Technical University of Crete, Greece

Department of Electronic and Computer Engineering

Player Behavior and Team Strategy for the RoboCup 3D Simulation League

Georgios Methenitis

Thesis Committee

Assistant Professor Michail G. Lagoudakis (ECE)

Assistant Professor Georgios Chalkiadakis (ECE)

Professor Minos Garofalakis (ECE)

Chania, August 2012

ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΝΙΚΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Συμπεριφορά Παικτών και Στρατηγική Ομάδας για το Πρωτάθλημα RoboCup 3D Simulation

Γεώργιος Μεθενίτης

Εξεταστική Επιτροπή

Επίκουρος Καθηγητής Μιχαήλ Γ. Λαγουδάκης (ΗΜΜΥ)

Επίκουρος Καθηγητής Γεώργιος Χαλκιαδάκης (ΗΜΜΥ)

Καθηγητής Μίνως Γαροφαλάκης (ΗΜΜΥ)

Χανιά, Αύγουστος 2012

Abstract

Every team which participates in a game, requires both individual and team skills in order to be successful. We could define individual skills as the ability of each member of the team to do actions which are going to be productive for himself even if they are not close to the team's objective. On the other hand, we could define team skills as the actions of individuals, brought together for a common purpose. Each person on the team puts aside his or her individual needs to work towards the larger group objective. The interactions among the members of each team and the work they complete is called teamwork. Therefore, when we are talking about a team sport such as soccer, both individual and team skills are in a big need. For human teams, these two skills exist and be improved over time, however, for robot teams these skills are completely in absence. Robotic soccer as well as the simulated one include all the classic Artificial Intelligence's and robotics' problems such as, perception, localization, movement and coordination. Multi-agent systems in complex, real-time domains require agents to act effectively both autonomously and as parts of the team as well. This thesis addresses multi-agent systems consisting of teams of autonomous agents acting in real-time, noisy, collaborative, and competitive environments. First of all, every player in the team should percept his environment and has a reliable imaging of his surroundings. If he does, then he should be able to locate his actual position in the field which is a very important issue in robotic soccer. Nothing could be accomplished by a soccer team if players had a poor movement. For this reason, there must be stable and fast movements by the robot players, this can be prove to be crucial in a soccer game. Moreover, there are actions like a kick towards the opponents goal which combine movements and other actions in order to be executed by the agent. While these actions are vitally important in order to have a successful soccer playing agent, the agents must work together as a team and coordinate their actions maximizing the team's performance. In this thesis we describes the whole agent's framework emphasizing in the team's coordination.

Περίληψη

Κάθε ομάδα που συμμετέχει σε ένα ομαδικό παιχνίδι απαιτεί τις ατομικές ικανότητες κάθε παίκτη ξεχωριστά αλλά και την συνολική ικανότητα της σαν ομάδα ώστε να είναι πετυχημένη. Θα μπορούσαμε να χαρακτηρίσουμε τις ατομικές ικανότητες σαν ενέργειες ατόμων οι οποίες λαμβάνουν χώρα με σκοπό να γίνουν επικερδείς για την ομάδα. Από την άλλη μεριά, οι ομαδικές ικανότητες είναι ο συνδυασμός τον επιμέρους ενεργειών κάθε παίκτη που επιφέρει κέρδος στην ομάδα. Οι ενέργειες αυτές γίνονται από την πλευρά κάθε παίκτη βάζοντας στην άκρη τις προσωπικές φιλοδοξίες ή ανάγκες για την επίτευξη ενός μεγαλύτερου σκοπού. Ειδικότερα όταν μιλάμε για ένα άθλημα όπως το ποδόσφαιρο, υπάρχει η απαίτηση και των δυο παραπάνω γνωρισμάτων από όλους τους παίκτες της ομάδας. Για τις ανθρώπινες ομάδες αυτό είναι κάτι τετριμμένο που υπήρχε πάντα και συνεχώς βελτιώνεται. Αλλά όταν μιλάμε για ρομποτικές ομάδες ποδόσφαιρου όλες αυτές οι ικανότητες δεν υφίσταται. Το ρομποτικό πρωτάθλημα ποδσφαίρου όπως και αυτό της προσομοίωσης περιέχει όλα τα κλασσικά προβλήματα της τεχνητής νοημοσύνης αλλά και των ρομποτικών συστημάτων όπως η αντίληψη, το πρόβλημα του εντοπισμού, η κίνηση και η συνεργασία. Αρχικά κάθε παίκτης πρέπει να είναι ικανός να αντιλαμβάνεται το περιβάλλον του και να έχει μια αξιοπρεπή απεικόνιση των πραγμάτων που βρίσκονται γύρω από αυτό. Αν είναι ικανός να το κάνει, τότε θα πρέπει να βρίσκει την θέση του στο γήπεδο, κάτι που είναι πολύ σημαντικό στο ρομποτικό ποδόσφαιρο. Τίποτα δεν θα μπορούσε να επιτευχτεί αν δεν υπήρχε η κίνηση, πρέπει να υπάρχουν σταθερές και γρήγορες κινήσεις που θα βοηθήσουν τα μέγιστα και είναι ιδιαίτερα σημαντικές σε τέτοιου τύπου αγώνες. Τέλος, είναι η συνεργασία που επιτυγχάνεται μέσω της επικοινωνίας η άλλων ενεργειών. Οι πράκτορες -ρομπότ- πρέπει να είναι σε θέση να συνεργάζονται μεταξύ τους ώστε να μπορούν να πετυχαίνουν το καλύτερο για την ομάδα τους. Σε αυτή την διπλωματική εργασία περιγράφουμε την δημιουργία κομμάτι-κομμάτι του πλαισίου για την κάλυψη των αναγκών μιας ρομποτικής ομάδας ποδόσφαιρου για το επίσημο πρωτάθλημα προσομοίωσης Robocup, δίνοντας έμφαση στον τομέα της συνεργασίας.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

What will happen if we place a team of robots into a soccer field? It is obvious for everyone to realize that nothing is going to happen. This occurs due to the fact that machines, such as robots, should be programmed to perceive their surroundings and act just like human soccer players. Therefore, everything in the robots' world start from scratch. Even if these robots had a perfect sense of their environment, it would be difficult for them to start taking part into the game immediately. There are plenty of things that have to be done before these robots start playing in the way human players do.

A simulation soccer game consists of two parts. There is a server which has the responsibility of sending perception messages to the agents, as well as, receiving effector messages from the agents to apply them into the soccer field. The second part is the agents which are processes running independently from each other without being able to communicate directly but only with the server. In the beginning there must be a connection with the simulation server. When we ensure that we are connected with the server, we are ready to proceed to the next steps. Server sends to each connected agent messages every 20ms, these messages include information about agent's vision and other perceptions. Each agent parses these messages to update his perceptions, At the end of the parsing the agent knows the values of every joint of his body, he has also knowledge about the location in relation to his body of every landmark, the ball and other players which are in the field of his view and finally possible messages from teammates. Now, agent is ready to continue to the main procedure of thinking. First of all, agent has to calculate his position in the soccer field, it is not so simple as it sounds and it requires at least two landmarks in the field of our view. We are going to explain this operation extensively later. Even if, our agent knows his positions in the soccer field and is able to calculate the position of every other agent in his sight, as well as, the soccer ball position, he is still not able to perform a single action. This will be feasible if he combines motions which are going to help him perform each action. Even in real life, a human soccer player has to combine simple movements for example, walking, turning and kicking, to perform a kick towards the opponents' goal. The same principle applies in simulation soccer too. In our approach, we have categorize the actions in relation to their complexity. At first simple actions, which just use motions in order to be completed. We continue with more complex actions which make use of more than one simple actions to be executed by the agent with success. An example of a simple action is a turn towards the ball and a more complex action could be walking to a specific coordinate in the soccer field. We can realize that a complex action such as the above is going to make use of more than one simple actions and movements. Until now, we have accomplished every agent in the field to be able to recognize objects, find its position and do simple and complex actions. Returning to the first question which we have put in the beginning of this introduction, we could answer with certainty that every agent in the soccer field now has a complete sense of its surroundings and is able to perform actions which are able to make changes in his environment. Even so, these improvements are not going to bring success to the team, agents have not the ability to communicate with their team-mates and reasonably they are not able to coordinate their actions. Even humans since the advent of their history form all kinds of groups striving to achieve a common goal, especially , for teams participating in games, where success can only be achieved through collaborative and coordinated efforts. As we realize, coordination and cooperation are the last pieces of the puzzle. This two team skills are going to be accomplished through communication process. This thesis as well as a proposed solution of all the problems generated in robotic soccer. The main objective is to develop an efficient software system to correctly model the behaviors of simulated Nao robots in such a competitive environment as the simulation soccer league. Additionally, we are coming up with an approach in which agents coordinate through the communication channel their actions which will be calculated to be costless and worthy for the team. The challenging and the most time consuming part of this project was the coordination part which I firmly believe is a skill of major importance either in a simulated team or in a real soccer team.

1.1 Thesis Outline

Chapter 2 provides some background information on the RoboCup competition. In Chapter 3 we present the SimSpark simulation platform and the general framework of the challenging RoboCup 3D Simulation League domain we are going to work on. Continuing to Chapter 4, the core ideas, the architecture of our agents, and the individual player behavior are presented. Moving on to Chapter 5, we present in detail our team strategy and our coordination method over a custom communication protocol among the team players. In Chapter 6 the results and the evaluation of our work are presented though several experiments and test games. Chapter 7 presents similar systems developed by other RoboCup teams including a brief comparison between those systems and ours. Finally, Chapter 8 serves as an epilogue to this thesis, including proposals on extending and improving our framework and a discussion about the experience we gained from this work.

Chapter 2

The RoboCup Competition

RoboCup is an international robotics competition founded in 1997. The aim is to promote robotics, artificial intelligence, and machine learning research by offering a publicly appealing, but formidable, challenge. The name *RoboCup* is a contraction of the competition's full name, "Robot Soccer World Cup". The official goal of the project is stated as an ambitious endeavor: "By the year 2050, a team of fully autonomous humanoid robot soccer players shall win the soccer game, complying with the official rule of the FIFA, against the winner of the most recent World Cup" [2]. This endeavor may seem impossible with today's technology. I would say that a more realistic goal would be to make a team of robots play soccer similarly, but not necessarily better, than humans. In any case, the true goal is to push research efforts towards technological breakthroughs and will be the one of the grand challenges shared by robotics and AI community for next 50 years. RoboCup is an annual event in which lots of research teams around the world participate in various leagues including RoboCupSoccer, RoboCup@Home, RoboCupRescue, and RoboCupJunior, each of these leagues and its sub-leagues will be presented extensively below. Participation in this annual event is growing year by year reaching in a number of more than 400 teams from 40 countries around the world in RoboCup 2011 which held in Istanbul, Turkey. The RoboCup competitions provide an excellent channel for the dissemination and validation of innovative concepts and approaches for autonomous robots and multi-robot systems under very challenging and adverse conditions.

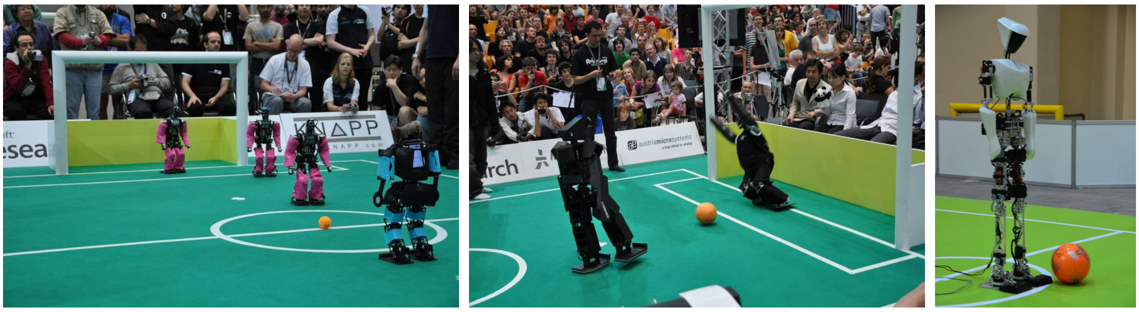

Figure 2.1: Humanoid Kid, Teen, Adult -Size League at Robocup 2011.

2.1 RoboCup Soccer

The main focus of the RoboCup competitions is the game of soccer, where the research goals concern cooperative multi-robot systems in dynamic adversarial environments. All robots in this league are fully autonomous. A competition which gives the possibility of doing research in a more entertaining way. Moreover, we can realize that soccer is selected due to the fact that it is a popular sport and widely spread throughout the world. Moreover, rules governing it are also known. Robocup Soccer encompasses all the known problems of artificial intelligence such as machine vision, perception, behavior and cooperation which are of paramount importance in multi-agent environments like this.

Humanoid League

In the Humanoid League, autonomous robots with a human-like body and human-like senses play soccer against each other. Dynamic walking, running, and kicking the ball while maintaining balance, visual perception of the ball, other players, and the field, self-localization, and team play are among the many research issues investigated in the league. The league is divided into 3 subleagues, according to robot sizes: Teen, Kid and Adult Size. Figure 2.1 shows these three size leagues.

Middle-Size League

Middle-sized wheeled robots of no more than 50 cm diameter play soccer in teams of up to six robots with regular size FIFA soccer ball on a field similar to a scaled human soccer field. All sensors are on-board. Robots can use wireless networking to communicate.

Figure 2.2: Middle-Size League at RoboCup 2011.

The research focus is on full autonomy and cooperation at plan and perception levels. Figure 2.2 shows a Middle-Size League's game at RoboCup 2011.

Simulation League

This is one of the oldest leagues in RoboCup's Soccer. The Simulation League focus on artificial intelligence and team strategy. Independently moving software players (agents) play soccer on a virtual field inside a computer. There are two subleagues: 2D and 3D. Simulation league 3D is going to be presented extensively in the next chapter. Figure 2.3 shows how the 2D versus 3D simulation league looks like.

Small-Size League

The Small Size league or F180 league as it is otherwise known, is one of the oldest RoboCup Soccer leagues. It focuses on the problem of intelligent multi-robot/agent cooperation and control in a highly dynamic environment with a hybrid centralized/distributed system. The robot must fit within an 180mm diameter circle and must be no higher than 15cm. The robots play soccer with an orange golf ball on a green carpeted field that

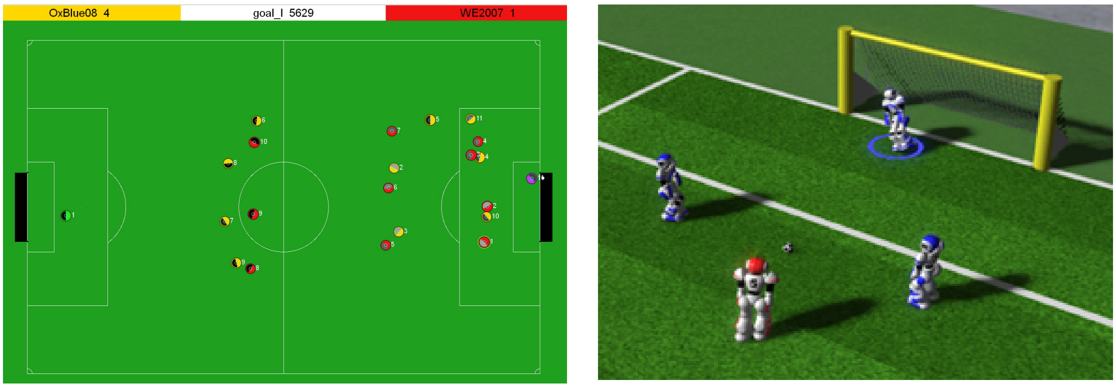

Figure 2.3: Simulation League 2D (left) and 3D (right).

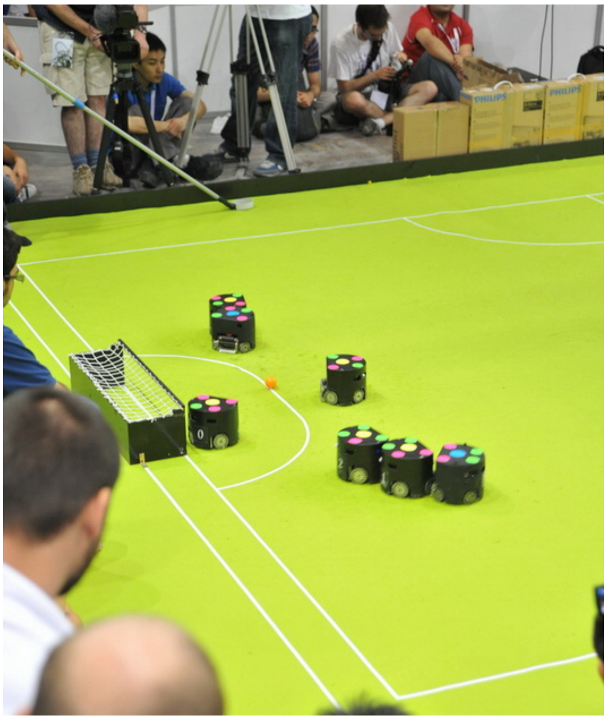

Figure 2.4: Small-Size League at RoboCup 2011.

is 6.05m long by 4.05m wide. All objects on the field are tracked by a standardized vision system that processes the data provided by two cameras that are attached to a camera bar located 4m above the playing surface. The vision system called SSL-Vision. Figure 2.4 shows a game during RoboCup competition in Istanbul, 2011.

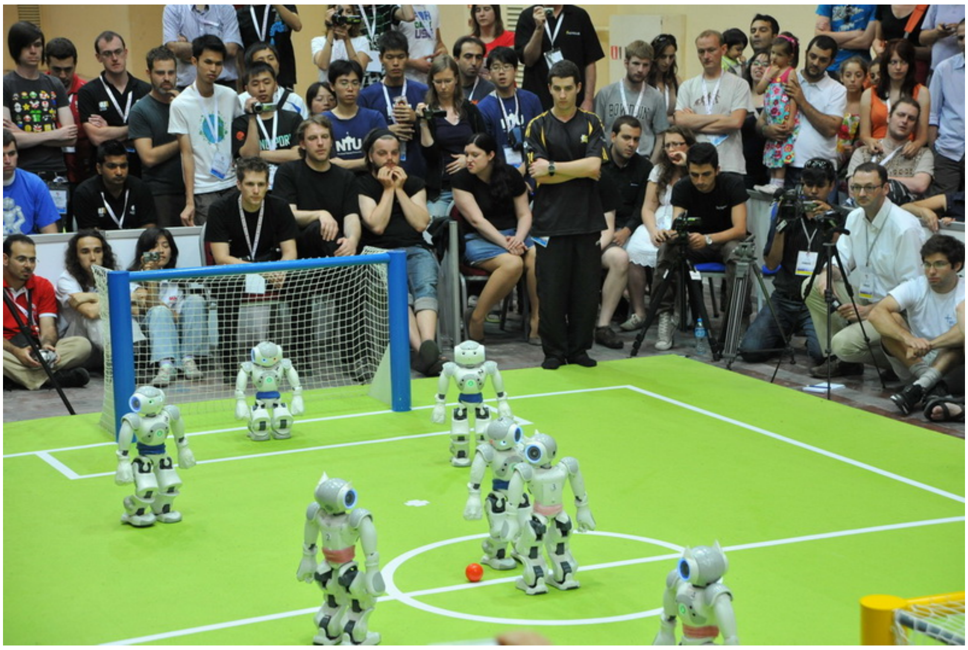

Figure 2.5: Standard Platform League at RoboCup 2011.

Standard Platform League

In this league all teams use same robots. Therefore, the teams concentrate on software development only, while still using state-of-the-art robots. Directional vision forces decision-making to trade vision resources for self-localization and ball localization. The league is based on Aldebaran'As Nao humanoids. Team “Kouretes” [www.kouretes.gr] from the Technical University of Crete is the only Greek representative in this league, having continuous participation since 2006 and several distinctions. Figure 2.5 shows a highlight of the Standard Platform League.

2.2 RoboCup Rescue

The intention of the RoboCup Rescue project is to promote research and development in this socially significant domain at various levels involving multi-agent team work coordination, physical robotic agents for search and rescue, information infrastructures, personal digital assistants, a standard simulator and decision support systems, evaluation benchmarks for rescue strategies and robotic systems that are all integrated into a

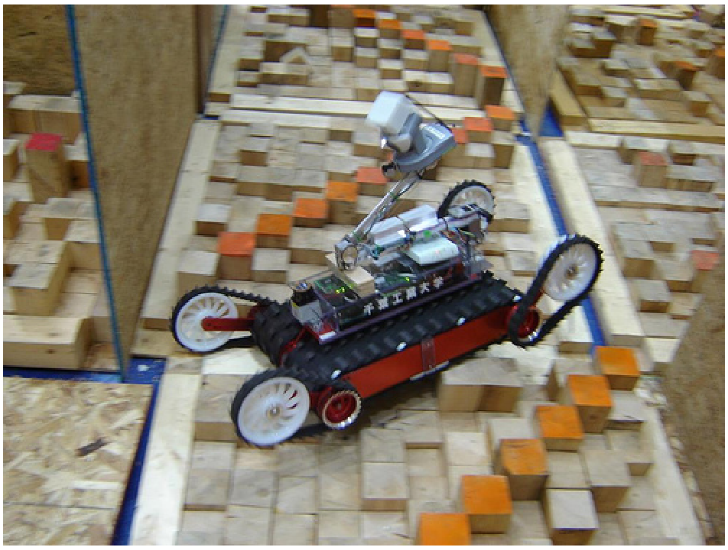

Figure 2.6: Rescue Robot League at RoboCup.

comprehensive systems in future.

Robot League

The goal of the urban search and rescue (USAR) robot competitions is to increase awareness of the challenges involved in search and rescue applications, provide objective evaluation of robotic implementations in representative environments, and promote collaboration between researchers. It requires robots to demonstrate their capabilities in mobility, sensory perception, planning, mapping, and practical operator interfaces, while searching for simulated victims in unstructured environments. The RoboCupRescue arenas constructed to host these competitions consist of emerging standard test methods for emergency response robots developed by the U.S. National Institute of Standards and Technology through the ASTM International Committee on Homeland Security Applications; Operational Equipment; Robots (E54.08.01). The competition field is divided into color-coded arenas that form a continuum of challenges with increasing levels of difficulty for robots and operators and highlight certain robotic capabilities. Greece participates in this league (RoboCup 2008, 2009, 2011) with team "P.A.N.D.O.R.A" [pandora.ee.auth.gr] based at the Aristotle University of Thessaloniki. Figure 2.6 shows a wheeled robot in RoboCupRescue robot competition.

Figure 2.7: Rescue Simulation League at RoboCup 2006.

Simulation League

The purpose of the RoboCup Rescue Simulation league is twofold. First, it aims to develop simulators that form the infrastructure of the simulation system and emulate realistic phenomena predominant in disasters. Second, it aims to develop intelligent agents and robots that are given the capabilities of the main actors in a disaster response scenario. The Virtual Robots Competition aims to be the meeting point between researchers involved in the Agents Competition and those active in the RoboCupRescue League. It is based on USARSim, a high fidelity simulator based on the UnrealTournament game engine. USARSim currently features wheeled, tracked and legged robots, as well as a wide range of sensors and actuators. Moreover, users can easily develop models of new robotic platforms, sensors and test environments. Validation experiments have shown close correlation between results obtained within USARSim and the corresponding real robots. Figure 2.7 shows a wheeled robot in RoboCupRescue robot competition.

Figure 2.8: RoboCup@Home at RoboCup 2011.

2.3 RoboCup@Home

The RoboCup@Home league aims to develop service and assistive robot technology with high relevance to future personal domestic applications. It is the largest international annual competition for autonomous service robots and is part of the RoboCup initiative. A set of benchmark tests is used to evaluate the robots' abilities and performance in a realistic non-standardized home environment setting. Focus lies on, but is not limited to, the following domains: Human-Robot Interaction and Cooperation, Navigation and Mapping in Dynamic Environments, Computer Vision and Object Recognition under Natural Light Conditions, Object Manipulation, Adaptive Behaviors, Behavior Integration, Ambient Intelligence, Standardization and System Integration.

2.4 RoboCup Junior

RoboCupJunior is a project-oriented educational initiative that sponsors local, regional and international robotic events for young students. It is designed to introduce RoboCup

Figure 2.9: Junior Soccer League at RoboCup 2011.

to primary and secondary school children, as well as undergraduates who do not have the resources to get involved in the senior leagues yet.

Soccer

2-on-2 teams of autonomous mobile robots play in a dynamic environment, tracking a special light-emitting ball in an enclosed, landmarked field. Figure 2.9 shows Junior Soccer League at RoboCup 2011.

Dance

One or more robots join human dancers and give a dance performance dressed in costume and moving in creative harmony.

Rescue

Robots identify simulated victims within re-created disaster scenarios, varying in complexity from line-following on a flat surface to negotiating paths through obstacles on uneven terrain.

Chapter 3

RoboCup 3D Simulation League

The 3D Simulation League [3] increases the realism of the simulated environment used in the 2D Simulation League by adding an extra dimension and more complex physics. At its beginning, the only available robot model was a spherical agent. In 2006, a simple model of the Fujitsu HOAP-2 robot was made available, being the first time that humanoid models were used in the simulation league. This shifted the aim of the 3D Simulation League from the design of strategic behaviors in playing soccer towards some low-level control of humanoid robots and the creation of basic behaviors, like walking, kicking, turning and standing up, among others.

In 2008, the introduction of a Nao robot model to the simulation gave another perspective to the league. The real Nao robot from Aldebaran robotics has been the official robot for the Standard Platform League since 2008. Using the same model for the simulation competitions represents a great opportunity for researchers wanting to test their algorithms and ideas before trying them into the real robots. The interest in the 3D Simulation League is growing fast and research is slowly getting back to the design and implementation of multi-agent higher-level behaviors based on solid low-level behavior architectures for realistic humanoid robot teams. SimSpark is used as the official Robocup 3D simulator.

3.1 SimSpark Soccer Simulator

SimSpark [4] is a generic physics simulator system for multiple agents in three-dimensional environments. It builds on the flexible Spark application framework. In comparison to

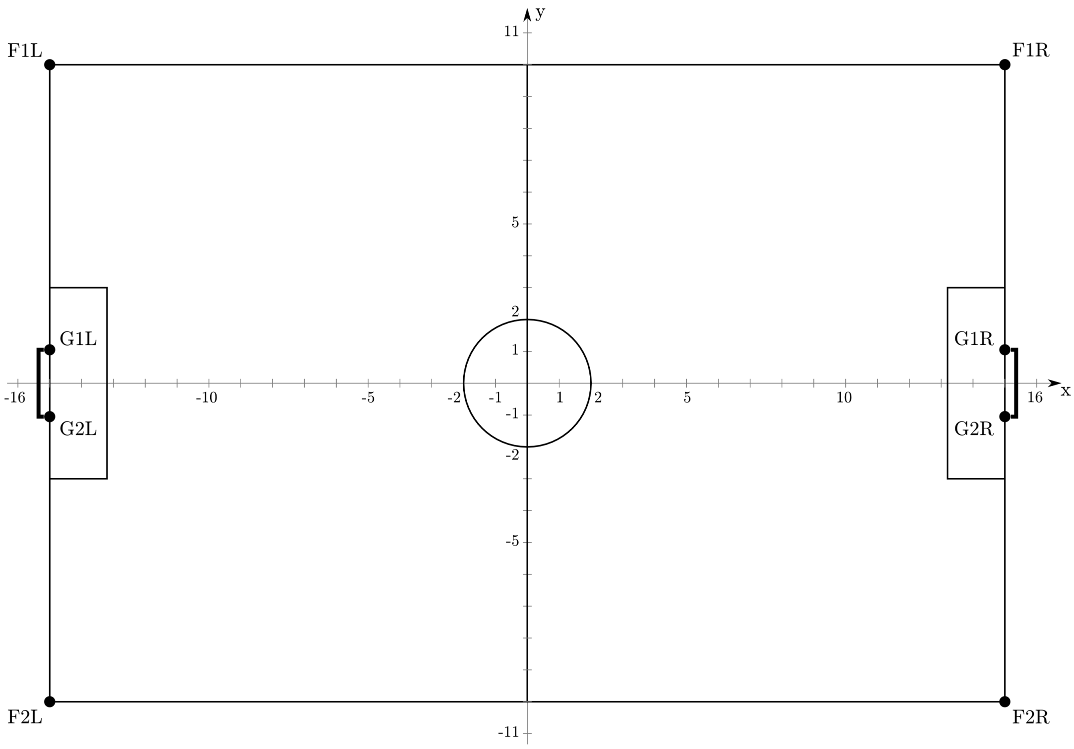

Figure 3.1: RoboCup 3D Simulation League Field

specialized simulators, users can create new simulations by using a scene description language. SimSpark is a powerful tool to study different multi-agent research questions.

Rcssserver3d is the official competition environment for the RoboCup 3D Simulation League. It implements a simulated soccer environment, whereby two teams of up to nine, and in the latest version up to eleven, humanoid robots play against each other. Figure 3.1 shows the dimensions and the layout of the simulated soccer field.

3.2 Robot Model

Rcssserver3d comes with the Nao robot model for use by the agents in the soccer simulation. The physical specifications of each model is stored in an `.rsg` file.The real Nao humanoid robot is manufactured by Aldebaran Robotics in Paris, France. Its height is about 57cm and its weight is around 4.5kg. The simulated model comes with 22 degrees of freedom, which allow Nao to have great mobility. Although, we are discussing about

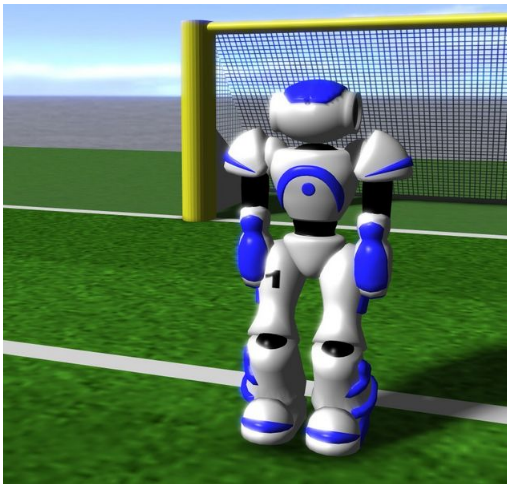

Figure 3.2: Nao Robot Model in the Rcssserver3d Soccer Simulation Environment

the same robot, there are significant differences between the real and the simulated Nao robot. The real Nao comes with two cameras attached to its head in vertical alignment; these two cameras cannot operate simultaneously, but only one at a time. It also has two sonar devices on the chest, which are completely absent from the simulated Nao. The real Nao comes with 21 degrees of freedom in contrast to 22 of the simulated Nao, because the two pelvis joints are coupled together and are not able to move independently. In addition, two bumpers are positioned in front of the feet of the real Nao, providing information about possible collisions; these are absent in the simulated model. Figure 3.2 shows a depiction of the Nao robot model within the into the Rcssserver3d simulation environment along with the real one.

3.3 Server

The SimSpark server hosts the process that manages and advances the simulation. The simulation state is constantly modified through the simulation update loop. Each simulation step corresponds to 20ms of simulated time. Objects in the scene change their state, i.e. one or more of their properties, such as position, speed, angular velocity, etc., due to inherent or external influences. These properties are under the control of a rigid body physical simulation that resolves collisions, applies drag, gravity, etc. Agents that take part in the simulation also modify objects with the help of their effectors, which may move and apply forces to other objects. SimSpark implements a simple internal event model that immediately executes every action received from an agent.

Another responsibility of the server is to keep track of connected agent processes. The SimSpark server exposes a network interface to all agents on TCP port 3100 (default value). In each simulation cycle, the server collects and reports sensor information for each of the sensors of all connected agents. It further carries out received action sequences triggered by the connected agents through their available effectors. The server does not try to compensate for network latencies or differences in computing resources available to the connected agents. A consequence is that simulations are not reproducible. This means repeated simulations may have a different outcome, depending on network delays or load variations on the machines hosting the agents and the server.

3.4 Monitor

The server can render the simulation itself, depending on its configuration. It implements an internal monitor that omits the network overhead. However, it supports streaming data to remote monitor processes, which take responsibility for rendering the 3D scene for remote viewing.

3.4.1 SimSpark Monitor

The SimSpark monitor is responsible for rendering the current simulation. It connects to a running server instance from which it continuously receives a stream of updates that describe the simulation state, either as full snapshots or as incremental updates. The format of the data stream the server sends to the monitor is called *Monitor Format*. It is a customizable language used to describe the simulation state in text format. Apart from describing the pure simulation state, each Monitor Format may provide a mechanism to transfer additional game-specific state. For the soccer simulation, this game-specific state may include, for example, current play mode and goals scored so far. The monitor client itself only renders the pure scene and defers the rendering of the game-specific state to plugins. These plugins are intended to parse the game-specific state and display it as an overlay printed out on screen.

Figure 3.3: Roboviz (left) vs SimSpark (right) Monitors.

3.4.2 Roboviz Monitor

RoboViz [5] was created by Justin Stoecker in collaboration with the RoboCup group (RoboCanes) at the University of Miami's Department of Computer Science. RoboViz is a software program designed to assess and debug agent behaviors in the RoboCup 3D Simulation League. RoboViz is an interactive monitor that renders agent and world state information in a three-dimensional scene. In addition, RoboViz provides programmable drawing and debug functionality to agents that can communicate over a network. The tool facilitates the real-time visualization of agents running concurrently on the SimSpark simulator and provides higher-level analysis and visualization of agent behaviors not currently possible with existing tools. Figure 3.3 shows a visual comparison of the RoboViz and SimSpark Monitors.

3.5 Perceptors

Perceptors are the senses of an agent, allowing awareness of the agent's model state and the environment. The server sends perceptor messages to connected agents via the network protocol at each cycle of the simulation. There are both general perceptors available in all simulations and soccer perceptors specific to the soccer simulation.

3.5.1 General perceptors

HingeJoint Perceptor A hinge joint perceptor receives information about the angle of the corresponding single-axis hinge joint. It contains the identifier HJ, the name of the perceptor, and the position angle of the axis in degrees. A zero angle corresponds to straightly aligned bodies. The position angle of each hinge joint perceptor is sent at each cycle. Each hinge joint has minimum and maximum limits on its angular position. This varies from hinge to hinge and depends upon the model being used. Nao has 22 hinge joint perceptors; this is the only joint type used in this robot.

Message format: (HJ (n <name>) (ax <ax>))

Frequency: Every cycle

Noise Model: None, however values are truncated to two decimal places, which is equivalent to a uniform error of up to 0.01 degrees.

ForceResistance Perceptor This perceptor informs about the force that acts on a body. After the identifier FRP and the name of the body, the perceptor message contains two three-dimensional vectors. The first vector describes the coordinates of the point of origin on the body where the force is applied to (in meters) and the second vector is the force vector (magnitude and direction) of the force applied to this point (in Newtons). This information is just an approximation of the real applied force. The point of origin is calculated as the weighted average of all contact points to which force is applied, while the force vector represents the total force applied to all of these contact points. The perceptor message for force resistance perceptors is sent only in case of a collision of the corresponding body with another simulated object. Nao has two of these perceptors, located below each foot and named lf and rf.

Message format: (FRP (n <name>) (c <px> <py> <pz>) (f <fx> <fy> <fz>))

Frequency: Only in cycles where a body collision occurs

Noise Model: None, however values are truncated to two decimal places, which is equivalent to a uniform error of up to 0.01 meters or Newtons.

GyroRate Perceptor The gyro rate perceptor delivers information about the change in orientation of a body. The message contains the `GYR` identifier, the name of the body to which the gyro perceptor belongs to, and the rates of change of the three rotation (Euler) angles. These values describe the rates of change in orientation of the body during the last cycle, in other words the current angular velocities about the three rotation axes of the corresponding body in degrees per second. To enable keeping track of the orientation of the body, the information to each gyro rate perceptor is sent at each cycle. Nao has one gyro perceptor in the upper torso.

Message format: `(GYR (n <name>) (rt <x> <y> <z>))`

Frequency: Every cycle

Noise Model: None, however values are truncated to two decimal places, which is equivalent to a uniform error of up to 0.01 degrees.

Accelerometer Perceptor This perceptor measures the proper acceleration a body experiences relative to free fall. As a consequence an accelerometer at rest relative to the simulated earth's surface will indicate an acceleration of approximately 1g upwards. To obtain the acceleration due to motion with respect to the earth, this gravity offset should be subtracted. After the identifier `ACC` and the name of the body, the perceptor message contains a three-dimensional vector with the acceleration values along the three Cartesian axes in m/s^2. Nao has one accelerometer in the upper torso.

Message format: `(ACC (n <name>) (a <x> <y> <z>))`

Frequency: Every cycle

Noise Model: None, however values are truncated to two decimal places, which is equivalent to a uniform error of up to 0.01 m/s^2.

3.5.2 Soccer perceptors

Vision Perceptor The most important perceptor of the Nao robot is the vision perceptor, which delivers information about seen objects in the environment, where objects are either others players, the ball, field lines, or markers on the field. Currently there are eight markers on the field: one at each corner point of the field and one at each goal post. Each player has up to five visible body parts (two arms, two legs, head). Each field line is characterized by two points (starting and ending points). The perceptor message begins with the identifier `See` and for each visible object it contains a vector described in spherical coordinates. In other words, it contains the distance `d` (in meters) together with the horizontal `a1` and vertical `a2` angles (in degrees) to the center of the object relatively to the focal point of the camera. Nao possesses a restricted vision perceptor at the center of its head. This perceptor's type is `RestrictedVisionPerceptor`, which limits the field of view to 120°.

Message format:
```
(See +(<name> (pol <d> <a1> <a2>))
     +(P (team <name>) (id <ID>) +(<bodypart> (pol <d> <a1> <a2>)))
     +(L (pol <d> <a1> <a2>)(pol <d> <a1> <a2>)) )
```

Frequency: Every third cycle (60ms)

Noise Model: Calibration error (a fixed offset of around $\pm 0.004m$ in each of $x/y/z$ axes), zero-mean Gaussian noise, and values truncated to two decimal places, which is equivalent to a uniform error of up to 0.01 meters or degrees.

Hear Perceptor The agent processes are not allowed to communicate with each other directly, but the agents may exchange messages via the simulation server. For this purpose agents are equipped with the so-called hear perceptor, which serves as an aural sensor and receives messages shouted by other players. A hear perceptor message begins with the `hear` identifier, followed by the simulation time at which the given message was heard in seconds, either a relative horizontal direction in degrees indicating where the sound originated or `self` indicating that the player is hearing their own shouted message, and finally the message itself in plain ASCII text (parentheses cannot be part of the message). Messages should not have a length of more than 20 ASCII characters. Messages shouted from beyond a maximal distance (currently 50 meters) cannot be heard. Most important restriction is that only one message can be heard at any given time and messages from the same team can be heard only every other cycle. All unheard messages are lost. Thus, the maximum communication bandwidth is 20 ASCII characters every $40ms$.

Message format: `(hear <time> self/<direction> <message>)`

Frequency: Only in cycles, where a message is heard

GameState Perceptor The game state perceptor delivers information about the actual state of the soccer game environment. A game state message begins with the `GS` identifier, followed by two pieces of game state information: the actual play time and the current play mode (`BeforeKickOff`, `PlayOn`, `KickOff_Left`, `KickOff_Right`, `GoalLeft`, `GoalRight`, `corner_kick_left`, `corner_kick_right`, `KickInLeft`, `KickInRight`, `goal_kick_left`, `goal_kick_right`). Play time starts at 0 at the kickoff of the first half and at 300 at the kickoff of the second half and is given in seconds with a precision of two decimal places.

Message format: `(GS (t <time>) (pm <playmode>))`

Frequency: Every cycle

3.6 Effectors

Effectors allow agents to perform actions within the simulation. Agents control them by sending messages to the server and the server changes the game state accordingly. Effector control messages are sent via the network protocol. There are both general effectors that apply to all simulations, and soccer effectors that are specific to the soccer simulation.

3.6.1 General Effectors

Create Effector When an agent initially connects to the server, it is invisible and cannot affect a simulation in any meaningful way. It only possesses a so-called CreateEffector, whose message begins with the `scene` identifier. An agent uses this effector to advice the server to construct the physical representation and all further effectors and perceptors of the agent in the simulation environment according to a scene description file it passes as a parameter.

Message format: `(scene <filename>)`

Frequency: Only once

HingeJoint Effector Effector for all axes with a single degree of freedom. The first parameter is the name of the axis. The second parameter is a speed value given in degrees per second. Setting a speed value on a hinge means that the speed will be maintained until a new value is provided. Even if the hinge meets its extremity, it will bounce around the extremity until a new speed value is requested.

Message format: `(<name> <ax>)`

Frequency: Once per cycle maximum

Synchronize Effector Agents running in Agent Sync Mode must send this command at the end of each simulation cycle. Note that the server ignores this command, if it is received in Real-Time Mode, so it is safe to configure agents to always append this command to responses.

Message format: `(syn)`

Frequency: Every cycle

3.6.2 Soccer Effectors

Init Effector The init command is sent once for each agent, after the create effector message has been sent. The init effector registers the agent as a member of a team with a specific player number, both of which are passed as arguments in the init effector message after the identifier `init`. All players of one team must use the same team name and different player numbers. When an agent connects to the server, he must first send a `CreateEffector` message followed by an `InitEffector` message in order to initialize himself into the soccer field.

Message format: `(init (unum <playernumber>) (teamname <teamname>) )`

Frequency: Only once

Beam Effector The beam effector allows a player to position itself anywhere on the field only before any kick-off (at the start of each half or right after a goal has been scored). After the `beam` identifier, the `x` and `y` coordinates define the position on the field with respect to the field's coordinate system in meters, where $(0,0)$ is the absolute center of the field. The `rot` argument specifies the facing angle of the player in degrees. A value of 0 points towards the positive x-axis, whereas a value of 90 points to positive y-axis.

Message format: `(beam <x> <y> <rot>)`

Frequency: Once before each kick-off

Say Effector The say effector permits communication among agents by broadcasting messages in plain ASCII text (20 characters maximum). In order to say something, the following command has to be employed.

Message format: `(say <message>)`

Frequency: Once per cycle maximum

Chapter 4

Player Skills

In this chapter we are going to present the main functions that are necessary for the agent to be functional in the field. Every part of the agent's software and supported player skills will be extensively described below.

4.1 Agent Architecture

Before examining each individual skill of our players, it is important to describe the general architecture of our agents, which is shown in Figure 4.1. The Soccer Simulation Server (*rcssserver3d*) is responsible for communicating perceptor messages to the agent. The Connection component handles this connection between the agent and the server. These messages are handled by a string parser, which stores the incoming observations in various data structures. Consequently, the functions that require these new observations to update the agent's Beliefs are now ready to proceed. Self-localization of the agent into the field or a check if the agent has fallen on the ground are few of those belief updates. Behavior is a major component of any agent. The agent has to combine all the available knowledge and beliefs about the world state and act properly. Behavior is the function which takes as input argument the agent's beliefs about the world state and computes an action to be executed by the agent as output. The Coordination component is responsible for assigning roles to different agents implementing the strategy of team. Communication and Motion are responsible for handling agent's requests for sending messages to teammates and executing movements respectively. These two components

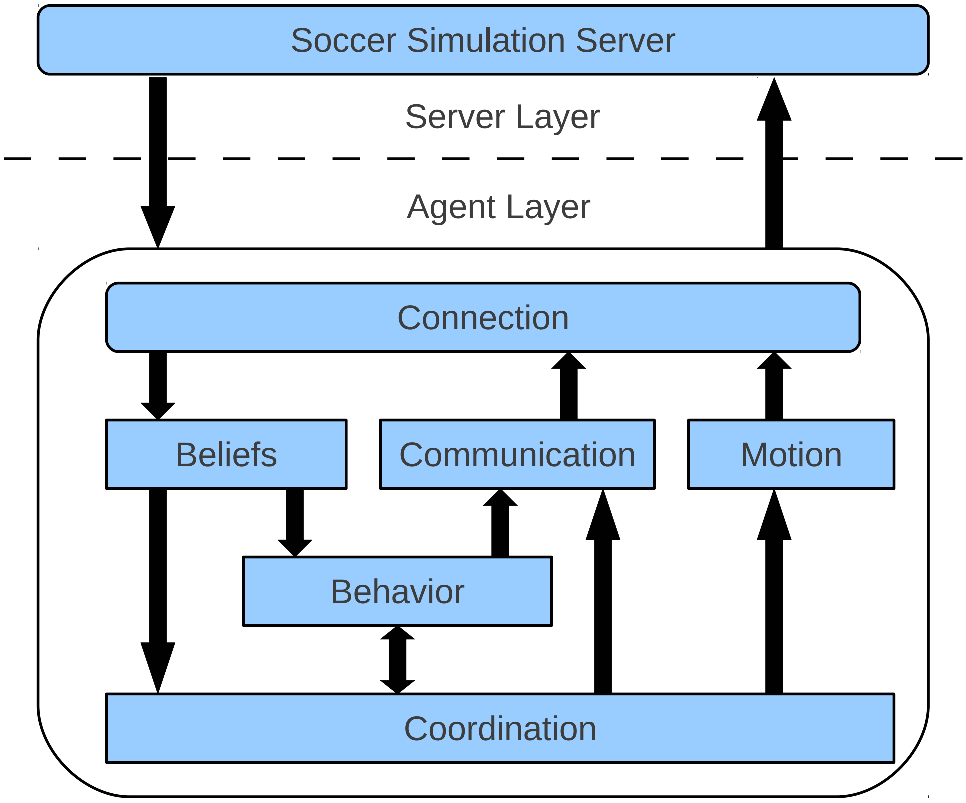

Figure 4.1: The Agent Architecture.

send effector messages to the Connection component in each cycle, if necessary, and these messages are relayed to the soccer simulation server.

4.2 Connection

The SimSpark server hosts the simulation process that manages the soccer simulation. It is responsible for advancing the game from each cycle to the next. So, it is obligatory for each agent to be connected to the server at all times during a simulated game. Agents receive sense messages from the server every 20ms at the beginning of each simulation cycle; these messages include information about all agent's perceptions. Agents willing to send action messages, can do so at the end of their think cycles, which may or may not coincide with the simulation cycles. If these two cycles coincide, the server is going to receive the action message at the same time it sends the next sense message. Figure 4.2

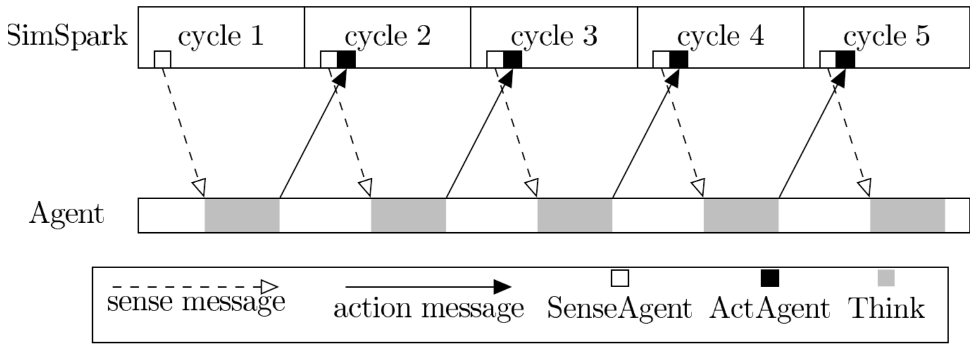

Figure 4.2: Server and Agent Communication.

shows how the communication between server and agent takes place over consecutive cycles.

4.3 Perceptions

Perceptions in simulated soccer are quite different compared to those in real soccer games. Agents do not have to process raw data coming directly from sensors, but rather listen to sensor and higher-level observation messages sent by the server at each cycle. These messages take the following form:

```
(time (now 46.20))(GS (t 0.00) (pm BeforeKickOff))(GYR (n torso)
(rt 0.00 0.00 0.00))(ACC (n torso) (a 0.00 -0.00 9.81))(HJ (n hj
1)(ax 0.00))(HJ (n hj2) (ax 0.01))(See (G2R (pol 14.83 -11.81 1.
08))(G1R (pol 14.54 -3.66 1.12)) (F1R (pol 15.36 19.12 -1.91))(F
2R (pol 17.07 -31.86 -1.83)) (B (pol 4.51 -26.40 -6.15)) (P (tea
m AST_3D)(id 8)(rlowerarm (pol 0.18 -35.78 -21.65)) (llowerarm (
pol 0.19 34.94-21.49)))(L (pol 8.01 -60.03 -3.87) (pol 6.42 51.1
90 -39.13 -5.17))(L (pol 5.91 -39.06 -5.11) (pol 6.28-29.26 -4.8
8)) (L (pol 6.28 29.34 -4.95)(pol 6.16 -19.05 -5.00)))(HJ(n raj1
) (ax -0.01))(HJ (n raj2) (ax -0.00))(HJ (n raj3)(ax -0.00))(HJ(
n raj4) (ax 0.00))(HJ (n laj1) (ax 0.01))(HJ (n laj2) (ax 0.00)) ...
```

The above message is an example message our agent has received from the simulation server during simulation. It includes information about the server time, the game state

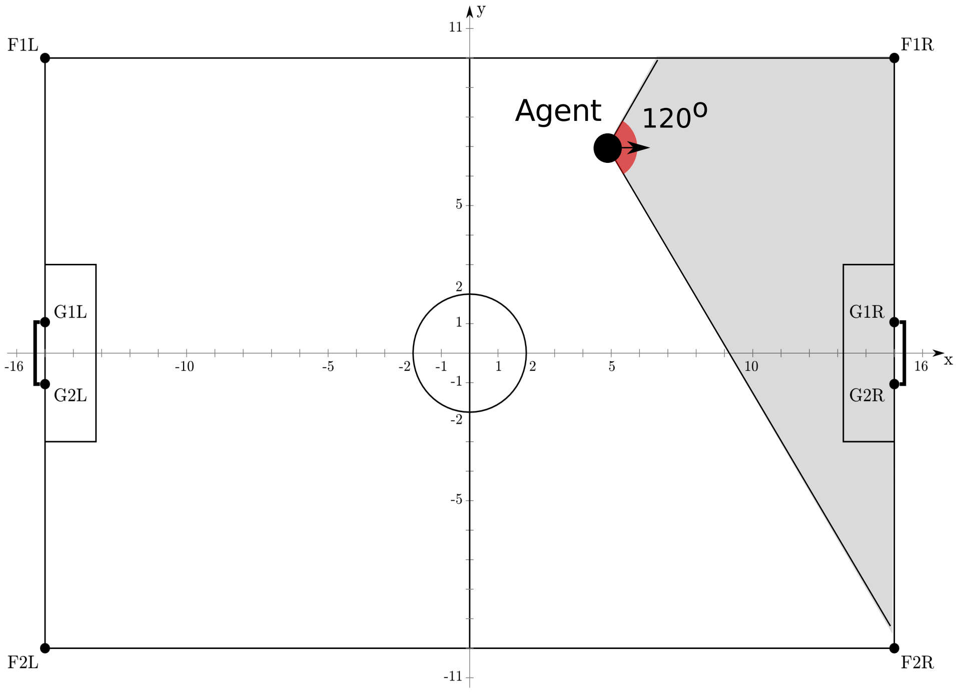

Figure 4.3: Nao's Restricted Field of View and Field Landmarks.

and time, values for each one of the joints, visual observations from the camera, and data from acceleration, gyroscope, and force sensors. We parse these messages and save the enclosed information in data structures appropriate for each type of perception.

4.4 Localization

Once we have all the new perceptions from the server available, we can update our agents' belief about its own current location in the field and the current location of other objects of interest (ball, teammates, opponents). Our localization scheme is largely based on a method proposed by a colleague within a common course project [6]. Localization, as a process, is executed every three cycles (60ms), in fact every time we receive observations from the vision perceptor.

A key restrictive factor is that the agents are equipped with a restricted vision perceptor which limits the field of their view to 120 degrees. An example of this limitation is shown in Figure 4.3. It is easy to realize that the localization process would have been easier, if there was an omni-directional vision perceptor.

4.4.1 Self Localization

The potentially visible objects in our current field of view may be of different types: ball, landmarks, teammates, and opponents. After registering all currently visible objects, we first use only the landmarks, which are located at permanent known positions in the field, to derive candidate self-locations and update the agent's belief about the current position and orientation in the field. There are eight landmarks in the field, shown in Figure 4.3: the four field corners (F1R, F2R, F1L, F2L) and the goalposts of the two goals (G1R, G2R, G1L, G2L). These eight landmarks cannot be all visible simultaneously at the same time; in general, the number of visible landmarks at any time will range from zero to four. For example, given the current location of the agent in Figure 4.3, there are only three visible landmarks: F1R, G1R, G2R.

The self-localization function takes the distance, as well as the horizontal and vertical angles of two currently visible landmarks as input and returns a candidate self location (x, y, θ) as output, where (x, y) are the coordinates in the field and θ is the orientation of the body with respect to the angle system of the field. These two landmarks form two circles with radius equal to the observed distance to each one of them centered at the static and known coordinates of these landmarks. Obviously, these two circles intersect at two points, which represent two candidate self locations. We reject the wrong candidate using the horizontal viewing angles of the two landmarks and also the fact that the correct candidate cannot be way outside the field. Figure 4.4 shows a typical example of self localization with two landmarks. In cases where the agent sees more than two landmarks, the self-localization function for each pair of landmarks. The final estimated location is computed as the average of the outcomes for all pairs. Apparently, when the agent has less than two visible landmarks in the field of view, self localization does no take place.

4.4.2 Object Localization

Apart from self localization, it is important to be able to compute the position of other visible objects, such as other players and the ball. Knowing our own location in the field helps us locate other visible objects too. For each currently visible object, the vision perceptor informs us about its vertical and horizontal angles and its distance from

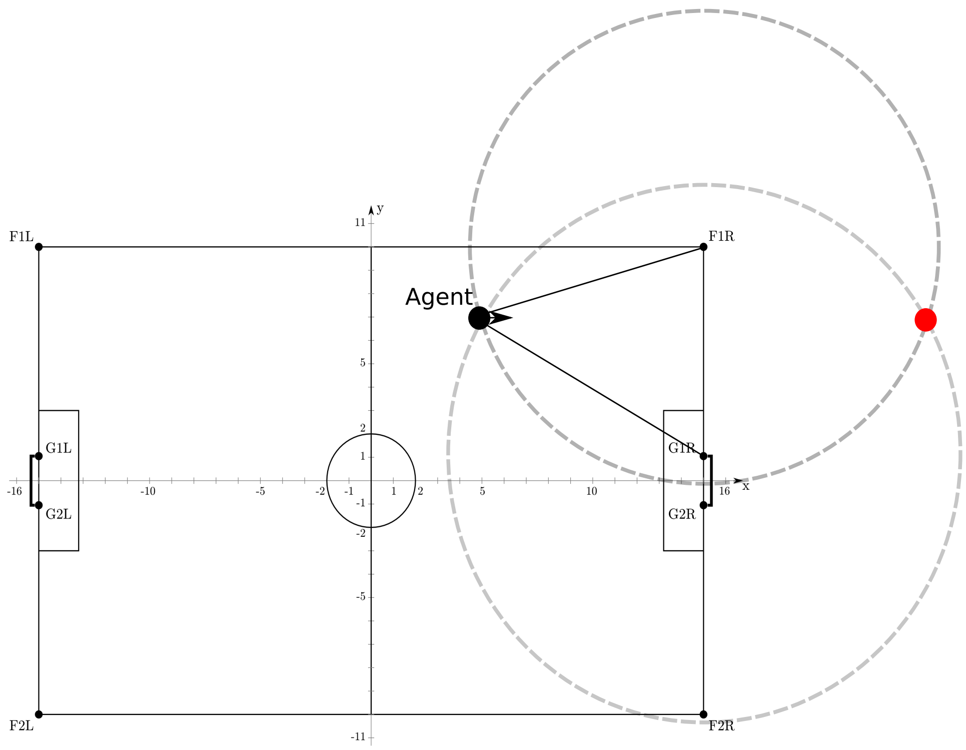

Figure 4.4: Self Localization Example.

the agent. This information is sufficient for the calculation of their exact positions into the field's coordinate system. Figure 4.5 shows an example scenario of calculating the positions of the ball and an opponent. To successfully locate other objects in the field, the same condition as in self localization applies: there must be at least two visible landmarks. If the currently visible landmarks are less than two, other objects cannot be located in the field, but the agent knows where they are located with respect to itself.

4.4.3 Localization Filtering

In absence of a more sophisticated probabilistic localization scheme, we are forced to ensure that localization results are qualitative enough for us to rely on. Due to the temporary absences of landmarks from the field of view and the noisy observations from the vision perceptor, localization is not always accurate enough to depend upon. Therefore, a simple filtering procedure on estimates computed by the localization process is employed

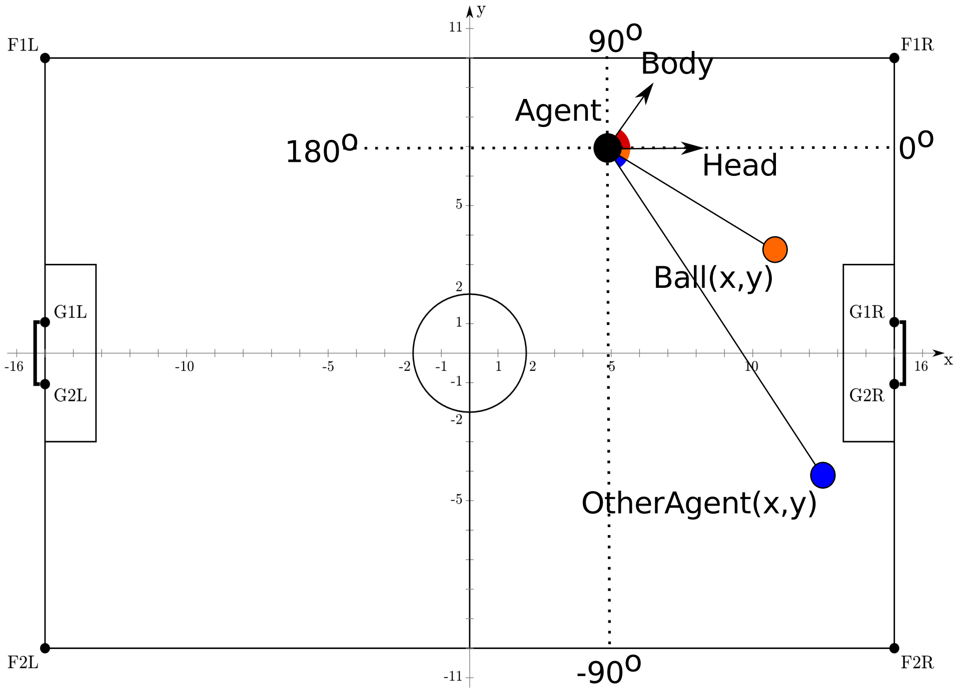

Figure 4.5: Object Localization Example.

to update the agent's belief about self locations and positions of other objects. Algorithm 1 outlines the process of localization filtering. In general, the localization process provides the agent with hundreds of estimated locations over time. The general idea we adopt is based on the fact that these estimates do not include long sequences of consecutive faulty estimates. Therefore, it is fairly easy to ignore sporadic faulty estimates (outliers), while updating the agent's belief.

To overcome this difficulty, we came up with a simple algorithm. We maintain a FIFO queue which stores the most recent non-faulty location estimates. The average of the locations in the queue the agent's belief about its location in the field. When an estimated location arrives from the localization process, we first check if the queue is empty; in this case, we simply insert this estimate into the queue. Otherwise, we check if the newly arrived estimate "agrees" with the current belief. If not, this estimate is ignored and one element of the queue is removed. This step represents a simple way of discounting the current belief in the presence of an outlier estimate. If the outlier estimate corresponds to a correct location, it will persist and eventually will discard the entire queue with the current belief and will initiate a new belief in the empty queue.

Algorithm 1 Localization Filtering

Input: $LastEstimate$
Output: $FilteredLocation$
$Queue$: a FIFO queue storing the $MaxSize$ (default=10) most recent estimates

if $size(Queue) = 0$ **then**
  $Queue.Add(LastEstimate)$
else if $LastEstimate \not\approx AverageLocation(Queue)$ **then**
  $Queue.Remove()$
else
  if $size(Queue) = MaxSize$ **then**
    $Queue.Remove()$
  end if
  $Queue.Add(LastEstimate)$
end if
return $AverageLocation(Queue)$

Finally, if the new estimate "agrees" with the current belief, it is inserted in the queue to reinforce the current belief, replacing one element if the queue has reached capacity. This simple filtering scheme smooths the belief of our agent's location and rejects most faulty estimates.

Localization filtering applies both to the calculation of our agent's location and to the calculation for the ball's position. For the sake of efficiency, we do not use it to filter opponent positions (teammate positions are known anyway, since they are shared via communication). The end effect of localization filtering is a significant improvement on the localization outcome, so that we can rely upon it with confidence.

4.5 Motion

The simulated Nao robot comes with 22 degrees of freedom, corresponding to 22 hinge joints. Figure 4.6 shows Nao's anatomy with all joints, split in five kinematic chains (head, left arm, right arm, left leg, right leg). In robotics, a complex motion is commonly defined as a sequence of timed joint poses. A pose is a set of values for every joint in

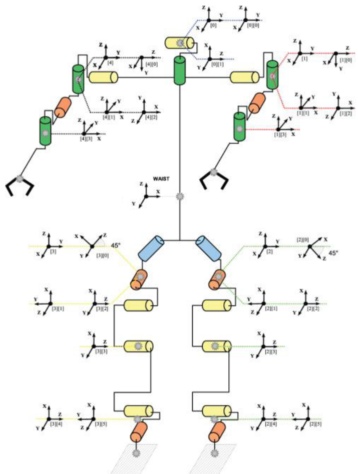

Figure 4.6: Nao's anatomy: kinematic chains and joints.

the robot's body or in a specific kinematic chain at a given time. For any given set of n joints a pose at time t is defined as:

$$Pose(t) = \{J_1(t), J_2(t), \ldots, J_n(t)\}$$

Motion is an important part of every team participating into the RoboCup 3D Simulation League. Motions can be static or dynamic. Most teams in this league make use of dynamic motions, which give a major advantage on their side, at the expense of complexity. In our work, we are using predefined static motion files. Motion files describe timed sequences of poses, which provide fixed values for each joint at specific times; when executed, these sequences achieve some kind of desired motion. The difference between static motion files and dynamic motion is that the latter makes use of forward and inverse kinematics, as well as calculation of the center of the robot's body mass, to derive sequences of poses dynamically. These dynamically computed motions can be corrected on the fly for better body balance and faster movement. Since our goal was to study coordination algorithms, we chose to work with the simpler approach of static, yet effective, motion files with some dynamic enhancements. In our approach, we are using two kinds of such files: XML-based and text-based.

4.5.1 XML-Based Motion Files

These motion files have been created and distributed by FIIT RoboCup 3D project [] and they provide forward walk, left and right side step, strong and regular kick, stand-up, and left and right goalkeeper fall motions. These files have an intuitive XML structure, which facilitates integration into our project. The general structure of these XML-based motion files is shown below.

```
<phase name="Start" next="Phase1">
   <effectors>
      Joint Values
   </effectors>
   <duration>duration</duration>
</phase>

<phase name="Phase1" next="Phase2">
   <effectors>
      Joint Values
   </effectors>
   <duration>duration</duration>
</phase>

<phase name="Phase2" next="Phase1">
   <effectors>
      Joint Values
   </effectors>
   <duration>duration</duration>
   <finalize>Final</finalize>
</phase>

<phase name="Final">
   <effectors>
      Joint Values
   </effectors>
   <duration>duration</duration>
```

```
</phase>
```

As we can realize by the structure of this XML file, each movement is split into phases. Each phase has a duration and values for every joint of the robot or the kinematic chain(s) related to the movement. Moreover, every phase has an index which points to the next phase. For example, the first phase (“Start”) has an index to the next phase (“Phase1”). Phases with a finalize field can be used to end the corresponding movement. For example, phase “Phase2” has a finalize index which points to phase “Final”; this means that, if we want to end the execution of this movement, we have to do it in phase “Phase2” by transitioning to phase “Final” instead of continuing with the next phase (“Phase1”).

4.5.2 XML-Based Motion Controller

The motion controller is a major component which controls and enables the movement ability of the robot. It is responsible for handling the movement requests of the agent. Agents do not have access to the motion controller directly, but can trigger desired motions by posting requests to the motion controller in the form of values to a specific variable. Each agent declares in this variable the movement he is willing to perform. In each cycle, the motion controller reads this variable and generates an appropriate hinge joint effector string as a result.

Figure 4.7 describes the general architecture of the motion controller. Motion controller checks if there is a motion which is playing already. If the currently playing motion is the same as the requested one, the motion controller continues with the next of its phases; if not, the controller tries to finalize the playing movement in order to start playing the newly requested one.

Figure 4.8 describes the exact motion sequence. In general, XML motions define cyclic phases to generate continuous movement. For example, the walking motion has three main phases which create a cycle. If the motion trigger has not changed at the last phase, we have to continue with the execution of the first phase, not with the final one. As we saw in the structure of every XML-based motion file, each phase has a set of joint values. These values are in degrees. To generate motions for our agent we need to create a motion string, which encloses information about each joint's velocity. This velocity is

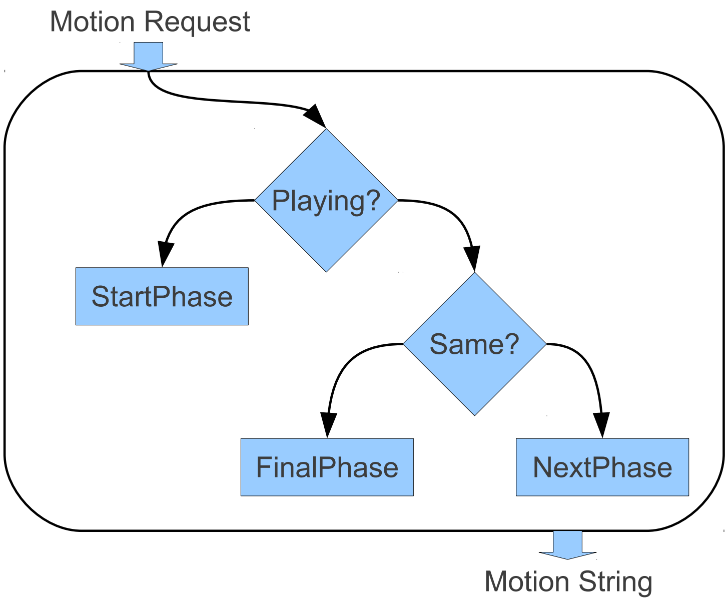

Figure 4.7: XML-Based Motion Controller.

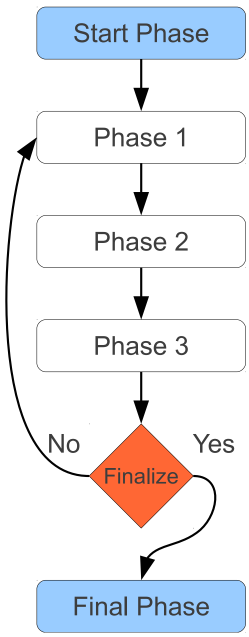

Figure 4.8: Phase Sequence.

computed as follows:

$$JointVelocity = \frac{DesiredJointValue - CurrentJointValue}{PhaseDuration}$$

A velocity value is calculated for each joint involved in the motion and the final output of the motion controller is sent to the server. In addition, zero velocity is set for every joint not included in the `effector` field of each phase, so that they stop moving.

4.5.3 Text-Based Motion Files

The text-based motion files we use have been adopted from the Webots simulator [7] and they provide left and right turn motions. These text-based motion files have simpler structure than the XML-based motion files. By default, each pose lasts two simulation cycles (40ms). The structure of these motion files is shown below.

```
#WEBOTS_MOTION,V1.0
LHipYawPitch,LHipRoll,LHipPitch,LKneePitch,LAnklePitch,...
00:00:000,Pose1,0,-0.012,-0.525,1.05,-0.525,0.012,0,...
00:00:040,Pose2,0,-0.011,-0.525,1.05,-0.525,0.011,0,...
00:00:080,Pose3,0,-0.009,-0.525,1.05,-0.525,0.009,0,...
00:00:120,Pose4,0,-0.007,-0.525,1.05,-0.525,0.007,0,...
00:00:160,Pose5,0,-0.004,-0.525,1.05,-0.525,0.004,0,...
00:00:200,Pose6,0,0.001,-0.525,1.051,-0.525,-0.001,0,...
00:00:240,Pose7,0,0.006,-0.525,1.05,-0.525,-0.006,0,...
00:00:280,Pose8,0,0.012,-0.525,1.05,-0.525,-0.012,0,...
00:00:320,Pose9,0,0.024,-0.525,1.05,-0.525,-0.024,0,...
```

Lines starting with `#` are comments. The fist non-comment line must list all joints involved in the defined movement using their names separated by commas. As an example, a walk motion requires only the joints from the leg kinematic chains. The remaining lines define one pose each. From left to right, each line contains the duration of the pose, the pose number, and the desired angle (in radians) for each joint in the same order they were given at the beginning.

4.5.4 Text-Based Motion Controller

The motion controller for text-based motions is based on the same principle as the XML-based controller. The joint values in the motion files represent radians, so we have to convert these values into degrees before we proceed. Each pose lasts one or two cycles depending on the speed at which we want the motion to be executed. The motion controller could be customized easily to perform these motions in different ways. The following parameters can be modified:

Duration The time between poses in simulation cycles. By default, $Duration = 2$.

PoseStep The step for advancing from pose to pose. By default, $PoseStep = 1$, but we can subsample the motion with other values, e.g. for $PoseStep = 2$, we execute pose1, pose3, pose5, ...

The desired velocity of each joint is computed by:

$$JointVelocity = \frac{DesiredJointValue - CurrentJointValue}{Duration \times CycleDuration}$$

A velocity value is calculated for each joint involved in the motion and the final output of the motion controller is sent to the server.

4.5.5 Dynamic Motion Elements

In contrast to the general idea of static motion files, we have tried to implement some dynamic features into our motions. There is no much room for improvement in static motions, but with these features we managed to achieve some nice results.

Walk Leaning The XML-based walk motion can be dynamically modified to lean to the right or to the left. This is accomplished by altering the joint values of the (left or right) HipPitch and AnkePitch joints in specific phases of the walk motion. In effect, these changes force the corresponding leg to perform a slightly smaller step compared to the nominal step size. The end effect is a smooth walk motion which leans slightly to the left or to the right, as shown in Figure 4.9, saving time from a full body turn motion.

Walk Slowdown Its important for our agent to slowdown while stopping in order to maintain stability. The XML-based walk motion can be dynamically modified by scaling the phase durations in order to achieve such a slowdown. Increasing the phase durations dynamically by about 35% yields a smooth approach to a stopping position.

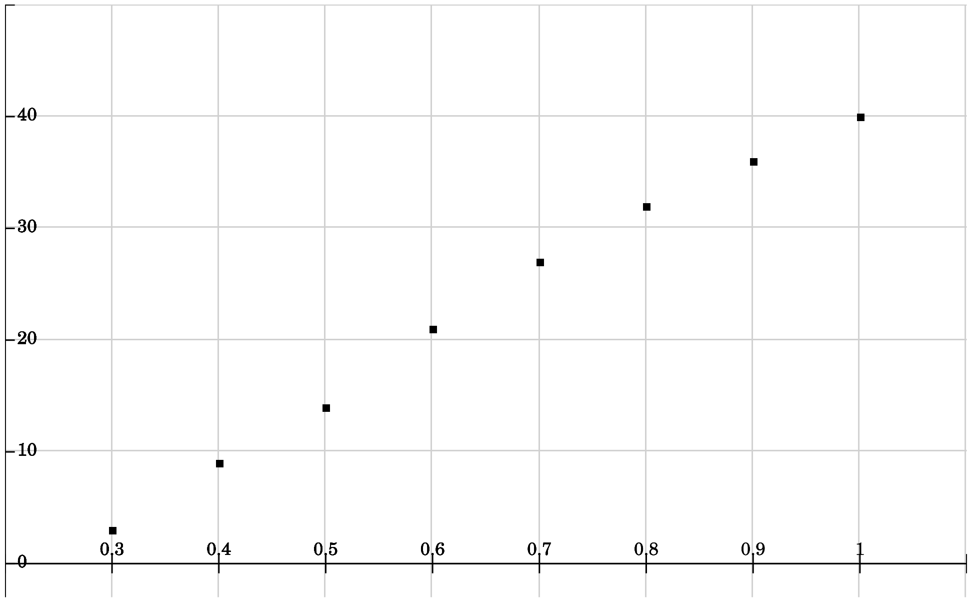

Figure 4.9: Dynamic Walk Leaning: Left Leaning, No Leaning, Right Leaning.

Turn Gain The text-based turn motion can be dynamically modified using a gain value for scaling the resulting velocities in order to perform the motion in a smoother or rougher way. By default, this gain value is set to 1.0, however slightly smaller or larger values can result in useful variations of the defined motion. By dynamically changing this value between 0.3 and 1.0, the agent is able to turn its body anywhere between 7 and 40 degrees, as shown in Figure 4.10.

4.6 Actions

In this section, we describe the way agents affect and change their environment. In our approach, actions are split into groups in terms of their complexity and type.

4.6.1 Basic Actions

Basic actions combine perceptual information and motion files in simple ways to achieve something useful. These basic actions are:

Look Straight Moves the head to its nominal position. Both head joints are set to 0.

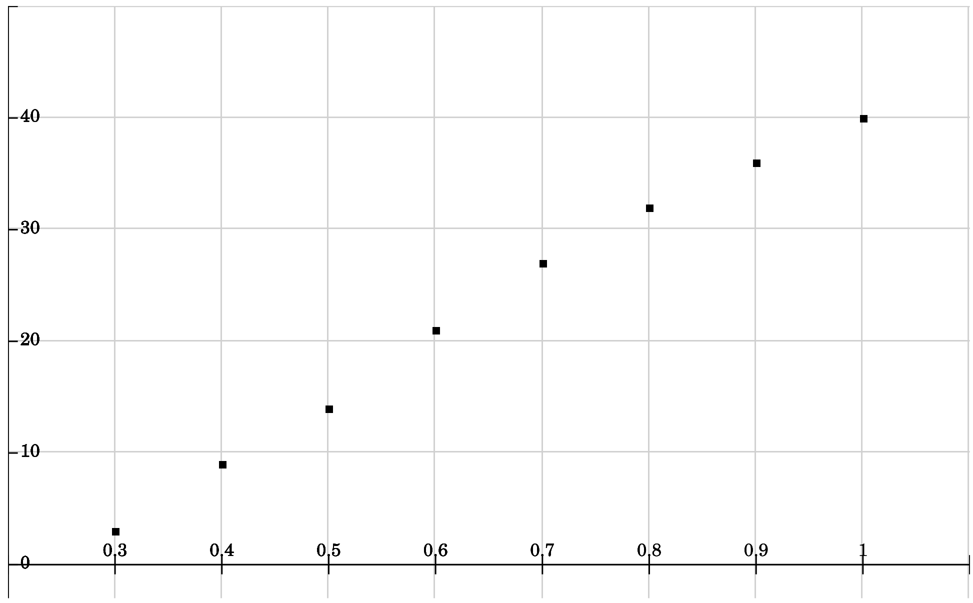

Figure 4.10: Dynamic Turn Gain: Turn Degrees (y-axis) against Gain Factor (x-axis).

Scan Moves the head to perform periodic panning and tilting.

Pan Head Moves the head to perform periodic panning at zero tilt.

Track Object Moves the head to bring a particular object to the center of the field of view. This action is applicable only when the object being tracked is visible, but is limited by the joint ranges.

Track Moving Object This action estimates the direction and the speed of a moving object using a small number of observations, obtained while performing the Track Object action. It records a set of five consecutive observations and another set of five consecutive observations delayed by a fixed time period (the default is 5 cycles). The difference between the average positions of each set gives a vector that reveals the direction of motion. Taking the ratio of the magnitude of this vector and the time delay yields the speed of the moving object. This action is applicable only when the moving object being tracked is visible, but is limited by the joint ranges.

Find Opponent's Goal This action estimates the direction of the opponent's goal with respect to the agent by performing the Scan action.

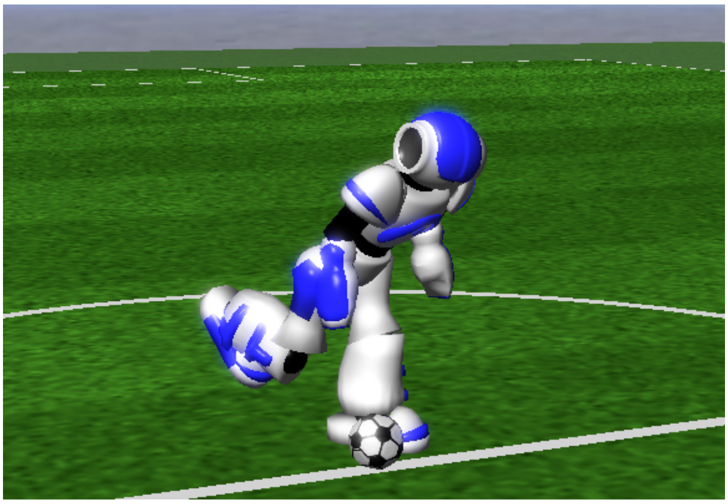

Figure 4.11: Nao Performing a Kick after Positioning for Kick.

Look for Ball Turns the body of the agent, while performing the Scan action, until the ball appears within the field of view.

Turn to Ball Turns the body of the agent towards the direction of the ball, while performing the Track Ball action. It can applied only when a ball is visible.

Turn to Localize Turns the body of the agent, while performing the Pan Head action, until the agent's belief about its own location is updated with confidence. It can be used when the agent needs to (re)localize itself into the field.

Stand Up Makes the agent stand up on its feet, after a confirmed fall on the ground, whether face-up or face-down. This action monitors the inertial sensors (accelerometers and gyroscopes) to check if our agent has fallen on the ground. Incoming gyroscope and accelerometer values above a specific threshold indicate a possible fall, but this has to be confirmed, because it is not unusual to receive values above threshold due to collisions without a fall. To confirm a fall, the action checks the force resistance perceptors located under the agent's feet. If these perceptors imply that the legs do not touch the ground, then we are pretty sure that a fall has occurred. In this case, a stand up motion is executed. Foot pressure values are also used to determine whether the stand up motion succeeded or not. The stand up

Algorithm 2 Escape Angle Set Calculation

Input: $Obstacles = \{O_1, O_2, ..., O_n\}$
Output: $EscapeAngleSet$

for $i = 1$ **to** n **do**
  find $LeftEscapeAngle_i$ for obstacle O_i
  find $RightEscapeAngle_i$ for obstacle O_i
end for
$EscapeAngleSet = \emptyset$
for $i = 1$ **to** n **do**
  if $LeftEscapeAngle_i \notin [LeftEscapeAngle_j, RightEscapeAngle_j], \forall j \neq i$ **then**
    $EscapeAngleSet = EscapeAngleSet \cup \{LeftEscapeAngle_i\}$
  end if
  if $RightEscapeAngle_i \notin [LeftEscapeAngle_j, RightEscapeAngle_j], \forall j \neq i$ **then**
    $EscapeAngleSet = EscapeAngleSet \cup \{RightEscapeAngle_i\}$
  end if
end for
return $EscapeAngleSet$

motion is repeated, until it succeeds. This action is applicable at all times, however a stand up motion is executed only when the robot lies on the ground.

Position for Kick Positions the agent to an appropriate position with respect to the ball in order to perform a kick successfully. Only forward and side steps are used in this fine positioning. Figure 4.11 shows an example of robot kicking a ball after completing this positioning action.

4.6.2 Complex Actions

Complex actions combine perceptual information, motion files, and basic actions. They have a more complicated structure and aim to achieve specific goals. These complex actions are:

Avoid Obstacles This action records all obstacles around the agent and derives an obstacle-free route to a specified target. Initially, this action stores the positions of all obstacles located within 2m from the agent by performing the Pan Head action and watching the perceptor messages. Due to the fact that the simulated Nao's head can pan from $-120°$ to $+120°$ and the field of view is $120°$, we can obtain a complete imaging of all obstacles located close to our agent. It is common to observe the same obstacle more than once; in this situation we only store the average of these observations. In a dynamic, multi-agent environment it is important to avoid collisions with other agents or fixed landmarks, such as goal posts in our simulated soccer games. To avoid obstacles we rely on a simple, yet reliable and effective, method. For each recorded obstacle, we calculate two escape angles that determine the two directions which guarantee avoidance of the obstacle at a safe distance. All other angles between these two escape angles are considered to be forbidden. Afterwards, any escape angle of some obstacle that falls within the forbidden area of some other obstacle is discarded. The precise calculation of the escape angle set is described in Algorithm 2. The remaining escape angles, and particularly the escape way points they define (the points closest to the obstacle along the direction of the escape angles), are evaluated in terms of the angle and distance overhead they incur with respect to the agent orientation (for the angle) and the target (for the distance). The way point that minimizes the total overhead is selected as a temporary target for avoiding the obstacles, while making progress towards the target. This method will yield dynamically consistent results, meaning that the temporary target will not change in subsequent cycles as long as the obstacles remain stationary, until they are cleared on the way to the real target. Obstacle avoidance is demonstrated in Figure 4.12, in a simple scenario where there are two obstacles between the agent and its target position. Two of the escape angles are discarded; the way point on the left will be selected as a temporary target.

Walk To Ball Makes the agent walk towards the ball and stop when the ball is close enough to perform a kick. First, it performs the Turn to Ball action and then walk towards the ball, slowing down when it comes close to the ball. Recall that the ball distance returned by the vision perceptor is the distance between the camera, which is attached to agent's head, and the ball. However, for approaching the ball, we need the distance on the ground between its feet and the ball. To compute this distance, we first use forward kinematics along the sagittal plane of the robot to derive the

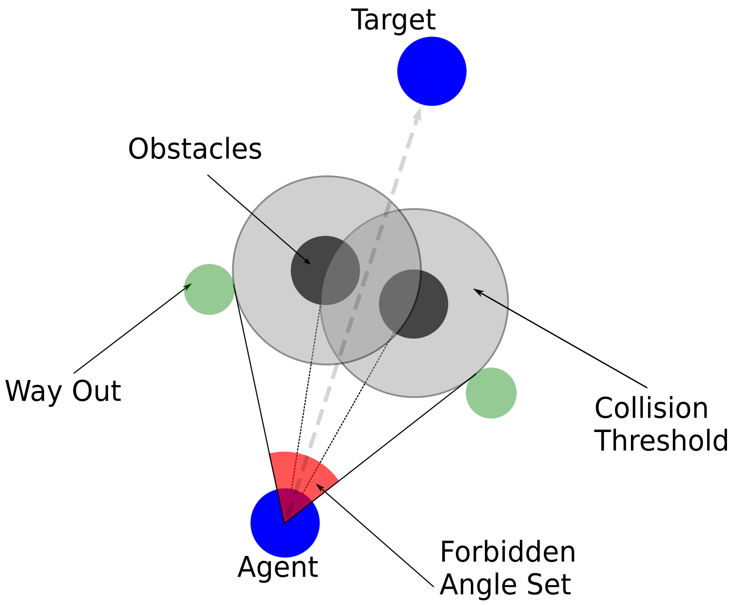

Figure 4.12: Obstacle Avoidance.

current height of the camera. Taking the agent's ankle as the origin, it is easy to calculate every joint's position from ankle to head in the two-dimensional space of the sagittal plane using only the current values of the AnklePitch, KneePitch, HipPitch joints. Having the ball distance and the height of the camera, the ground distance can be easily derived using the Pythagorean theorem.

$$GroundDistance = \sqrt{BallDistance^2 + CameraHeight^2}$$

Figure 4.13 explains the derivation of the ground distance to the ball.

On Ball Action This action moves the agent close to the ball and executes an appropriate kick depending on the current state of the game. This action has a finite state machine logic shown in Figure 4.14. It first performs the Walk To Ball action in order to reach the ball. After the successful completion of the Walk To Ball action, the agent performs the Find Opponent's Goal action. Subsequently, it aligns itself with the direction of the opponent's goal; the precision of this alignment is inversely proportional to the distance from the opponent's goal, meaning that far away from the opponent's goal there is more tolerance, since we only want to clear the ball from our own half of the field. Afterwards, it performs the Position for

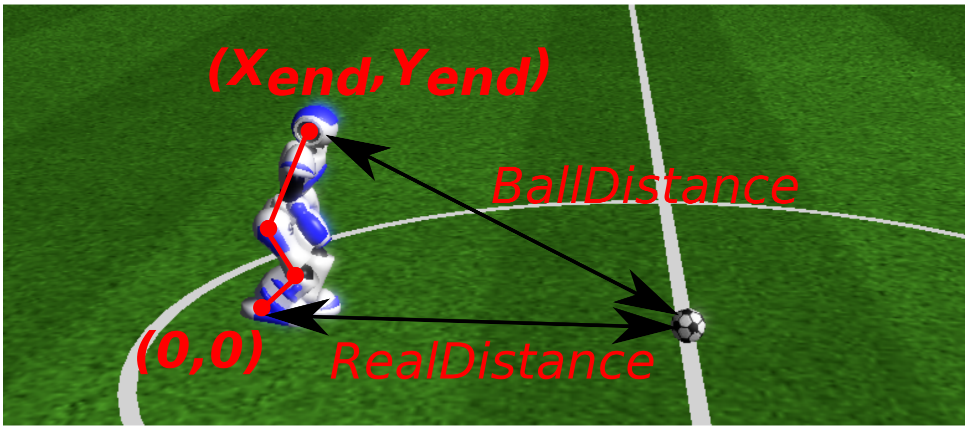

Figure 4.13: Ground Distance between the Agent and the Ball.

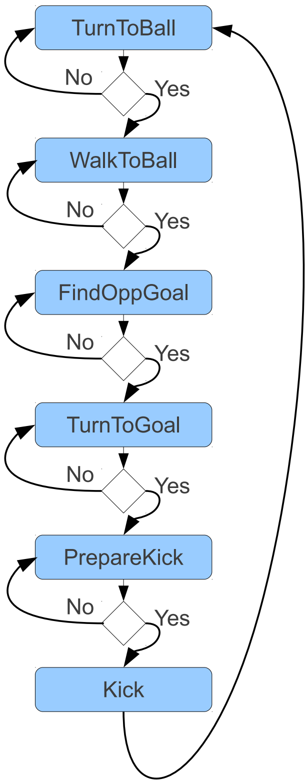

Figure 4.14: On Ball Action Logic Sequence.

Kick action and finally it executes a kick motion. At any point of time, it is possible that an opponent agent takes the ball away from our agent; if that happens, the agent returns to the beginning.

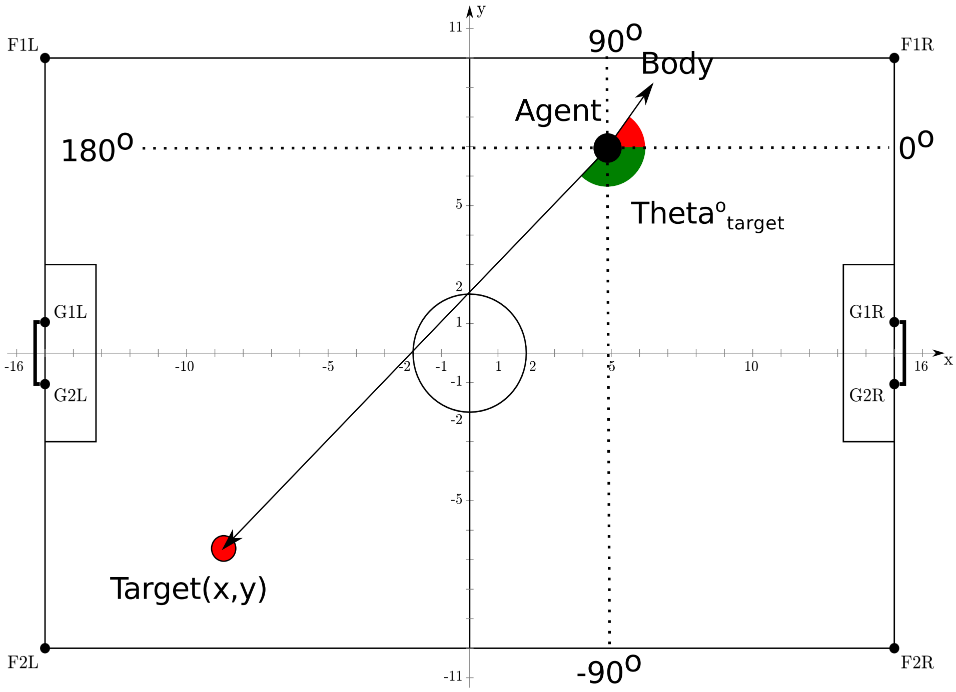

Figure 4.15: Walk To Coordinate Action.

Walk To Coordinate This action moves the agent to a specific location (x_t, y_t, θ_t) in the field. To perform this action we need to know our own location (x_a, y_a, θ_a) in the field; from there it is easy to calculate in which direction to walk in order to reach the given target. Figure 4.15 shows how the distance d and the direction ϕ to the target are calculated:

$$\phi = \text{atan2}(x_t - x_a, y_t - y_a)$$
$$d = \sqrt{(x_t - x_a)^2 + (y_t - y_a)^2}$$

Been helped from the above calculations agent is always aware of the distance and the direction it has to travel towards its target position.

Walk To Direction This action leads agent to walk towards a specific direction.

Walk With Ball To Direction As far as agent reaches the ball, he will try to keep the ball in front of its feet and walk towards a direction keeping into mind that the ball has to be always in front. This action is not yet functional in our approach as

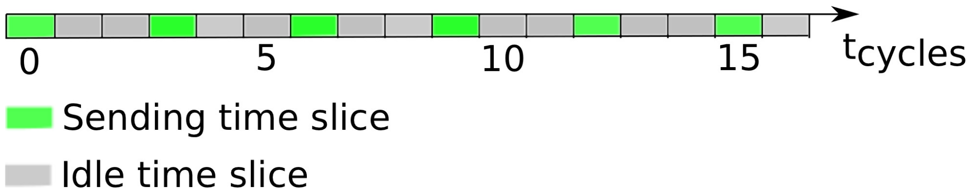

Figure 4.16: Time Slicing Communication.

movements based on motion files make it hard for us keeping ball in front of our agent all the time.

4.7 Communication

Communication in Simspark is not ideal. There are not restrictions about the use of say effector and every agent can use it in every cycle. However, the hear perceptor comes up with some restrictions. Messages shouted from beyond a maximal distance (currently 50 meters) cannot be heard. Note that as the field is currently only 21x14(0.6.5) or 20x30(0.6.6) meters (25 or 36 diagonally), this does not turn out to be a limit in practice. Due to the limited communication bandwidth we utilize the communication channel in the following way, making sure that every message which is sent from an agent will be heard by other agents in time. A simple communication protocol is created in which time is sliced into pieces each one of them lasts one server cycle (20ms) and repeats every three cycles (60ms). Figure 4.16 shows how time is sliced. Every three cycles there is one of these pieces in which only one agent is able to send its message to the others. Every slice has an integer label on it which states the uniform number of the player which is able to send its message. This label grows by one in every time a player send its message until it reaches the maximum uniform number, then it returns to the number one. Agents are not permitted to use a common chronometer for this task but we make sure that each player is synchronized with the others making use of the changing game states. By using this simple protocol we achieve that every player can receive the other eight agents' messages in 540ms.

4.8 Goalkeeper Behavior

In our approach, only goalkeeper "runs" an independent behavior, the rest field agents start a communication procedure between them and goalkeeper in order to inform him about their beliefs of the world state and their state whenever it is needed. Goalkeeper, is going to decide about the actions that every field player should do. So, we can realize that field players do not execute any behavior. In contrast, goalkeeper executes this task for them.

This section presents the behavior that leads goalkeeper to make decisions and choose actions for itself. As we said in Section 4.1, goalkeeper is the only agent in our team who "runs" his own behavior. His behavior depends on a finite state machine. His initial state is "**start**" state. In this state goalkeeper tries to move himself in the center of his goal. When he accomplishes moving there, we change his FSM's state to "**Guard**" state. In guard state he makes use of the Watch Object Movement action to figure out the ball's current position and the direction and the speed of its possible movement.

Figure 4.17 is going to help you understand easier this state's basic idea. Goalkeeper Considers his position as the start of both axis x and y due to difficulties in making use of the localization process. So, red dashed line determines the goalkeeper's y-axis and x-axis. For every movement of the ball he tries to compute if there is an intersection point between its y-axis and the grey dashed line which starts from ball's position in the direction of ball's movement. If there is an intersection point between these two lines then agents computes if this point is out of the two thresholds ($Threshold_{Right}$,$Threshold_{Left}$). If not, we are pretty sure that ball is heading towards our goal. We compute how much time will take to the ball to meet our y-axis according to its speed and taking account the friction between ball and the ground. If this time is equal or less than the time takes our agent to fall, agent performs a right or a left fall. You can see agent falling to prevent a goal in Figure 4.18. There are also other states. State "**Libero**" is a state in which goalkeeper sees the ball into his box and there are no other agents near to it. Then goalkeeper goes to clear the ball from his box. Through coordination process informs other field players that he is at "libero" state to prevent them from going towards the ball too. When he clears the ball, he returns to his initial position.

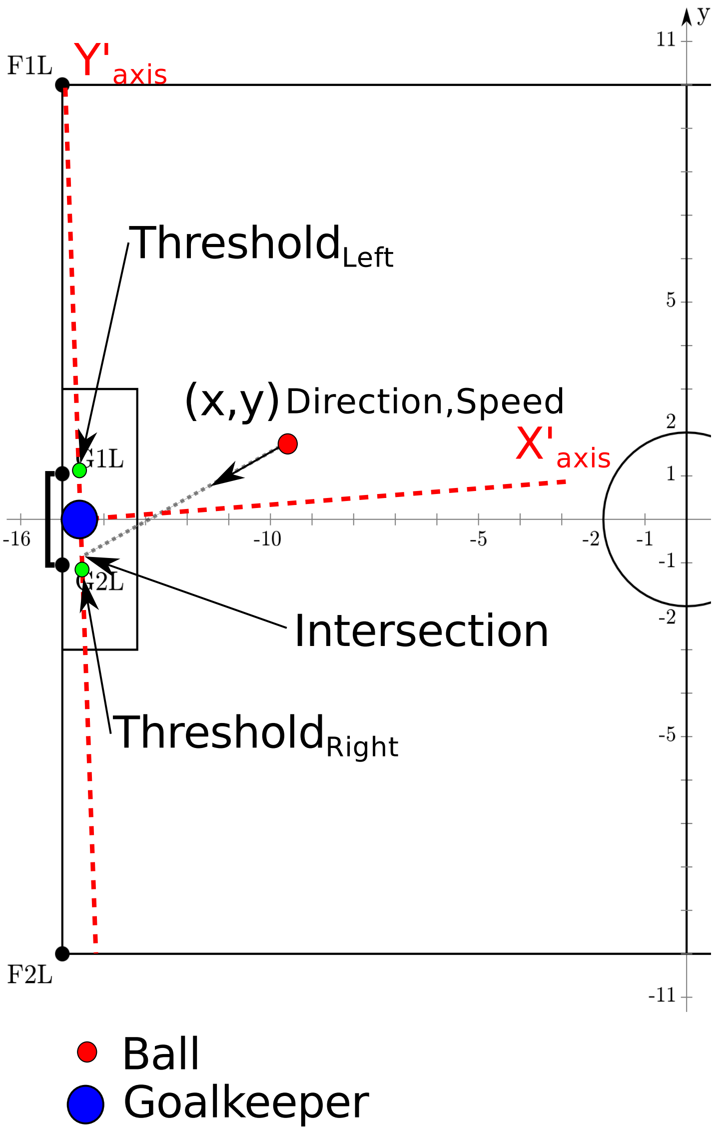

Figure 4.17: Goalkeeper Fall Function.

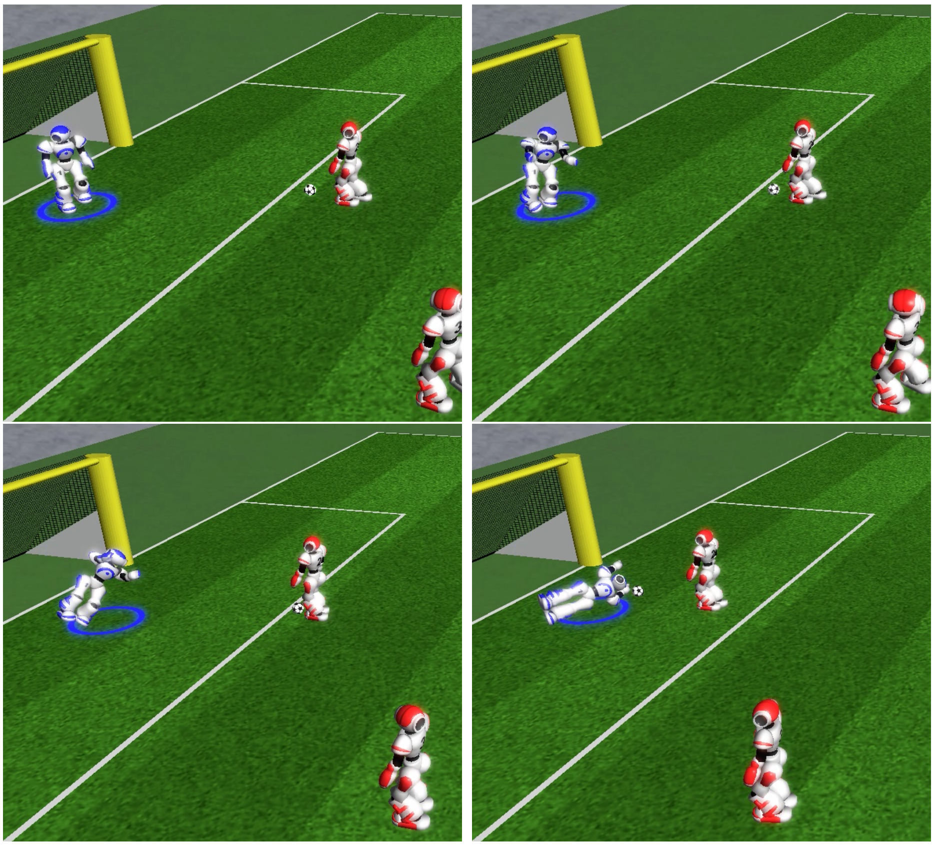

Figure 4.18: Goalkeeper Falls to Prevent Opponents from Scoring.

Chapter 5

Team's Coordination

In this chapter, we are going to present the most important, exciting and time consuming part of this thesis. Until now, we have discussed all parts that agent is going to need in order to be functional into the soccer field. With all these functionalities agents are able to locate themselves in the field, communicate with each other and execute actions combining movements through motion controller. However, agents miss a thinking process with which they will be able to decide about what action they should do for their team's benefit. For example, imagine a human soccer player who is able to do all the things needed in a football match but he has not the ability to choose what to do. Therefore, there must be presented a high-level process which will combine all these skills, motions,communication ability and actions having as a result a complete agent's behavior. As a behavior, we could define the process in which each agent takes as arguments his beliefs and decides what he will do as an output. In our approach, instead of each agent having his own behavior, players are depend on a centralized process which is called coordination. Coordination's algorithm is responsible to gather messages from all agents and as an output it produces actions which are costless for all players who are included into this process. We chose goalkeeper as the coordination's administrator to be the one who is going to execute this procedure. This means that goalkeeper will only be "running" his own behavior and other field players will not. Field players are just sending their beliefs to the goalkeeper and he is sending back the actions which are calculated by the coordination's algorithm execution.

Figure 5.1 shows the difference between a field player and the goalkeeper. Goalkeeper has to calculate actions for all field players as well as execute his own behavior. In

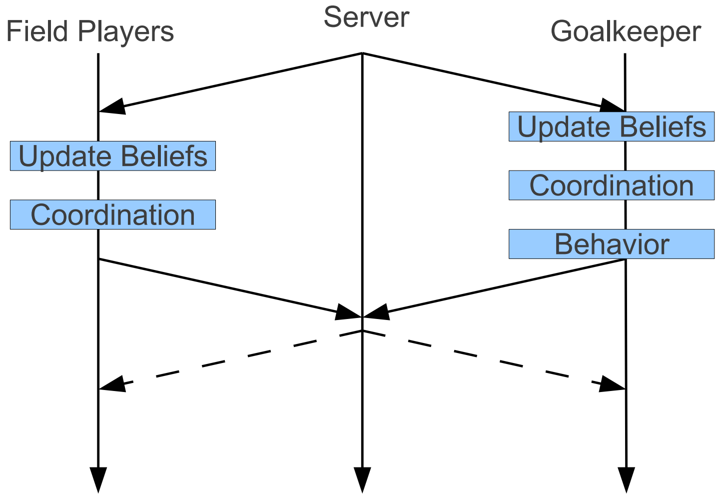

Figure 5.1: Coordination cycle.

contrast, field players do not execute any behavior but only send messages to goalkeeper and receive the calculated actions from him. We selected the goalkeeper to take the responsibility for this task due to the fact that he has the less significant role in the simulation soccer. Coordination's procedure is executed in several phases and not at once. These phases are:

Update coordination beliefs Multiple world state beliefs from field players have to be combined in order to update our belief for the world state.

Split field player into groups Field players are split into groups according to their significance to the game state. The groups are:

- **Active** This group consists of three players and their responsibility is to support and protect the on ball player who is chosen by coordination to be the one who will be given an action which is related to the ball.
- **Inactive** This group consists of players which are not able to know their positions in the field or they have been fallen on the ground.
- **Support** This group consists of the team's rest players and their responsibility is to fill team's formation positions with the best way possible.

Compute positions for active players All possible positions which are best candidates for assigning active players there.

Assign actions for active players Computation of the best two positions according to their cost.

Generate team's formation Formation is generated according to ball's position.

Assign roles for all players Team players are assigned roles in relation to the team's formation and their current position.

Find positions for support players All possible positions which are best candidates for assigning support players there.

Assign actions for support players Computation of the best mapping according its cost.

Algorithm 3 describes the coordination procedure. In every cycle-step, only few of the above phases are executed. We chose doing that because of the time limitation of the think's cycle. So, in order to have a smooth algorithm's execution we decided to separate functions in a number of server cycles.

5.1 Messages and Communication

Coordination could only be accomplished through communication. We use the common communication channel through simulation server in order to provide the messaging between players involved in coordination process. For this reason, communication plays a major role in our approach.

5.1.1 Message Types and Formats

There are multiple types of messages, each one of them has a different functionality and serves an exact purpose. These message types are:

Init Message This type of message declares the start of the coordination procedure for each agent into the field. All field players should sent this message to the coordination's administrator in order coordination procedure to begin.

Message format: `i,<Uniform number>`

Algorithm 3 Coordination Algorithm

Input: $CoordinationMessages = \{M_1, M_2, ..., M_{N-1}\}, N = numberofplayers$
Output: $Actions = \{A_1, A_2, ..., A_{N-1}\}$
if $Step = 1$ **then**
  $B \leftarrow UpdateBeliefs()$
else if $Step = 2$ **then**
  $S \leftarrow CoordinationSplitter(B)$
else if $Step = 3$ **then**
  $A_p \leftarrow ActivePositions(B, S)$
else if $Step = 4$ **then**
  $A_c \leftarrow ActiveCoordination(A_p, S)$
else if $Step = 5$ **then**
  $F \leftarrow TeamFormation(B)$
  $R \leftarrow RoleAssignment(A_c, B, F)$
  $S_p \leftarrow SupportPositions(R, F, S)$
else if $Step = 6$ **then**
  $SupportCoordination(R, F, S, B, A_c, S)$
end if

Start Message This type of message is only sent by the administrator, it declares that all agents are now initialized in the process. Each receiver of this message should immediately start sending coordination messages.

Message format: `s,<Uniform number>`

Coordination Message This is the most important type message. It has information about each agent's beliefs. There are four types of these messages in respect to the agent's situation. these types are:

Type C Agent has complete awareness of the world state. He sends his uniform number, his position and the ball's position accurately..

Message format: `c,<Uniform number>,<Agent X>,<Agent Y>, <Ball X>,<Ball Y>`

Type L Agent has complete awareness only for his position in the field, ball is not in his field of view and it could be best not to send us faulty observations about ball's position. He sends his uniform number and his position.

Message format: `l,<Uniform number>,<Agent X>,<Agent Y>`

Type B Agent has complete awareness only about the ball's distance from his body. its horizontal and its latitudal angle. He sends his uniform number, and the ball's distance and angle in relation to his body angle.

Message format: `b,<Uniform number>,<Ball Distance>,` `<Ball Horizontal-Angle>`

Type X Agent has complete unawareness of the world state. He is sending only his uniform number.

Message format: `x,<Uniform number>`

End Message This type of message serves to stop field players from sending coordination messages. In this step administrator of the coordination is ready to execute the procedure and calculate the actions for all field players.

Message format: `e,<Uniform number>`

Action Message This type of message is only sent by the administrator, it declares which action an agent has been assigned by the coordination process. These messages are sent in the end of the coordination procedure when actions for all field players have been computed.

Message format: `a,<Uniform number>,<Action ID>,<Action parameter 1>,` `<Action parameter 2>,<Action parameter 3>`

Figure 5.2 shows the whole procedure of communication between the agents in order to coordinate their actions. First of all, agents have to initialize their presentation in the field with an "init" message. Goalkeeper saves these message in a temporal array and when he realizes that all players have been initialized themselves he sends them a "start" message. This message means that all players in the field are ready to start the coordination process. In this phase field players send their "coordination" messages to the goalkeeper. When goalkeeper gathers these messages from all field players, he

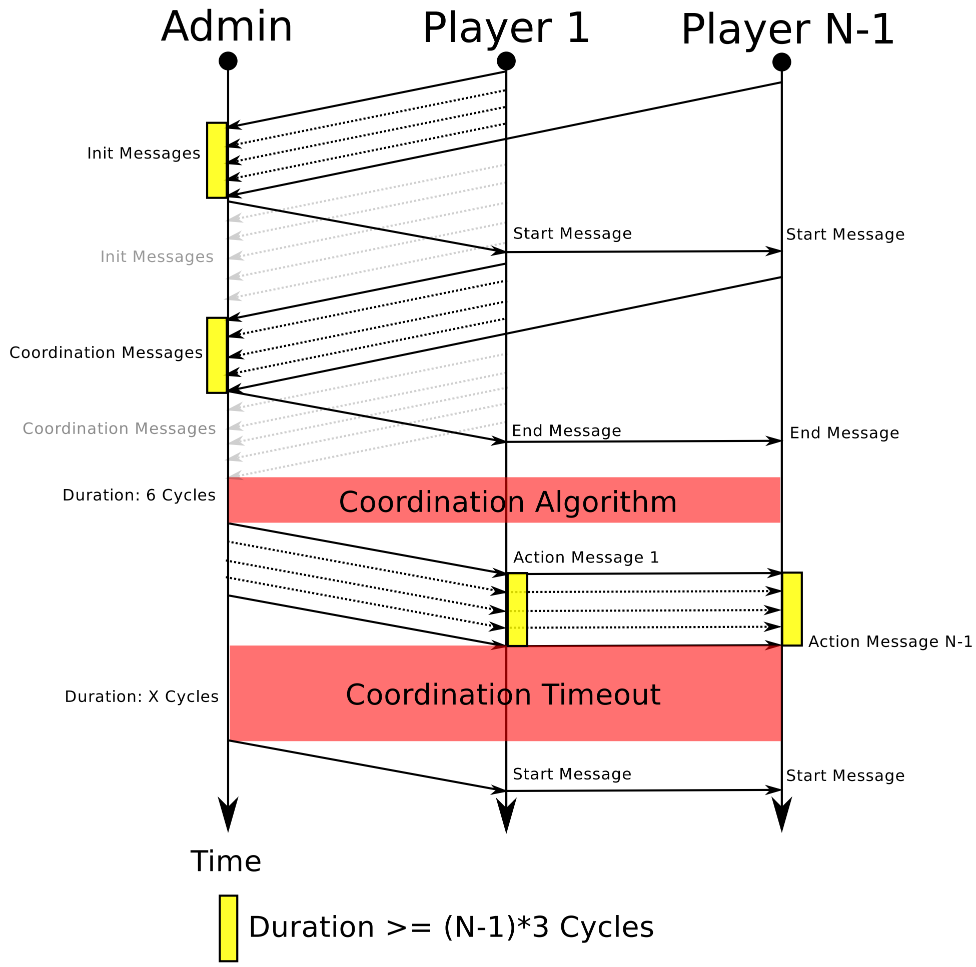

Figure 5.2: Communication Process in Coordination.

sends them an "end" message to stop broadcasting. The next phase of the process is the execution of the coordination algorithm which lasts for six server cycles, approximately 120ms. After coordination phase calculated action must be sent to field players. We are using "action" messages to inform each player about the action he should do. After this phase, the same process is repeated after a timeout which can be defined by us.

5.2 Coordination Beliefs

In the above section we have presented about how field players exchange messages with the goalkeeper which is the administrator of the coordination and the agent who executes the coordination's algorithm. In this section we are going to discuss about how this specific agent could have the adequate knowledge of the world state receiving different observations from different agents. This is a field of major importance in such a multi agent system like simulation soccer. Having multiple observations of the same world could be a problem. Administrator has to combine all these observations without knowing which of them is faulty or correct in order to obtain a realistic representation of the world. Knowing ball position and agents' positions will be more than enough to execute the algorithm without making guesses.

5.2.1 Ball Position Weighted Samples

Ball position is calculated only from agents' observation who are able to locate it in the field and have a good knowledge about their position as well. We can infer that these observations are obtained by agents who have sent "type C" coordination messages. Furthermore, if goalkeeper has also a ball observation he uses it as well.

As we can see in Figure 5.3 ball observation can differ from each other. In our approach, we use a simple algorithm to approach ball's position with great accuracy. A threshold is defined in order to form sets of observations. This threshold is called margin of observation error. Every observation set has a weight. In the beginning, for a given number of n observations we have n observation's sets each one of them has a default weight (1).

$$\mathbf{Obsevation_i} = (x, y)$$
$$\mathbf{Weight_i} = 1$$

Then, we try to correlate these sets with each other, for every two sets which are been correlated, is formed a new set which contains observations from the two parent sets. The weight of the new set will be the sum of the parents' weight.

$$\mathbf{ObsevationSet_i} = \{(x_1, y_1), ..., (x_k, y_k)\} \vee k \in [1, n], k \in \mathbb{Z}$$
$$\mathbf{Weight_i} = k$$

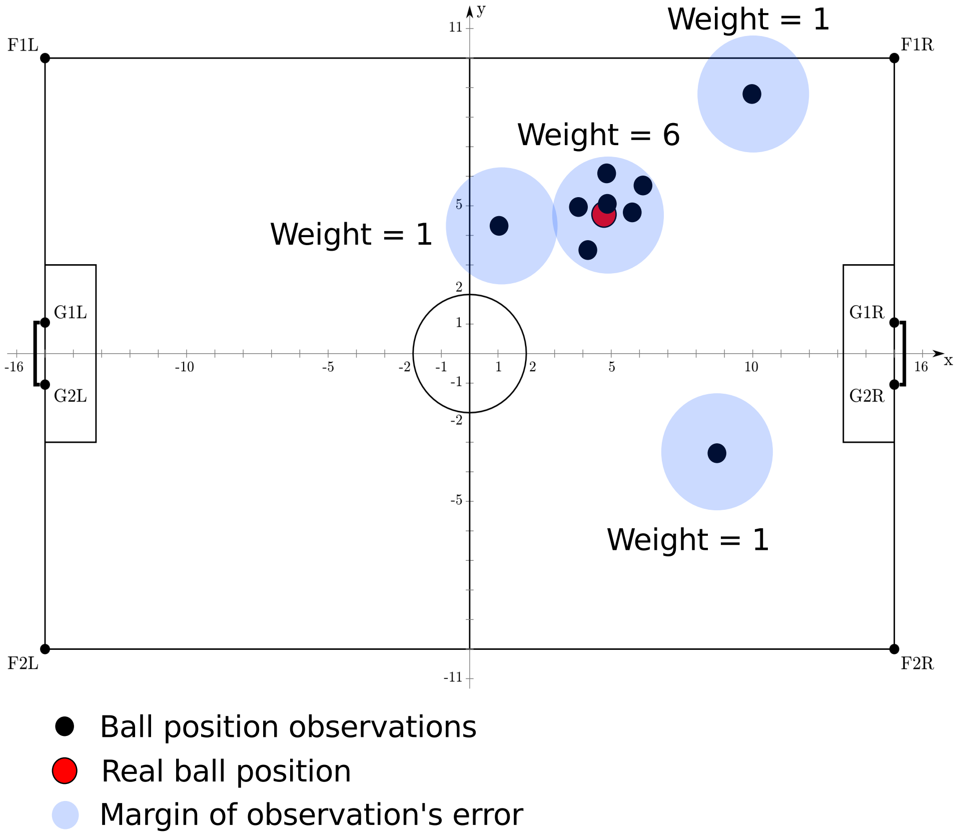

Figure 5.3: Ball's Position Observations.

Figure 5.3 shows the procedure. We can see four sets of observations which have been assigned a weight. The set with the most observations included is naturally assigned the biggest weight. Consequently, we have to compute our belief about ball position. Given a total number of N-observations, n-observation's sets, each one of them has k-observations, the final ball belief will be:

$$\textbf{BallBelief} = \sum_{i=1}^{n} \frac{Weight_i}{N * k} \sum_{j=1}^{k} ObsevationSet_i[j]$$

5.2.2 Agent Distance from Ball

The next step into beliefs section is to determine each agent's distance from the ball. This can be accomplished by two ways. First, for agents who have sent "Type C" and

"Type L" coordination messages and they are able to know their exact position in the field, this distance is calculated by finding the distance between the ball belief's position and the agent's position. For players who have sent "Type B" coordination messages we just take the distance part of the message. Finally, for "Type X" messages we assume ∞ distance.

5.3 Subsets in Coordination Process

The existence of multiple agents makes coordination function to be too complex and computationally expensive to solve by one single agent. In our case, it would be goal-keeper who had to solve this huge problem for nine players or eleven which is the number of players in the next server's version. One possible solution to this problem is to split players into subsets which would be easier to coordinate their actions in real-time. In our approach, there are three subsets:

Active subset Active subset consists of three agents and it is the most important set of agents in the coordination. Agents who constitute this subset have the responsibility of making worthy actions for their team. Moreover, having to calculate actions for three players is not complex for such an important group.

Inactive subset Inactive subset consists of agents who have sent "Type X" messages. It is the less important set of agents in the coordination. Agents who constitute this subset assigned the same action, to find their positions in the field. Finding their positions will be resulted to be inserted either in the active or in the support subset in the next coordination cycle.

Support subset Support subset consists of agents who are neither in the active subset nor in the inactive one. Coordinate actions for these agents is the most time consuming and expensive part of the coordination algorithm.

5.4 Coordination Splitter

In this section we are going to discuss how the above three groups are generated by coordination splitter. An array full of team's agents is sorted according to the distance

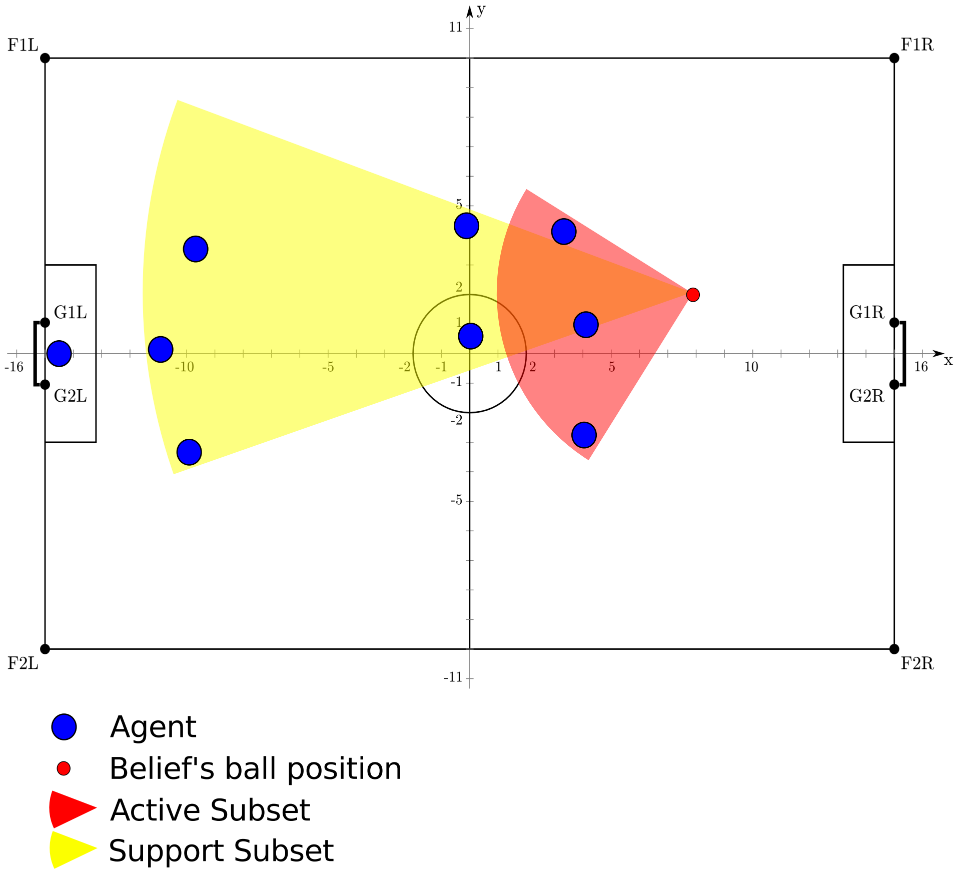

Figure 5.4: Coordination Splitter.

each agent has from the ball. We assign to the active subset the agents in the three first positions of the sorted array. Other agents with distance less than infinity join the support subset. Remind that we assume ∞ distance from ball for the agents who have no idea for their position and have not the ball into the field of their view. In Figure 5.4 is presented an example of the coordination splitter's process. Assuming that all agents have complete awareness of their position, we could realize that the agents in the red distance's threshold will join the active subset. The other six agents who have farther distances from ball will join the support subset.

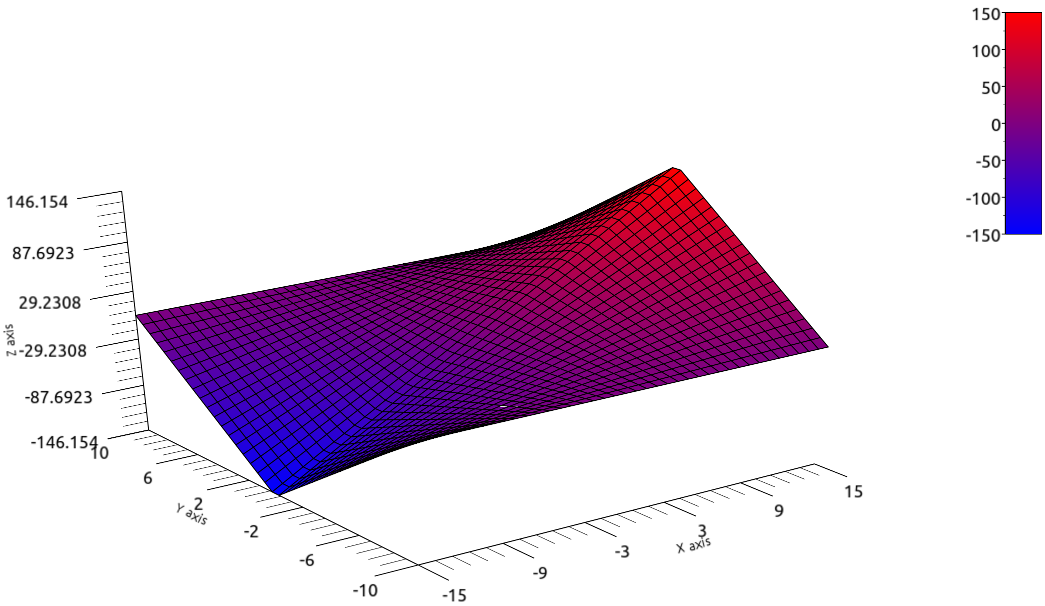

Figure 5.5: Soccer Field Value.

5.5 Soccer Field Value

In order to proceed our discussion about coordination process, we have to demonstrate a simple but functional way to give a value in every spot of the soccer field. In Figure 5.5 we see that each spot in the field takes a value from a function. The main idea is that as we ball is moving towards the opponent goal, this value is becoming higher. In contrast as ball is moving towards our goal, this value is becoming lower. This procedure will prove to be useful in the next steps of coordination process.

5.6 Active-Positions Computation

Until now, we have updated the coordination beliefs and we have split agents into subsets. in this phase of the coordination process, we have to compute adequate and worthy positions for the active subset. We distinguish two cases, in the first case, ball is located in our field's half. In this case we have to find worthy positions which have a defensive approach. On the other hand, if ball is located in the other field's half we have to

Figure 5.6: Active positions before elimination.

find positions which have an attacking approach. In both cases we create an array of equidistant coordinates which are located in a radius which is determined by the ball's location and they are not out of field's limits. Figure 5.6 shows how these positions are shown in the soccer field through roboviz monitor.

From these set of coordinates we choose the best according to their values. As we saw in the previous section each coordinate of the field has a utility value. Consequently, we will try to choose a number of coordinates which summarize in a max value in an attacking approach or in a min value in a defensive approach, being careful not to overcome a max number of coordinates which is nine. This will help us later keeping iterations below a maximum threshold. Figure 5.7 shows how active positions are shown in the soccer field through roboviz monitor after elimination.

5.7 Active-Subset Coordination

Once we have find worthy positions for the active players it is time to find the player who is more adequate than others to become the on ball player. Moreover, we should assign each of the rest two players into an active position. This is called mapping function and will have a significant role in the next coordination's phases.

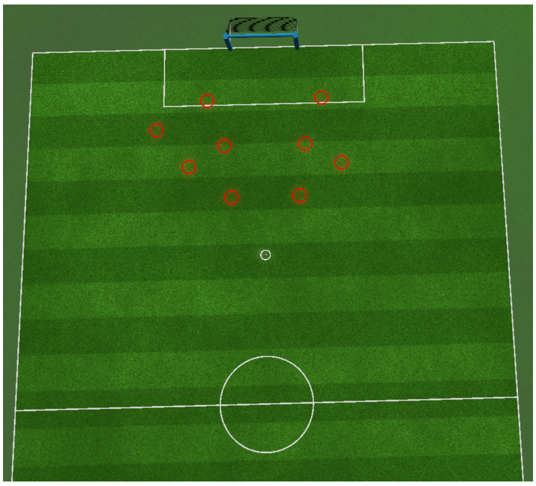

Figure 5.7: Active positions after elimination.

5.7.1 Player on Ball

An agent from the active subset has to be selected in order to be sent an action which is related to the ball. We have to find the agent who has minimum value according to two parameters:

1. **Distance from ball** d_i, ball's distance from each agent in the subset.
2. **Angle towards goal** ϑ_i, this angle is the sum of the angle between agent's body and the ball and the angle between ball and the opponent's goal.

Given an active subset:

$$\mathbf{ActiveSubset} = \{Agent_1, Agent_2, Agent_3\}$$
$$\mathbf{Value_i} = d_i + a * \vartheta_i, a \in \mathbb{R}$$
$$\mathbf{OnBallPlayer} = \arg\min_i(Value_i)$$

Additionally, we give to the agent who had been assigned an action towards the ball in the previous coordination cycle a small advantage over the others to be again the on ball player. We do this due to the fact that there will be continuously changes in the on ball player in cases in which two agents have approximately same distances and angles from the ball.

5.7.2 Active-Subset Best Mapping

Next in the active coordination phase, we have to assign positions for the other two agents who have left in the active subset. Algorithm 4 shows how we can find the optimized mapping. In a greedy approach, we calculate the cost of every possible mapping. In addition, in every mapping we take into account the on ball agent's mapping ($OnBallPlayer \rightarrow Ball$), which will be helpful in order to find possible collisions between the on ball agent and the active ones.

Algorithm 4 Active-Subset Best Mapping

1: **Input:** $ActivePlayers = ActiveSubset - Agent_{OnBall}$
2: **Input:** $Activepositions = \{P_1, P_2, ..., P_N\}, N \leq 9$
3: **Output:** Optimized Active Mapping
4: $OptimActiveMap = \varnothing$
5: $S = \binom{N}{2}$
6: **for each** s in S **do**
7: $\quad ActiveMap = RoleMap[s] \cup (OnBallPlayer \rightarrow Ball)$
8: $\quad OptimActiveRoleMap = mincost(ActiveMap, OptimActiveMap)$
9: **end for**

We can realize that even we are using a brute force method the number of possible mapping remains able to be computed in real-time. Assuming maximum number of active positions in our case nine, the possible mappings are: $\binom{9}{2} = 72$ mappings.

5.8 Team Formation

Team formation itself is not a main contribution of this thesis but serves to set up the role assignment function and the coordination of the support subset. In general, team's formation is determined by the ball's position in the field. The formation is broken up into three groups including all players of the team except from goalkeeper.This section presents the team's formation used in our approach for both 0.6.5 and 0.6.6 rcssserver3d versions.

5.8.1 9-Players Server Version (0.6.5)

Attacking group which consists of three positions:

Fc *Forward center*

Fl *Forward left*

Fr *Forward right*

Defensive group which consists of three positions:

Dc *Defender center*

Dr *Defender right*

Dl *Defender left*

Finally, midfield group which consists of two positions:

Ml *Midfielder left*

Mr *Midfielder right*

As an example, Figure 5.8 shows how the different role positions of the formation are depicted in the soccer pitch. In general attackers are responsible to be assigned positions near to ball when ball is on opponents' half of the field. Then, the forward center is given a position close to the ball and the other two forwards are given positions on either side of the ball in an angle and a distance offset which are determined and dynamically changed according to the ball's exact coordinate. If ball is located in our half, then forwards are given positions which are in the middle of the field.

On the other hand, defenders are mainly positioned to guard our goal. To determine their position on the field a straight line is computed between our team's goal and the ball. Central defender is given a position placed on this line and his distance from our goal is proportional to the ball's position. The other two defenders' positions are located on either side of the defender center.

Midfielders' positions are determined by the ball's position as well. For an attacking phase, in which ball is located to the opponents' half of the pitch, midfielders are given

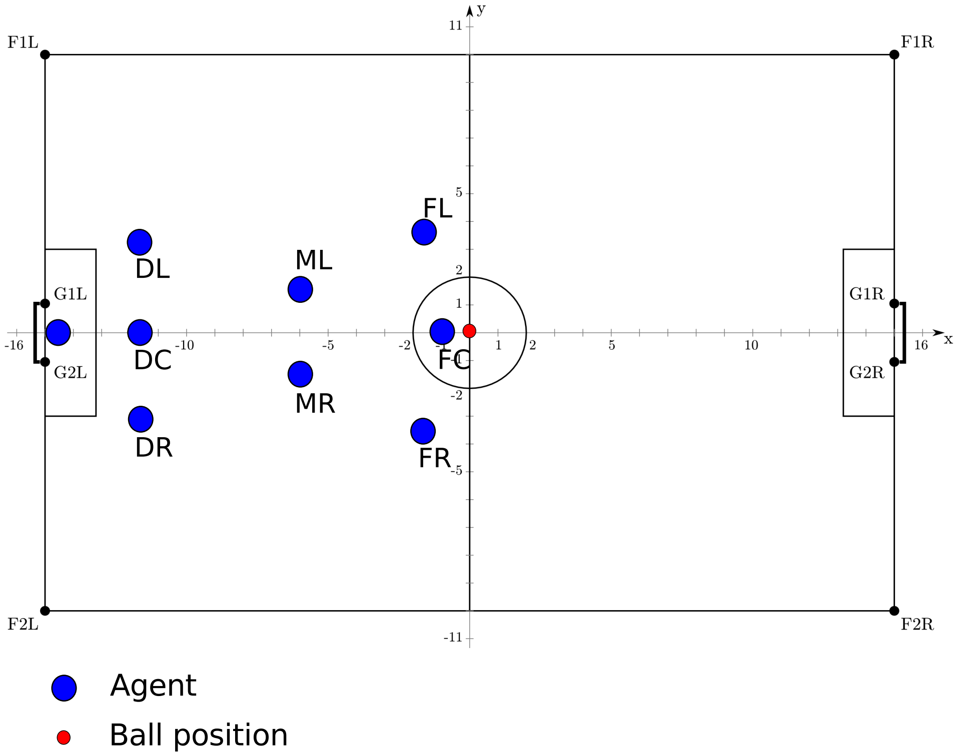

Figure 5.8: Formation Role Positions for 9 vs 9.

position near to the forwards in order to support our attack. In the opposite situation they are given position in front of our defense line to help defenders.

Finally, goalkeeper positions himself independently to always be in the best position to stop a shot towards our goal. In some cases, when ball is located near to the field's edges formation positions are adjusted not exceed its limits.

5.8.2 11-Players Server Version (0.6.6)

Attacking group which consists of four positions:

Fc *Forward center*

Fl *Forward left*

Fr *Forward right*

Sf *Support Forward*

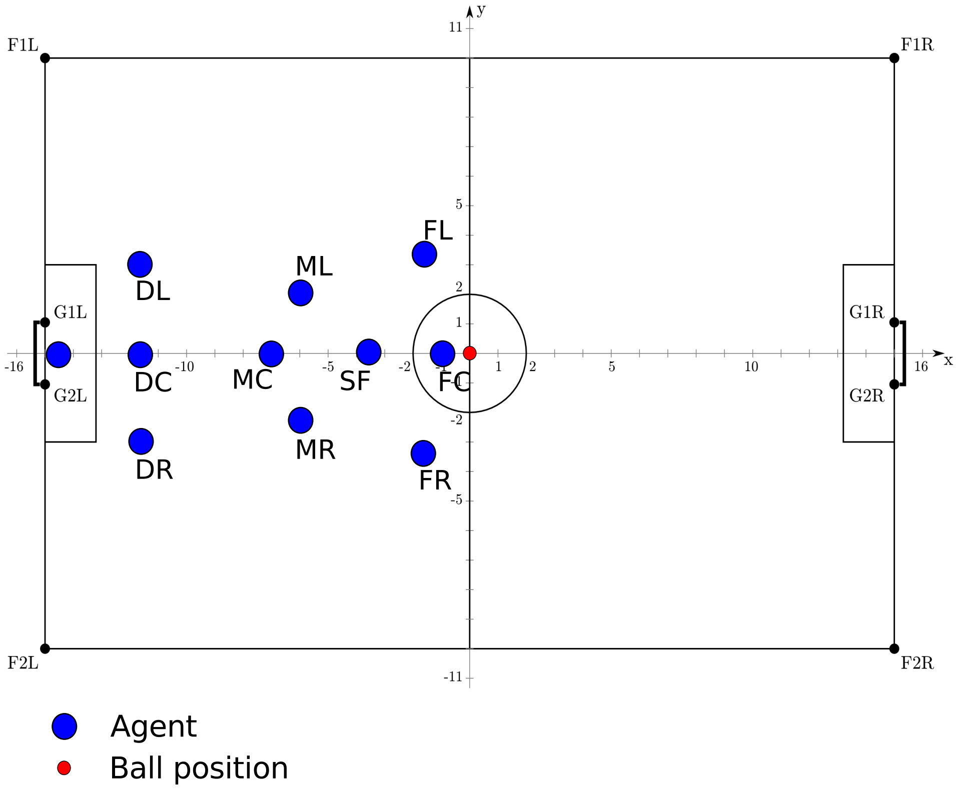

Figure 5.9: Formation Role Positions for 11 vs 11.

Defensive group which consists of three positions:

Dc *Defender center*

Dr *Defender right*

Dl *Defender left*

Finally, midfield group which consists of two positions:

Mc *Midfielder center*

Ml *Midfielder left*

Mr *Midfielder right*

Figure 5.9 depicts how the different role positions of the formation shown in the soccer pitch for the newer server version in which team consists of eleven players. Forward group is based on the same principle as the previous version's approach. In addition, a player is added beyond the forward center's position. In the midfield region there are now three players. Midfield center's position is behind with an offset distance from the forward center's position. Moreover, the other two midfielder positions are on either side of the midfield center position in an angle and a distance offset which are determined and dynamically changed according to the ball's exact coordinate. Defense line is exactly the same as it was in the previous version.

5.9 Role Assignment Function

In this section we present the role assignment function. This function after the evaluation of the current beliefs of the game state and the optimized active positions, tries to assign roles for all agents. This will prove to be very helpful on the next coordination steps when we will have to find positions for the support subset's agents. Given a computed team formation we have to assign roles to the active subset's players. As you already knew, positions for active agents are strictly connected and near to the ball's position on the field. So, for N active players we choose N team formation positions which minimize the distance from the ball. Roles which these positions represent will be assigned to the active players. The other team roles will be available to the support subset during support coordination process. Figure 5.10 shows how the role assignment function works. Active players will be assigned the red team roles due to the fact that they are located near to the ball's position. Once these positions will be bounded by the active players, the only roles that support players can compete about will be the grey colored positions. A naive role mapping would have assign roles permanently to specific players. This will perform poorly in such a dynamic environment. It would be also weak in situations where an agent assigned to a defensive role may end up out of position without being able to change roles with another player who may be in a better position to defend our goal. In our approach, every role mapping is calculated with a full sense of the world's state, resulting to a dynamic and unpredictable way of assigning roles to the agents. During testing, there were several cases in which a forward player ended up to have a defense role at the end of the game or the opposite.

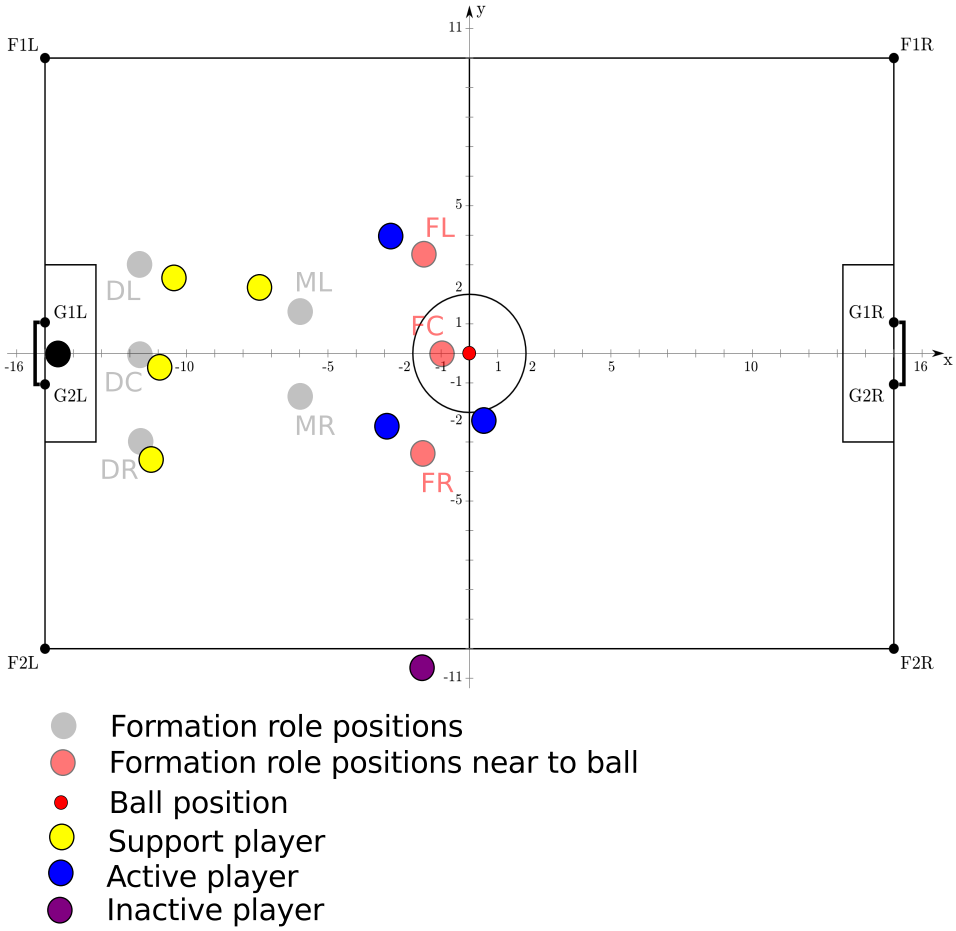

Figure 5.10: Role Assignment Function.

5.10 Positions for Support-Subset

In this section, we are going to discuss about which positions support subset's agents will be assigned. In an ideal case, we would have same number of support agents and same number of positions, this is going to happen when inactive subset is completely empty. In this case, this section has not any meaning. In other cases, when there are agents who are not able to know their positions or seeing ball, we have to decide about the positions of the team's formation which will be eliminated from the support coordination. Considering the ball's position, we have to make sure that there will be positions for support players near to the ball. So, given N players in the support subset we simply compare these team's formation position to find the N closest to the ball positions. Coordination's final

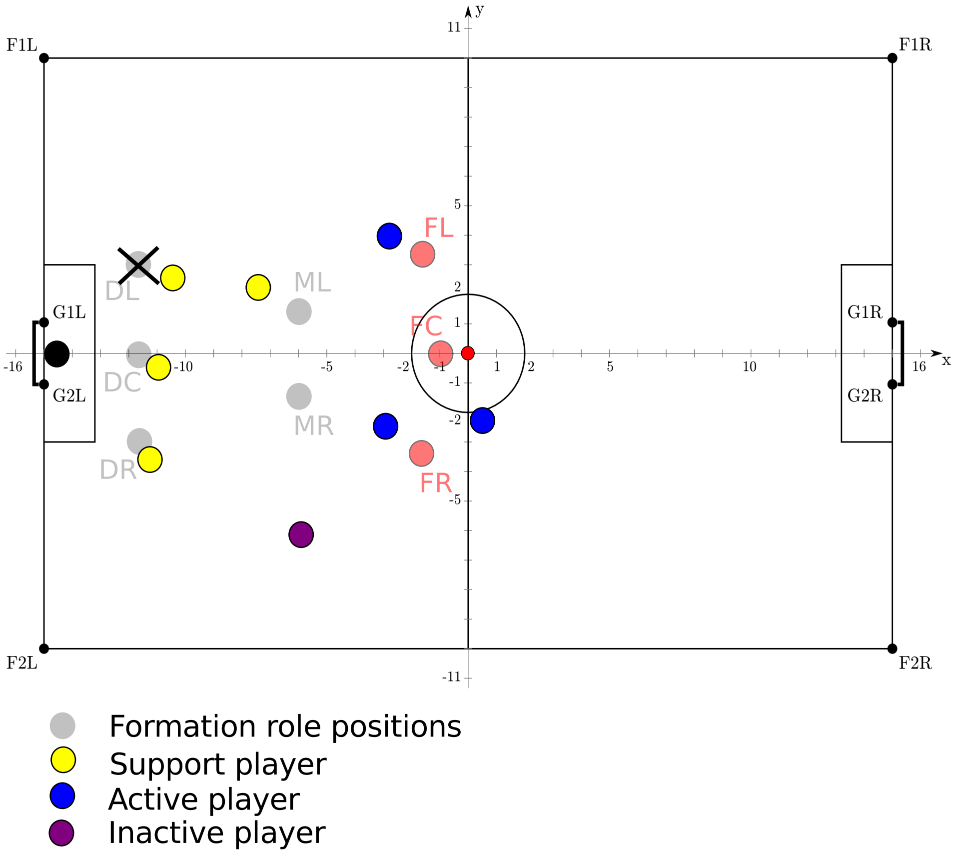

Figure 5.11: Support Positions.

step will find an optimized way to map support subset's agents to these positions.

5.11 Support-Subset Coordination

This is the final step of the coordination process. Until, now we have calculated the optimal mapping of the active subset's agents. So, it's time to find a mapping which will give as an optimal solution for the support agents as well. Given a support positions set which has been discussed in the previous two sections we have to assign each agent from the support subset in a specific position from the set. Using a greedy algorithm which would calculate all possible mappings to find the optimal one was the first solution to this problem. However, a brute force approach would only be applicable for the previous server version in which each team consists of nine players, in which we would have to calculate all possible mappings only for the support subset which consists of five players at maximum. This means a factorial complexity about: $\leqslant 5! \Leftrightarrow \binom{5}{5} = 120$ mappings. Unfortunately, moving from nine to eleven players this can be a problem, having to calculate at worst case $7! \Leftrightarrow \binom{7}{7} = 5040$ mappings.

It could be difficult for an agent to calculate all these mappings in real-time without any delay in sending effector messages to the server. We find the solution in a UT Austin Villa's paper [1] which was appeared in the RoboCup international Symposium in Mexico,2012. A dynamic programming implementation which is able to compute an optimal solution within the time constraints imposed by the decision cycle's length ($\approx$ 20ms).

Algorithm 5 Dynamic programming implementation [1]

```
Inputs: SupportPlayers = {A_1, A_2, ..., A_n}
SupportPositions = {P_1, P_2, ..., P_n}
Outputs: OptSupportMap
OptSupportMap = ∅
for k = 1 → n do
  for each α in SupportPlayers do
    S = (n-1 choose k-1), sets of k-1 agents for supportPlayers - {α}
    for each s in S do
      SupportRoleMap m_0 = RoleMap[s]
      SupportRoleMap m = m_0 ∪(α → P_k)
      OptSupportMap[{α} ∪ s] = mincost(m, OptSupportMap[{α} ∪ s])
    end for
  end for
end for
```

This dynamic Algorithm 5 is based on a key recursive property. This property stems from the fact that for every mapping there is a subset of a lower cost with which we can reduce the cost of the complete mapping by augmenting it with that of the subset's lower cost mapping. An example of this procedure is shown in Table 5.11. As we see in this table, an optimal mapping is built iteratively for position sets from $\{P_1\}$ to $\{P_1, P_2, ..., P_n\}$. In every step of this algorithm we use the lower cost's mapping for a subset of agents and positions which are compatible with our current mapping.

$\{P_1\}$	$\{P_1, P_2\}$	$\{P_1, P_2, P_3\}$
$A_1 \to P_1$	$A_1 \to P_2, min(A_2 \to P_1)$	$A_1 \to P_3, min(\{A_2, A_3\} \to \{P_1, P_2\})$
$A_2 \to P_1$	$A_1 \to P_1, min(A_3 \to P_1)$	$A_2 \to P_3, min(\{A_1, A_3\} \to \{P_1, P_2\})$
	$A_2 \to P_2, min(A_1 \to P_1)$	$A_3 \to P_3, min(\{A_1, A_2\} \to \{P_1, P_2\})$
	$A_2 \to P_2, min(A_3 \to P_1)$	
	$A_3 \to P_2, min(A_1 \to P_1)$	
	$A_3 \to P_2, min(A_2 \to P_1)$	

Table 5.1: Mappings Evaluated During Dynamic Algorithm [1].

Remind that in the *Kth* iteration of the algorithm, each agent will be assigned to the P_K position. Then the possible positions K-1 will be assigned to the other n-1 agents. These assignments result in a total of $\binom{n-1}{k-1}$ mappings to be evaluated in each iteration. Summing to $\sum_{i=1}^{N} \binom{n-1}{k-1}$ possible mappings.

$$\sum_{i=1}^{N} \binom{n}{k-1} = \sum_{i=0}^{n-1} \binom{n-1}{k} = 2^{n-1}$$

Therefore, the total number of mappings that we have to calculate their costs using this approach are $n2^{n-1}$. For nine players in each team, this algorithm would not have any impact in making coordination faster as the previous brute force algorithm will have 5! (120 mappings) not much bigger computational complexity than this approach $5 * 2^4$ (80 mappings). However, in the new version of soccer simulator in which we can have even seven players in our support subset this approach gives us great improvement in our coordination time, $7 * 2^6$ (448 mappings) $\ll$ 7! (5040 mappings).

5.12 Mapping Cost Computation

In this section we present how each mapping's cost is computed. This function serves our approach in two cases. First, in active coordination in which we want to find an optimized mapping between active agent and the possible active positions. Second, in support coordination in which an optimized mapping between support agents and the team's formation positions have to be computed.

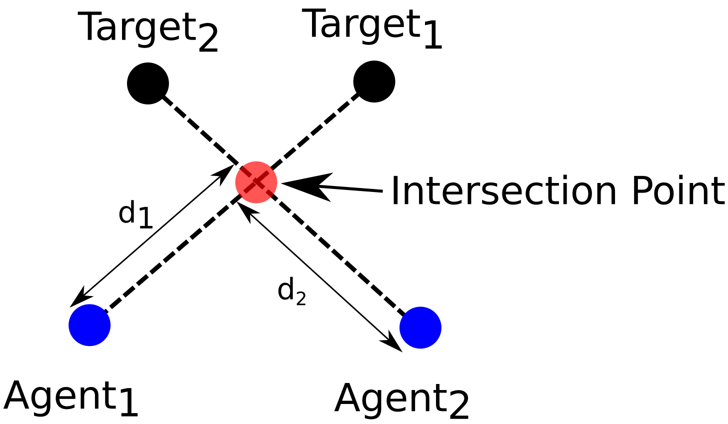

Figure 5.12: Collision Detection Approach.

5.12.1 Properties for Support-Subset Mapping Cost

For support players things were easy. In support coordination we have same number of agents and positions. So there only two properties:

1. **Total distance** C_d - Total distance agents have to travel in order to reach in their optimized mapping positions. It is a positive cost, so agents will try to minimize this cost.

2. **Possible Collisions** C_c - In each mapping we check every combination of two agents and their assigned positions if is is possible for them to collide with each other. For each agent and his target position there is a straight line which we assume approximately as his route to the target. If these lines intersects in a point which has almost the same distance from each agent then we add a very big cost in this mapping. It is a positive cost and agents will try to minimize it. Figure 5.12 shows the idea behind the detection of a possible collision between two agents. In order to detect a possible collision these two distances d_1,d_2 have to have a small difference between them.

$$TotalCost_i = C_{d,i} + C_{c,i}$$
$$OptimizedCost = \arg\min_i(TotalCost_i)$$

As you can realize, the mapping with the smallest cost will be chosen by coordination's executor.

5.12.2 Properties for Active-Subset Mapping Cost

In active coordination we have two agents and a maximum of nine positions. It is obvious that we have to take into consideration more than the two above properties. Using the same support's properties will force our players to go always to the nearest positions. So, we had to think about other properties in order to assign target positions for our players which will be valuable for the team's defensive and offensive movement without the ball. These properties are shown below:

1. **Total distance** C_d
2. **Field's value** C_v - Agents will try to maximize this cost according to the game's state and the value which has every position in the field, see Section 5.5. For example, in an attacking phase agents will try to select positions which maximize this value. It is a negative cost.
3. **Possible Collisions** C_c
4. **Close routes** C_r - Agents routes should be safe, so we calculate the difference between start positions' distance and target positions' distance. This cost is a negative cost and agents will try to maximize this.
5. **Neighboring positions** C_p - In general agents will try to avoid be assigned in neighboring positions. So if their target positions are near to each other this is going to add more cost. It is a negative cost and agents will try to maximize this.
6. **Positions aligned to X-axis** C_a - We want team stretching into the field in order to have players in most regions of the soccer pitch. Agents will try to maximize their Y-axis difference and this cost is negative too.

$$Totalcost = C_{d,i} - C_{v,i} + C_{c,i} - C_{r,i} - C_{p,i} - C_{a,i}$$
$$OptimizedCost = \arg\min_i(TotalCost_i)$$

The same principle applies here too, the mapping with the smallest cost will be chosen by coordination's executor.

Chapter 6

Results

In this Chapter are presented the results of our approach in every part of our work. We are going to see what we achieve in motions part, in communication part, in coordination part and finally and most important the overall results which is the real competitive soccer matches against other teams which participated in Robocup competition of 2011 held in Istanbul.

6.1 Impovements in Movement

This section presents the improvements we have done in the motions' part. In general, as we have said above motions files are not created by us but from other teams or other platforms such as Webots Simulator. The only thing that we could do is to try improving these motions until we have reached an adequate result for our team. Table 6.1 shows the improvements made in motions whenever was possible. Optimized walk motion has reached a speed of .45m/s which is comparable but much slower than the UT Austin Villa's walking engine which produces a walk motion of .71m/s. Furthermore strong kick movement has reached a 5.5 meters range in just 2.5 seconds. For turn motion, we use webots motion files in which we have achieved a turn with a speed of 30 degrees per second.

Motion Version	**Walk(m/s)**	**Turn(d/s)**	**Kick(m)**	**Strong Kick(m)**
Webots (Text-Based)	0.11	21	3	-
FIIT (XML)	0.22	25	3 (4 sec)	4 (5 sec)
AST_3D	**0.45**	**30**	**3** (2.5 sec)	**5.5** (2.5 sec)

Table 6.1: Motion's Performance Improvement

Communication Phase	Ideal (Cycles)	During Match (Cycles)
Init Messages	24 (.48 Sec)	24 (.48 Sec)
Coordination Messages	24 (.48 Sec)	42.5 (.85 Sec)
Action Messages	24 (.48 Sec)	24 (.48 Sec)

Table 6.2: Communication Results in Ideal and Match Conditions

6.2 Communication Results

Testing communication process through ideal external communication when only our team has the ability to send messages gave nice results. Agents were able to "hear" all their teammates in an averaged 24 Server-Cycles. However, even in competition's situations when both teams have the ability to send messages to their teammates the results remained approximately the same. Table 6.3 presents the communication phases' performance during communication process. We can see that there are not serious delays in these communication phases. This happens due to the fact soccer simulation server does not allow players to send messages in the same server cycle. We take advantage of the fact that there are separately tracked capacities for both teams, because teams should not be able to block the hear perceptors of their opponents by shouting permanently. In fact, we send messages every three cycles, so, it does not a restriction for our team, and server allows our team to shout messages in most cases.

6.3 Goalkeeper

Goalkeeper's behavior was tested against the best team in Robocup 3D Simulation League UT Austin Villa. To determine his ability to stop opponents from scoring we first use a goalkeeper which had an "empty" behavior in which he was not able to perform any movement or track the ball, standing useless at the center of our goal. Opponent team managed to score seven goals in this occasion. However, when goalkeeper made use of his current developed behavior he achieved to reduce concede goal from seven to three.

6.4 Coordination Results

To be written...

Advantages... 1. 2. 3. screenshots from matches without opponents which demonstrate these advantages screenshots from real matches which demonstrate these advantages

Drawbacks... 1. 2. screenshots from matches without opponents which demonstrate these advantages screenshots from real matches which demonstrate these advantages

6.5 Overall Results

In order to test our software in the most realistic way. We decided to play against teams that have already participate in robocup soccer simulation competition. Most of the teams have been participating in this competition for more than one year and consists of more than one members. We have select nine teams from Instabul's competition and one team(MAK) from Iran open 2011. These teams are:

RoboCanes University of Miami, USA

UT Austin Villa University of Texas at Austin, USA

NomoFC Osaka University, Japan

OxBlue University of Oxford, UK

L3MSIM Paris8 University,France

Kaveh Shahid Rajaee University, Iran University of Science and Technology, Iran

beeStanbul Istanbul Technical University, Turkey

Farzanegan Farzanegan high school, Iran

Team	W	D	L	AGD[3]	Games
UTAustinVilla	0	0	4	-5.2	4
Robocanes	0	0	1	-6.0	1
BeeStanbul	0	0	3	-4.0	3
NomoFC	1	2	0	+0.3	3
Rail	0	4	0	0.0	4
OxBlue	0	0	2	-1.5	2
FUTK3D	0	5	0	0.0	5
FARZANEGAN	1	1	0	+0.5	2
MAK	2	0	0	+2.0	2
L3M-SIM	3	2	0	+0.6	5

Table 6.3: Full-Game Results

MAK Ehsan Mosavi, University Of Kerman Mehravaran ,3D Robotics, Iran

FUTK3D Fukui University of Technology, Japan

All teams' binaries are from SimSpark Wiki -Previous Events Binaries. All games have 10 minutes duration same with real competition matches in Robocup. Server and monitor were running in the same machine[1]. Each team binary was running in separate machines[2].

After all these matches against teams who have participated into one or more Robocup competitions we have gained a lot of experience and we have seen how our team reacts in different situations in such a dynamic environment. Due to lack of dynamic movements, our agents has poor movement especially in comparison with the RoboCup Simulation league's best teams. However, we were able to perform well and score some goals against weaker teams of this competition. Better movement will give us exactly what we need in order to be competitive towards the best teams of the league. Judging by the results, I am absolutely sure that we could compete in equal terms with other teams for a position in simulation 3D league either in an open competition or in Robocup itself.

[1] **Server**: Intel Core 2 Duo 3.16 Ghz, 5.8GiB Ram
[2] **Client1**: Intel Core 2 Duo 1.86 Ghz, 2GiB Ram
[2] **Client2**: Intel Quad Core i5 3.3 Ghz,4GiB Ram
[1] AGD: Averaged Goal Difference

Chapter 7

Related Work

7.1 UT Austin Villa

[8] UT Austin Villa is the most known and the best team which is participating in the Robocup's simulation league. Its first appearance was in the Robocup 2007 held in Atlanta, U.S.A, in July 2007. It belongs to University of Texas and consists of five members, professor Peter Stone, graduate students Patrick MacAlpine and Samuel Barrett and finally two undergraduate students Nick Collins and Adrian Lopez-Mobilia. The main characteristic of this team is its state-of-art dynamic movement. Its fast and stable walk is recognizable and offers them great results. A typical example of this great team's results can be that in Robocup's competition in Istanbul 2011 this team won all 24 games it played and scored a total of 136 goals without conceding any.

In their last paper [1] about positioning, it is explained their approach of player positioning in the field. First, a full team formation is computed. Second, each players calculates the best assignment of players according to his belief about the world. Finally, a coordination mechanism is used to choose among all players' suggestions. This coordination mechanism using a voting system. Players assignment with the most votes will be used as a result.

I am not the appropriate person to criticize their longterm work and contribution to the Robocup's simulation league, as I deal with this league only for a few months. However, I would like to mention that in our approach there is a major difference in the way that players coordinate their actions. The separation of the team into subsets makes it easier to solve all these problem caused due to complexity constraints. Furthermore, we are using active's group players in order to have a better role assignment to positions near to ball which have huge importance in games like soccer. Finally, I wish we had such a perfect movement controller like UT Austin Villa's one. It would be a nice challenge to compare these two coordination systems in the same movements' level.

7.2 Robocanes

7.3 BeeStanbul

[9] The beeStanbul project from the Artificial Intelligence and Robotics laboratory (AIR lab) at Istanbul Technical University (ITU) is the first initiative from ITU to participate in RoboCup competitions. It consists of five members and has been participating in the Robocup's competitions since RoboCup 2010 held in Singapore. It is a nice team which accomplished to qualify up to second round in the last Robocup competition in Mexico 2012.

First of all, they are making use of both static and dynamic movements and their walking machine is more than adequate. Concentrating in their work at the coordination part of their project.They split team agents into three groups, defenders and attackers. The attackers group involves the forward and the midfielder agents while the defend- ers group involves only the defender agents.Since two agents are assigned to the goalkeeper and the forward roles, the remaining seven agents are to be assigned to these roles. This is accomplished by a distributed Voronoi cell construction approach in which each agent calculates its own cell independently from that of the others. Therefore, every agent has a differently shaped cell and these can overlap. The time complexity of the method is $O(n^2)$ where n is the number of agents in the team. After constructing the cell for itself, each agent determines the center of the cell as its new target. Agents become closer to each other by using this strategy. In their approach, only teammates in the viewpoint of the agent are considered. So we can realize that there can be situation in a soccer game when each agent who computes his own cell could be completely unaware if any of his teammates is located in his field of view. Having a better knowledge of teammates position in the field is a key feature in our approach.

7.4 Kaveh

7.5 L3MSIM

7.6 FUT-K_3D

7.7 Farzanegan

Team formation is an important part in soccer simulation system, since it allows team members to focus on team-goal, which is simpler than a global-goal over an entire system. In addition team formation allows sharing of information.[9] We presented our approach to multi-agent collaboration, based on strategical positions, roles and responsibilities. Also like human soccer, each agent has a strategical position that defines its default position and movement range inside the soccer field. So we have classified our strategical position into two categories: initial and game-paly. Strategical positioning, roles and responsibilities are inevitable in soccer domain. Each agent has its own movement range based on its role and responsibilities. When keeping an eye on the ball, the movement range will determine whether the agent should go for the ball, or leave it to its team mates.

Chapter 8

Conclusion

We have presented a team's framework for the Robocup Simulation League 3D - a physically realistic environment that is partially observable, non-deterministic, noisy and dynamic, as well as a dynamic coordination system which evaluates all the necessary variables and be executed only by one player. Creating a team's framework from scratch was a big challenge especially when this project does not depend in any other third party software's part except from motions files and the dynamic programming implementation created by UT Austin Villa. In general, teams participating in this league consists of more than two members in most cases. It is my personal belief that in order to be competitive in this league there has to be a team effort in which each member should concentrate in a single part and not in the whole problem.

8.1 Future Work

Reaching in a level to oppose teams that have already participated in this robotic soccer competition gives us an incentive to keep working in order to improve further our framework. In this section, we present some of our these improvements.

Dynamic Movement

Most of the teams which have been participating in the Robocup's simulation soccer league make use of dynamic movement. This is a major drawback for our side and I really hope this issue to be resolved in the near future.

Passing

Hopefully, is a short-term goal for us to add passing feature in our framework. You could realize that passing is a key attribute in every soccer team's success. There have to be improvements in team formation in order for passing to be implemented well into it.

Testing and Debugging in New Server's version

There are things to be tested in order our team to meet the standards of the new Server's Version 0.6.6 in which there are some changes with the most important that there are now eleven players for each side and field's size has changed. It will be easy to make these changes in our source code, as the whole code is written in a way that allows these changes to be done easily.

Participation in Robocup

Robocup is a well-known competition especially for people who are interested in robotic soccer. Since I started this project, during the last winter semester in the course of Autonomous Agents, I was having the ambition for our team to participate in this league. It will not be easy to be competitive at once but it will be a nice experience. Furthermore, we are going to have the opportunity to test our agent in real conditions.

References

[1] MacAlpine, P., Barrera, F., Stone, P.: Positioning to win: A dynamic role assignment and formation positioning system. In: Proceedings of the RoboCup International Symposium. (2012) xi, xiii, 73, 74, 83

[2] Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., Osawa, E., Matsubara, H.: Robocup: A challenge problem for AI. AI Magazine **18**(1) (1997) 73–85 5

[3] Robocup: Soccer simulation league wiki Only available online: http://wiki.robocup.org/wiki/Soccer_Simulation_League. 15

[4] SimSpark: Wiki Only available online: http://simspark.sourceforge.net/wiki. 15

[5] Abeyruwan, S., Seekircher, A., Stoecker, J., Visser, D.U.: Roboviz monitor Only available online: https://sites.google.com/site/umroboviz/. 19

[6] Papadimitriou, V.: Localization in simspark environment (2012) Autonomous Agents Fall Semester, Only available online: http://www.intelligence.tuc.gr/~robots/ARCHIVE/2011w/projects/TUCagent3D/home.html. 30

[7] Michel, O.: Webots: Professional mobile robot simulation. Journal of Advanced Robotics Systems **1**(1) (2004) 39–42 39

[8] UTAustinVilla: Utaustinvilla Only available online: http://www.cs.utexas.edu/~AustinVilla/sim/3dsimulation/. 83

[9] Demirdelen, B., Toku, B., Ulusoy, O., Sonmez, T., Ayvaz, K., Senyurek, E., Sariel-Talay, S.: beestanbul robocup 3d simulation league team description paper

2012 (2012) Only available online: http://air.cs.itu.edu.tr/publications-1/beeStanbul_TDP2012.pdf. 84