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

Cooperative Global Game State Estimation for the RoboCup Standard Platform League

Nikolaos Pavlakis

Thesis Committee
Associate Professor Michail G. Lagoudakis (ECE)
Professor Minos Garofalakis (ECE)
Assistant Professor Aggelos Bletsas (ECE)

Chania, June 2013

Πολυτεχνειο Κρητης

Σχολη Ηλεκτρονικων Μηχανικων και Μηχανικων Υπολογιστων

Συνεργατική Εκτίμηση Καθολικής Κατάστασης Παιχνιδιού για το Πρωτάθλημα Standard Platform του RoboCup

Νικόλαος Παυλάκης

Εξεταστική Επιτροπή

Αναπ. Καθ. Μιχαήλ Γ. Λαγουδάκης (ΗΜΜΥ)

Καθ. Μίνως Γαροφαλάκης (ΗΜΜΥ)

Επικ. Καθ. Άγγελος Μπλέτσας (ΗΜΜΥ)

Χανιά, Ιούνιος 2013

Abstract

RoboCup is an international competition that promotes research in multi-robot teams operating in dynamic environments. In the Standard Platform League (SPL) all teams use identical robots, namely the Aldebaran NAO humanoid robots. SPL robots have limited perceptual abilities, meaning that they can perceive only parts of their environment, due to directed vision, existence of dynamic obstacles, and high uncertainty in locomotion actions and recognition ability. This fact implies that any single robot on its own can only form a partial and possibly inaccurate estimate about the state of the world around it. The importance of fusing these local beliefs into a single and consistent global belief can be easily understood, considering that such information enables the possibility of collaborative team planning and coordination. This thesis addresses the problem of global game state estimation over a team of robots from their local state estimates. Our approach takes as input each robot's local belief about the world around it (pose within the field, location of the ball, etc.) and combines all these local beliefs to estimate the most probable global state of the world consistent with these local beliefs. In order to generate this information in a principled way, an Extended Kalman Filter (EKF) is employed with appropriate state transition and observation models, applying linearization where needed. An important aspect of this work is our implementation within the software framework of team Kouretes, which allows for decentralized and asynchronous belief updates and is optimized for real-time, on-board execution on the NAO robots. The benefits of global game state estimation are demonstrated on two common game scenarios: (a) cooperative, accurate estimation of ball location within the field from multiple, possibly inaccurate, local estimations and (b) ability to acquire missing ball information and walk towards the ball "seeing" it through the help of teammates. Our shared global state information mechanism is currently used in the development of collaborative strategies and coordinated planning.

Περίληψη

Το RoboCup είναι ένας διεθνής διαγωνισμός που προωθεί την έρευνα σε πολυ-ρομποτικές ομάδες που λειτουργούν σε δυναμικά περιβάλλοντα. Στο Πρωτάθλημα Standard Platform (SPL) όλες οι ομάδες χρησιμοποιούν πανομοιότυπα ρομπότ και συγκεκριμένα τα ανθρωποειδή ρομπότ Aldebaran NAO. Τα ρομπότ που χρησιμοποιούνται στο SPL έχουν περιορισμένες ικανότητες αντίληψης, πράγμα που σημαίνει ότι μπορούν να αντιληφθούν μόνο μέρος του περιβάλλοντός τους, λόγω κατευθυντικής όρασης, ύπαρξης δυναμικών εμποδίων, και υψηλής αβεβαιότητας στις ενέργειες μετακίνησης και στην αναγνωριστική τους ικανότητα. Το γεγονός αυτό σημαίνει ότι κάθε ένα ρομπότ από μόνο του μπορεί να διαμορφώσει μόνο μια μερική και ενδεχομένως ανακριβή εκτίμηση της κατάστασης του κόσμου γύρω του. Η σημασία της σύντηξης αυτών των τοπικών πεποιθήσεων σε μια ενιαία και συνεπή πεποίθηση που αφορά όλο το περιβάλλον μπορεί να γίνει εύκολα κατανοητή, λαμβάνοντας υπόψη ότι τέτοιου είδους πληροφορία δίνει στην ομάδα τη δυνατότητα συνεργατικού ομαδικού σχεδιασμού και συντονισμού. Η παρούσα διπλωματική εργασία εξετάζει το πρόβλημα της καθολικής εκτίμησης της κατάστασης ενός παιχνιδιού στο SPL για μια ομάδα από ρομπότ μέσω των τοπικών τους εκτιμήσεων. Η προσέγγισή μας λαμβάνει ως είσοδο την τοπική πεποίθηση του κάθε ρομπότ για τον κόσμο γύρω του (τη θέση του και τον προσανατολισμό του μέσα στο γήπεδο, τη θέση της μπάλας, κλπ.) και συνδυάζει όλες αυτές τις τοπικές πεποιθήσεις για να εκτιμήσει την πιο πιθανή καθολική κατάσταση του κόσμου σύμφωνα με αυτές. Για να δημιουργηθεί αυτή η πληροφορία με έναν αξιωματικό τρόπο, χρησιμοποιείται ένα Extended Kalman Filter (EKF) με κατάλληλα μοντέλα μετάβασης και παρατήρησης, εφαρμόζοντας γραμμικοποίηση όπου χρειάζεται. Μια σημαντική πτυχή αυτής της εργασίας είναι η υλοποίησή της στο πλαίσιο του λογισμικού της ομάδας «Κουρήτες», η οποία επιτρέπει αποκεντρωμένες και ασύγχρονες ενημερώσεις πεποίθησης και έχει βελτιστοποιηθεί για εκτέλεση σε πραγματικό χρόνο, πάνω στο ρομπότ NAO. Τα οφέλη της καθολικής εκτίμησης κατάστασης του παιχνιδιού επιδεικνύονται σε δύο συνηθισμένα σενάρια παιχνιδιού: (α) την συνεταιριστική, ακριβή εκτίμηση της θέσης της μπάλας μέσα στο γήπεδο από πολλές, ενδεχομένως ανακριβείς, τοπικές εκτιμήσεις και (β) την ικανότητα απόκτησης μη υπάρχουσας πληροφορίας για την μπάλα και προσέγγισής της «παρατηρώντας» την με τη βοήθεια των συμπαικτών. Ο μηχανισμός καθολικής εκτίμησης κατάστασης που δημιουργήθηκε χρησιμοποιείται σε έρευνα που στοχεύει στην ανάπτυξη συνεργατικών στρατηγικών και συντονισμένου σχεδιασμού.

Acknowledgements

First of all, I would like to thank my advisor Michail G. Lagoudakis for his inspiration and guidance.

Next, I would like to thank Manolis Orfanoudakis (a.k.a. vosk) for his suggestions and help during the implementation of the work.

Next I would like to thank my parents for their awesome support and encouragement.

Team Kouretes, including older and newer members (N. Kofinas, A. Topalidou, M. Karamitrou, D. Janetatou, vosk, I. Kyranou, E. Chantzilaris, N. Kargas, E. Michelioudakis, S. Piperakis), played an important role during the duration of this thesis. We also had a lot of fun at RoboCup events, as well as in our Lab (a.k.a. "ypoga").

Last but not least, I would like to thank my friends with whom I had amazing moments during my stay in Chania. E. Alimpertis, N. Kofinas (he paid five Euros for the second reference), D. Iliou, K. Perros, and E. Soulas, thank you for being there for me whenever I need you! Along the way I also made some new friends with whom I also had some great times and I would like to thank them too. G. Demertzis, T. Demertzis, T. Magounaki ($\Theta = \Theta$), S. Nikolakaki (a.k.a. fiouki), and D. Paliatsa (a.k.a. koumpara) you are all awesome! (Please note that the names in this paragraph are arranged in alphabetical order so that nobody complains.)

As a hidden bonus, I would like to thank a person that always makes me smile (even without knowing it) and is an important part of my life. This "entry" exists for personal reasons, so that it always reminds me of this person.

Contents

List of Figures

LIST OF FIGURES

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 concern cooperative 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 concentrate on software development only. 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. In order for those problems to be tackled, all the agents of the team need to have a common belief about the global state of the environment. This belief must be a common reference

frame shared between all the team robots. Furthermore, apart from information sharing, it needs to provide error correction in order to account for local module errors.

The simplest example to illustrate the importance of this common global world state belief is the following. Imaging two players of the same team, both confident about their own location inside the field, but only one of them observing the ball. Even though the other player is not directly observing the ball (e.g. due to occlusion), if these two robots maintain a common belief about the surrounding environment, they will both share information about the location of ball and will be able to use this information for appropriate decision making. This thesis aims at addressing the problem of multi-agent belief fusion and information sharing which enables the possibility for collaborative team planning and coordination.

1.1 Thesis Contribution

The contribution of this thesis is the creation of a mechanism that addresses the problem of global game state estimation over a team of robots during a robotic soccer game. More specifically, this mechanism takes as input each robot's personal belief about the world around it (its location inside the field, the ball's location, teammate locations, opponent locations, etc.) and combines all these beliefs to estimate the most probable global state of the world. This means that the output of this mechanism is a state vector, which includes the poses (positions and orientations) of all robots of the team, the position of the ball, and the poses of opponent robots, if opponent robot recognition is available.

In order to generate and provide this information, this mechanism employs a filtering algorithm, and more specifically, an Extended Kalman Filter (EKF). A key component of this work is the definition of the appropriate EKF models (state transition model and observation model) and their linearization where needed. A second component is the implementation of a decentralized and asynchronous belief update method for the EKF, so that it can be used in real game situations where computational and network resources are limited.

The benefits of this method are numerous, yet the most important one is that all robots share a common belief about the global game state. This implies that some robots may consult this global belief to acquire information missing from their local beliefs. For example, a robot can obtain the location of the ball in the field even without observing it

on its own, as long as at least one of its teammates does so. This feature is very important since the domain of robotic soccer is characterized by partial observability, which implies that local beliefs by themselves cannot always contain accurate information about the global game state. Another great benefit of this method is that the robot team can exploit this shared global state information to develop collaborative strategies and coordinated planning, instead of relying on independent (egocentric) behaviors.

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 the Kalman Filter and the Extended Kalman Filter that is used in this approach. In Chapter 3 we define the problem of global game state estimation 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 SPL teams. In Chapter 4 we describe our approach in detail, including a justification for choosing an Extended Kalman Filter and the selection of the required transition and observation models. 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 global game state estimation 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, comply 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 [3], 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 [4] started in 2004. In August 2007 Nao officially

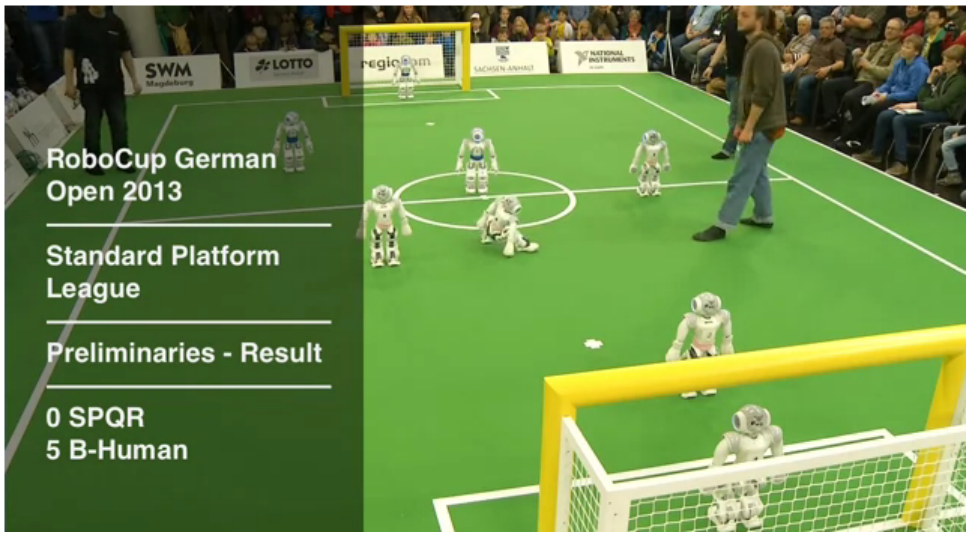

Figure 2.1: Standard Platform League at RoboCup German Open 2013

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) [5] 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 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

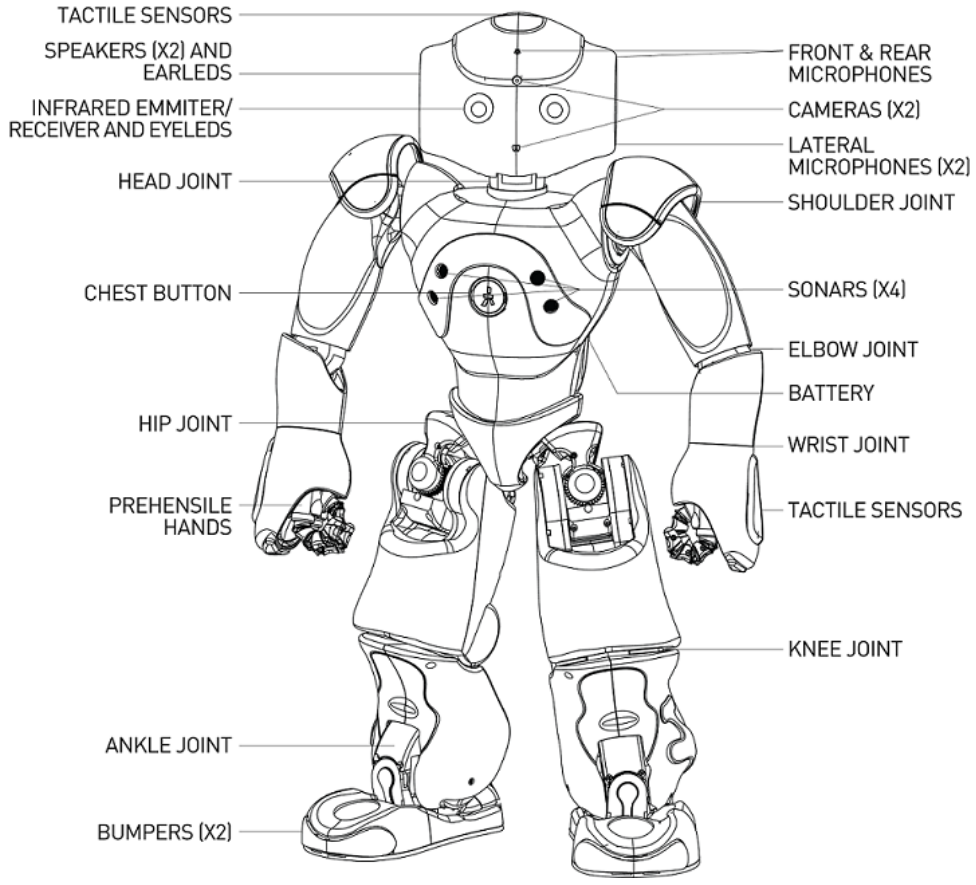

Figure 2.2: Aldebaran Nao robot (v3.3, academic edition) and its components

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 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

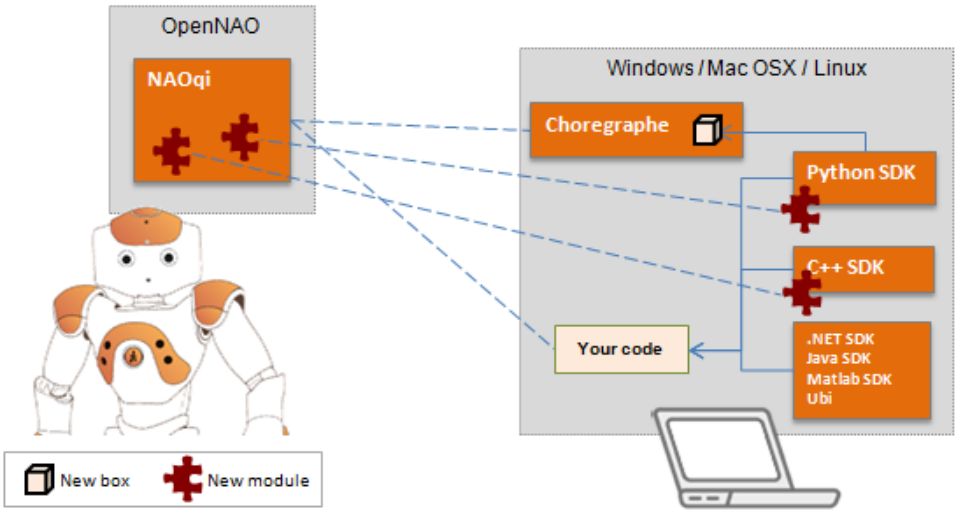

Figure 2.3: Embedded and desktop software for the Nao robot

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 ever since. The team's publicly-available code repository includes a custom software architecture, a custom communication framework, a graphical application for behavior

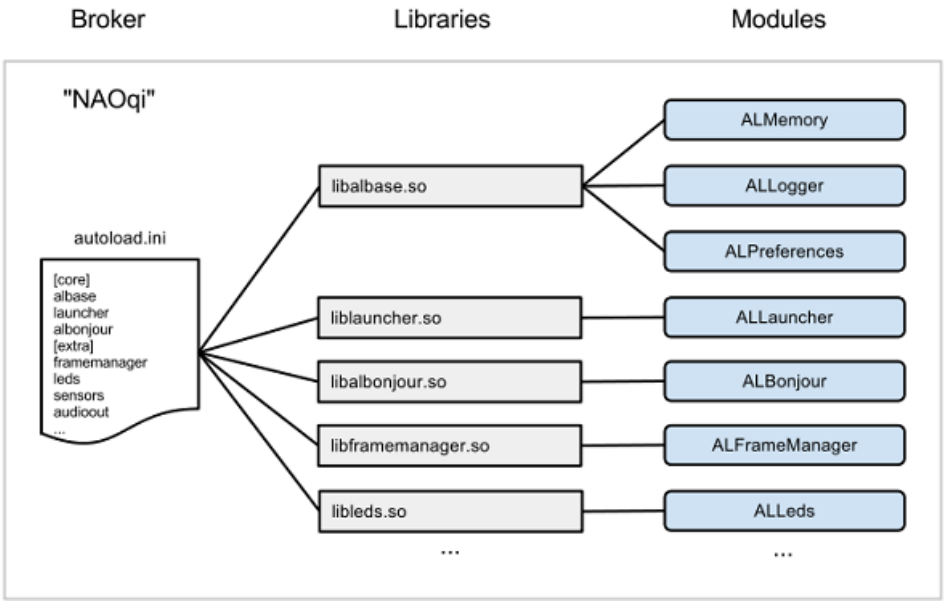

Figure 2.4: The NAOqi process

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 the RoboCup German Open 2012 competition in Magdeburg, in RoboCup Iran Open 2012 in Tehran, in RoboCup 2012 in Mexico City (Figure 2.5), in AutCup 2012 and in RoboCup Iran Open 2013 (Figure 2.6). In the most recent RoboCup 2012 competition, the team succeeded to proceed to the second round-robin round and rank among the top-16 SPL teams in the world.

Figure 2.5: Team Kouretes at RoboCup 2012 in Mexico City

2.2.1 Monas Software Architecture

Monas [6] 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` [7] is a light-weight image processing method for humanoid robots, via which Kouretes team has accomplished 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 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

Figure 2.6: Team Kouretes at RoboCup Iran Open 2013

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` [8] is the activity which used to realize Monte-Carlo localization (Particle Filters - PFs) and recently switched to using a Kalman Filter (KF). 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 KF. Belief update is performed using an odometry motion model for omni-directional 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 (KF).

- `SharedWorldModel` is the end product of this thesis; it combines the local beliefs of all robots to create a common and shared estimation of the current state of the world. Details will be provided in Chapters 4 and 5.

- `ObstacleAvoidance` [9] is the activity which accomplishes obstacle avoidance by first building a local 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.

- `Behavior` is the activity which implements the desired robotic behavior. It operates on the outputs of the `Vision`, `LocalWorldState`, and `ObstacleAvoidance` activities and decides which one is the most appropriate action to be executed next.

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

- `MotionController` [10] 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 [11, 12], 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 [13, 14, 15] are executed 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 [16] is a 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 [17] 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 Kalman Filtering

The Kalman filter [18], also known as linear quadratic estimation (LQE), is an algorithm that uses a series of measurements observed over time, containing noise (random variations) and other inaccuracies, and produces estimates of unknown variables that tend to be more precise than those based on a single measurement alone. More formally, the Kalman filter operates recursively on streams of noisy input data to produce a statistically optimal estimate of the underlying system state. The filter is named after Rudolf E. Kalman, one of the primary developers of its theory.

The Kalman filter is a set of mathematical equations that provides an efficient computational (recursive) solution of the least-squares method. The filter is very powerful in several aspects: it supports estimations of past, present, and even future states, and it can do so even when the precise nature of the modeled system is unknown.

2.3.1 Brief Description

The Kalman filter uses a system's dynamics model (e.g., physical laws of motion), known control inputs to that system, and multiple sequential measurements (from sensors) to form an estimate of the system's varying quantities (its state) that is better than the estimate obtained by using any one measurement alone. As such, it is a sensor fusion and data fusion algorithm.

All measurements and calculations based on models are estimates to some degree. Noisy sensor data, approximations in the equations that describe system changes, and external factors that are not accounted for introduce uncertainty about the inferred values

for a system's state. The Kalman filter averages a prediction of a system's state with a new measurement using a weighted average. The purpose of the weights is that values with less uncertainty are "trusted" more. The weights, also known as Kalman gain, are calculated taking into account the covariance of the estimation which relates to the estimated uncertainty of the prediction of the system's state and the uncertainty of the measurements. The result of the weighted average is a new state estimate that lies between the predicted and the measured state and has a lower estimated uncertainty than either alone. This process is repeated every time step, with the new estimate and its covariance informing the prediction used in the following iteration. This means that the Kalman filter works recursively and requires only the last "best guess", rather than the entire history, of a system's state to calculate a new state.

Since the certainty of the measurements is often difficult to measure precisely, it is common to discuss the filter's behavior in terms of gain. The Kalman gain is a function of the relative certainty of the measurements and current state estimate and can be "tuned" to achieve a certain level of performance. With a high gain, the filter places more weight on the measurements and thus follows them more closely. With a low gain, the filter follows the model predictions more closely, smoothing out noise but decreasing the responsiveness. At the extremes, a gain of one causes the filter to ignore the state estimate entirely, while a gain of zero causes the measurements to be ignored.

When performing the actual calculations for the filter (as discussed below), the multi-dimensional state estimate and covariances are coded into matrices to handle collectively the multiple dimensions involved in a single set of calculations. This allows for representation of linear relationships between different state variables (such as position, velocity, and acceleration) in any of the transition models or covariances.

2.3.2 Estimation Procedure

In order to use the Kalman filter to estimate the internal state of a process given only a sequence of noisy observations, one must model the process in accordance with the framework of the Kalman filter. This means specifying the following matrices: $\mathbf{F}_k$, the state-transition model; $\mathbf{H}_k$, the observation model; $\mathbf{Q}_k$, the covariance of the process noise; $\mathbf{R}_k$, the covariance of the observation noise; and sometimes $\mathbf{B}_k$, the control-input model, for each time-step, k.

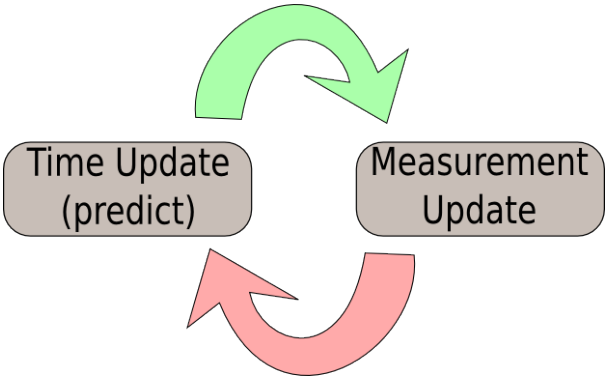

Figure 2.7: Kalman Filter phases

The Kalman filter addresses the general problem of trying to estimate the state $\mathbf{x} \in \Re^n$ of a discrete-time controlled process with control input $\mathbf{u} \in \Re^l$, which is governed by the linear stochastic difference equation:

$$\mathbf{x}_k = \mathbf{F}_k \mathbf{x}_{k-1} + \mathbf{B}_k \mathbf{u}_k + \mathbf{w}_k$$

using measurements $\mathbf{z} \in \Re^m$ which depend on the state, that is

$$\mathbf{z}_k = \mathbf{H}_k \mathbf{x}_k + \mathbf{v}_k$$

The random variables $\mathbf{w}_k$ and $\mathbf{v}_k$ represent the process and measurement noise respectively. They are assumed to be independent (of each other), white, and with normal probability distributions:

$$\mathbf{w} \sim N(0, \mathbf{Q})$$

$$\mathbf{v} \sim N(0, \mathbf{R})$$

The $n \times n$ matrix $\mathbf{F}_k$ relates the state at time step k to the state at step $k+1$, in the absence of either a control input or process noise. The $n \times l$ matrix $\mathbf{B}$ describes the effect of the control input $\mathbf{u}_k \in \Re^l$ on the state $\mathbf{x}_k$. The $m \times n$ matrix $\mathbf{H}_k$ relates the state to the measurement $\mathbf{z}_k$.

The Kalman filter is most often conceptualized as two distinct phases: *predict* and *update* (Figure 2.7). The *predict* phase uses the state estimate from the previous timestep to produce an estimate of the state at the current timestep. This predicted state estimate is also known as the a priori state estimate because, although it is an estimate of

the state at the current timestep, it does not include observation information from the current timestep. In the *update* phase, the current prediction is combined with current observation information to refine the state estimate. This improved estimate is termed the a posteriori state estimate.

Typically, the two phases alternate, with the prediction advancing the state until the next scheduled observation, and the update incorporating the observation. However, this is not necessary; if an observation is unavailable for some reason, the update may be skipped and multiple prediction steps may be performed. Likewise, if multiple independent observations are available at the same time, multiple update steps may be performed.

Predict:

$$\begin{aligned} \text{Predicted (a priori) state estimate} \quad & \hat{\mathbf{x}}_{k|k-1} = \mathbf{F}_k\hat{\mathbf{x}}_{k-1|k-1} + \mathbf{B}_{k-1}\mathbf{u}_{k-1} \\ \text{Predicted (a priori) estimate covariance} \quad & \mathbf{P}_{k|k-1} = \mathbf{F}_k\mathbf{P}_{k-1|k-1}\mathbf{F}_k^\top + \mathbf{Q}_k \end{aligned}$$

Update:

$$\begin{aligned} \text{Innovation or measurement residual} \quad & \tilde{\mathbf{y}}_k = \mathbf{z}_k - \mathbf{H}_k\hat{\mathbf{x}}_{k|k-1} \\ \text{Innovation (or residual) covariance} \quad & \mathbf{S}_k = \mathbf{H}_k\mathbf{P}_{k|k-1}\mathbf{H}_k^\top + \mathbf{R}_k \\ \text{Optimal Kalman gain} \quad & \mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{H}_k^\top\mathbf{S}_k^{-1} \\ \text{Updated (a posteriori) state estimate} \quad & \hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k\tilde{\mathbf{y}}_k \\ \text{Updated (a posteriori) estimate covariance} \quad & \mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_{k|k-1} \end{aligned}$$

2.3.3 Extended Kalman Filter

The primary drawback of the Kalman filter is that it offers optimal estimates only for linear system models with additive independent white noise in both the transition and the measurement models. Unfortunately, in engineering most systems are nonlinear, so some attempt was immediately made to apply this filtering method to nonlinear systems. The Extended Kalman Filter [18] which adapted techniques from calculus, namely multivariate Taylor Series expansions, to linearize about a working point became the working solution.

In the Extended Kalman filter, the state transition and observation models need not be linear functions of the state, but may instead be any differentiable functions.

$$\mathbf{x}_k = f(\mathbf{x}_{k-1}, \mathbf{u}_{k-1}) + \mathbf{w}_{k-1}$$

$$\mathbf{z}_k = h(\mathbf{x}_k) + \mathbf{v}_k$$

where $\mathbf{w}_k$ and $\mathbf{v}_k$ are the process and observation noises which are both assumed to be zero-mean multivariate Gaussian noises with covariance $\mathbf{Q}_k$ and $\mathbf{R}_k$ respectively.

The function f can be used to compute the predicted state from the previous estimate and similarly the function h can be used to compute the predicted measurement from the predicted state. However, f and h cannot be applied to the covariance directly. Instead a matrix of partial derivatives (the Jacobian) is computed. At each time step, the Jacobian is evaluated using the current predicted state. These matrices can be used in the Kalman filter equations. This process essentially linearizes the non-linear functions around the current state estimate.

Similarly to the linear Kalman filter, there are two phases: *predict* and *update*.

Predict:

Predicted (a priori) state estimate	$\hat{\mathbf{x}}_{k\|k-1} = \overline{\mathbf{F}}_k \hat{\mathbf{x}}_{k-1\|k-1} + \mathbf{B}_{k-1}\mathbf{u}_{k-1}$
Predicted (a priori) estimate covariance	$\mathbf{P}_{k\|k-1} = \overline{\mathbf{F}}_k \mathbf{P}_{k-1\|k-1} \overline{\mathbf{F}}_k^\top + \mathbf{Q}_k$

Update:

Innovation or measurement residual	$\tilde{\mathbf{y}}_k = \mathbf{z}_k - \overline{\mathbf{H}}_k \hat{\mathbf{x}}_{k\|k-1}$
Innovation (or residual) covariance	$\mathbf{S}_k = \overline{\mathbf{H}}_k \mathbf{P}_{k\|k-1} \overline{\mathbf{H}}_k^\top + \mathbf{R}_k$
Kalman gain	$\mathbf{K}_k = \mathbf{P}_{k\|k-1} \overline{\mathbf{H}}_k^\top \mathbf{S}_k^{-1}$
Updated (a posteriori) state estimate	$\hat{\mathbf{x}}_{k\|k} = \hat{\mathbf{x}}_{k\|k-1} + \mathbf{K}_k \tilde{\mathbf{y}}_k$
Updated (a posteriori) estimate covariance	$\mathbf{P}_{k\|k} = (\mathbf{I} - \mathbf{K}_k \overline{\mathbf{H}}_k)\mathbf{P}_{k\|k-1}$

where the state transition and observation matrices are defined to be the following Jacobians

$$\overline{\mathbf{F}}_{k-1} = \left.\frac{\partial f}{\partial \mathbf{x}}\right|_{\hat{\mathbf{x}}_{k-1|k-1}, \boldsymbol{u}_{k-1}} \qquad \overline{\mathbf{H}}_k = \left.\frac{\partial h}{\partial \mathbf{x}}\right|_{\hat{\mathbf{x}}_{k|k-1}}$$

Due to errors introduced by the linearization, the Extended Kalman filter does not perform optimal estimate updates and may even diverge in certain cases.

Chapter 3

Problem Statement

3.1 Global Game State Estimation

The need to operate under partial observability and interact with objects in the environment makes the creation of a world model a necessity for most robotic systems. In a multi-robot system, such as a Robocup game, where several agents interact simultaneously with each other and observe only portions of the environment, the need for a consistent view of the world is even greater.

When using directed vision as the primary sensor for perception, soccer-playing agents are usually unable to observe the entire environment at once. If no communication between teammates is available, each robot must form a belief (a local world model) about the environment without input from other agents. On the other hand, if inter-robot communication is available, then there exists a possibility of creating a global world model that will be shared by all the robots of the team and will reflect the fusion of each robot's personal belief with those of the others.

In any game of soccer, most activity takes places around the ball and strategies tend to be based on where the ball is, so accurate knowledge about its position in the field is very important. The same applies to the Robocup Standard Platform League where teams of five robots must co-ordinate themselves to play games of soccer. In RoboCup, the ball is the most important object and therefore a need for a shared world model that provides an accurate position estimate for the ball at all times arises. Effective ball modelling can be helpful in many situations. Through this shared world model, robots should be able to cooperatively know where the ball is located, even if it is not within their own field of

view. If a robot has no sight of the ball, whether it is due to obstruction by another robot or due to looking towards a different direction, it can easily deduce the location of the ball, if all the robots of the team work together to estimate it cooperatively. Therefore, it was highly motivating to find ways to maintain a ball model at a global frame.

Furthermore, apart from the ball, it is of high importance for each robot to gain and maintain knowledge about its teammates' positions. Moreover, the creation of such a model about the ball and the incorporation of teammate positions is very important since it provides the opportunity of developing whole-team strategies contrary to single-player behaviors. This fact introduces cooperation and coordination between robots, which allow them to assume different roles within the field (e.g. attacker, defender, supporter, etc.). Imagine a simple scenario where two robots of the same team see the ball in front of an empty opponent goal. In the case where there is no communication and no shared world model, the two robots would probably both rush towards the ball to kick it and score, however they might bump to each other along the way, fall down, and fail to score. On the contrary, if they share information about their positions and the ball, using plain and simple role planning can handle the situation more effectively. For example, the closest robot to the ball can assume the role of the attacker, while the other robot becomes supporter; only one of them will go towards the ball to score and the other one can move to a position near the ball and be ready to assume attacker role, in case the first robot fails to score.

One can clearly see that in scenarios like the ones described above, the use of a shared global world model would highly benefit the team. Of course, there is the need for network communication for the creation and use of the model proposed, but in domains like Robocup and robotic soccer leagues in general, this need is usually covered. Therefore, Robocup soccer teams would greatly benefit from the creation of a mechanism that gathers all local world beliefs from all the robots as input and generates a fusion of these beliefs to create a global world model shared by all the robots of the team.

3.2 Related Work

Many Robocup teams have developed information sharing mechanisms, since they are essential as discussed above. Below, we will briefly present some of these works. Considering all the methods reviewed, one cannot easily distinguish a single best choice to

address the problem stated. This is the reason there are many variations of such methods among the SPL teams and there is no "standard" way to go.

3.2.1 Weighted Belief Averaging

B-Human [19] and rUNSWift [20], two of the best teams in the Standard Platform League, are using a simple, yet effective, technique to address this problem. Given that they have developed their own walk engine and they have designed their own odometry which leads to very accurate self-localization, their robots simply exchange their local position estimates and, without any further processing, this collection of positions serves as the fusion of their beliefs. Apart from the positions of the teammates, they also generate a common, shared belief about the position of the ball within the field. Both teams create this belief by computing a weighted average over all the local beliefs about the ball. The weights, meaning how much the belief of each robot contributes to the resulting belief, depend on various factors that each team determines according to their needs. The most common factors taken into account is the uncertainty of each robot about their own position inside the field, the time that has passed since they last saw the ball, the uncertainty of the ball observation (usually higher ball bearing, means higher uncertainty about the observation), etc.

This weighted-average approach works well, when the local beliefs are accurate. As described above, the local beliefs depend on several factors: the accuracy of the walk engine, the accuracy of the odometry, the accuracy of observations, the choice of the self-localization algorithm, the choice of filtering techniques, the variety of recognized landmarks, etc.

Similar fusion and information sharing techniques were also used by the CMU team [21] at the time when the Robocup Standard Platform League was using the Sony Aibo robots.

3.2.2 Remote Filter Updates

Austin Villa [22], one of the best SPL teams, are using a slightly different approach. Each robot uses an Unscented Kalman Filter (UKF) to keep track of its own pose and the position of the ball, but it also allows other robots to perform updates to its own filter, as if it has gotten a new observation. Since the robot's pose and the position of the ball are part of the same filter, if other robots update the position of the ball, this instantly

reflects to the pose of this robot. This means that when a robot lacks observations of localization landmarks (goal posts, lines, etc.), but receives several ball observations (e.g. this robot is an attacker and has its head "locked" to the ball while approaching it), its pose estimate can be corrected by other robots also observing the ball and updating the position of the ball in its own local filter. Since the estimate about the position of the ball will have a much lower uncertainty compared to the estimate of the robot's pose, the pose will eventually get adjusted to match the ball estimate.

This approach is more sophisticated than the one discussed in Section 3.2.1 and harder to implement and debug, but it provides the opportunity of pose correction only through ball observations, provided some assistance by other robots also observing the ball and having accurate self-position estimate.

3.2.3 Global Filtering

Nao Devils [23, 24] and NTU RoboPAL [25] are using another approach to address the problem we study. They are creating an augmented "state" which includes the poses of all the robots of the team and information about the ball and possibly other moving objects (e.g. opponent robots in the approach of Nao Devils). Then they perform a filtering algorithm over this augmented state. Nao Devils are using Unscented Kalman filtering, while NTU RoboPAL are using Extended Kalman filtering. The choice between Extended and Unscented Kalman filter for this task is widely discussed in the literature, so one cannot be certain about which one is the best choice, and should decide after considering all needs and facts.

Apart from the algorithmic differences, the method used by both teams is similar, with the difference that NTU RoboPAL are using vision observations as direct measurements in their collective filter, while Nao Devils are using either personal localization information (e.g. pose and uncertainty) in the case where any robots of their team observes a static world element or they run a Maximum Likelihood (ML) algorithm in the case where they observe a dynamic feature. This dynamic feature could be a teammate or an opponent robot. The reason they use the ML algorithm is to determine whether the observation made corresponds to one of the existing entries in the state of their filter. If all the choices are highly unlike, a new entry (model) is inserted into the state to represent the newly

detected object, which is most likely an opponent robot, given that ML did not find a suitable enough entry.

In conclusion, both methods are quite similar with the main difference being that NTU RoboPAL are performing one layer of global filtering, while Nao Devils are performing two layers of filtering, namely local filtering whose output serves as input (observations) to global filtering.

Chapter 4

Our Approach

Our solution to the problem of shared global game state estimation is based on the creation of a mechanism that is responsible for collecting all the local world state beliefs (output of the local estimation filter), performing filtering in order to integrate them, and producing a consistent belief about the global collective state of the game. This belief will be common to all robots, so that they all share the same information which allows the opportunity for cooperative decision making. Our approach is similar to the ideas described in Section 3.2.3.

4.1 The Idea

For the purposes of this thesis we define the global game state to be the field poses (positions and orientations) of all robots in our team and the location of the ball within the field. Other state variables could be included in the global game state, such as the field poses of opponents and the field positions of dynamic objects (e.g. human referees), however we chose to ignore them in this work, due to lack of recognition ability for these features. Nevertheless, our methodology could incorporate such elements, provided there exists corresponding recognition and local estimation procedures for these.

The general idea of our approach for global game state estimation is fairly simple. Apart from their local (egocentric) belief, all robots maintain a global belief, which is updated using data from the whole team. Each robot monitors the network and listens for local belief messages from all the robots (including self) containing information about their pose and information about the ball (if any). For each incoming message, the

receiving robot performs a filtering step in order to incorporate the new information into the current global belief. Therefore, sharing and fusing the local beliefs, enables each robot to correct local errors and gain knowledge about the state of its teammates and the location of the ball.

4.2 Kalman Filter

We chose to use a Kalman Filter (KF) for this state estimation problem, because it seems to be a suitable option for our needs and it is proven to produce optimal results, if the models used are realistic. Each robot needs to maintain a $(3N + 2)$-dimensional Kalman filter, where N represents the number of robots in the team. We need three variables (x_i, y_i, θ_i) for the pose (location and orientation) of robot i and two variables (x_b, y_b) for the location of the ball. The said Kalman filter will use local belief messages from each robot as its observations. An update step takes the observation $(robot_x, robot_y, robot_\theta, ball_x, ball_y)$ and updates only the cells of the Kalman matrices that correspond to this specific robot. This is achieved through the **H** matrix which translates how the observation relates to the state.

The Kalman matrices for our problem are described below:

$\hat{\mathbf{x}}$: $(3N + 2) \times 1$, represents the global state we need to estimate.

P : $(3N + 2) \times (3N + 2)$, represents the uncertainty of the filter.

F : $(3N + 2) \times (3N + 2)$ is the matrix of the state transition model.

Q : $(3N + 2) \times (3N + 2)$, is the covariance matrix of the state transition model noise.

H : $3 \times (3N+2)$ (no ball information), $5 \times (3N+2)$ (with ball information) is the matrix of the observation model.

R : 3×3 (no ball information), 5×5 (with ball information), is the covariance matrix of the observation model noise.

z : 3×1 (no ball information), 5×1 (with ball information), represents the observation.

$\tilde{\mathbf{y}}$: 3×1 (no ball information), 5×1 (with ball information), the innovation matrix capturing the measurement residual (difference between observation and estimation).

$\mathbf{S}$: 3×3 (no ball information), 5×5 (with ball information), represents the covariance matrix of the innovation noise.

$\mathbf{K}$: $(3N+2) \times 3$ (no ball information), $(3N+2) \times 5$ (with ball information), represents the optimal Kalman gain and determines the effect of the innovation matrix on the current belief.

Note that typical Kalman filtering also includes a matrix $\mathbf{B}$, as described in Section 2.3, which expresses the effect of the control input on the state. Since there is no explicit control over the process at the global level in our case, $\mathbf{B}$ is taken to be a zero matrix. In fact, control inputs at the local level are viewed as process "noise" at the global level, hence we only track the system's state through observations. Control inputs are taken explicitly into account at the level of local state estimation.

As shown by the dimensions of some of the matrices described above, in our Kalman filter we consider two possible scenarios; the received belief message may or may not contain information about the ball (apart from information about the robot's pose). In the first scenario, the observation is 5-dimensional, whereas in the second scenario, the observation is 3-dimensional. Thus, we need to maintain two different versions of some matrices $(\mathbf{H}, \mathbf{R}, \mathbf{z}, \tilde{\mathbf{y}}, \mathbf{S}, \mathbf{K})$ in order to deal with each possible scenario. This feature does not affect the functionality of our filter, because each update, using the appropriate matrix, only updates state variables related to the current observation.

Given the above, the KF equations used in our filter are summarized below:

$$\begin{aligned} \textit{Predict}: \quad \hat{\mathbf{x}}_{k|k-1} &= \mathbf{F}_k \hat{\mathbf{x}}_{k-1|k-1} \\ \mathbf{P}_{k|k-1} &= \mathbf{F}_k \mathbf{P}_{k-1|k-1} \mathbf{F}_k^\top + \mathbf{Q}_k \end{aligned}$$

$$\begin{aligned} \textit{Update}: \quad \tilde{\mathbf{y}}_k &= \mathbf{z}_k - \mathbf{H}_k \hat{\mathbf{x}}_{k|k-1} \\ \mathbf{S}_k &= \mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^\top + \mathbf{R}_k \\ \mathbf{K}_k &= \mathbf{P}_{k|k-1} \mathbf{H}_k^\top \mathbf{S}_k^{-1} \\ \hat{\mathbf{x}}_{k|k} &= \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k \tilde{\mathbf{y}}_k \\ \mathbf{P}_{k|k} &= (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \mathbf{P}_{k|k-1} \end{aligned}$$

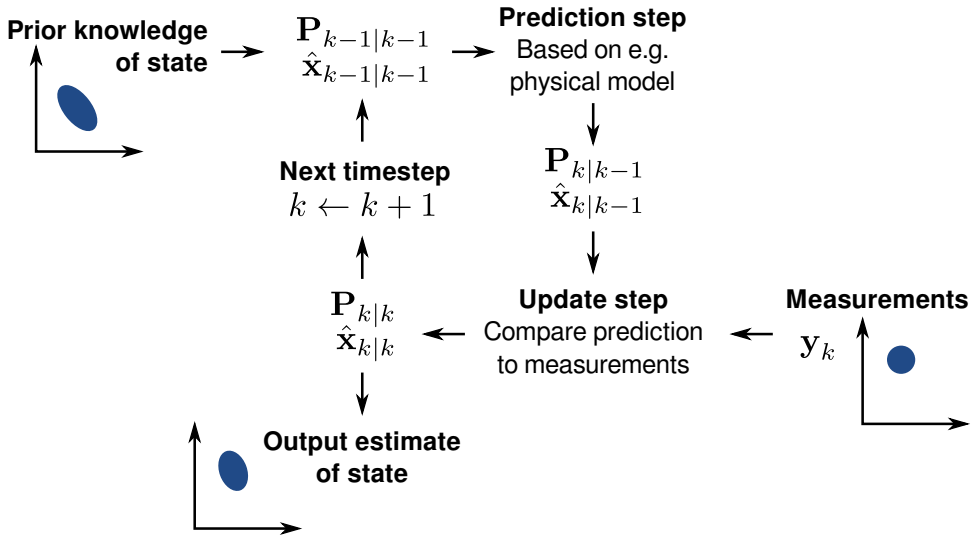

Figure 4.1: Kalman Filter procedure [1]

The filtering procedure is also illustrated in Figure 4.1. The key filtering steps (predict and update) and the definitions of the chosen models are described in the following sections.

4.2.1 Prediction Step

The prediction step of our filter is straightforward. The main assumption we make is that on the next time step, the pose of each robot will be somewhere "in the neighborhood" of its current pose, independently of the poses of the other robots in the team, because of the nature of the system we are observing. Therefore, at the global level this assumption implies that the global state at the next time step will not be too different compared to the current global state. As a result, $\mathbf{F}$ is chosen to be the identity matrix $\mathbf{I}$ and $\mathbf{Q}$ is chosen to be a block-diagonal matrix with values that reflect the size of this neighborhood independently for each robot and the ball.

$$\mathbf{Q} = \begin{bmatrix} \mathbf{Q}_1 & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{Q}_2 & \vdots & \mathbf{0} \\ \vdots & \cdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{Q}_b \end{bmatrix}$$

Under these assumptions, the pose of each robot i can be seen as a 3-dimensional Gaussian distribution, centered around its current pose estimate with covariance $\mathbf{Q}_i$. Similarly, the

location of the ball can be seen as a 2-dimensional Gaussian distribution, centered around its current location estimate with covariance $\mathbf{Q}_b$. While the global state estimate does not change during the prediction step ($\hat{\mathbf{x}}_k = \mathbf{I}\hat{\mathbf{x}}_{k-1}$), the uncertainty of the filter is increased additively by the covariance matrix of the transition model noise ($\mathbf{P}_k = \mathbf{I}\mathbf{P}_{k-1}\mathbf{I}^\top + \mathbf{Q}$).

4.2.2 Update Step

Our update step is somewhat unusual. Normally, one would have to accumulate all the messages from all the robots, then combine them into one "large" observation and perform the typical Kalman update steps. Yet, in our case, each observation either $(robot_x, robot_y, robot_\theta)$ or $(robot_x, robot_y, robot_\theta, ball_x, ball_y)$ from any specific robot, only updates parts of the filter that relate to this robot. This is controlled through the $\mathbf{H}$ matrix, which translates how the observation relates to the state. So, if $\mathbf{H}$ is chosen correctly, observations from a specific robot will only affect the parts of the filter that relate to this particular robot.

Let's assume we have three robots currently in the team (with IDs 1, 2 and 3). An observation received from robot 1 containing no information about the ball, will be processed using the following $\mathbf{H}$ matrix:

$$\mathbf{H} = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \mathbf{0} & \mathbf{0} & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Accordingly, an observation from robot 2 containing no information about the ball, will be processed using the following $\mathbf{H}$ matrix:

$$\mathbf{H} = \begin{bmatrix} 0 & 0 & 0 & \mathbf{1} & \mathbf{0} & \mathbf{0} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{0} & \mathbf{1} & \mathbf{0} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{0} & \mathbf{0} & \mathbf{1} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{0} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{0} & \mathbf{1} \end{bmatrix}$$

So, the $\mathbf{H}$ matrix is constructed by blocks of either identity or zero matrices depending on the ID of the sender robot. In the first example, the sender robot has ID 1, so the first 3×3 block is an identity matrix, while the other two 3×3 blocks are zero. In the second example, the $\mathbf{H}$ matrix is augmented by two more rows in order to account for the information received about the ball. Hence, the second (horizontally) 3×3 block is

an identity matrix (since the sender robot has ID 2) and the bottom right 2×2 block is an identity matrix corresponding to the ball state variables. The uncertainty in the observation is expressed by the covariance matrix $\mathbf{R}$, which is either 3×3, if there is no ball information, or 5×5, if there is ball information.

It should be noted that the $\mathbf{H}$ matrix could alternatively be defined as a $(3N + 2) \times (3N + 2)$ block-diagonal matrix (with appropriate identity or zero block matrix entries) with the observation being a vector of constant size $(3N + 2) \times 1$. The net effect on the update step of the filter would be the same, however the factored representation presented above was preferred over this choice to avoid costly matrix operations, such as the inversion of the $\mathbf{S}$ matrix. For example, for $N = 5$ we would have to invert a 17×17 matrix, whereas in our case we only have to invert either a 3×3 or a 5×5 matrix.

4.2.3 Transition Model

To complete the prediction step, the Kalman filter requires a transition model, namely the $\mathbf{F}$ and $\mathbf{Q}$ matrices. It has been mentioned already that, in our case, $\mathbf{F}$ is taken to be the identity matrix $\mathbf{I}$. Matrix $\mathbf{Q}$ represents the covariance of the transition model noise, meaning the noise in the motion of the robots and the ball in our case. This noise at the global state level corresponds to all possible locomotion actions that can be taken by the robots and all common displacements of the ball respectively. This expresses in a way the maximum possible movement in each dimension of the state; the noise in each of these dimensions is assumed to be independent from the others. Therefore, we define $\sigma^2_{x_i}$, $\sigma^2_{y_i}$, $\sigma^2_{\theta_i}$ to be the variances for robot's i state variables and the corresponding 3×3 block $\mathbf{Q}_i$ in $\mathbf{Q}$ will only contain these values along the diagonal. Similarly, we define $\sigma^2_{x_b}$ and $\sigma^2_{y_b}$ to be the variances for ball's state variables and the 2×2 block $\mathbf{Q}_b$ in $\mathbf{Q}$ will only contain these values along the diagonal. These variances need to be scaled by Δt^2, where Δt is the time step between successive filter iterations, so that the noise is proportional to the elapsed time. Note that Δt need not be constant; the actual elapsed time since the previous iteration can be measured dynamically at each iteration and inserted into the model. In summary, $\mathbf{Q}$ becomes a fully diagonal $(3N + 2) \times (3N + 2)$ matrix, as shown

below:

$$\mathbf{Q} = \Delta t^2 \begin{bmatrix} \begin{bmatrix} \sigma_{x_1}^2 & 0 & 0 \\ 0 & \sigma_{y_1}^2 & 0 \\ 0 & 0 & \sigma_{\theta_1}^2 \end{bmatrix} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \begin{bmatrix} \sigma_{x_2}^2 & 0 & 0 \\ 0 & \sigma_{y_2}^2 & 0 \\ 0 & 0 & \sigma_{\theta_2}^2 \end{bmatrix} & \vdots & \mathbf{0} \\ \vdots & \cdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \begin{bmatrix} \sigma_{x_b}^2 & 0 \\ 0 & \sigma_{y_b}^2 \end{bmatrix} \end{bmatrix}$$

Robot motion model variances

The values of $\sigma_{x_i}^2$, $\sigma_{y_i}^2$, $\sigma_{\theta_i}^2$ are determined by looking at robot's i specifications, taking the maximum range along each dimension, and assuming that this represents 3σ or 99.7% of our distribution. In our case (Standard Platform League), these values do not vary from robot to robot, however this need not be the case; our model allows for custom values for each robot, therefore it can accommodate diverse robot platforms. For the NAO robot, we have determined the following values:

- $\sigma_{x_i}^2$: NAO's maximum speed is about $30cm/s$, therefore $3\sigma_{x_i} = 0.3 \implies \sigma_{x_i}^2 = 0.01$
- $\sigma_{y_i}^2$: NAO's maximum speed is about $30cm/s$, therefore similarly to σ_{x_i}, $\sigma_{y_i}^2 = 0.01$
- $\sigma_{\theta_i}^2$: NAO's maximum angular velocity is about $2.1rad/s$, so $\sigma_{\theta_i}^2 = 0.49$

Ball motion model variances

The ball's movement is more unpredictable compared to that of a robot. The range of ball's movement is highly dependent on the power and the angle of the kick (if one occurred) or any other unpredictable occurred event (e.g. human referee mistakes). Since each kick is different and new kicks are constantly designed, we cannot know the exact covariance of the noise that should be used. Hence, we chose an arbitrarily large value ($\sigma_{x_b}^2 = \sigma_{y_b}^2 = 100$) for the covariance of the noise for each ball state variable to account for this unpredictability.

As confirmed by various experiments with different values for these variances, there is no significant impact on the outcome of the algorithm as long as there are observations

for the ball. The only element affected is how fast the uncertainty of our filter about the ball rises in the case where there are no ball observations.

4.2.4 Observation Model

To complete the update step, the Kalman filter requires an observation model, namely the **H** and **R** matrices. It has been mentioned already that, in our case, **H** is a block matrix with some blocks being identity matrices. Matrix **R** represents the covariance of the observation model noise, meaning the uncertainty in the local belief of each robot. If there is no ball information, **R** is a 3×3 matrix expressing the uncertainty in the pose estimate of the sender robot. If there is ball information, **R** is a 5×5 matrix. Depending on the implementation of the local filtering, **R** is either a full 5×5 matrix or a block-diagonal 5×5 matrix, consisting of a 3×3 block followed by a 2×2 block. If robot pose and ball location are estimated jointly at the local level, then **R** is the full covariance matrix of the local joint estimation outcome. On the other hand, if robot pose and ball location are estimated independently at the local level, which is the case in our team, the two blocks of **R** are the corresponding covariance matrices of the local estimation outcomes.

Robot observation model variances

In our team, each robot performs local filtering for self-localization and the observation for a robot's pose is simply the outcome of this local filter. Therefore, matrix **R** at the global level is taken to be equal to the uncertainty of the local filter. Our team currently uses 3-dimensional Kalman filtering for self-localization, so the uncertainty of the local filter of the sender robot, expressed by the 3×3 covariance matrix of the local estimate, is copied directly into **R**.

Ball observation model variances

In our team, each robot performs local filtering for ball location estimation and the observation for the ball is simply the outcome of this local ball filter. Therefore, the 2×2 block of matrix **R** at the global level is taken to be equal to the uncertainty of the local ball filter. Our team currently uses two independent 1-dimensional Kalman filters for

ball localization, so the uncertainties of these two local ball filters of the sender robot fill the diagonal of the 2×2 block of matrix $\mathbf{R}$.

4.3 Extended Kalman Filter

The Kalman filter in its original form is optimal for systems which fulfill a number of assumptions, such as the sole involvement of zero-mean Gaussian noise and linear transition and observation models. This is rarely true in practical applications, since most systems of interest are non-linear in one aspect or another. The Kalman concept is popular and successful nonetheless, due to the possibility of linearizing the non-linear models around the current estimate. This provides a decent enough approximation to allow fairly accurate tracking of the state. This is the main idea behind the Extended Kalman Filter (EKF).

The classic Kalman filter algorithm proved to be unsuitable in our case for a single, yet important, reason. So far, we have assumed that ball observations are given in global field coordinates to match the corresponding ball state variables at the global level. However, in practice, ball observations are *relative* to the robot's pose. This means that the position of the ball is expressed with respect to the pose of the robot, which is taken to be the origin. If robot i offers a ball observation and its exact pose in global field coordinates is known, then the exact global position of the ball (x_b, y_b) within the field is derived by the following equation:

$$\begin{bmatrix} x_b \\ y_b \end{bmatrix} = \begin{bmatrix} x_{rb} \\ y_{rb} \end{bmatrix} \begin{bmatrix} \cos\theta_i & -\sin\theta_i \\ \sin\theta_i & \cos\theta_i \end{bmatrix} + \begin{bmatrix} x_i \\ y_i \end{bmatrix}$$

where (x_i, y_i, θ_i) stands for robot's i pose, while (x_{rb}, y_{rb}) stands for the ball's coordinates relative to the pose of robot i. The relationship between (x_{rb}, y_{rb}) and (x_b, y_b) is clearly non-linear thanks to θ_i.

Initially, we attempted to apply the linear Kalman filter described above by manually converting the relative ball observations to global field coordinates using the equations above. However, since the exact true pose of the sender robot is not known, we relied on using either the current pose estimate from the global filter or the current robot pose observation. Both choices, being estimates, contain significant errors and this uncertainty is not reflected on the computed ball observations, which end up being tampered with

non-linear noise. These attempts failed. Furthermore, we tried to address this issue by introducing cross-correlations between robot pose and ball position through matrix $\mathbf{R}$. However, by definition, these cross-correlations can only capture linear dependencies. As expected, this fix failed, too.

To address this problem formally, we followed a direct, analytical approach. First, we inverted the non-linear equations above to create a causal observation model describing how an observation (relative ball position) is "produced" directly from the state (robot pose and ball position in global field coordinates). Hence, solving for (x_{rb}, y_{rb}), we get the following equation:

$$\begin{bmatrix} x_{rb} \\ y_{rb} \end{bmatrix} = \begin{bmatrix} \cos(-\theta_i) & -\sin(-\theta_i) \\ \sin(-\theta_i) & \cos(-\theta_i) \end{bmatrix} \begin{bmatrix} x_b - x_i \\ y_b - y_i \end{bmatrix}$$

which leads to the following observation model:

$$h(\mathbf{x}) = \begin{bmatrix} h_x \\ h_y \end{bmatrix} = \begin{bmatrix} \cos(-\theta_i) & -\sin(-\theta_i) \\ \sin(-\theta_i) & \cos(-\theta_i) \end{bmatrix} \begin{bmatrix} x_b - x_i \\ y_b - y_i \end{bmatrix}$$

This observation model is clearly non-linear and cannot be described by a matrix $\mathbf{H}$, therefore linear Kalman filtering is not applicable. However, $h(\mathbf{x})$ is a differentiable function. Therefore, the next step was to make a transition to an Extended Kalman Filter by linearizing this function about the current a priori state estimate (after the prediction step). Linearization constructs the matrix $\overline{\mathbf{H}}_k$ using the partial derivatives (Jacobian) of $h(\mathbf{x})$:

$$\overline{\mathbf{H}}_k = \left. \frac{\partial h(\mathbf{x})}{\partial \mathbf{x}} \right|_{\hat{\mathbf{x}}_{k|k-1}}$$

Matrix $\overline{\mathbf{H}}$ is similar to $\mathbf{H}$ described above in the linear Kalman filter, however but the block that correlates the pose of the robot with the ball position, as well as the block referring to the ball itself, are different. More specifically, using the same example as before, where there are three robots and robot with ID 1 sends an observation without information about the ball, $\overline{\mathbf{H}}$ would have the same form as $\mathbf{H}$ above:

$$\overline{\mathbf{H}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

However, if robot 2 sends an observation which includes ball information, $\overline{\mathbf{H}}$ would take the following form:

$$\overline{\mathbf{H}} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{\partial h_x}{\partial x_i} & \frac{\partial h_x}{\partial y_i} & \frac{\partial h_x}{\partial \theta_i} & 0 & 0 & 0 & \frac{\partial h_x}{\partial x_b} & \frac{\partial h_x}{\partial y_b} \\ 0 & 0 & 0 & \frac{\partial h_y}{\partial x_i} & \frac{\partial h_y}{\partial y_i} & \frac{\partial h_y}{\partial \theta_i} & 0 & 0 & 0 & \frac{\partial h_y}{\partial x_b} & \frac{\partial h_y}{\partial y_b} \end{bmatrix}$$

where

$$\begin{bmatrix} \frac{\partial h_x}{\partial x_i} & \frac{\partial h_x}{\partial y_i} & \frac{\partial h_x}{\partial \theta_i} \\ \frac{\partial h_y}{\partial x_i} & \frac{\partial h_y}{\partial y_i} & \frac{\partial h_y}{\partial \theta_i} \end{bmatrix} = \begin{bmatrix} -\cos(\theta_i) & -\sin(\theta_i) & (x_i - x_b)\sin(\theta_i) + (y_b - y_i)\cos(\theta_i) \\ \sin(\theta_i) & -\cos(\theta_i) & (x_i - x_b)\cos(\theta_i) - (y_b - y_i)\sin(\theta_i) \end{bmatrix}$$

and

$$\begin{bmatrix} \frac{\partial h_x}{\partial x_b} & \frac{\partial h_x}{\partial y_b} \\ \frac{\partial h_y}{\partial x_b} & \frac{\partial h_y}{\partial y_b} \end{bmatrix} = \begin{bmatrix} \cos(\theta_i) & \sin(\theta_i) \\ -\sin(\theta_i) & \cos(\theta_i) \end{bmatrix}$$

As described in Section 2.3, the EKF equations we used are the following:

$$\begin{aligned} \textit{Predict}: \quad \hat{\mathbf{x}}_{k|k-1} &= \mathbf{F}_k \hat{\mathbf{x}}_{k-1|k-1} \\ \mathbf{P}_{k|k-1} &= \mathbf{F}_k \mathbf{P}_{k-1|k-1} \mathbf{F}_k^\top + \mathbf{Q}_k \end{aligned}$$

$$\begin{aligned} \textit{Update}: \quad \tilde{\mathbf{y}}_k &= \mathbf{z}_k - \overline{\mathbf{H}}_k \hat{\mathbf{x}}_{k|k-1} \\ \mathbf{S}_k &= \overline{\mathbf{H}}_k \mathbf{P}_{k|k-1} \overline{\mathbf{H}}_k^\top + \mathbf{R}_k \\ \mathbf{K}_k &= \mathbf{P}_{k|k-1} \overline{\mathbf{H}}_k^\top \mathbf{S}_k^{-1} \\ \hat{\mathbf{x}}_{k|k} &= \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k \tilde{\mathbf{y}}_k \\ \mathbf{P}_{k|k} &= (\mathbf{I} - \mathbf{K}_k \overline{\mathbf{H}}_k) \mathbf{P}_{k|k-1} \end{aligned}$$

The only difference introduced in this Extended Kalman filter compared to the linear Kalman filter described above is the use of a variable matrix $\overline{\mathbf{H}}_k$ at each iteration k, which is produced by linearization about the current state estimate. All other elements are identical.

Chapter 5

Implementation

5.1 Matlab Simulation

Before creating an actual Monas activity in order to implement this idea on the actual robots, we created a simulation in Matlab to test whether the idea of global shared state estimation was worth to be implemented.

We created a set of fake observations, including robot poses and possibly ball information, and ran them through a Kalman filtering algorithm implemented in Matlab. The state transition model and the observation model used in this simulation are rough estimates of the models described in Sections 4.2.3 and 4.2.4. Since no full local estimation algorithm was executed at the local level, we took the covariance of the observation noise to be equal to the covariance of the state transition noise. This choice is certainly far from optimal, however it was sufficient to get a first glimpse of whether our global filtering approach would work.

Figure 5.1 shows a screenshot of the simulation output. In this test scenario, three robots move at constant speed diagonally in the field; additionally, a ball moves with some initial speed towards the top of the field and eventually comes to rest after some steps. Each robot provides an observation about its pose estimate at each execution cycle. With a probability of 0.5 it also provides a ball observation directly in global coordinates to avoid the problems of non-linear noise correlation and use a linear Kalman filter. These observations (local beliefs) are noisy measurements of the true state. The true state in Figure 5.1 is shown with red crosses, while the local beliefs are shown with blue dots. The outcome of the linear Kalman filter, that is the estimated global state, is shown with

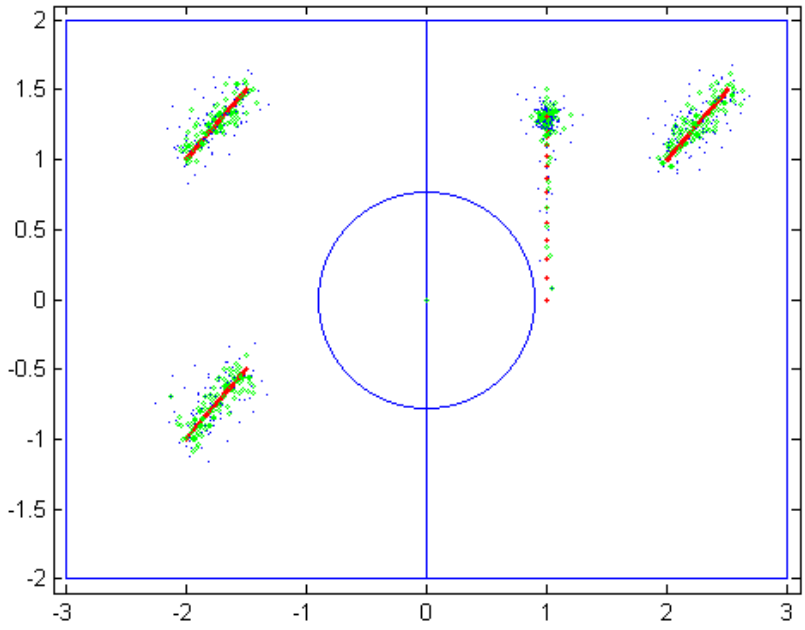

Figure 5.1: Simulation: true state (red), local beliefs (blue), global filter estimates (green).

green marks. One can easily see that even with this rough first approach, the global filter yields better estimates than the local beliefs (the green marks are "closer" to the red crosses than the blue dots).

5.2 KMat: Kouretes Math Library

KMat [7] is a software library that supports a selected subset of algebraic matrix operations. The focus of the library is mainly on real number operations and the primary goals of KMat are low memory footprint and calculation efficiency. Existing linear algebra libraries typically perform run-time validation for the compatibility of the operands and are optimized for large matrices. On the other hand, KMat is optimized for small matrices (up to 5×5) and supports only a selected subset of operations (addition, subtraction, multiplication, scalar addition, scalar multiplication, transposition, inversion). KMat is optimized for and supports two type of matrices: dense matrices and affine transformation matrices. In our work, all the essential Kalman filtering operations were

implemented using KMat.

5.3 The SharedWorldModel Monas Activity

In order to implement our approach on the real NAO robots, we created a Monas activity, named `SharedWorldModel`. This activity subscribes to the blackboards of all robots and monitors the topic where local estimation (self and ball localization) messages are being published. Our activity is currently executed as a separate agent (cf. Section 2.2.1) and is specified as such inside our `agents.xml` configuration file that determines which activities will be running on the robot and at what frequency. Our activity is currently being executed at a frequency of 5Hz (five times per second).

One could easily wonder: "In filtering, the estimation gets more accurate with more iterations, so why do you perform only five iterations per second?". The answer to this question lies in 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 resources that are shared throughout all programmed functionality, as well as middleware operations. Hence, the workload of each activity must be in accordance with the total workload on the robot. The frequency we chose for our activity was empirically found to strike a good balance between computational load and filter efficiency for global game state estimation.

In each execution cycle, our activity performs a filter prediction step and then it iterates over all the `known_hosts` (all the robots in our case, including self) and checks if a new local belief message exists. If it finds one, it performs a filter update step, using that message as an observation. If multiple local belief messages are found, a separate update step is performed for each one of them. When the update iteration is over, meaning that the robot running this activity communicated with all the robots that are currently connected to the network, it creates a message containing the outcome of the global filtering algorithm (the global state estimate containing each robot's pose and a global position for the ball) and publishes it to the corresponding topic (`worldstate`). This way, any other activity (on the same robot, on a different robot, or on a remote computer) that needs this global state information, can simply subscribe to this topic and receive this message.

5.4 Distributed Individual Computation

After having read the description of our approach, one would assume that global state estimation is performed only on a single robot and is then broadcast to all other robots. This was indeed our initial idea, but we decided to discard it, because it is quite common for the UDP network to "lose" packets; if packet loss occurs while the outcome is broadcast, we would lose all this valuable information we strived to compute and share.

What we decided to do instead, was to let each robot execute our `SharedWorldModel` activity and perform its own global game state estimation. This proved to be a much better choice for two main reasons. One is that each robot generates this estimate locally and there is no need to actually transmit it through the UDP network. Each robot still creates a `SharedWorldInfo` message and publishes it to the appropriate topic, but this message only serves activities running on the same robot. The other reason is that a centralized approach, whereby only one robot performs the global estimation and publishes it, is prone to failures, if the robot responsible for the computation goes inactive (e.g. penalization, battery shortage, hardware failure, etc.). In addition, we would also need an algorithm to detect the failure and determine which robot would assume this centralized role next, which would induce extra costs in both aspects of computational power. Since the robots would need to exchange additional messages to decide which one will assume this role, there will be network usage penalties as well. Furthermore, these messages might even get lost, so apart from the actual algorithm, we would also need to create a retransmission protocol that would introduce even more complexity and network overload.

An obvious question instantly rises from the statement above: "Are all the global game state estimates identical on all robots?". The quick theoretical answer to this question would be "No, they may not be exactly the same!". This lies on the fact that we are using a UDP network that is known to lose messages, so if for example robot 1 broadcast its local belief and robot 2 received it, while robot 3 did not due to network error, then robot 2 and robot 3 would produce a different global state estimate. This inconsistency does not create any substantial problem in practice, because of two reasons. The first is that filtering at the global level is mostly performed to provide a collective knowledge about the rough state of the world, not its details. The poses of the robots and the position of the ball do not need to be totally identical between two robots, when

they are using them to decide which role to assume. As we observed, the differences are extremely small (not visible by eye when observing and comparing the `SharedWorldInfo`s of two robots through KMonitor). This means that with the tolerance of a small error, we save on valuable resources by avoiding a centralized approach. The second reason lies in the nature message exchange in our team's code. In our implementation, we are using messages of type `data` which remain on the blackboard as long as the available buffer isn't full. In our filtering algorithm, we only need the last published observation, so even if a message fails to be delivered to a robot, it will remain written in the blackboard of the sender robot and most likely it will be received on the next execution cycle. This means that the `SharedWorldInfo`s of different robots might not be in total sync at some cycle, but they will most likely become identical after a few cycles.

Chapter 6

Results

In this chapter we present the results of our work and show how it benefits the rest of the team. First, we need to clarify that our approach typically introduces a second, more abstract, layer of filtering. The first layer of filtering is performed inside the local estimation module of each robot and our approach takes the outputs of all these filters as input, combines them, and performs another layer of filtering trying to fuse them in the best possible way. The standard approach to show that a framework like the one created works well, would be to compare the output estimate of the global filter to the true state of the world and use a standard error metric (e.g. mean square error). Then we should do the same by comparing the results of individual local beliefs to the true state and computing a cumulative error metric (e.g. sum of mean square error over all the local beliefs). This way we would easily be able to see if our global state estimation provides a better estimate of the true state of the world than individual local beliefs. Unfortunately, in our case this comparison cannot be performed because we do not have an effective mechanism that allows us to know the exact true state of the world (i.e. the pose of each robot and the position of the ball within the field), also known as a "ground truth" mechanism. In lack of ground truth, we cannot provide quantitative results, yet we will provide some qualitative results on certain scenarios that could occur during a robotic soccer game and show that our approach proves to be really useful.

Figure 6.1: Scenario I: the true state of the world

6.1 Scenario I: *Global estimate better than local ones*

Our first goal is to check how accurate our global estimate is compared to the local estimates. In this scenario (Figure 6.1) there are two stationary robots in the field. One is placed at the center of the field, while the other one is placed on the side line, aligned with the penalty cross. The ball is placed on the penalty cross. Both robot observe the ball and each one of them maintains a local estimate of the ball position. We used KMonitor [16] in order to check the global estimate produced by our activity as well as the local estimates of the two robots. As shown in Figure 6.2, the global estimate about the position of the ball is better than each of the local estimates and closer to the true location of the ball. This is highly important because, even when none of the robots has a good enough estimate, the collective state estimate provides an accurate ball position, which can be used by all robots. This can greatly help in cases where robot cameras are faulty or missalligned, which occurs quite often in practice.

Figure 6.2: Scenario I: global (pink circle) and local (small gray circles) ball estimates

6.2 Scenario II: *Approaching the ball without seeing it*

Another goal was to test if our approach can be helpful for the entire team, for example if a robot can approach the ball without necessarily observing it. The ball could be hidden from the robot's view due to obstacles (e.g. another robot) or physical limitations (e.g. too far). As long as there is at least one teammate robot observing the ball, a global ball position estimate can be formed and any robot can start moving towards the estimated position of the ball, at the same time scanning the field trying to locate the ball on its own.

This ability of shared world information is quite essential, especially since the dimensions of the field changed from 6×4 to 9×6 in the rules of RoboCup SPL 2013 [3]. The larger field introduces more chance that the ball will not be visible by some robots, so it is of great advantage if all the robots could know where the ball is located, even when only one robot observes the ball. Of course, there is a key prerequisite to achieve this outcome, namely that the output of local estimation is fairly correct, otherwise, since the ball estimate is relative to the robot's pose, the global ball estimate would be far from correct.

Before the creation of `SharedWorldModel`, in such a scenario, each robot not observing

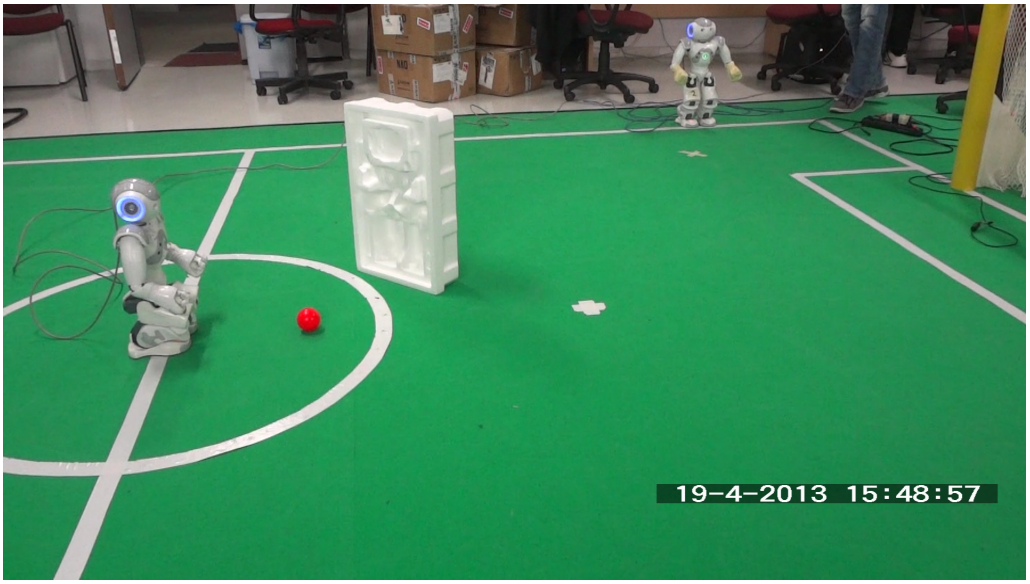

Figure 6.3: Scenario II: the true state of the world

the ball would need to start scanning the field area around it for the ball. If the scan ends up being unsuccessful, the robot would need to start a random walk in order to scan another area for the ball. One can easily see that this solution is highly inefficient compared to the outcome of the solution described above using our global estimation.

To test this solution, we designed a scenario with two robots (Figure 6.3). One of them is located at the center of the field, is stationary, and stares at the ball which was positioned in front of it. The other robot is located on the side line of the field, as if it was returning from a `penalized` state. An obstacle is placed between the moving robot and the ball, so that the robot returning to the game cannot directly observe the ball. Even though this robot could not observe the ball at any time, it successfully approaches it solely by using the estimate provided by our global estimation filter, since the other robot was updating the position of the ball by "feeding" the filter with ball observations. Some snapshots of the robot's course can be seen in Figure 6.4. Note that the local pose estimate of the moving robot is not totally accurate, yet it manages to approach the ball.

Figure 6.4: Scenario II: true state (top) and estimated local and global states (bottom)

Chapter 7

Conclusion and Future Work

7.1 Conclusion

This thesis summarizes the work performed 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 higher level of filtering that takes the outcomes of the local estimation filters and performs a global Extended Kalman Filtering to combine them and produce a global game state estimate. The outcome of our algorithm is the collective state estimate about the world (robot poses and ball position). 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 information sharing mechanism also creates room for further development of new features. Some of them are briefly discussed below.

7.2 Future Work

7.2.1 Localization Feedback: Ball as a Landmark

One of the most important features that could be implemented is a closed-loop feedback mechanism to the local module of each robot for self-localization. Currently, the self-localization module performs its filtering without taking into account the output of our activity. It could possibly improve its local estimates, if there was feedback from our global filter back to the local filter of each robot. More specifically, given that the global

estimate about the ball is better than the local ones, this global ball estimate could be introduced into the self-localization algorithm as an additional field landmark.

Since the switch of the Standard Platform League to using two goals with the same color (yellow) in 2012, it is quite common for a robot with increased uncertainty about its pose (e.g. a robot that was fallen and got up) to face ambiguities in its estimation due to field symmetries. Symmetries are quite dangerous during a game, because if a wrong choice is made, the robot may score a goal against its own team. In fact, in an extreme attempt to overcome such situations, certain teams prefer to leave their robots on the ground after a fall, so that they get penalized for inactivity and placed on the sideline of the field in their own half, thus resetting their localization estimates!

The existence of the global estimate of the ball as a landmark inside the existing self-localization algorithm could be used to resolve those symmetries. If the ball is treated as a landmark and an estimate is available (meaning that at least one robot in the team observes the ball), then the robots' local filter will easily resolve symmetries.

Another use of this feedback apart from solving symmetries is self position estimate correction. Consider the scenario described in Section 3.2.2, where an attacker is heading towards the ball starring solely at the ball. This robot receives many observations about the ball, but possibly no observations about field landmarks, such as goal posts and field lines, because its head is "locked" to the ball. If its teammates are also observing the ball and the global ball estimate is corrected by them, then the pose of the attacker could also be corrected, if the ball is treated as a landmark. Since the attacker's uncertainty about the relative position of the ball will be fairly low (due to many observations) and the uncertainty of the global ball estimate will also be fairly low, despite the uncertainty of the attacker's pose being high (due to very few observations), the attacker's pose will automatically be corrected by the local filter to account for the landmark (global ball) observation.

7.2.2 Team Strategy and Coordination

There is currently work in progress in our team on team strategies, coordination, and dynamic role assignment based on the output of our filter. It is quite obvious that there is no possibility of creating effective collaborative multi-agent behaviors without a shared global state estimation mechanism. Creating coordinated team behaviors is a necessity

in robotic soccer for various reasons. The simplest and most important is that not all the robots of the team need to run after the ball, if they are all observing it. For example, if two robots are observing the ball, a rational decision would be to let only one of them approach the ball, while the other heads towards the opponent goal to anticipate a pass or simply be at a position close to where the ball will end up in case of an unsuccessful kick by the first robot. Without coordination and role assignment, such decisions cannot be realized; both robots would probably head towards the ball, resulting in either bumping into each other or, in the best case, one taking a shot, while the other one stands right next or behind it.

7.2.3 Opponent Modelling

Currently, our team does not have the ability to recognize opponent robots through the camera, the main sensor for perception. However, when this feature is implemented and added, our approach could be easily applied on the augmented global game state which includes three extra state variables for the pose of each opponent robot. This could lead to making even better and more strategic team planning during the game, such as avoiding passing the ball to a robot blocked by opponents, choosing a kicking direction towards the opponent goal that clears all players (opponents and teammates), or avoiding collision with and obstructions by opponent robots.

References

[1] Wikipedia: Kalman filter — wikipedia, the free encyclopedia (2013) http://en.wikipedia.org/w/index.php?title=Kalman_filter. xiii, 30

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

[3] RoboCup SPL Technical Committee: Standard Platform League rule book (2013) Only available online: www.tzi.de/spl/pub/Website/Downloads/Rules2013.pdf. 6, 47

[4] 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

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

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

[7] Orfanoudakis, E.: Reliable object recognition for the RoboCup domain. Diploma thesis, Technical University of Crete, Greece (2011) 11, 40

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

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

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

[11] 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

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

[13] 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) 13

[14] Topalidou-Kyniazopoulou, A., Spanoudakis, N.I., Lagoudakis, M.G.: A CASE tool for robot behavior development. In: Proceedings of the 16th RoboCup International Symposium. (2012) 13

[15] Topalidou-Kyniazopoulou, A.: A CASE (computer-aided software engineering) tool for robot-team behavior-control development. Diploma thesis, Technical University of Crete, Greece (2012) 13

[16] Karamitrou, M.: KMonitor: Global and local state visualization and monitoring for the RoboCup SPL league. Diploma thesis, Technical University of Crete, Greece (2012) 13, 46

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

[18] Welch, G., Bishop, G.: An introduction to the kalman filter. Technical Report TR 95-041, University of North Carolina at Chapel Hill (2011) 15, 18

[19] Röfer, T., Laue, T., Müller, J., Fabisch, A., Gillmann, K., Graf, C., Härtl, A., Humann, A., Wenk, F.: B-Human 2011 team description for RoboCup 2011. Technical report, The University of Bremen (2011) Only available online: http://www.b-human.de/downloads/bhuman11_tdp.pdf. 23

[20] Teh, B.: Ball modelling and its applications in robot goalie behaviours. Master's thesis, The University of New South Wales (2011) Only available online: http://cgi.cse.unsw.edu.au/~robocup/2011site/reports/Teh-BallModelling-GoalieBehaviour.pdf. 23

[21] Roth, M., Vail, D., Veloso, M.: A real-time world model for multi-robot teams with high-latency communication. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). Volume 3. (2003) 2494–2499 23

[22] Barrett, S., Genter, K., Hester, T., Khandelwal, P., Quinlan, M., Stone, P.: Austin Villa 2011 technical report UT-AI-TR-12-01. Technical report, The University of Texas at Austin (2011) Only available online: www.cs.utexas.edu/~pstone/Papers/bib2html-links/UTAITR1201-sbarrett.pdf. 23

[23] Tasse, S., Kerner, S., Urbann, O., Hofmann, M., Schwarz, I.: Nao Devils Dortmund team report 2011 (2011) Only available online: www.irf.tu-dortmund.de/nao-devils/download/2011/TeamReport-2011-NaoDevilsDortmund.pdf. 24

[24] Tasse, S., Hofmann, M., Urbann, O.: On sensor model design choices for humanoid robot localization. In Chen, X., Stone, P., Sucar, L.E., der Zan, T.V., eds.: RoboCup 2012: Robot Soccer World Cup XVI. Lecture Notes in Computer Science. Springer Berlin / Heidelberg (2013) to appear 24

[25] Chang, C.H., Wang, S.C., Wang, C.C.: Vision-based cooperative simultaneous localization and tracking. In: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA). (2011) 5191–5197 24