Technical University of Crete, Greece
Department of Electronic and Computer Engineering

Player Behavior and Team Strategy in SimSpark Soccer Simulation 3D

Georgios Methenitis

Thesis Committee
Assistant Professor Michail G. Lagoudakis (ECE)
Assistant Professor Georgios Chalkiadakis (ECE)
Professor Miron 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 and close to the team's goal. 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 in the field. 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. Last but not least, is the coordination, which is a factor of major importance in a multi-agent system like this. Agents, should be able to coordinate their actions through communication or other effectors in order to work as a team and towards the team's success. This thesis describes, how we created every single part needed by a team for the Robocup soccer simulation league. Additionally, we emphasized in agents' coordination and cooperation.

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 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 percept their surroundings and act just like human soccer players. Therefore, everything in the robots' world, start from the absolute zero. 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 until 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 demonstrate the main platform in which the simulation league based on. Continuing to chapter 4, where the core ideas and an outline of the architecture of our proposal is discussed. Moving on to chapter 5, where there is an esxtensive documentation about the main objective of this thesis which is coordination of the agents through communication channel . In Chapter 6 a discussion on the results is taking place by providing several experiments in order to evaluate our work. The following chapter 7, presents similar systems developed by other robocup teams including a brief comparison between those systems and ours. Future work and proposals on extending and improving our framework are the subject of the chapter 8. The final chapter 9 serves as an epilogue to this thesis, including a small overview of the system and some long terms plans about it.

Chapter 2

Background

2.1 RoboCup Competition

RoboCup is an international robotics competition founded in 1997. The aim is to promote robotics and AI 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: ”By mid-21st century, 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.” Something that may seem impossible with today's technology. I would say that a more realistic goal is to make a team of robots playing soccer like humans and not better than them.

2.2 RoboCup Soccer

The main focus of the RoboCup competitions is the game of football/soccer, where the research goals concern cooperative multi-robot and multi-agent 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.

2.2.1 Humanoid

In the Humanoid League, autonomous robots with a human-like body plan 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.

2.2.2 Middle Size

Middle-sized robots of no more than 50 cm diameter play soccer in teams of up to 6 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. The research focus is on full autonomy and cooperation at plan and perception levels.

2.2.3 Simulation

This is one of the oldest leagues in RoboCupSoccer. 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 2 subleagues: 2D and 3D.

2.2.4 Small Size

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.

2.2.5 Standard Platform

In this league all teams use identical (i.e. standard) robots. Therefore the teams concentrate on software development only, while still using state-of-the-art robots.

Omnidirectional vision is not allowed, forcing decision-making to trade vision resources for self-localization and ball localization. The league is based on Aldebaran'As Nao humanoids. "Kouretes" from Technical University of Crete is the only Greek representative in this league, having continuous participations and lots of discriminations.

2.3 RoboCup Rescue

2.3.1 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. Greece has also a participation (2009) in this league by the Aristotle University's team called "P.A.N.D.O.R.A".

2.3.2 Simulation League

The purpose of the RoboCup Rescue Simulation league is twofold. First, it aims to develop 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.

2.4 RoboCup @Home

The RoboCup @Home league aims to develop service and assistive robot technology with high relevance for 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'A abilities and performance in a realistic non-standardized home environment setting. Focus lies on the following domains but is not limited to: 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.

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

2.5.1 Soccer

2-on-2 teams of autonomous mobile robots play in a highly dynamic environment, tracking a special light-emitting ball in an enclosed, landmarked field.

2.5.2 Dance

One or more robots come together with music, dressed in costume and moving in creative harmony.

2.5.3 Rescue

Robots identify 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

SimSpark

SimSpark is a generic physical multiagent simulator system for agents in three-dimensional environments. It builds on the flexible Spark application framework. It is used as the official Robocup 3D simulation server. In comparison to specialized simulators, users can create new simulations by using a scene description language. SimSpark is a powerful tool to state different multi-agent research questions.

3.1 Soccer simulation

RoboCup is an initiative to foster artificial intelligence and robotics research by providing a standard problem in the form of robot soccer competitions. **rcssserver3d** is the official competition environment for the 3D Soccer Simulation League at RoboCup. It implements a soccer simulation where two teams of up to eleven humanoid robots play against each other. You can see the soccer field in figure 3.1.

3.2 Server

The SimSpark server hosts the simulation process that manages the simulation. It is responsible for advancing the simulation. The simulation state is constantly modified during the Simulation Update Loop. Objects in the scene change their

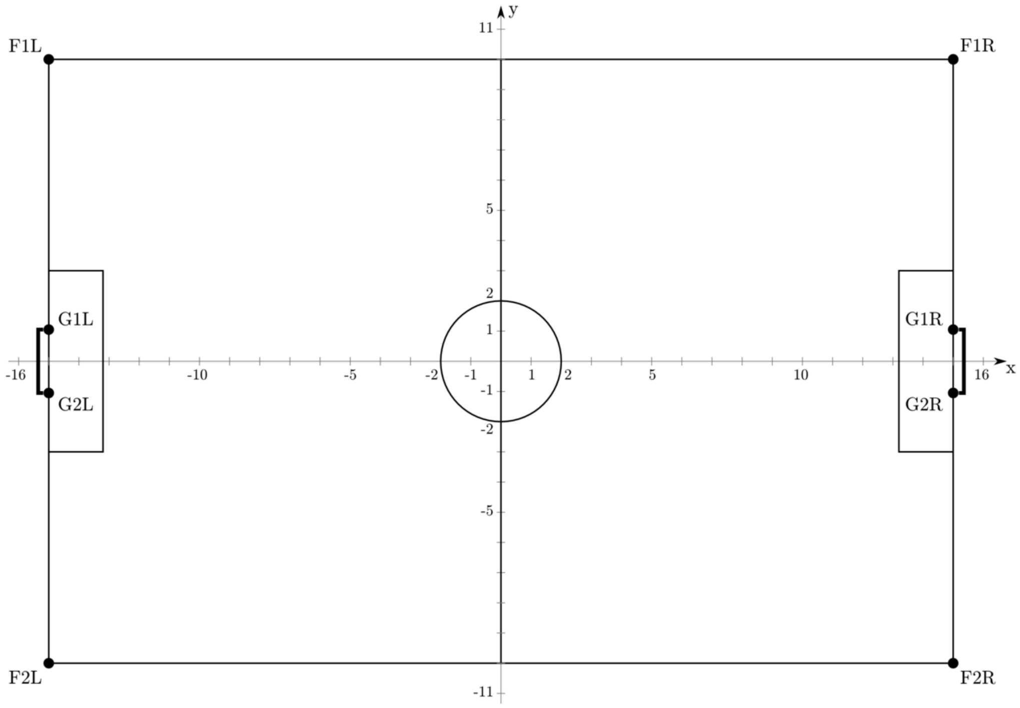

Figure 3.1: Simulation Soccer Field

state, i.e. one ore more of their properties like position, speed or angular velocity changes due to several influences. They 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. Another responsibility of the server is to keep track of connected agent processes. 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 that an agent triggers using its available effectors. The server can, depending upon its config, renders the simulation itself. It implements an internal monitor that omits the network overhead. Additionally, it supports streaming data to remote monitor processes which take responsibility for rendering the 3D scene.

3.3 Simulation Update Loop

SimSpark implements a simple internal event model that immediately executes every action received from an agent. It does not try to compensate any network latency or compensate for different computing resources available to the connected agents.A consequence is that SimSpark currently does not guarantee that events are 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 Network Protocol

The server exposes a network interface to all agents, on TCP port 3100 by default. When an agent connects to the server the agent must first send a CreateEffector message followed by a InitEffector message. Once established, the server sends groups of messages to the agent that contain the output of the agent's perceptors, including any hinge positions of the model, any heard messages, seen objects, etc. The exact messages sent depend upon the model created for the agent. Details of effector messages are given on the perceptors page. In response to these perceptor messages, the agent may influence the simulation by sending effector messages. These perform tasks such as moving hinges in the model. Details of effector messages are given on the effectors section.

3.5 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 that the server sends to the monitor is called Monitor Format. It is a customizable language used to describe the simulation state. 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 means 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 state to plugins. These plugins are intended to parse the game state and display it as an overlay, e.g. print out playmode and scores on screen.

3.6 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 agents, via the network protocol, for every cycle of the simulation. Perceptor messages are sent via the network protocol.There are both general perceptors that apply to all simulations, and soccer perceptors that are specific to the soccer simulation.

3.6.1 General perceptors

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 and three rotation angles. These rotation angles describe the change rates in orientation of the body during the last cycle. In other words the current angular velocities along the three axes of freedom of the corresponding body in degrees per second. To keep track of the orientation of the body, the information to each gyro rate perceptor is sent every cycle.

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

Frequency: Every cycle

HingeJoint Perceptor A hinge joint perceptor receives information about the angle of the correponding 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 every 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.

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

Frequency: Every cycle

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 vectors. The first vector describes the point of origin relative to the body itself and the second vector the resulting force on this point. The two vectors are just an approximation about the real applied force. The point of origin is calculated as weighted average of all contact points to which the force is applied, while the force vector represents the total force applied to all of these contact points. The information to a force resistance perceptor is just sent in case of a present collision of the corresponding body with another simulation object. If there is no force applied, the message of this perceptor is omitted.

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

Frequency: Every cycle, but only in case of a present collision.

Accelerometer This perceptor measures the proper acceleration it experiences relative to free fall. As a consequence an accelerometer at rest relative to the Earth's surface will indicate approximately 1g upwards. To obtain the acceleration due to motion with respect to the earth, this gravity offset should be subtracted.

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

Frequency: Every cycle

3.6.2 General perceptors

Vision Perceptor The Vision perceptor 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 8 markers on the field: one at each corner point of the field and one at each goal post. With each visible object you get a vector described in spherical coordinates. In other words the distance together with the horizontal and latitudal angle to the center of a visible object relative to the orientation of the camera.

Message format:

```
(See +(<name> (pol <distance> <angle1>
<angle2>))+(P (team <teamname>) (id <playerID>)
+ (<bodypart> (pol <distance> <angle1> <angle2
>)))+(L (pol <distance> <angle1> <angle2>)(pol
<distance> <angle1> <angle2>)))
```

Frequency: Every third cycle (every 0.06 seconds)

GameState Perceptor The game state perceptor delivers several information about the actual state of the soccer game environment. A game state message is started with the GS identifier, followed by a list of different state information. Currently just the actual play time and play mode are transmitted in each cycle. Play time starts from zero at kickoff of the first half, and 300 at kickoff of the second half and is given as a floating point number in seconds, to two decimal places.

Message format:

```
(GS (t <time>) (pm <playmode>))
```

Frequency: Every cycle

Hear Perceptor Agent processes are not allowed to communicate with each other directly, but 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.

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

Frequency: Every cycle

3.7 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. Effectors are the logical dual of perceptors. Effector control messages are sent via the network protocol. Details of each message type are shown in each section below. There are both general effectors that apply to all simulations, and soccer effectors that are specific to the soccer simulation.

3.7.1 General Effectors

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

Message format: `(scene <filename>)`

HingeJoint Effector Effector for all axis with a single degree of freedom. The first parameter is the name of the axis. The second parameter is a speed value, passed in radians 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 at the extremity until a new speed value is requested.

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

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 your agent to always append this command to your agent's responses.

Message format: `(syn)`

3.7.2 Soccer Effectors

Init Effector The init command is sent once for each agent after the create effector sent the scene command. It registers this agent as a member of the passed team with the passed number. All players of one team have to use the same teamname and different playernumber values.

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

Beam Effector The beam effector allows a player to position itself on the field before the start of each half. The x and y coordinates define the position on the field with respect to the field's coordinate system, where (0,0) is the absolute center of the field.

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

Say Effector The say effector permits communication among agents by broadcasting messages. In order to say something, the following command has to be employed.

Message format: `(say <message>)`

3.8 Model

SimSpark comes with Nao robot model for use by agents. The physical representation of each model is stored in an .rsg file.The Nao humanoid robot manufactured by Aldebaran Robotics. Its height is about 57cm and its weight is around 4.5kg. Its biped architecture with 22 degrees of freedom allows Nao to have great mobility. **rcssserver3d** simulates Nao nicely.

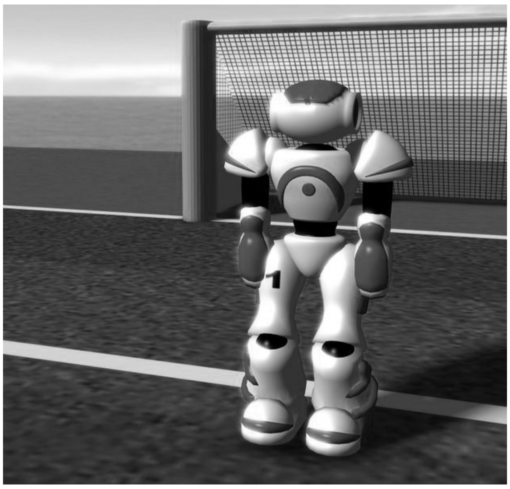

Figure 3.2: Nao in simulation screen

Chapter 4

Agent

In this chapter we are going to understand how the agent functions. Each agent consists from several parts which are described in detail.

4.1 Agent Architecture

Before seeing each part of the agent's software separately, it is time to describe the framework's architecture. Soccer Simulation Server known as rcssserver3d is responsible for sending to our agent perception messages. Communication layer is the one that handles all the received messages and pass them to the agent. In agent layer, these messages are handled by a message parser which is responsible for updating all beliefs of the agent. Consequently, all functions that require new perceptions start then. Now, agent is able to do what the behavior tells him. In our approach, only goalkeeper "runs" an independent behavior, the other eight field players start a communication procedure in order to inform the goalkeeper about the worldstate and their attributes. 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. We are going to describe coordination procedure later in a separate chapter. Communication controller and motion controller are responsible for handling the agent's requests for sending a message to the his team mates or a movement that he has to execute. These two controllers send in every cycle effection messages to the connection layer which will send them back to soccer simulation server.

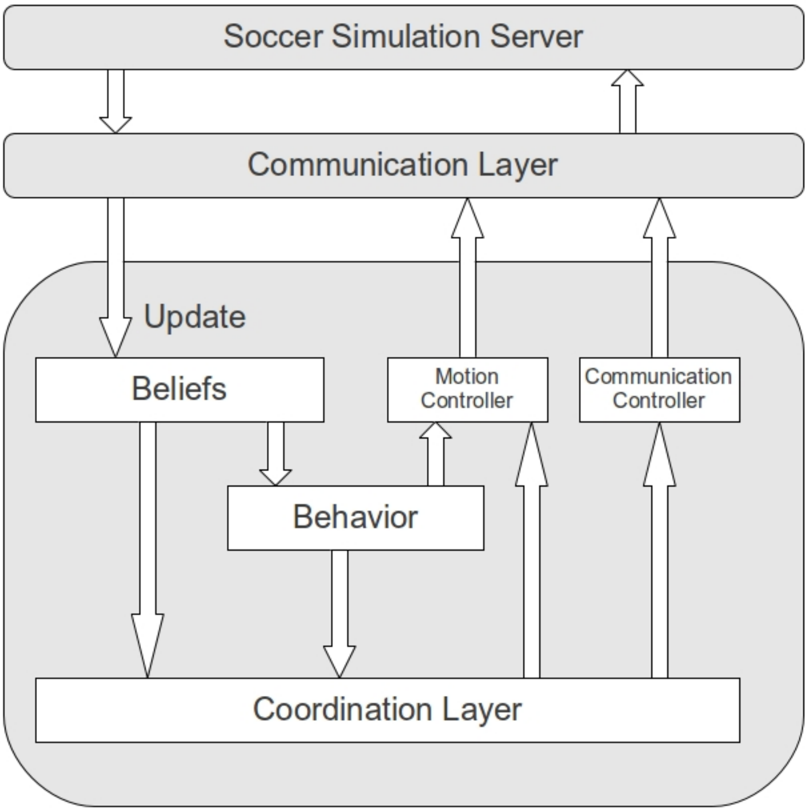

Figure 4.1: Agent's Architecture.

4.2 Connection

The SimSpark server hosts the simulation process that manages the simulation. It is responsible for advancing the simulation. So, each agent connects to this server. Agents receives messages from the server every 20ms; These messages includes information about all agent's perceptions.As we can see in the figure 4.2, SimSpark Server send to agents sense messages in the beginning of every cycle.

Each agent who is willing to send an action message, can send it in the end of his cycle, Server is going to receive at the same time it will send the next sense message.

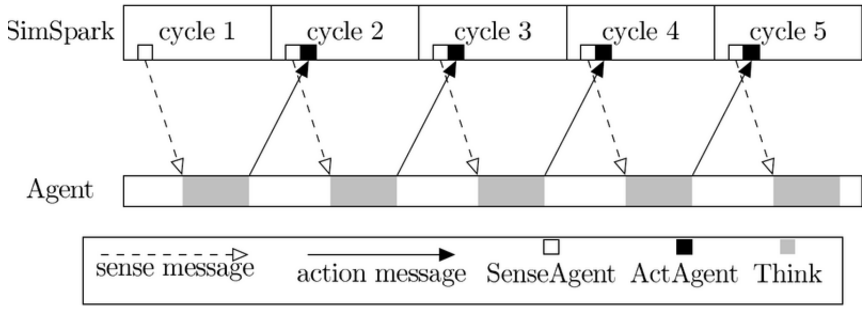

Figure 4.2: Simulation Update Loop.

4.3 Perceptions

Perceptions in simulation soccer are quite different in comparison with a real robots' competition. We do not receive data from agent's sensors but from the server, which send them to us in every cycle. These messages have this 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 just an example message our agent has been sent during game time. It includes information about the server time, the game state and time, the values of each one of his joints and data from vision, acceleration, gyroscope and force sensors.

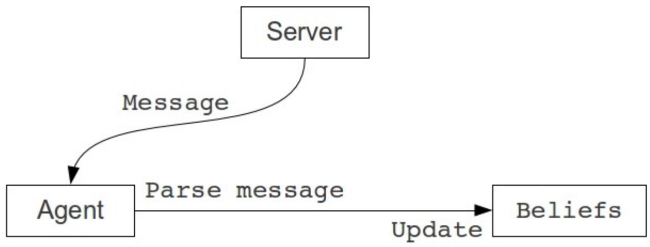

Figure 4.3: Beliefs Update.

4.4 Localization

Once we have all the necessary beliefs updated, it is time for us to use them in order to locate our agent in the field. Localization is created by Vassilis Papadimitriou in winter's 2011-2012 class Autonomous Agents. A brief description of the localization process is following.

Localization Process

Localization process is executed every three cycles (60ms) and when we receive observations from the vision perceptor. If we have visible objects in our sight we organize them in terms of their type. There are three types: Landmarks, Co-Players and Opponent Players. We make use of the Landmarks to find our position in the field. The Nao's restricted vision perceptor limits the field of view to 120 degrees. An example of this limitation is described in the figure 4.4. Localization process became possible through three main functions. The first function, takes two landmarks as arguments and returns to us a possible position for our agent. If our agent sees more than two landmarks, then this function is called

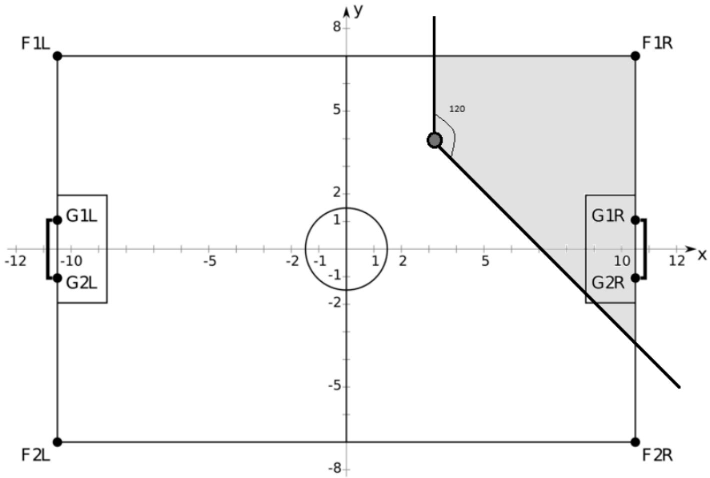

Figure 4.4: Nao's field of view.

for every combination of two landmarks and in the end we calculate the average position. If our agent sees less than two landmarks, then he has a complete unawareness of his position in the soccer field. The figure 4.5 shows how this function works. Except from the calculation of our position in the soccer field,

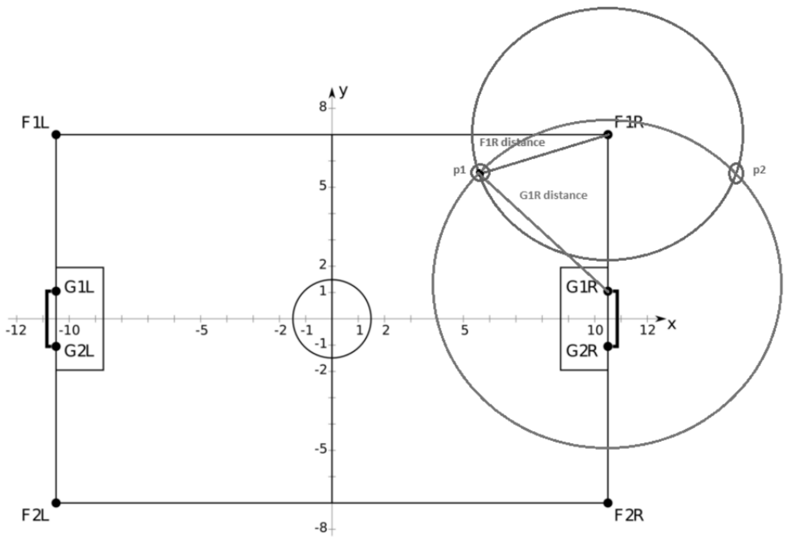

Figure 4.5: Localization.

localization is responsible to locate ball and other agents in the field. Knowing our position helps us locate other objects too. For every other object which is locates in our field of view, vision perceptor informs us about its vertical angle, its horizontal angle and its distance from our agent. This information is enough for the calculation of their exact positions. Finally, after the localization process end, we are able to have the following observations:

Our Position Only if our agent sees more than one landmarks.

Body Angle Only if our agent knows his position.

Other Agents Positions Only if our agent knows his position and other agents are located in the field of his view.

Ball Position Only if our agent knows his position and ball is located in the field of his view.

In the figure 4.6 we can see the results which are given by the localization process.

4.5 Localization Filtering

In absence of a stochastic localization system, we are forced to ensure that localization results are good enough for us to rely on.Due to the symmetry of the field's landmarks, localization is not always accurate enough to depend on. This, requires a kind of filtering for the observations we take by the localization process. The algorithm 1 describes the process of localization filtering. The general idea that we follow in our approach is that if our agent takes one thousand observations per minute it will be easy for him to not to take into consideration the observations with the biggest fault. In general, localization provides us with not consecutive faulty observations. To overcome this difficulty, we came up with a simple and clever approach. A queue full of observations is always gives us our agent's position in the field. When an observation is coming, we check if the queue is empty or full; If it is empty then we just add the observation into the

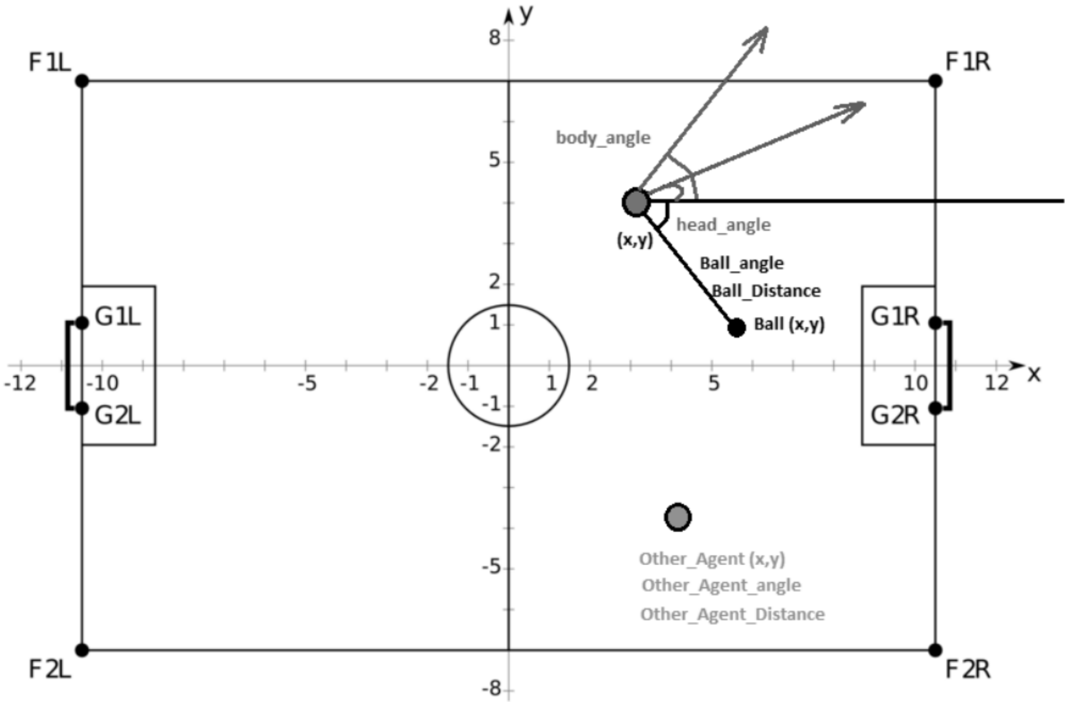

Figure 4.6: Localization Results.

queue. If it is full of elements, then we check if the new observation seems faulty in comparison to the average of the queue. If it seems faulty, we do not take it into account and we just remove an element from the queue. If not, then we add it to the queue. If queue is neither empty nor full, then we make the same procedure checking if it is a faulty observation, with the only difference that we do not remove any element if it is not.Localization filtering applies for both the calculation of our agent's position and the ball's position. Its result was the improvement of the localization results in an adequate degree in order to rely on them with more confidence. This filtering smooths the belief of our position and rejects every faulty observation.

Algorithm 1 Localization Filtering($Observation(x, y)$)

```
if x, y ≠ NaN then
   if size(Queue) = 0 then
      Queue.Add(Observation)
      MyPosition = AVG(Queue)
   else if size(Queue) < Max then
      if Observation ≉ AVG(Queue) then
         Queue.Remove()
      else
         Queue.Add(Observation)
         MyPosition = AVG(Queue)
      end if
   else
      if Observation ≉ AVG(Queue) then
         Queue.Remove()
      else
         Queue.Remove()
         Queue.Add(Observation)
         MyPosition = AVG(Queue)
      end if
   end if
end if
```

4.6 Motions

In robotics, we could define a motion as a sequence of joint poses. A pose is a set of values for every joint in the robot's body at a given time. For example, for a given set of n-joints a pose could be defined as:

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

Motions are very important part of every team take part in the simulation league. Most of the teams in this league make use of dynamic movement which is a major advantage for their side. In this approach, we are using motion files. Motion files are set of poses which has static and standard values for each joint for every movement. The difference between static motion files and dynamic movement is that dynamic movement takes into consideration the center of the body's mass and the direction in which we are want to head. This movement gives to the robot better body balance and fast movement especially in situations that the robot wants to change direction or to make a turn. In this approach we are using two kinds of static motion files. Text based and XML based motion files. Agent before initializes himself in the field read these files and saves them into the dynamic memory to be ready to use them without any need of reading them every time he needs them.

4.6.1 XML Based Motions

This motion files has been created from FIIT RoboCup 3D project. They are in XML structure and it was easy to implement them into our project. The following lines show the structure of these xml motion files.

```
<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>
```

It is easy to understand that each movement is split into phases. Each phase has a duration and values for every joint of the robot. Moreover, every phase has an index which points to the next phase. For example, we see that the first phase ”Start” has an index for the next phase: ”Phase1”. Phases with a finalize field help us to end each movement. For example, the phase:”Phase2” has a finalize index which points to the phase: ”Final”, this means that, if we want to end this motion, we should continue the motion with the finalize phase not with the next.

4.6.2 XML Based Motion Controller

Motion controller is responsible for handling the movement requests by the agent. Agent has not access in motion controller itself but he has access in the motion trigger. We can imagine this trigger as a variable which can only be changed by the agent. Each agent declares the movement he is willing to do in this variable. Motion controller reads this variable in every cycle and generates a string which is the result of his process. In the figure 4.7 we show the general architecture of the motion controller. Motion controller checks if there is a motion which is playing already. If yes, motion controller tries to finalize the playing movement in order to start playing the new requested movement. In the next figure 4.8 is described the exact motion sequence. In general, XML motions is created to include cycles. For example, walking motion has three main phases which create a cycle. If motion trigger does not change at the last phase, we will continue with the first phase not with the final. As we saw in the structure of every XML

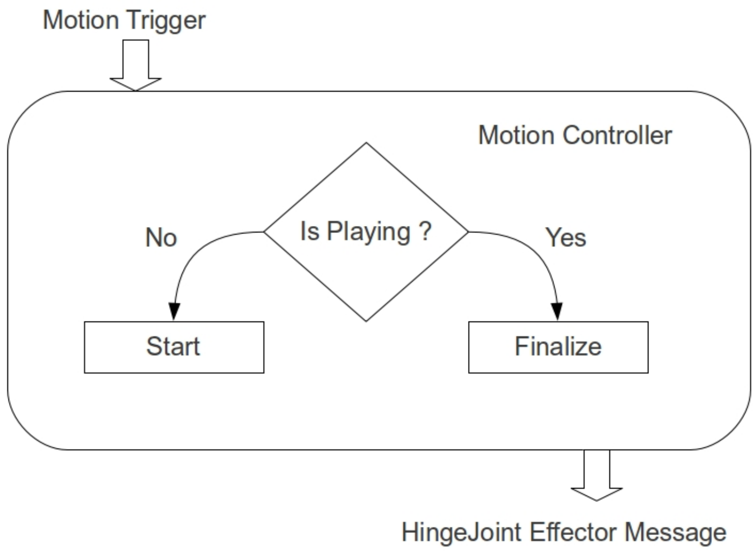

Figure 4.7: Motion Controller.

based motion file, each phase has a set of joint values. These values is in degrees. To generate motion for our agent we need to create a motion string. This string holds info about the velocity we want to give in every joint involved in the motion phase. This velocity can be calculated by:

$DesiredVelocity = AlreadyJointValue$ - $DesiredJointValue$

This is the velocity of every joint. Furthermore, every phase has a duration in which has to be executed. So, phase duration has to be divide with the duration of every server cycle. This will give us the number of cycles this phase will be playing.

$CyclesNumber = \frac{PhaseDuration}{CycleDuration}$

Now, we have the phase's velocity and the duration in cycles. We can calculate, how much will be the speed of every joint in order to reach to the desired

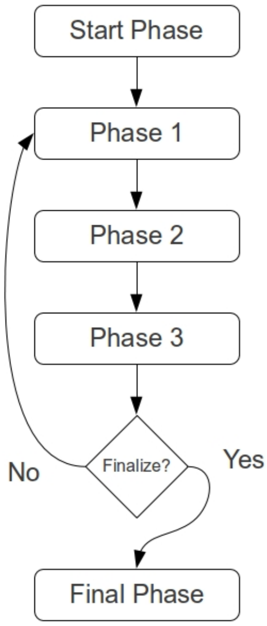

Figure 4.8: Phase Sequence.

joint value in this time limit.

$Velocity = \frac{DesiredVelocity}{CyclesNumber} degrees/sec$

This velocity is calculated for every involved joint in the motion. The final output of the motion controller will be send to the server.

4.6.3 Text Based Motions

The other kind of motion files that we use is created by Webots simulator. These text based motion files have simpler structure than the XML have. At the second row, there are the definition for all joints which are related to each motion. For example, walking motion requires only the joints from both robot's legs. The next rows from left to right have information for the duration of each pose, the pose name and finally the joints' values for each joint in the same order as they are defined in the second row.

```
#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,...
```

4.6.4 Text Based Motion Controller

Motion controller for text based motions is based on the same principle as the XML controller. The joint values in the motion files represent radians. So we convert these values into degrees and then we proceed with next steps. Each pose lasts for one or two cycles depending on the speed we want each motion to be executed. This motion controller could be customized easily to perform motions differently. There are parameters that can be changed such as:

Speed How fast we want pose to be executed.

Duration How many cycles from pose to pose.

Pose Offset Pose Offset = 2, we execute pose1,pose3,pose5,...

Hardness Factor Hardness Factor = 0.9, we multiply the velocity with this factor.

The velocity of every joint is calculated by:

$$DesiredVelocity = AlreadyJointValue \text{ - } RadiansToDegrees(DesiredJointValue)$$

$Velocity = \frac{DesiredVelocity*HardnessFactor}{Speed} degrees/sec$

This velocity is calculated for every involved joint in the motion. The final output of the motion controller will be send to the server.

4.7 Actions

Actions are the results of the agent's perception in combination with his procedure of thinking. In our approach actions are split into groups in terms of their complexity and their type.

4.7.1 Simple

First of all, simple actions which are make use only motions and have a simple structure. These simple actions are:

TurnToSeeBall This action results in turning the agent until ball is in his field of view.

TurnToBall This action turns agent towards the ball.

TurnToLocate This is the default action each agent does when he loses his position (sees less than two landmarks) in the field.

WalkToBall Agent walks towards the ball. He stops when the ball is close enough to him to shoot it.

StandUp Agent executes it when he is fallen on the ground.

PrepareKick Agent executes it before performs a kick. This action is needed in order to have a successful kick.

4.7.2 Complex

Complex actions are created to make use of more than one simple actions and motions and have a more complicated structure. These complex actions are:

GoKickBallToGoal This action uses WalkToBall in order the agent to reach the ball. In this action we use the agent's belief about his location in the field to help us find the direction in which the agent has to kick the ball. This action has a fsm logic. Figure 4.9 shows how looks like this action's execution.

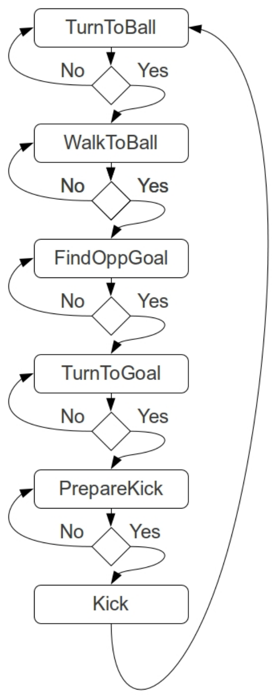

Figure 4.9: GoKickBallToGoal Action.

GoClearBall This action uses WalkToBall in order the agent to reach the ball. In this action we use the agent's belief about his location in the field to help us find the direction in which the agent should not kick the ball.

WalkToCoordinate This action takes the agent to a specific coordinate in the soccer field. To achieve this action we need to know our position in the field and the target coordinate. Agent is able to know his position so it is easy

for us to calculate in which direction agent has to walk in order to get in the specific coordinate. The figure 4.10 shows us that agent should travel from the point (X_{start}, Y_{start}), to the point (X_{target}, Y_{target}). It is easy to find $\vartheta^{\circ}_{target}$:
$\partial_X = X_{target} - X_{start}$
$\partial_Y = Y_{target} - Y_{start}$
$\vartheta^{\circ}_{target} = atan2(\partial_X, \partial_Y)$

With these calculations our agent is always aware of the distance and the direction he has to travel.

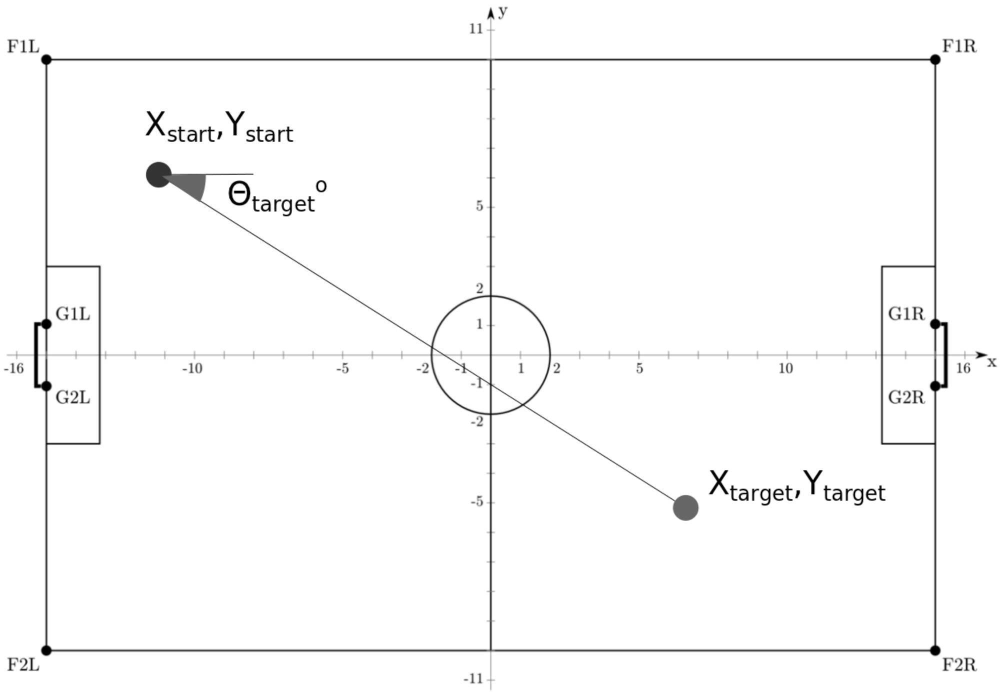

Figure 4.10: WalkToCoordinate.

WalkToDirection With this action agent walks towards a specific direction.

WalkWithBallToDirection As far as agent reaches the ball, he will try to keep the ball in front of him and walk towards a direction keeping into mind that the ball has to be always in front of him.

4.7.3 Vision

Vision related actions are created to control the vision perceptor which is attached to the robot's head as well as to collect data from this perceptor in order to execute related actions such as obstacle avoidance. These vision related actions are:

MoveHead This action is related with the movement of the head. Nao robot has two joints attached in the neck which give us the freedom of moving the head in relation to the action is being performed.

- **type 1** Head moves to its original position.
- **type 2** Head moves until agent see the ball.
- **type 3** Head moves in relation to the ball's movement.
- **type 4** Head make a harmonic movement in order agent to have a nice perception of his environment.
- **type 5** Head moves until agent can localize himself in the field.

WatchObjectMovement This action requires that object is in agent's field of view. Knowing the direction and the speed of the moving object is only feasible if we keep in memory a short number of observations. We keep two sets of five observations which we take within a time difference. Finding the average position of each set gives a distance between these two positions. If this distance will be divided with the time difference of the two observation set we are going to have the direction and the speed of the moving object.

FindOpponentsGoal This action is used in GoKickBallToGoal in order to take observations about the direction of the opponents goal in relation to agent's body angle.

ObstaclePerceptor Obstacle Perceptor is an action that has the responsibility of having a good view of all obstacles that there are in our close range. Due to the fact that simulated nao's head can move in horizontal axis from $120°$ to $-120°$ and our field of view is $120°$ means that we can have a complete imaging from all obstacles which are located close to our agent. So, in every cycle of Nao's head we save all obstacles in an array. It is usual to observe the same obstacle more than once, at this situation we find the average of these of observations. At the end of head's cycle we call the main action which tries to find alternative routes if there is an obstacle in our way.

ObstacleAvoidance In a dynamic and a multi-agent environment like simulation soccer this action is more than neccessary. However, there are some teams in simulated soccer competition which have not yet develop an obstacle avoidance system. In our framework there is a reliable and a well-tested system to avoid possible collisions.

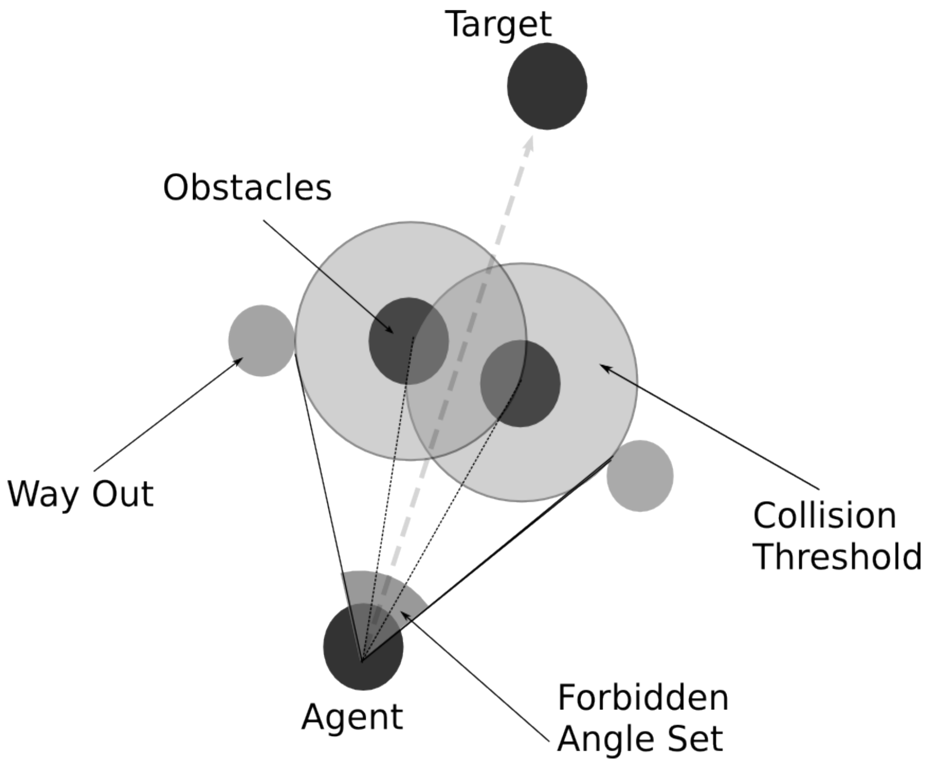

Figure 4.11: Obstacle Avoidance.

The figure 4.11 shows an example where there are two obstacles between the agent and his target. During his walking to his target agent scans the field for possible obstacles. If agent realizes that there is an object which blocks his way to the target in the same simulation cycle he starts calculate the possible way out angles that he could choose in relation to his observation about all obstacles. For every obstacle we calculate a set of two angles. These angles is determined by the distance between our agent and the obstacle and they show in which direction we can avoid this obstacle. When these angles are calculated, we check each angle of each set if it belongs to another angle set as well. Angles which belongs to another set are removed from the final list. This process is described as a pseudo-code 2. Once we have all the qualified angle sets from the algorithm, it is time to find

Algorithm 2 Way Out Angle Set

$Obstacles = \{O_1, O_2, ..., O_n\}$
for each i in Obstacles **do**
  $WayOutSet.Add(Calculate(O_a, t))$
end for
for each j in WayOutSet **do**
  for each t in $\{r, l\}$ **do**
    if $WayOutSet_{j,t} \in WayOutSet_k, \vee k \in \{1, 2 * n\}, k \neq j$ **then**
      $WayOutSet.Remove(j, t)$
    end if
  end for
end for

coordinates which are safe in order to avoid the obstacle. For each angle in these sets we calculate a specific coordinate. These coordinates in the soccer field will give us routes that are safe to follow. Now, we are going to calculate the cost for each route in respect with our body angle, the whole distance we have to travel to target if we follow this route. The route with the minimum cost is qualified to be followed by the agent.

4.7.4 Other Sensors

Other Sensors related actions are created to collect data from gyroscope, accelerometer and force resistance perceptors. In this category there is only one action. This action is called **CheckIfFall** and is responsible to check if our agent is fallen on the ground. In a multi-agent environment like this we should be aware about possible collisions with other agents or falls because the instability of movement. First of all, incoming perceptual inputs related to both gyroscope and accelerometer values are used to detect whether the robot has become subject of a big turmoil. Taking values above a threshold from these two perceptors, it is possible that the robot has fallen, but not completely sure to perform a stand up action yet. It is not unusual to receive values above threshold due to a collision without a fall. So, we have to check the force resistance perceptors which are located on the sole of agent's feet. If these perceptors imply that the legs do not touch the ground then we are pretty sure to perform a stand up action.Foot pressure value is also used to determine whether the stand up action is succeeded.

4.8 Communication

Communication in simspark is not ideal. There are not restrictions about the say effector and every player can use it in every cycle. However, the hear perceptor comes up with some restrictions. Messages should not have a length more than twenty characters from the ASCII subset [0x21; 0x7E] excluding [0x28; 0x29] which are the parenthesis characters, (and). Messages shouted from beyond a maximal distance (currently 50 meters) cannot be heard. Note that as the field is currently only 20x30 meters (36 diagonally), this does not turn out to be a limit in practice. The number of messages which can be heard at the same time is bounded. Finally, each player has the maximal capacity of one heard message by a specific team every two simulation cycles (thus every 0.04 seconds per team). 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 for one cycle (2ms) and repeats every three cycles (6ms). Figure 4.12 shows how time is sliced. Every three cycles there is one of these piece in which only one agent is able to send his message to the others. Every one of these slices has an integer label on it which states the uniform number of the player which is able to send his message. This label grows by one in every time a player send his 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 just 54ms.

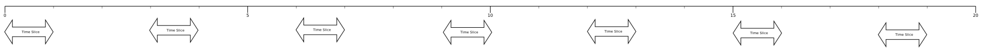

Figure 4.12: Time Slices.

Chapter 5

Coordination

5.1 Messages

5.2 Beliefs

5.3 Splitter

5.4 Active Positions

5.5 Active Coordination

5.6 Team Formation

5.7 Role Assignment

5.8 Support Positions

5.9 Support Coordination

5.10 Mapping Cost

Chapter 6

Results

Chapter 7

Related Work

Chapter 8

Future Work

8.1 Software's Conversion

This is a major issue that has to be resolved immediately. There are things to be changed 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 one that there are now eleven players for each side. 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.

8.2 Participation in Robocup

Robocup is a well-known competition in which everybody want to take part. 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.

8.3 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 hope this issue to be resolved in the near future.

8.4 Optimization and Debugging

Every project, no matter how well it is tested optimized during the development phase. I think that there are weak spots in the source code which have to be optimized.

Chapter 9

Conclusion

References