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

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

Georgios Methenitis

Thesis Committee
Assistant Professor Michail G. Lagoudakis (ECE)
Assistant Professor Georgios Chalkiadakis (ECE)
Professor Minos Garofalakis (ECE)

Chania, August 2012

ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΝΙΚΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

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

Γεώργιος Μεθενίτης

Εξεταστική Επιτροπή
Επίκουρος Καθηγητής Μιχαήλ Γ. Λαγουδάκης (ΗΜΜΥ)
Επίκουρος Καθηγητής Γεώργιος Χαλκιαδάκης (ΗΜΜΥ)
Καθηγητής Μίνως Γαροφαλάκης (ΗΜΜΥ)

Χανιά, Αύγουστος 2012

Abstract

Any team participating in a team sport requires both individual and team skills in order to be successful. For human teams, these skills are inherent and naturally improve over time. However, for robot teams these skills must be programmed by the designers of the team. Robotic soccer, known as RoboCup, represents a complex, stochastic, real-time, multi-agent, competitive domain. In such domains, team skills are as important as individual player skills, considering that in soccer simulation leagues there are up to 9 or 11 players per team. This thesis presents a complete team design for the RoboCup 3D Simulation League focusing on player behavior, team strategy, and team coordination. Our agents are designed in a way that enables them to act effectively both autonomously and as members of the team. Initially, the development of the individual player skills is described. These skills include robust self localization and object tracking, effective locomotion and soccer motions, basic and complex action execution, and communication with teammates. Subsequently, a hierarchical coordination protocol is described, which coordinates all the individual player skills yielding a complete behavior for each agent within the frame of a global team strategy. Our approach is based on first sharing and fusing information about the game state and then decomposing the global coordination problem for the 9 or 11 players to smaller coordination problems over dynamically-determined subsets of players adhering to an adaptive global team formation. An exhaustive algorithm is used over the most important subset of active players (the three ones closest to the ball) to derive an optimal set of actions, whereas a less-expensive dynamic programming algorithm is used over the remaining players (support players) to derive their actions. Coordinated actions are evaluated through a function that combines costs related to positions, distances, potential collisions, and field coverage. Our approach and our Java implementation enable the team to compute coordinated actions approximately every two seconds yielding quick responsiveness to dynamically changing game states. The results of complete games against existing teams, some of which compete for several years in the RoboCup 3D Simulation League, reveal that our team is quite competitive mostly thanks to the proposed coordination approach.

Περίληψη

Κάθε ομάδα που συμμετέχει σ΄ ένα ομαδικό άθλημα απαιτεί δεξιότητες τόσο σε ατομικό όσο και σε ομαδικό επίπεδο, προκειμένου να είναι επιτυχημένη. Για ανθρώπινες ομάδες τέτοιες δεξιότητες είναι έμφυτες και φυσιολογικά βελτιώνονται με την πάροδο του χρόνου. Ωστόσο, για ρομποτικές ομάδες αυτές οι ικανότητες θα πρέπει να προγραμματισθούν από τους σχεδιαστές τους. Το ρομποτικό ποδόσφαιρο, γνωστό ως RoboCup, αποτελεί ένα σύνθετο, στοχαστικό, πολυ-πρακτορικό και ανταγωνιστικό περιβάλλον πραγματικού χρόνου. Σ΄ ένα τέτοιο περιβάλλον, οι ομαδικές ικανότητες είναι εξίσου σημαντικές με τις ατομικές ικανότητες των παικτών, αναλογιζόμενοι ότι στα πρωταθλήματα προσομοίωσης ποδοσφαίρου υπάρχουν μέχρι και 9 ή 11 παίκτες ανά ομάδα. Η παρούσα διπλωματική εργασία παρουσιάζει τον πλήρη σχεδιασμό μιας ομάδας για το πρωτάθλημα RoboCup 3D Simulation, δίνοντας έμφαση στη συμπεριφορά των παικτών, στην στρατηγική της ομάδας και στον συντονισμό της. Οι πράκτορές μας σχεδιάστηκαν με τρόπο που τους επιτρέπει να δρουν αποτελεσματικά τόσο αυτόνομα όσο και ως μέλη της ομάδας. Αρχικά, περιγράφεται η ανάπτυξη των ατομικών δεξιοτήτων των παικτών. Αυτές οι δεξιότητες περιλαμβάνουν τον αξιόπιστο εντοπισμό της θέσης τους και την αξιόπιστη παρακολούθηση αντικειμένων, την μετακίνηση και τις ποδοσφαιρικές κινήσεις, την εκτέλεση βασικών και πολύπλοκων ενεργειών, καθώς και την επικοινωνία με τους συμπαίκτες. Στη συνέχεια, περιγράφεται ένα ιεραρχικό πρωτόκολλο συντονισμού, το οποίο συντονίζει όλες τις επιμέρους ικανότητες των παικτών παρέχοντας μια πλήρη συμπεριφορά για κάθε πράκτορα στα πλαίσια μιας συνολικής στρατηγικής για την ομάδα. Η προσέγγισή μας βασίζεται αρχικά στην ανταλλαγή και σύνθεση πληροφοριών για την κατάσταση του παιχνιδιού και κατόπιν στην αποσύνθεση του προβλήματος του συνολικού συντονισμού για τους 9 ή τους 11 παίκτες σε μικρότερα προβλήματα συντονισμού για υποσύνολα παικτών που καθορίζονται δυναμικά βάσει ενός προσαρμοζόμενου σχηματισμού της ομάδας. Ένας εξαντλητικός αλγόριθμος χρησιμοποιείται για να εξαχθεί ένα βέλτιστο σύνολο ενεργειών για το σημαντικότερο υποσύνολο ενεργών παικτών (οι τρεις πλησιέστεροι στην μπάλα), ενώ ένας λιγότερο δαπανηρός αλγόριθμος δυναμικού προγραμματισμού χρησιμοποιείται για να αντλήσουν τις ενέργειές τους οι υπόλοιποι παίκτες (υποστηρικτικοί παίκτες). Οι συντονισμένες ενέργειες αξιολογούνται μέσω μιας συνάρτησης που συνδυάζει κόστη σχετικά με τις θέσεις, τις αποστάσεις, τις πιθανές συγκρούσεις και την κάλυψη του γηπέδου. Η προσέγγισή μας και η υλοποίησή μας σε Java επιτρέπουν στην ομάδα μας να υπολογίζει συντονισμένες δράσεις περίπου κάθε δύο δευτερόλεπτα αποδίδοντας γρήγορη ανταπόκριση στις δυναμικά μεταβαλλόμενες συνθήκες του παιχνιδιού. Τα αποτελέσματα που προέκυψαν από πλήρεις αγώνες απέναντι σε υπάρχουσες ομάδες, μερικές από τις οποίες αγωνίζονται για αρκετά χρόνια στο πρωτάθλημα RoboCup 3D Simulation, αποκαλύπτουν

ότι η ομάδα μας είναι ιδιαίτερα ανταγωνιστική ως επί το πλείστον χάρη στο προτεινόμενο πρωτόκολλο συντονισμού.

Acknowledgements

First of all, I would like to thank my parents and my friends for all their support and their encouragement.

Of course, I would like to thank my advisor Michail G. Lagoudakis for his help and the trust that he showed to me from the first moment.

Contents

1 Introduction **1**

1.1 Thesis Contribution . . . 1

1.2 Thesis Outline . . . 2

2 The RoboCup Competition **3**

2.1 RoboCup Soccer . . . 4

2.2 RoboCup Rescue . . . 7

2.3 RoboCup@Home . . . 10

2.4 RoboCup Junior . . . 10

3 RoboCup 3D Simulation League **13**

3.1 SimSpark Soccer Simulator . . . 13

3.2 Robot Model . . . 14

3.3 Server . . . 15

3.4 Monitor . . . 16

3.4.1 SimSpark Monitor . . . 16

3.4.2 Roboviz Monitor . . . 17

3.5 Perceptors . . . 17

3.5.1 General perceptors . . . 18

3.5.2 Soccer perceptors . . . 19

3.6 Effectors . . . 21

3.6.1 General Effectors . . . 21

3.6.2 Soccer Effectors . . . 22

4 Player Skills **25**

4.1 Agent Architecture . . . 25

4.2 Connection . . . 26
4.3 Perceptions . . . 27
4.4 Localization . . . 28
    4.4.1 Self Localization . . . 28
    4.4.2 Object Localization . . . 29
    4.4.3 Localization Filtering . . . 30
4.5 Motion . . . 33
    4.5.1 XML-Based Motion Files . . . 34
    4.5.2 XML-Based Motion Controller . . . 35
    4.5.3 Text-Based Motion Files . . . 37
    4.5.4 Text-Based Motion Controller . . . 38
    4.5.5 Dynamic Motion Elements . . . 39
4.6 Actions . . . 40
    4.6.1 Basic Actions . . . 40
    4.6.2 Complex Actions . . . 42
4.7 Communication . . . 48

5 Team Coordination 51
5.1 Coordination Protocol . . . 51
5.2 Coordination Modes . . . 54
5.3 Coordination and Communication . . . 55
5.4 Coordination Beliefs Update . . . 58
5.5 Determination of Coordination Subsets . . . 60
5.6 Soccer Field Utility Fuction . . . 61
5.7 Determination of Active Positions . . . 62
5.8 Active Players Coordination . . . 64
5.9 Team Formation Generation . . . 67
5.10 Team Roles Assignment . . . 72
5.11 Determination of Support Positions . . . 73
5.12 Support Players Coordination . . . 74
5.13 Goalkeeper Behavior . . . 77

6 Results 81
6.1 Motion . . . 81
6.2 Communication . . . 82

6.3 Goalkeeper 83
6.4 Coordination 83
6.5 Games 90

7 Related Work **93**
7.1 UT Austin Villa 93
7.2 BeeStanbul 94
7.3 FUT-K_3D 95
7.4 Farzanegan 95

8 Conclusion **97**
8.1 Future Work 97

References **100**

List of Figures

2.1 Humanoid Kid, Teen, Adult -Size League at Robocup 2011. 4
2.2 Middle-Size League at RoboCup 2011. 5
2.3 Simulation League 2D (left) and 3D (right). 6
2.4 Small-Size League at RoboCup 2011. 6
2.5 Standard Platform League at RoboCup 2011. 7
2.6 Rescue Robot League at RoboCup 2011. 8
2.7 Rescue Simulation League at RoboCup 2006. 9
2.8 RoboCup@Home at RoboCup 2011. 10
2.9 Junior Soccer League at RoboCup 2011. 11

3.1 RoboCup 3D Simulation League Field 14
3.2 Rcssserver3d Simulated (left) and Aldebaran Real (right) Nao Robot Models. 15
3.3 Roboviz (left) vs SimSpark (right) Monitors. 17

4.1 The Agent Architecture. 26
4.2 Server and Agent Communication. 27
4.3 Nao's Restricted Field of View and Field Landmarks. 29
4.4 Self-Localization Example with the F1R and G1R Landmarks. 30
4.5 Object (Ball and Opponent) Localization Example. 31
4.6 Nao's Anatomy: Kinematic Chains and Joints. 33
4.7 XML-Based Motion Controller. 36
4.8 Phase Execution Sequence for a Typical XML-Based Motion. 37
4.9 Dynamic Walk Leaning: Left Leaning (left), Right Leaning (right). . . . 39
4.10 Dynamic Turn Gain: Turn Degrees (y-axis) against Gain Factor (x-axis). 40
4.11 Nao Performing a Kick after Positioning for Kick. 43
4.12 Obstacle Avoidance. 45

4.13 Ground Distance between the Agent and the Ball. 46
4.14 On Ball Action Logic Flowchart. 47
4.15 Walk To Coordinate Action. 48
4.16 Time Slicing Communication Protocol. 49

5.1 The Coordination Protocol. 52
5.2 Communication Process in Coordination. 57
5.3 Global Ball Position Estimation from Multiple Ball Observations. 59
5.4 Coordination Splitter. 62
5.5 Soccer Field Value. 63
5.6 Initial Candidate Active Positions: Defense (left) and Offense (right). . . 64
5.7 Final Candidate Active Positions: Defense (left) and Offense (right). . . . 64
5.8 Collision Detection Feature in the Evaluation Function. 67
5.9 Template of Role Positions in the Team Formation for the 9-Players Version. 69
5.10 Template of Role Positions in the Team Formation for the 11-Players Version. 70
5.11 Role Positions in Team Formation for Various Ball Positions (in red). . . 71
5.12 Role Assignment Function for a Given Team Formation. 73
5.13 Determination of Support Positions from a Given Team Formation. . . . 74
5.14 Finite State Machine for the Goalkeeper Behavior. 78
5.15 Goalkeeper Behavior in the **Guard** State. 78
5.16 An Example of a Goalkeeper Fall to Prevent Opponents from Scoring. . . 79

6.1 Estimated Position of the Ball in Four Examples. 84
6.2 Offensive Positioning Resulting by Coordination Protocol, Example 1. . . 86
6.3 Defensive Positioning Resulting by Coordination Protocol, Example 1. . . 87
6.4 Defensive Positioning Resulting by Coordination Protocol, Example 2. . . 88
6.5 Formation Consistency Resulting by Coordination Process Via Team Roles Assignment. 89

List of Tables

5.1 Team Formation Description for the 9-Players Version 69
5.2 Team Formation Description for the 11-Players Version 70
5.3 Mappings Evaluated by the Support Players Mapping Algorithm. 77

6.1 Motion's Performance Improvement (Averaged Speeds and Ranges) . . . 82
6.2 Communication Results in Ideal and Match Conditions 82
6.3 Goalkeeper Averaged Results in Half-Games. 83
6.4 All Played Games Results 91
6.5 Mini Tournament 92

List of Algorithms

1 Localization Filtering 32
2 Escape Angle Set Calculation 44
3 Coordination Protocol 54
4 Active Players Optimal Mapping 65
5 Support Players Mapping 76

LIST OF ALGORITHMS

Chapter 1

Introduction

Any team participating in a team sport requires both individual and team skills in order to be successful. For human teams, these skills are inherent and naturally improve over time. However, for robot teams these skills must be programmed by the designers of the team. Robotic soccer, known as RoboCup, represents a complex, stochastic, real-time, multi-agent, competitive domain. In such domains, team skills are as important as individual player skills, considering that in soccer simulation leagues there are up to 9 or 11 players per team.

1.1 Thesis Contribution

This thesis presents a complete team design for the RoboCup 3D Simulation League focusing on player behavior, team strategy, and team coordination. Our agents are designed in a way that enables them to act effectively both autonomously and as members of the team. Initially, the development of the individual player skills is described. These skills include robust self localization and object tracking, effective locomotion and soccer motions, basic and complex action execution, and communication with teammates. Subsequently, a hierarchical coordination protocol is described, which coordinates all the individual player skills yielding a complete behavior for each agent within the frame of a global team strategy. Our approach is based on first sharing and fusing information about the game state and then decomposing the global coordination problem for the 9 or 11 players to smaller coordination problems over dynamically-determined subsets of players adhering to an adaptive global team formation. An exhaustive algorithm is used over the most important subset of active players (the three ones closest to the ball) to derive an optimal set of actions, whereas a less-expensive dynamic programming algorithm is used over the remaining players (support players) to derive their actions. Coordinated actions are evaluated through a function that combines costs related to positions, distances, potential collisions, and field coverage. Our approach and our Java implementation enable the team to compute coordinated actions approximately every two seconds yielding quick responsiveness to dynamically changing game states. The results of complete games against existing teams, some of which compete for several years in the RoboCup 3D Simulation League, reveal that our team is quite competitive mostly thanks to the proposed coordination approach.

1.2 Thesis Outline

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

Chapter 2

The RoboCup Competition

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

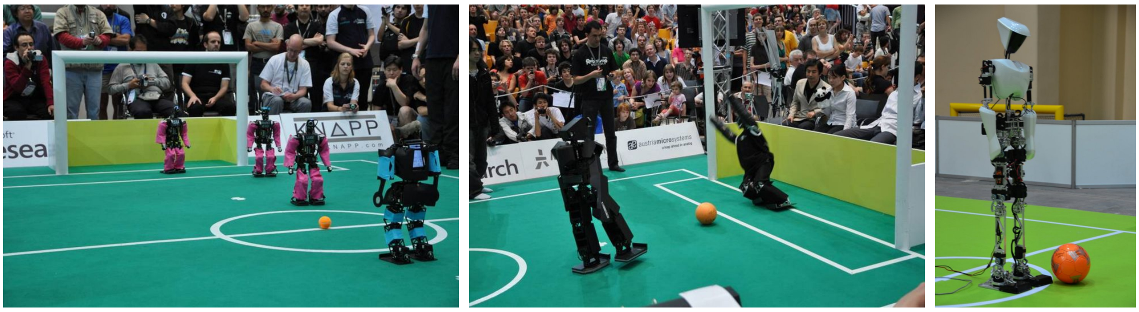

Figure 2.1: Humanoid Kid, Teen, Adult -Size League at Robocup 2011.

2.1 RoboCup Soccer

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

Humanoid League

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

Middle-Size League

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

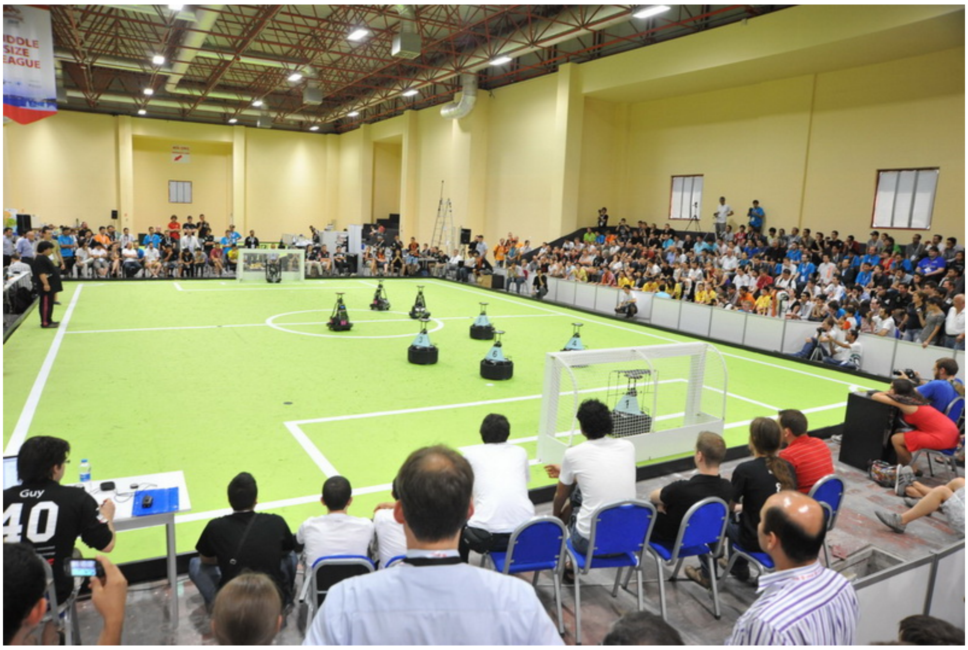

Figure 2.2: Middle-Size League at RoboCup 2011.

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

Simulation League

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

Small-Size League

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

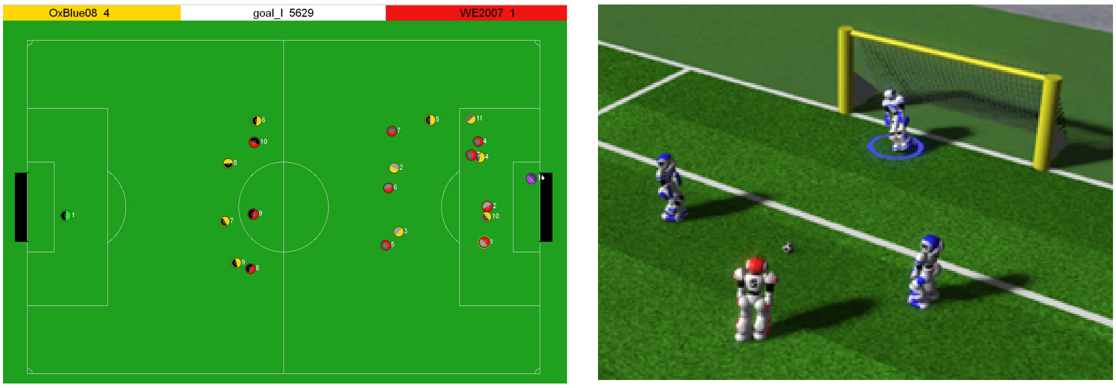

Figure 2.3: Simulation League 2D (left) and 3D (right).

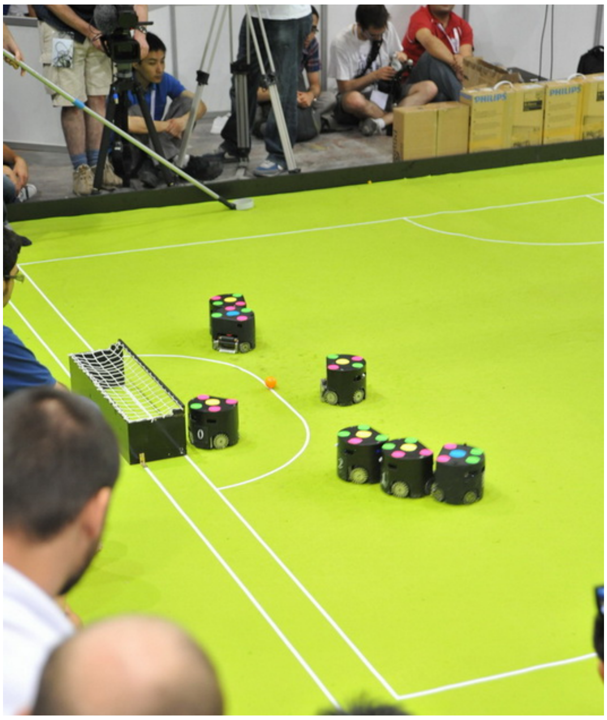

Figure 2.4: Small-Size League at RoboCup 2011.

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

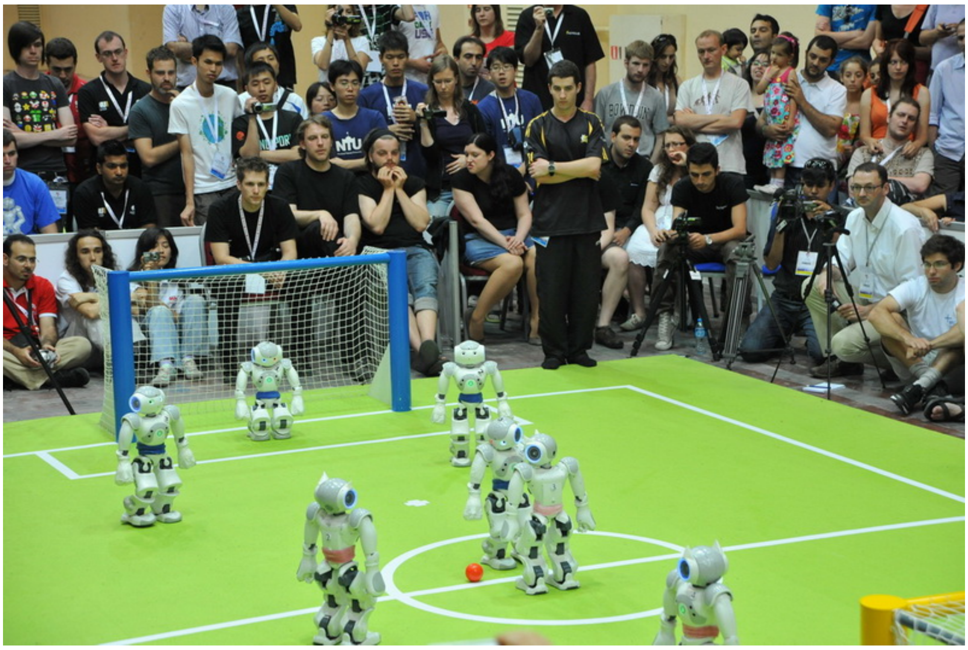

Figure 2.5: Standard Platform League at RoboCup 2011.

Standard Platform League

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

2.2 RoboCup Rescue

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

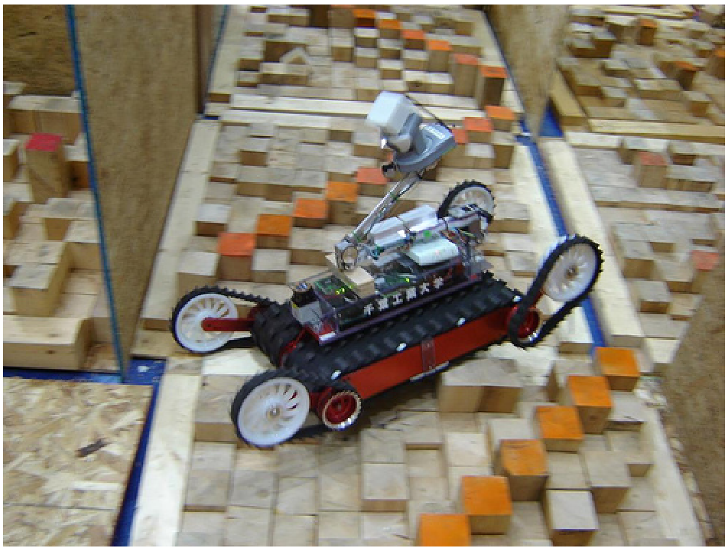

Figure 2.6: Rescue Robot League at RoboCup 2011.

comprehensive systems in future.

Robot League

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

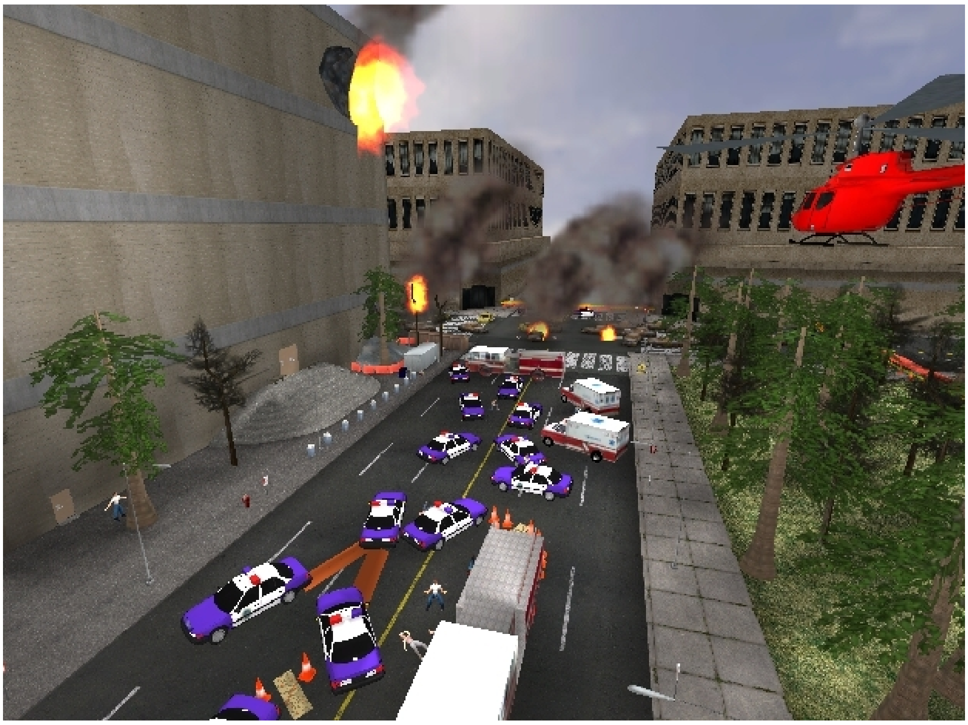

Figure 2.7: Rescue Simulation League at RoboCup 2006.

Simulation League

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

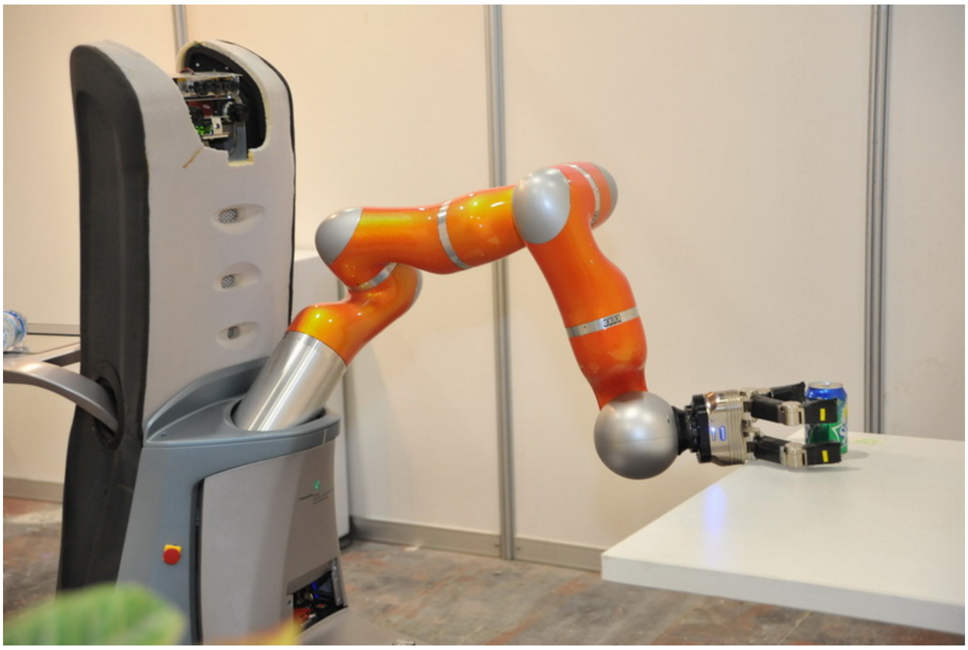

Figure 2.8: RoboCup@Home at RoboCup 2011.

2.3 RoboCup@Home

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

2.4 RoboCup Junior

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

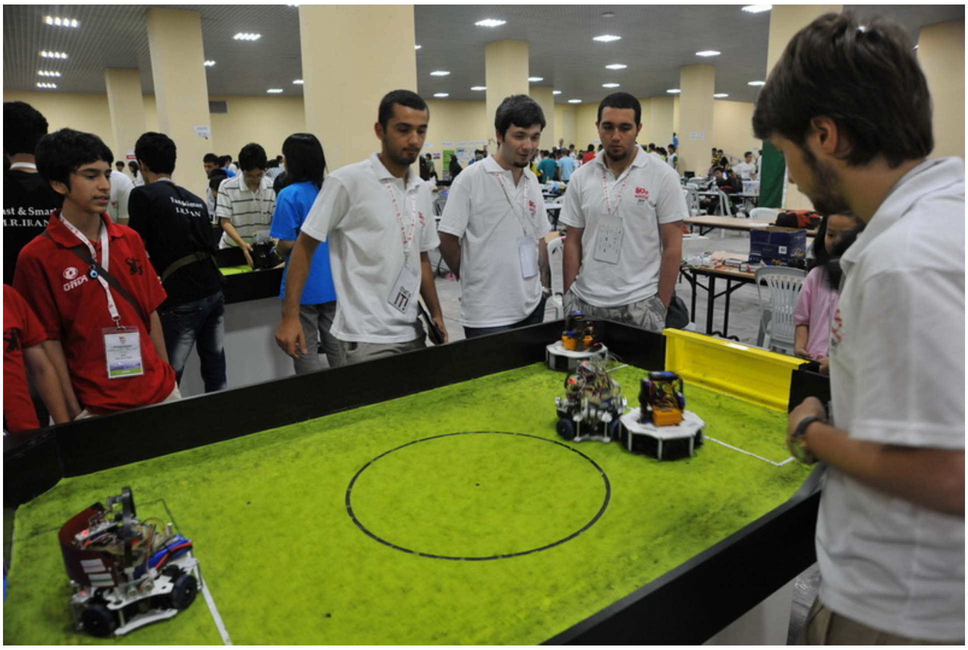

Figure 2.9: Junior Soccer League at RoboCup 2011.

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

Soccer

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

Dance

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

Rescue

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

Chapter 3

RoboCup 3D Simulation League

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

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

3.1 SimSpark Soccer Simulator

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

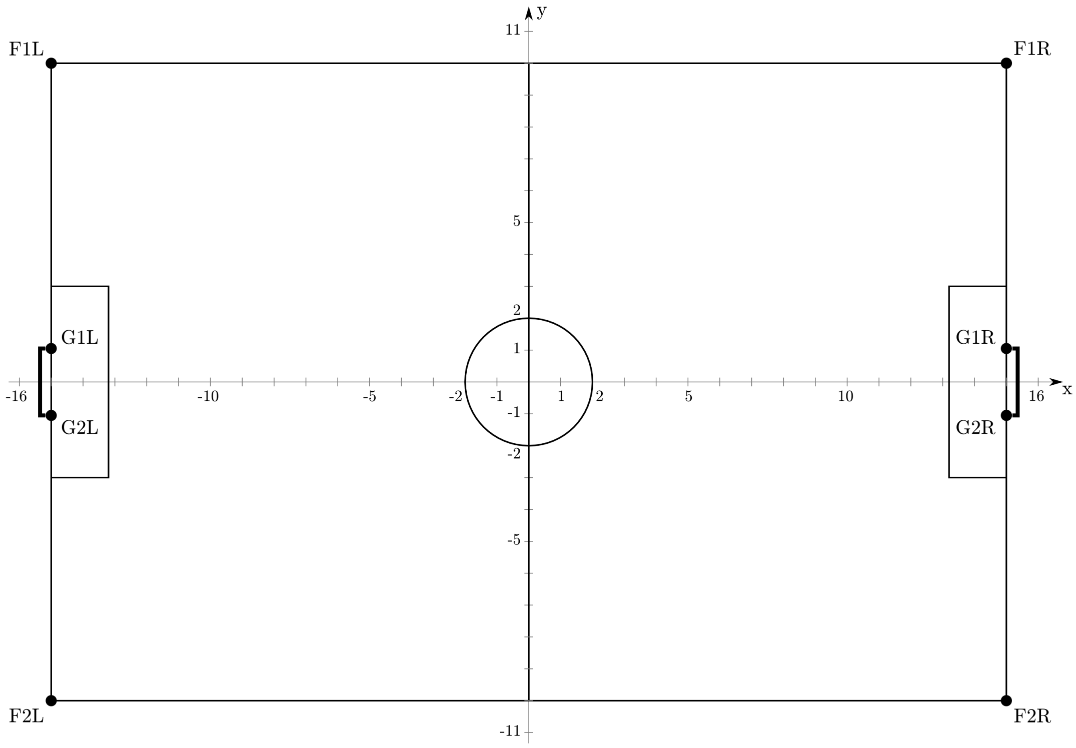

Figure 3.1: RoboCup 3D Simulation League Field

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

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

3.2 Robot Model

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

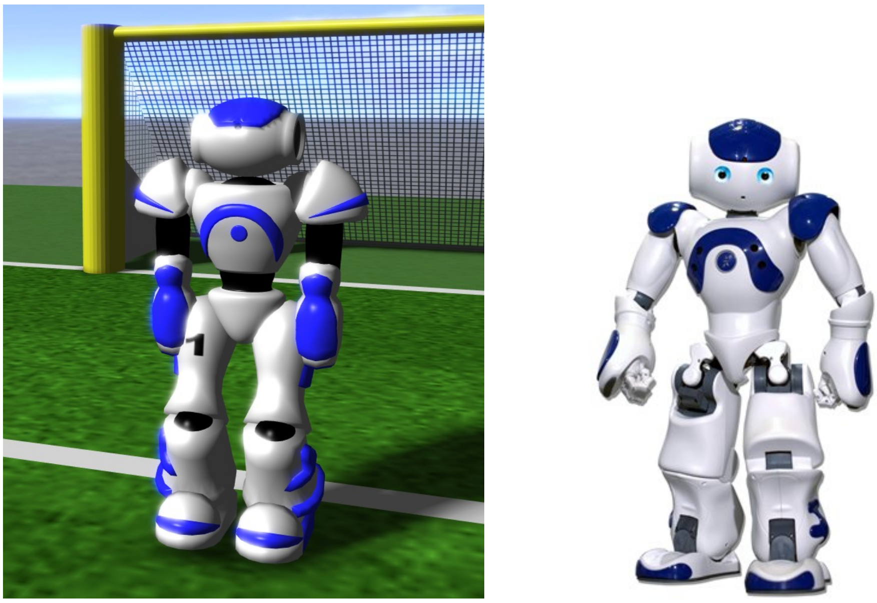

Figure 3.2: Rcssserver3d Simulated (left) and Aldebaran Real (right) Nao Robot Models.

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

3.3 Server

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

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

3.4 Monitor

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

3.4.1 SimSpark Monitor

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

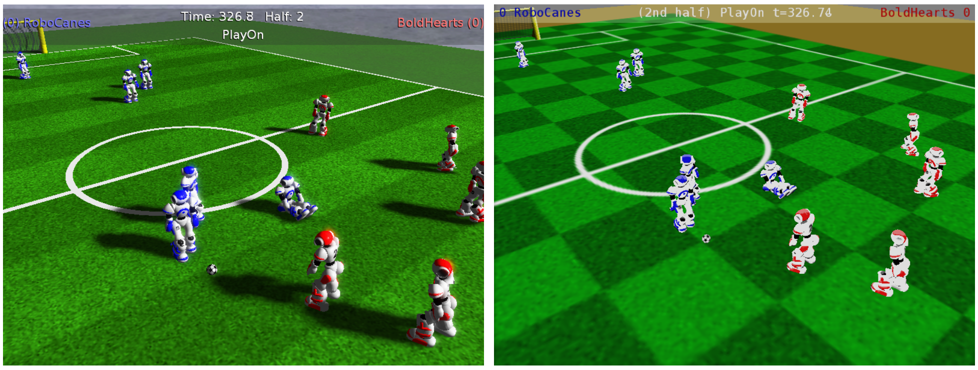

Figure 3.3: Roboviz (left) vs SimSpark (right) Monitors.

plugins. These plugins are intended to parse the game-specific state and display it as an overlay printed out on screen.

3.4.2 Roboviz Monitor

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

3.5 Perceptors

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

3.5.1 General perceptors

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

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

Frequency: Every cycle

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

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

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

Frequency: Only in cycles where a body collision occurs

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

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

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

Frequency: Every cycle

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

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

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

Frequency: Every cycle

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

3.5.2 Soccer perceptors

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

Message format:

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

Frequency: Every third cycle (60ms)

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

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

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

Frequency: Only in cycles, where a message is heard

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

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

Frequency: Every cycle

3.6 Effectors

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

3.6.1 General Effectors

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

Message format: `(scene <filename>)`

Frequency: Only once

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

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

Frequency: Once per cycle maximum

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

Message format: `(syn)`

Frequency: Every cycle

3.6.2 Soccer Effectors

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

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

Frequency: Only once

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

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

Frequency: Once before each kick-off

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

Message format: `(say <message>)`

Frequency: Once per cycle maximum

Chapter 4

Player Skills

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

4.1 Agent Architecture

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

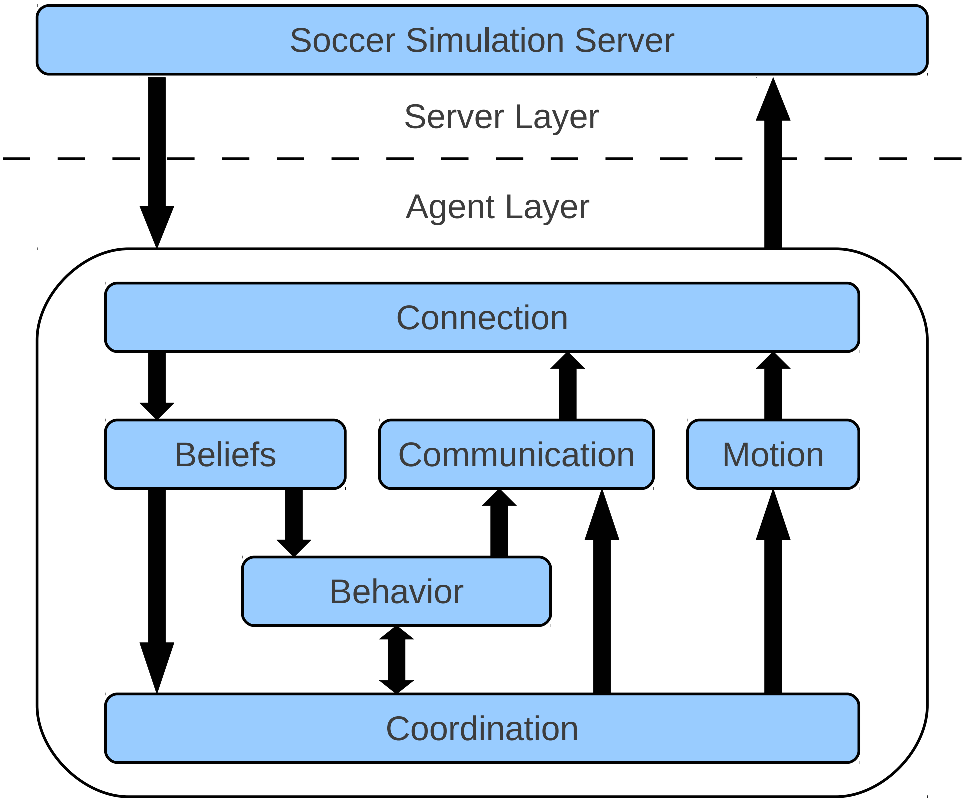
Soccer Simulation Server
Server Layer
Agent Layer
Connection
Beliefs
Communication
Motion
Behavior
Coordination

Figure 4.1: The Agent Architecture.

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

4.2 Connection

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

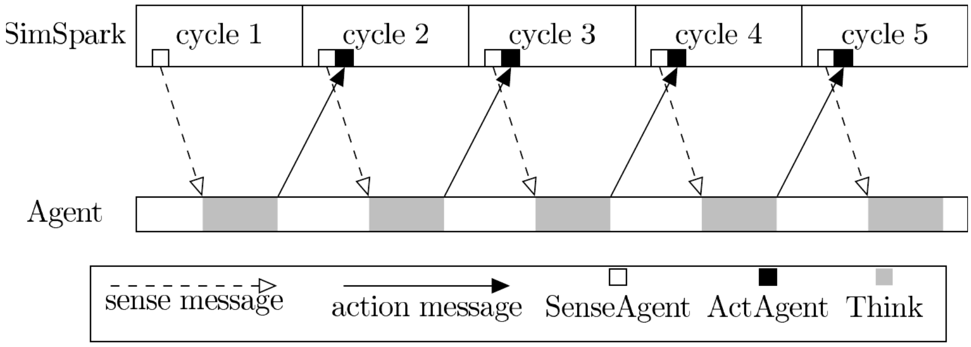

Figure 4.2: Server and Agent Communication.

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

4.3 Perceptions

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

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

The above message is an example message our agent may receive from the server during simulation. It includes information about the server time, the game state and time, values for each one of the joints, visual observations from the camera, and data from the accelerometer, gyroscope, and force sensors. We parse these messages and save the enclosed information in data structures appropriate for each type of perception.

4.4 Localization

Once we have all the new perceptions from the server available, we can update our agents' belief about its current self-location in the field and the current location of other objects of interest (ball, teammates, opponents). Our localization scheme is largely based on a method proposed by a colleague within a common course project [6]. Localization, as a process, is executed every three cycles (60ms), in fact every time we receive observations from the vision perceptor. A key restrictive factor is that the agents are equipped with a restricted vision perceptor which limits the field of their view to 120 degrees. It is easy to realize that the localization process would have been easier, if there was an omni-directional vision perceptor.

4.4.1 Self Localization

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

The self-localization function takes the distance, as well as the horizontal and vertical angles of two currently visible landmarks as input and returns a candidate self location (x, y, θ) as output, where (x, y) are the coordinates in the field and θ is the orientation of the body with respect to the angle system of the field. The two visible landmarks form two circles with radius equal to the observed distance to each one of them centered at the

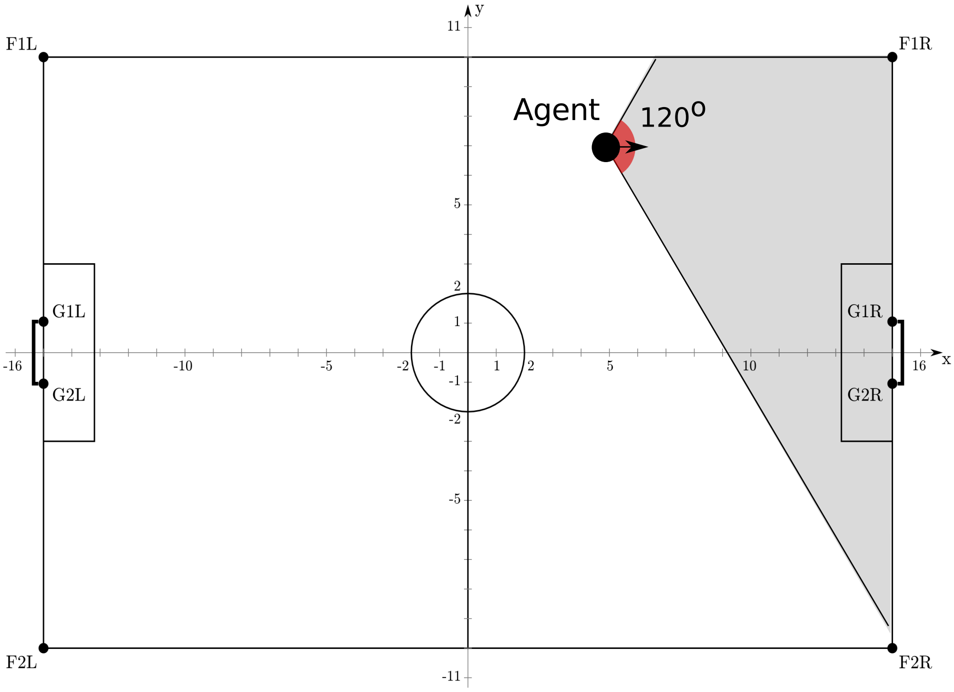

Figure 4.3: Nao's Restricted Field of View and Field Landmarks.

static and known coordinates of these landmarks. Obviously, these two circles intersect at two points, which represent two candidate self locations. We reject the wrong candidate using the horizontal viewing angles of the two landmarks and also the fact that the correct candidate cannot be way outside the field. Figure 4.4 shows a typical example of self localization with two landmarks. In cases where the agent sees more than two landmarks, the self-localization function is called for each pair of landmarks. The final estimated location is computed as the average of the outcomes of all pairs. Apparently, when the agent has less than two visible landmarks in the current field of view, the self-localization update does not take place.

4.4.2 Object Localization

Apart from self localization, it is important to be able to compute the position of other visible objects, such as opponents and the ball (note that there is no need to localize teammates in the field; their beliefs are shared via communication). Knowing our own location in the field helps us locate other visible objects too. For each currently visible

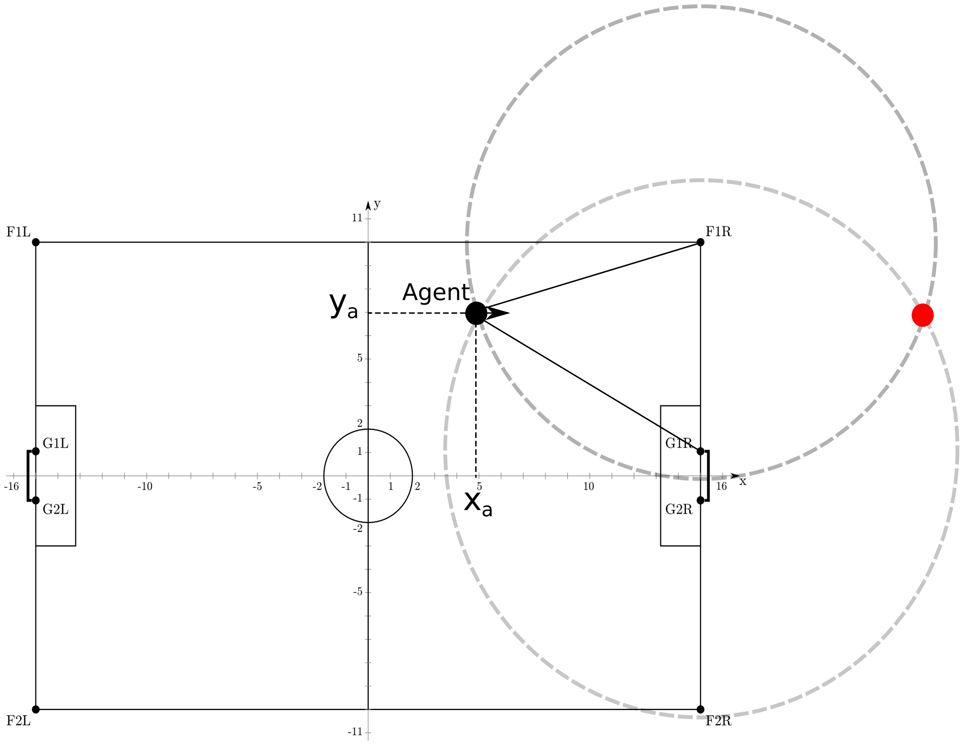

Figure 4.4: Self-Localization Example with the F1R and G1R Landmarks.

object, the vision perceptor informs us about its vertical and horizontal angles and its distance from the agent. This information is sufficient for the calculation of their exact positions into the field's coordinate system. Figure 4.5 shows an example scenario of calculating the positions of the ball and an opponent. To successfully locate other objects in the field, the same condition as in self localization applies: there must be at least two visible landmarks. If the currently visible landmarks are less than two, other objects cannot be located in the field using the current perceptions; nevertheless, the agent knows where they are located with respect to itself.

4.4.3 Localization Filtering

In the absence of a more sophisticated probabilistic localization scheme, we are forced to ensure that localization results are qualitative enough for us to rely on. Due to the temporary misses of landmarks from the field of view and the noisy observations from the

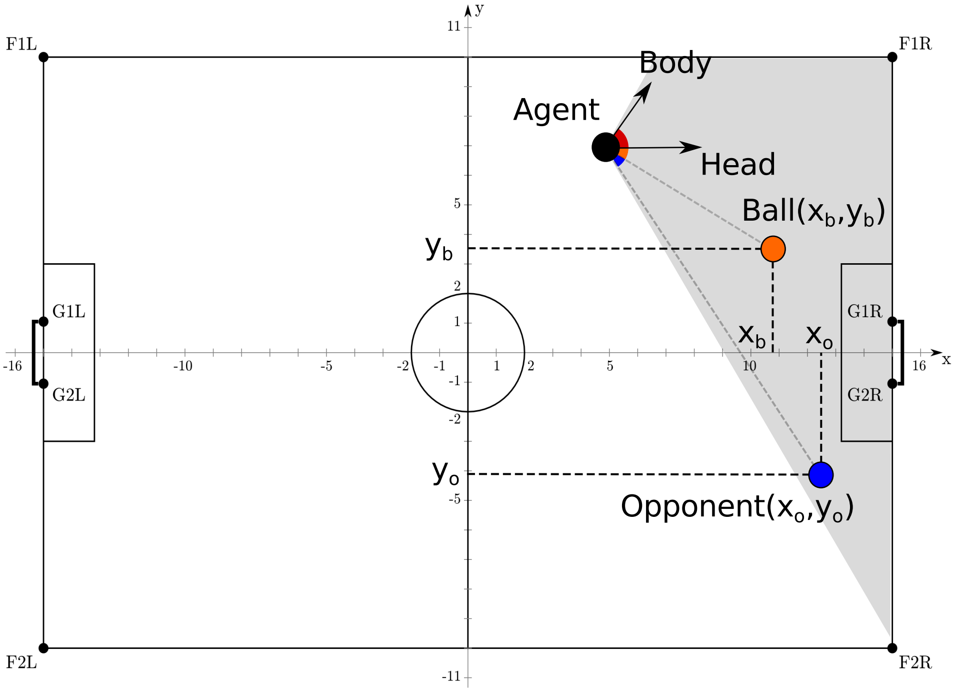

Figure 4.5: Object (Ball and Opponent) Localization Example.

vision perceptor, localization based only on the current perception is not always accurate enough to depend upon. Therefore, a simple filtering procedure on estimates computed by the localization process is employed to update the agent's belief about self locations and positions of other objects over time.

In general, the localization process provides the agent with numerous estimated locations over time. The general idea we adopt is based on the fact that these estimates do not include long sequences of consecutive faulty estimates. Therefore, it is fairly easy to ignore sporadic faulty estimates (outliers), while updating the agent's belief. To address this problem, we came up with a simple localization filtering algorithm outlined in Algorithm 1. We maintain a FIFO queue which stores the most recent non-faulty location estimates. The average of the locations in the queue consist the agent's belief about its location in the field at any time. When a new estimated location arrives from the localization process, we first check if the queue is empty; in this case, we simply insert this estimate into the queue. Otherwise, we check if the newly arrived estimate "agrees" with the current belief, whereby "agrees" means that they are not far apart.

Algorithm 1 Localization Filtering

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

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

If not, this estimate is ignored and additionally one element of the queue is removed. This step represents a simple way of discounting the current belief in the presence of an outlier estimate. If the outlier estimate corresponds to a correct location, it will persist, eventually it will discard the entire queue with the current belief, and afterwards it will initiate a new belief in the empty queue. Finally, if the new estimate "agrees" with the current belief, it is inserted in the queue to reinforce the current belief, replacing one element (the oldest one), if the queue has reached capacity. This simple filtering scheme smooths out the belief of our agent's location and rejects most faulty estimates.

Localization filtering applies both to the calculation of the agent's self location and to the calculation of the ball's position. For the sake of efficiency, we do not use it to filter opponent positions. As mentioned already, the filtered locations of teammates are shared among team members via communication. The end effect of localization filtering is a significant improvement on the localization outcome, so that we can rely upon it with confidence.

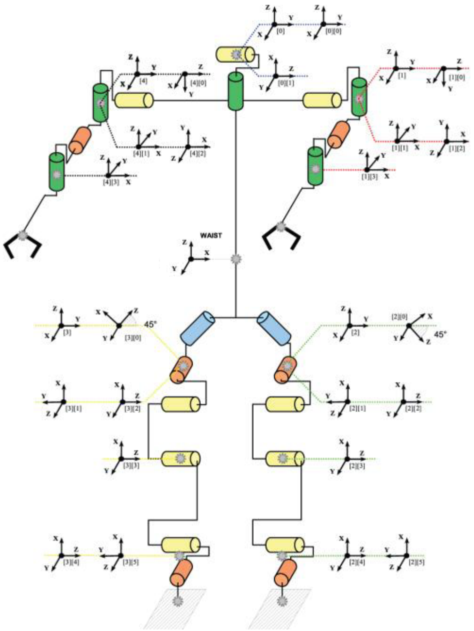

Figure 4.6: Nao's Anatomy: Kinematic Chains and Joints.

4.5 Motion

The simulated Nao robot comes with 22 degrees of freedom, corresponding to 22 hinge joints. Figure 4.6 shows Nao's anatomy with all joints, split in five kinematic chains (head, left arm, right arm, left leg, right leg). In robotics, a complex motion is commonly defined as a sequence of timed joint poses. A pose is a set of values for every joint in the robot's body or in a specific kinematic chain at a given time. For any given set of n joints a pose at time t is defined as:

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

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

4.5.1 XML-Based Motion Files

These motion files have been created and distributed by FIIT RoboCup 3D project [7] and they provide forward walk, left and right side steps, strong and regular kicks, stand-up, and left and right goalkeeper fall motions. These files have an intuitive XML structure, which facilitates integration into our project. The general structure of these XML-based motion files is shown below.

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

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

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

```
</phase>

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

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

4.5.2 XML-Based Motion Controller

The motion controller is a major component in our software which enables and controls the movement abilities of the robot. It is responsible for handling the movement requests of the agent. Agents do not have direct access to robot joints and joint poses, but can trigger desired motions by posting requests to the motion controller in the form of values to a specific variable. Each agent declares in this variable the movement he is willing to perform. In each cycle, the motion controller reads this variable and generates an appropriate hinge joint effector string, aiming at either terminating a running motion request (if different than the new one), or initiating the requested one (if there was no running request), or simply continuing with the execution of the current request (if initiated earlier). In effect, the motion controller satisfies a motion request posted by the agent, even if the completion of the motion takes several cycles, ensuring that transitions between different motions are smooth.

Figure 4.7 describes the general architecture of the XML-based motion controller. The motion controller checks if there is a motion which is playing already. If the currently

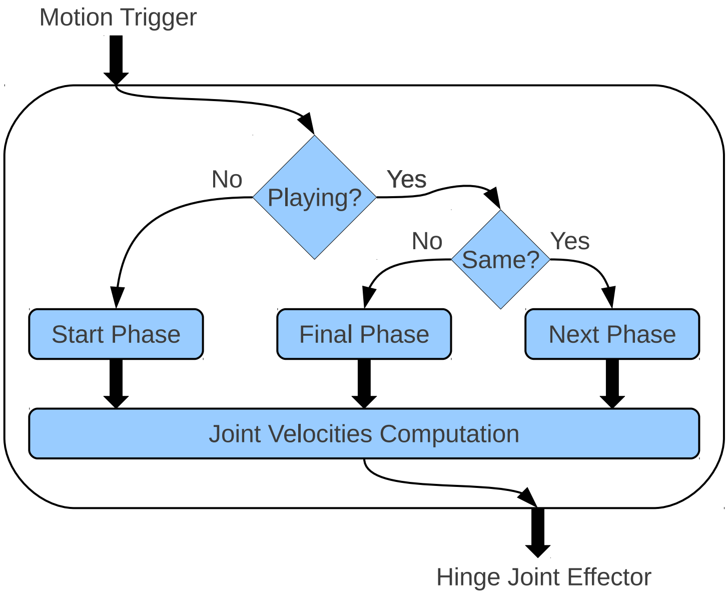

Figure 4.7: XML-Based Motion Controller.

playing motion is the same as the requested one, the motion controller continues with the next of its phases; if not, the controller tries to finalize the playing movement in order to start playing the newly requested one (in a future cycle).

Figure 4.8 shows the execution sequence of an example motion file. In general, XML-based motions define cyclic phases to generate continuous movement. For example, the walking motion has three main phases which create a cycle. If the motion trigger has not changed at the end of the last phase (`Phase 3`), we have to continue with the execution of the first phase (`Phase 1`), not with the final one. If the motion trigger is different at the end of the last phase, then we proceed to the final phase to terminate the current motion, before starting the new one (if any). As we saw in the structure of every XML-based motion file, each phase lists a set of joint values. These values are in degrees. To generate motions for our agent we need to create a motion string, which encloses information about each joint's velocity. The velocity of a joint i is computed as follows:

$$JointVelocity_i = \frac{DesiredJointValue_i - CurrentJointValue_i}{CurrentPhaseDuration}$$

The $CurrentJointValue_i$ is provided by the corresponding perceptor message. At the beginning of a phase, a velocity value is calculated for each joint involved in the motion

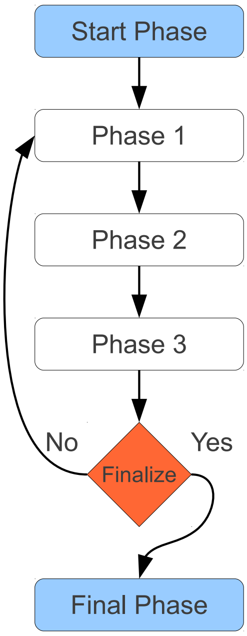

Figure 4.8: Phase Execution Sequence for a Typical XML-Based Motion.

and the final output of the motion controller is sent to the server. Note that zero velocity is set for every joint not included in the `effector` field of the current phase, so that these joints stop moving. These velocity values are not modified until the phase is completed.

4.5.3 Text-Based Motion Files

The text-based motion files we use have been adopted from the Webots simulator [8] and they provide left and right turn motions. These text-based motion files have simpler structure than the XML-based motion files. By default, each pose lasts two simulation cycles (40ms). The structure of these motion files is shown below.

```
#WEBOTS_MOTION,V1.0
LHipYawPitch,LHipRoll,LHipPitch,LKneePitch,LAnklePitch,...
00:00:000,Pose1,0,-0.012,-0.525,1.05,-0.525,0.012,0,...
00:00:040,Pose2,0,-0.011,-0.525,1.05,-0.525,0.011,0,...
00:00:080,Pose3,0,-0.009,-0.525,1.05,-0.525,0.009,0,...
00:00:120,Pose4,0,-0.007,-0.525,1.05,-0.525,0.007,0,...
00:00:160,Pose5,0,-0.004,-0.525,1.05,-0.525,0.004,0,...
```

```
00:00:200,Pose6,0,0.001,-0.525,1.051,-0.525,-0.001,0,...
00:00:240,Pose7,0,0.006,-0.525,1.05,-0.525,-0.006,0,...
00:00:280,Pose8,0,0.012,-0.525,1.05,-0.525,-0.012,0,...
00:00:320,Pose9,0,0.024,-0.525,1.05,-0.525,-0.024,0,...
```

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

4.5.4 Text-Based Motion Controller

The motion controller for text-based motions is based on the same principle as the XML-based controller. The joint values in the motion files represent radians, so we have to convert these values into degrees before we proceed. Each pose nominally corresponds to two cycles, but we can make it last one or two or three cycles depending on the speed (faster, normal, slower) at which we want the motion to be executed. The text-based motion controller can be customized to perform these motions in different ways. The following parameters can be modified:

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

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

The desired velocity of each joint i is computed by:

$$JointVelocity_i = \frac{DesiredJointValue_i - CurrentJointValue_i}{Duration \times CycleDuration}$$

Again, the $CurrentJointValue_i$ is provided by the corresponding perceptor message. A velocity value is calculated for each joint involved in the motion and the final output of the motion controller is sent to the server.

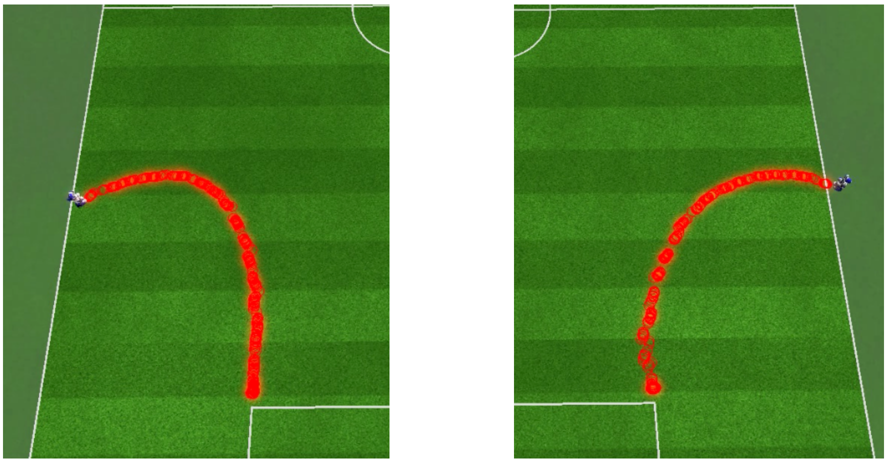

Figure 4.9: Dynamic Walk Leaning: Left Leaning (left), Right Leaning (right).

4.5.5 Dynamic Motion Elements

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

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

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

Turn Gain The text-based turn motions can be dynamically modified using a gain value for scaling the resulting velocities in order to perform the motion in a smoother or rougher way. By default, this gain value is set to 1.0, however slightly smaller or

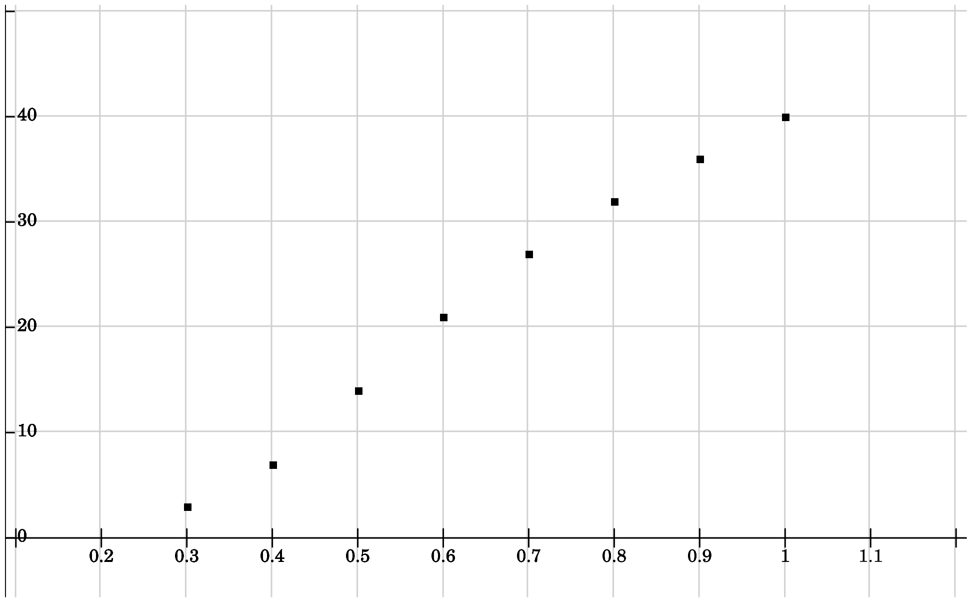

Figure 4.10: Dynamic Turn Gain: Turn Degrees (y-axis) against Gain Factor (x-axis).

larger values can result in useful variations of the defined motion. By dynamically changing this value between 0.3 and 1.0, the agent is able to turn its body anywhere between 7 and 40 degrees, as shown in Figure 4.10.

4.6 Actions

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

4.6.1 Basic Actions

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

Look Straight

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

Scan

Moves the head to perform periodic panning and tilting.

Pan Head

Moves the head to perform periodic panning at zero tilt.

Track Object

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

Track Moving Object

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

Find Opponent's Goals

This action finds the direction of the opponent's goal with respect to the agent by performing the Scan action until the opponent's goal becomes visible.

Look For Ball

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

Turn To Ball

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

Turn To Localize

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

Stand Up

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

Prepare for Kick

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

4.6.2 Complex Actions

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

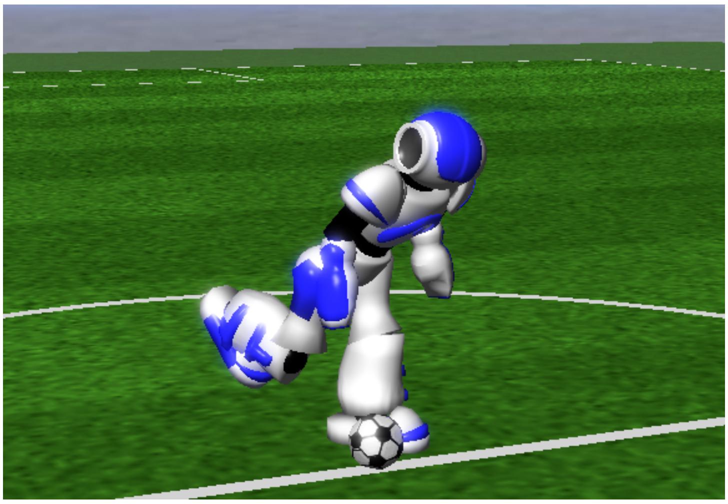

Figure 4.11: Nao Performing a Kick after Positioning for Kick.

Avoid Obstacles

This action records all obstacles around the agent and derives an obstacle-free route to a specified target. Initially, this action stores the positions of all obstacles located within 2m from the agent by performing the Pan Head action and watching the perceptor messages. Due to the fact that the simulated Nao's head can pan from $-120°$ to $+120°$ and the field of view is $120°$, we can obtain a complete imaging of all obstacles located close to our agent. It is common to observe the same obstacle more than once; in this situation we only store the average of these observations. In a dynamic, multi-agent environment it is important to avoid collisions with other agents or fixed landmarks, such as goal posts in our simulated soccer games. To avoid obstacles we rely on a simple, yet reliable and effective, method. For each recorded obstacle, we calculate two escape angles that determine the two directions (relative to the agent), which guarantee avoidance of the obstacle at a safe distance. All other angles between these two escape angles are considered to be forbidden. Afterwards, any escape angle of some obstacle that falls within the forbidden area of some other obstacle is discarded. The precise calculation of the escape angle set is described in Algorithm 2. The remaining escape angles, and particularly the escape way points they define (the points closest to the obstacle along the direction of the escape angles), are evaluated in terms of the angle and distance overhead they incur with respect to the agent orientation (for the angle) and the target (for the

Algorithm 2 Escape Angle Set Calculation

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

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

distance). The way point that minimizes the total overhead is selected as a temporary target for avoiding the obstacles, while making progress towards the target. This method yields dynamically consistent results, meaning that the temporary target does not change in subsequent cycles as long as the obstacles remain stationary, until they are cleared on the way to the real target. Obstacle avoidance is demonstrated in Figure 4.12, in a simple scenario, where there are two obstacles between the agent and its target position. Two of the escape angles are discarded; the way point on the left is selected as a temporary target to clear the obstacles.

Walk to Ball

Makes the agent walk towards the ball and stop when the ball is close enough to perform a kick. First, it performs the Turn to Ball action and then walk towards the ball, slowing down when it comes close to the ball. Recall that the ball distance returned by the vision perceptor is the distance between the camera, which is attached to agent's head, and

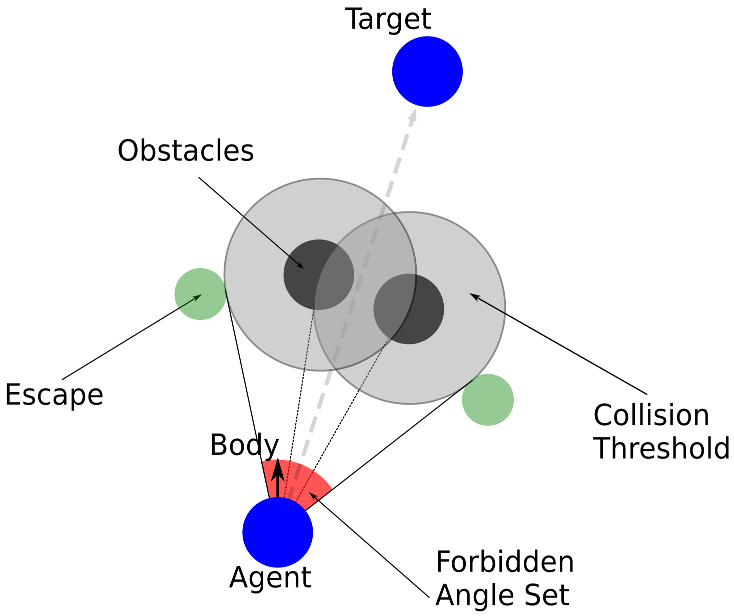

Figure 4.12: Obstacle Avoidance.

the ball, which lies on the ground. However, for approaching the ball with precision, we need the distance on the ground between the agent's feet and the ball. To compute this distance, we first use forward kinematics along the sagittal plane of the robot to derive the current height of the camera. Taking the agent's ankle as the origin, it is easy to calculate every joint's position from ankle to head in the two-dimensional space of the sagittal plane using only the current values of the AnklePitch, KneePitch, HipPitch joints. Having the ball distance and the height of the camera, the ground distance can be easily derived using the Pythagorean Theorem.

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

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

On Ball Action

This action moves the agent close to the ball and executes an appropriate kick depending on the current state of the game. This action has a finite state machine logic shown in Figure 4.14. It first performs the Walk To Ball action in order to reach the ball. After the successful completion of the Walk To Ball action, the agent performs the Find Opponent's

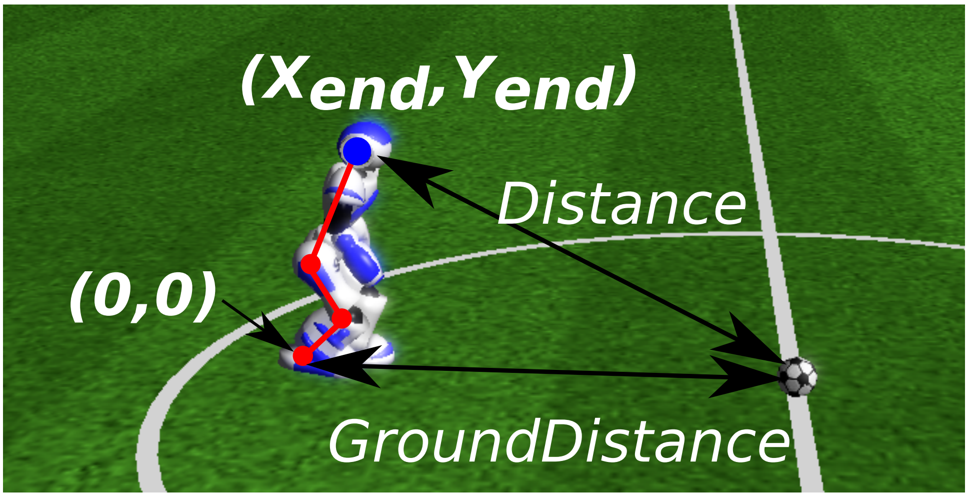

Figure 4.13: Ground Distance between the Agent and the Ball.

Goal action. Subsequently, it aligns itself with the direction of the opponent's goal; the precision of this alignment is inversely proportional to the distance from the opponent's goal, meaning that far away from the opponent's goal there is more tolerance, since we only want to clear the ball from our own half of the field. Afterwards, it performs the Position for Kick action and finally it executes a kick motion. At any point of time, it is possible that an opponent agent takes the ball away from our agent; if that happens, the agent returns to the beginning.

Walk to Coordinate

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

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

Figure 4.15 shows an example of this calculation. The actual locomotion of the agent towards the target is done through the Avoid Obstacles action, which takes the pair (d, ϕ) as input and ensures that obstacles are avoided on the way to the target. Note that

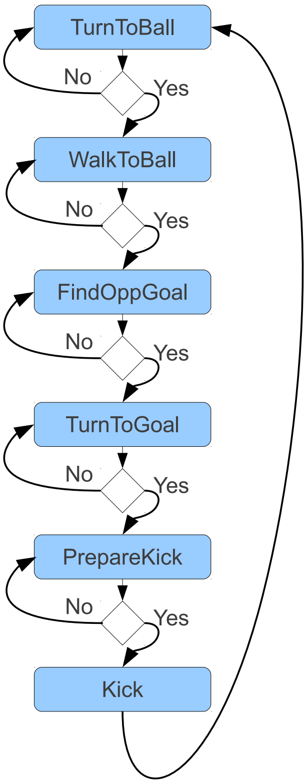

Figure 4.14: On Ball Action Logic Flowchart.

distance and direction are recalculated as the agent makes progress toward its target. After the position (x_t, y_t) has been reached, a final rotation in place turns the agent towards the desired direction θ_t. The action terminates when the agent reaches the desired location (x_t, y_t, θ_t).

Walk To Direction

This action makes the agent walk towards a specific direction. If the change with respect to the current orientation is small (default: 7°), it employs a left or right leaning walk action. For larger changes, it employs a turn action to align with the given direction in parallel to a straight walk action to move along the given direction. The turn action ceases as soon as the agent has aligned its orientation with the given direction. The Walk to Direction action terminates only when a new action request or an "empty" action request is posted. The actual locomotion of the agent towards the target direction make use of the Avoid Obstacles action at all times.

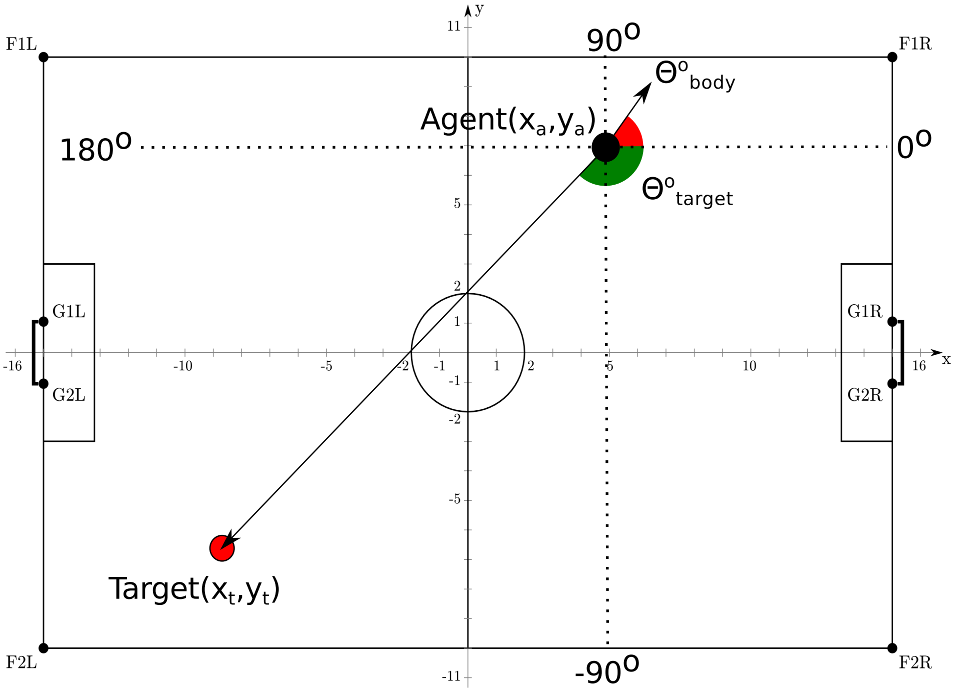

Figure 4.15: Walk To Coordinate Action.

Dribble Ball To Direction

This action attempts to dribble the ball towards a specific direction. It is quite similar to the Walk To Direction action, however the agent tries to keep the ball in front of its feet at all times. The actual locomotion of the agent towards the target direction is done through the Avoid Obstacles action. This action is not fully functional in our software, since movements based on motion files make it hard to keep ball consistently in front of the agent.

4.7 Communication

Communication in Simspark is not ideal. There are no restrictions about the use of the say effector and every agent can use it at each cycle. However, the hear perceptor comes with some restrictions (cf. Section 3.5.2). Messages shouted from beyond a maximum distance (currently 50 meters) cannot be heard. Given that the dimensions of the field are currently only $21m \times 14m$ (version 0.6.5) or $20m \times 30m$ (version 0.6.6) (about $25m$

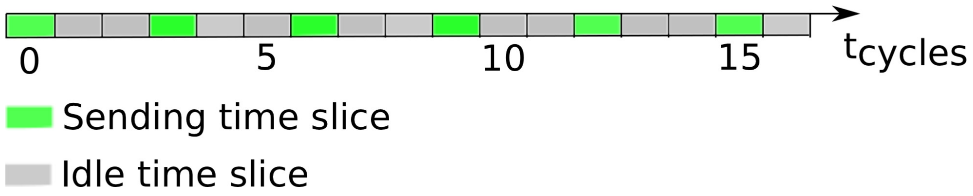

Figure 4.16: Time Slicing Communication Protocol.

or 36m in diagonal length), there is no limit in practice. However, due to the limited communication bandwidth, we utilize the communication channel in the following way, to ensure that every message sent from an agent will be heard by the other agents in time. A simple communication protocol has been created in which time is sliced into pieces lasting one server cycle (20ms) each and repeats every three cycles (60ms). Figure 4.16 shows how exactly time is sliced. The first of these three cycles is used for sending and the next two cycles are left idle on purpose to ensure that the message sent in the first cycle is delivered to all teammates. During the sending cycle, only one agent is allowed to send its message to the others. Every time slice of the protocol has an associated integer label which indicates the uniform number of the player able to send its message at that slice. This label starts at 1 and grows by 1 every time a player sends a message, until it reaches the maximum uniform number and loops back to number 1. Since there is no common clock, to ensure that the agents are synchronized we make use of the changing game states, which are shared and known to all players; whenever the game state changes, the agents reset their integer label counters. Through this simple protocol, every player can receive reliably the messages from all teammates every 540ms (27 cycles) for a team of 9 players or every 660ms (33 cycles) for a team of 11 players.

Chapter 5

Team Coordination

In this chapter, we are going to present the most important, exciting, and time-consuming part of this thesis. Up to this chapter, we have discussed all skills that agents need in order to be functional in the soccer field. With these functionalities agents are able to locate themselves in the field, communicate with each other, and execute actions combining movements through the motion controller. However, agents miss a thinking process with which they will be able to decide what action they should take for the global benefit of the team. In real human soccer, this will correspond to players with excellent individual skills for a soccer match, but no reasoning ability to choose what is best to do at each time. Therefore, it is crucial to come up with a high-level process which will coordinate all these skills, motions, communication ability, and actions yielding as a result a complete behavior for each agent within the frame of a global team strategy. As behavior, we define the process in which an agent takes as input arguments its beliefs and decides what to do as an output.

5.1 Coordination Protocol

In our approach, instead of each agent deciding its own behavior, players depend on a centralized process, called team coordination, which implicitly determines individual behaviors for each agent. The team coordination algorithm is responsible for gathering messages from all agents and producing appropriate actions for all agents towards achieving a common goal. We choose one player (by default, the goalkeeper) to act as the central coordinator, that is the one who is going to execute this team coordination

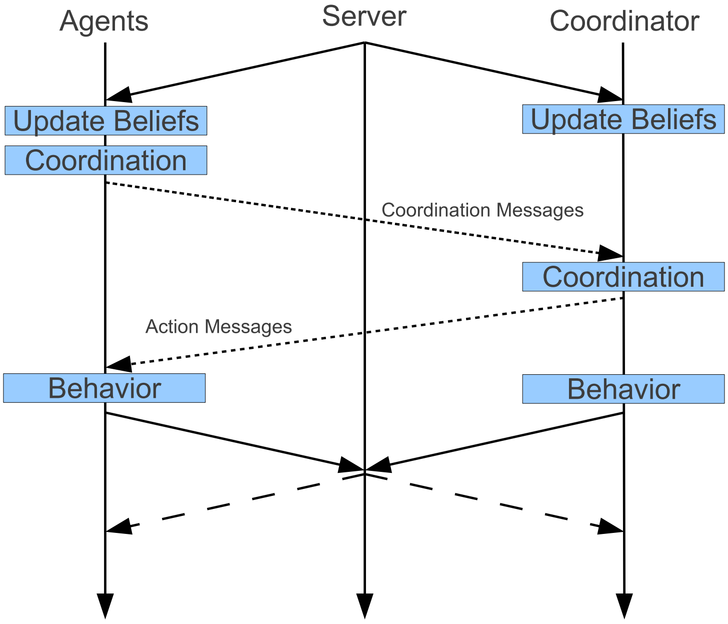

Figure 5.1: The Coordination Protocol.

procedure on behalf of the entire team. This means that all players communicate their beliefs to the goalkeeper, the goalkeeper executes the team coordination procedure for the entire team, and finally the goalkeeper sends back to the players the actions they are required to take.

Figure 5.1 shows the entire coordination protocol (which may last several simulation cycles). All players initially update their beliefs. The coordinator (goalkeeper) waits for messages from all other players. Once these messages are gathered at the coordinator, the coordination procedure is executed and the resulting actions for each player are communicated back to the players. At this point all players execute their actions, in essence realizing their behavior. We selected the goalkeeper to act as coordinator, because its role is quite distinct and independent from the other players and therefore it can afford to dedicate more computational resources to the team coordination algorithm compared to the other players.

The team coordination procedure executed only by the coordinator after all messages have been received is split in several phases:

Update Coordination Beliefs The local world state beliefs from the other players are combined in order to update the global belief about the world state (global ball location, distances of all players from the global ball location, locations of all players).

Determine Coordination Subsets Field players are split into non-overlapping subsets according to their significance in the current game state. These subsets are:

- *Goalkeeper*: one player, the goalkeeper
- *Inactive*: players fallen on the ground or players with lost self-location
- *Active*: three players, the ones closest to the ball
- *Support*: all remaining players

Determine Active Positions Several candidate positions are determined for the active players.

Coordinate Active Players The active player closest to the ball is assigned to go to the ball and the best pair of candidate positions is selected for the remaining active players according to a cost function.

Generate Team Formation A formation is generated for the entire team (excluding the goalkeeper) depending on the current position of the ball in the soccer field.

Assign Team Roles All team players, except the goalkeeper, are assigned roles taking into account the desired formation of the team and their current position.

Determine Support Positions Candidate positions are determined for the support players. These are determined by the desired team formation, after excluding the roles assumed by the active players and a number of least-significant roles equal to the number of inactive players.

Coordinate Support Players The best mapping between support players and support positions according to a cost function is computed.

Algorithm 3 describes the entire coordination procedure, which currently lasts six simulation cycles. In each simulation cycle, some, but not all, of the above phases are executed. This choice was dictated by the time limitation of the agent think cycle; with this choice, each group of phases fits within a single simulation cycle causing no delays and enabling real-time operation.

Algorithm 3 Coordination Protocol

```
**Input:** $CoordinationMessages = \{M_1, M_2, ..., M_{N-1}\}, N = NumberOfPlayers$
**Output:** $Actions = \{A_1, A_2, ..., A_{N-1}\}$

**if** $CoordinationCycle = 1$ **then**
    $B \leftarrow UpdateCoordinationBeliefs(CoordinationMessages)$
    $CoordinationCycle = CoordinationCycle + 1$
**else if** $CoordinationCycle = 2$ **then**
    $S \leftarrow DetermineCoordinationSubsets(B)$
    $CoordinationCycle = CoordinationCycle + 1$
**else if** $CoordinationCycle = 3$ **then**
    $P_{active} \leftarrow DetermineActivePositions(B, S)$
    $CoordinationCycle = CoordinationCycle + 1$
**else if** $CoordinationCycle = 4$ **then**
    $A_{active} \leftarrow CoordinateActivePlayers(P_{active}, S, B)$
    $CoordinationCycle = CoordinationCycle + 1$
**else if** $CoordinationCycle = 5$ **then**
    $F \leftarrow GenerateTeamFormation(B)$
    $R \leftarrow AssignTeamRoles(A_{active}, B, F)$
    $P_{support} \leftarrow DetermineSupportPositions(R, F, S)$
    $CoordinationCycle = CoordinationCycle + 1$
**else if** $CoordinationCycle = 6$ **then**
    $A_{support} \leftarrow CoordinateSupportPlayers(P_{support}, S, B, R, F, A_{active})$
    $Actions = A_{active} \cup A_{support} \cup A_{inactive}$
    $CoordinationCycle = 0$
**end if**
```

5.2 Coordination Modes

Coordination is not a static procedure and may change dynamically during different game states. There are three modes of team coordination:

Active This is the normal mode of coordination. Every aspect of the coordination process we have discussed above is used for the team's coordination and the computation of action for all field players.

Support In this mode, all players, excluding the goalkeeper, join by default the support subset. It is used in situations where our goalkeeper takes control of the ball or where only the opponent team has the right to perform a kick to the ball, e.g. opponent's kick-off, opponent's goal kick, etc.

Wait In this mode, all players, excluding the goalkeeper, join by default the inactive subset. It is used in situations where both teams have to wait for the kick-off signal before a kick-off.

5.3 Coordination and Communication

Coordination is accomplished through communication. We use the common communication channel through the simulation server in order to provide the messaging between players involved in the coordination process. For this reason, communication plays a major role in our approach. The general idea of this communication process among players is that the coordinator (goalkeeper) needs to know all agents' beliefs about the world state before proceed with the execution of the coordination protocol. Furthermore, communication is needed to send the outcome of coordination from the coordinator to all players.

There are several types of messages, each one of them having different functionality and serving a specific purpose. The arguments of each message are preceded by a single-letter identifier indicating the type of the message. The message types used in our coordination protocol are:

Init Message This type of message declares the presence of each agent in the simulation environment. All players, other than the coordinator, must send this message to the coordinator before the coordination protocol begins.

Message format: `i,<Agent Uniform Number>`

Start Message This type of message is sent only by the coordinator; it declares that all agents are now initialized and ready to begin the coordination process. Each agent receiving this message should immediately start sending coordination messages.

Message format: `s,<Coordinator Uniform Number>`

Coordination Message This is the most important type of message. It includes information about each agent's beliefs. There are four subtypes of this message depending on the current beliefs of the agent:

Type C The agent has complete awareness of the ball and self location; the message includes the uniform number, the self position, and the ball position.

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

Type L The agent has complete awareness only of his own position in the field; the ball is not currently within the field of view and, even though there may be a filtered belief about the ball location, it is better to avoid sending possibly faulty information. The message includes only the uniform number and the self position.

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

Type B The agent has complete awareness of the ball's location, only with respect to itself. The message includes the uniform number, the distance of the ball and the horizontal angle of the ball relative to its own body angle.

Message format: `b,<Agent Uniform Number>,<Ball Distance>,<Ball Angle>`

Type X The agent has complete unawareness of the ball and self location or has fallen on the ground. The message includes only the uniform number. These agents join the inactive subset.

Message format: `x,<Agent Uniform Number>`

End Message This type of message asks the players, other than the coordinator, to stop sending coordination messages. At this point, the coordinator is ready to execute the coordination procedure and calculate actions for all players.

Message format: `e,<Coordinator Uniform Number>`

Action Message This type of message is sent only by the administrator; it declares which action each agent has been assigned by the coordination process. These messages are sent at the end of the coordination procedure, when actions for all players have been computed. The message includes the uniform number of the recipient agent, the action identifier, and the possible action parameters.

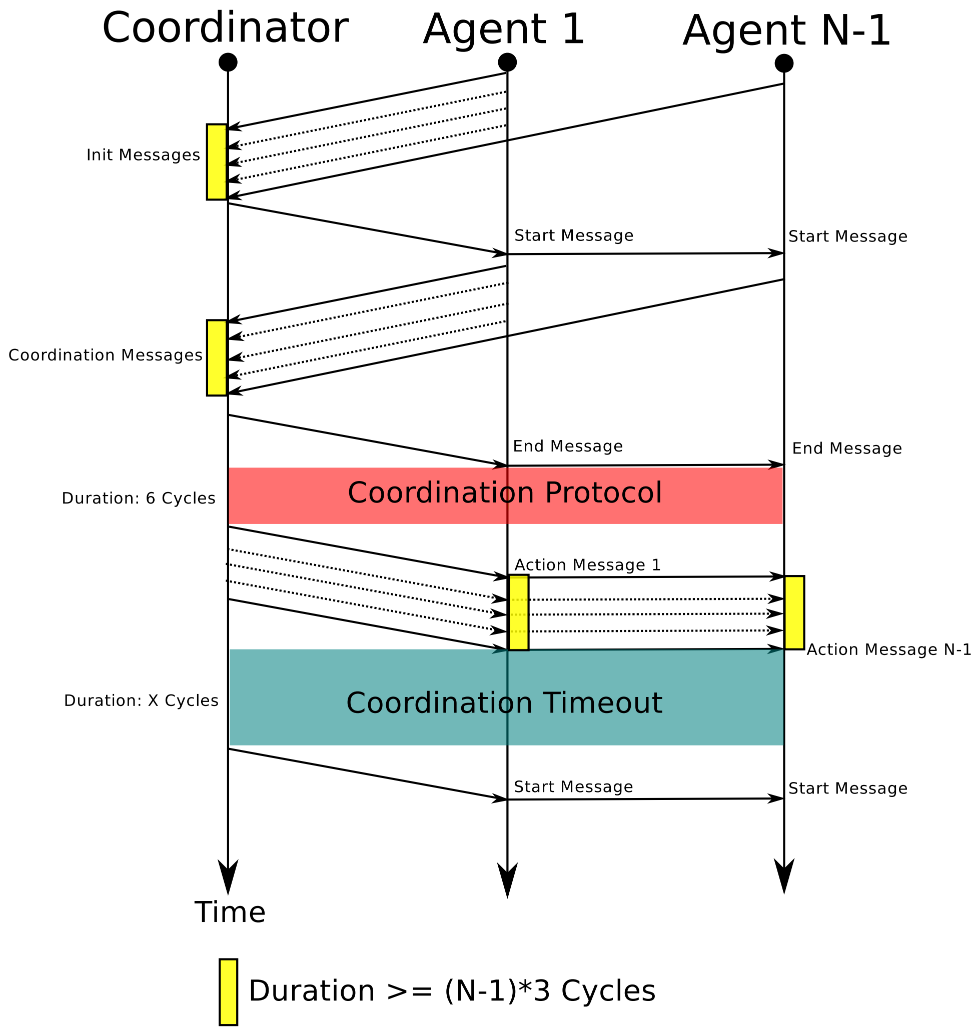
Coordinator
Agent 1
Agent N-1
Init Messages
Start Message
Start Message
Coordination Messages
End Message
End Message
Duration: 6 Cycles
Coordination Protocol
Action Message 1
Action Message N-1
Duration: X Cycles
Coordination Timeout
Start Message
Start Message
Time
Duration >= (N-1)*3 Cycles

Figure 5.2: Communication Process in Coordination.

Message format: `a,<Agent Uniform Number>,<Action ID>,<Action Parameters>`

Figure 5.2 presents the messaging procedure between the agents in order to coordinate their actions. First, agents have to initialize their presence in the field with “init” messages. The coordinator saves these messages and, when all other players have initialized themselves, broadcasts a “start” message. This message means that all players are now ready to start the coordination process. In this phase, all players, other than the coordinator, send their “coordination” messages to the coordinator. When the coordinator gathers these messages from all players, it broadcasts an “end” message to make them stop sending unnecessary “coordination” messages. The next phase of this process is the execution of the coordination protocol which lasts six simulation cycles, approximately 120ms. When it finishes, the coordinator broadcasts the resulting actions to each one of the other players using individual “action” messages. The receiving agents execute the commanded actions, until a new “action” message arrives. While action execution is ongoing, after a timeout period defined by the user (currently set at 20 simulation cycles), the same coordination process is repeated, excluding the initialization phase.

5.4 Coordination Beliefs Update

In the previous section we presented how players exchange messages with the coordinator. In this section we are going to discuss how the coordinator fuses the individual beliefs received by the other agents into a single global belief. This step is of major importance in any multi-agent system. Having multiple observations of the same world could potentially be a problem. The coordinator has to combine these observations without knowing which one of them is faulty or correct in order to obtain a global realistic representation of the world. Knowledge of the ball and agents’ positions is sufficient to execute the coordination algorithm without making guesses.

The global ball position is computed taking into account only the information communicated by the agents using “type C” coordination messages. Furthermore, information of the coordinator is taken into account, if the coordinator also has sufficient knowledge about the ball and self location. Apparently, the maximum number of ball observations at any time is the number of all agents, when all of them have sufficient knowledge about the ball and self location. As shown in Figure 5.3, different ball observations can differ from each other. In our approach, we use a simple algorithm to estimate the global ball position with accuracy. A threshold is defined (by default, 1 meter) in order to split the observations in clusters. Each cluster is represented by the average of the observations it contains. The first incoming observation defines the first cluster. For each subsequent incoming observation, we check its distance against all representatives of all existing clusters. Focusing only on the distances that fall below the threshold and the corresponding cluster, the new observation is assigned to the largest of these clusters, updating at the same time the representative of that cluster. Otherwise, the new observation gives rise to a new cluster with a single member. Each cluster is assigned a weight so that the cluster

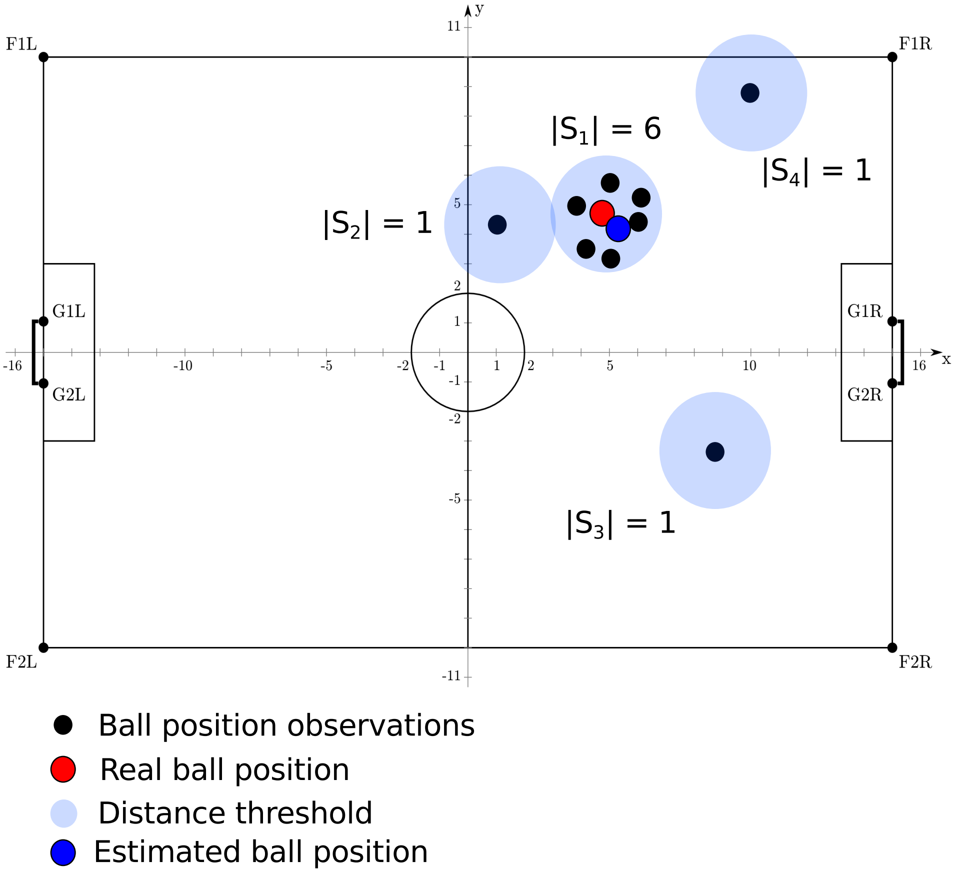
y
11
F1L
F1R
|S1| = 6
|S4| = 1
|S2| = 1
5
2
1
G1L
G2L
G1R
G2R
-16
-10
-5
-2
-1
1
2
5
10
16
x
-1
-2
-5
|S3| = 1
F2L
F2R
-11
● Ball position observations
● Real ball position
● Distance threshold
● Estimated ball position

Figure 5.3: Global Ball Position Estimation from Multiple Ball Observations.

with the most observations is naturally assigned the biggest weight. Our choice for the weight function is the size $|s_i|$ of cluster s_i cubed, that is $w(s_i) = |s_i|^3$. Figure 5.3 shows the resulting four cluster for a given set of nine observations. Consequently, we have to compute our belief about the global ball position. This is taken to be the weighted average of the cluster representatives. More specifically, given m clusters s_i containing ball observations o_{ij} each, the final ball belief is computed as:

$$\textbf{GlobalBallBelief} = \sum_{i=1}^{m} \frac{w(s_i)}{\sum_{k=1}^{m} w(s_k)} \sum_{o_{ij} \in s_i} \frac{o_{ij}}{|s_i|}$$

The next step towards forming the global belief is to determine each agent's position and its distance from the estimated global ball position. Players who have sent a "Type C" or "Type L" coordination message have already communicated their exact position in the field; their distance from the ball is simply calculated as the distance between the estimated global ball position and the players' position. Players who have sent a "Type B" coordination message don't know their exact position in the field, but this can be inferred using the estimated global ball position and the information they submitted about the distance and angle of the ball observation with respect to themselves; their distance from the ball is simply the one they reported. Finally, for players who have sent a "Type X" coordination message, we assume an infinite distance to distinguish them from all other players; their position in the field cannot be inferred.

5.5 Determination of Coordination Subsets

The existence of multiple agents makes coordination a complex and computationally expensive problem to be solved by a single agent. In our case, the coordinator (goalkeeper) would have to solve this huge problem for a total of nine players or eleven players (depending on the server's version). Instead of trying to solve the coordination problem at one level, a simple approach is to first decompose the problem into smaller coordination problems and then solve each one of them in isolation. To this end, we split the players into small subsets in which coordination can be achieved quickly to meet real-time requirements. In our approach, there are four such subsets:

Goalkeeper This subset includes only one agent, the goalkeeper. The player with the uniform number 1 is selected to be the agent responsible for guarding our goal. Goalkeeper also serves as the coordinator. His independent and individual behavior is explained in Section 5.13.

Inactive subset The Inactive subset consists of agents who have sent "Type X" coordination messages, that is players fallen on the ground or players with lost self-location. It is the less important subset of agents in the coordination procedure and in fact there is no need to coordinate them. Agents in this subset are assigned the same action, namely the Turn to Localize action. After they find their positions, they will have a chance to enter the active or support subsets in the next coordination cycle.

Active subset The Active subset consists of three agents, the ones closest to the ball, and it is the most important subset of agents in the coordination. Agents in this subset are responsible for executing the most critical actions for their team. Moreover, due to the small size of this subset, we can afford to solve their coordination problem by exhaustive enumeration to obtain an optimal solution.

Support subset The Support subset consists of all remaining agents. The size of this subset, which varies between 0 and 8 or 10 (depending on the server's version), may result in prohibitive computational costs for optimal coordination, therefore we adopt an approximate, yet effective, coordination method based on dynamic programming.

Figure 5.4 presents an example of coordination subsets. Assuming that all agents have complete awareness of their position, it is easy to realize that the agents within the red distance threshold will join the active subset and the other five agents who have farther distances from ball (yellow distance threshold) will join the support subset. Moreover, a possible existence of one or more agents having unawareness about the world state, should lead them joining the inactive subset. The goalkeeper is in its own subset.

5.6 Soccer Field Utility Fuction

In order to proceed our discussion about the coordination process, we have to define a simple, yet functional, way to give a value to every spot of the soccer field. Figure 5.5 presents this utility function over the entire soccer field, whose analytical form is:

$$FieldUtility(x, y) = CurrentSide \times x \times \left(\frac{FieldWidth}{2} - |y|\right)$$

where $CurrentSide$ is either $+1$ or -1 depending on the side of the field we currently play, so that the highest value corresponds to the opponent's goal. The key idea is that, as the ball is heading towards the opponent goal, this value is becoming higher. In contrast, as the ball is heading towards our goal, this value is becoming lower. It is easy to realize that a high value at some spot means more chances to score a goal, whereas a spot with small value implies a dangerous game situation. This function will prove to be useful in the next steps of the coordination process.

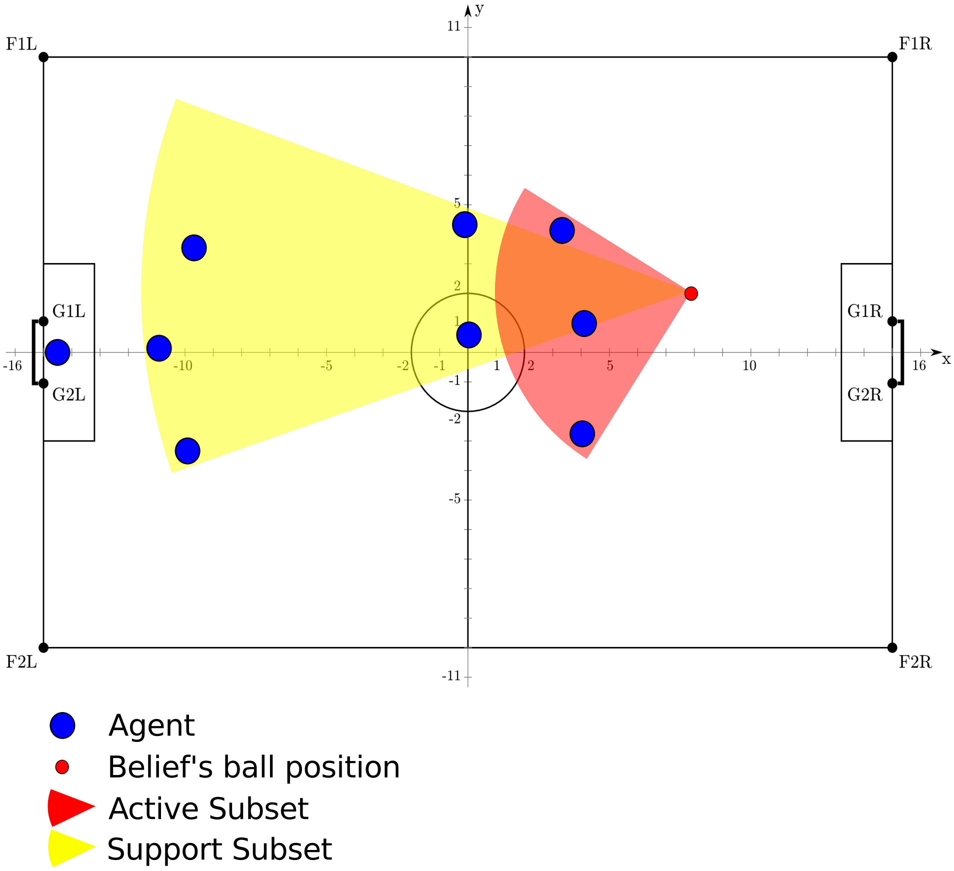
y
11
F1L
F1R
5
2
1
G1L
G2L
G1R
G2R
-16
-10
-5
-2
-1
1
2
5
10
16
x
-1
-2
-5
F2L
F2R
-11
Agent
Belief's ball position
Active Subset
Support Subset

Figure 5.4: Coordination Splitter.

5.7 Determination of Active Positions

Until now, we have updated the coordination beliefs and we have split agents into subsets. In this phase of the coordination process, we have to compute promising candidate positions for the Active subset. We distinguish between two cases. In the first case, the ball is located in our half of the field. In this case we have to find candidate positions which have a defensive approach. In the other case, if the ball is located in the opponent's half of the field, we have to find candidate positions which have an offensive approach.

In both cases, we start by creating a set of equidistant positions around the ball located at different radii, discarding any such positions that fall outside or near the soccer field's limits. For the defensive approach, we choose 12 positions at a radius of 1 meter and 30° apart and another 12 positions at a radius of 1.5 meters and 30° apart. For the offensive

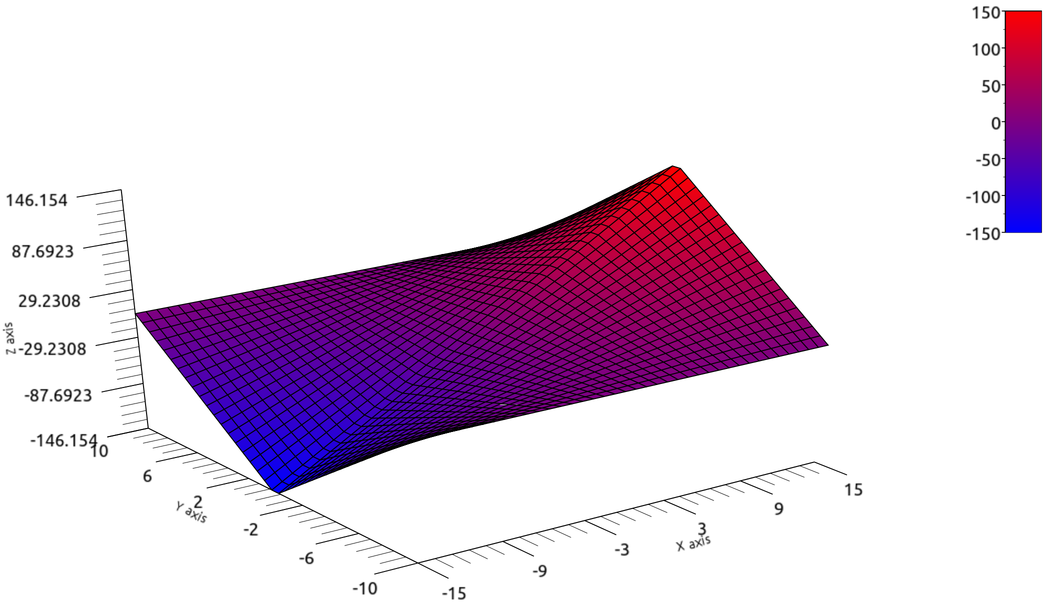

Figure 5.5: Soccer Field Value.

approach, we choose 12 positions at a radius of 2 meters and 30° apart and another 24 positions at a radius of $\max\{3, (|x| + |y|)/(FieldLength/2 + FieldWidth/2)\}$ meters and 15° apart. The idea in the offensive case, is that the more we approach the corners of the field, the more the candidate positions are spread around the ball. Figure 5.6 shows how these positions are determined in two different scenarios.

These sets of candidate positions is further filtered using the soccer field utility function. In the defensive case, we keep up to nine positions, the ones with the least utility values. These positions will mostly be between the ball and our own goal aiming to protect against an opponent strike. In the offensive case, we keep up to nine positions, the ones with the highest utility values. These positions will mostly be between the ball and the opponent's goal aiming to position players at attacking spots. Figure 5.7 shows the filtered sets of candidate positions for the Active subset in the same scenarios.

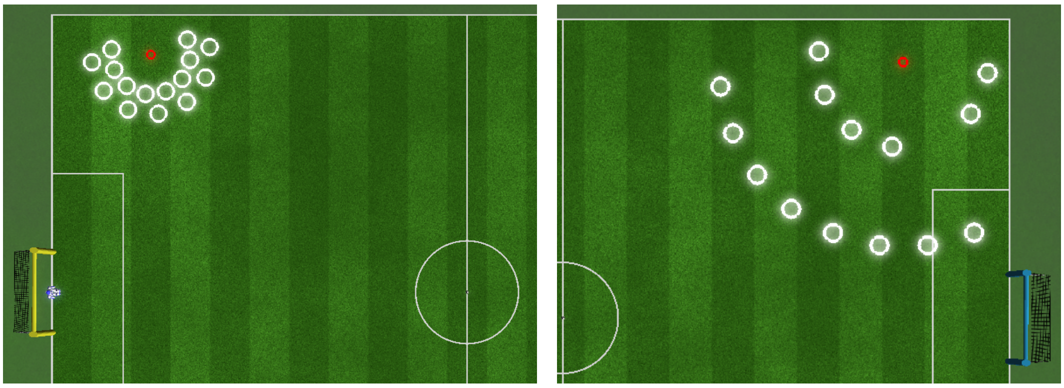

Figure 5.6: Initial Candidate Active Positions: Defense (left) and Offense (right).

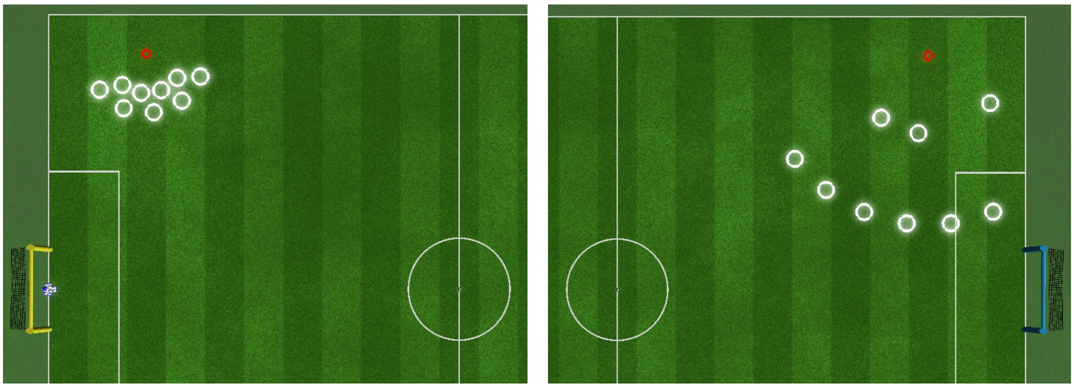

Figure 5.7: Final Candidate Active Positions: Defense (left) and Offense (right).

5.8 Active Players Coordination

Our coordination scheme for the Active subset requires that one player will undertake the task of approaching the ball, whereas the other two players will have to choose a pair of positions from the candidate Active positions. This is accomplished by a mapping function.

Player on Ball

An agent from the Active subset has to be selected in order to execute the On Ball action. We have to find the agent who minimizes a weighted sum over the following two terms:

1. **Distance from ball** d_i, the distance of agent i from the ball.

Algorithm 4 Active Players Optimal Mapping

```
Input: $ActivePlayers = ActiveSubset - OnBallPlayer = \{Agent_1, Agent_2\}$
Input: $ActivePositions = \{P_1, P_2, ..., P_N\}, N \leq 9$
Output: $OptimalActiveMapping$

$OptimalActiveMappingCost = +\infty$
for each $(P_i, P_j) \in ActivePositions \times ActivePositions, i \neq j$ do
  $ActiveMapping = \{Agent_1 \leftarrow P_i,\ Agent_2 \leftarrow P_j,\ OnBallPlayer \leftarrow Ball\}$
  $ActiveMappingCost = ActiveCost(ActiveMapping)$
  if $ActiveMappingCost < OptimalActiveMappingCost$ then
    $OptimalActiveMapping = ActiveMapping$
    $OptimalActiveMappingCost = ActiveMappingCost$
  end if
end for
return $OptimalActiveMapping$
```

2. **Angle towards goal** ϑ_i, the sum of the absolute angles between the agent and the ball and between the ball and the opponent's goal.

Given an Active subset with 3 players, the On Ball player is determined as follows:

$$\begin{aligned}\textbf{ActiveSubset} &= \{Agent_1, Agent_2, Agent_3\} \\ \mathbf{Value_i} &= d_i + a\vartheta_i, \text{where } a \in \mathbb{R} \text{ is a weight, default=0.01} \\ \textbf{OnBallPlayer} &= \arg\min_i(Value_i)\end{aligned}$$

Additionally, we give a small advantage to the agent who had been assigned this action in the previous coordination cycle over the other players. We do this to prevent continuous changes in the assignment of the On Ball player, when several two agents have similar distances and angles from the ball. This advantage takes the form of a value decrease (default: 1).

Active Players Optimal Mapping

Next in the active coordination phase, we have to assign positions for the other two agents left in the Active subset. Algorithm 4 shows how we can find the optimal mapping by exhaustively enumerating and evaluating each possible assignment. During evaluation, we also take into account the position assignment of the On Ball agent, which will be helpful in order to find possible collisions between the On Ball agent and the remaining active ones. More specifically, the evaluation function scores each possible mapping using the following features defined for each agent i:

1. **Distance** $C_{d,i}$ - This is the distance between the current and the target positions of agent i. Agents try to minimize this feature to be able to reach their target positions as soon as possible.

2. **Potential Collisions** $C_{c,i}$ - Approximating the route of each agent as a straight line between the current and the target positions, we check if the route of agent i intersects with the routes of the other two agents. For any pair of routes, if there is no intersection, the cost is 0 (infinitesimal probability of collision). If there is intersection and the intersection point is approximately equidistant from the current positions (the absolute difference is less than 1.5 meters), the cost is 100 (high probability of collision). Finally, if there is intersection but the intersection point is not equidistant from the current positions, the cost is 2 (small probability of collision). This feature returns the sum of these costs over the two pairs of routes considered for agent i. Figure 5.8 shows the key idea behind the detection of a possible collision between two agents. If $|d_1 - d_2| \leq 1.5$, a collision is highly possible. Agents try to minimize this feature to avoid collisions.

3. **Field Utility** $C_{u,i}$ - This is the absolute value of the field utility (cf. Section 5.6) for the position assigned to player i. Agents try to maximize this feature to give preference to positions with better utilities.

4. **Close Targets** $C_{t,i}$ - This is the sum of absolute differences between the target positions of agent i and the other two agents. Agents try to maximize this feature to give preferences to targets that are not too close.

5. **Horizontal Stretch** $C_{h,i}$ - This is the sum of absolute differences between the X-axis coordinates of the target positions of agent i and the other two agents. Agents seek to maximize this feature to stretch horizontally in the field and have better region coverage and unblocked fields of view.

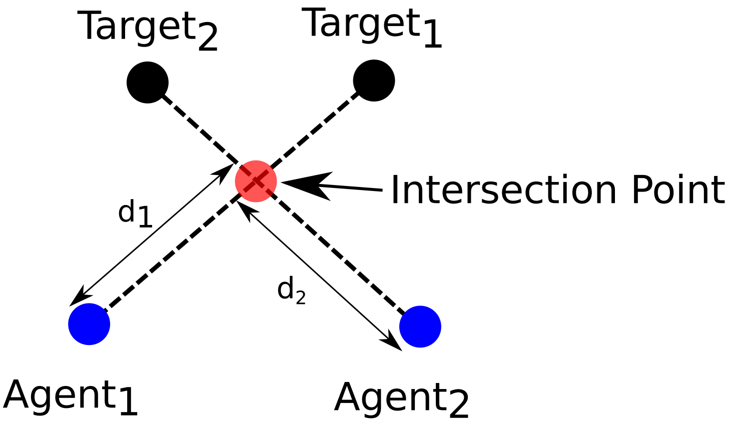

Figure 5.8: Collision Detection Feature in the Evaluation Function.

The features described above are computed for each of the three agent, are weighted, and are summed to form the final evaluation function of a mapping:

$$ActiveCost(ActiveMapping) = \sum_{i=1}^{3} w_d C_{d,i} + w_c C_{c,i} - w_u C_{u,i} - w_t C_{t,i} - w_h C_{h,i}$$

where $(w_d, w_c, w_u, w_t, w_h)$ are the weights of the features, currently set at $(1, 1, 1/7, 1, 1)$.

Despite the exhaustive enumeration, the number of possible mappings remains small enough to find the best one in real-time. Given that the maximum number of candidate active positions in our case is 9 and the numbers of agents is 2, the number of all possible mappings is $\binom{9}{2}2! = 72$. In general, for n positions and k agents, we would have to consider $\binom{n}{k}k! = \frac{n!}{(n-k)!}$.

5.9 Team Formation Generation

Team formation itself is not of major focus in this thesis, but serves to set up the role assignment function and the coordination of the support subset. In general, the formation of the team is determined by the position of the ball into the field. The formation is broken up into four groups and includes all players of the team. This section presents a simple for generating team formations for both the 0.6.5 and 0.6.6 versions of the *rcssserver3d* soccer simulator.

9-Players Server Version (0.6.5)

Table 5.1 shows the four groups of roles in our formation for the 9-player team, the corresponding short identifier, and a short description for each role. Figure 5.9 shows how the different role positions of the formation are depicted into the soccer pitch. Since the formation is a function of the ball position, when the ball is located near or outside the field limits formation positions are adjusted to not exceed the field limits.

The Forwards are positioned so that they are ready for an attack at all times. If the ball is on the opponent's half of the field, Forwards are assigned positions near the ball. In particular, the Forward Center is given a position close to the ball and the other two Forwards are given positions on either side of the ball in an angle and a distance offset which are dynamically determined by the exact coordinates of the ball. If the ball is located in our half of the field, then the Forwards are given positions near the ball as in the previous case, however their positions cannot be below a limiting X-axis coordinate which is close to the middle of the field.

On the other hand, Defenders are mainly positioned to guard our goal. To determine their position on the field a line segment is computed between our team's goal and the ball. The Defender Center is given a position on this line segment at a distance that ranges between 2 and 3 meters from our goal proportional to the length of this segment. The positions of the other two Defenders are located on either side of the Defender Center.

The Midfielders are positioned to support either the Forwards or the Deferenders. When the ball is located in the opponent's half of the pitch, Midfielders are given positions near the Forwards in order to support a possible attack. When the ball is located in our half of the pitch, they are given positions in front of our defense line defined by the three Defenders.

Finally, the Goalkeeper position is determined independently to always be in the best position to stop a shot towards our goal.

11-Players Server Version (0.6.6)

Table 5.2 shows the four groups of roles in our formation for the 11-player team, the corresponding short identifier, and a short description for each role. Figure 5.10 shows

Table 5.1: Team Formation Description for the 9-Players Version

Group	Role	Description
Goalkeeper	GK	Goalkeeper
Defenders	DR	Defender Right
	DL	Defender Left
	DC	Defender Center
Midfielders	MR	Midfielder Right
	ML	Midfielder Left
Forwards	FR	Forward Right
	FL	Forward Left
	FC	Forward Center

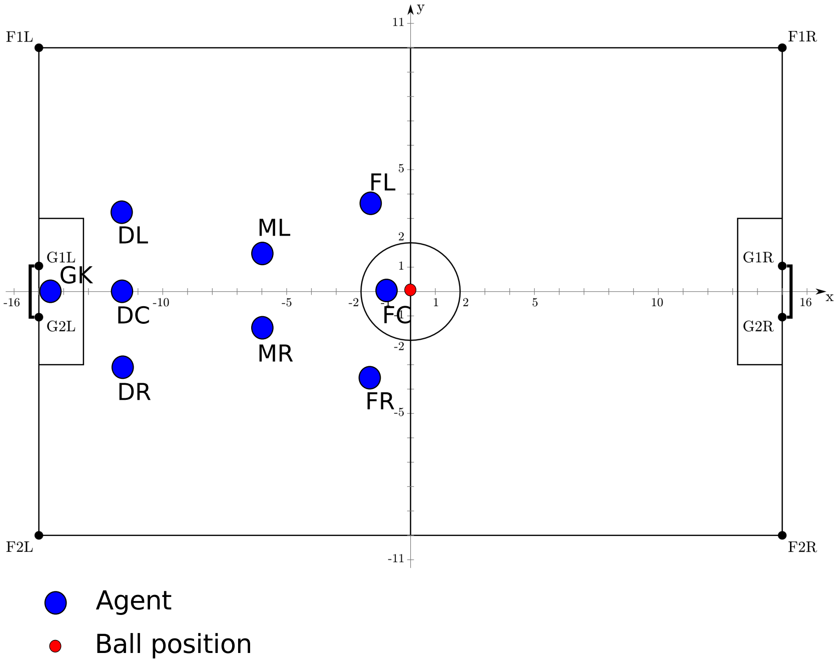
y
11
F1L
F1R
5
FL
DL
ML
2
1
G1L
GK
G1R
-16
-10
-5
-2
1
2
5
10
16
x
DC
FC
G2L
G2R
MR
-2
DR
FR
-5
F2L
F2R
-11
● Agent
● Ball position

Figure 5.9: Template of Role Positions in the Team Formation for the 9-Players Version.

how the different role positions of the formation are depicted into the soccer pitch for the newest server version in which team consists of eleven players. Again, since the formation is a function of the ball position, when the ball is located near or outside the field limits formation positions are adjusted to not exceed the field limits.

Table 5.2: Team Formation Description for the 11-Players Version

Group	**Role**	**Description**
Goalkeeper	GK	Goalkeeper
Defenders	DR	Defender Right
	DL	Defender Left
	DC	Defender Center
Midfielders	MR	Midfielder Right
	ML	Midfielder Left
	MC	Midfielder Center
Forwards	SF	Forward Support
	FR	Forward Right
	FL	Forward Left
	FC	Forward Center

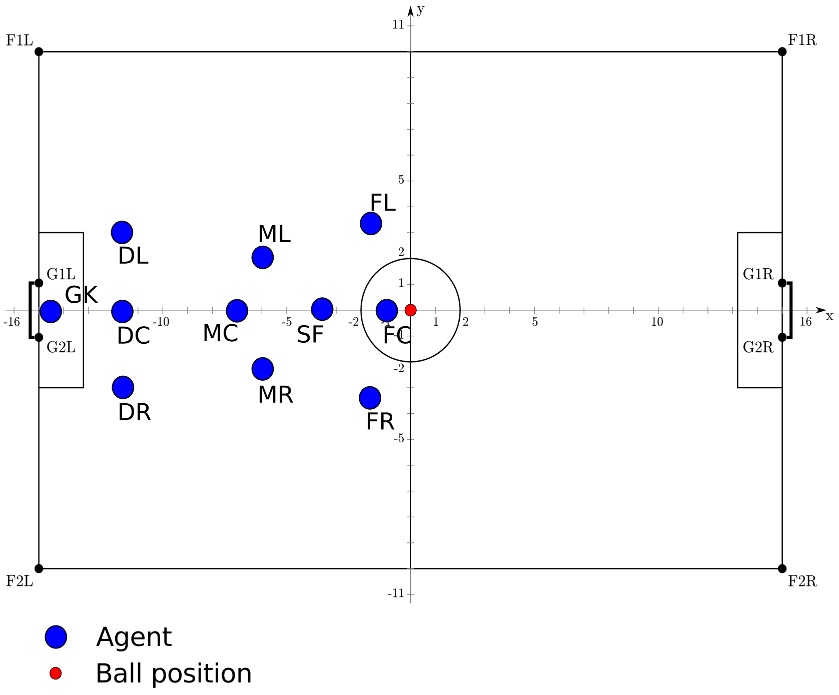
y
11
F1L
F1R
5
FL
ML
DL
2
1
G1L
G1R
GK
-16
DC
-10
MC
-5
SF
-2
FC
1
2
5
10
16
x
G2L
G2R
-2
MR
DR
FR
-5
F2L
F2R
-11
Agent
Ball position

Figure 5.10: Template of Role Positions in the Team Formation for the 11-Players Version.

The Forwards are positioned based on the same principles as the previous version's approach. The additional player, Forward Support, is positioned behind the Forward Center at a fixed distance. The additional player in the Midfielders, Midfielder Center,

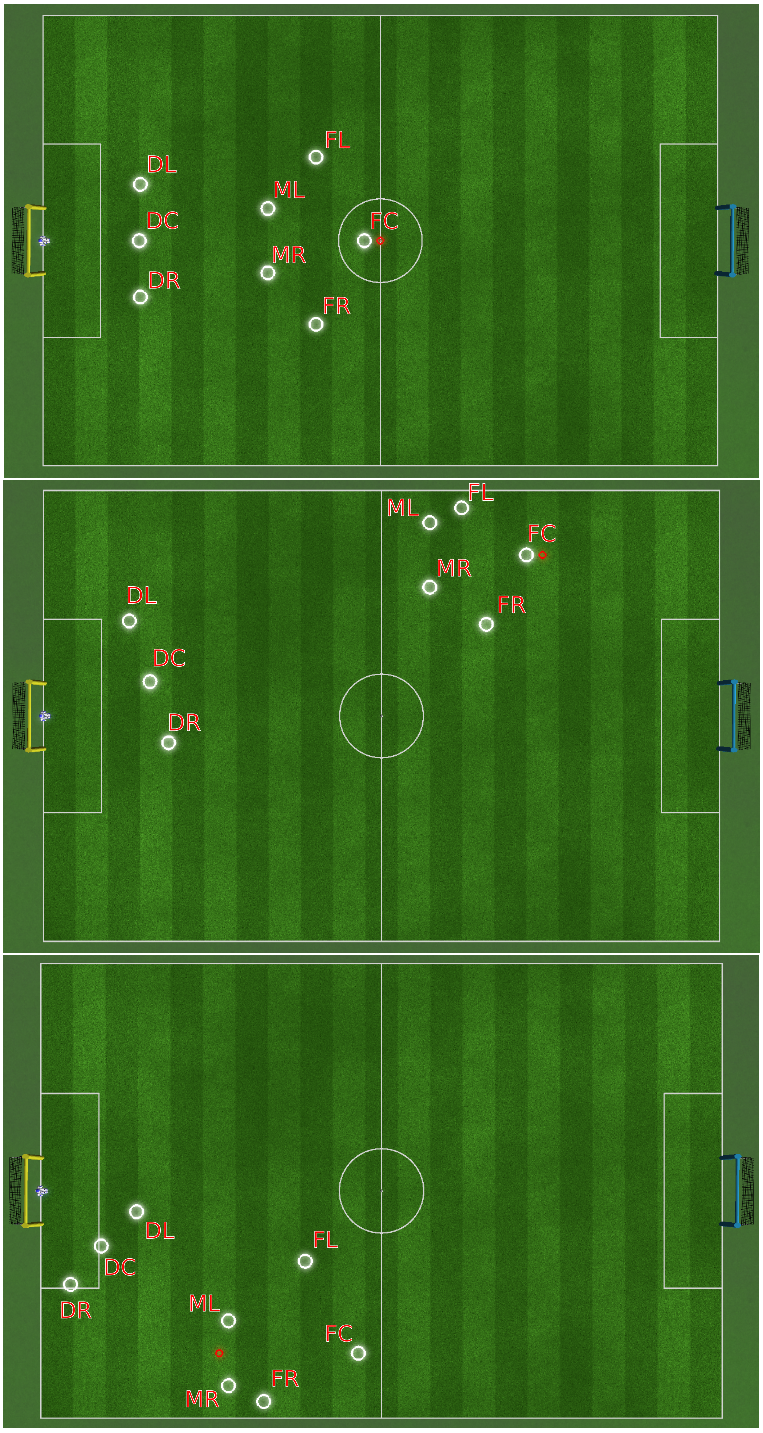
FL
DL
ML
DC
FC
MR
DR
FR
ML
FL
FC
MR
DL
FR
DC
DR
DL
FL
DC
DR
ML
FC
FR
MR

Figure 5.11: Role Positions in Team Formation for Various Ball Positions (in red).

is positioned behind the Forward Support at a fixed distance. The other two Midfielder positions are on either side of the Midfielder Center in an angle and a distance offset which are dynamically determined by the exact coordinates of the Midfielder Center. The Defense line is determined exactly as in the previous version. As before, the Goalkeeper is totally independent. Figure 5.11 presents three different team formation corresponding to three different ball positions.

5.10 Team Roles Assignment

In this section we present the role assignment function, which assigns roles to all agents. This will prove to be very helpful in the next coordination phase, when we have to find candidate positions for the support subset. Given a generated team formation we have to assign roles to the active subset. Recall that positions for the active agents are strictly tied to the position of the ball in the field. So, for any given number of active players, we choose an equal number of team formation positions which are closest to the ball and we assign the corresponding roles to the active players. The remaining team roles and their positions, in particular, will be available to the support subset as candidate positions during the support coordination process. Figure 5.12 shows how the role assignment function works. Active players will be assigned the red team roles due to the fact that they are located near to the position of the ball. Note that the roles assigned to the active players will not necessarily be the three Forwards roles. Excluding the roles bound to active players, the remaining roles marked in grey are the ones the support players will compete for. A naive role mapping function would have assigned roles statically to specific players. This approach will perform poorly in such a dynamic environment. It would also be weak in situations where an agent assigned to a defensive role may end up being beamed out of field due to a penalty without being able to exchange roles with another player who may be in a better position to defend our goal. In our dynamic approach, every role mapping is calculated with a full sense of the world state, resulting in a dynamically adaptive way of assigning roles to the agents. During testing, there were several cases in which a forward player ended up assuming a defense role at the end of the game or the opposite.

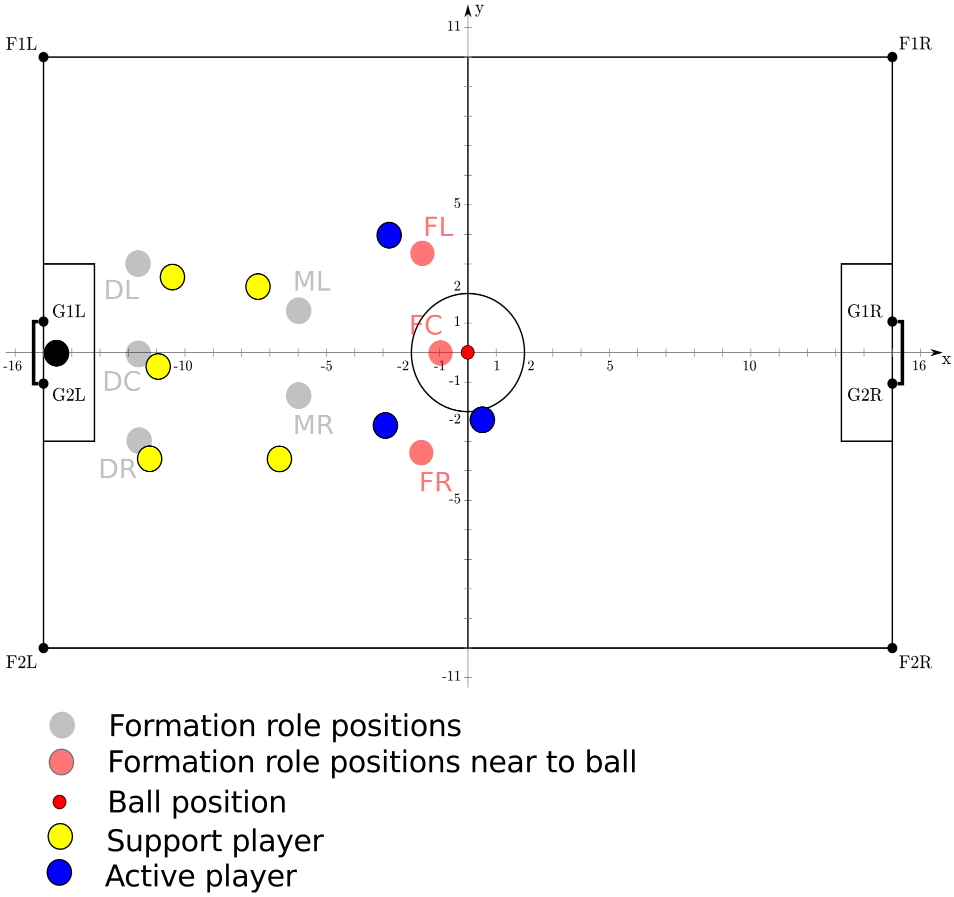
y
11
F1L
F1R
5
FL
DL
ML
2
1
FC
G1L
G1R
-16
-10
DC
-5
-2
-1
1
2
5
10
16
x
-1
G2L
G2R
-2
MR
DR
FR
-5
F2L
F2R
-11
● Formation role positions
● Formation role positions near to ball
● Ball position
● Support player
● Active player

Figure 5.12: Role Assignment Function for a Given Team Formation.

5.11 Determination of Support Positions

In this section, we are going to discuss about which roles and positions of the team formation will be assigned to the support subset. In an ideal case, there would be an equal number of support agents and roles/positions; this is true only when the inactive subset is empty. In this case, the positions for the support subset are determined automatically. In most cases, however, there are some inactive players and therefore we have to decide which positions of the team formation will be selected for the support subset. Using the position of the ball as a guide, we have to ensure that there will be positions for support players near the ball. So, for any given number of support players, we choose an equal number of team formation positions (excluding the ones given to the active players) which are closest to the ball and we assign the corresponding roles to the support players.

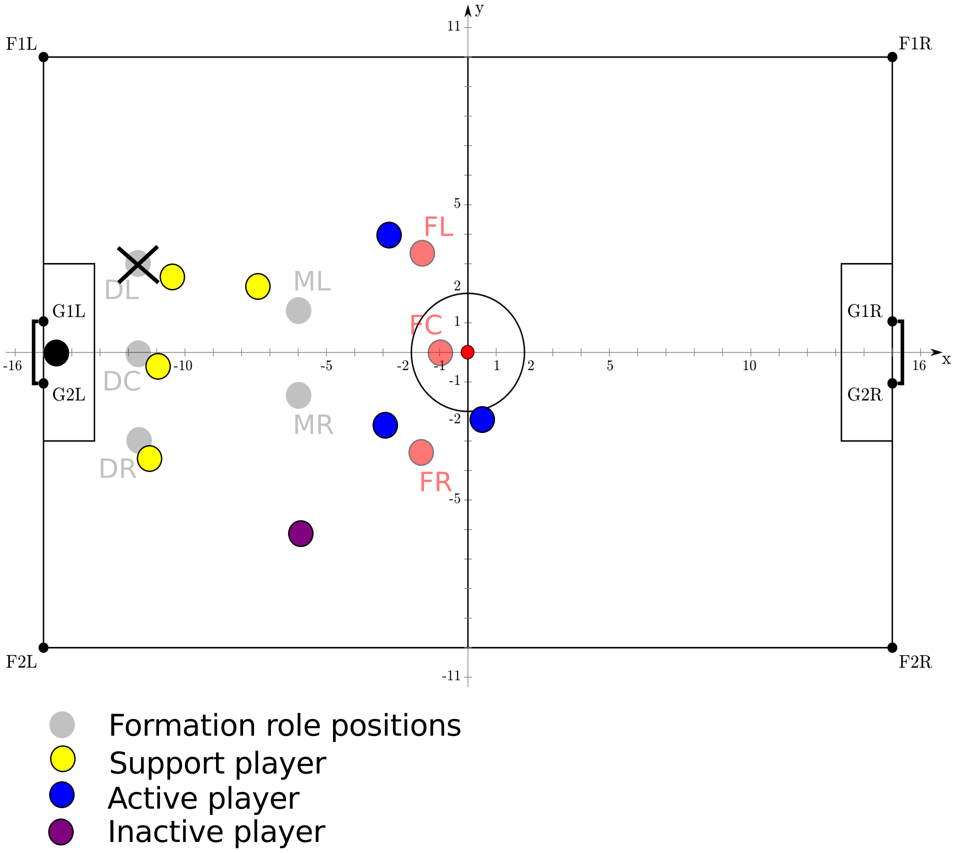
y
x
11
5
2
1
-1
-2
-5
-11
-16
-10
-5
-2
-1
1
2
5
10
16
F1L
F1R
F2L
F2R
G1L
G2L
G1R
G2R
DL
DC
DR
ML
MR
FL
FC
FR
Formation role positions
Support player
Active player
Inactive player

Figure 5.13: Determination of Support Positions from a Given Team Formation.

Figure 5.13 shows an example where there is one inactive player and therefore one role from the team formation is discarded, namely the one farthest away from the ball.

5.12 Support Players Coordination

This is the final step of the coordination process, which maps support players to support positions. In Section 5.8 we showed how the active players determine such a mapping optimally. Unfortunately, this optimal algorithm is not applicable to the support subset due to prohibitive computational costs in certain cases. Given that the size of the support subset varies between 0 and 8 or 10 (depending on the server's version and the coordination type) and considering only an equal number of support positions, in the worst case the exhaustive algorithm would have to examine between 40320 and 3628800 possible mappings. Our experience has shown that the exhaustive algorithm can work satisfactorily on support subsets of size up to 6 (720 possible mappings), assuming an equal number of candidate support positions.

Given the difficulties of applying the exhaustive algorithm to the support subset, we turned our attention to approximate, yet fast and effective, approaches in order to avoid delays in our think cycle. Our support coordination algorithm scheme is based on a method proposed by the UT Austin Villa team [1]. Their approach is based on dynamic programming and is able to compute an approximately optimal solution within the time constraints imposed by the duration of the think cycle ($\approx$ 20ms). Our version of this algorithm is shown as Algorithm 5. Each mapping is evaluated using an evaluation function that combines some of features we used in active coordination (cf. Section 5.8). More specifically, the evaluation function scores each possible support mapping using the Distance $C_{d,i}$ and Potential Collisions $C_{c,i}$ features defined for each agent i. The features described above are computed for each of the support agents, are weighted, and are summed to form the final evaluation function of a mapping:

$$SupportCost(SupportMapping) = \sum_{i=1}^{n} w_d C_{d,i} + w_c C_{c,i}$$

where (w_d, w_c) are the weights of the features, currently set at $(1, 1)$. The remaining three features from active coordination are not applicable in support coordination, because the number of positions is equal to the number of agents and these features degenerate to meaningless constants.

An example of support coordination is shown in Table 5.3. Mappings are built iteratively for position sets from $\{P_1\}$ to $\{P_1, P_2, P_3\}$ (columns from left to right) and agents sets from $\{A_1\}$ to $\{A_3\}$ and the corresponding subsets (rows from top to bottom). At each step of the algorithm, we make use of the mapping with the least cost for a subset of agents and positions (found from the previous column) which is compatible with the mapping we currently consider.

The original algorithm [1] is able to deliver an optimal solution due to a key recursive property, which states that in any complete mapping, if a lower cost mapping is found within some subset, then the cost of the complete mapping can be reduced by replacing the complete mapping within the subset with the lower cost mapping. This property holds for their evaluation function, which returns the maximum distance over all agents,

Algorithm 5 Support Players Mapping

```
Input: $SupportPlayers = \{A_1, A_2, ..., A_n\}$, $SupportPositions = \{P_1, P_2, ..., P_n\}$
Input: $OAM = OptimalActiveMapping$
Output: $BestSupportMapping$

$BestSupportMapping[s] = \emptyset$, where $s \subseteq SupportPlayers$
$BestSupportMappingCost[s] = +\infty$, where $s \subseteq SupportPlayers$
for $k = 1 \to n$ do
  for each $\alpha$ in $SupportPlayers$ do
    $S = \binom{n-1}{k-1}$ sets of $k-1$ agents from $SupportPlayers - \{\alpha\}$
    for each $s$ in $S$ do
      $SupportMapping = \{\alpha \leftarrow P_k\} \cup BestSupportMapping[s]$
      $SupportMappingCost = SupportCost(SupportMapping, OAM)$
      if $SupportMappingCost < BestSupportMappingCost[\{\alpha\} \cup s]$ then
        $BestSupportMapping[\{\alpha\} \cup s] = SupportMapping$
        $BestSupportMappingCost[\{\alpha\} \cup s] = SupportMappingCost$
      end if
    end for
  end for
end for
return $BestSupportMapping[SupportPlayers]$
```

given an assignment of positions to agents. However, this is not true for our evaluation function because of the Potential Collisions feature and therefore the best we can hope for is a good solution with no guarantees for optimality. Nevertheless, there is significant reduction in computational cost. Recall that, in the kth iteration of the algorithm, each agent will be assigned to the P_k position. The previous $k-1$ positions will be assigned to the other $n-1$ agents. These assignments result in a total of $\binom{n-1}{k-1}$ mappings to be evaluated in each iteration for each agent. Therefore, the total number of mappings considered for all n agents and n positions is:

$$n \sum_{k=1}^{n} \binom{n-1}{k-1} = n \sum_{k=0}^{n-1} \binom{n-1}{k} = n2^{n-1}$$

This represents a reduction to 1024 and 5120 mappings for 8 and 10 agents/positions

Table 5.3: Mappings Evaluated by the Support Players Mapping Algorithm.

$\{P_1\}$	$\{P_1, P_2\}$	$\{P_1, P_2, P_3\}$
$\{A_1 \leftarrow P_1\}$	$\{A_1 \leftarrow P_2\} \cup \arg\min(\{A_2\} \leftarrow \{P_1\})$	$\{A_1 \leftarrow P_3\} \cup \arg\min(\{A_2, A_3\} \leftarrow \{P_1, P_2\})$
$\{A_2 \leftarrow P_1\}$	$\{A_1 \leftarrow P_2\} \cup \arg\min(\{A_3\} \leftarrow \{P_1\})$	$\{A_2 \leftarrow P_3\} \cup \arg\min(\{A_1, A_3\} \leftarrow \{P_1, P_2\})$
$\{A_3 \leftarrow P_1\}$	$\{A_2 \leftarrow P_2\} \cup \arg\min(\{A_1\} \leftarrow \{P_1\})$	$\{A_3 \leftarrow P_3\} \cup \arg\min(\{A_1, A_2\} \leftarrow \{P_1, P_2\})$
	$\{A_2 \leftarrow P_2\} \cup \arg\min(\{A_3\} \leftarrow \{P_1\})$	
	$\{A_3 \leftarrow P_2\} \cup \arg\min(\{A_1\} \leftarrow \{P_1\})$	
	$\{A_3 \leftarrow P_2\} \cup \arg\min(\{A_2\} \leftarrow \{P_1\})$	

respectively compared to 40320 and 3628800 mappings of the exhaustive algorithm.

5.13 Goalkeeper Behavior

This section presents the behavior that leads goalkeeper to make decisions and choose actions for itself. As mentioned in Section 4.1, the goalkeeper is the only agent in our team who "runs" his own behavior. His behavior depends on a finite state machine shown in Figure 5.14. The initial state is the **Localize** state in which the goalkeeper tries to position himself between the two goal posts of the own goal facing the field. Once this is accomplished, the FSM switches to the **Guard** state in which it makes use of the Track Moving Object action on the ball to figure out its current position, direction, and speed. The goalkeeper stays in the **Guard** state as long as the ball is outside the goal area. If the ball moves within the goal area and there are no other agents near the ball, the goalkeeper switches to the **Libero** state in which it tries to clear the ball from the goal area while the coordination process switches to Support mode so that other players refrain from going to the ball. When the ball is cleared, the goalkeeper returns to the **Localize** state.

Figure 5.15 demonstrates the goalkeeper behavior in the **Guard** state. Recall that the Track Moving Object action using an egocentric reference system, marked with red dashed lines in the figure. At all times, the goalkeeper tries to compute the intersection point between its Y-axis and the grey dashed line which represents the direction of motion

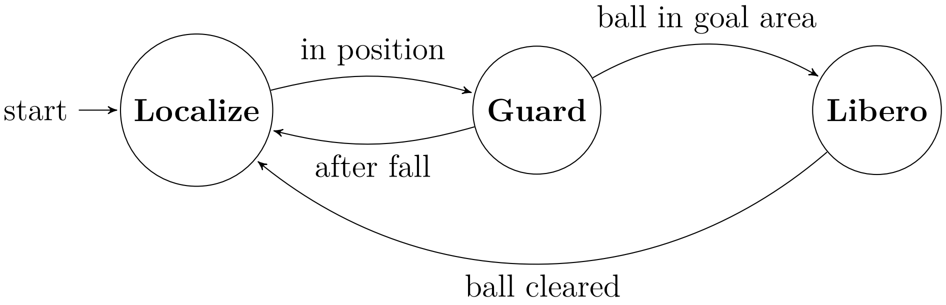
start
Localize
in position
after fall
Guard
ball in goal area
Libero
ball cleared

Figure 5.14: Finite State Machine for the Goalkeeper Behavior.

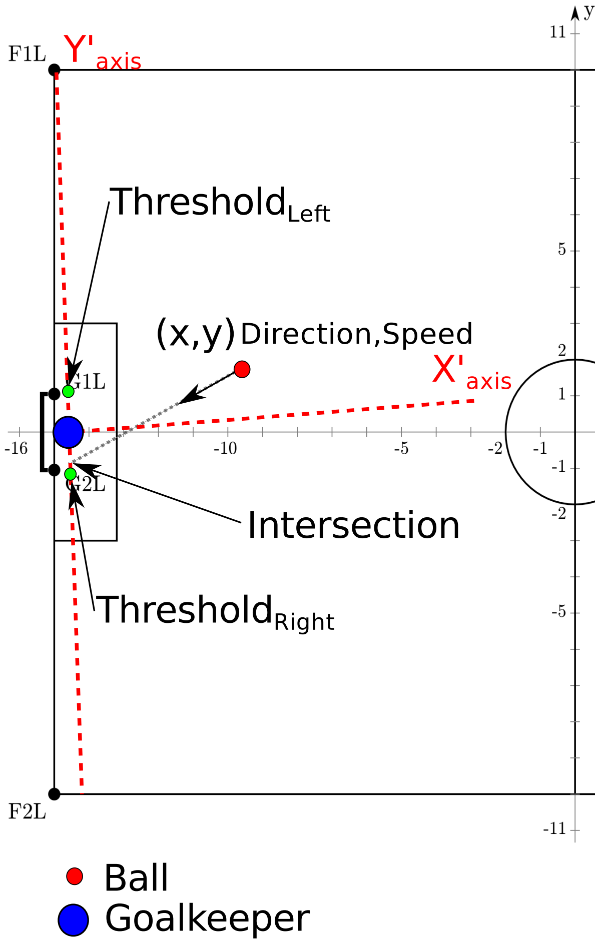
y
11
5
2
1
-1
-2
-5
-11
-16
-10
-5
-2
-1
F1L
F2L
G1L
G2L
Y'axis
X'axis
ThresholdLeft
ThresholdRight
(x,y) Direction,Speed
Intersection
Ball
Goalkeeper

Figure 5.15: Goalkeeper Behavior in the **Guard** State.

of the ball. If there is an intersection point between these two lines, the agent checks if this point falls between the two thresholds ($Threshold_{Right}$,$Threshold_{Left}$) marked in the

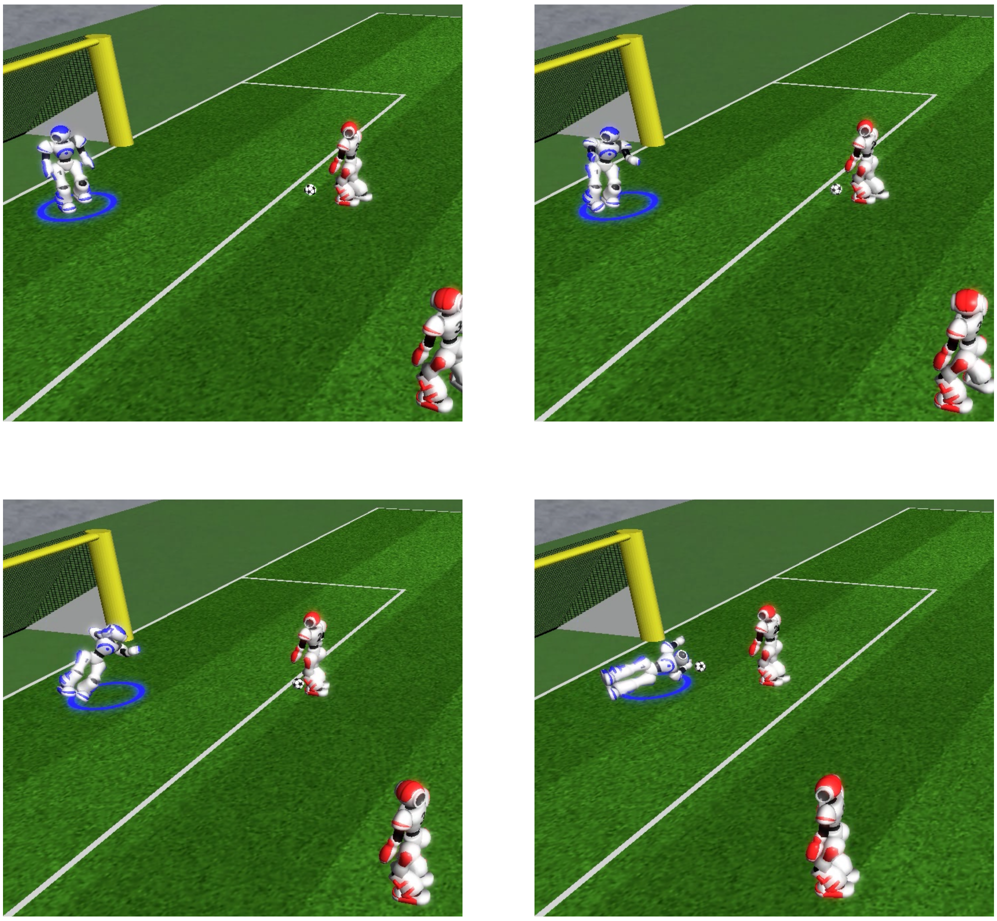

Figure 5.16: An Example of a Goalkeeper Fall to Prevent Opponents from Scoring.

figure. If yes, we are quite sure that the ball is heading towards our goal. We compute the time it will take for the ball to reach our Y-axis according to its speed and taking account the friction between the ball and the ground. If this time is less than or equal to the time it takes for our agent to fall, then the goalkeeper performs a right or left fall. An example goalkeeper fall to prevent a goal is shown in Figure 5.16.

Chapter 6

Results

In this chapter we present the results of our approach in every important part of our work. We are going to describe through examples and test results what we have achieved in the motion part, in the communication part, in the coordination part and finally and the most important the overall results which are obtained from real competitive soccer matches against other teams who have participated in at least one time at RoboCup 3D Simulation League in the past.

6.1 Motion

This section presents the improvements we have accomplished to achieve in the motion part of our agent. In general, as we described in motion Section 4.5 , motion files are generated by other teams or other platforms such as FIIT project or Webots Simulator. We can only optimize these motions through several tests. Optimization of these motions, make motion part of our player skills to reach an adequate level. Table 6.1 presents the improvements made in motions only in cases that it was possible. Optimized walk motion has reached a speed up to .45m/s which is comparable but much slower than the UT Austin Villa's walking engine which produces a walk motion of .71m/s. Furthermore strong kick movement has reached a 5.5 meters range in just 2.5 seconds of execution. Turn motion is using Webots motion files. Turning which has reached to a speed up to 30 degrees per second.

Table 6.1: Motion's Performance Improvement (Averaged Speeds and Ranges)

Motion Version	**Walk(m/s)**	**Turn(d/s)**	**Kick(m)**	**Strong Kick(m)**
Webots (Text-Based)	0.11	21	3	-
FIIT (XML)	0.22	25	3 (4 Sec.)	4 (5 Sec.)
Kouretes3D	0.45	30	3 (2.5 Sec.)	5.5 (2.5 Sec.)

6.2 Communication

Testing communication process through ideal external communication, when only our team has the ability to send messages achieved good results. Agents were able to "hear" all their teammates in an averaged 24 simulation cycles. In addition, even in real matches conditions when both teams had the ability to send messages to their teammates the results remained approximately the same. Table 6.2 presents the performance of each communication phase during coordination process. We can realize that there are not serious delays in these communication phases. This happens due to the fact soccer simulation server does not allow players to send messages in the same server cycle. We take advantage of the fact that there are separately tracked capacities for both teams, sending our messages only in labeled timed slices we ensure that opponent team will not able to block our messages even in extreme conditions when opponents shout messages constantly. The only case that our team is exposed in opponents broadcasting is when opponents choose the same time slices (cycles), as we do, to shout their messages.

Table 6.2: Communication Results in Ideal and Match Conditions

Communication Phase	**Ideal (Cycles (Sec.))**	**During Match (Cycles (Sec.))**
Init Messages	24 (0.48)	24 (0.48)
Coordination Messages	24 (0.48)	42.5 (0.85)
Action Messages	24 (0.48)	24 (0.48)

Table 6.3: Goalkeeper Averaged Results in Half-Games.

GoalKeeper Type	Goals Conceded
No Goalkeeper	7
Goalkeeper, “Empty” Behavior	7
Goalkeeper, “Full” Behavior	3

6.3 Goalkeeper

To determine his ability to stop or to delay seriously opponents from scoring, we ran several tests in three different conditions. In these three tests, players other than goalkeeper did not take part. First, we did not use a goalkeeper, allowing our opponents to reach our goal without any obstacle in their way. Opponents team managed to score an averaged of seven goals in this case. Next, we used an agent as the goalkeeper but with an “empty” behavior with which he was not able to perform any motion or to track the ball, standing useless and still at the center of our goal. Opponents team managed to score equal amount of goal with (No Goalkeeper) test. In our final test, we used a goakeeper which made use of its current behavior developed by us. We achieved in reducing conceded goal from seven to three. Table 6.3 presents these results.

6.4 Coordination

Coordination Beliefs Update

As we have mentioned before in this thesis (cf. Section 5.4), coordination beliefs are very important in order for coordinator to have an adequate knowledge of the world state. Most of functionalities of the coordination protocol are depended on these beliefs. Most important of these beliefs is the estimation about the position of the ball. Estimation derived by weighted observation sets gave nice results. Figure **??** presents the estimated position of the ball (blue circle) in several examples. Recall, that every observation set has a distance threshold; these thresholds are defined in the corresponding figures with a cyan and red circle. Observation sets depicted with a red circle imply that they have more that one observation in them. On the other hand cyan ones include only one observation. The white circles having small radii, represent individual observations coming from agents.

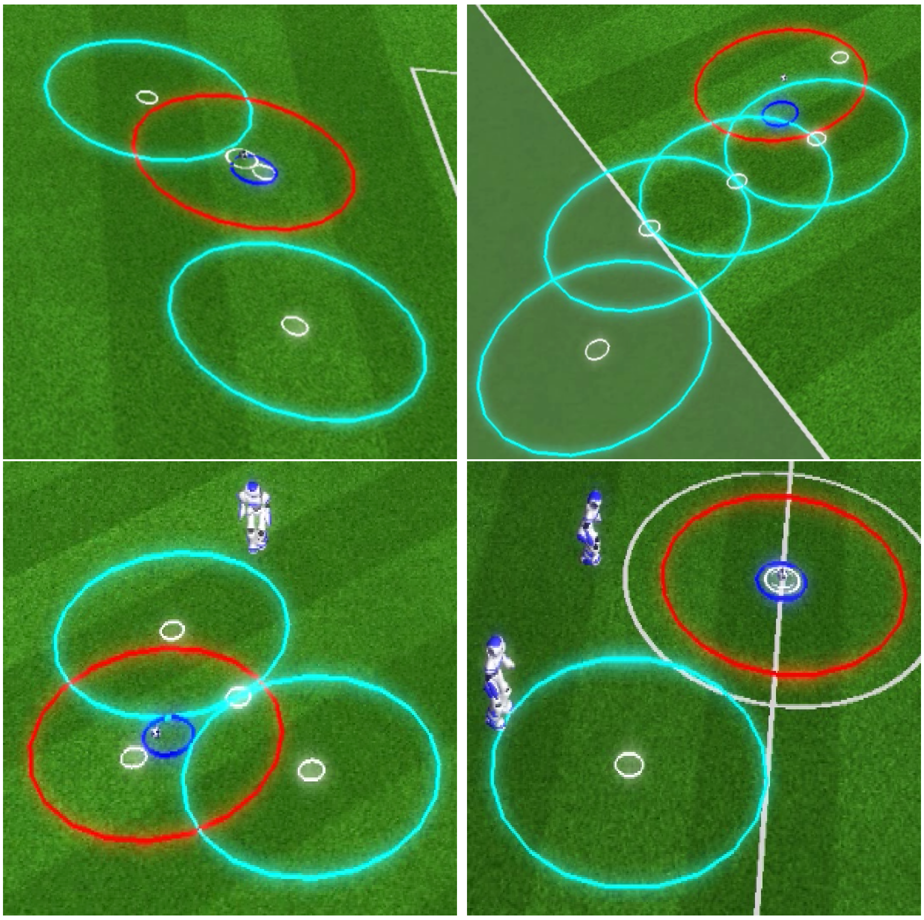

Figure 6.1: Estimated Position of the Ball in Four Examples.

Coordination Achievements

Coordination is the most significant part in this thesis. As we have presented in Chapter 5 of this thesis, coordination is responsible for assigning roles to the players, finding strategic positions, computing optimal mappings among players and positions. We have discussed every aspect of this procedure and now we present the results of this dynamic procedure. Coordination, expectedly, gave very good results. In a dynamic environment like this, in which there is a big need for dynamic coordination and cooperation among the agents we achieved to create a coordination system which has several advantages over other coordination systems with more static approaches. Some of the most important advantages of our coordination protocol is described below.

Offensive Positioning Finding and assigning worthy positions to the agents while our team is in an offensive situation was a main goal of our positioning system. In every occasion in which an agent of our team has the ball in its possession, Active agents are assigned optimal positions according to a specific evaluation method to support the on ball player. Figure 6.2 presents an example of this positioning system. As we can realize, active players are given routes to follow to be close to the opponents' goal, seeking to exploit any arising opportunity to score a goal. Figure **??** presents another example, a possible shoot by the on ball player will give active agents the chance to score a goal.

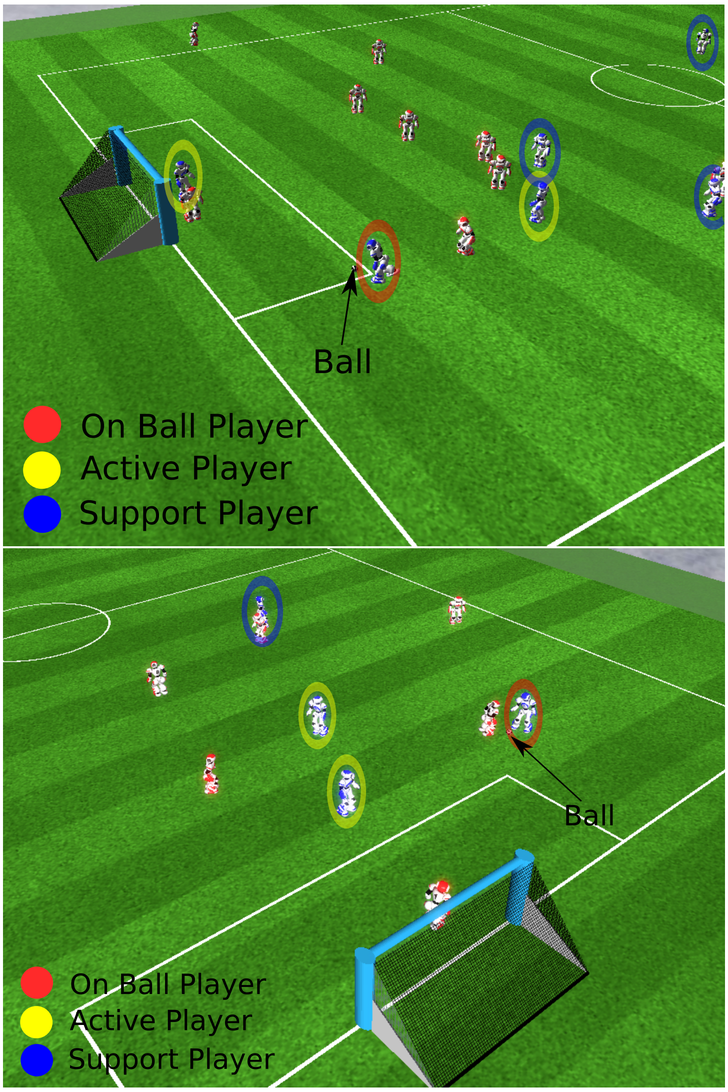
Ball
● On Ball Player
● Active Player
● Support Player
Ball
● On Ball Player
● Active Player
● Support Player

Figure 6.2: Offensive Positioning Resulting by Coordination Protocol, Example 1.

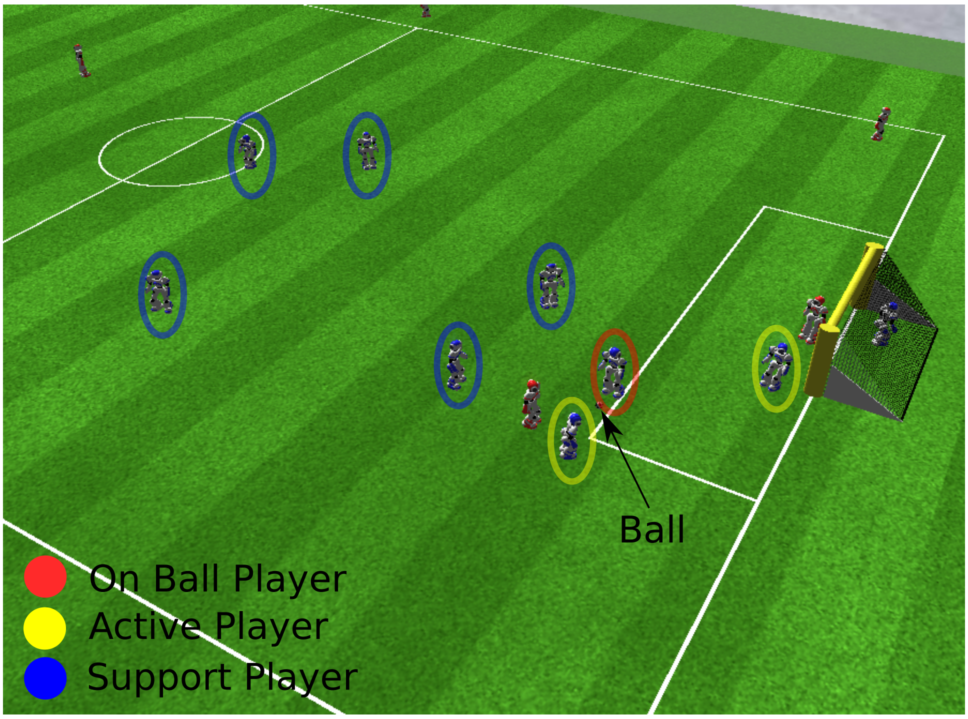

Figure 6.3: Defensive Positioning Resulting by Coordination Protocol, Example 1.

Defensive Positioning Finding and assigning positions to the agents while our team was in a defensive situation was another main target of our positioning system. In every occasion, in which our team is on a defensive role in the field, active agents are assigned worthy positions to support the on ball player and protect possible opponent's strike towards our goal. As we can realize, active players are assigned positions in the field which are proven disastrous for the opponents. Figures 6.3, 6.4 presents two examples of the defending positioning system, Active players have blocked in both cases the opponent agent's way to our goal.

Formation Consistency Dynamic role assignment was one of the most significant parts of our team coordination system. In many occasions during game-play we have seen too many times agents changing roles and duties with each other. Team coordination system was built for that, to be adaptable in different situations inside a mostly dynamic environment like robotic soccer. This function resulting our team to have an adequate role distribution in every moment within a simulation soccer game. Agents are assigned roles which are proven to be worthy and costless in

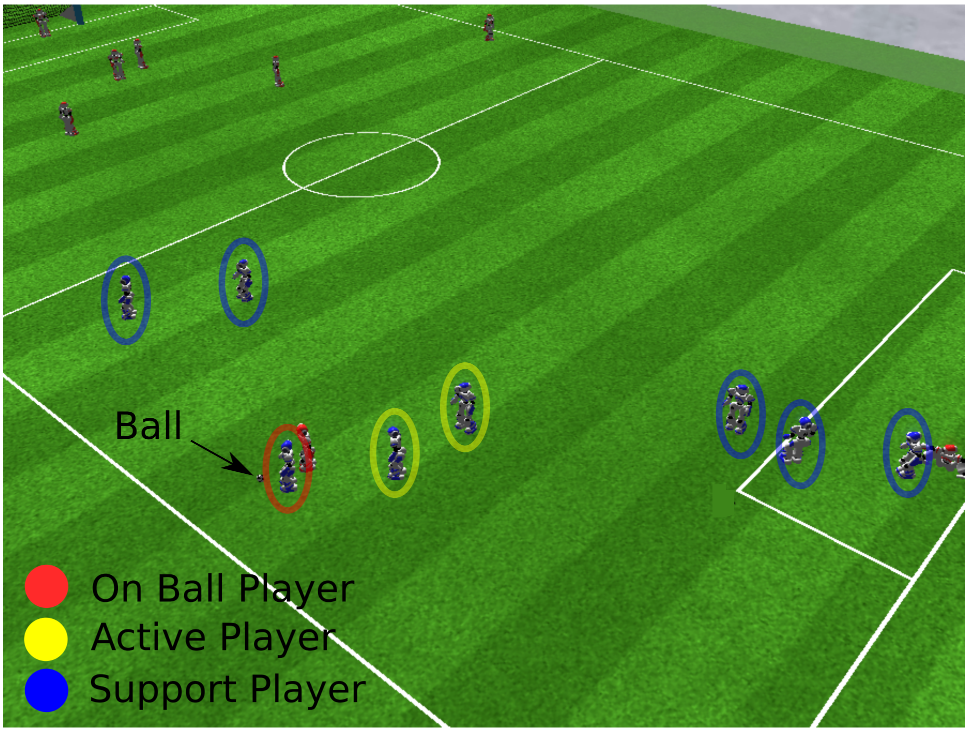

Figure 6.4: Defensive Positioning Resulting by Coordination Protocol, Example 2.

terms of the overall movement of our team into the soccer field. Figure 6.5 presents how this advantage of our coordination protocol depicted in the soccer field during game time. Active agents, as well as, on ball player are assigned specific roles from the available team formation roles. Remaining roles are available to be assigned to other players. Each one of the agents in blue circle is assigned a specific role which is described. For every given time in a soccer match our team will follow this property, resulting to a well placed team in the field, where all agents are going to have positions which are worthy and reasonable towards soccer variables.

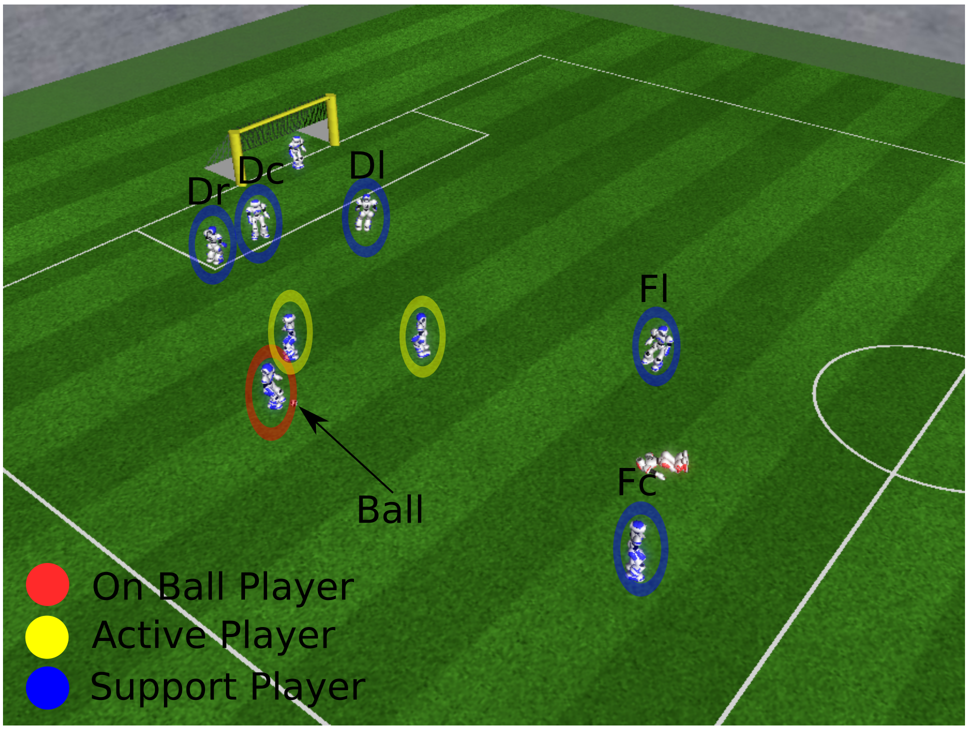
Dr
Dc
Dl
Fl
Fc
Ball
On Ball Player
Active Player
Support Player

Figure 6.5: Formation Consistency Resulting by Coordination Process Via Team Roles Assignment.

6.5 Games

Finally, in order to test our created team in the most realistic way. We decide to test our team in real conditions playing against teams that have already participated in RoboCup 3D Soccer Simulation League. We have selected nine teams from the competition held in Istanbul, 2011 and one team (MAK) from Iran open, 2011. These teams are:

RoboCanes University of Miami, USA

UT Austin Villa University of Texas at Austin, USA

NomoFC Osaka University, Japan

OxBlue University of Oxford, UK

L3MSIM Paris8 University,France

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

beeStanbul Istanbul Technical University, Turkey

Farzanegan Farzanegan high school, Iran

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

FUTK3D Fukui University of Technology, Japan

Individual Games

Playing against these teams, most of these have participated into one or more Robocup competitions, we have gained a lot of experience and we have seen how our team and especially our coordination protocol functions in different situations in a such dynamic environment. Due to lack of dynamic motions, our agents have poor movement especially in comparison with the Simulation league's best teams. However, we were able to perform well and score some goals against weaker teams of this competition.

After all these test-matches against teams who have participated into one or more Robocup competitions, we have gained a lot of experience and we have seen how our team reacts in different situations in such a dynamic environment. Due to lack of dynamic

Table 6.4: All Played Games Results

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

motions, our agents has poor movement especially in comparison with the RoboCup Simulation league's best teams. However, we were able to perform well and score some goals against weaker teams of this competition.

All executable binaries from teams are from previous events binaries, [Available online: `http://simspark.sourceforge.net/binaries/RoboCup2011/`]. All games had 10 minutes duration, same as real competition matches in official Robocup competition. Server and monitor were running in the same machine[2]. Each one of the teams binaries was running in a separate machine[3]. Table 6.4 presents the results between our team and opponent teams. There is information about the average goal difference, the number of wins, the number of draws, the number of loses, and the total number of games we played with each team.

---

[1] AGD: Averaged Goal Difference

[2] **Server**: Intel Core 2 Duo 3.16 Ghz, 5.8GiB Ram

[3] **Client1**: Intel Core 2 Duo 1.86 Ghz, 2GiB Ram

[3] **Client2**: Intel Quad Core i5 3.3 Ghz,4GiB Ram

Table 6.5: Mini Tournament

Team	P	W	D	L	F	A	Pts
Kouretes3D	**8**	**2**	**6**	**0**	**2**	**0**	**12**
Farzanegan	8	1	7	0	1	0	10
Rail	8	1	6	1	1	1	9
L3M-SIM	8	1	5	2	1	2	8
NomoFC	8	1	4	3	1	3	7

Mini Tournament

Finally, we held a mini tournament in which teams participated in it were selected in order to have a competitive and in equal terms competition. Table 6.5 presents the result table of this tournament. Win: 3 points, draw: 1 points, loss: 0 points.

Chapter 7

Related Work

7.1 UT Austin Villa

UT Austin Villa [9] is the most known and the best team which is participating in the Robocup's simulation league. Their first appearance was in the Robocup 2007 held in Atlanta, U.S.A, in July 2007. UT Austin Villa belongs to University of Texas and consists of five members, professor Peter Stone, graduate students Patrick MacAlpine and Samuel Barrett and finally two undergraduate students Nick Collins and Adrian Lopez-Mobilia. The main characteristic of this team is its “state-of-art” omni-directional walking engine. Its fast and stable walk is recognizable and offers them great movement. A typical example of this great team's results can be that in Robocup's competition in Istanbul 2011 this team won all 24 games it played and scored a total of 136 goals without conceding any.

In their last paper about positioning [1], they explained an approach of player positioning in the field. First, a full team formation is computed. Second, each agent computes the best assignment between agents and team formation positions according to its belief about the world state. Finally, a coordination mechanism is used to choose among all players' suggestions. This coordination mechanism uses a voting system. Most voted mapping will be used.

I am not the most appropriate person to judge their longterm work and contribution to the Robocup's simulation league, as I deal with this league only for a few months. However, I would like to mention that in our approach there is a major difference in the way that players coordinate their actions. The separation of the team into subsets makes it easier for us to solve all these problems caused due to complexity constraints. Furthermore, we are using active's group players in order to have a more dynamically position assignment close to the position of the ball. Finally, I wish we had such a perfect locomotion system as UT Austin Villa. It would be a nice challenge to compare these two dynamic coordination systems in the same levels of motion's skills.

7.2 BeeStanbul

The beeStanbul project [10] from the Artificial Intelligence and Robotics laboratory (AIR lab) at Istanbul Technical University (ITU) is the first initiative from ITU to participate in RoboCup competitions. It consists of five members and has been participating in the Robocup's competitions since RoboCup 2010 held in Singapore. It is a nice team which accomplished to qualify up to second round in the last Robocup competition in Mexico 2012. First, they are making use of both static and dynamic movements and their walking machine is more than adequate. Concentrating in their work at the coordination part of their project.They split team agents into three groups, defenders and attackers. The attackers group involves the forward and the midfielder agents while the defenders group involves only the defender agents.Since two agents are assigned to the goalkeeper and the forward roles, the remaining seven agents are to be assigned to these roles. This is accomplished by a distributed Voronoi cell construction approach in which each agent calculates its own cell independently from that of the others. Therefore, every agent has a differently shaped cell and these can overlap. The time complexity of the method is $O(n^2)$ where n is the number of agents in the team. After constructing the cell for itself, each agent determines the center of the cell as its new target. Agents become closer to each other by using this strategy. In their approach, only teammates in the viewpoint of the agent are considered.

We can realize that there can be situations in soccer games, in where each agent who computes his own cell could be completely unaware if none of its teammates is located in the field of its view. Having a better knowledge of teammates position in the soccer field is a key feature in our coordination protocol.

7.3 FUT-K_3D

FUT-K_3D [12] that is mainly composed of undergraduate students of Fukui University of Technology in Japan has been organized since fall 2007. In their team description paper for RoboCup 2011 competition, they presented coordinated motion by communication protocol and an implementation of probabilistic behavior selection. Communication algorithm is inspired by a token passing mechanism of access control method for network systems. An agent tries to broadcast its message to the others. Then, agent waits for a confirmation to stop broadcasting this procedure continues with the next agent. In coordination part of their paper, the nearest agent from the ball performs to approach to the ball, and if that keeps the ball, it begins the movement of dribbling or kicks the ball. Other agents begin the coordinated motion keeping a formation with each other. In addition, if the agent keeping the ball goes into other agent'As location, the rest starts the position change corresponding to the location on agent keeping the ball. Their behavior selection policy is based on a stochastic model called "Probabilistic Behavior Selection" that even if they encounter the identical situation, they stochastically select one action from multiple pre-defined behavior. They assume the probabilities of each behavior each game. In addition, depending on the result of Probabilistic Behavior Selection, they also consider a method to update action probabilities during the game.

7.4 Farzanegan

Farzanegan [13] Highschool Laboratory has been working on Robocup science for many years. In their last team description paper for RoboCup 2011, they describe their work about team coordination. They presented an approach about multi-agent collaboration, based on strategical positions, roles and responsibilities. Also like human soccer, each agent has a strategical position that defines its default position and movement range inside the soccer field. So they have classified their strategical position into two categories: initial and game-play. Strategical positioning, roles and responsibilities are inevitable in soccer domain. Each agent has its own movement range based on its role and responsibilities. When keeping an eye on the ball, the movement range will determine whether the agent should go for the ball, or leave it to its team mates.

As we can realize, a static approach for the team coordination in such a dynamic environment.

Chapter 8

Conclusion

We have presented a team's framework for the Robocup 3D Simulation League - a physically realistic environment that is partially observable, non-deterministic, noisy and dynamic, as well as a dynamic coordination system which evaluates all the necessary world state variables and be executed only by one agent -independent software process. Creating a team's framework from scratch was a big challenge especially when this project does not depend in any other external software except from motions files and the dynamic programming implementation created by UT Austin Villa.

8.1 Future Work

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

Probabilistic Localization System

The robot localization problem is a key problem in making truly autonomous robots. If a robot does not know where it is, it can be difficult to determine what to do next. In our approach, there is an adequate localization scheme which can be improved further. Some of our work and especially coordination protocol requires a more probabilistic localization scheme.

Dynamic Omni-Directional Movement

Most teams which have been participating into the RoboCup 3D Simulation League make use of dynamic movement. This is a major drawback for our side and I really hope this issue to be resolved in the near future. Team coordination will operate even more efficiently since the faster movement of our agents will give even more dynamically consistent results.

Passing

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

Testing and Debugging in New Server's version

There are things to be tested in order our team to meet the standards of the new server's version 0.6.6 in which there are some changes with most important that there are now eleven players for each side and field's size has changed.

Participation in Robocup

Robocup is a well-known competition especially for people who are interested in robotic soccer and artificial intelligence in general. It is not going to be easy for our team to be competitive at once but it will be a nice experience. Furthermore, we are going to have the opportunity to test our agent in real competition matches.

References

[1] MacAlpine, P., Barrera, F., Stone, P.: Positioning to win: A dynamic role assignment and formation positioning system. In: Proceedings of the RoboCup International Symposium. (2012) 75, 93

[2] Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., Osawa, E., Matsubara, H.: Robocup: A challenge problem for AI. AI Magazine **18**(1) (1997) 73–85 3

[3] Robocup: Soccer simulation league wiki Only available online: http://wiki.robocup.org/wiki/Soccer_Simulation_League. 13

[4] SimSpark: Wiki Only available online: http://simspark.sourceforge.net/wiki. 13

[5] Abeyruwan, S., Seekircher, A., Stoecker, J., Visser, D.U.: Roboviz monitor Only available online: https://sites.google.com/site/umroboviz/. 17

[6] Papadimitriou, V.: Localization in simspark environment (2012) Autonomous Agents Fall Semester, Only available online: http://www.intelligence.tuc.gr/~robots/ARCHIVE/2011w/projects/TUCagent3D/home.html. 28

[7] FIIT: Robocup 3d Slovak University of Technology in Bratislava, Only available online: http://fiitrobocup3d.sourceforge.net/. 34

[8] Michel, O.: Webots: Professional mobile robot simulation. Journal of Advanced Robotics Systems **1**(1) (2004) 39–42 37

[9] UTAustinVilla: Utaustinvilla Only available online: http://www.cs.utexas.edu/~AustinVilla/sim/3dsimulation/. 93

[10] Demirdelen, B., Toku, B., Ulusoy, O., Sonmez, T., Ayvaz, K., Senyurek, E., Sariel-Talay, S.: beestanbul robocup 3d simulation league team description paper (2012) Only available online: http://air.cs.itu.edu.tr/publications-1/beeStanbul_TDP2012.pdf. 94

[11] Mohammadi, N., Yassari, M., Zahiri, S.A., Salehi, M.: Kaveh robocup 3d simulation league team description paper (2011) Only available online: http://hedayat.fedorapeople.org/misc/rc2011tdps/.

[12] Yashiki, M., Miyajima, K., Sugihara, K., Ohkuma, K., Yamanishi, T.: Fut-k_3d robocup 3d simulation league team description paper (2011) Only available online: http://hedayat.fedorapeople.org/misc/rc2011tdps/. 95

[13] Mousaviyan, A.S., No, S.J., Davari, A.S.A.M., Makki, F.S., Dastserri, N.S.: Farzanegan robocup 3d simulation league team description paper (2011) Only available online: http://hedayat.fedorapeople.org/misc/rc2011tdps/. 95