Technical University of Crete

School of Electronic and Computer Engineering

Dynamic Multi-Robot Coordination for the RoboCup Standard Platform League

Evangelos Michelioudakis

Chania, September 2013

Πολυτεχνειο Κρητης

Σχολη Ηλεκτρονικων Μηχανικων και Μηχανικων Υπολογιστων

Δυναμικός Πολυρομποτικός Συντονισμός για το Πρωτάθλημα **Standard Platform** του **RoboCup**

Ευάγγελος Μιχελιουδάκης

Χανιά, Σεπτέμβριος 2013

”The question of whether computers can think is like the question of whether submarines can swim.” (Edsger W. Dijkstra)

Abstract

Coordination among robots is a popular research topic in robotics, artificial intelligence, and multi-agent systems. Robotic soccer, known as RoboCup, represents a complex, stochastic, real-time, multi-agent competitive domain. In such domains, cooperative strategy is required for the success of the team's mission, thus the ability of team coordination is crucial. Team coordination in RoboCup offer mechanisms for player positioning and role assignment. This thesis studies the problem of coordination for a robotic soccer team and offers a real-time module for on-board execution on Nao robots that achieves dynamic positioning and role assignment using the basic principles of utility theory. Specifically, a formation generator component is responsible for generating a set of candidate positions in the field, based on the global estimated ball position in the field, as provided by the shared estimated global game state. The formation type (offensive or defensive) is determined dynamically, depending on which half of the field the ball lies in, and the candidate positions represent certain roles (supporter, attacker, defender, etc.). Then, a role assignment component uses a team utility function to evaluate mappings of players (robots) to positions (roles) and decides which mapping is best for the current game situation. The utility function combines a variety of features to characterize good aspects of teamwork. Since the candidate positions/roles may be more than the robots, the number of possible mappings can be extremely large. Two role assignment algorithms are proposed; one based on exhaustive search, which guarantees optimal solution albeit with high computational cost, and another based on stochastic local search (Particle Swarm Optimization), which cannot guarantee an optimal solution, but comes with very low computational cost. The selected mapping is adopted by all robots and each robot executes the behavior that implements the assigned role. Re-coordination takes place whenever a significant change in the game state occurs. Our coordination module has been integrated into the software architecture of our RoboCup team "Kouretes" and is used for coordinating the team robots during RoboCup games.

Περίληψη

Ο πολυρομποτικός συντονισμός είναι ένα δημοφιλές θέμα έρευνας στη ρομποτική, την τεχνητή νοημοσύνη και τα πολυπρακτορικά συστήματα. Το ρομποτικό ποδόσφαιρο, γνωστό και ως RoboCup, αποτελεί ένα περίπλοκο, στοχαστικό, πολυπρακτορικό και ανταγωνιστικό περιβάλλον πραγματικού χρόνου. Σε τέτοιου είδους περιβάλλοντα, η στρατηγική συνεργασίας είναι απαραίτητη και συνεπώς ζωτικής σημασίας για την αποτελεσματικότητα της ομάδας. Ο συντονισμός στο διαγωνισμό RoboCup προσφέρει μηχανισμούς για την τοποθέτηση της ομάδας στο γήπεδο, καθώς και την ανάθεση ρόλων. Η παρούσα διπλωματική εργασία μελετά το πρόβλημα του συντονισμού για μια ρομποτική ποδοσφαιρική ομάδα και προσφέρει λογισμικό, κατάλληλο για εκτέλεση πραγματικού χρόνου πάνω σε ρομπότ Nao, που επιτυγχάνει δυναμική τοποθέτηση και ανάθεση ρόλων χρησιμοποιώντας τις βασικές αρχές της θεωρίας χρησιμότητας. Πιο συγκεκριμένα, ένας μηχανισμός παραγωγής συστημάτων είναι υπεύθυνος για την δημιουργία ενός συνόλου θέσεων στο γήπεδο, με βάση την καθολικά εκτιμώμενη θέση της μπάλας, όπως προβλέπεται από το κοινό μοντέλο καθολικής κατάστασης του παιχνιδιού. Ο τύπος του συστήματος (επιθετικό ή αμυντικό) καθορίζεται δυναμικά, ανάλογα σε ποιο μισό του γηπέδου βρίσκεται η μπάλα και οι υποψήφιες θέσεις αντιπροσωπεύουν ειδικούς ρόλους (υποστηρικτής, επιθετικός, αμυντικός...). Στην συνέχεια, ο μηχανισμός ανάθεσης ρόλων χρησιμοποιεί μια συνάρτηση χρησιμότητας για την αξιολόγηση αντιστοιχίσεων των παικτών (ρομπότ) σε θέσεις (ρόλοι) και αποφασίζει ποια αντιστοίχιση είναι καλύτερη για την τρέχουσα κατάσταση του παιχνιδιού. Η συνάρτηση χρησιμότητας συνδυάζει πολλά κριτήρια για να χαρακτηρίσει θετικά στοιχεία της ομαδικής εργασίας. Δεδομένου ότι οι υποψήφιες θέσεις/ρόλοι μπορεί να είναι περισσότερες από τα ρομπότ, ο αριθμός των πιθανών αντιστοιχίσεων μπορεί να γίνει εξαιρετικά μεγάλος. Δύο αλγόριθμοι προτείνονται για την ανάθεση ρόλων, ένας που βασίζεται στην εξαντλητική αναζήτηση και εγγυάται βέλτιστη λύση, ωστόσο με μεγάλο υπολογιστικό κόστος, και ένας δεύτερος βασιζόμενος σε στοχαστική τοπική αναζήτηση (Particle Swarm Optimization), ο οποίος δεν εγγυάται βέλτιστη λύση, αλλά έχει πολύ χαμηλό υπολογιστικό κόστος. Η επιλεγμένη αντιστοίχιση υιοθετείται από όλα τα ρομπότ και κάθε ρομπότ είναι υπεύθυνο να εκτελέσει την συμπεριφορά που υλοποιεί τον ρόλο που του ανατέθηκε. Ο συντονισμός λαμβάνει χώρα κάθε φορά που υπάρχει κάποια σημαντική αλλαγή στην κατάσταση του παιχνιδιού. Οι παραπάνω μηχανισμοί έχουν ενσωματωθεί στην αρχιτεκτονική λογισμικού της ομάδας RoboCup «Κουρήτες» και χρησιμοποιούνται για τον συντονισμό των ρομπότ της ομάδας σε αγώνες RoboCup.

Acknowledgements

First of all, I would like to thank my advisor Michail G. Lagoudakis for his inspiration, guidance and the trust that he showed in me.

I would also like to thank Team Kouretes (N. Kofinas, N. Pavlakis, N. Kargas, S. Piperakis) and Manolis Orf for their great help and suggestions during the implementation of my work. We had a lot of fun at our Lab as well as at the RoboCup events.

Next, I would like to thank my friends for all these great years and moments we had. I am lucky to have met you G. Liverios, A. Mavrommatis, S. Maropaki, K. Pechlivanis, S. Sourlantzis, G. Tabakakis, A. Chorianopoulou. (The names are arranged in alphabetical order so that nobody complains.)

I would also like to thank my childhood friends, which are like brothers to me, A. Armaos, P. Kazatzas, A. Kechribaris, N. Polyzos, S. Christinakis.

Last, but most important, I would like to thank my parents for everything they have done and dedicate my thesis to them.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

The RoboCup Competition is an international annual aggregation of robotic competitions which intends to promote Robotics and Artificial Intelligence (AI) research. RoboCup Soccer constitutes the main RoboCup division and focuses on the game of soccer. The research goals in RoboCup Soccer concerns multi-robot and multi-agent systems in dynamic adversarial environments and all the participating teams have to find real-time solutions to some of the most difficult problems in robotics, such as perception, cognition, action, and coordination. In the Standard Platform League (SPL) all teams use identical robots (standard platform), therefore they are not concerned with hardware improvements, they only concentrate on software development. Currently, the chosen SPL platform is the Aldebaran Nao humanoid robot.

Software development for robots competing in the RoboCup SPL essentially aims at developing autonomous agents. An autonomous robotic agent is a system that continuously perceives its environment through the robotic sensors, analyzes the percept sequence using various AI techniques, and takes actions through the robotic actuators with the goal of maximizing a utility function. The central problems of an autonomous robotic agent include environment perception, robot localization, robotic mapping, path planning, decision making under uncertainty, and learning. Apart from those, the central problems of a team of autonomous agents working to achieve a common goal include multi-agent planning, team coordination, and collaborative decision making. This thesis studies the problem of team coordination in order for the robotic soccer team to achieve efficient positioning in the field depending on the current game situation and make collaborative decisions about role assignment.

To illustrate the importance of team coordination and planning, consider the following scenario. Imagine a team of players during a moment of the game in which each player is observing the ball, but they have no way of coordinating their actions towards a common goal. This common goal of the team is typically to score against the other team, while simultaneously defending their own goalpost. Therefore, without coordination each player can only assume a predefined role, either defend the own goalpost or approach the ball and attack regardless of the other aspects of the current game situation (proximity to the ball, players' locations in the field, missing/penalized players, etc.). Clearly, the outcome of such predefined strategies is suboptimal; the team may be left with no attackers or no defenders or several players may conflict each other by attacking at the same time. In this scenario the team lacks dynamic soccer strategy, such as sharing perceptual information, coordinating in real time, and assigning roles dynamically to achieve the desirable team behavior suitable for the current game situation. This thesis aims at addressing exactly this problem, namely dynamic multi-robot coordination which enables the possibility for more efficient game playing.

1.1 Thesis Contribution

This thesis contributes the development of a mechanism that addresses the problem of team coordination and planning, during a robotic soccer game, using dynamic positioning and role assignment. More specifically, this mechanism takes the estimated global game state shared among the robots as input and produces an (optimal) position/role for each player/robot according to a certain team utility function. After coordination, each player/robot executes the assigned role, which may include positioning in the field, approaching the ball, or any other functionality described in the role, until a significant change in the game situation occurs or a certain amount of time has passed, at which point the team coordinates again.

In order for the above mechanism to provide the position/role for each player/robot the shared belief about the global game state is required. This shared belief may include information about the location of the ball in the field, number of active/penalized players, location of teammates in the field, opponent locations, etc. Our approach consists of two major components: formation generator and role assignment. The formation generator component is responsible for generating a set of promising candidate positions on the field, based on the global ball position as provided by the shared belief. The role assignment component produces and evaluates possible mappings of players/robots to positions/roles using a custom-designed utility function and decides which mapping is the best. The chosen mapping is shared among all teammates and the corresponding roles are subsequently executed. Our approach is implemented in a distributed way, so that coordination can be achieved by any combination of active and penalized players.

The benefits of the proposed method are numerous, yet the most important one is that the robots can collaborate dynamically as a team in order to maximize their game performance, instead of relying on independent predefined behaviors. Moreover our design is modular, in the sense that formations are dynamically generated and are independent from role assignment. This way we can easily change the team's formation strategy just by replacing that component or even have multiple formation generators which are used at different times during a game. Respectively, the evaluation of the possible mappings can be changed by replacing the utility function. Furthermore, the functionality of the roles can be changed by simply modifying the corresponding individual robot behavior.

1.2 Thesis Outline

Chapter 2 describes the RoboCup competition, the Standard Platform League (SPL), the Aldebaran Nao humanoid robot, our SPL team Kouretes, our software architecture Monas, and our communication framework Narukom. Furthermore, it provides basic background information about Utility Theory and the Particle Swarm Optimization (PSO) algorithm that are being used in this approach. In Chapter 3 we define the problem of team coordination and we discuss the significance of developing an efficient and effective mechanism for addressing the problem in real-time. Additionally, we briefly review related work by other RoboCup teams. In Chapter 4 we describe our approach in detail, separating the problem in formation generation, definition of utility, and role assignment. In Chapter 5 we briefly present our implementation within the software framework of Team Kouretes, including our choices for real-time, on-board, asynchronous execution. In Chapter 6 we present a couple of scenarios demonstrating the effectiveness and the efficiency of our team coordination mechanism. Finally, in Chapter 7 we propose directions for future work and conclude.

Chapter 2

Background

2.1 RoboCup

RoboCup, an abbreviation of “Robot Soccer World Cup”, is an international annual competition which intends to promote robotics and artificial intelligence research. The founding father of RoboCup, Professor Alan Mackworth, inspired the idea of building a robot to play a soccer game autonomously in 1992. One year later, Hiroaki Kitano [2] and his research group decided to launch a novel robotic competition. Finally, in 1997 the actual establishment of the International RoboCup Federation occurred. The ambitious goal of the RoboCup Initiative is stated as follows:

> “By mid-21st century, a team of fully autonomous humanoid robot soccer players shall win the soccer game, complying with the official rule of the FIFA, against the winner of the most recent World Cup.”

All the teams participating in RoboCup have to find real-time solutions to some of the most difficult problems in robotics (perception, cognition, action, coordination) and apply their approaches on the various leagues of the four RoboCup divisions (RoboCup Soccer, RoboCup Rescue, RoboCup@Home, Robocup Junior). Until today, noteworthy progress has been made in advancing the state-of-the-art technology, while the number of the participating researchers who aim to fulfill the initial challenge is constantly growing.

2.1.1 Standard Platform League

RoboCup Soccer constitutes one of the four RoboCup divisions and focuses mainly on the game of soccer, where the research goals concern cooperative multi-robot and multi-agent systems in dynamic adversarial environments. All robots in this division are fully autonomous. RoboCup Soccer consists of five different leagues (Humanoid, Middle Size, Simulation, Small Size, and Standard Platform). In the Standard Platform League (SPL) all teams use identical robots (standard platform). Currently, the chosen SPL platform is the Aldebaran Nao humanoid robot, therefore the teams concentrate only on software development. The participating teams are prohibited to make any changes to the hardware of the robot, meaning that off-board sensing or processing systems are not allowed. The use of directional, as opposed to omnidirectional, vision forces a trade-off of vision resources between self-localization, ball localization, player identification, and obstacle detection. The robots are completely autonomous and no human intervention from team members is allowed during the games. The only interaction of the robots with the "outer human world" is the reception of data from the Game Controller, a computer that broadcasts information about the state of the game (score, time, penalties, etc.).

The SPL games as of 2013 are conducted on a $9m \times 6m$ soccer field which consists of a green carpet marked with white lines and two yellow goals (Figure 2.1). The ball is an orange street hockey ball. Each team consists of five robots, one goal keeper, and four field players. The robot players are distinguished by colored jersey shirts, blue for one team and red for the other. The total game time is 20 minutes divided in two halves; each half lasts 10 minutes. During the 10-minutes half-time break, teams have to switch field sides and jerseys and only during this time is it permitted to change robots, change programs, etc. The complete rules of the SPL games are stated in detail in the RoboCup Standard Platform League (Nao) Rule Book [1], which is annually updated with enhancements and additional challenging requirements that propel the general progress of the league.

2.1.2 Aldebaran Nao Humanoid Robot

The current hardware platform which all SPL teams are obliged to work with is Nao, an integrated, programmable, medium-sized humanoid robot developed by Aldebaran Robotics in Paris, France. Project Nao [3] started in 2004. In August 2007 Nao officially

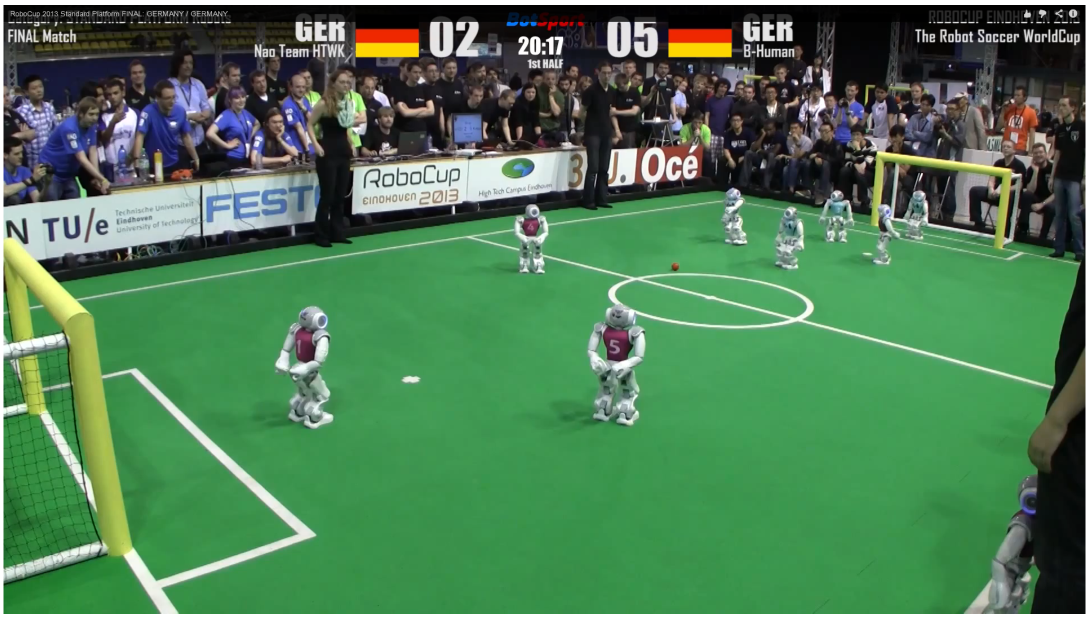

Figure 2.1: Standard Platform League at RoboCup 2013 in Eindhoven, Netherlands

replaced Sony's Aibo quadruped robot in the RoboCup SPL. In the past few years Nao has evolved over several designs and several versions.

Nao (version V3.3) [4] is a 58cm, 5kg humanoid robot (Figure 2.2). The Nao robot carries a fully capable computer on-board with an x86 AMD Geode processor at 500 MHz, 256 MB SDRAM, and 2 GB flash memory running an Embedded Linux distribution. It is powered by a 6-cell Lithium-Ion battery which provides about 30 minutes of continuous operation and communicates with remote computers via an IEEE 802.11g wireless or a wired ethernet link.

Nao RoboCup edition has 21 degrees of freedom; 2 in the head, 4 in each arm, 5 in each leg, and 1 in the pelvis (there are two pelvis joints which are coupled together on one servo and cannot move independently). Nao, also, features a variety of sensors and transmitters. Two cameras are mounted on the head in vertical alignment providing non-overlapping views of the lower and distant frontal areas, but only one is active each time and the view can be switched from one to the other almost instantaneously. Each camera is a 640 x 480 VGA device operating at 30fps. The native colorspace provided by the cameras is the YUV422. Four sonars (two emitters and two receivers) on the chest allow Nao to sense obstacles in front of it. In addition, the Nao has a rich inertial unit, with

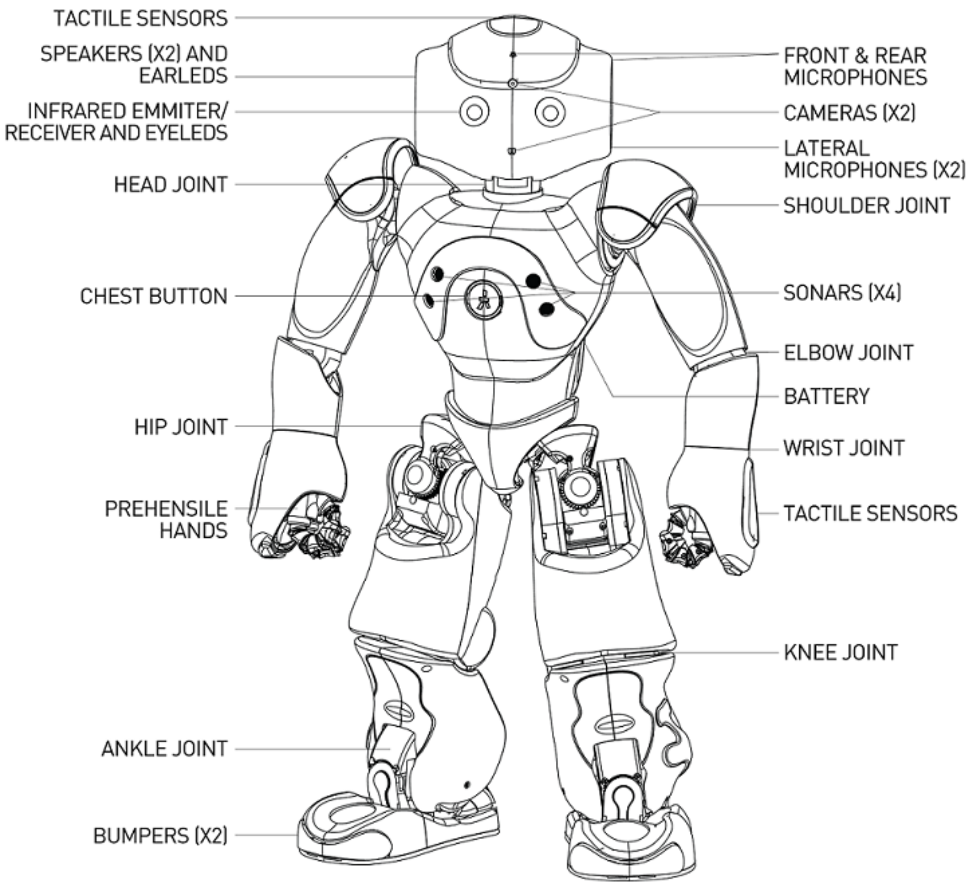

Figure 2.2: Aldebaran Nao robot (v3.3, academic edition) and its components

one 2-axis gyroscope and one 3-axis accelerometer, in the torso that provides real-time information about its instantaneous body movements. Two bumpers located at the tip of each foot are simple ON/OFF switches and can provide information on collisions of the feet with obstacles. Finally, an array of force sensitive resistors on each foot delivers feedback of the forces applied to the feet, while encoders on all servos record the actual values of all joints at each time.

Aldebaran Robotics has equipped Nao with both embedded and desktop software to be used as a base for further development (Figure 2.3). The embedded software, running on the motherboard located in the head of the robot, that the company provides includes an embedded GNU/Linux distribution and NAOqi, the main proprietary software that runs on the robot and controls it. Nao's desktop software includes Choregraphe, a visual programming application which allows the creation and the simulation of animations and

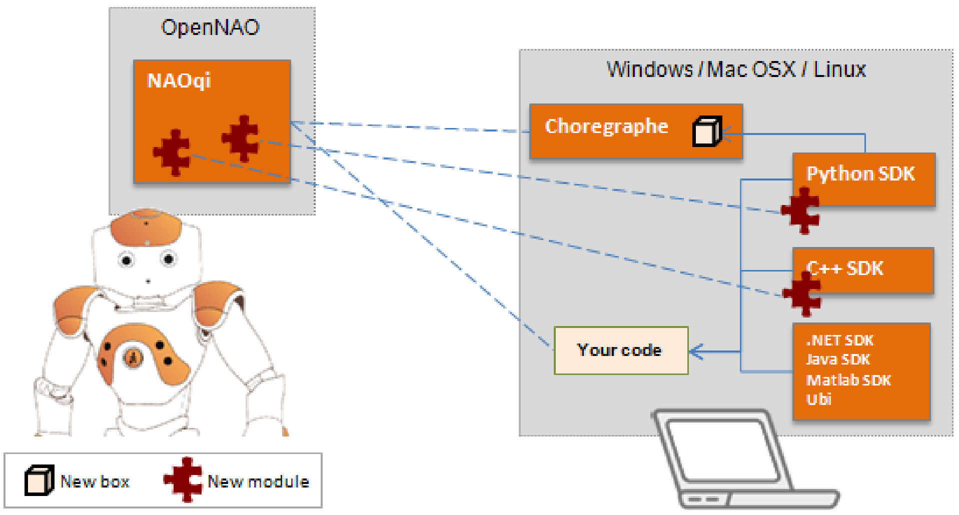

Figure 2.3: Embedded and desktop software for the Nao robot

behaviors for the robot before the final upload to the real Nao, and Telepathe which provides elementary feedback about the robot's hardware and a simple interface to accessing its camera settings. As far as the NAOqi framework is concerned, it is cross-platform, cross-language, and provides introspection which means that the framework knows which functions are available in the different modules and where. It provides parallelism, resources, synchronization, and events. NAOqi, also, allows homogeneous communication between different modules (motion, audio, video), homogeneous programming, and homogeneous information sharing. Software can be developed in C++, Python, and Urbi. The programmer can state which libraries have to be loaded when NAOqi starts via a preference file called `autoload.ini`. The available libraries contain one or more modules, which are typically classes within the library and each module consists of multiple methods (Figure 2.4).

2.2 RoboCup SPL Team Kouretes

Team Kouretes is the first and currently the only RoboCup SPL team founded in Greece, hosted in the Intelligent Systems Laboratory of the School of Electronic and Computer Engineering at the Technical University of Crete. Kouretes started developing their own robotic software framework in 2008 and the code is constantly developed and maintained

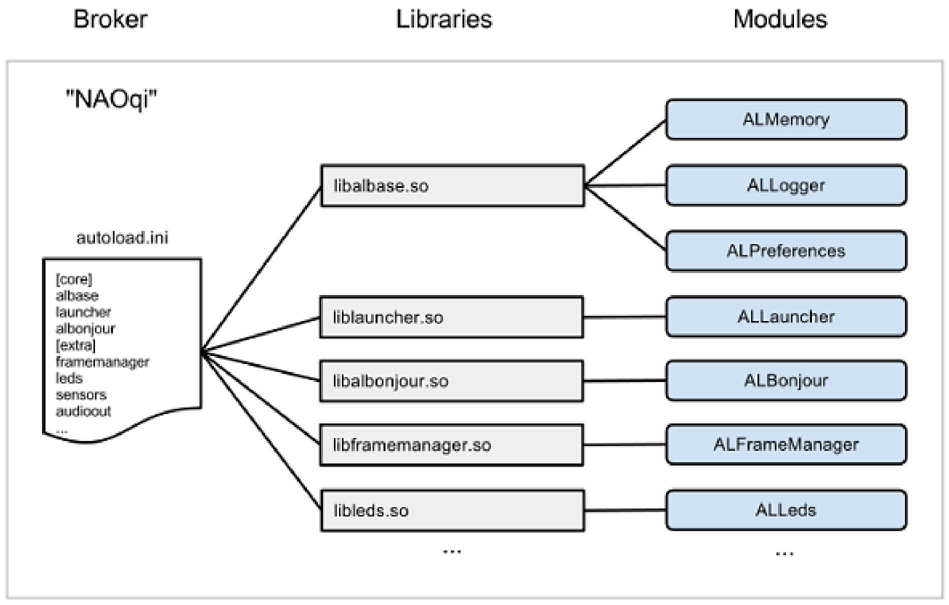

Figure 2.4: The NAOqi process

ever since. The team's publicly-available code repository includes a custom software architecture, a custom communication framework, a graphical application for behavior specification, and modules for object recognition, state estimation, localization, obstacle avoidance, behavior execution, and team coordination, which are briefly described below.

The team participates in the main RoboCup competition since 2006 in various soccer leagues (Four-Legged, Standard Platform, MSRS, Webots), as well as in various local RoboCup events (German Open, Mediterranean Open, Iran Open, RC4EW, RomeCup) and RoboCup exhibitions (Athens Digital Week, Micropolis, Schoolfest). Distinctions of the team include: 2nd place in MSRS at RoboCup 2007; 3rd place in SPL-Nao, 1st place in SPL-MSRS, among the top 8 teams in SPL-Webots at RoboCup 2008; 1st place in RomeCup 2009; 6th place in SPL-Webots at RoboCup 2009; 2nd place in SPL at RC4EW 2010; and 2nd place in SPL Open Challenge Competition at RoboCup 2011 (joint team Noxious-Kouretes). Recently, the team participated in RoboCup German Open 2012 in Magdeburg, in RoboCup Iran Open 2012 in Tehran, in RoboCup 2012 in Mexico City, in AutCup 2012 in Tehran, in RoboCup Iran Open 2013 in Tehran (Figure 2.5) and in RoboCup 2013 in Eindhoven, Netherlands (Figure 2.6).

Figure 2.5: Team Kouretes at RoboCup Iran Open 2013

2.2.1 Monas Software Architecture

Monas [5] is a flexible software architecture which provides an abstraction layer from the hardware platform and allows the synthesis of complex robot software as XML-specified Monas modules, Provider modules, and/or Statechart modules. Monas modules, the so-called agents, focus on specific functionalities and each one of them is executed independently at any desired frequency completing a series of activities at each execution. The base activities, that an agent may consist of, are described briefly below:

- `Vision` [6] is a light-weight image processing method for humanoid robots, via which Kouretes team accomplishes visual object recognition. The vision module determines the exact camera position in the 3-dimensional space and subsequently the view horizon and the sampling grid, so that scanning is approximately uniformly projected over the ground (field). The identification of regions of interest on the pixels of the sampling grid follows next utilizing an auto-calibrated color recognition scheme. Finally, detailed analysis of the identified regions of interest seeks potential

Figure 2.6: Team Kouretes at RoboCup 2013 in Eindhoven, Netherlands

matches for corresponding target objects. These matches are evaluated and filtered by several heuristics, so that the best match (if any) in terms of color, shape, and size for a target object is finally extracted. Then, the corresponding objects are returned as perceived, along with an estimate of their current distance and bearing.

- `LocalWorldState` [7] is the activity which used to realize Monte-Carlo localization (Particle Filters - PFs) and recently switched to using an Extended Kalman Filter (EKF) [8]. The belief of the robot is a probability distribution over the 3-dimensional space of coordinates and orientation (x, y, θ) represented approximately using a population of particles in the case of PFs or using a 3-dimensional Gaussian distribution in the case of EKF. Belief update is performed using an odometry motion model for omnidirectional locomotion and a landmark sensor model for the goalposts (landmarks). The robot's pose is estimated as the pose of the particle with the highest weight (PFs) or the mean (highest probability) of the Gaussian distribution (EKF).

- `SharedWorldModel` [9] is the activity that combines the local beliefs of all robots to create a common and shared estimation of the current state of the world consistent with these local beliefs. In order to generate this information, an Extended Kalman Filter (EKF) is employed with appropriate state transition and observation models, applying linearization where needed.

- `PathPlanning` [10] is the activity which accomplishes path planning with obstacle avoidance by first building a local, polar, obstacle occupancy map, which is updated constantly with real-time sonar information, taking into consideration the robot's locomotion. Afterwards, an A* search algorithm is used for path planning, the outcome of which suggests an obstacle-free path for guiding the robot to a desired destination. The way-points of the planned path are finally translated into walk commands to guide the robot along the path.

- `Behavior` is the activity which implements the desired robotic behavior. It operates on the outputs of the `Vision`, `LocalWorldState`, and `SharedWorldModel` activities and decides which one is the most appropriate action to be executed next (walk, kick, etc.). Locomotion actions are passed to the `PathPlanning` activity for obstacle-free navigation, while motion actions are sent to the `MotionController` activity for execution. Our mechanism for team coordination and planning proposed in this thesis was integrated into this activity. Details will be provided in Chapters 4 and 5.

- `HeadController` manages the movements of the robot head (camera).

- `MotionController` [11] is used for managing and executing robot locomotion commands and special actions.

- `RobotController` handles external signals on the game state.

- `LedHandler` controls the robot LEDs (eyes, ears, chest button, feet).

Provider modules accomplish the complete decoupling of the robotic hardware by collecting and filtering measurements from the robot sensors and cameras and forming them as messages in order to be utilized as input data by any interested Monas agents. Each provider module can be executed independently and at any desired frequency.

Custom Forward and Inverse Kinematics [12, 13], designed specifically for the NAO humanoid robot, have been implemented as an independent software library optimized for speed and efficiency. The library is currently being used in other team projects, such as omni-directional walk engine and dynamic kick engine.

Statechart modules, which offer an alternative intuitive graphical specification of robot behavior, have also been integrated into Monas [14]. Kouretes Statechart Engine (KSE) [15, 16] is our own graphical tool for designing and editing statecharts for robot behavior. Statecharts are automatically transformed into code and are executed on-board using a generic multi-threaded statechart engine, which provides the required concurrency and meets the real-time requirements of the activities on each robot.

KMonitor [17] is our own debugging tool created specifically for the Monas architecture that takes advantage of the modularity of Kouretes code and helps in finding errors or verifying that newly implemented features work correctly. It also allows for the easy creation of colortables, the transmission of remote commands over the network, etc.

2.2.2 Narukom Communication Framework

Narukom [18] is the communication framework developed for the needs of the team's code and it is based on the publish/subscribe messaging pattern. Narukom supports multiple ways of communication, including local communication among the Monas modules, the Providers modules, and the Statechart modules that constitute the robot software, and remote communication via multicast connection among multiple robot nodes and among robot and external computer nodes. The information that needs to be communicated between nodes is formed as messages which are tagged with appropriate topics and host IDs. Three types of messages are supported:

- `state`, which remain in the blackboard until replaced by a newer message,
- `signal`, which are consumed at the first read, and
- `data`, which are time-stamped to indicate the time their values were acquired.

To facilitate the serialization of data and the structural definition of the messages, Google Protocol Buffers were utilized. The user defines the data structure once and then uses the generated source code to write and read the defined structures to and from a variety of data streams using a variety of programming languages. Another great advantage of protocol buffers is that data structures can be enhanced without breaking the already deployed programs, which are capable of handling the old format of the structures. To use protocol buffers one must describe the information for serialization by defining protocol buffer messages in `.proto` files. A protocol buffer message is a small record of information, containing name-value pairs. The protocol buffer message format is simple and flexible. Each message type has at least one numbered field. Each field has a name and a value type. The supported types are integer, floating-point, boolean, string, raw bytes, or other complex protocol buffer message types, thus hierarchical structure of data is possible. Additionally, the user can specify rules, if a field is mandatory, optional, or repeated. These rules enforce both the existence and multiplicity of each field inside the message. As a next step, the user generates code for the desired language by running the protocol buffer compiler. The compiler produces data access classes and provides accessors and mutators for each field, as well as serialization/unserialization methods to/from raw bytes. Officially, Google supports C++, Java, and Python for code generation, but there are several other unofficially supported languages.

Additionally, the blackboard paradigm is utilized to provide efficient access to shared information stored locally at each node and is extended to support history queries and a mechanism that controls the information updates. Finally, to meet the delivery requirements among the remote and/or the local nodes, messages are relayed though a message queue. The message queue is responsible for collecting the published messages and allocating them to the interested subscribers through multiple buffers. Messages that have to be delivered to remote nodes are committed to the KNetwork module, which implements the multicast connection.

2.3 Utility Theory

In Economics, utility is a quantitative representation of preferences over some set of goods and services. In Artificial Intelligence (AI), we design agents that make intelligent decisions and when this decision-making process is under uncertainty it can be proved to be quite complicated. Thus, the purpose for using utility theory in decision-making is to create a mathematical model to aid the process by giving the decision maker the ability to quantify the desirability of certain alternatives. Utility theory [19] is used for analyzing scenarios where uncertainty and risk are considered. The benefit of using such a principled approach is the reduction of decision making into a problem of optimizing a

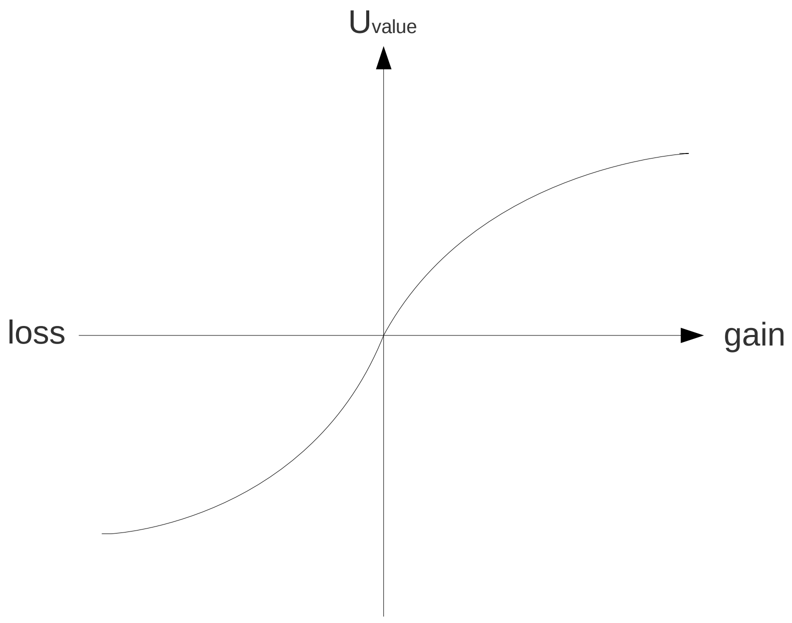

Figure 2.7: The relationship between utility function and desirability

function which represents the agent designer's preferences, given a certain set of design attributes, and can change the behavior of the agent to a desired behavior.

2.3.1 Utility Functions

Utility theory uses functions, commonly known as utility functions, to denote mathematical models representing certain criteria or preferences and can be used in the decision-making process to evaluate states of the world. A utility function delivers a single number to express the desirability of a state; the higher the value of the utility function the larger the gain (more desirable state) and likewise the lower the value the larger the loss (less desirable state), as shown in Figure 2.7. More formally, $U : S \mapsto \mathbb{R}$ is used to denote the utility of a state, where S is the state space of a problem and $\mathbb{R}$ is the set of real numbers. A utility function represents a preference relation $\preceq$ on S if and only if $\forall s_1, s_2 \in S$, $U(s_1) \leq U(s_2)$ implies $s_1 \preceq s_2$. If U represents $\preceq$, then this implies that $\preceq$ is complete and transitive, and hence rational. A rational agent would consistently make decisions which lead to states with higher utility. Completeness and transitivity are axioms which are meant to eliminate inconsistencies and suboptimal choices when it comes to trade-offs and uncertainty.

- Completeness of complete order: $(X \succ Y) \vee (X \prec Y) \vee (X \sim Y)$, which implies that X is either preferred, less-preferred, or equally-preferred to Y.
- Transitivity: if $X \succeq Y$ and $Y \succeq Z$, then $X \succeq Z$.

2.3.2 Multi-Attribute Utility Functions

There are two general types of utility functions, single-attribute (SAU) and multi-attribute (MAU). The single-attribute utility functions are monotonic functions, where the best outcome is set to 1 and the worst to 0. SAU functions are then developed to describe the designer's compromise between the best and worst alternatives. On the other hand, when certain independence conditions are met, a mathematical combination of several SAU functions, with scaling constants (weights), results in a multi-attribute (MAU) function, which is the overall utility function with all attributes considered. Scaling constants reflect designer's preference on the attributes, which is based on scaling constant lottery questions and preference independence questions.

In designing intelligent systems, one usually encounters at some stage the problem of evaluating states with respect to multiple performance criteria. The evaluation methods often vary in terms of complexity depending on the nature of the states themselves, the structure of the problem domain, the importance of a specific task, and other factors. In such cases, we need to determine a multi-attribute utility function (MAU) to match our problem preferences. Below, we present more formally the MAU functions.

Let $X_1, ..., X_n$, $n \geq 2$, be a set of attributes associated with the consequences of a decision problem. The utility of a consequence $(x_1, ..., x_n)$ can be determined in two ways:

- *direct assessment*: compute the combined utility $U(x_1, ..., x_n)$ over all attributes.
- *decomposed assessment* [20]:
  - compute n single-attribute utilities $U_i(x_i)$ for the n attributes.
  - compute $U(x_1, ..., x_n)$ by combining the $U_i(x_i)$ of all n attributes.

$$U(x_1, ..., x_n) = f\Big(U(x_1), ..., U(x_n)\Big)$$

There are multiple different functional forms for the combining function f of the decomposed assessment. If X and Y are two attributes (generalization to $n \geq 2$ is straightforward), the main forms are the following:

- U has an *additive* form, if for scaling constants w_X and w_Y

  $$U(X,Y) = w_X U_X(X) + w_Y U_Y(Y)$$

  or, in general, for n attributes

  $$U(X_1, ..., X_n) = \sum_{i=1}^{n} w_i U_i(X_i)$$

- U has an *multi-linear* form, if for scaling constants w_X, w_Y, and w_{XY}

  $$U(X,Y) = w_X U_X(X) + w_Y U_Y(Y) + w_{XY} U_X(X) U_Y(Y)$$

  or, in general, for n attributes

  $$U(X_1, ..., X_n) = \sum_{i=1}^{n} w_i U_i(X_i) + \sum_{i=1}^{n} \sum_{j>i} w_{ij} U_i(X_i) U_j(X_j) + ... + w_{12...n} U_1(X_1) ... U_n(X_n)$$

- U has an *multiplicative* form, if for scaling constants w_X, w_Y, c_X, and c_Y

  $$U(X,Y) = \big(w_X U_X(X) + c_X\big)\big(w_Y U_Y(Y) + c_Y\big)$$

  or, in general, for n attributes

  $$U(X_1, ..., X_n) = \prod_{i=1}^{n} (w_i U_i(X_i) + c_i)$$

2.4 Particle Swarm Optimization

Particle swarm optimization (PSO) was developed by Kennedy and Eberhart in 1995 [21], based on the swarm behavior, such as fish and bird schooling in nature. Since then, PSO has generated much wider interest, and forms an exciting, ever-expanding research subject, called swarm intelligence [22]. PSO has been applied to almost every area in optimization, computational intelligence, and design/scheduling applications.

2.4.1 PSO Description

In computer science, particle swarm optimization (PSO) [23] is a computational method that optimizes a problem by iteratively trying to improve a candidate solution with regard to a given measure of quality. PSO optimizes a problem by having a population of candidate solutions, here dubbed particles, and moving these particles around in the search space according to simple mathematical formula over the particle's position and velocity. Each particle's movement is influenced by its local best known position and is also guided toward the global best known position in the search space, which is updated as better positions are found by other particles. This is expected to move the particle swarm toward the best solutions.

PSO is a metaheuristic as it makes few or no assumptions about the problem being optimized and can search very large spaces of candidate solutions. However, metaheuristics, such as PSO, being local search methods, cannot guarantee an optimal solution will ever be found. On the other hand, PSO does not use the gradient of the problem being optimized, which means that PSO does not require the optimization problem to be differentiable, as is required by classic optimization methods, such as gradient descent and quasi-newton methods. Therefore, it can also be used on optimization problems that are partially irregular, noisy, and changing over time.

2.4.2 PSO Algorithm

The basic PSO algorithm works by having a population, called swarm, of candidate solutions, called particles. These particles are moved around in the search space according to a simple formula. The movement of a swarming particle consists of two major components, a stochastic component and a deterministic component. Each particle is attracted toward the positions of the current global best solution $\mathbf{g}^*$ and its own best local solution $\mathbf{x}_i^*$ in history, while at the same time it has a tendency to move randomly. When improved positions are being discovered these will then come to guide the movements of the swarm. The process is repeated and by doing so it is hoped, but not guaranteed, that a satisfactory solution will eventually be discovered.

More formally, let $\mathbf{x}_i$ and $\mathbf{v}_i$ be the position vector and velocity for particle i respectively, where $\mathbf{x}_i \in \mathbb{R}^n$ and $\mathbf{v}_i \in \mathbb{R}^n$. The new velocity and position at step $t+1$, given

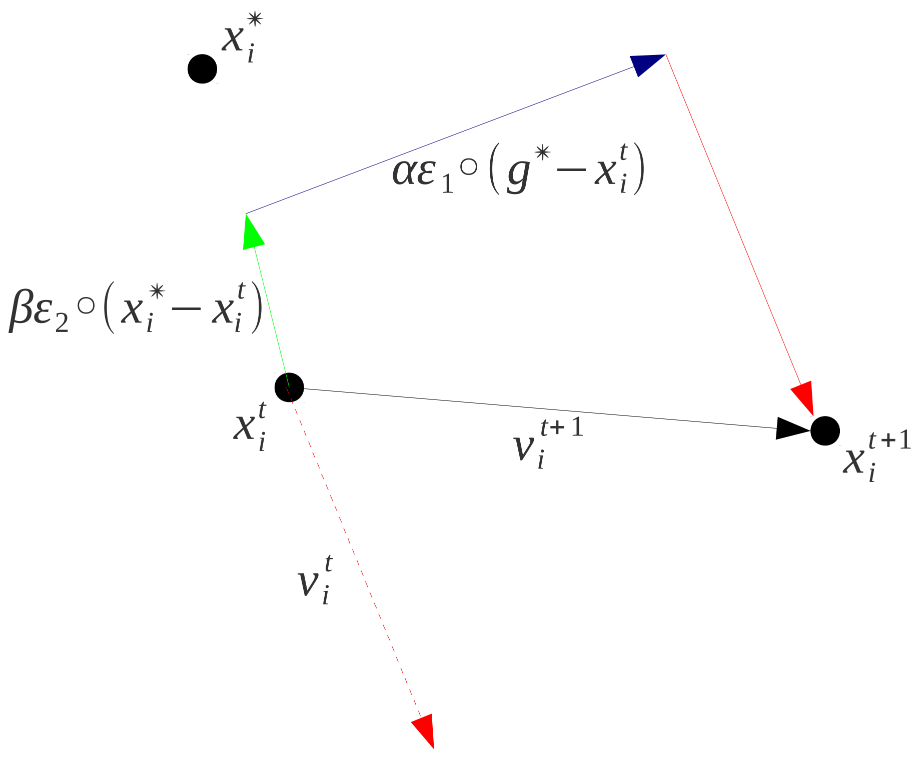
x_i^*
g^*
$\alpha\varepsilon_1 \circ (g^* - x_i^t)$
$\beta\varepsilon_2 \circ (x_i^* - x_i^t)$
x_i^t
v_i^{t+1}
x_i^{t+1}
v_i^t

Figure 2.8: Visualization of the PSO update step

the velocity and position at step t, are determined by the following formulas:

$$\mathbf{v}_i^{t+1} = \theta(t)\mathbf{v}_i^t + \alpha\boldsymbol{\epsilon}_1 \circ \left(\mathbf{g}^* - \mathbf{x}_i^t\right) + \beta\boldsymbol{\epsilon}_2 \circ \left(\mathbf{x}_i^* - \mathbf{x}_i^t\right)$$
$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \mathbf{v}_i^{t+1}$$

where $\boldsymbol{\epsilon}_1$ and $\boldsymbol{\epsilon}_2$ are two random vectors with entries taking values between 0 and 1 and $\circ$ is the element-wise (Hadamard) product between two vectors. The parameters α and β are the learning parameters or acceleration constants, which can typically be taken as, $\alpha \approx \beta \approx 2$. The most noticeable influence on the performance of the algorithm derives from the use of the inertia function $\theta(t)$, where $\theta(t) \in (0, 1)$. In the simplest case, the inertia function can be taken as a constant, typically $\theta \approx 0.5 \sim 0.9$. This is equivalent to introducing a virtual mass to stabilize the motion of the particles and thus the algorithm is expected to converge more quickly. A visualization of PSO update is shown in Figure 2.8.

PSO searches the space of an objective function by adjusting the trajectories of individual agents, called particles, as the piecewise paths formed by positional vectors in a quasi-stochastic manner. Formally, let $f : \mathbb{R}^n \mapsto \mathbb{R}$ be the cost function which must be minimized. The function takes a candidate solution as argument in the form of a vector of real numbers and produces a real number as output which indicates the objective function value of the given candidate solution. As we mentioned before the gradient of f is not known. The goal is to find a solution a for which $f(a) < f(b)$ for all b in the search space, which would mean a is the global minimum (Figure 2.9). Maximization can also be performed by considering the function $h = -f$ instead.

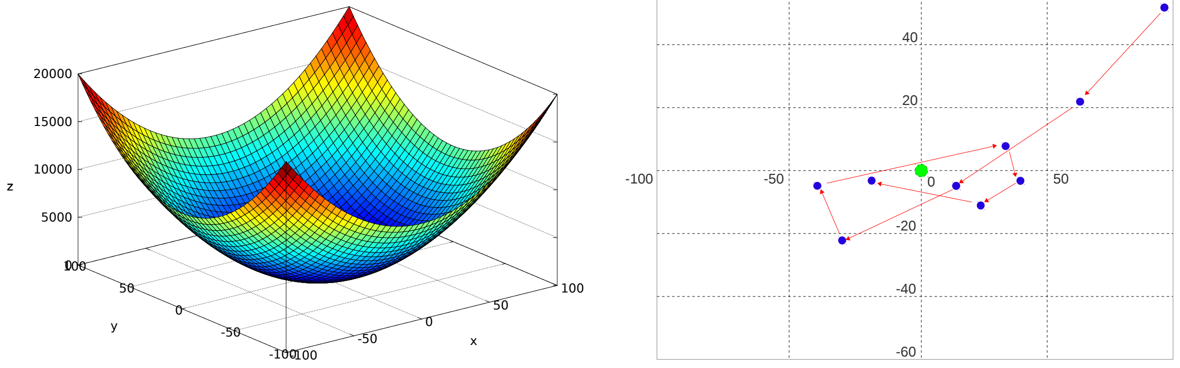

Figure 2.9: PSO search for global minimum: objective function $f(x, y) = x^2 + y^2$ (left) and particle movement towards the optimal solution $(0, 0)$ (right)

Chapter 3

Problem Statement

3.1 Dynamic Multi-Robot Coordination

In a multi-robot system, such as a RoboCup game, where several intelligent agents must interact simultaneously under partial observability to achieve a common goal as a team, the need for coordination and planning is a necessity, because such a mechanism offers better game performance and leads to efficient team-play. Therefore, it is crucial to come up with a high-level process, which will coordinate skills, such as player actions, perceptions, and communication means, yielding as a result a complete behavior for each agent within the frame of a global team strategy. As behavior we understand the execution of a selected pre-determined plan (among several choices) using local perception as input and local actions as output. In a sense, coordination provides the robots with a thinking process which enables them to decide what plan they should follow at any time for the global benefit of the team.

In any game of soccer, one of the most important issues, which favors effective team-play, is the formation or positioning of the team in the field, which defines where each player should be located inside the field in order to tackle the opponent team strategy. The same applies to the RoboCup Standard Platform League where teams of five robots each develop strategies through coordination to play soccer games effectively. Therefore, team positioning can be helpful in many situations. More specifically, robots should be able to generate a formation dynamically based on the global estimated ball position, which is provided by the shared world model of our team, by examining several candidate positions of importance to them as well as for the team for the current game situation. The problem of formation generation is one of the problems we address in this thesis.

Furthermore, apart from formation, it is of high importance to develop a team strategy, that is for each robot to undertake a specific role in order to execute the corresponding behavior, which is best suited for the current game situation. Obviously, the assignment of a role depends on the current location of each player in the field and takes into account the candidate positions in the generated formation. Therefore, it is necessary to come up with a role assignment algorithm, which allows the robots to assume different roles within the field (e.g. attacker, defender, supporter, etc.). To illustrate the importance of such a decision-making process, let us consider the following example. Imagine a simple scenario where two robots of the same team see the ball somewhere inside the field. In the case where there is no coordination mechanism, the robots must act individually to achieve their goal, that is to score. The result would probably be that both robots would rush towards the ball to kick it and score, however in this case they might bump into each other along the way, fall down, and fail to score. Imagine the worst-case scenario in which all four robot players (the goalkeeper excluded) are trying to score in the same way; the outcome will most likely be a congestion around the ball with no goals scored. Moreover, in the above examples no players assume responsibilities, such as acting as defenders or supporters. On the contrary, if they coordinate, generate a formation, and assume a specific role corresponding to a position, they can clearly handle the situation more effectively. For example, only one of them will approach the ball, while the others will either choose to protect their goalpost acting as defenders or choose to support the attacking robot by staying alert to take action if necessary. The problem of role assignment is the second major problem we address in this thesis.

One can clearly see that in scenarios like the ones described above, the use of coordination would highly benefit the team. Of course, such a coordination mechanism must be supported by a reliable communication method between the robots, by a shared world model that provides the necessary state information, and by individual robot skills so that the decided roles can be executed. Given that the required support is now available in our RoboCup SPL team, the goal of this thesis is to provide an effective coordination mechanism, which includes dynamic positioning and role assignment.

3.2 Related Work

Many RoboCup teams have developed coordination and planning mechanisms, since they are essential as discussed above. Since these mechanisms are crucial for team performance, most SPL teams choose to not publish their strategies. Therefore, below we will briefly present some of the related work done by teams on other RoboCup leagues. Considering all the methods reviewed, one can hardly distinguish a single best choice to address the problem stated. This is the reason behind the numerous coordination methods among RoboCup teams and there is no single "standard" way to go.

3.2.1 Dynamic Role Assignment and Formation

UT Austin Villa is one of the best teams in RoboCup 3D Simulation League and the RoboCup 2011 champion in that league. In a paper they published that year [24], they purposed a coordination mechanism for dynamic positioning on the field and role assignment. Specifically, a formation is produced based on the ball location and the candidate positions, corresponding to different roles, are generated using relative offsets from the ball and the formation is broken into two different groups, an offensive and a defensive (Figure 3.1). Then by using a custom role assignment function, using mainly distance as a feature, they evaluate all possible mappings of agents/players to positions/roles through a dynamic programming algorithm to reduce the search in the space of mappings. Given that this procedure is executed locally on each agent and the information about the world is not always perfect, after the optimal mapping is found by each agent, they communicate and through a voting system they share their mappings and decide to adopt the mapping with the most votes.

3.2.2 Non-Communicative Action-Level Coordination

UvA Trilearn [25], also one of the best teams in simulated robot soccer, is using a different approach that focuses on action coordination among the agents rather than positioning and role assignment. In their work they use a game theoretic approach to address the problem of coordination by defining roles and attach to each role a set of possible actions. Then by using coordination graphs they exploit dependencies among the agents, meaning that each agent has to coordinate only with a small subset of other agents (Figure 3.2).

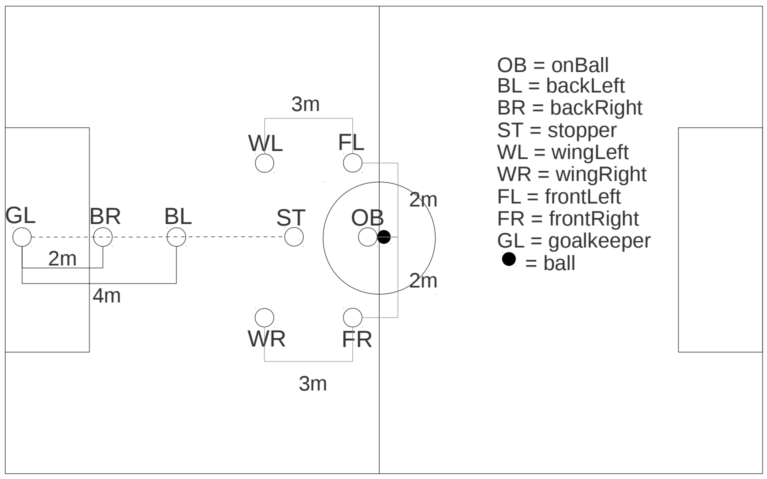

Figure 3.1: UT Austin Villa formation

Thus, assuming that a global payoff function can be decomposed into a linear combination of local payoff functions, each involving only a few agents, they perform a variable elimination algorithm to achieve coordination among role's actions. In order to perform the graph elimination a role assignment process is done by using associated potentials for each role, which agents use to estimate how appropriate they are for a specific role. Moreover, the elimination algorithm can be applied even when communication is unavailable, assuming the payoff function of each agent is common knowledge among all agents that are directly reachable from him in the coordination graph. Although, this approach focuses on action coordination that can achieve a more strategic playing, it requires a set of actions for each role, which depend on complex skills such as passing, ball controlling, and even opponent recognition and modeling, when we refer to actual robots.

3.2.3 Dynamic Role Assignment Through Bidding Functions

CMU team [26] at the time of the Four-Legged League with Sony Aibo robots had proposed a mechanism for dynamic role assignment using a common set of bidding functions and shared information about the ball location and robot beliefs about their position in the field. These functions encode heuristic information about the world and return an estimate of how suitable each robot is for a particular task. More specifically, robots first

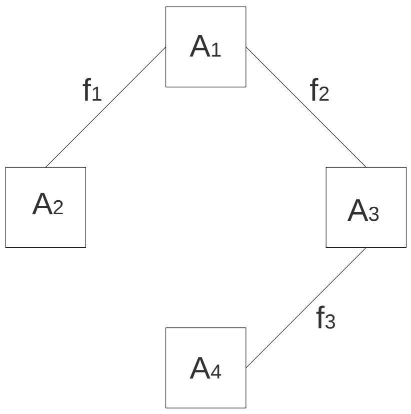

Figure 3.2: Coordination graph with four agents (nodes) and loose dependencies (edges)

calculate their own suitability using local information from their own world model and then they use the same functions to calculate the bids of their teammates using only the shared information provided by each teammate. Once each robot calculates the bids for itself and each of its teammates, it compares them. If it has the highest bid for the task being considered, it undertakes that task. If it was not the winner, it assumes that the winning robot will take up that particular task and performs calculations for the next task in the list. Due to the simplicity of auction clearance, this approach works well, when the task bids do not depend on the final assignments for the other tasks, in other words the various tasks are loosely dependent, and additionally when the local beliefs are fairly accurate.

Chapter 4

Our Approach

Our solution to the problem of coordination is based on the creation of a mechanism that is responsible for generating dynamically a team formation based on the global estimated ball position, provided by our shared world model, and choosing a mapping of robots to positions of the formation, that also yields a role assignment, according to a team utility function. The output of the coordination mechanism allows the robots to assume a best role and execute the corresponding behavior (plan). Our approach is similar to the ideas described in Sections 3.2.1 and 3.2.3.

4.1 The Idea

For the purposes of this thesis, we define coordination to be a positioning and role assignment problem that allows the team robots to assume roles based on the current game situation. Coordination can take place at different levels of granularity. We chose the level of player roles, because the current skills of our robots allow interaction between the robots only at this level. The lower action level, which currently includes locomotion, kicking, blocking, and ball tracking, does not allow for direct interaction between the robots, such as ball passing, and therefore is not suitable for coordination. Given the choice of the role level, complex and computationally intensive coordination methods at a detailed action level, such as coordination graphs described in Section 3.2.2, did not fit our purposes. Nevertheless, our methodology can be extended to a more detailed level and possibly incorporate such methods, provided there exists corresponding actions/skills at that level.

The general idea of our approach for coordination is fairly simple. Each robot is using the global estimated ball position to generate a set of candidate positions in the field. Each of these positions correspond to a specific role that is decided during the generation process. The set of positions that is produced is consistent among robots because it is derived directly from the common global belief they maintain. Then, each robot executes the same role assignment algorithm, evaluates possible mappings of the currently active robots to that formation in order to decide the best one. Once the best mapping is decided, each robot undertakes the corresponding role and executes the corresponding role behavior. The entire process is repeated when a significant change in the game situation occurs or a certain amount of time has passed to keep the team constantly coordinated.

4.2 Formation Generation

In this section we describe the first phase of our coordination mechanism, the formation generator, which takes as input the currently global estimated ball position $(x_{ball},\ y_{ball})$, the size n of the formation to be generated ($n \geq 4$ positions) and computes dynamically a set of promising candidate positions on the field. The generator produces both offensive and defensive types of formations, depending on which side of the field the ball lies and thus its functionality can be distinguished in two cases. In each case a subset of all possible roles (Table 4.1) is used based on the formation type and the size n requested. Specifically, in order to determine where each role should be positioned, the generator assumes a simple parametric grid adapted to the dimensions of the field (Figure 4.1) and places the formation roles based on which area of the grid the ball lies, using a set of simple equations for each role type. These equations exploit specific offsets and percentages combined with the field specifications (Figure 4.2, Table 4.2) to produce a consistent set of positions regardless of the actual field dimensions. Therefore, it determines the formation type by examining the value of the x_{ball} coordinate and updates the corresponding roles for that type according to the grid area the ball is located in using each role's equations. The offsets and percentages used in these equations were determined through experimentation and are presented in Table 4.3. By convention, we consider the left side of the field to be the own side and the right side of the field to be the opponent side. We further assume that the field is centered at $(0,\ 0)$.

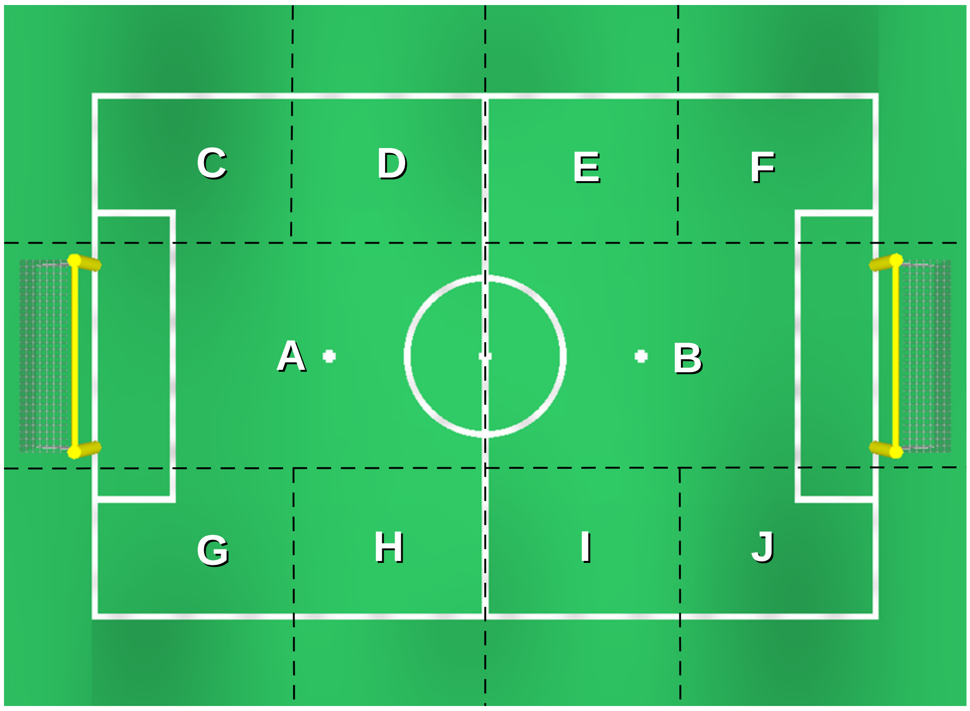

Figure 4.1: Field areas of possible ball locations for determining formation

Table 4.1: Possible roles of robot players for realizing a team strategy

Code	**Role**	**Description**
GK	Goalkeeper	Protects the team's goalpost in the goal area
D	Defender	Protects the team's goalpost in the middle field zone
DL	Left Defender	The Defender role in the left field zone
DR	Right Defender	The Defender role in the right field zone
OB	On Ball/Attacker	Approaches the ball in order to claim possession
S	Supporter	Assists the On Ball/Attacker role
LS	Left Supporter	The Supporter role in the left field zone
RS	Right Supporter	The Supporter role in the right field zone

4.2.1 Offensive Formation

In the offensive formation case we assume that the most important subset of roles are the goalkeeper, the defender, the on ball, and the supporter (when $n = 4$) or the left and the right supporters (when $n \geq 5$). Therefore, whenever the ball is located in a grid area

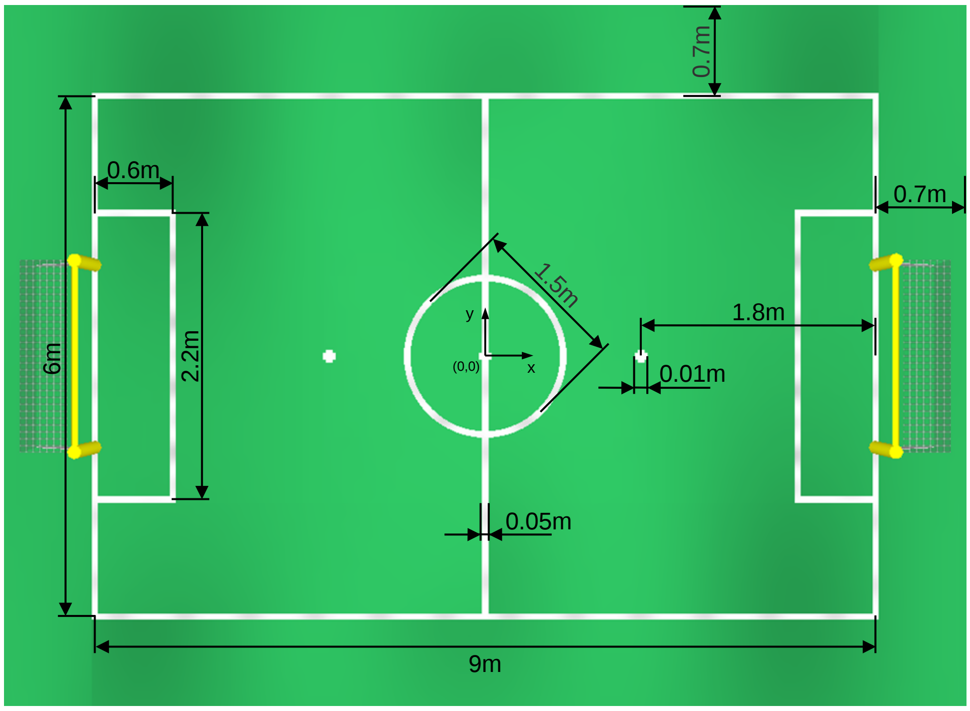

Figure 4.2: SPL 2013 field specifications [1]

Table 4.2: Field parameters for the SPL 2013 field

Name	Value/Interval (Meters)	Description
X_{field}	$[-4.5, +4.5]$	Field x-axis dimension
Y_{field}	$[-3.0, +3.0]$	Field y-axis dimension
$X_{Lgoalarea}$	$[-4.5, -3.9]$	Left goal area in x-axis
$X_{Rgoalarea}$	$[+3.9, +4.5]$	Right goal area in x-axis
$Y_{goalarea}$	$[-1.1, +1.1]$	Goal area in y-axis
$Y_{Lgoalpost}$	$+0.7$	Left goalpost y-coordinate
$Y_{Rgoalpost}$	-0.7	Right goalpost y-coordinate
X_{Lmark}	-2.7	Left penalty mark x-coordinate
X_{Rmark}	$+2.7$	Right penalty mark x-coordinate
Y_{mark}	0.0	Penalty mark y-coordinate
d_{center}	1.5	Center circle diameter

which is on the opponent side of the field (areas B, E, F, I, J or $x_{ball} \geq 0$), the generator

Table 4.3: Offsets and percentages used in our formation generator

Name	Value	Description
$sideZone$	50%	Size of side zones (C/D,G/H) relative to half-field x-dimension
$midZone$	26.8%	Size of grid middle zones (A,B) relative to field y-dimension
$ballOffset$	-0.2m	Offset from the ball along the x-axis for the on ball role
$xFactor$	22%	Relative offset on the x-axis for support roles
$xFwdFactor$	33%	Relative offset on the x-axis for forward support roles
$xBwdFactor$	44%	Relative offset on the x-axis for backward support roles
$goalOffset$	0.1m	Offset from goalpost along the y-axis for the goalkeeper role
$offset$	0.2m	Offset along the x- or y- axis for goalkeeper/defender roles
$xBound$	75%	Relative bound to x-coordinate for defender roles
$yBound$	50%	Relative bound to y-coordinate for defender roles

constructs a set of these roles and computes their positions, according to the field grid, as described above. Thereafter, if the size n of the formation requested is greater than this set, additional positions are generated (Section 4.2.3). Below, we present the set of equations for each role of the offensive type of formation based on the location of the ball in the field grid (Figure 4.1).

- *Goalkeeper*:

$$x_{goalkeeper} = \min(X_{field})$$
$$y_{goalkeeper} = 0$$

- *Defender*:

$$x_{defender} = xBound \times \min(X_{field})$$
$$y_{defender} = \begin{cases} +offset & \text{if } y_{ball} < 0 \\ -offset & \text{if } y_{ball} \geq 0 \end{cases}$$

- *On Ball/Attacker*:

$$x_{onball} = x_{ball} + ballOffset$$
$$y_{onball} = y_{ball}$$

- *Supporter* ($n = 4$):

$$x_{supporter} = \begin{cases} x_{ball} - xFwdFactor \times \max(X_{field}) & \text{ball in area F or J} \\ x_{ball} + xFwdFactor \times \max(X_{field}) & \text{ball in area E or I} \\ x_{ball} & \text{ball in area B} \end{cases}$$

$$y_{supporter} = \begin{cases} \min(Y_{goalarea}) & \text{ball in area F or (in B and } y_{ball} > 0) \\ \max(Y_{goalarea}) & \text{ball in area J or (in B and } y_{ball} \leq 0) \\ 0 & \text{ball in area E or I} \end{cases}$$

- *Left Supporter* ($n \geq 5$):

$$x_{lsupporter} = \begin{cases} x_{ball} - xFactor \times \max(X_{field}) & \text{ball in area B or J} \\ x_{ball} + xFwdFactor \times \max(X_{field}) & \text{ball in area E} \\ x_{ball} & \text{ball in area F or I} \end{cases}$$

$$y_{lsupporter} = \begin{cases} +|x_{lsupporter}|^{-1} & \text{ball in area B} \\ \max(Y_{goalarea}) & \text{ball in area J} \\ 0 & \text{ball in area E, F, or I} \end{cases}$$

- *Right Supporter* ($n \geq 5$):

$$x_{rsupporter} = \begin{cases} x_{ball} - xFactor \times \max(X_{field}) & \text{ball in area B or F} \\ x_{ball} + xFwdFactor \times \max(X_{field}) & \text{ball in area I} \\ x_{ball} & \text{ball in area E or J} \end{cases}$$

$$y_{rsupporter} = \begin{cases} -|x_{rsupporter}|^{-1} & \text{ball in area B} \\ \min(Y_{goalarea}) & \text{ball in area F} \\ 0 & \text{ball in area E, F, or I} \end{cases}$$

Figure 4.3 presents the formations generated, when the ball is located in area B, either back or forth (ball is the orange circle beside the on ball role). Figure 4.4 presents the formations generated, when the ball is located in areas J and E.

4.2.2 Defensive Formation

In a similar way, we assume that the most important subset of roles for the defensive formation case are the goalkeeper, the on ball, the left defender, the right defender, and the supporter ($n \geq 5$). Therefore, whenever the ball is located in a grid area which is on the own side of the field (areas A, C, D, G, H or $x_{ball} < 0$), the generator constructs a set of these roles and computes their positions, according to the field grid, as described above. Thereafter, if the size n of the formation requested is greater than this set, additional

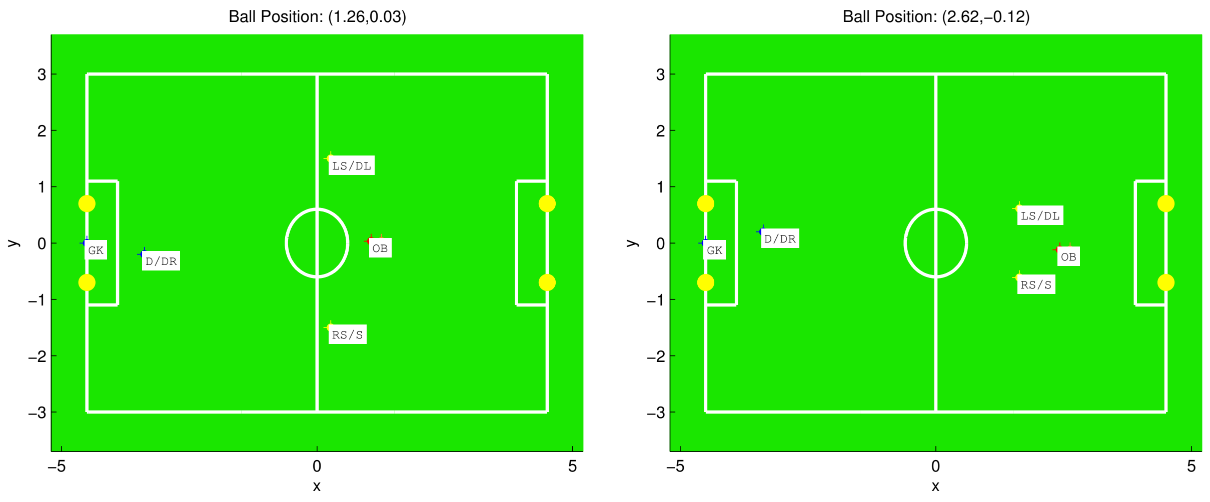

Figure 4.3: Offensive formation when the ball in area B (back) and in area B (forth)

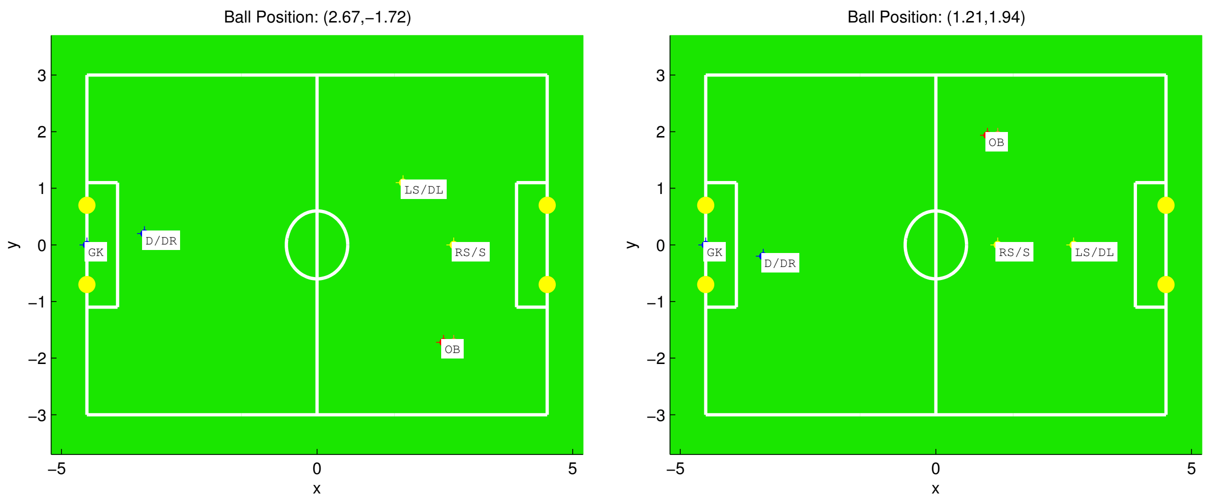

Figure 4.4: Offensive formation when the ball in area J (left) and in area E (right)

positions are generated (Section 4.2.3). Below, we present the set of equations for each role of the defensive type of formation based on the location of the ball in the field grid (Figure 4.1).

- *Goalkeeper*:

$$x_{goalkeeper} = \begin{cases} \min(X_{field}) & \text{ball in area A} \\ \min(X_{field}) + offset & \text{ball in area C or G} \\ \min(X_{field}) + \dfrac{y_{ball} - \max(Y_{goalarea})}{2\max(Y_{field})} & \text{ball in area D} \\ \min(X_{field}) - \dfrac{y_{ball} - \min(Y_{goalarea})}{2\max(Y_{field})} & \text{ball in area H} \end{cases}$$

$$y_{goalkeeper} = \begin{cases} 0 & \text{ball in area A} \\ Y_{Lgoalpost} - goalOffset & \text{ball in area C} \\ Y_{Rgoalpost} + goalOffset & \text{ball in area G} \\ \dfrac{y_{ball} - \max(Y_{goalarea})}{\max(Y_{field})} & \text{ball in area D} \\ \dfrac{y_{ball} - \min(Y_{goalarea})}{\max(Y_{field})} & \text{ball in area H} \end{cases}$$

- *On Ball/Attacker*:

$$x_{onball} = \begin{cases} \max\begin{Bmatrix} x_{ball} + ballOffset \\ \max(X_{Lgoalarea}) \end{Bmatrix} & \min(Y_{goalarea}) \leq y_{ball} \leq \max(Y_{goalarea}) \\ x_{ball} + ballOffset & y_{ball} > \max(Y_{goalarea}) \text{ or } y_{ball} < \min(Y_{goalarea}) \end{cases}$$

$$y_{onball} = y_{ball}$$

- *Left Defender*:

$$x_{ldefender} = \begin{cases} xBound \times \min(X_{field}) - \dfrac{y_{ball} - \max(Y_{goalarea})}{25} & \text{ball in area C or D} \\ xBound \times \min(X_{field}) & \text{ball in area A, G, or H} \end{cases}$$

$$y_{ldefender} = \begin{cases} yBound \times \max(Y_{goalarea}) - \dfrac{y_{ball} - \max(Y_{goalarea})}{2|\max(Y_{field})| - 1} & \text{ball in area C or D} \\ yBound \times \max(Y_{goalarea}) & \text{ball in area A, G, or H} \end{cases}$$

- *Right Defender*:

$$x_{rdefender} = \begin{cases} xBound \times \min(X_{field}) + \dfrac{y_{ball} - \min(Y_{goalarea})}{25} & \text{ball in area G or H} \\ xBound \times \min(X_{field}) & \text{ball in area A, C, or D} \end{cases}$$

$$y_{rdefender} = \begin{cases} yBound \times \min(Y_{goalarea}) - \dfrac{y_{ball} - \min(Y_{goalarea})}{2|\min(Y_{field})| - 1} & \text{ball in area G or H} \\ yBound \times \min(Y_{goalarea}) & \text{ball in area A, C, or D} \end{cases}$$

- *Supporter* ($n \geq 5$):

$$x_{supporter} = xBwdFactor \times \max(X_{field})$$

$$y_{supporter} = \begin{cases} \max(Y_{goalarea}) & \text{ball in area C or D} \\ \min(Y_{goalarea}) & \text{ball in area G or H} \\ y_{ball} & \text{ball in area A} \end{cases}$$

Figure 4.5 presents the formations generated, when the ball is located in area A, either back (inside the goal area) or forth (ball is the orange circle beside the on ball role). Figure 4.6 presents the formations generated, when the ball is located in areas D and G. Note that the formation ensures that no roles, other than the goalkeeper, are positioned inside the own goal area, according to the SPL rules about illegal defenders.

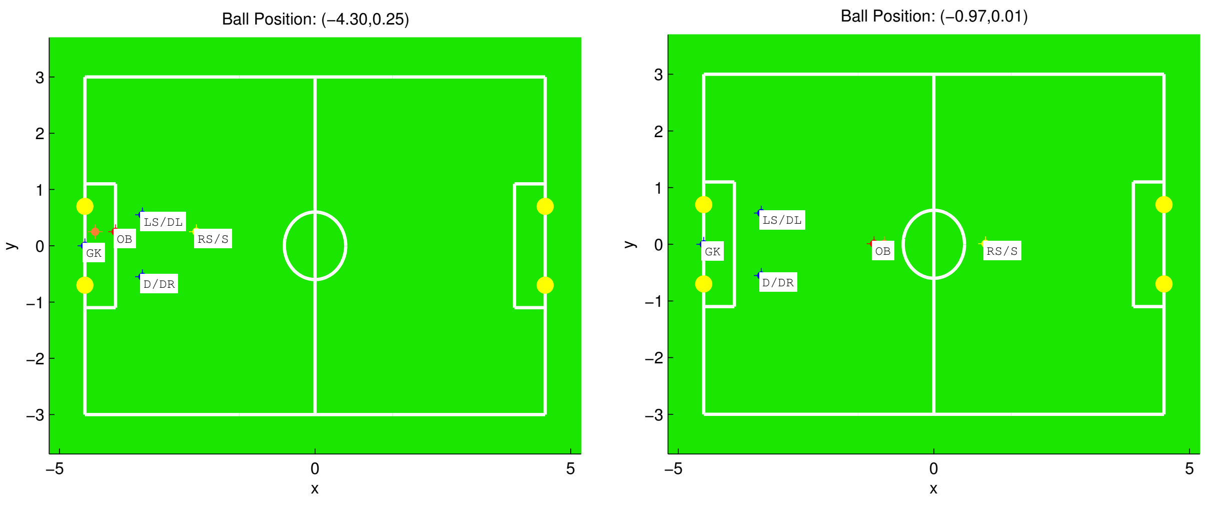

Figure 4.5: Defensive formation when the ball in area A (back) and in area A (forth)

4.2.3 Multiple Positions Formation

Moreover, in order to be able to evaluate formations with a larger set of candidate positions and search a larger space of mappings of robots to a subset of these positions, we enabled the formation generator to generate multiple positions in a simple way. Given a requested number n of positions greater than five ($n > 5$), after the five standard positions are generated using the update equations described in the above sections, the remaining ones are divided into groups for each possible role, except the goalkeeper whose

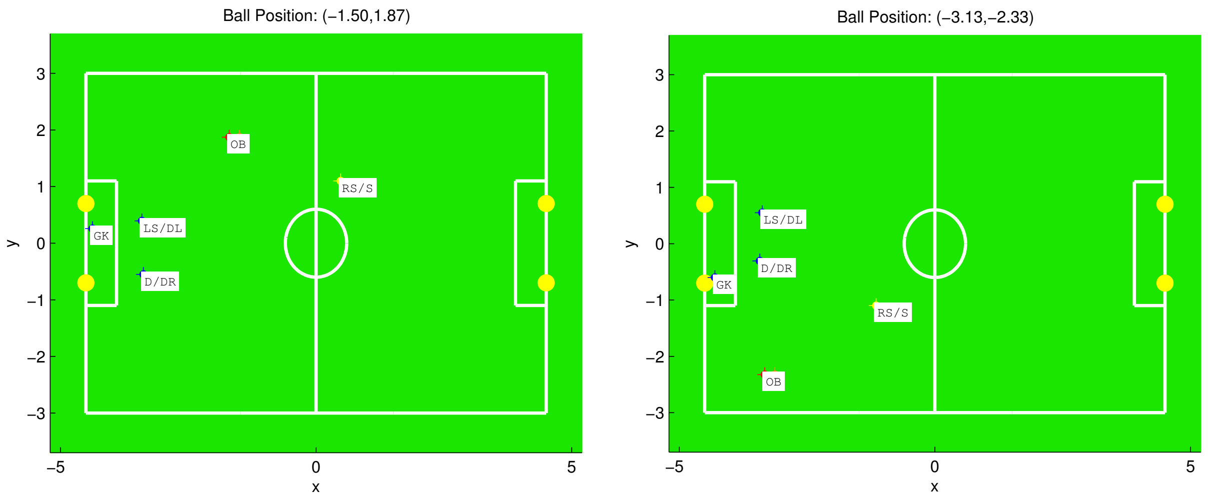

Figure 4.6: Defensive formation when the ball in area D (left) and in area G (right)

mapping is predetermined. These additional positions for each role are placed around the nominal role position (on a circle of fixed radius), therefore creating small clouds of positions around each role. Figure 4.7 presents formations of 12 and 20 positions.

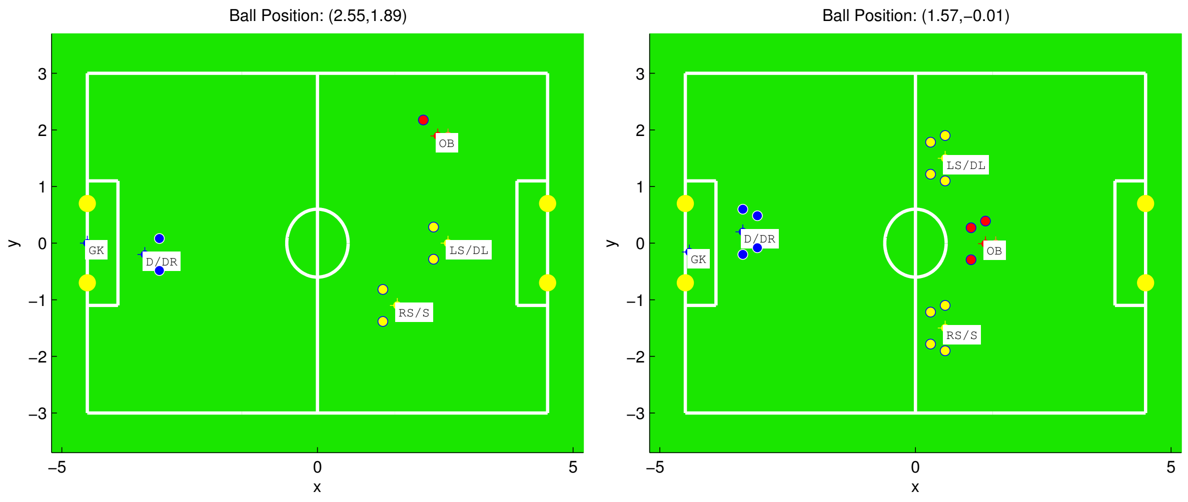

Figure 4.7: Formation generation with 12 (left) and 20 (right) positions

4.3 Utility Function

Given a formation, an appropriate mapping of robots to roles in the formation is required. Therefore, in order for our team robots to be able to quantify the desirability of any possible mapping, we developed a multi-attribute utility function (Section 2.3.2), combining many features to represent preferences related to the desired team strategy. More formally, a mapping *map* is defined as a function from the currently active robots to positions/roles in the formation:

$$map : Robots \mapsto Positions$$

where $Robots = \{r_1, r_2, \ldots, r_k\}$ and $Positions = \{p_1, p_2, \ldots, p_n\}$. Each robot carries information about its pose and each position carries information about its location in the field. Therefore, a mapping can be viewed as follows:

$$map(r_i) = p_j \quad \text{or} \quad r_i \rightarrow p_j, \quad \text{where} \quad r_i = (x_i^r,\ y_i^r,\ \theta_i^r) \quad \text{and} \quad p_j = (x_j^p,\ y_j^p)$$

The features used in the utility function to score each possible mapping are the following:

1. **Distance** C_d - This cost feature represents the distance between the current and the target position of robot i. Robots try to minimize this feature to be able to reach their target positions as soon as possible.

$$C_d(r_i \rightarrow p_j) = \frac{\sqrt{(x_i^r - x_j^p)^2 + (y_i^r - y_j^p)^2}}{\sqrt{\min(X_{field})^2 + \max(X_{field})^2 + \min(Y_{field})^2 + \max(Y_{field})^2}}$$

2. **Rotation Angle** C_r - This cost feature represents the minimum angle the robot i must rotate to face the target position (Figure 4.8). Robots try to minimize this feature to be able to reach their target positions as soon as possible.

$$C_r(r_i \rightarrow p_i) = \left|\frac{\phi}{\pi}\right|, \quad \phi \in [-\pi, +\pi]$$

3. **Potential Collisions** C_c - Approximating the route of each robot as a straight line between its current and its target position, we are able to check if that route intersects with the routes of other robots. For any pair of routes, if there is no intersection, the cost is 0 (infinitesimal probability of collision). If there is intersection

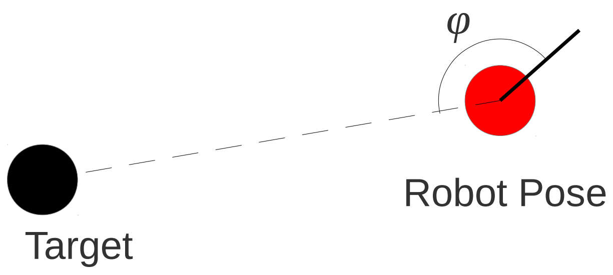

Figure 4.8: Minimum rotation angle feature in the utility function

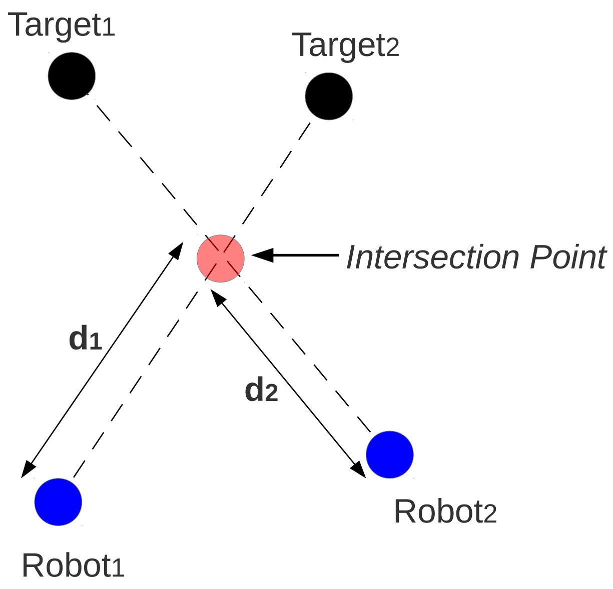

Figure 4.9: Collision detection feature in the utility function

and the intersection point is approximately equidistant (the absolute difference is less than 1.5 meters) from the current positions of the corresponding robots, the cost is 1 (high probability of collision). Finally, if there is intersection, but the intersection point is not equidistant from the current positions, the cost is 0.4 (small probability of collision). Figure 4.9 shows the key idea behind the detection of a possible collision between two robots; if $|d_1 - d_2| \leq 1.5m$, a collision is highly possible. Robots try to minimize this feature to avoid collisions.

$$C_c(r_{i_1} \rightarrow p_{j_1},\ r_{i_2} \rightarrow p_{j_2}) = \begin{cases} 0 & \text{no intersection} \\ 1 & \text{intersection and } |d_{i_1} - d_{i_2}| \leq 1.5m \\ 0.4 & \text{otherwise} \end{cases}$$

4. **Field Scoring** F_s - In order to include preferences over the set of candidate positions, we had to define a simple, yet functional, way to give a value to every spot of the soccer field. Thus, we constructed a feature consisting of a sum of Gaussian distributions on the two-dimensional space, each one having as a center one of the candidate positions generated. The key idea is that the value is spread proportionally around each position and we can define a preference for any position by changing its distribution amplitude A and variance σ^2. For example, the positions around the ball will impose higher values in contrast to defending positions and thus robots will be encouraged to undertake the on ball (OB) role. Each robot computes the field scoring feature at its assigned position taking into account the value contribution of all positions in the formation. Figure 4.10 presents the above field scoring function over the entire soccer field for the formation outcome presented in Figure 4.4 (left), whose analytical form for this specific case is:

$$\begin{aligned}
F_s(r_i \rightarrow p_j) = A_{onball} \exp\left(-\left(\frac{(x_j^p - x_{onball})^2}{2\sigma_x^2} + \frac{(y_j^p - y_{onball})^2}{2\sigma_y^2}\right)\right) \\
+A_{lsupporter} \exp\left(-\left(\frac{(x_j^p - x_{lsupporter})^2}{2\sigma_x^2} + \frac{(y_j^p - y_{lsupporter})^2}{2\sigma_y^2}\right)\right) \\
+A_{rsupporter} \exp\left(-\left(\frac{(x_j^p - x_{rsupporter})^2}{2\sigma_x^2} + \frac{(y_j^p - y_{rsupporter})^2}{2\sigma_y^2}\right)\right) \\
+A_{defender} \exp\left(-\left(\frac{(x_j^p - x_{defender})^2}{2\sigma_x^2} + \frac{(y_j^p - y_{defender})^2}{2\sigma_y^2}\right)\right)
\end{aligned}$$

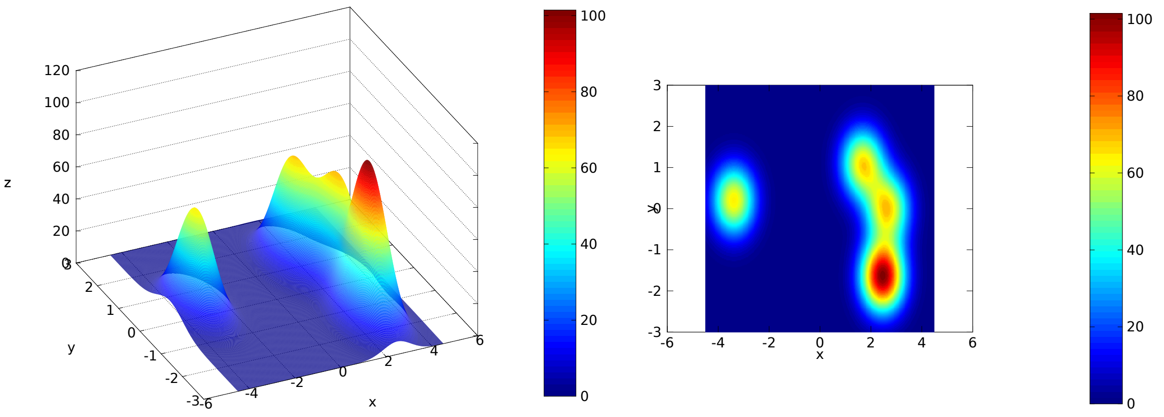

Figure 4.10: Field scoring function with four positions: side (left) and top (right) view

5. **Sparse Targets** C_{st} - This cost represents the sum of distances between the target positions of all robots. Robots try to maximize this feature to give preference to targets that are not too close (less than 0.7 meters) to targets of other robots. The purpose of the this feature is to discourage congestion in small areas of the field.

$$C_{st}(r_{i_1} \rightarrow p_{j_1},\ r_{i_2} \rightarrow p_{j_2}) = \begin{cases} 100 & \sqrt{(x^p_{j_1} - x^p_{j_2})^2 + (y^p_{j_1} - y^p_{j_2})^2} \leq 0.7m \\ 0 & \text{otherwise} \end{cases}$$

6. **Robot Stability** C_s - This cost represents the attacker robot stability/health. The key idea behind this feature is to keep a count c_i of the times robot i has fallen or has been penalized and give a higher cost as this number increases. The analytical form of this function is shown below:

$$C_s(r_i \rightarrow p_j) = \begin{cases} 0.1c_i & \text{if } p_j = p_{onball} \text{ and } c_i \leq 10 \\ 1 & \text{if } p_j = p_{onball} \text{ and } c_i > 10 \\ 0 & \text{if } p_j \neq p_{onball} \end{cases}, \quad c_i \in \mathbb{N}^0$$

The features described above are computed for each of the robots, are weighted, and are summed to form the final utility function of a mapping *map*:

$$U(map) = \sum_{i=1}^{k-1} \Big(w_u F_s(r_i \rightarrow p_j) - w_d C_d(r_i \rightarrow p_j) - w_r C_r(r_i \rightarrow p_j) - w_s C_s(r_i \rightarrow p_j) \Big) + \sum_{i_1=1}^{k-1} \sum_{\substack{i_2=1 \\ i_2 \neq i_1}}^{k-1} \Big(- w_c C_c(r_{i_1} \rightarrow p_{j_1},\ r_{i_2} \rightarrow p_{j_2}) - w_{st} C_{st}(r_{i_1} \rightarrow p_{j_1},\ r_{i_2} \rightarrow p_{j_2}) \Big)$$

where $(w_u, w_d, w_r, w_c, w_{st}, w_s)$ are the weights of the features, currently set at 1, and $k-1$ is the number of robots excluding the goalkeeper (r_k) whose mapping is predetermined.

4.4 Role Assignment

Eventually, a role assignment algorithm is required to search for the best possible mapping. In this section, we describe the algorithms we used in our coordination mechanism to perform this search over mappings in order for the team robots to be able to assume a role and undertake the corresponding behavior. Specifically, the algorithm is responsible to generate mappings, evaluate them using the utility function described above (Section 4.3) and compute a good, or optimal in some cases, mapping. Note that the candidate positions/roles may be more than the number of active robots, therefore the number of possible mappings may be extremely large. More formally, if $Robots = \{r_1, r_2, \ldots, r_k\}$ and $Positions = \{p_1, p_2, \ldots, p_n\}$ with $k \leq n$, the number of possible mappings is:

$$M = \binom{n-1}{k-1}(k-1)! = \frac{(n-1)!}{(k-1)!(n-k)!}(k-1)! = \frac{(n-1)!}{(n-k)!}$$

Essentially, we choose $k-1$ out of $n-1$ positions and we take all their permutations before assigning them to the $k-1$ robots. Note that the goalkeeper role and the goalkeeper player are excluded from the role assignment process. Clearly, the search space becomes huge even for small values of k and n. To cope with this problem, we introduce a pair of algorithms; one that performs exhaustive search and guarantees optimal solution, but is extremely expensive, and another one that performs only local search and guarantees only a good solution, but its complexity is independent of n.

4.4.1 Exhaustive Search

The exhaustive search algorithm is a simple search method, which enumerates all possible mappings, evaluates them, and keeps the best one found. Although, this method is very simple, fast for a small set of positions, and always finds the optimal solution, it is very slow when the position set grows above a certain threshold. Given that in our SPL coordination problem the number of robots is at most five ($k \leq 5$), using formations with only five positions ($n = 5$) leads up to only $M = 24$ mappings at most, which can be easily evaluated. By simply using three times as many positions ($n = 15$) in the formation, the number of possible mappings rises to $M = 24024$! The exhaustive search algorithm is summarized in Algorithm 1.

4.4.2 Particle Swarm Optimization

In order to overcome the high complexity of the exhaustive search algorithm, when the space of mappings grows large, we also introduce a different approach which enables the robots to compute a suboptimal, although fairly good, solution independently of the number of mappings. This alternative algorithm is based on the Particle Swarm Optimization

Algorithm 1 Exhaustive Search

```
Input: $Positions = \{p_1, \dots, p_{n-1}\}$, $Robots = \{r_1, \dots, r_{k-1}\}$, Utility function $U$
Output: best mapping $map_{best}$
 1: $M = \frac{(n-1)!}{(n-k)!}$
 2: $Mappings = \{map_i : i = 1, \dots, M,\ map_i$ is a valid mapping $Robots \mapsto Positions\}$
 3: $maxU = -\infty$
 4: for all $map_i \in Mappings$ do
 5:     $mapCost = U(map_i)$
 6:     if $(mapCost > maxU)$ then
 7:         $maxU = mapCost$
 8:         $map_{best} = map_i$
 9:     end if
10: end for
11: return $map_{best}$
```

(PSO) algorithm (Section 2.4) adjusted to our problem. Specifically, the algorithm uses particles to represent mappings, thus each particle assigns target position coordinates to each robot. More formally, we define as particle a tuple $\mathbf{x}_i = (x_1, y_1, \dots, x_{k-1}, y_{k-1})$, where $k-1$ is the number of team robots and $(x_i,\ y_i)$ are the coordinates of the target position of robot i, and therefore $2(k-1)$ is the dimension of the search space. Note that the goalkeeper is not participating in the role assignment process. The algorithm initializes the set of particles randomly by choosing random target positions from the set of candidate positions for each robot and then iteratively updates their velocity and moves them in the search space in order to discover the best possible solution. As an evaluation function we use the utility function described in Section 4.3.

Let S be the number of particles in the swarm, each having a position $\mathbf{x}_i \in \mathbb{R}^{2(k-1)}$ in the search space and a velocity $\mathbf{v}_i \in \mathbb{R}^{2(k-1)}$. Additionally, let $\mathbf{x}_i^* \in \mathbb{R}^{2(k-1)}$ be the best known position of particle i and let $\mathbf{g}^* \in \mathbb{R}^{2(k-1)}$ be the best known position of the entire swarm. Given these definitions, we summarize the PSO algorithm for role assignment in Algorithm 2. In our implementation, we chose a constant value of 2 for the learning parameters (acceleration constants) α and β and a constant-valued inertia function θ equal to 0.5.

As you probably noticed, the PSO algorithm is using randomness to discover new,

Algorithm 2 Particle Swarm Optimization

```
Input: $Positions = \{p_1, \dots, p_{n-1}\}$, $Robots = \{r_1, \dots, r_{k-1}\}$, Utility function $U$,
       Number of particles $S$, Number of iterations $I$
Output: global best mapping $\mathbf{g}^*$ of the entire swarm
 1: $\mathbf{g}^* = \vec{\mathbf{0}}$
 2: for $i = 1$ to $S$ do
 3:   Initialize particle $\mathbf{x}_i$ using a randomly selected set of positions from $Positions$
 4:   $\mathbf{x}_i^* \leftarrow \mathbf{x}_i$
 5:   if $\big(U(\mathbf{x}_i^*) > U(\mathbf{g}^*)\big)$ then
 6:     $\mathbf{g}^* \leftarrow \mathbf{x}_i^*$
 7:   end if
 8:   $\mathbf{v}_i = \vec{\mathbf{0}}$
 9: end for
10: for $iteration = 1$ to $I$ do
11:   for $i = 1$ to $S$ do
12:     for $d = 1$ to $2(k-1)$ do
13:       $\epsilon_1 \sim \mathbb{U}(0,1)$, $\epsilon_2 \sim \mathbb{U}(0,1)$
14:       $\mathbf{v}_{i,d} \leftarrow \theta \mathbf{v}_{i,d} + \alpha \epsilon_1 \big(\mathbf{g}_d^* - \mathbf{x}_{i,d}\big) + \beta \epsilon_2 \big(\mathbf{x}_{i,d}^* - \mathbf{x}_{i,d}\big)$
15:     end for
16:     Update: $\mathbf{x}_i \leftarrow \mathbf{x}_i + \mathbf{v}_i$
17:     if $\big(U(\mathbf{x}_i) > U(\mathbf{x}_i^*)\big)$ then
18:       $\mathbf{x}_i^* \leftarrow \mathbf{x}_i$
19:       if $\big(U(\mathbf{x}_i^*) > U(\mathbf{g}^*)\big)$ then
20:         $\mathbf{g}^* \leftarrow \mathbf{x}_i^*$
21:       end if
22:     end if
23:   end for
24: end for
25: return $\mathbf{g}^*$
```

probably better, solutions. Therefore, because each robot in the team is running the algorithm locally, there is a high probability for the individual solutions to differ. To enforce consistency, after the PSO algorithm has run, each robot broadcasts the best mapping found along with its utility value to the other robots. All robots gather these mappings and adopt the one with the highest utility value to ensure that role assignment is consistent within the team.

Chapter 5

Implementation

5.1 Matlab Simulation

Before integrating our coordination mechanism on our Monas software architecture for execution on the real robots, we implemented a simulation in Matlab to test the feasibility and the effectiveness of our approach.

We initially implemented the formation generator in a graphical user interface, so that the formation is plotted dynamically on a depiction of the actual SPL 2013 field (Figure 4.2), as the user moves the ball at different locations around the field. This simulation enabled us to visualize our ideas, evaluate the outcome, and tune the parameters, constants, and percentages used in the algorithm. Thereafter, we implemented the utility function and both role assignment algorithms (Exhaustive Search and PSO) to observe and evaluate their behavior. The graphical user interface was enhanced with the option of executing one of the two role assignment algorithms and visualizing their outcome. This Matlab simulation is certainly not fully representative of the actual implementation on the real robots, since it assumes an ideal environment without noise and makes use of ample computational resources, when in fact this is not true. Nonetheless, it proved to be sufficient to get a first glimpse of whether our coordination approach would work.

Figure 5.1 (left) shows the visual output of a generated formation. In this test scenario, four robots displayed in blue, plus the goalkeeper who is not participating in the coordination process, are about to coordinate using the exhaustive search algorithm. In Figure 5.1 (right) the robots have found the optimal role assignment, which is marked with blue lines on the graphical user interface. Additionally, in Figure 5.2 (left) the resulting formation with more candidate positions for the four robots having the same initial poses is shown. In Figure 5.2 (right) the robots have run the PSO algorithm and the resulting role assignment is marked again with blue lines. The careful observer can easily notice through this simulation and graphical user interface that the outcome of our approach yields reasonable results.

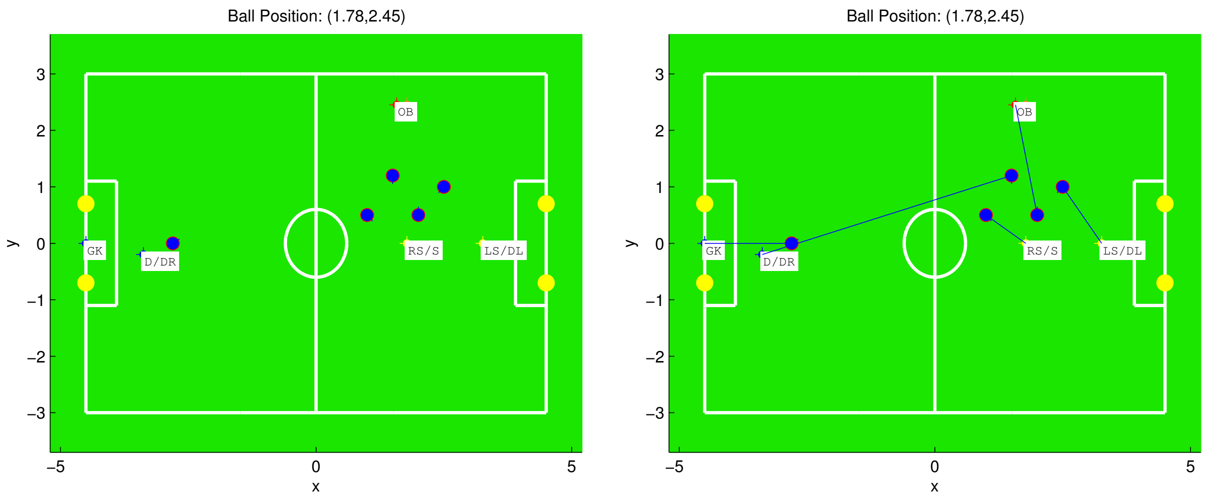

Figure 5.1: Formation (left) and role assignment (right) using Exhaustive Search

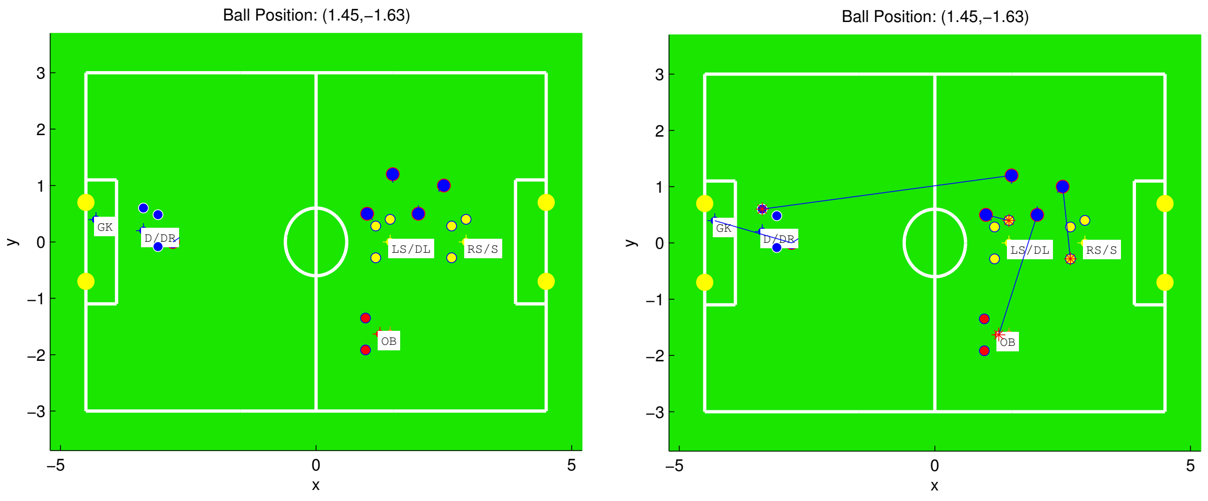

Figure 5.2: Formation (left) and role assignment (right) using PSO

5.2 The `Behavior` Monas Activity

In order to implement our approach on the real NAO robots, we developed our coordination mechanism and integrated it on the Monas `Behavior` activity. This activity is responsible for the high-level behavior of our robots, it operates on the outputs of lower-level activities and decides which one is the most appropriate action to be executed next. The `Behavior` activity is currently executed on the `KVision` agent and is specified inside our `agents.xml` configuration file that determines which activities will be running on the robot and at what frequency. Currently is being executed at a frequency of 25Hz (25 times per second).

The high frequency of execution implies that coordination must take place in less than 40 milliseconds. To understand the significance of this constraint we must take into account the limited processing power available on the robot. We need to keep in mind that we are developing software for an embedded platform with limited computational resources, which are shared among all programmed operations, as well as middleware operations. Hence, the workload of each activity must be in accordance with the total workload on the robot. Given these constraints, we paid attention to the real-time performance of our implementation. It turns out that the execution of exhaustive search was within limits for formations up to five positions/roles and up to five active robots. On the other hand, the execution of the PSO algorithm is independent from the number of positions/roles and its real-time performance essentially depends on the number of iterations and the number of particles, assuming up to five active robots. Thus, to meet the real-time requirements stated above we had to limit the number of iterations to 10 and the number of particles to 10. Larger numbers would increase the effectiveness of the PSO algorithm, however at the cost of increasing the computational load and thus the overhead of the whole robot, resulting on under-activity of the other activities.

Despite the real-time performance of our coordination mechanism, it is clear that coordination needs not take place at such a high frequency. Re-coordination is really necessary only when there is significant change in the game state. Therefore, our coordination mechanism executes, if at least one of the following two conditions is met:

- the estimated global ball location has changed significantly (by more than 70cm) since the latest coordination.

- a significant time interval (currently 10 seconds) since the latest coordination has passed.

The change in the estimated global ball position is computed from the information in the shared world model. Since this information is shared among all robots, they will simultaneously (more or less) notice the change in ball location and coordinate (almost) in sync. For the second condition the robots monitor the global game timer provided by the SPL game controller. In case some robot cannot listen to the network, and thus to the game controller, or some message is lost, then it also checks a local timer to be able to figure out when the specified time interval has passed. Using the global game timer ensures that the robots will coordinate (more or less) in sync.

Chapter 6

Results

In this chapter we present the results of our work and show how it benefits the rest of our RoboCup team. The best approach to show that a framework, like the one proposed, works well would be to compare the functionality and effectiveness of a team using our coordination mechanism to a team which has no sophisticated coordination and record the results. For example, in a number of games between the two teams, one can record how many times each team has won. Unfortunately, in our case this comparison cannot be performed, because we do not have the means to conduct such large-scale experiments, i.e. enough functional robots to form two teams and sufficient time (and patience) to run a large number of games. Thus, we could not provide such quantitative results, yet we will provide qualitative results by presenting certain scenarios that could occur during a robotic soccer game and compare them to our Matlab simulation output on the same scenarios. This way we can show that our approach can be really useful and works well even in the real, noisy world.

6.1 Coordination Performance

The first step in our evaluation was to compare the two coordination algorithms and show the actual difference in computational overhead. To this end, we measured the actual execution time of each algorithm for producing a solution given a certain number of positions and a fixed number of robots ($k = 5$) across all the runs. In Figure 6.1 we can see the results of this experiment both in terms of execution time and solution quality. The results clearly indicate that the PSO algorithm is much faster after a specific

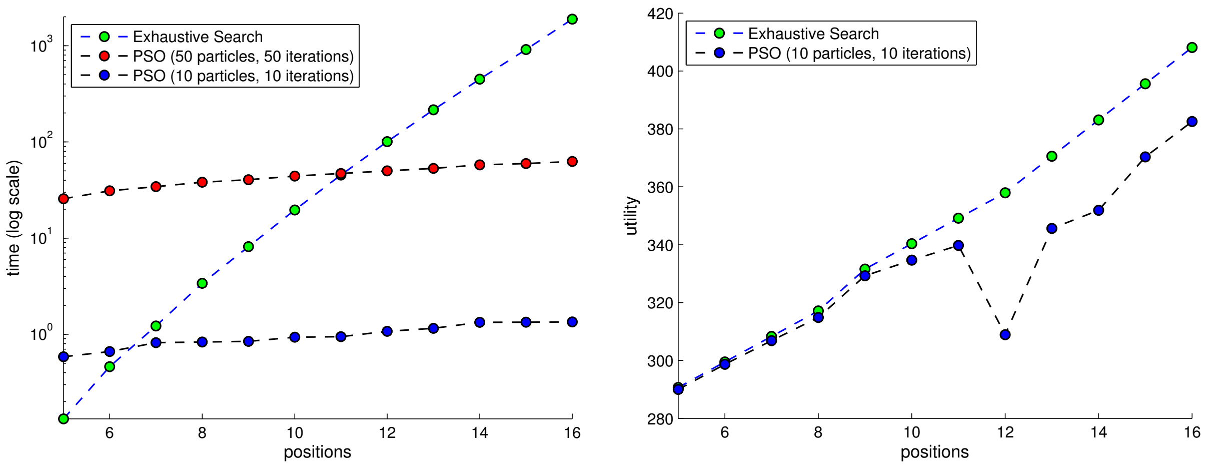

Figure 6.1: Execution time (left) and utility of mapping found (right) against positions

threshold, yet without compromising the quality of solutions compared to the optimal ones.

We currently perform role assignment using the exhaustive search algorithm in our software architecture, because we are generating formations with up to five positions/roles, thus it is fast and always finds the optimal solution, yet we may switch to the PSO algorithm in the future to perform role assignment on complex formations which may be produced in a more sophisticated way.

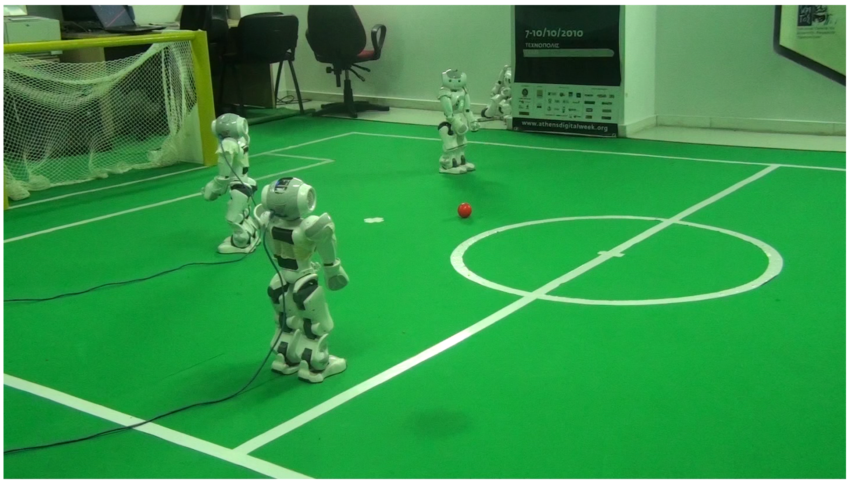

Figure 6.2: Scenario I: Setup for role assignment using the exhaustive search algorithm

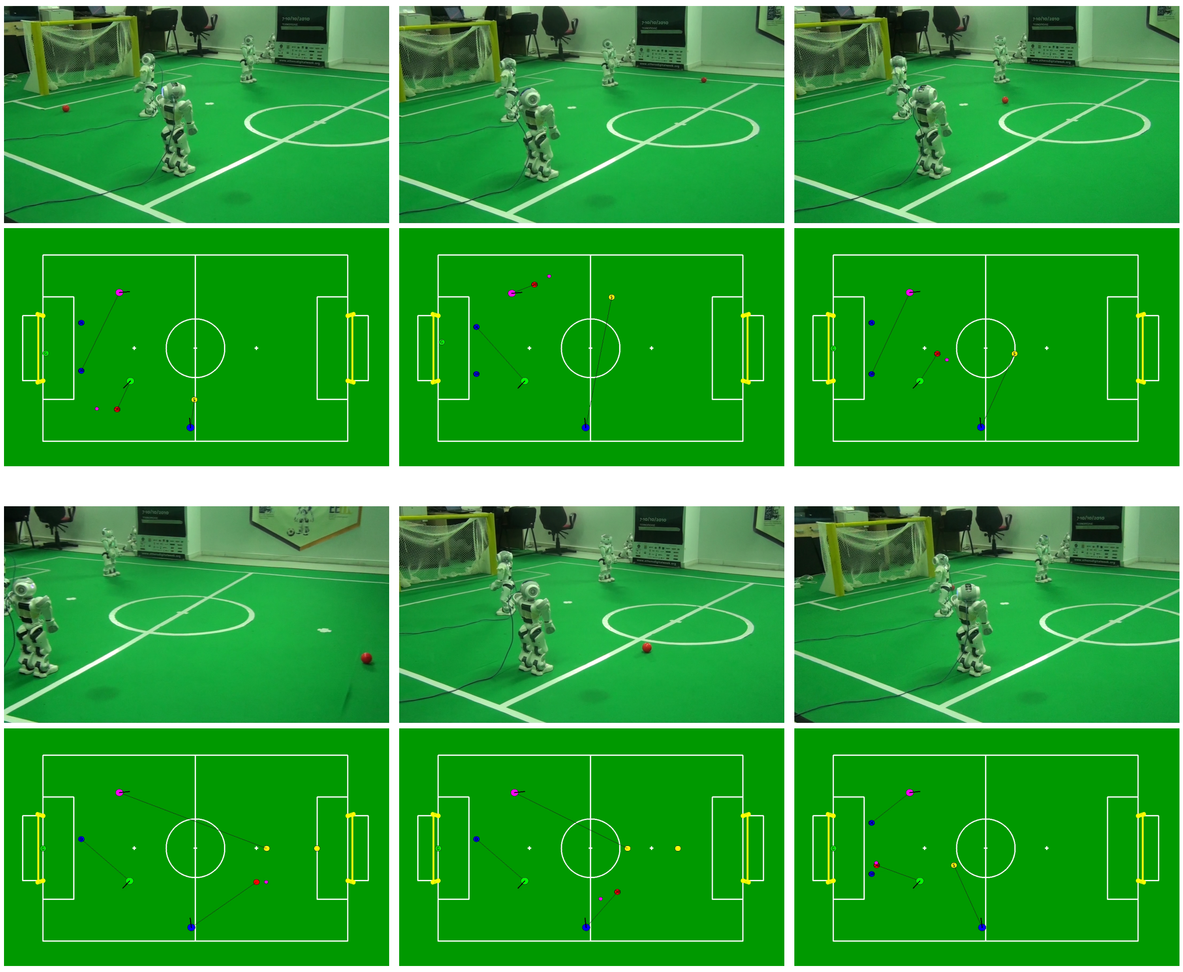

Figure 6.3: Scenario I: true state (top), formation and role assignment (bottom)

6.2 Scenario I: *Coordination using Exhaustive Search*

Our first goal is to check the functionality of our coordination mechanism on a simple scenario generating five candidate positions and performing role assignment using the exhaustive search algorithm. In this scenario (Figure 6.2) there are three stationary robots in the field facing at different directions. One is placed at the bottom right corner of the own field side near the middle and the side lines; the other robot is placed close to the opposite side line almost aligned with the penalty mark; the last one is placed near the own goal area aligned to the right goalpost. During this scenario, the ball is placed to a number of different locations in the field, so that each time at least one robot can

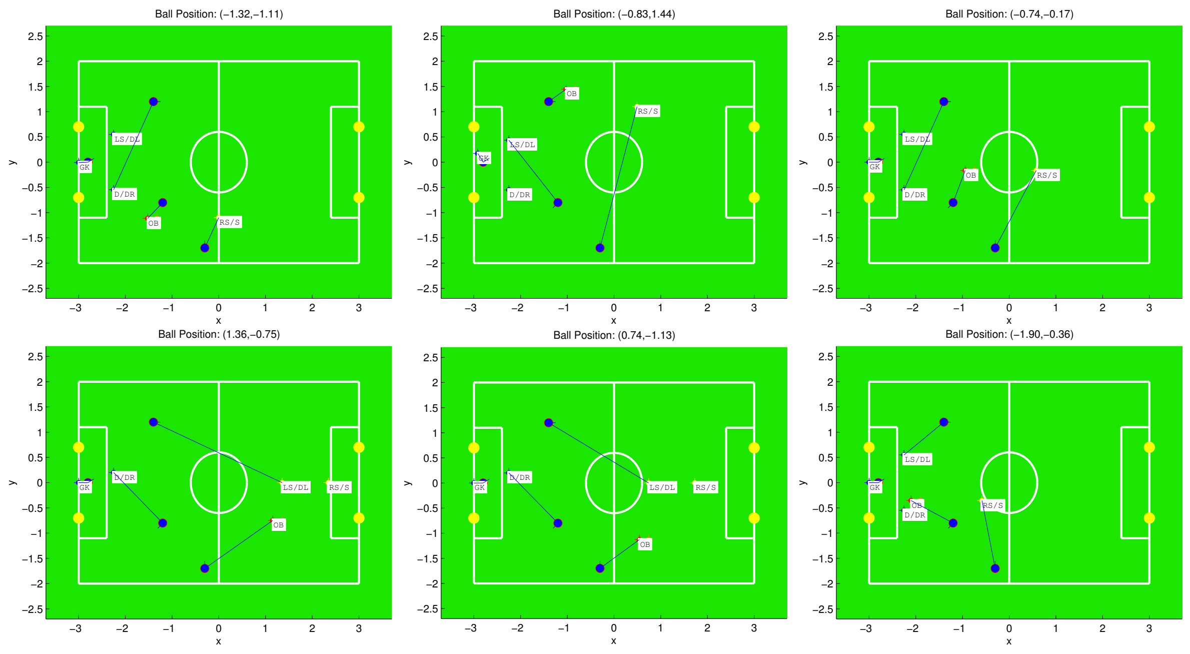

Figure 6.4: Scenario I: Formation and role assignment using the Matlab simulation

see it (to maintain a local estimate and update the shared world model). We used our graphical tool KMonitor [17] in order to check the formation generated as well as the role assignment that was decided by the robots each time the ball was moved.

In Figure 6.3 and Figure 6.4 we present formation and role assignment output of our coordination mechanism for the different ball locations of this scenario on real robots and simulation respectively. As shown by these figures, the formations and role assignments are almost the same in all cases. This is highly important because, it indicates that our coordination mechanism performs robustly even in a noisy real environment.

6.3 Scenario II: *Coordination using PSO*

Our second goal is to test the functionality of our coordination mechanism on a scenario generating multiple candidate positions in the formation (12 in this scenario) and performing role assignment using the PSO algorithm. In this scenario, we placed the robots on the same positions as in Scenario I (Figure 6.5). Then, we placed the ball to a number of different locations in the field, so that each time at least one robot can see it, as we did in Scenario I. Again, we used KMonitor [17] in order to check the formation generated

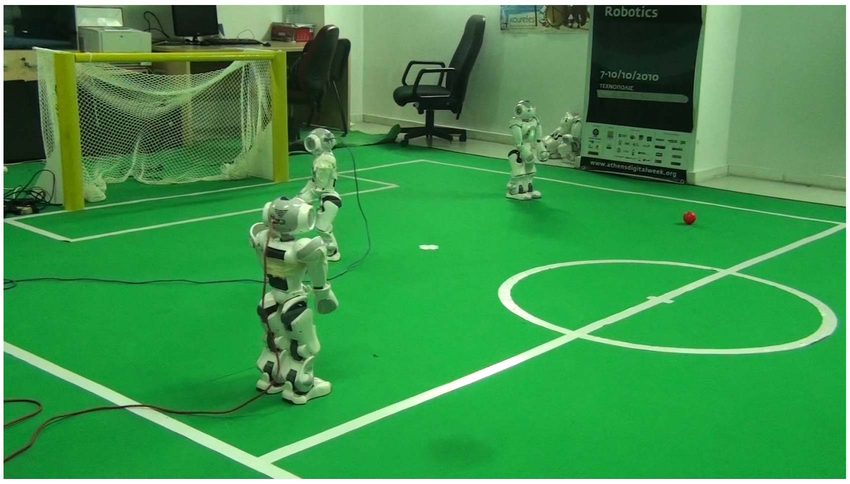

Figure 6.5: Scenario II: Setup for role assignment using the PSO algorithm

as well as the role assignment decided by the robots each time the ball was moved. As shown in Figure 6.6, the outcome of the PSO algorithm is fairly good and reasonable for each formation generated, even though it is in general suboptimal. Unfortunately, in this scenario their is no point in presenting the corresponding Matlab simulation results, since the PSO algorithm uses randomness and outcomes typically vary on each run.

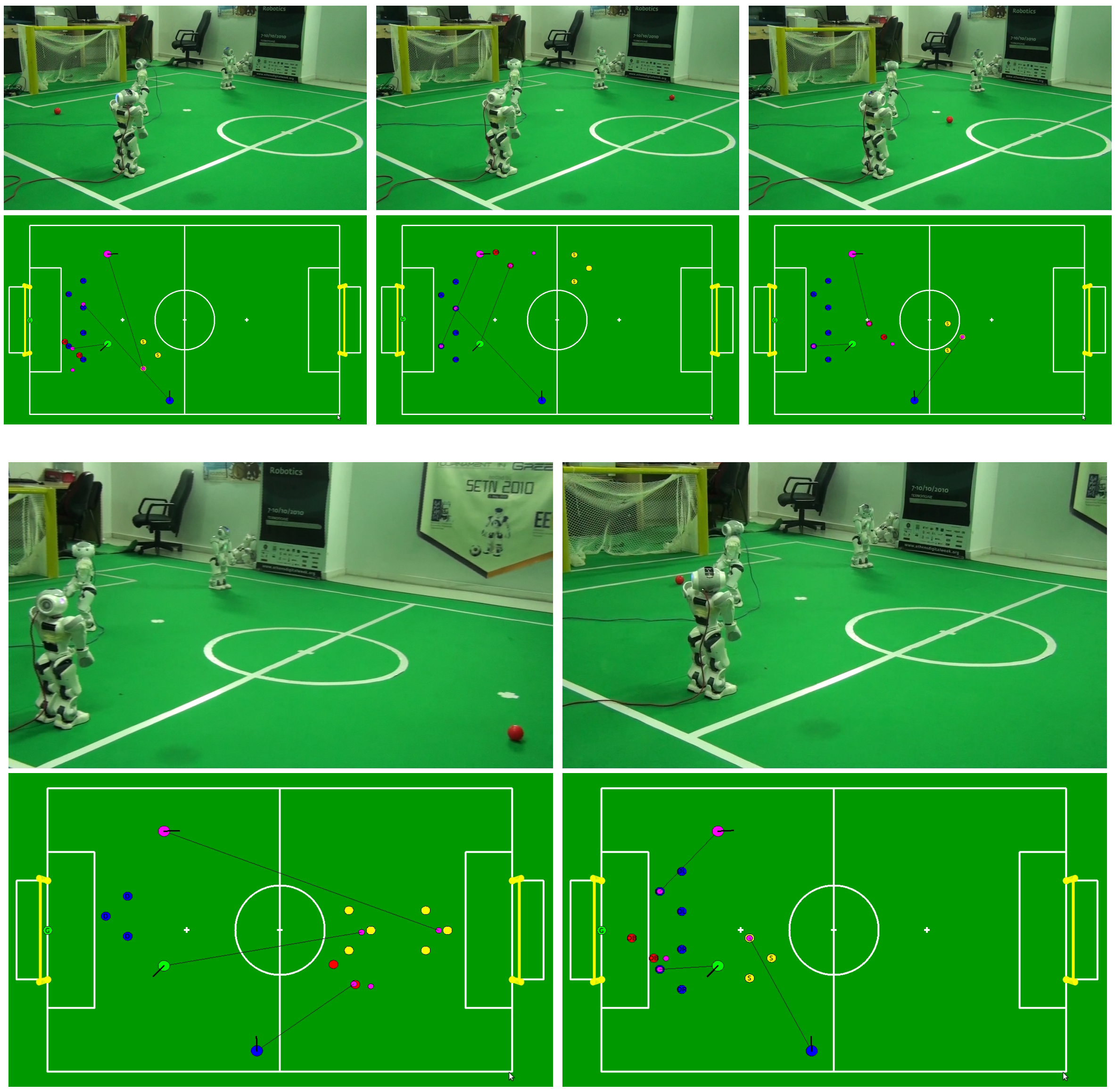

Figure 6.6: Scenario II: true state (top), formation and role assignment (bottom)

Chapter 7

Conclusion and Future Work

7.1 Conclusion

This thesis summarizes the coordination mechanism implemented and incorporated into the software architecture and repository of the RoboCup team Kouretes. As explained in Chapter 4, the end result of our work is a high-level coordination mechanism, which takes the output of global state estimation and performs dynamic positioning and role assignment. The outcome of our algorithm is a mapping of robots to positions/roles of a certain formation. As shown in Chapter 6, our work proves useful in common scenarios during RoboCup games and provides the opportunity of building extra features that require our work as a prerequisite. Our coordination mechanism also creates room for further development of new features. Some of them are briefly discussed below.

7.2 Future Work

7.2.1 Machine Learning

One of the most important features that could be implemented is a machine learning approach to inferring values for the scaling constants of the multi-attribute utility function presented in Section 2.3, which are currently set to 1. Scaling constants reflect a preference on the attributes. Thus, it could possibly improve the role assignment output and team's overall performance, if there was a way to learn a set of values which are expected to behave better in choosing assignments which lead to winning games.

Since such a learning approach requires carrying out a large number of games, it is relatively difficult to infer such a set of scaling constants in practice. Therefore, an alternative solution would be to simulate robotic soccer games on a simulator for the purposes of learning. The simulation is certainly not fully representative of the actual implementation on the robots, because it assumes an ideal environment without noise, when in fact this is not true. Nonetheless, the constants learned in the simulation maybe used as baseline for further training on the actual robots.

7.2.2 Action-Level Coordination

This thesis approached the coordination process at the level of dynamic positioning and role assignment. Other coordination methods operate at the action level and allow the players to directly coordinate their individual actions. However, such methods depend on existing roles, formations, and complex action skills, such as passing, dribbling, strategic movements, and even opponent recognition and modeling. Therefore, our mechanism could be extended to incorporate such types of coordination, provided there exist corresponding skills at a lower level. For example, after the robotic team has generated a formation and assumed roles using our approach, it could execute a process of team planning to coordinate individual actions between certain team players to come closer to real human strategies.

7.2.3 Opponent Modeling

Currently, our team does not have the ability to visually recognize opponent robots. However, when this feature is implemented and opponent positions are added to global state estimation, our approach could modified to include evaluation features for positioning, which would take advantage of this additional information for generating even more strategic formations. This could lead to making even better and more strategic team planning during the game, such as generating formations which would favor the team to score, avoiding passing the ball to a robot blocked by opponents, blocking opponent players from scoring, choosing a kicking direction towards the opponent goal that clears all players (opponents and teammates), or avoiding collisions with and obstructions by opponent robots.

References

[1] RoboCup SPL Technical Committee: Standard Platform League rule book (2013) Only available online: www.tzi.de/spl/pub/Website/Downloads/Rules2013.pdf. xi, 6, 32

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

[3] Gouaillier, D., Blazevic, P.: A mechatronic platform, the Aldebaran Robotics humanoid robot. In: Proceedings of the 32nd IEEE Annual Conference on Industrial Electronics (IECON). (2006) 4049–4053 6

[4] Aldebaran Robotics: NAO documentation (2012) Only available online: www.aldebaran-robotics.com/documentation. 7

[5] Paraschos, A.: Monas: A flexible software architecture for robotic agents. Diploma thesis, Technical University of Crete, Greece (2010) 11

[6] Orfanoudakis, E.: Reliable object recognition for the RoboCup domain. Diploma thesis, Technical University of Crete, Greece (2011) 11

[7] Chatzilaris, E.: Visual-feature-based self-localization for robotic soccer. Diploma thesis, Technical University of Crete, Greece (2009) 12

[8] Kargas, N.: Robust localization for the RoboCup standard platform league. Diploma thesis, Technical University of Crete, Greece (2013) 12

[9] Pavlakis, N.: Cooperative global game state estimation for the RoboCup standard platform league. Diploma thesis, Technical University of Crete, Greece (2013) 12

[10] Kyranou, I.: Path planning for NAO robots using an egocentric polar occupancy map. Diploma thesis, Technical University of Crete, Greece (2012) 13

[11] Tzanetatou, D.: Interleaving of motion skills for humanoid robots. Diploma thesis, Technical University of Crete, Greece (2012) 13

[12] Kofinas, N., Orfanoudakis, E., Lagoudakis, M.G.: Complete analytical inverse kinematics for NAO. In: Proceedings of the 13th International Conference on Autonomous Robot Systems and Competitions (ROBOTICA). (2013) 13

[13] Kofinas, N.: Forward and inverse kinematics for the NAO humanoid robot. Diploma thesis, Technical University of Crete, Greece (2012) 13

[14] Paraschos, A., Spanoudakis, N.I., Lagoudakis, M.G.: Model-driven behavior specification for robotic teams. In: Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS). (2012) 14

[15] Topalidou-Kyniazopoulou, A., Spanoudakis, N.I., Lagoudakis, M.G.: A case tool for robot behavior development. In: RoboCup 2012: Robot Soccer World Cup XVI. Volume 7500 of Lecture Notes in Computer Science. Springer (2013) 225–236 14

[16] Topalidou-Kyniazopoulou, A.: A CASE (computer-aided software engineering) tool for robot-team behavior-control development. Diploma thesis, Technical University of Crete, Greece (2012) 14

[17] Karamitrou, M.: KMonitor: Global and local state visualization and monitoring for the RoboCup SPL league. Diploma thesis, Technical University of Crete, Greece (2012) 14, 54

[18] Vazaios, E.: Narukom: A distributed, cross-platform, transparent communication framework for robotic teams. Diploma thesis, Technical University of Crete, Greece (2010) 14

[19] Jonathan E. Ingersoll, J.: Theory of Financial Decision Making. Rowman and Littlefield (1987) Chapter 1. 15

[20] Abbas, A.E.: General decompositions of multiattribute utility functions with partial utility independence. Journal of Multi-Criteria Decision Analysis **17**(1–2) (2010) 37–59 17

[21] Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proceedings of the IEEE International Conference on Neural Networks. Volume 4. (1995) 1942–1948 18

[22] Kennedy, J., Eberhart, R.: Swarm intelligence. Academic Press (2001) 18

[23] Wikipedia: Particle swarm optimization — wikipedia, the free encyclopedia (2013) `http://en.wikipedia.org/wiki/Particle_swarm_optimization`. 19

[24] MacAlpine, P., Barrera, F., Stone, P.: Positioning to win: A dynamic role assignment and formation positioning system. In: RoboCup-2012: Robot Soccer World Cup XVI. Lecture Notes in Artificial Intelligence. Springer (2013) 25

[25] Kok, J.R., Spaan, M.T.J., Vlassis, N.A.: Non-communicative multi-robot coordination in dynamic environments. Robotics and Autonomous Systems **50**(2–3) (2005) 99–114 25

[26] Vail, D., Veloso, M.: Multi-robot dynamic role assignment and coordination through shared potential fields. In: Multi-Robot Systems. Kluwer (2003) 26