Technical University of Crete, Greece

Department of Electronic and Computer Engineering

Player Behavior and Team Strategy in SimSpark Soccer Simulation 3D

Georgios Methenitis

Chania, August 2012

Πολυτεχνειο Κρητης

Τμημα Ηλεκτρονικων Μηχανικων και Μηχανικων Υπολογιστων

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

Γεώργιος Μεθενίτης

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. In essence, each person on the team puts aside his or her individual needs to work towards the larger group objective. The interactions among the members and the work they complete is called teamwork. So, when we are talking about such a team sport like 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. This thesis describes, how we create a team's framework step by step, for the Robocup's soccer simulation league. This league includes all the classic Artificial Intelligence's and robot problems such as, perception, localization, movement and coordination. 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 very important issue in robotic soccer games. Nothing could be accomplished by a soccer team if players have a poor movement in the field. There must be stable and fast movements by the players, this can be prove to be crucial in a soccer game. Last but not least, is the coordination, which is a major's importance factor 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. In this thesis, we are coming up with an approach in each one of these problems emphasizing in agents' coordination. The whole idea is based on agents' ability to communicate with each other. Using a simple communication protocol goalkeeper gathers information from all field players and then it his responsibility to make the coordination and send them in the end of the process the actions which have been calculated to be costless and worthy for the team.

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 εκάστοτε ομάδα. Οι ενέργειες αυτές γίνονται από την πλευρά κάθε παίκτη βάζοντας στην άκρη τις προσωπικές φιλοδοξίες 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. Αυτή η διπλωματική εργασία περιγράφει την δημιουργία βήμα-βήμα ολοκλήρου του πλαισίου για την κάλυψη των αναγκών μιας ρομποτικής ομάδας ποδόσφαιρου για το επίσημο πρωτάθλημα προσομοίωσης Robocup. Αυτό το πρωτάθλημα 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 ερχόμαστε με μια προσέγγιση που δίνει απαντήσεις στα παραπάνω ερωτήματα και δίνει έμφαση στον τομέα της συνεργασίας. Η όλη ιδέα βασίζεται στην ικανότητα των παικτών να επικοινωνούν μεταξύ τους. Με ένα απλό και εύχρηστο πρωτόκολλο επικοινωνίας ο τερματοφύλακας μαζεύει πληροφορία από τους υπόλοιπους παίκτες της ομάδας και μετά είναι δική του ευθηνή ο καταμερισμός ενεργειών που θα προσδώσουν στην ομάδα το μέγιστο κέρδος με το μικρότερο δυνατό κόστος.

Contents

List of Figures

Chapter 1

Introduction

What will happen if we place a team of robots into a soccer field? It is obvious for everyone to realise 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 a 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 playing soccer immediately. There are plenty of things that have to be done until these robots start playing in the way human players do. Starting from the beginning there must be a connection with the simulation server; Server sends to each connected agent messages every 20ms, these messages include information about agent's sight and other perceptions. Each agent use these messages to update his perceptions. At the end of the scanning of each message 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 his sight's field. Having this knowledge we can proceed to more complicated things. First of all, agents have to calculate their position in the soccer field, it is not so simple as it sounds and it requires at least two landmarks in our sight's field. We are going to explain this operation extensively later. Even if, these agents know their positions in the soccer field and can calculate the position of every other agent in their sight as well as the soccer ball position, they are still not able to perform a single action. This will be feasible if they combinate motions which are going to help them perform each action. Even in real life, a 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. On the other hand complex actions make use of more than one simple actions to be performed 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 other more simple actions. Until now, we have accomplished every agent in the field to be able to recognize objects, find its position in the soccer field 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 has a complete sense of its environment and is able to perform actions. This is 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 , in our time and age ,for teams participating in games, where success can only be achieved through collaborative and coordinated efforts. This is going to be accomplished through communication. This thesis describes a way of making the coordination of the agents through communication as well as a proposed solution of all the problems generated in robotic soccer. The main objective of the this thesis is to develop an efficient software system to correctly model the behaviors of simulated Nao robots in such a competitive environment as simulation soccer league. The challenging and the most time consuming part of this project was the coordination part which I firmly believe that is the most important part either in a simulated team or in a real soccer team.

1.1 Thesis Outline

Chapter 2

Background

2.1 RoboCup Competition

In the history of artificial intelligence and robotics, the year 1997 will be remembered as a turning point. In May 1997, IBM Deep Blue defeated the human world champion in chess. Forty years of challenge in the AI community came to a successful conclusion. On July 4, 1997, NASA'As pathfinder mission made a successful landing and the first autonomous robotics system, Sojourner, was deployed on the surface of Mars. Together with these accomplishments, 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 feasible 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.

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. Technical University of Crete is proud of the continuous participation of the team Kouretes in this league.

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. It is colocated with the RoboCup symposium.

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 at 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

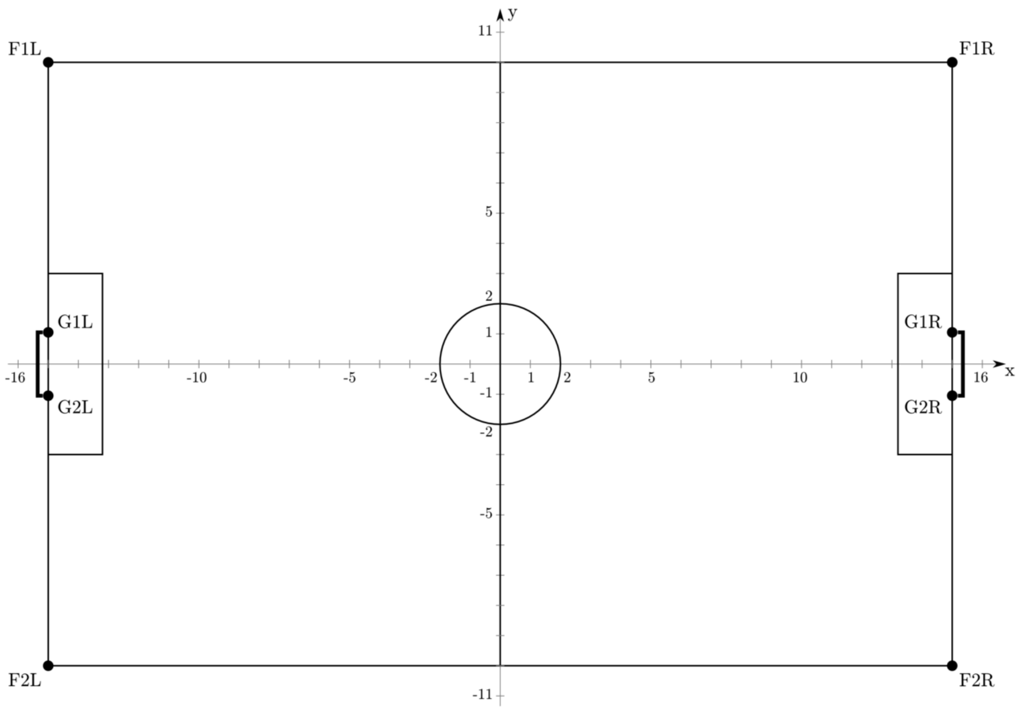

Figure 3.1: Simulation Soccer Field

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 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 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:

```
(syn)
```

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.

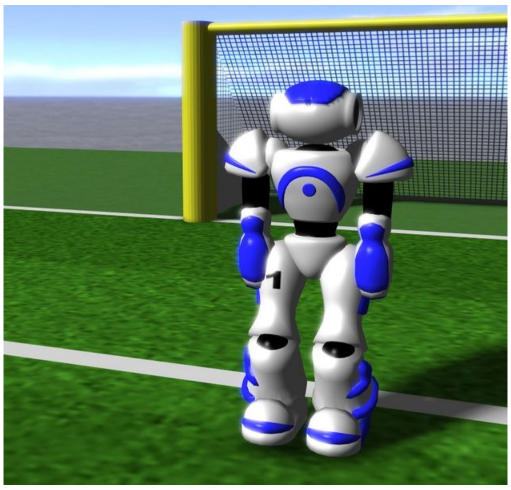

Figure 3.2: Nao in simulation screen

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 free to do what the behavior tells him. In our approach, only goalkeeper is running an independent behavior, the other eight field players start a communication procedure in order to inform the goalkeeper about the worldstate and their attributes. So, we can realise 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 responsibe for handling the agent's requests for a message that he has to say or a movement that he has to execute. These two controller are sending in every cycle effection messages to the connection layer which will send them back to soccer simulation server.

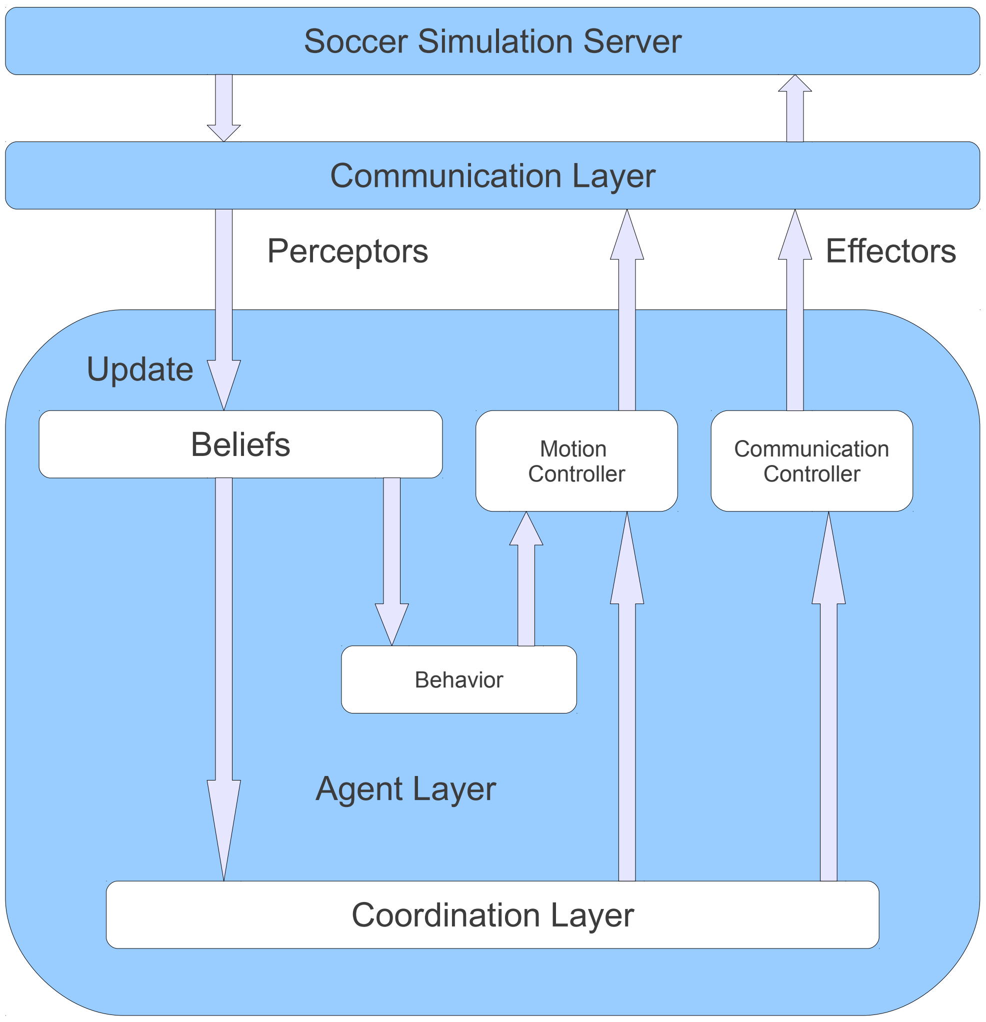

Figure 4.1: Agent's Architecture.

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 beggining 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.

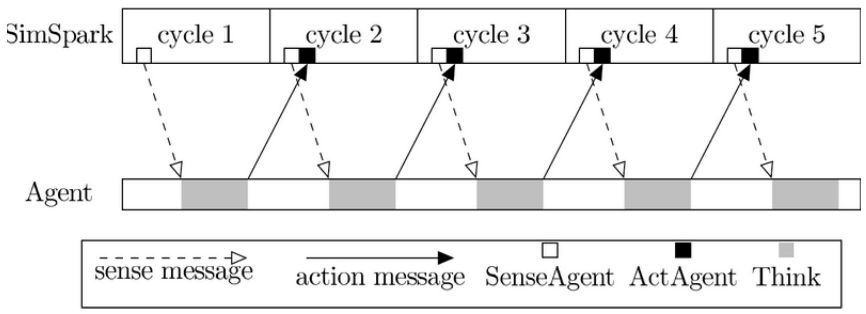

Figure 4.2: Simulation Update Loop.

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.

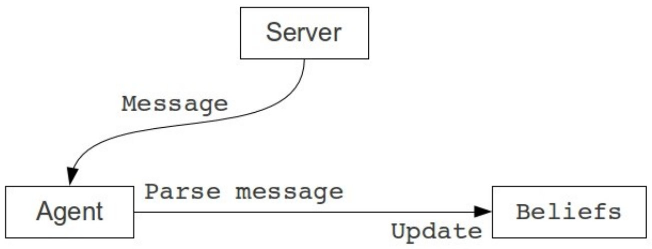

Figure 4.3: Beliefs Update.

4.4 Localization

Once we have all the neccessary 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 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.

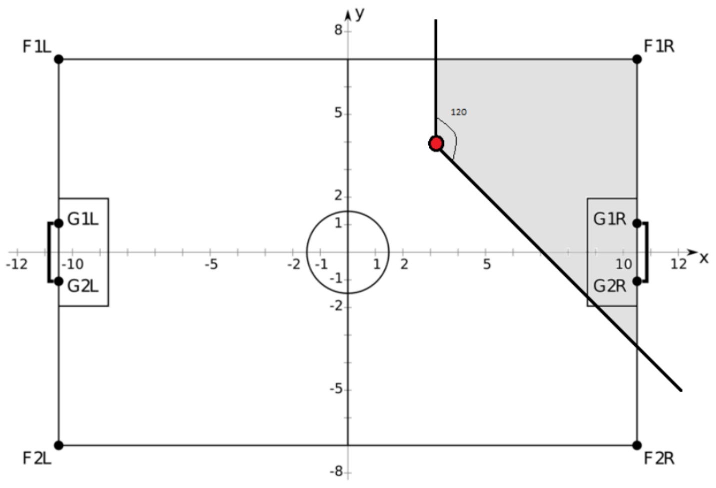

Figure 4.4: Nao's field of view.

If our agent sees more than two landmarks, then this function is called 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, 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.

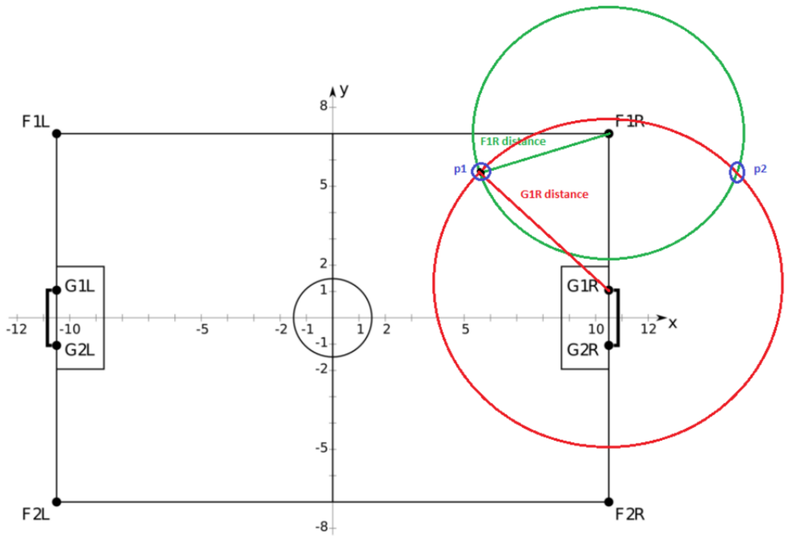

Figure 4.5: Localization.

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

In the following 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, 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 above 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

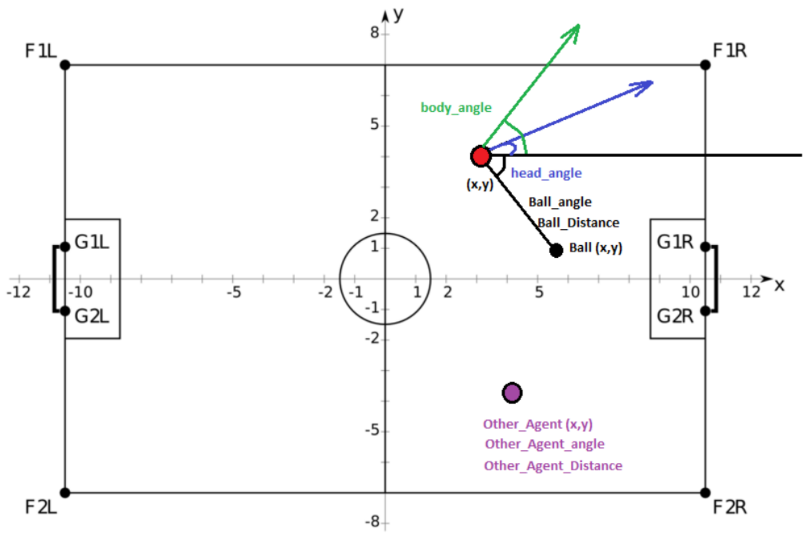

Figure 4.6: Localization Results.

agent's position in the field. When an observation is coming, we check if the queue is empty or full; If it is emtpy then we just add the observation into the 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.

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:

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

```
1: if $x, y \neq NaN$ then
2:   if $size(Queue) = 0$ then
3:     $Queue.Add(Observation)$
4:   else if $size(Queue) < Max$ then
5:     if $Observation \not\approx AVG(Queue)$ then
6:       $Queue.Remove()$
7:     else
8:       $Queue.Add(Observation)$
9:       $MyPosition = AVG(Queue)$
10:    end if
11:  else
12:    if $Observation \not\approx AVG(Queue)$ then
13:      $Queue.Remove()$
14:    else
15:      $Queue.Remove()$
16:      $Queue.Add(Observation)$
17:      $MyPosition = AVG(Queue)$
18:    end if
19:  end if
20: end if
```

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

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 dunamic 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 splitted 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

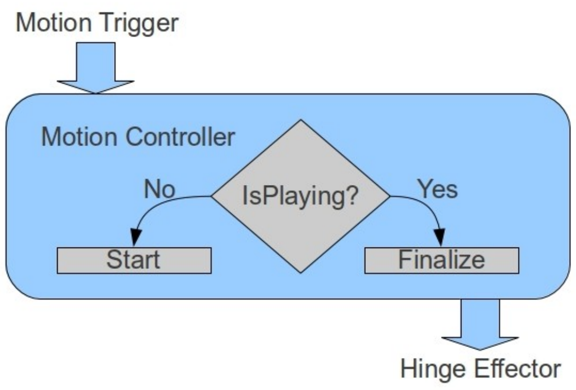

Figure 4.7: Motion Controller.

of the motion controller. Motion cotroller checks if there is a motion which is playing already. If yes, motion controller tries to finalize the playing movement

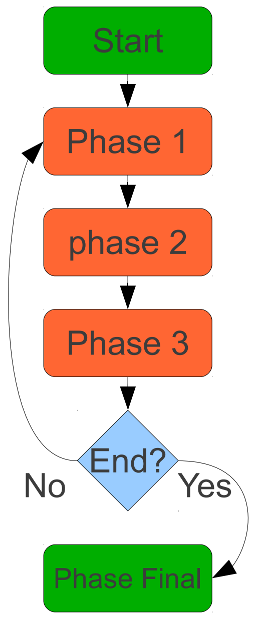

Figure 4.8: Phase Sequence.

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 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:

$$DesiredVelocity = AlreadyJointValue \text{ - } DesiredJointValue$$

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 joint value in this time limit.

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

This velocity is calculated for every joint invloved 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 simplier 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:

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

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

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

4.7 Actions

advantage

4.7.1 Simple

advantage

4.7.2 Complex

advantage

4.7.3 Vision

advantage

4.8 Communication

advantage

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

In this section we give a short overview of what other RoboCup teams have published to our knowledge about their communication system and compare their work to ours. There are various approaches for sharing information among processes on a robot and among processes on different robots. These approaches can be summarized into the following main ideas:

7.1 Shared Memory

The main idea here is to define a shared memory area on each node, which holds information about the current world model. Each module can write to and/or read data from this shared location. Under this scheme, different nodes use a simple UDP protocol to exchange important data. This is quite similar to our approach, however communication is mainly point-to-point and lacks the organization and efficiency imposed by the hierarchical topic-based publish/subscribe scheme. Furthermore, depending on the implementation, problems with corrupted data could be appear, if no lock and safety mechanisms are implemented. This idea with variations has been adopted by teams as Austin Villa [1],BreDoBrothers [2]

7.1.1 Blackboard/Shared World Model

This is a similar idea with the previous, however different components access the central storage data via well defined interfaces, providing with the mechanisms that might be absent from the approach above. Team robots share the information of their blackboard creating a world model composed of the individual robots' perceptions. There is one variation from GTCMU [?] where validity flags are added to the objects stored in order to declare "use with caution" when the objects are not updated for a long period of time. For intra-robot or debugging purposes custom communication systems are used over UDP or TCP. The purpose of data exchange suggests the used protocol, for example for debugging purposes TCP is used, while when in game UDP transmission is preferred. The shared model is close to Narukom's implementation of distributed blackboard, but the exchange of newly created data requires from the programmer writing code for sending and receiving those data, whereas on Narukom newly declared Messages are immediately ready for network transfer. The vast majority of robocup teams use a slight variation of this approach such as Zadeat [3], rUNSWift [?].

7.1.2 Message queue and Streams

This idea is advocated recently by team B-Human [4] and earlier by the German Team [5]. It is based on using message queues for constructing three different types of cross-platform communication streams: (a) inter-process communication between the cognition and the motion module, (b) debug communication between robots and remote computers for debugging purposes, and (c) information exchange between the robots. The first type is similar to Narukom's inter-process communication. The other two types are implemented differently in Narukom, but they serve the same purposes. Again, this idea lacks the organization and efficiency of Narukom. It must be noted , however that a lot of effort has been put into creating a full featured Stream library to support the required functionality, which is not present in Narukom.

7.1.3 Publisher/Subscribe

This idea is based on the publish/subscribe paradigm and has been adopted only the Austrian Kangaroos [6] team. The idea is to use a publish/subscribe architecture for exchanging data between the threads on the same machine by triggering signals in all connected modules if a subscribed memory location is altered. This design allows an IRQ-like implementation of service calls at a high system level. Network communication is done via SOAP wrapped functions calls, which generally are considered heavy compared to Narukom's approach. On the other hand, in this approach there each module a more direct access to remote data by subscribing to remote memory locations.

7.2 Overview

Generally speaking, Narukom integrates features that support any communication need in the SPL; these features are not found collectively in any other existing communication system. One may argue that Narukom's functionality could be implemented using the infrastructure provided by the proprietary NaoQi middleware offered by Aldebaran Robotics with the Nao robots. This is true, however Narukom's purpose is to provide a communication system that does not depend on proprietary technology and can be used widely on many different robotic teams, apart from Nao robots, and other real-time distributed systems. Narukom also represents our efforts in departing from the dependence on the NaoQi framework towards platform-independent customizable software architecture for robotic teams.

Chapter 8

Future Work

This is just a small , but well founded step towards our notion of the ideal communication framework for robotic teams and other similar distributed systems. Thus, there are plans for extending, adding and optimizing further the proposed framework.

8.1 Alternative Channels

8.2 Narukom outside robocup

8.3 Optimization and Debugging

Every project, no matter how well it is tested optimized during the development phase after it is released bugs, bottlenecks are found. Our intention and desire is to continue to actively develop Narukom in order to meet the needs of as many people as possible. As a result a constant procedure of optimizing , debgugin and testing is needed to ensure that Narukom is a robust, easy to use framework.

Chapter 9

Conclusion

Narukom is an attempt to address a lot of communication problems our team had during the last years in Robocup (SPL), while at the same time is a first step towards a cross-platform architecture used in robotic agents. Our journey does not end here our Narukom has, hopefully, a long way, yet to go. The first real world test is the Robocup 2010 in Singapore where both our inter-process and team communication would be based on our framework.

9.1 Out To The Real World

Narukom is based on open source projects, so the least we could do is to open source it in the near future. We believe that Narukom can contribute to open source community as a cross-platform, distributed, transparent communication framework for everyone willing to use it. I, personally, would be more than happy to assist anyone dare to use our framework. Additionally, any suggestion , comment, critiscism, imporvement and modification is at least welcomed. By open sourcing Narukom we try to achieve two goals take Narukom to the first step and encourage new coders to make available to the community of open source their code in order to advance the collaboration between programmers and avoid reinventing the wheel every now and then. Finally, I would like to thank those who spent some time using Narukom.

References

[1] Hester, T., Quinlan, M., Stone, P.: UT Austin Villa 2008: Standing on two legs (2008) Only available online: www.cs.utexas.edu/~AustinVilla. 37

[2] Czarnetzki, S., Hauschildt, D., Kerner, S., Urbann, O.: BreDoBrothers team report for RoboCup 2008 (2008) Only available online: www.bredobrothers.de. 37

[3] Ferrein, A., Steinbauer, G., McPhillips, G., Niemüller, T., Potgieter, A.: Team Zadeat - team report (2009) Only available online: www.zadeat.org. 38

[4] Röfer, T., Laue, T., Müller, J., Bösche, O., Burchardt, A., Damrose, E., Gillmann, K., Graf, C., de Haas, T.J., Härtl, A., Rieskamp, A., Schreck, A., Sieverdingbeck, I., Worch, J.H.: B-Human team report and code release 2009 (2009) Only available online: www.b-human.de/download.php?file=coderelease09_doc. 38

[5] Rofer, T., Laue, T., Weber, M., Burkhard, H.D., Jungel, M., Gohring, D., Hoffmann, J., Krause, T., Spranger, M., von Stryk, O., Brunn, R., Dassler, M., Kunz, M., Oberlies, T., Risler, M., Schwiegelshohn, U., Hebbel, M., Nistico, W., Czarnetzki, S., Kerkhof, T., Meyer, M., Rohde, C., Schmitz, B., Wachter, M., Wegner, T., Zarges, C.: GermanTeam 2005 (2005) Only available online: www.germanteam.org/GT2005.pdf. 38

[6] Bader, M., Hofmann, A., Schreiner, D.: Austrian Kangaroos team description paper (2009) Only available online: www.austrian-kangaroos.com. 39