If the results show that opponent will reach before the robot, the interception point is decided based on opponent speed and direction. This has one major drawback of assuming that opponent will hit the ball in its

heading direction only. However, the opponent robot can plan a kick in any arbitrary direction and the assumption might lead to wrong estimate of interception point. Also, in cases, where robot can reach quickly than the opponent, the robot moves to ball position. One thing to highlight in this approach is that in both cases, the interception is considered in case stationary ball. However, out presented approach focuses on interception of moving ball.

The experiments performed on simulation environment for soccer players. The framework was developed on UT Austin Villa base code. Artificial neural network is used. The ANN takes world state as input and outputs the next state. For interception of moving target, only the point perpendicular to the robot's trajectory is assumed. However, in our scenarios, we not only consider the shortest path but also the heading-based [24]intercept point if feasible. Also, the details about validity of chosen intercept point are highlighted.

2.3 Nao Humanoid Robot

2.3.1 Hardware Description

Nao Robot (Figure 3) is developed by Aldebaran Robotics. The robot is categorized as a medium-sized robot with a height of 58cm. The robot weighs around 5kg. The latest version of this bi-ped humanoid robot is named as NAO V6. It carries an on-board ATOM Quad core processor at 1.91 GHz, 4GB DDR3 RAM and 32 GB SSD. The system runs on embedded Linux distribution. The powerful processor enables the used of neural networks for predicting world states using image processing. It is also equipped with ethernet port and Wi-Fi 802.11b/g for communication purposes. It is also equipped with Li-Ion rechargeable battery that provides a continuous operation of nearly 90 minutes.

The 25 DOF complex robot consist of 5 independent kinematic chains. Left and Right arm chains with 5 joints each, a head chain with 2 joints, left and right leg chains with 5 independent joints each and a pelvis joint connecting both legs. The robot includes 7 touch sensors on arms, legs and head. The robot has audio and inertia sensors to understand the surroundings. Also, the robot has the ability to recognize speech. More sensors include 2 cameras for detecting shapes, objects and people in environment. The cameras provide lower and distant frontal view. Camera images have resolution of 640*480 and a frame rate of 30 frames per second. The robot is equipped with a 3-axisaccelerometer and a 2-axis gyroscope in the torso to provide real-time

information about the acceleration and orientation of the base link. Two bumpers located at the front of each foot are simple switches and can provide information about collisions of the feet with obstacles. Lastly, to deliver the feedback of forces applied on each foot, four force sensors are provided on each foot.

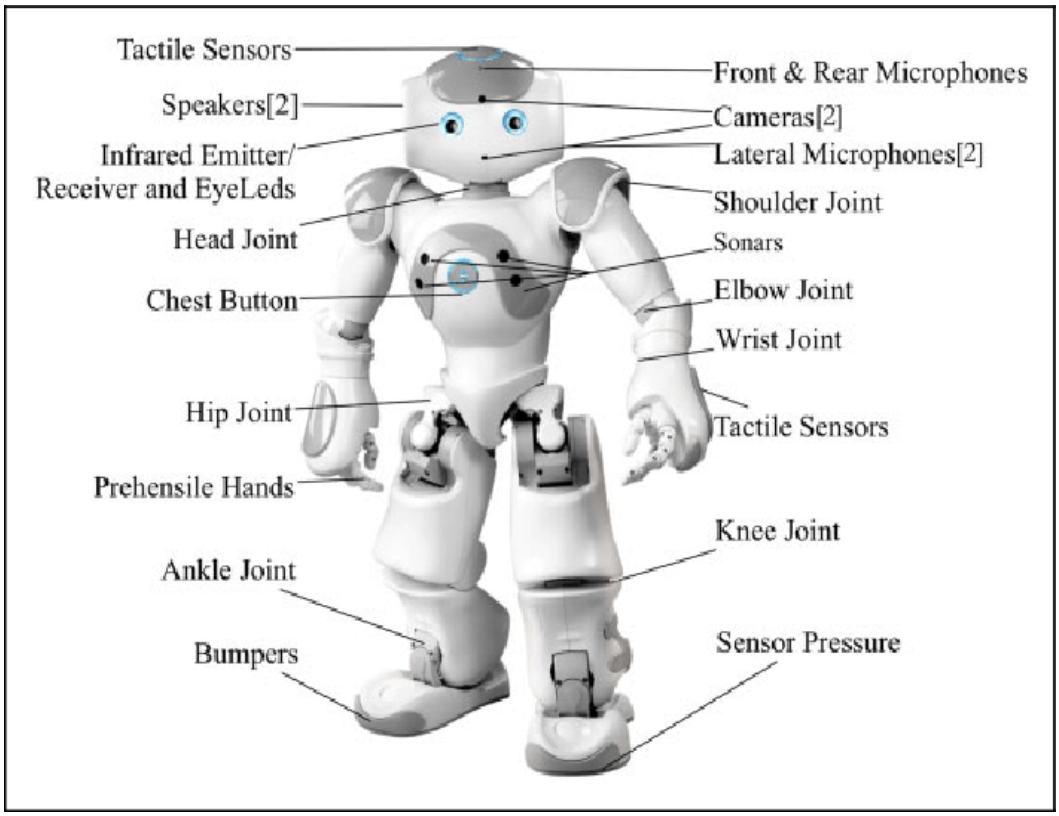


Figure 3 : NAO Humanoid with Joints and Sensors

2.4 Basics and Mathematical Definitions

In this section, details about all the mathematical relations used for the calculation of interception points are discussed. Also, the relationship between velocity and friction for rolling ball are presented.

2.4.1 Geometric Definitions and Relations

Shortest distance of point from a line can be computed using multiple mathematical relation. The elementary methods include using algebraic relation, vectors, areas of polygon and methods based

on coordinate geometry. The optimization based methods use calculus based derivations and constrained optimization methods for solving the shortest distance from a point to a line[25]. In this chapter, details about all the mathematical relations used for the calculation of interception points are discussed. Also, the relationship between velocity and friction for rolling ball are presented.

Shortest distance of point from a line can be computed using multiple mathematical relation. The elementary methods include using algebraic relation, vectors, areas of polygon and methods based on coordinate geometry. The optimization-based methods use calculus based derivations and constrained optimization methods for solving the shortest distance from a point to a line.[25]

The algebraic method, implemented in the code is discussed below. The different entities relating to this problem are included as part of Figure 4.

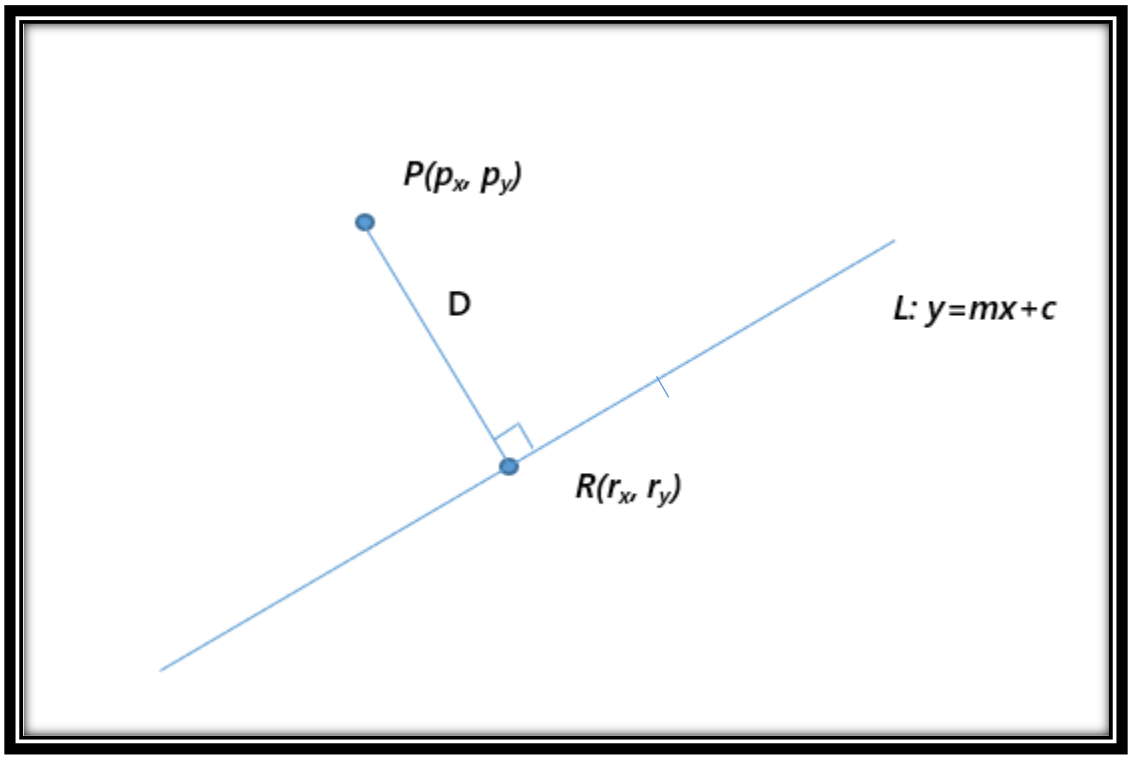


Figure 4 : Diagram for Line and Point Relations

Given point: $P(p_x,p_y)$

Equation of line: $y = mx + c$

Point of shortest distance: $R(r_x,r_y)$

2.4.1.1 Shortest Distance of Point from Line

Perpendicular drawn from a point to the line is the shortest distance. Given, the slope of line as m, the slope of line perpendicular to it would be -1/m. Hence, the equation of line passing through point P and intersection point R is given using slope-point from as

$$(y - py) = -\frac{1}{m}\ (x - px) \qquad (1)$$

Now, we have two equations. Each represents a line. The intersection point R is the point of intersection of line L and line PR. Solving the equation of these two lines simultaneously gives us the co-coordinates of point R.

$$r_x = \frac{mp_y + p_x - mc}{1+m^2} \qquad (2)$$

$$r_y = \frac{m^2 * p_y + m * p_x + c}{1+m^2} \qquad (3)$$

Using the distance formula, the distance D=l(PR) is given as

$$D = \sqrt{(r_x - p_x)^2 + (r_y - p_y)^2} \qquad (4)$$

2.4.1.2 Intersection Point between two lines

Given two lines

$$L1 : a_1 x_1 + b_1 y_1 + c_1 = 0 \quad (5)$$

$$L2 : a_2 x_2 + b_2 y_2 + c_2 = 0 \quad (6)$$

The co-ordinates of point of intersection in 2 dimensional field can be given using relation below.(x_c, y_c) is the intersection point of two lines

$$x_c = \frac{b_1 c_2 - b_2 c_1}{a_1 b_2 - a_2 b_1} \quad (7)$$

$$y_c = \frac{c_1 a_2 - c_2 a_1}{a_1 b_2 - a_2 b_1} \quad (8)$$

2.5 Ball Physics

Ball is the main object in a soccer play and to gather information about its position, velocity, and the time it takes to reach the point is important. Using this information, important decisions about individual player and team behaviors are made. In this section, description about formulas used for computations involving motion of rolling ball are stated. For simulation, a linear model for ball deceleration is assumed. Mainly, the second equation of motion is used. The relation is stated below.

$$S = v * t + \frac{1}{2} a t^2 \quad (9)$$

Where S is the rolled distance, v is the ball velocity and a is the ball friction

2.5.1 Ball Position After Time t

To calculate the position of ball after time, following equations are used.

$$p_x = p_{x0} + v_x * t + \frac{1}{2} b_{friction} * t^2 \quad (10)$$

$$p_y = p_{y0} + v_y * t + \frac{1}{2} b_{friction} * t^2 \quad (11)$$

where t is the time in seconds, $P(p_x, p_y)$ is the ball current position , V(v_x, v_y) is ball velocity in *m/sec* and $b_{friction}$ is the ball friction in *m/sec^2*

2.5.2 Ball Velocity After Time t

To calculate the velocity of ball after time, following equations are used.

$$v_x = v_{x0} + b_{friction} * t \quad (12)$$

$$v_y = v_y + b_{friction} * t \quad (13)$$

$b_{friction}$ is the ball friction in *m/sec^2*. It is basically the deceleration term for rolling ball.

2.5.3 Ball End Position

To compute end position in field coordinates where the rolling ball is expected to stop rolling, following equations are used.

$$t_{stop} = V_{total} * (-1)/b_{friction} \quad (14)$$

where t_{stop} is the time taken for the ball to stop. $b_{friction}$ is the ball friction and V_{total} is computed using x and y components of velocity as

$$V_{total} = \sqrt{v_x^2 + v_y^2} \quad (15)$$

Now, using t_{stop} , ball current position *p*, ball velocity *V* and ball friction $b_{friction}$, the ball position is computed using equations(10) and (11).

2.5.4 Time to pass distance

The time required for the ball to pass a certain distance is given as

$$t = \sqrt{{V_b}^2 + \frac{2*S}{b_{friction}}} \quad (16)$$

Where S is the ball distance and V_b is given as

$$V_b = \frac{V_{total}}{b_{friction}} \quad (17)$$

2.5.5 Velocity to kick ball to certain distance

The velocity needed to kick the ball can be computed using first equation of motion. The relation representing first equation of motion is stated as

$$2\ a\ S = V_{final}^2 - V_{initial}^2 \quad (18)$$

As the initial velocity is zero, the final velocity to kick a ball to a certain distance S is given as

$$V_{final} = \sqrt{2\ a\ S} \quad (19)$$

2.5.6 Ball velocity after specific time for moving from one point to another

Let's say, the ball starts at position P_o and reaches position P_1, the time taken to by ball to reach from point P_o to P_1 is given as t_{delta} .

The current velocity is calculated as

$$V_{current} = \frac{S_{total}}{t_{delta}} + \frac{1}{2} a t_{delta} \quad (20)$$

Where S_{total} is given as

$$S_{total} = \sqrt{(p_{1x} - p_{0x})^2 + (p_{1y} - p_{oy})^2} \quad (21)$$

2.5.7 Time until ball stops

To calculate the time until ball stops, we use the following relation

$$t = \frac{V}{a} \qquad (22)$$

Where V is the ball velocity and a is the ball acceleration (i.e., friction in case of rolling ball)

CHAPTER 3: Simulation Environment

This chapter aims to discuss the simulation environment used for carrying out research related to ball interception task. Many teams participating in Robocup release their code base on GitHub to encourage teams from around the world to work on various problems and challenges relating to robot soccer. Codes for simulation-based competitions and Standard Platform League are available for building on their architecture on framework instead of re-inventing the wheel. The platform chosen belongs to B-Human Team. The main reason is the fact that the code is easily deployable for testing on real robots. Also, the team has active community that guides the new researchers to carry out the research. Team NUST was working on their code; however, the developed code was using Vrep as Simulator platform. Also, the code was mainly developed for Ubuntu 16.0. Due to continuous improvements and new launch of Ubuntu, working on team-NUST code seems difficult. So, a better approach would be towards understanding of a mature framework and implement special behaviors and advanced techniques to architecture and begin research in soccer gameplay scenarios.

3.1 About Team B-Human

Team B-Human is a project of Department of Computer Science (University of Bremen) and Department of Cyber-Physical System (DFKI). Standard Platform league mainly focuses on the software developments. This is mainly because the hardware platform is same for all teams. Team B-Human has participated in German Open, European Open and in Robocup World championships since 2008.

3.2 B-Human Code Architecture

The architecture is based on framework of German Team 2007[26]. The main features of this architecture include binding, threads, representations and modules, communication and debugging support. The binding is related to hardware NAO while other features play a major part in software architecture. Following few sub-sections aim to highlight the details of features of architecture.

3.2.1 Threads

The architecture uses threads; however the number of threads depend upon the external requirements of robot hardware or operating system. The B-Human framework[27] currently uses five threads named Upper, Lower, Cognition, Motion and Debug. The Upper and Lower threads are for the two cameras of NAO robot. These threads run at frequency of 30Hz. They are also called as perception threads and they receive camera images from Video for Linux. They get data about world model from Cognition thread. They also get sensor information from motion thread. The images are processed, and results are sent to cognition thread. The cognition thread uses the gathered information along with sensors data from motion thread to model the world. It runs at a frequency of 60 Hz. Cognition thread sends high-level motion commands to thread Motion. The motion thread is responsible for generating target angles for 25 joints of NAO robot. The Motion thread runs at a framerate of 83Hz. The Debug thread communicates with host PC. While this thread remains in-active during actual games, in other cases, it is useful for sharing data between other threads and host PC. The threads used in NAO are shown in figure below.

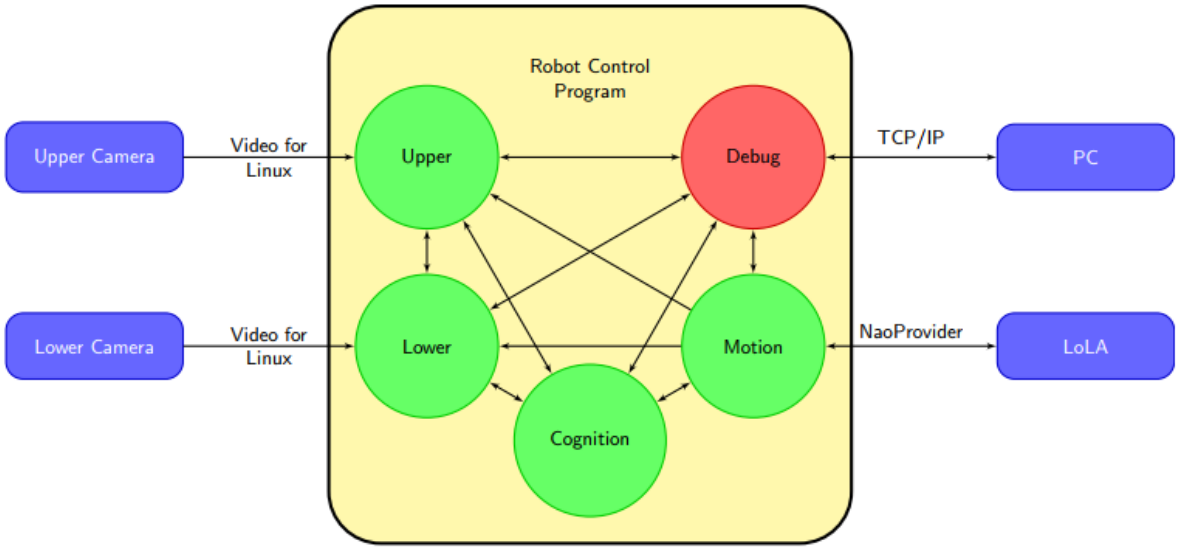


Figure 5 : BHuman Architecture[27]

3.2.2 Representations and Modules

In a rea-world scenario, a robot needs to calculate multiple factors for localization, image-procession, motion etc. For better handling of huge calculations, the code is divided into a system that uses modules and representations. The modules framework used was introduced in[26] .

Representations can be considered as structures or classes that contain information. Modules mainly aims to update representations. The blackboard is a central storage for information. Every thread has its own instance of blackboard. Inter-thread communication is used extensively as a module generally depends on multiple representations to calculate a new representation of some other type. The detail that includes which module provide which representation is found in configuration file thread.cfg. This, however, does not specify the sequences in which the representations are updated.

3.2.3 Communication

The framework uses 4 types of communications including Inter-thread communication, message queues debug communication and team communication. ModuleGraphCreator decides the direction of communication for shared representations. Inter-thread communications are triple buffered so that threads never block each other. Communication threads use message queues to store and transfer data as sequence of messages. Debug communication manages communication between robot control program and other PC tools. Team communication enables the robots to share important information among themselves to have a better view of world model.

3.2.4 Debugging

The frameworks' strong debugging support include variety of features. Debug requests are used to enable and disable parts of code. Debug images are used for visualization of image processing data. 2D and 3D drawings can be drawn in 2D and 3D views respectively using debug drawings and 3D debug drawings. Lines, circles, dots and other complex drawings can be made with the help of debug drawings. Modify macro allows for visualizing the data from streamable classes in data views.

3.3 SimRobot

SimRobot simulator[28], [29] models realistic forces including gravity and friction on objects present inside a 3-dimensional simulated environment. SimRobot uses Open Dynamics Engine (ODE)[30] that is a popular implementation of rigid body dynamics for robotics simulation applications. Using simulators as a pre-requisite for development and testing of hardware

platforms is highly beneficial due to multiple advantages including low installation and operational costs. Also, simulators simplify assumptions about the real-world making the proof of concept easy.

In SimRobot environment, each part of robot body is modelled as a rigid body with mass. All parts are connected with each other using joints. The ODE computes the transition dynamics of system by considering forces applied. It also considers the forces of gravity, friction, and collision. Sensors can also be included as part of robot model. The information from various sensors including cameras and gyroscopes will be gathered and processed to get meaningful information about the world state. The information will then be used to decide the robot behavior according to current world state.

3.4 Force Application on objects of simulation environment

In simulation environment, different positions of robots and ball in the field represent different phase of game. For simulating a scenario, it is useful that user have provision to change positions of objects conveniently. When testing B-Human code using SimRobot, following commands are used in console for moving objects.

- mvb x y z

  To move ball to a position. X and Y position in field coordinates in millimeters. Z is set to 50 mm for positioning of ball on field
- mb RoboCup.robots.robot3 x y x angx angy angz

  To move robot to a position. X and Y position in field coordinates in millimeters. Z is set to 330mm. angz is rotation in degrees about z-axis for setting orientation of robot.

The available kick motion in simulation, can apply fixed force on ball. However, we wished to simulate a rolling ball in environment. Also, the freedom to experiment with varying ball speeds was also needed. Hence, I incorporated the functionality that can be used to apply force to ball's center of mass in simulation environment.

To do this, a pure virtual function is added to Body interface defined in SimRobotCore2.h. The file can be found in folder BHumanCodeRelease/Util/SimRobot/Src/SimRobotCore2 .Afterwards, the function is implemented in class Simulation/Body (.cpp and header file). The function uses call to ODE functionality dBodyAddForce. The force is applied at center of mass. Changes were

made in the file ConsoleRobocupCtrl.cpp so that force can be applied to ball using SimRobot console command. The code changes in each file are attached as part of Appendix.

3.5 Code Architecture changes in Cognition Module

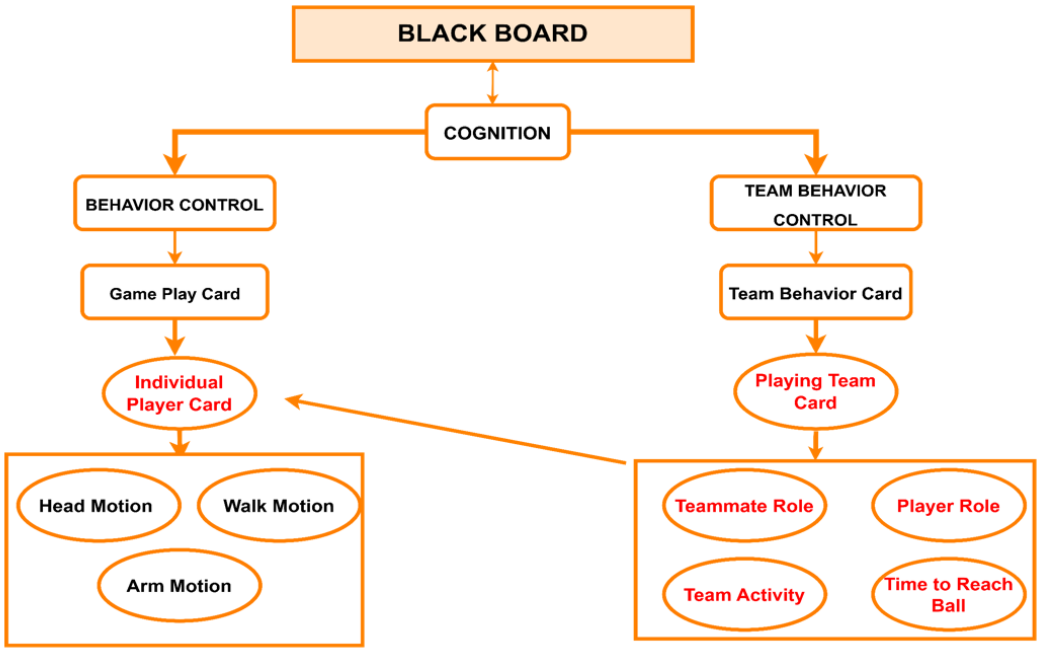


Figure 6: Code Architecture Update

CHAPTER 4: Methodology

4.1 Calculation of Interception Point for moving ball

To calculate the interception point, team ball model is used. It is important for the team ball model to be valid. For purpose of simulation, we have used the parameters of ball that are directly being communicated from the simulator.

Also, the robot pose at current instant of time is also taken using simulator data. For a moving ball, we calculate the ball end position using relations from Sec 2.5. Using ball's starting and end position, equation of line is found. Afterwards the formulas from Sec 2.4 are used to compute interception point.

4.2 Validity of Interception

Calculation of interception point in field co-ordinates is done using geometrical relations described in Section 2.4. However, the validity of interception point is extremely important. As, we are using equations of line to compute the shortest possible intercept point. For cases where the shortest intercept point is at a distance greater than 1m, a heading-based intercept point is computed as the current walk engine footstep planning works in such a way that the robot turns and then moves to a point if distance from current robot location to the point is greater than 1m.

Following factors need to be considered for validity of interception point

- The interception point should lie between the ball starting and end position. If a calculated interception point doesn't lie between ball starting and end position, it is considered invalid.

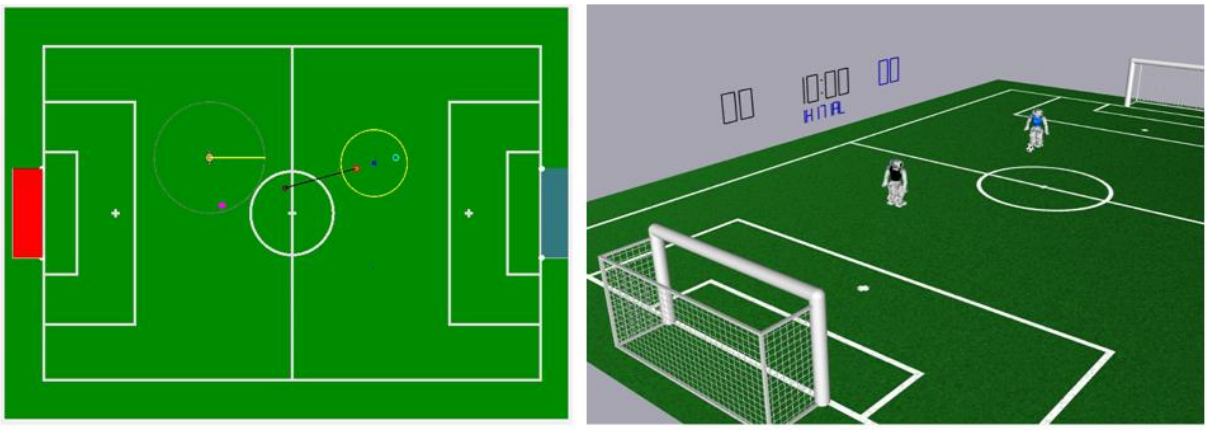


Figure 7:Invalid Interception for Shortest Intercept Point

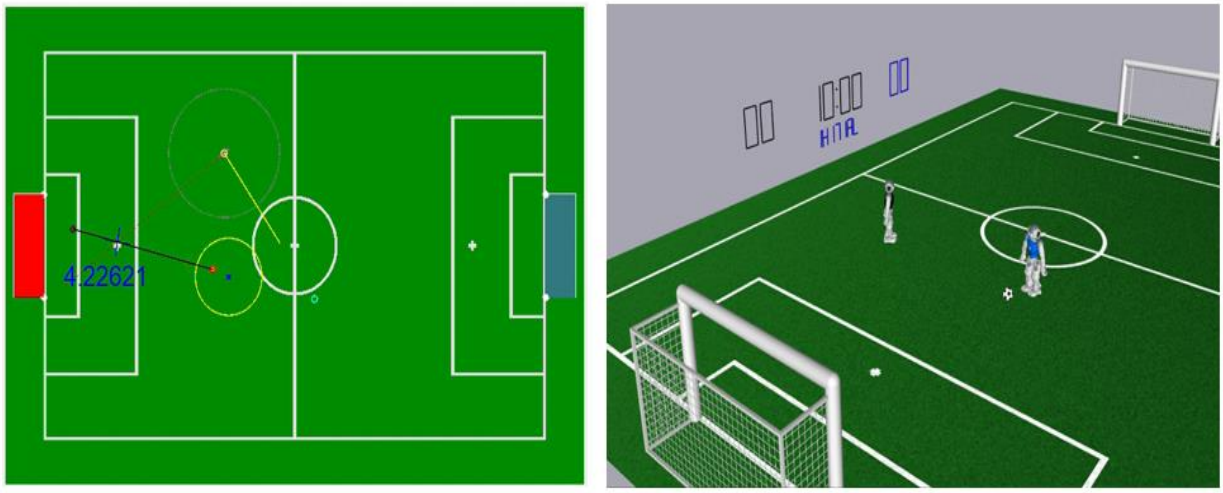


Figure 8 : Invalid Interception for Heading Based Intercept Point

- For heading based interception point, the calculated interception point should lie within the projected field of view of robot. A calculated interception point behind a robot is invalid. This is also explained in figure below.

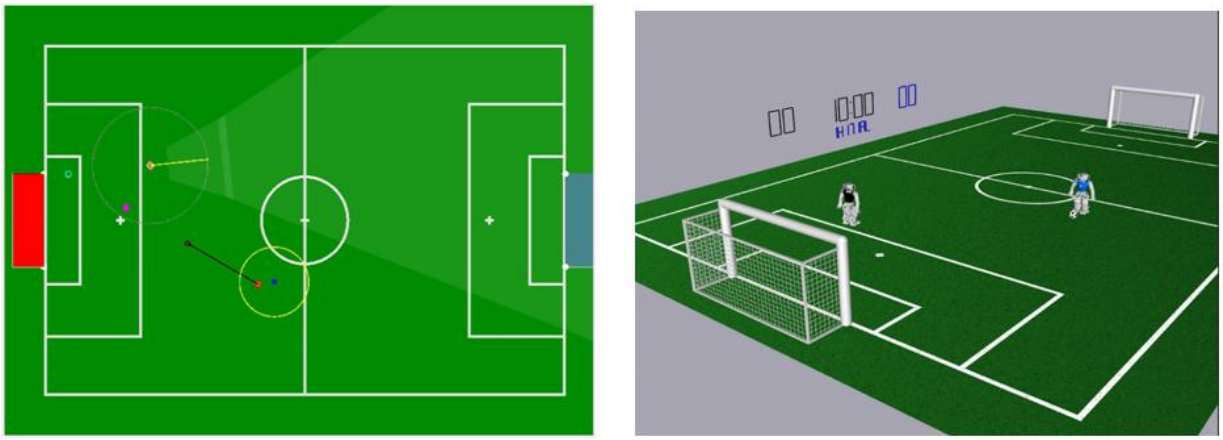


Figure 9: Interception Point behind robot field of view

- The robot time to reach the interception point should be less than the ball time to reach interception point. If the robot cannot reach this point before the ball, interception remains invalid.

All the above constraints are needed to be considered before declaring the validity of interception point.

4.3 Methodology

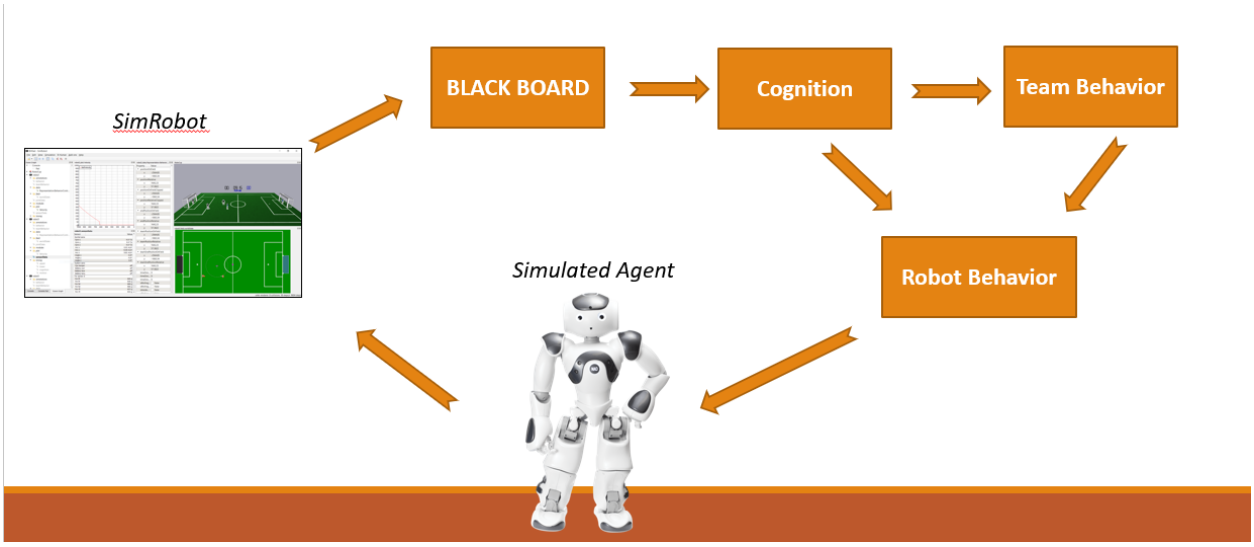


Figure 10 : Methodology

4.3 Behavior of Single Agent in moving ball scenario

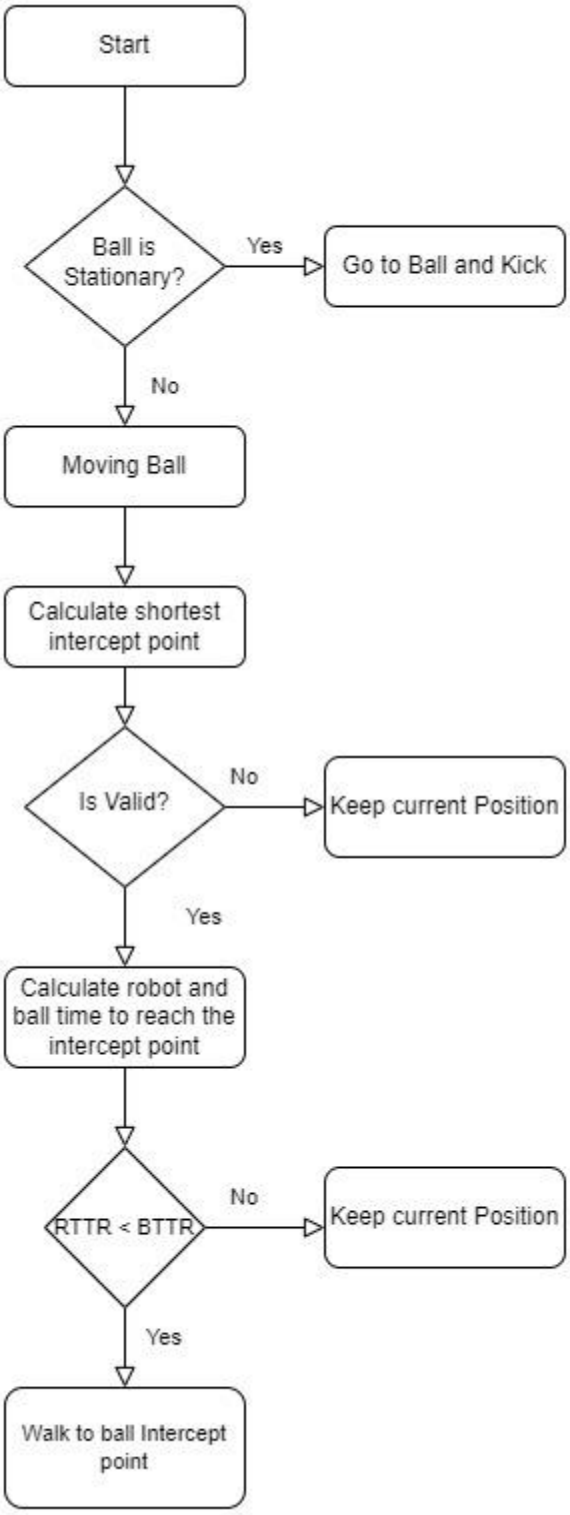


Figure 11 : Flow chart of Single Agent Behavior

4.4 Behavior of Multiple Agents in moving ball scenario

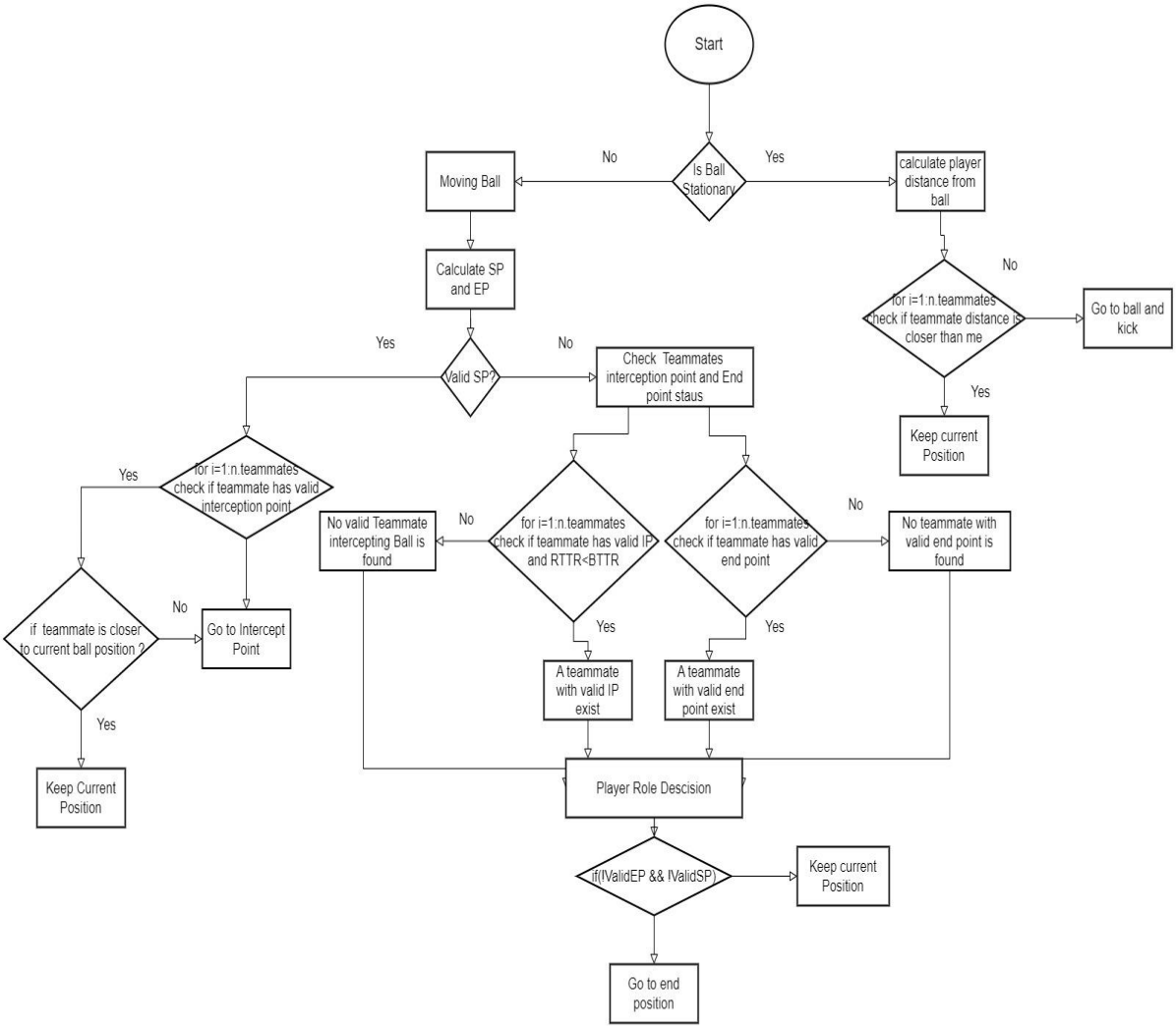


Figure 12: Multi-Agent Behavior for ball Interception

CHAPTER 5: Results and Discussions

This section outlines the case studies that were included as part of thesis to demonstrate the application on proposed robot behavior in case of moving ball. The case studies show the effectiveness of active robot reaction in case of moving ball. The sub-sections below will discuss about the positioning of robots on field for each case. All the parameters including ball position, robot position, net force on ball for motion, interception points and calculated time for robot and ball will be discussed. Some of simulation parameters are liste

5.1 Single Agent Scenario

For simulation purposes, a single- agent behavior in case of rolling ball is discussed. When a striker form opponent team, strikes towards the goal, the defenders from own team should try to gain ball possession as quickly as possible. In case of mobile robots, the fast reactive behavior enables the robot to get to the ball either by tracking or intercepting it. For humanoids, as the motion is relatively very slow as compared to ball, the reaction time is very limited. However, in situations, where it might be possible to block the incoming ball, it is a wise idea to attempt reaching the ball considering the feasibility. To demonstrate this, for single agent, two conditions are simulated.

5.1.1 Single Agent Active Interception

The walk-engine of B-Human is similar to a robot with omni-directional wheels. For distances less than or equal to 1m, the robot plans stepping motion in such a way that robot will not turn and move towards a point. This can be explained as for instance, if the robot is standing at (0,0) – facing 0 deg in field coordinates and the walk engine gets request to move the robot to (1000,1000), a change of 45 deg in robot heading, the robot will not attempt turning towards the point but the foot placement is adjusted so that the robot will reach the given point as fast as possible. This is explained using (Figure 13) .

For distances greater than 1m, the stepping selection would direct the robot to turn and approach the point. So, for distances greater than 1m, walking to shortest distances won't be feasible for cases where the computed interception point lies such that the robot has to turn and walk. In those cases, the better approach is to calculate interception point that lies in robot's field of view. For the points that lie in robot's field of view, the robot will simply walk towards the point if it can reach there before the ball. However, the simulation results reveal that for any distance greater

than 1m, the reaction time of robot to reach the interception point is more than 5 seconds. So, however moving to heading based intercept point was feasible, there won't be such situations where robot will have more than 5 seconds to react so these scenarios might not even exist in actual gameplay.

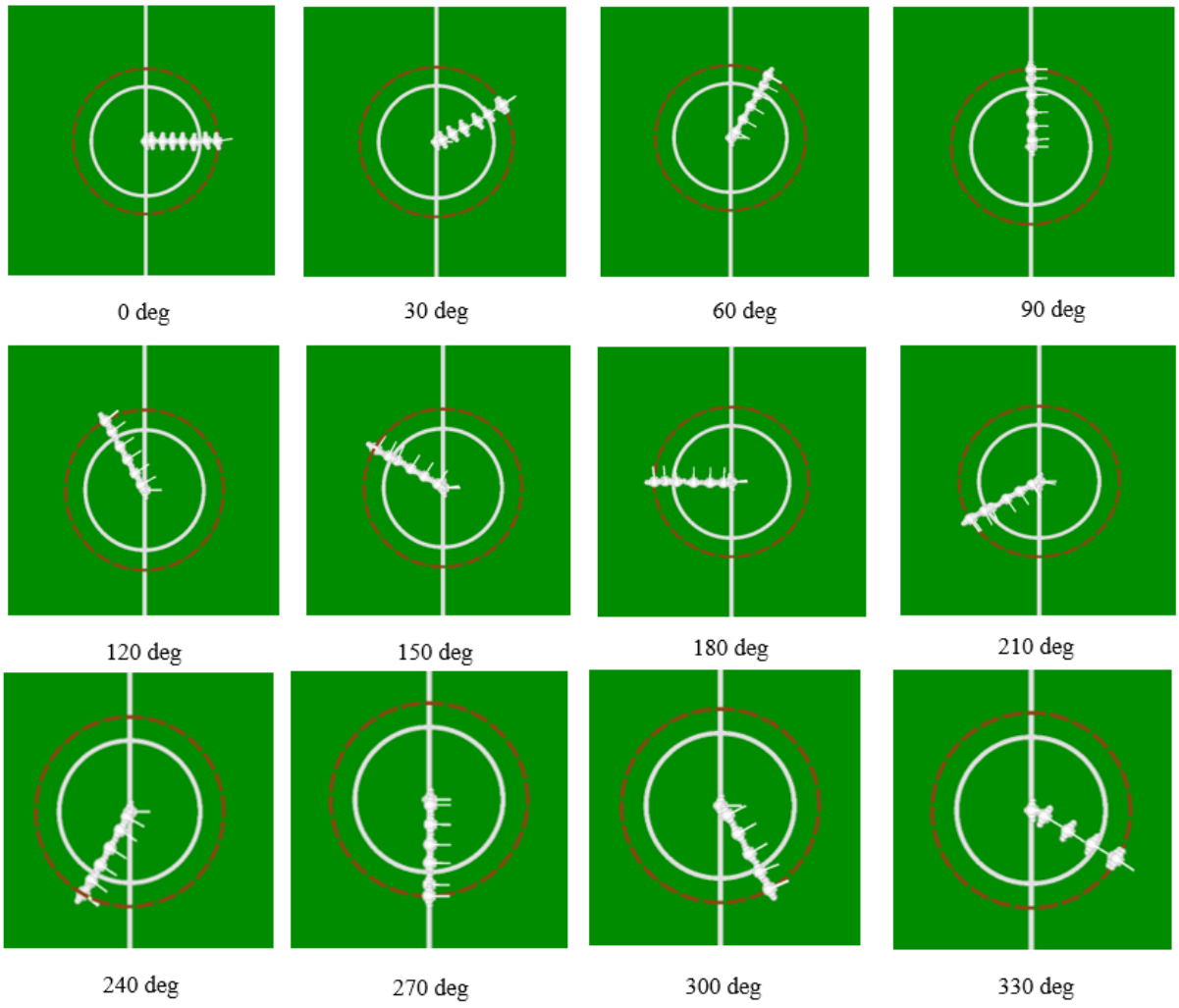


Figure 13 : Omni-Directional Walk Engine of NAO

Exploiting the omni-directional behavior, the shortest path from robot to the line in which the ball is moving is calculated using relations explained in Chapter 2. The validity is tested and if the robot can reach the point before the ball, interception is attempted. In the second iteration of single-agent behavior, two robots are added however, the decision is done as per the strategy of B-Human team in which the robot that is closest to ball end position will move towards the ball end position. Velocity comparison of both cases showed our strategy to be faster in gaining ball possession.

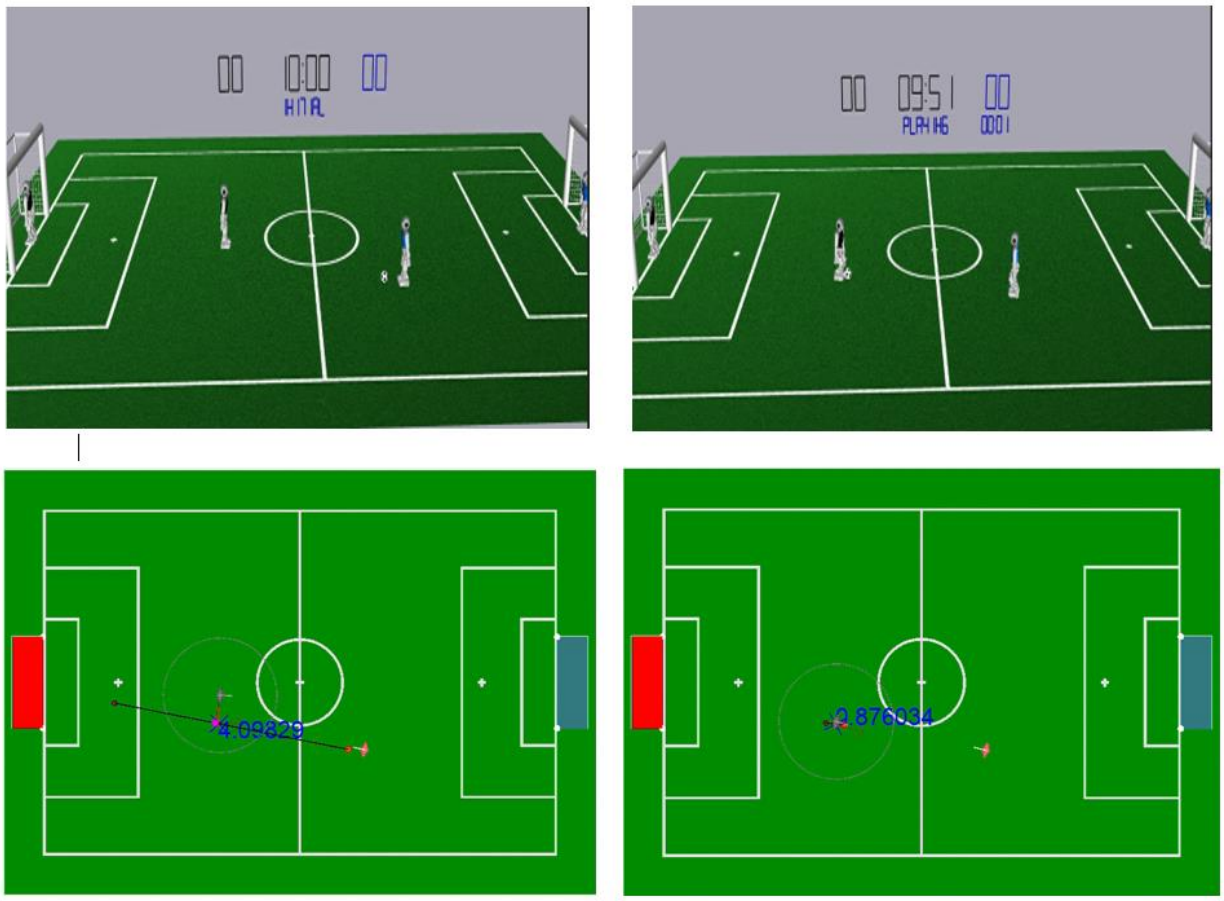


Figure 14: Single Agent proposed strategy

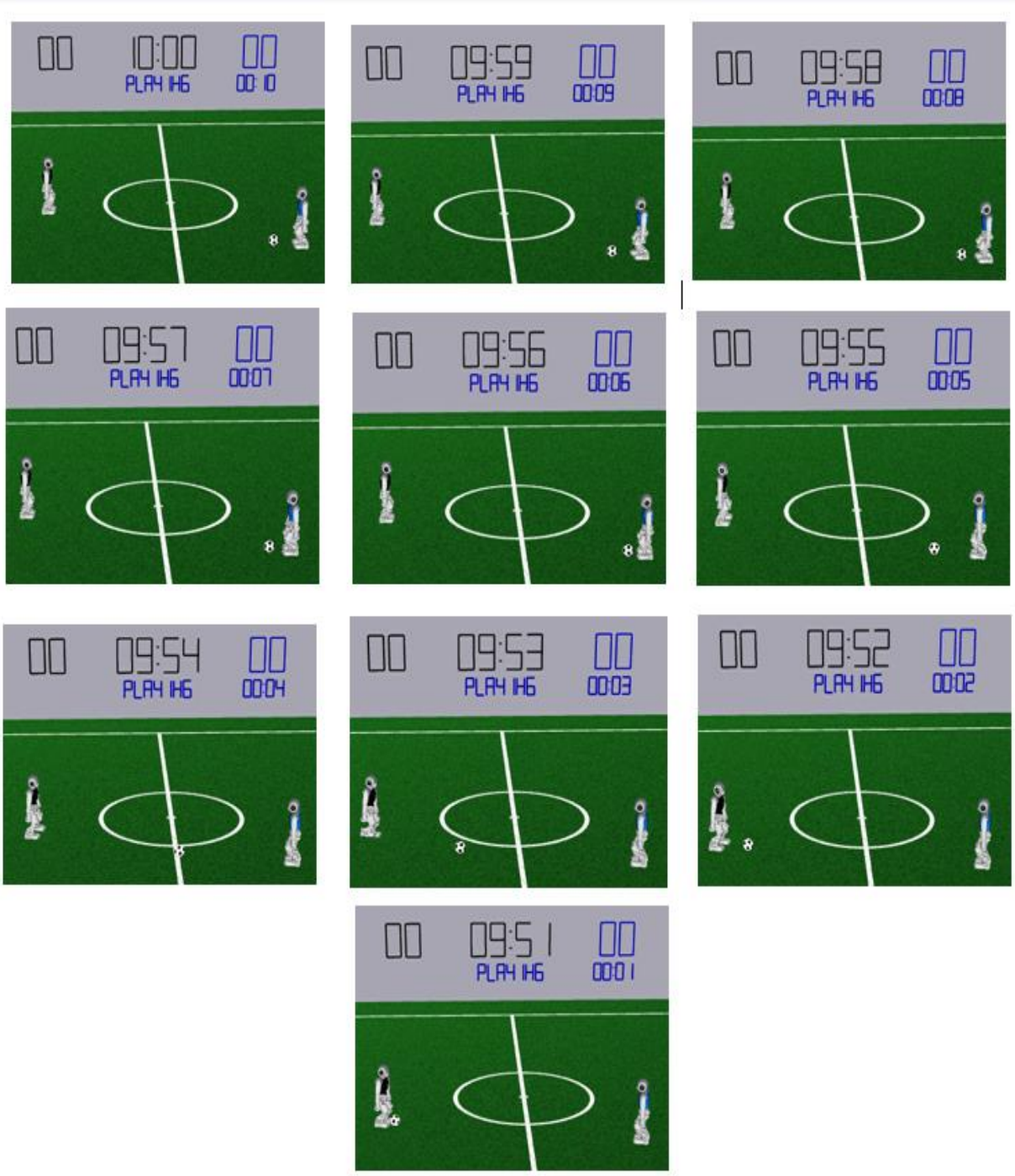


Figure 15 : Single Agent Proposed Scheme

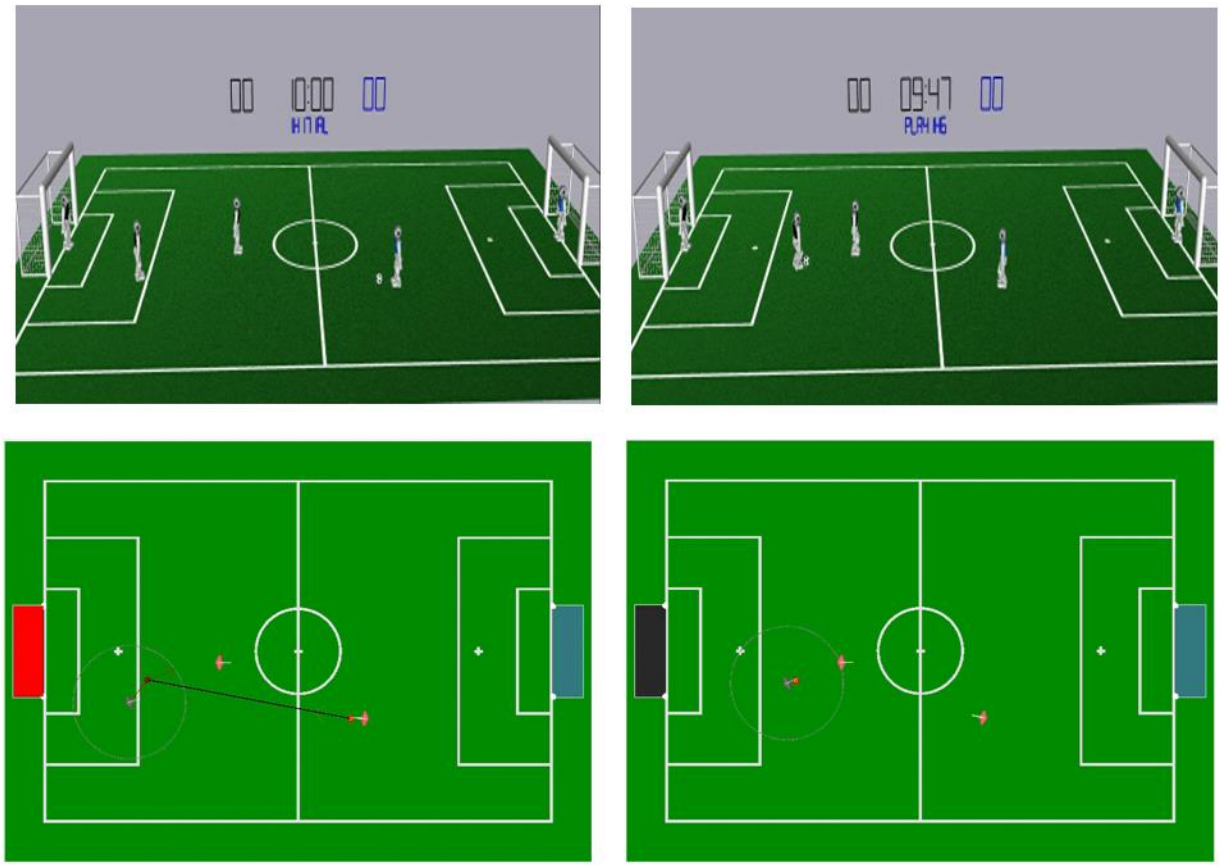


Figure 17: Intercepting Agent moving to Ball end position

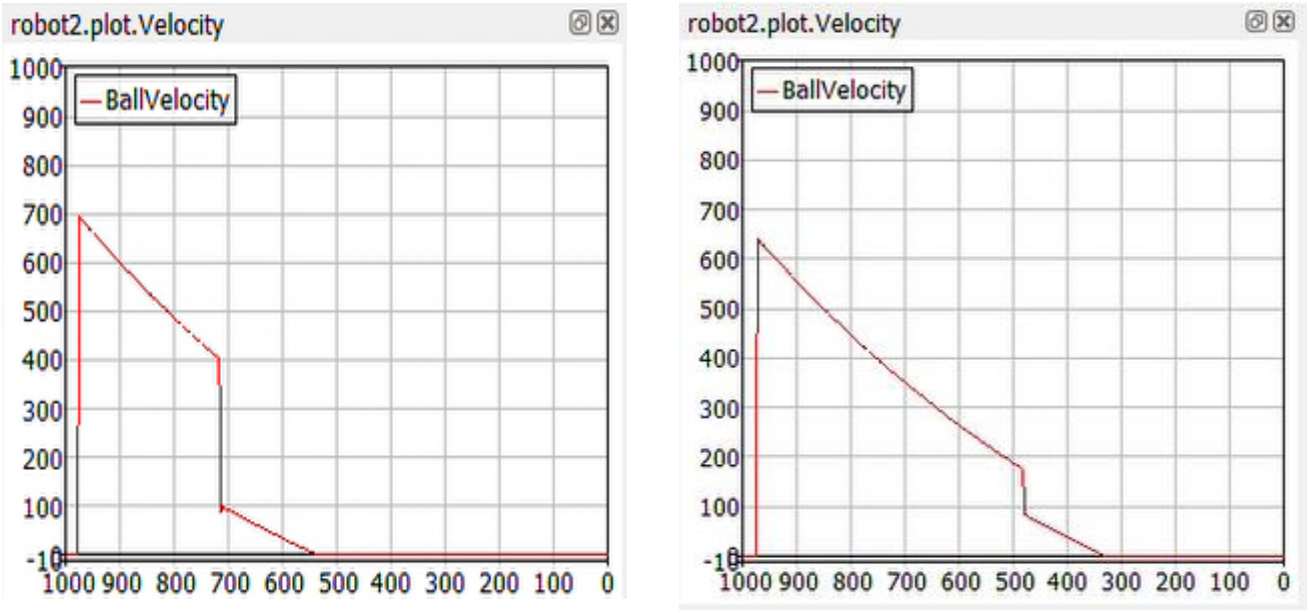


Figure 18: Velocity Comparison for interception Strategy

Parameters	Value
V_{robot}	200 $mmsec^{-1}$
V_{ball}	500-700 $mmsec^{-1}$
$b_{friction}$	0.03

Table 1: Simulation Parameters

	Robot Time to Reach (IP/EP)	Ball Time to Reach (EP/IP)
Case 1	3.2	4.9
Case 2	3.4	7.2

Table 2: Time Comparison of Single Agent Strategy

Analyzing the results from above simulated scenarios, it is evident that the strategy that directs the robot to walk towards calculated interception point will help the team to get ball possession quickly. Also, when a teammate that is closest to ball end position moves to that point, even if the robot gets to point more quicky, still, the robot will have to wait the time the ball will reach its end point. This adds to a delay in getting ball possession.

5.1.2 Multi-agent Scenario

In case of multiple defenders in field, the selection decision is made considering multiple factors. The flowchart explained in section gives a detail review of approach used. The experimental details for multi-agent scenario are given as

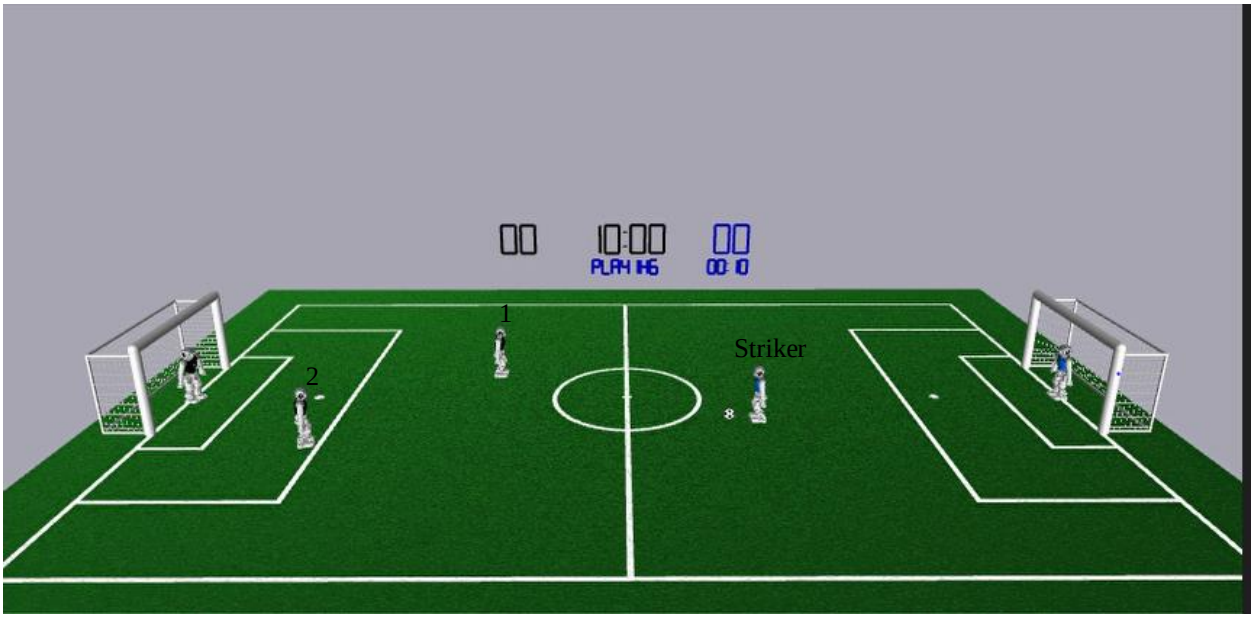
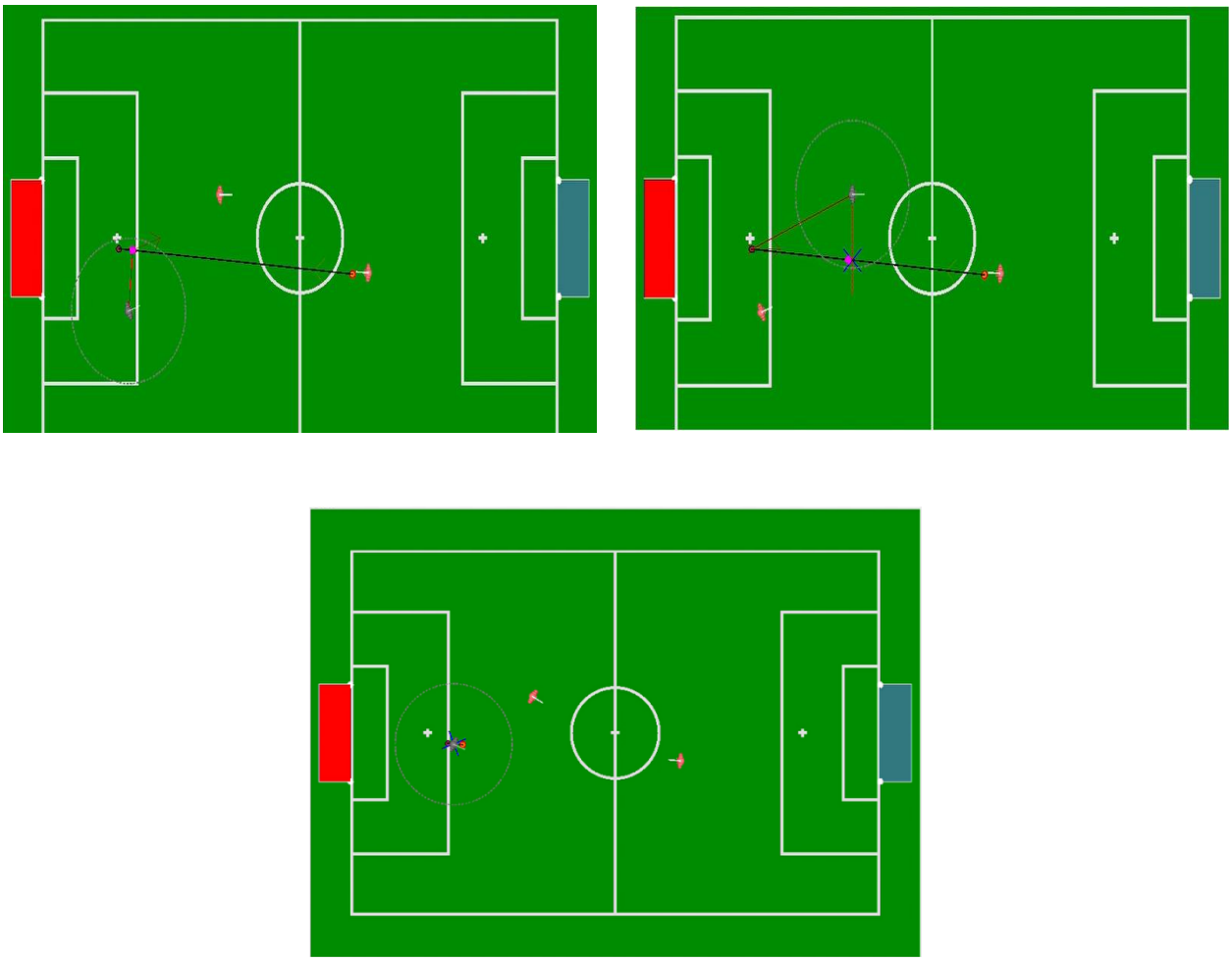


Figure 19 : Playing Field View of Multi-agent Scenario

Player	Position X (m)	Position Y (m)
Striker	-1.3	0.5
Case 1		
Player 1	1.4	-0.6
Player 2	3.0	1.0
Case2		
Player 1	1.4	0.1
Player 2	3.0	1.0

Table 3: Player positioning for Multi-Agent Scenario



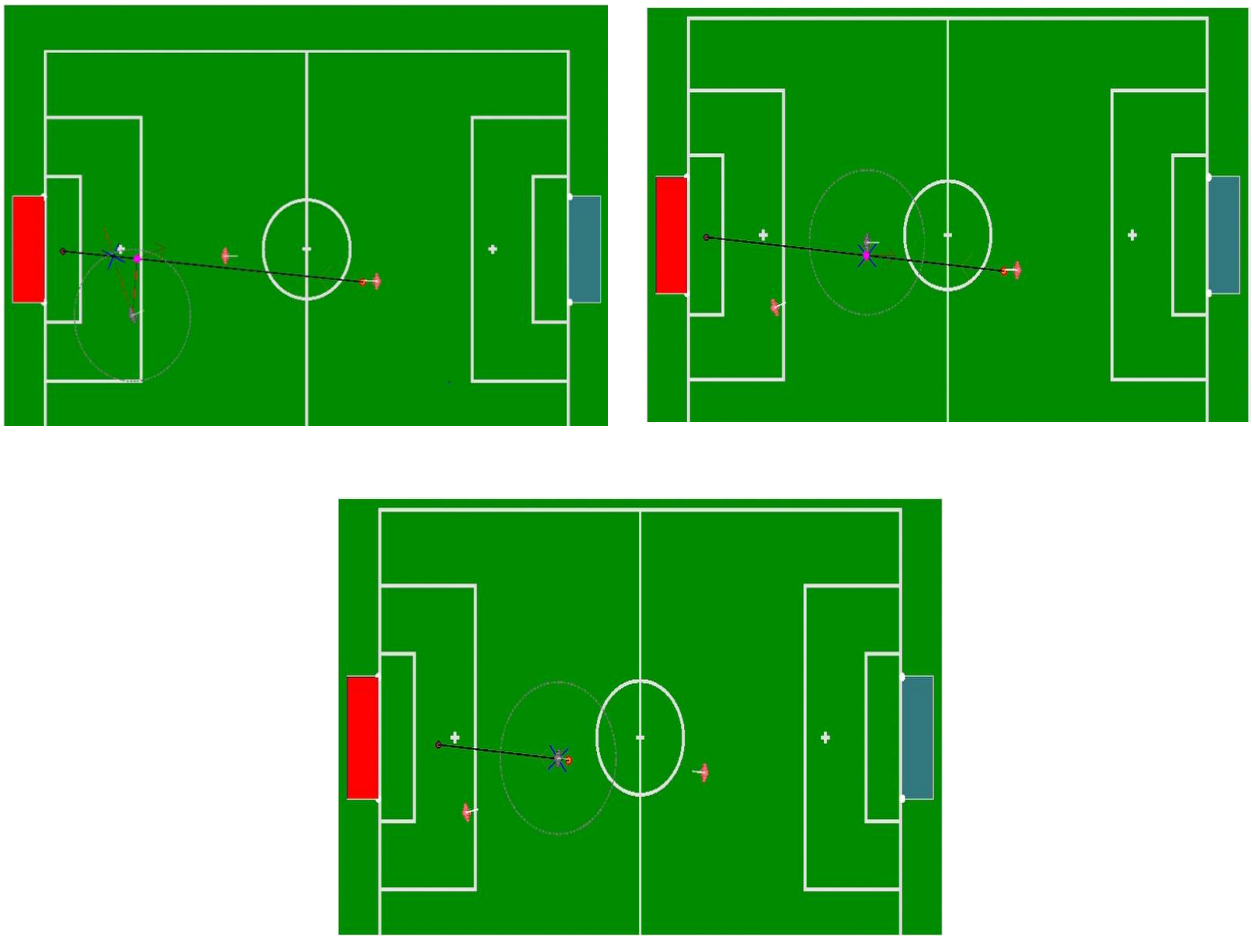
robot5.teamBehavior

TeamBehaviorControl	125.03
TeamBehaviorCard	25.03
PlayingTeamCard	4.85
TeamActivity	25.03
teamActivity = playingTeamMovingball;	
TimeToReachBall	25.03
timeToReachBall = { timeWhenReachBall = 8; timeWhe	
TeammateRoles	25.03
teammateRoles = { roles = []; captain = -1; timestamp	
Role	25.03
role = { role = none; numOfActiveSupporters = 0; play	

robot3.teamBehavior

TeamBehaviorControl	125.03
TeamBehaviorCard	25.03
PlayingTeamCard	4.87
TeamActivity	25.03
teamActivity = playingTeamMovingball;	
TimeToReachBall	25.03
timeToReachBall = { timeWhenReachBall = 15; timeWhenReachBallS	
TeammateRoles	25.03
teammateRoles = { roles = []; captain = -1; timestamp = 0; };	
Role	25.03
role = { role = ballInterceptor; numOfActiveSupporters = 0; playerDi	

Figure 20: Multi-agent scenario case 1(Activation Graph and Field View)



TeamBehaviorControl	113.17
TeamBehaviorCard	13.17
PlayingTeamCard	3.72
TeamActivity	13.17
teamActivity = playingTeamMovingball;	
TimeToReachBall	13.17
timeToReachBall = { timeWhenReachBall = 10; timeWh	
TeammateRoles	13.17
teammateRoles = { roles = []; captain = -1; timestamp	
Role	13.17
role = { role = ballInterceptor; numOfActiveSupporters	

TeamBehaviorControl	113.18
TeamBehaviorCard	13.18
PlayingTeamCard	3.72
TeamActivity	13.18
teamActivity = playingTeamMovingball;	
TimeToReachBall	13.18
timeToReachBall = { timeWhenReachBall = 18; timeWhenReachBallS	
TeammateRoles	13.18
teammateRoles = { roles = []; captain = -1; timestamp = 0; };	
Role	13.18
role = { role = none; numOfActiveSupporters = 0; playerDist = 3969.	

Figure 21 : Multi-agent scenario case 2 (Activation Graph and Field View)

For multi-agent scenario 1, though the player 1 had a valid interception point, but as it cannot reach the interception point before the ball, the decision is made for player 2 to be the ball interceptor. For scenario 2, both the robots had valid interception point and the time for both the robots to reach the interception point was less than the time the ball would have taken to reach interception points. Hence, as player 1 was closest player, it was considered the most feasible choice and the algorithm assigns role of ball interceptor to player 1.

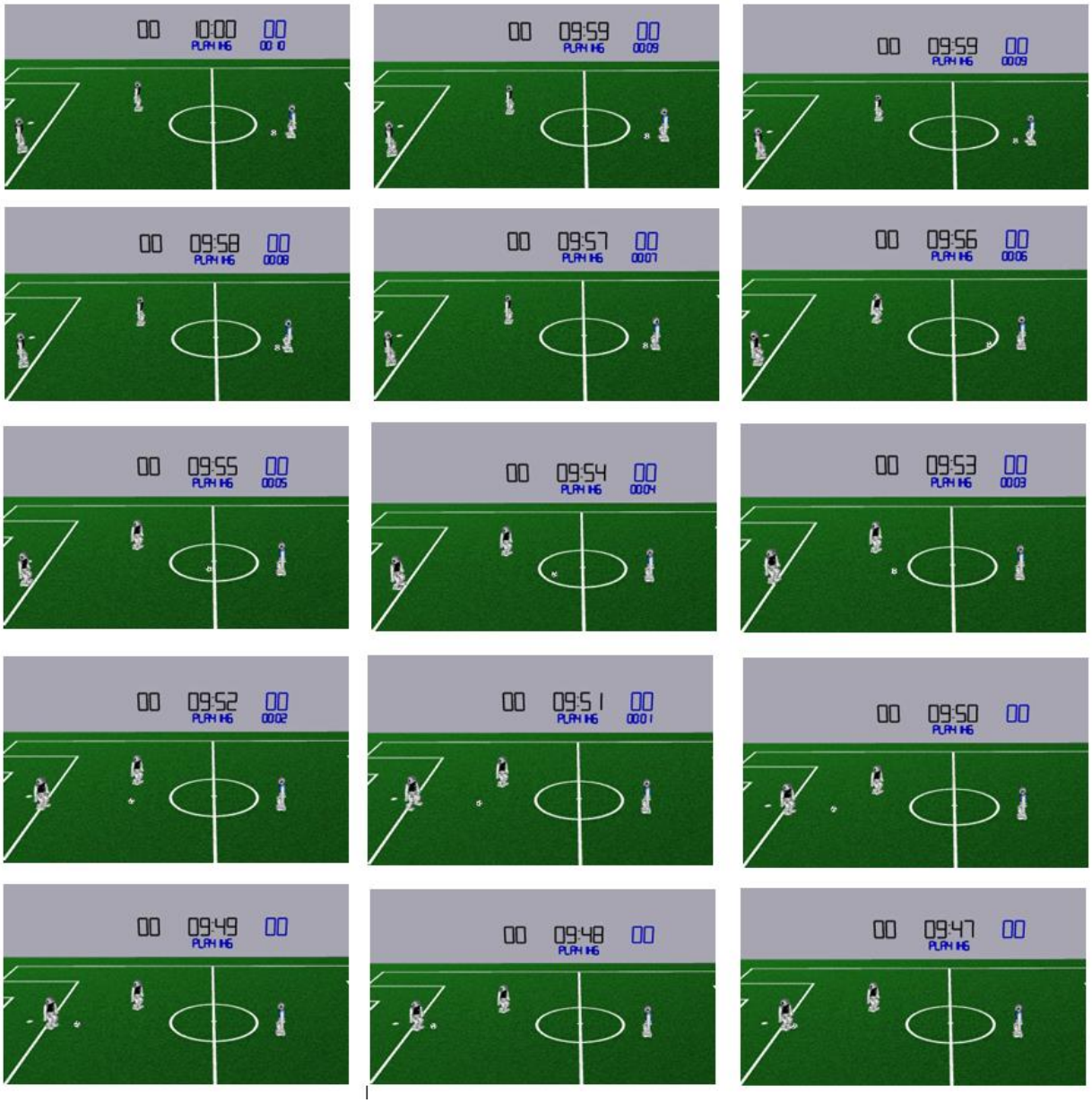


Figure 22: Multi-agent Scenario - Case 1

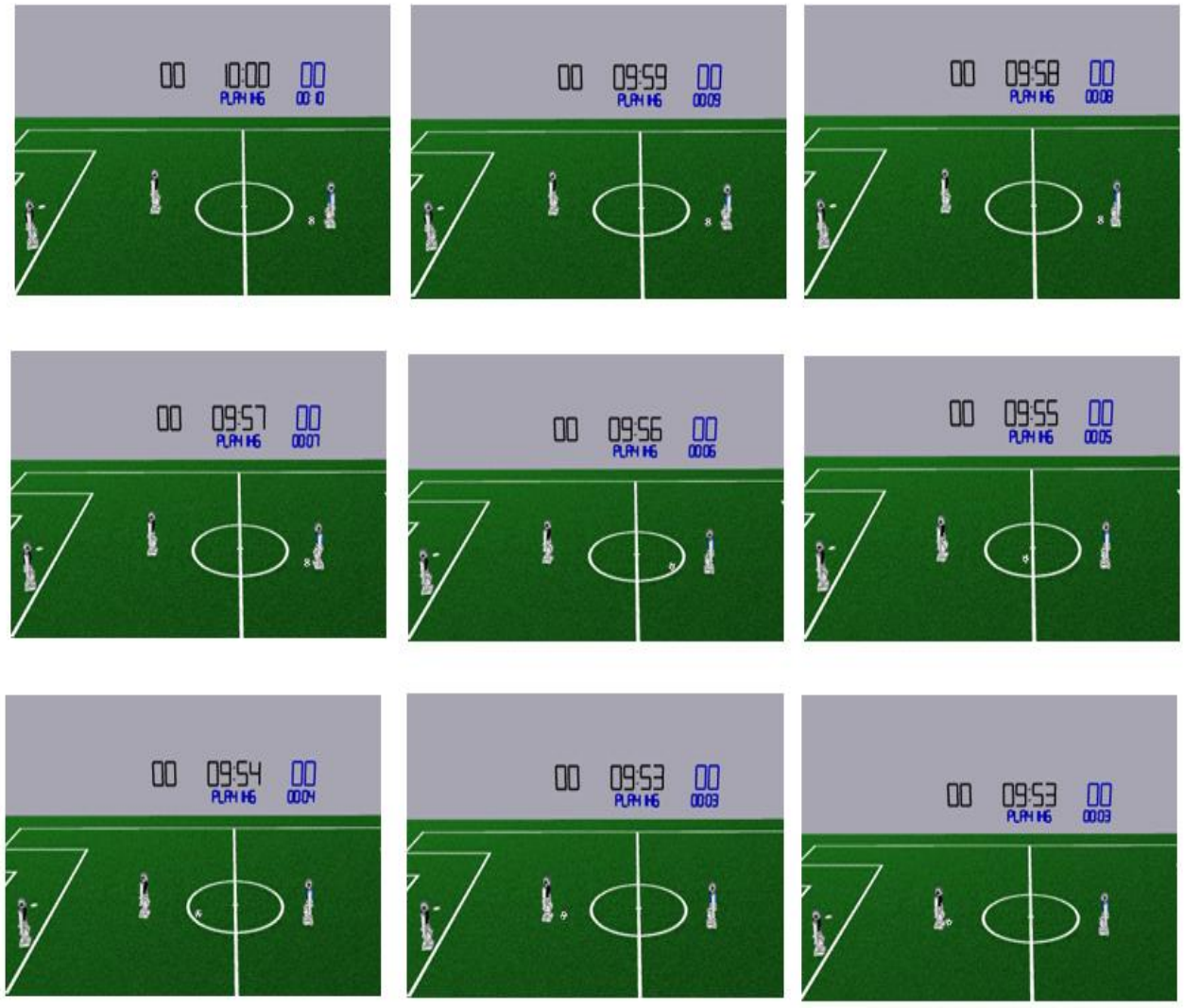


Figure 23: Multi-agent Scenario - Case 2

CHAPTER 6: Conclusion and Future Works

A behavior of humanoid robot in soccer gameplay in situations mainly related to rolling ball instances is presented as a part of this research. The usefulness of proposed robot response in defensive modes is highlighted. The architecture is included as a part of individual player behavior and team behavior in code base of Team B-Human (A team actively participating in Robocup since 2008). Team B-Human consist of students and researchers from Department of Computer Science of the University of Bremen and the DFKI research area Cyber-Physical Systems. As part of demonstrating the behavior of robot in case of rolling ball, two cases were discussed. 1)The ball passes within range of 1m radius of robot and the robot moves to intercept it. The results of this case were compared with the strategy of moving to ball end position and our approach yielded quick ball possession. 2) The role selection between multiple players for best player in case of rolling ball.

The research focuses on highlighting the benefits of robot response in scenarios where ball is moving. A prompt response in such cases is highly beneficial for the defenders to gain ball possession. In game scenarios, getting ball possession before the opponent or blocking passes between opponent players is highly advantageous and the team with capabilities to respond not only to the static ball in field but also to the moving ball is likely to perform better. Also, ball interception is one of the key components in real – world soccer matches. For humanoids, due to their limited capability of vision and slow maneuvering, the possibilities of intercepting moving targets isn't explored much.

The proposed behavior strategy has shown promising results as compared to the previous schemes of reaching to ball's expected end position. The approach, however, still needs to be tested on real-robots to test the performance in real world scenarios. The robot behavior after intercepting the ball needs to be re-defined for better team performance.

Deploying this behavior to real world scenario is only possible when robot localization is fairly accurate. Also, the team ball model needs to be relatively close to actual ball position on the field. The moving ball might not always be in robots' field of view so the reacting robot might be relying on team ball model. The response is studied currently for kicks with impact force of 2-3N resulting in ball starting velocity of 500 - 700 mm per sec. The friction parameters are adjusted so the ball rolls for a long time and moves large distance. The success of implementing this approach for defenders is also dependent to the activeness of robot response. Currently, with new researchers being carried out for faster and longer distance kicks, the response time of robot bi-ped locomotion needs to be reduced. Only fast and stable robot motions will help to achieve the goal of intercepting moving target in SPL game scenarios. The proposed methodology can also be incorporated in simulation leagues of Robocup.

In future, we wish to test the proposed behavior on actual robot which will help us identify the shortcomings and real-world constraints. We also wish to explore humanoid motion planning that is faster, stable and more react able to changes in ball movement. Also, obstacle avoidance while planning interception is also an area we wish to explore. Decision making during situations where game state is changing is an interesting area of research and real-time implementation of experimental work carried out in this research is the main goal for future.

.

REFERENCES

[1] H. Mahdi, S. A. Akgun, S. Saleh, and K. Dautenhahn, "A survey on the design and evolution of social robots — Past, present and future," *Rob Auton Syst*, vol. 156, Oct. 2022, doi: 10.1016/j.robot.2022.104193.

[2] E. Antonioni, V. Suriani, F. Riccio, and D. Nardi, "Game Strategies for Physical Robot Soccer Players: A Survey," *IEEE Trans Games*, vol. 13, no. 4, pp. 342–357, Dec. 2021, doi: 10.1109/TG.2021.3075065.

[3] N. Das, "Trajectory Modeling, Estimation and Interception of a Thrown Ball using a Robotic Ground Vehicle," 2018.

[4] F. Altché and A. de La Fortelle, "An LSTM Network for Highway Trajectory Prediction," Jan. 2018, [Online]. Available: http://arxiv.org/abs/1801.07962

[5] Y. Mirmohammad, S. Khorsandi, M. N. Shahsavari, B. Yazdankhoo, and S. Sadeghnejad, "Ball Trajectory Prediction for Humanoid Robots: Combination of k-NN Regression and Autoregression Methods Tele-Operation and Haptic Training Systems View project Ball Path Prediction for Humanoid Robots: Combination of k-NN Regression and Autoregression Methods," 2021. [Online]. Available: https://www.researchgate.net/publication/336798460

[6] B. Yazdankhoo, M. N. Shahsavari, S. Sadeghnejad, and J. Baltes, "Prediction of a Ball Trajectory for the Humanoid Robots: A Friction-Based Study," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2019, vol. 11374 LNAI, pp. 387–398. doi: 10.1007/978-3-030-27544-0_32.

[7] Universitas Gadjah Mada. Fakultas Teknik, Institute of Electrical and Electronics Engineers, I. International Conference on Mechanical and Manufacturing Engineering (7th : 2016 : Yogyakarta, and T. International Conference on Science, *Decision Tree Analysis for Humanoid Robot Soccer Goalkeeper Algorithm*.

[8] F. Stolzenburg, O. Obst, and J. Murray, "Qualitative Velocity and Ball Interception."

[9] C. Qu, J. He, J. Li, C. Fang, and Y. Mo, "Moving Target Interception Considering Dynamic Environment," May 2022, [Online]. Available: http://arxiv.org/abs/2205.07772

[10] F. Belkhouche and B. Belkhouche, "Modified parallel navigation for ball interception by a wheeled mobile robot goalkeeper," *Advanced Robotics*, vol. 20, no. 4, pp. 429–452, 2006, doi: 10.1163/156855306776562260.

[11] L. Freda and G. Oriolo, "Vision-based interception of a moving target with a nonholonomic mobile robot," *Rob Auton Syst*, vol. 55, no. 6, pp. 419–432, Jun. 2007, doi: 10.1016/j.robot.2007.02.001.

[12] M. M. Bachtiar, F. L. Hakim Ihsan, I. K. Wibowo, and R. E. Wibowo, "Intercept Algorithm for Predicting the Position of Passing the Ball on Robot Soccer ERSOW," *Inform : Jurnal Ilmiah Bidang Teknologi Informasi dan Komunikasi*, vol. 6, no. 1, pp. 35–39, Jan. 2021, doi: 10.25139/inform.v6i1.3353.

[13] J. Silva, N. Lau, J. Rodrigues, and J. L. Azevedo, "Ball sensor fusion and ball interception behaviours for a Robotic Soccer Team." [Online]. Available: http://www.robocup.org/

[14] H. Ghaderi, M. Yadegar, N. Meskin, and M. Noorizadeh, “A Novel Proportional Navigation Based Method for Robotic Interception Planning with Final Velocity Control,” *IEEE Access*, vol. 9, pp. 106428–106440, 2021, doi: 10.1109/ACCESS.2021.3091152.

[15] Q. Zhu, J. Hu, and L. Henschen, “A new moving target interception algorithm for mobile robots based on sub-goal forecasting and an improved scout ant algorithm,” *Applied Soft Computing Journal*, vol. 13, no. 1, pp. 539–549, Jan. 2013, doi: 10.1016/j.asoc.2012.08.013.

[16] R. Wang, M. Veloso, and S. Seshan, “Multi-robot information sharing for complementing limited perception: A case study of moving ball interception,” in *Proceedings - IEEE International Conference on Robotics and Automation*, 2013, pp. 1884–1889. doi: 10.1109/ICRA.2013.6630826.

[17] “RoboCup Federation.‘RoboCup Soccer Humanoid League Rules and Setup,’ RoboCup Humanoid League, 2017, pp.34–45.” [Online]. Available: http://www.robocup.org/leagues/3www.robocuphumanoid.org10

[18] R. D. Pristovani, B. E. Henfri Ajir, K. A. Subhan, D. Sanggar, and Pramadihanto. Dadet, “Implementation of Direct Pass Strategy during Moving Ball for ‘T-FLoW’ Humanoid Robot,” 2017, pp. 223–228.

[19] T. R. ¨ Ofer *et al.*, “Team Report and Code Release 2019.”

[20] M. Jafari, S. Saeedvand, and H. S. Aghdasi, “A Hybrid Q-learning Algorithm to Score a Moving Ball for Humanoid Robots,” in *2019 5th Conference on Knowledge Based Engineering and Innovation (KBEI)*, 2019, pp. 498–503. doi: 10.1109/KBEI.2019.8735027.

[21] B. Thesis, D.-I. habil Sahin Albayrak, J. Schneider Betreuer, and Y. Xu Johannes Schneider, “Reactive Full-Body Behaviour for Humanoid Robot-ball blocking behaviour for robot field player.”

[22] B. Liu, Y. Zhu, P. Macalpine, and P. Stone, “UT Austin Villa 3D Simulation Soccer Team 2020.” [Online]. Available: http://www.ode.org/

[23] R. Etemadi, H. Sajjadi, and A. Fathi, “Nexus3D Team Description Paper.”

[24] N. Gupta and S. Kalyanakrishnan, “Learning complex behaviours and Keepaway in 3D Robocup Environment.”

[25] B. W. Gore, “On finding the shortest distance of a point from a line: Which method do you prefer?,” *Resonance*, vol. 22, no. 7, pp. 705–714, 2017, doi: 10.1007/s12045-017-0514-x.

[26] T. Röfer *et al.*, “German Team 2007 : the German National RoboCup Team,” 2004.

[27] T. Röfer Thomas and Laue, “On B-Human’s Code Releases in the Standard Platform League – Software Architecture and Impact,” in *RoboCup 2013: Robot World Cup XVII*, 2014, pp. 648–655.

[28] K. and R. T. Laue Tim and Spiess, “SimRobot – A General Physical Robot Simulator and Its Application in RoboCup,” in *RoboCup 2005: Robot Soccer World Cup IX*, 2006, pp. 173–183.

[29] T. Laue and T. Röfer, “SimRobot – Development and Applications ?,” 2008.

[30] R. Smith, “Open Dynamics Engine.” 2008. [Online]. Available: http://www.ode.org/

Appendix

Playing Team Card

```
/**
 * @file PlayingTeamCard.cpp
 *
 * This file implements card that is active in situations where there is playing team behavior.
 *
 *
 */

#include "Representations/BehaviorControl/TeamSkills.h"
#include "Tools/BehaviorControl/Framework/Card/TeamCard.h"
#include "Representations/Communication/GameInfo.h"
#include "Representations/Communication/RobotInfo.h"
#include "Representations/Modeling/RobotPose.h"
#include "Representations/BehaviorControl/FieldBall.h"
#include "Representations/Modeling/BallModel.h"
#include "Representations/Configuration/BallSpecification.h"
#include "Tools/Math/Eigen.h"
#include "Representations/Communication/TeamData.h"
#include "Representations/Communication/TeamInfo.h"
#include "Tools/Modeling/BallPhysics.h"
#include "Tools/Math/Geometry.h"
#include <math.h>
#include "Tools/Math/Angle.h"


TEAM_CARD(PlayingTeamCard,
{,
  CALLS(Role),
  CALLS(TeamActivity),
  CALLS(TeammateRoles),
  CALLS(TimeToReachBall),
  REQUIRES(GameInfo),
  REQUIRES(OpponentTeamInfo),
  REQUIRES(RobotInfo),
  REQUIRES(RobotPose),
  REQUIRES(BallSpecification),
  REQUIRES(BallModel),
  REQUIRES(FieldBall),
  REQUIRES(TeamData),
});

class PlayingTeamCard : public PlayingTeamCardBase
{
  bool preconditions() const override
  {
            theGameInfo.state == STATE_READY || theGameInfo.state == STATE_PLAYING;
    return true;
  }

  bool postconditions() const override
  {
```

```
        theGameInfo.state != STATE_READY || theGameInfo.state != STATE_PLAYING;
  return true;
}

void execute() override
{
 // const RoboCup::RobotInfo& player[6] = theOpponentTeamInfo.players;

  if(theBallModel.estimate.velocity.norm()==0.0f) // ball is stationary
  {
   //const RoboCup::RobotInfo players[6] = theOpponentTeamInfo.players;
   // team activity description
   theTeamActivitySkill(TeamBehaviorStatus::playingTeamStationaryball);
   // calculating time to reach ball for current player
   TimeToReachBall mytimetoreachball;
   getTimetoReach(mytimetoreachball);
   theTimeToReachBallSkill(mytimetoreachball);
   // teammates roles - not set for now
   theTeammateRolesSkill(TeammateRoles());
   // player role for stationary ball
   PlayerRole myplayer;
   getPlayerRoleS(myplayer); // if both robots are fairly close, sometimes , multiple robots status sets as playing
   theRoleSkill(myplayer);
  }
  else //moving ball
  {
   theTeamActivitySkill(TeamBehaviorStatus::playingTeamMovingball);
   //
   TimeToReachBall mytimetoreachball;
   getTimetoReach(mytimetoreachball);
   theTimeToReachBallSkill(mytimetoreachball);
   theTeammateRolesSkill(TeammateRoles());

   PlayerRole myplayer;
   getPlayerRoleM_mul(myplayer);
   theRoleSkill(myplayer);
  }

}

void getPlayerRoleM1(PlayerRole& mypl)
{

  bool iamvalid = false;
  bool validep = false;
  int i = 0; int j = 0; float RTTR; float BTTR; Vector2f IP; Vector2f EP; float distEP = 0.0f;
  float distEPteammate = 0.0f; Vector2f EPteammate; Vector2f collidePt;
  Vector2f SP; float distSP;
  if (theRobotInfo.number == 2)
  {
   mypl.collidept.x() = 0.0f;
   mypl.collidept.y() = 0.0f;
   mypl.RTTR = 0.0;
   mypl.BTTR = 0.0;
   mypl.role = PlayerRole::none;
  }
```

```
   if (theRobotInfo.number == 3)
   {
     Vector2f endPF = theRobotPose * BallPhysics::getEndPosition(theBallModel.estimate.position,
theBallModel.estimate.velocity, theBallSpecification.friction);

     RTTR = calcTimetoReach(theRobotPose, endPF);
     Vector2f posfield = theRobotPose * theBallModel.estimate.position;
     // float balldist1 = (posfield - mypl.collidept).norm();
     float balldist = sqrt(sqr(endPF.y() - posfield.y()) + sqr(endPF.x() - posfield.x()));
     //OUTPUT_TEXT("Balldist " << static_cast<float>(balldist));
     BTTR = BallPhysics::timeForDistance(theBallModel.estimate.velocity, balldist, theBallSpecification.friction);
     Vector2f endPositionOnField = theRobotPose * BallPhysics::getEndPosition(theBallModel.estimate.position,
theBallModel.estimate.velocity, theBallSpecification.friction);
     mypl.collidept = endPositionOnField;
     mypl.role = PlayerRole::ballendpos;
     mypl.BTTR = BTTR;
     mypl.RTTR = RTTR;
   }
   else
   {
     SP = getshortestPt(theRobotPose, theBallModel);
     distSP = (theRobotPose.translation - SP).norm();
     // OUTPUT_TEXT("Dist " << static_cast<float> (SP.x()));
     if (SP.x() != 9999 && distSP <= 1000)
     {
       collidePt = SP;
       RTTR = calcTimetoReach(theRobotPose, collidePt);
       //OUTPUT_TEXT("Time " << static_cast<float>(RTTR));
       Vector2f posfield = theRobotPose * theBallModel.estimate.position;
       float balldist = sqrt(sqr(collidePt.y() - posfield.y()) + sqr(collidePt.x() - posfield.x()));
       //OUTPUT_TEXT("Ball dist " << static_cast<float>(balldist));
       BTTR = BallPhysics::timeForDistance(theBallModel.estimate.velocity, balldist, theBallSpecification.friction);
       //  OUTPUT_TEXT("Time " << static_cast<float>(BTTR));
      /*
       mypl.collidept = collidePt;
       mypl.RTTR = RTTR;
       mypl.BTTR = BTTR;
       mypl.role = PlayerRole::ballInterceptor;*/
       if (RTTR < BTTR)
       {
         mypl.collidept = collidePt;
         mypl.RTTR = RTTR;
         mypl.BTTR = BTTR;
         mypl.role = PlayerRole::ballInterceptor;
         //mypl.role = PlayerRole::none;
       }

     }
     else if (distSP >= 1000)
     {
       IP = getinterceptionPt(theRobotPose, theBallModel);
       if (IP.x() != 9999) // valid IP
       {
         RTTR = calcTimetoReach(theRobotPose, IP);
         Vector2f posfield = theRobotPose * theBallModel.estimate.position;
         // float balldist1 = (posfield - mypl.collidept).norm();
```

```
        float balldist = sqrt(sqr(IP.y() - posfield.y()) + sqr(IP.x() - posfield.x()));
        //OUTPUT_TEXT("Balldist " << static_cast<float>(balldist));
        BTTR = BallPhysics::timeForDistance(theBallModel.estimate.velocity, balldist,
theBallSpecification.friction);
        if (RTTR < BTTR)
        {
          mypl.BTTR = BTTR;
          mypl.RTTR = RTTR;
          mypl.collidept = IP;
          mypl.role = PlayerRole::ballInterceptor;
          //mypl.role = PlayerRole::none;

        }

      }
    }
    else
    {
      Vector2f endPositionOnField = theRobotPose * BallPhysics::getEndPosition(theBallModel.estimate.position,
theBallModel.estimate.velocity, theBallSpecification.friction);

      RTTR = calcTimetoReach(theRobotPose, endPositionOnField);
      Vector2f posfield = theRobotPose * theBallModel.estimate.position;
      // float balldist1 = (posfield - mypl.collidept).norm();
      float balldist = sqrt(sqr(endPositionOnField.y() - posfield.y()) + sqr(endPositionOnField.x() - posfield.x()));
      //OUTPUT_TEXT("Balldist " << static_cast<float>(balldist));
      BTTR = BallPhysics::timeForDistance(theBallModel.estimate.velocity, balldist, theBallSpecification.friction);
      mypl.BTTR = BTTR;
      mypl.RTTR = RTTR;
      mypl.collidept = endPositionOnField;
      mypl.role = PlayerRole::ballendpos;
    //  mypl.role = PlayerRole::none;

    }


  }
 }

 void getPlayerRoleM_mul(PlayerRole& mypl)
 {
  // int number = theRobotInfo.number;
  // OUTPUT_TEXT("Robot " << static_cast<int>(number));
  if (theRobotInfo.isGoalkeeper())
    mypl.role = PlayerRole::goalkeeper;
  else if (theRobotInfo.number == 2)
    mypl.role = PlayerRole::none;

  else
  {
    // except for goal keeper - you check for every teammate valid interception pt

    bool iamvalid = false;
    bool validep = false;
    int rob_num = theRobotInfo.number;
    int i = 0; int j = 0; float RTTR; float BTTR; Vector2f IP; Vector2f EP; float distEP = 0.0f;
```

```
    float distEPteammate = 0.0f; Vector2f EPteammate;      Vector2f SP;

    SP = getshortestPt(theRobotPose, theBallModel);
    EP = theRobotPose * BallPhysics::getEndPosition(theBallModel.estimate.position,
theBallModel.estimate.velocity, theBallSpecification.friction);
    Vector2f posOnfield1 = theRobotPose * theBallModel.estimate.position; // ball position on field
    mypl.playerDist = (theRobotPose.translation - posOnfield1).norm();
    distEP = (theRobotPose.translation - EP).norm();
    if (SP.x() != 9999 && SP.y() != 9999)
    {

     RTTR = calcTimetoReach(theRobotPose, SP);
     Vector2f posfield = theRobotPose * theBallModel.estimate.position;
     float balldist = sqrt(sqr(SP.y() - posfield.y()) + sqr(SP.x() - posfield.x()));
     BTTR = BallPhysics::timeForDistance(theBallModel.estimate.velocity, balldist, theBallSpecification.friction);
     if (RTTR < BTTR)
     {
      mypl.BTTR = RTTR;
      mypl.RTTR = BTTR;
      iamvalid = true;

     }
     else
     {
      mypl.BTTR = 0.0f;
      mypl.RTTR = 0.0f;
      iamvalid = false;

     }

     int ival = 0;
     bool anotherplayer = false;
     if (iamvalid)
     {

      for (auto const& teammate : theTeamData.teammates)
      {
       bool teamyes = false;

       if (teammate.number != 1 && teammate.number != 2)
       {
        Vector2f collidePtteammate = getshortestPt(teammate.theRobotPose, teammate.theBallModel);

        if (collidePtteammate.x() != 9999 && collidePtteammate.y() != 9999)
        {
         RTTR = calcTimetoReach(teammate.theRobotPose, collidePtteammate);
         Vector2f posfield = teammate.theRobotPose * teammate.theBallModel.estimate.position;
         float balldist = (posfield - collidePtteammate).norm();
         BTTR = BallPhysics::timeForDistance(teammate.theBallModel.estimate.velocity, balldist,
theBallSpecification.friction);


        if (RTTR < BTTR)
        {
         ival = ival + 1;
         teamyes = true;
```

```
            }
            else
            {
              RTTR = 0.0;
              BTTR = 0.0;
              teamyes = false;
            }
          }
          // calculate distance to ball
          float mateDist = (teammate.theRobotPose.translation - posOnfield1).norm();
          if (teamyes && (mateDist < mypl.playerDist))
          {
            OUTPUT_TEXT("Its here");
            OUTPUT_TEXT("Rob " << static_cast<int> (theRobotInfo.number));
            anotherplayer = true;
            break;
          }


        }

      }
      if (anotherplayer == true)
      {
        mypl.role = PlayerRole::none;
      }
      else
      {
        mypl.role = PlayerRole::ballInterceptor;
        mypl.collidept = SP;
      }
    }

    else // i am not valid
    {
      for (auto const& teammate : theTeamData.teammates)
      {
        validep = false;

        if (teammate.number != 1 && teammate.number != 2)
        {
          EPteammate = teammate.theRobotPose *
BallPhysics::getEndPosition(teammate.theBallModel.estimate.position, teammate.theBallModel.estimate.velocity,
theBallSpecification.friction);
          distEPteammate = (teammate.theRobotPose.translation - EPteammate).norm();
          if (distEPteammate < distEP)
            validep = true;
        }
        if (validep)
          break;
      }
      //  OUTPUT_TEXT("Rob " << static_cast<int> (theRobotInfo.number) << "distEP " << static_cast<float>
(distEP) << "teammate dist " << static_cast<float>(distEPteammate) );

      // OUTPUT_TEXT("Rob " << static_cast<int> (theRobotInfo.number) <<"ValidEP " <<
static_cast<bool>(validep) << "Another pl " << static_cast<bool>(anotherplayer));
```

```
        if (!validep && !anotherplayer)
        {
          mypl.role = PlayerRole::ballendpos;
          mypl.collidept = EP;
        }
        else if (validep && anotherplayer)
        {
          mypl.role = PlayerRole::none;
          mypl.collidept.x() = 0.0f;
          mypl.collidept.y() = 0.0f;
          mypl.BTTR = 0.0f;
          mypl.RTTR = 0.0f;
        }

      }


      // OUTPUT_TEXT("Value" << static_cast<int>(i));
    }
    else
    {
      bool teammatecol = false;
      bool validep = false;
      // check if another player has valid IP
      // check if another player has valid EP
      // if no other player is currently interceptor and i am closest
      // go to ball end position
      for (auto const& teammate : theTeamData.teammates)
      {
        if (teammate.number != 1 && teammate.number != 2)
        {
          EPteammate = teammate.theRobotPose *
BallPhysics::getEndPosition(teammate.theBallModel.estimate.position, teammate.theBallModel.estimate.velocity,
theBallSpecification.friction);
          distEPteammate = (teammate.theRobotPose.translation - EPteammate).norm();
          if (distEPteammate < distEP)
            validep = true;
        }
        if (validep)
          break;
      }
      for (auto const& teammate : theTeamData.teammates)
      {

        if (teammate.number != 1 && teammate.number != 2)
        {
          Vector2f collidePtteammate = getshortestPt(teammate.theRobotPose, teammate.theBallModel);
          if (collidePtteammate.x() != 9999 && collidePtteammate.y() != 9999)
          {
            RTTR = calcTimetoReach(teammate.theRobotPose, collidePtteammate);
            Vector2f posfield = teammate.theRobotPose * teammate.theBallModel.estimate.position;
            float balldist = (posfield - collidePtteammate).norm();
            BTTR = BallPhysics::timeForDistance(teammate.theBallModel.estimate.velocity, balldist,
theBallSpecification.friction);

          }
```

```
        if (RTTR < BTTR)
        {
          teammatecol = true;
          break;
        }
        else
          teammatecol = false;

      }
    }
    if (!teammatecol && !validep)
    {
      mypl.role = PlayerRole::ballendpos;
      mypl.collidept = EP;
    }


  }

 }
}

void getPlayerRoleS(PlayerRole& mypl)
{
 //int number = theRobotInfo.number;
 //OUTPUT_TEXT("Robot " << static_cast<int>(number));

 if (theRobotInfo.isGoalkeeper())
   mypl.role = PlayerRole::goalkeeper;
 else if (theRobotInfo.number == 2)
   mypl.role = PlayerRole::goalkeeper;
 else
 {
   float val = 0.0f;
   float val1 = 0.0f;
   int rob_num = theRobotInfo.number;
   int i = 0; int j = 0;
   val = (theRobotPose.translation - theFieldBall.positionOnField).norm();
   mypl.playerDist = val;
   for (auto const& teammate : theTeamData.teammates)
   {
     if (teammate.number != 1)
     {
       float dist = (teammate.theRobotPose.translation - theFieldBall.positionOnField).norm();
       if (dist < val)
       {
         i = i + 1;
       }
      /* if (teammate.theRobotInfo.isballPlayer())
       {
         j = j + 1;
       }
       */
     }

   }
```

```
      // OUTPUT_TEXT("Value" << static_cast<int>(i));
      if (i == 0)
        mypl.role = PlayerRole::ballPlayer;
      else
        mypl.role = PlayerRole::none;
    }
    int num = mypl.supporterIndex();

  }
  void getTimetoReach(TimeToReachBall& TTRB)
  {
    float minBallDistanceForVelocity = 0.8f;
    float robotSpeed = 200.0f;
    Vector2f interceptRelativeRobot = theRobotPose.inversePose * theFieldBall.positionOnField;
    TTRB.timeWhenReachBall = std::max(0.f, std::max(interceptRelativeRobot.x() / robotSpeed,
interceptRelativeRobot.x() / robotSpeed) - minBallDistanceForVelocity);
  }
  float calcTimetoReach(RobotPose rpose,Vector2f pos)
  {
    float minBallDistanceForVelocity = 0.0f;//0.8f
    float robotSpeed = 200.0f;
    Vector2f interceptRelativeRobot = rpose.inversePose * pos;
    //OUTPUT_TEXT("x " << static_cast<float>(interceptRelativeRobot.y()));

    float dist = interceptRelativeRobot.norm();
    //OUTPUT_TEXT("x " << static_cast<float>(dist));

   // return std::max(0.f, std::max(interceptRelativeRobot.x() / robotSpeed, interceptRelativeRobot.x() / robotSpeed) -
minBallDistanceForVelocity);
    //return std::max(0.f, std::max(interceptRelativeRobot.y() / robotSpeed, interceptRelativeRobot.y() / robotSpeed) -
minBallDistanceForVelocity);

    return std::max(0.f, std::max(dist / robotSpeed, dist / robotSpeed) - minBallDistanceForVelocity);

  }
  Vector2f getinterceptionPt(RobotPose RP,BallModel BM)
  {
    float robotSlope = tan(toDegrees(RP.rotation) * pi / 180);
    float robotIntercept = RP.translation.y() - (robotSlope * RP.translation.x());
    float value_x=0.0;
    float ang = toDegrees(RP.anglefull);
    float x1, x2, y1, y2;
    float ballSlope, ballIntercept, RobotSlope, RobotIntercept;
    Vector2f positionOnField = RP * BM.estimate.position;
    Vector2f endPositionOnField = RP * BallPhysics::getEndPosition(BM.estimate.position, BM.estimate.velocity,
theBallSpecification.friction);
    x1 = positionOnField.x();
    y1 = positionOnField.y();
    x2 = endPositionOnField.x();
    y2 = endPositionOnField.y();
    float a1, b1, a2, b2, c1, c2;
    //fieldBall.distance = sqrt(sqr(x2 - 0) + sqr(y2 - 0));
    ballSlope = ((y2 - y1) / (x2 - x1));
    ballIntercept = y1 - (ballSlope * x1);
    a1 = -robotSlope;
    a2 = -ballSlope;
```

```
    b1 = 1.0f;
    b2 = 1.0f;
    c1 = -robotIntercept;
    c2 = -ballIntercept;
    Vector2f  interceptPoint;
    interceptPoint.x() = (((b1 * c2) - (b2 * c1)) / ((a1 * b2) - (a2 * b1)));
    interceptPoint.y() = (((c1 * a2) - (c2 * a1)) / ((a1 * b2) - (a2 * b1)));
   // fieldBall.isOnLine = isOnLinecheck(fieldBall.interceptPoint, theRobotPose.translation);
   // fieldBall.balldistance = sqrt(sqr(fieldBall.interceptPoint.y() - y1) + sqr(fieldBall.interceptPoint.x() - x1));
   // float test1 = sqr(theRobotPose.translation.x() - fieldBall.interceptPoint.x());
   // float test2 = sqr(theRobotPose.translation.y() - fieldBall.interceptPoint.y());
   // fieldBall.distancefromInterceptPt = sqrt(test1 + test2);//+ ;
    //fieldBall.distancefromInterceptPt = sqrt(sqr(theRobotPose.translation.x() - 0) + sqr(theRobotPose.translation.y()
- 0));
    bool valid = Geometry::isPointInsideRectangle2(positionOnField, endPositionOnField, interceptPoint);
    bool validR = isinrobotFOV(RP, interceptPoint);

    Vector2f test;
    test.x() = 9999;
    test.y() = 9999;
    if (valid && validR)
      return interceptPoint;
    else
      return test;
  }
  Vector2f getshortestPt(RobotPose RP, BallModel BM)
  {
    float robx = 0.0f; float roby = 0.0f;
    robx = RP.translation.x();
    roby = RP.translation.y();
    float x1, x2, y1, y2;
    float m, c;
    float robslopenew, robintcnew;
    Vector2f positionOnField = RP * BM.estimate.position;
    Vector2f endPositionOnField = RP * BallPhysics::getEndPosition(BM.estimate.position, BM.estimate.velocity,
theBallSpecification.friction);
    x1 = positionOnField.x();
    y1 = positionOnField.y();
    x2 = endPositionOnField.x();
    y2 = endPositionOnField.y();
    //int sign = 0;
    //fieldBall.distance = sqrt(sqr(x2 - 0) + sqr(y2 - 0));
    m = ((y2 - y1) / (x2 - x1)); // ball slope
    c = y1 - (m * x1);// ball intercept



    Vector2f  interceptPoint;
    interceptPoint.x() = ((m * roby) + robx - (m * c)) / (1 + (m * m));
    interceptPoint.y() = ((m * m * roby) + (m * robx) + c) / (1 + (m * m));;
  /*
    OUTPUT_TEXT("X" << static_cast<float>(interceptPoint.x()));
    OUTPUT_TEXT("Y" << static_cast<float>(interceptPoint.y()));
    OUTPUT_TEXT("Xst" << static_cast<float>(positionOnField.x()));
    OUTPUT_TEXT("Yst" << static_cast<float>(positionOnField.y()));
    OUTPUT_TEXT("Xend" << static_cast<float>(endPositionOnField.x()));
```

```
    OUTPUT_TEXT("Yend" << static_cast<float>(endPositionOnField.y()));
    */

    bool valid = Geometry::isPointInsideRectangle2(positionOnField, endPositionOnField, interceptPoint);
    // bool validR = isinrobotFOV(RP, interceptPoint);

   // OUTPUT_TEXT("valid" << static_cast<bool>(valid));
   Vector2f test;
   test.x() = 9999;
   test.y() = 9999;

   if (valid)
     return interceptPoint;
   else
     return test;


 }
 bool isinrobotFOV(RobotPose pose, Vector2f pt)
 {
   // addd for when calculated min or max range exceeds 360 or becomes negative
   Angle delta = 60_deg;
   Angle minrange = pose.anglefull - delta;
   Angle maxrange = pose.anglefull + delta;
   float transx = pt.x() - pose.translation.x();
   float transy = pt.y() - pose.translation.y();

   float ang = atan2(transy , transx) ;
   Angle angd = Angle::fromDegrees(ang*180/pi);
   if (signbit(transy))
     angd = 360_deg + angd;
   //OUTPUT_TEXT("TransX " << static_cast<float>(transx));
   //OUTPUT_TEXT("TransY " << static_cast<float>(transy));

   bool val;

  // OUTPUT_TEXT("angd " << static_cast<Angle> (angd));
  // OUTPUT_TEXT("max " << static_cast<Angle> (maxrange));
   //OUTPUT_TEXT("min " << static_cast<Angle> (minrange));
   if ((angd <= maxrange) && (angd >= minrange))
     val = true;
   else val = false;
 //  OUTPUT_TEXT("Val " << static_cast<bool> (val));




   return val;
  }
};

MAKE_TEAM_CARD(PlayingTeamCard);
```

Ball Interceptor Card

```
/**
 * @file CodeReleaseKickAtGoalCard.cpp
 *
 * This file implements a basic ball interceptor behavior for the code release.
 *
 *
 */

#include "Representations/BehaviorControl/FieldBall.h"
#include "Representations/BehaviorControl/Skills.h"
#include "Representations/Configuration/FieldDimensions.h"
#include "Representations/Modeling/RobotPose.h"
#include "Tools/BehaviorControl/Framework/Card/Card.h"
#include "Tools/BehaviorControl/Framework/Card/CabslCard.h"
#include "Representations/BehaviorControl/TeamBehaviorStatus.h"
#include "Tools/Math/BHMath.h"

CARD(CodeReleaseBallInterceptor,
  { ,
    CALLS(Activity),
    CALLS(GoToBallAndKick),
    CALLS(LookForward),
    CALLS(Stand),
    CALLS(WalkAtRelativeSpeed),
    CALLS(WalkToPoint),
    REQUIRES(FieldBall),
    REQUIRES(FieldDimensions),
    REQUIRES(RobotPose),
    REQUIRES(TeamBehaviorStatus),
    DEFINES_PARAMETERS(
    {,
      (float)(0.8f) walkSpeed,
      (int)(0) initialWaitTime,
      (int)(7000) ballNotSeenTimeout,
    }),
  });

class CodeReleaseBallInterceptor : public CodeReleaseBallInterceptorBase
{
  bool preconditions() const override
  {

    return (theTeamBehaviorStatus.role.role == PlayerRole::ballInterceptor);
  }

  bool postconditions() const override
  {
    return (theTeamBehaviorStatus.role.role != PlayerRole::ballInterceptor);
  }

  option
  {
    theActivitySkill(BehaviorStatus::codeReleaseBallInterceptor);

    initial_state(start)
```

```
{
 transition
 {
  if (state_time > initialWaitTime)
  {
   goto goToBallAndKick;
   //OUTPUT_TEXT("The program has completed initial wait time");
   //goto testingwalk;

  }
 }

 action
 {
  // OUTPUT_TEXT("Time is : " << static_cast<int>(state_time));

   theLookForwardSkill();
    theStandSkill();
 }
}


state(goToBallAndKick)
{
 transition
 {
  float distance = (theRobotPose.translation - theTeamBehaviorStatus.role.collidept).norm();
  if (distance < 10.0f)
  {
  OUTPUT_TEXT("Time is : " << static_cast<int>(state_time));

  goto waitState;
  }
  /* if (!(theTimeRepresentation.robotTimeToReach < theTimeRepresentation.ballTimeToReach))
   {
    OUTPUT_TEXT("Robot cannot reach point before ball");

    goto waitState;
   }*/
 }

  action
 {

  // Vector2f Pos;
   Vector2f Pos_rel;
   Pos_rel = theRobotPose.inversePose * theTeamBehaviorStatus.role.collidept;
   theLookForwardSkill();
   //theStandSkill();
   theWalkToPointSkill(Pos_rel, 1.0, true, false, false, false);

 }
}

 state(ballisStationary)
 {
```

```
        transition
        {
            if (theFieldBall.isMoving) // is moving and we have valid interception point
             goto goToBallAndKick;
        }

        action
        {
          //OUTPUT_TEXT("Time is : " << static_cast<int>(state_time));
        theLookForwardSkill();
        theStandSkill();

        //theWalkAtRelativeSpeedSkill(Pose2f(walkSpeed, 0.f, 0.f));
          //theWalkAtRelativeSpeedSkill(Pose2f(0.0f, walkSpeed, 0.f));
       }
    }
    state(waitState)
    {
       transition
       {
         if (theFieldBall.commulativeVelocity < 5.0f)
         goto ballisStationary;
       }

       action
       {
         // OUTPUT_TEXT("The program is in wait state");
      theLookForwardSkill();
      theStandSkill();
      //theWalkAtRelativeSpeedSkill(Pose2f(walkSpeed, 0.f, 0.f));
      //theStandSkill();
      //theWalkAtRelativeSpeedSkill(Pose2f(0.0f, walkSpeed, 0.f));
      }
  }
   }

    Angle calcAngleToGoal() const
   {
    return (theRobotPose.inversePose * Vector2f(theFieldDimensions.xPosOpponentGroundLine, 0.f)).angle();
   }
   };
MAKE_CARD(CodeReleaseBallInterceptor);
```