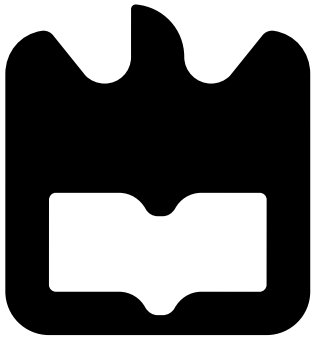
Universidade de Aveiro
2021

Ricardo Jorge
Maurício Dias

Actualização do Estado do Mundo para Equipas de Robôs Cooperativos

World Model Integration for Cooperative Robotic Teams



Ricardo Jorge
Maurício Dias

Actualização do Estado do Mundo para Equipas de Robôs Cooperativos

World Model Integration for Cooperative Robotic Teams

Tese apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Doutor em Engenharia Informática, realizada sob a orientação científica de José Nuno Panelas Nunes Lau, Professor do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro

Este projecto foi parcialmente suportado por Fátima Dias e António Dias; pela Universidade de Aveiro/DETI/IEETA; por fundos nacionais através da Fundação para a Ciência e Tecnologia (FCT), no contexto do projecto UID/CEC/00127/2013 e da bolsa de doutoramento SFRH/BD/116390/2016; e por fundos europeus FP7 no contexto da bolsa EuRoC CP-IP 608849.

o júri / the jury

presidente / president

Doutor António Manuel Rosa Pereira Caetano
Professor Catedrático da Universidade de Aveiro

vogais / examiners committee

Doutor António Paulo Gomes Mendes Moreira
Professor Associado Com Agregação, Universidade do Porto

Doutor António Fernando Macedo Ribeiro
Professor Associado Com Agregação, Universidade do Minho

Doutor Luis Paulo Gonçalves dos Reis
Professor Associado, Universidade do Porto

Doutor José Nuno Panelas Nunes Lau
Professor Associado, Universidade de Aveiro (orientador)

Doutora Pétia Georgieva Georgieva
Professora Associada, Universidade de Aveiro

agradecimentos

Obrigado pai, mãe, restante família, namorada, amigos, professores, colegas e membros do laboratório de robótica e da equipa CAMBADA que tantas vezes proferiram a frase "Então e essa tese?". Todos os vossos empurrões foram parte fundamental da força motriz necessária para finalizar este projecto. O sentimento de gratidão é enorme perante todas as condições favoráveis facilitadas pelo orientador Nuno Lau e restantes professores do IRIS-Lab, em particular os professores Bernardo Cunha, José Luis Azevedo e Artur Pereira, que mesmo não sendo oficialmente co-orientadores, sempre senti ao longo destes anos que o eram. Um agradecimento também ao professor Paulo Dias, que se disponibilizou para rever o documento.

Este projecto foi parcialmente suportado por Fátima Dias e António Dias; pela Universidade de Aveiro/DETI/IEETA; por fundos nacionais através da Fundação para a Ciência e Tecnologia (FCT), no contexto do projecto UID/CEC/00127/2013 e da bolsa de doutoramento SFRH/BD/116390/2016; e por fundos europeus FP7 no contexto da bolsa EuRoC CP-IP 608849.

acknowledgements

This project was partially supported by Fátima Dias and António Dias; by University of Aveiro/DETI/IEETA; by Portuguese National Funds through Foundation for Science and Technology (FCT), in the context of the project UID/CEC/00127/2013 and the PhD project grant SFRH/BD/116390/2016; and by European Union's FP7 under EuRoC grant agreement CP-IP 608849.

Palavras-Chave robótica móvel, sistemas multi-agente, fusão sensorial, coordenação

Resumo

Um sistema Multi-Robô pode beneficiar de uma atribuição cooperativa de tarefas para atingir um objectivo comum, bem como da integração de informação dos vários agentes para formar um melhor modelo do ambiente envolvente (World Model). Ter uma percepção duplamente precisa e responsiva é ainda um passo fundamental para uma efectiva cooperação em equipas de robôs, especialmente em ambientes dinâmicos, estocásticos e parcialmente observáveis, como é o caso das competições de futebol robótico RoboCup.

Este projecto de investigação foi desenvolvido no contexto da equipa de futebol robótico CAMBADA, da Universidade de Aveiro, e incluiu a participação em diversas competições locais e internacionais em eventos RoboCup, na liga MSL. Enquanto muita da literatura disponível apresenta resultados simulados, este projecto usou as competições MSL como uma plataforma de teste e validação das novas soluções apresentadas neste documento. O trabalho focou-se no desenvolvimento de novas soluções de fusão de dados numa perspectiva distribuída nesta equipa de futebol robótico, bem como no melhoramento de estratégias de coordenação de alto-nível.

Para tal, o projecto começa por consolidar a estimativa de estado (*state estimation*) local dos robôs, providenciando uma base sólida para técnicas mais avançadas - o ciclo de controlo principal foi revisto, as estimativas da pose e velocidade do robô foram melhoradas e um novo método de *tracking* de múltiplos objectos foi proposto. Com essa base, um método de *tracking* com sensorização distribuída foi desenvolvido, aprovisionando assim uma representação unificada dos oponentes no campo de jogo, não só para os jogadores de campo, mas também para o agente treinador, que não pode usar sensores.

Em continuação, foram discutidas técnicas de coordenação de agentes e um novo método para eleição de um agente foi proposto, testado e validado.

Com uma melhor representação do estado do mundo, um método de eleição de um líder e uma arquitectura de software modular madura em prática, foi possível desenvolver uma nova ferramenta de jogadas estudadas (CR7) que é *user-friendly*, robusta e eficiente em ambientes de competição.

Com uma abordagem bottom-up, este projecto foi incrementalmente tocando uma vasta gama de áreas e assim cumpriu os objectivos inicialmente propostos, bem como ajudou a CAMBADA a reafirmar a sua posição como uma das melhores equipas da MSL a nível mundial.

Keywords mobile robotics, multi-agent systems, sensor fusion, coordination

Abstract

A Multi-Robot system can benefit from cooperative task assignments to achieve a common goal, as well as from merging information from all agents to form a better World Model. Highly accurate and responsive perception is still a fundamental step for effective cooperative robotics teams, especially in dynamic, stochastic and partially observable environments like the ones provided by the RoboCup soccer competitions.

This research project was developed in the context of the CAMBADA RoboCup Middle-Size League robotics soccer team, from University of Aveiro, including the participation in local and international RoboCup competitions. While most of the literature presents simulated results, this project used the MSL competitions as a bench-marking and verification test-bed for the new solutions proposed in this document. The work aimed at developing new solutions for data fusion in a distributed approach in this team, as well as improving the existing high-level coordination strategy.

In order to do so, the project starts by consolidating the robots local state estimation, providing an essential solid baseline for more advanced techniques - the main control cycle was revised, the robot pose and velocity state estimation was enhanced, and a new method for multi-object tracking has been proposed. A distributed sensing multi-object tracking module was built on top of that, providing a unified representation of the opponents on the soccer field, not only for the field players, but also for the coach agent, that is not able to use any sensors.

Following, agent coordination techniques are discussed and a new method for leader election is proposed, tested and validated.

With the enhanced world model, the leader election method and the matured modular software architecture in place, it was possible to develop a new set-play engine (CR7) that is user-friendly, robust and performant in competition environments.

In a bottom-up approach, this project has incrementally touched a wide range of areas and has successfully met the initial objectives as well as helped to reinforce CAMBADA's position as one of the world-leading MSL teams.

Contents

List of Figures

List of Tables

Acronyms

CAMBADA Cooperative Autonomous Mobile roBots with Advanced Distributed Architecture. 5, 6, 8, 9, 11, 19, 20, 22, 23, 30, 34, 45, 47, 61, 63–65, 72, 78–80, 82

CAN Controller Area Network. 7

CR7 CAMBADA Cooperation fRamework 7. vi, 63, 65, 67, 69, 70, 75–80, 83, 84

FPR False-Positive Rate. 42, 44–46

KF Kalman Filter. 14–17, 22, 23, 35–37, 82

MAS Multi-Agent System. 2, 4, 10, 18, 19, 29, 61, 84

MSL Middle-Size League. 4, 6, 8, 9, 25, 31, 38, 40, 45, 50, 51, 79, 84

PID Proportional–Integral–Derivative. 23, 27

RANSAC Random Sample Consensus. 31, 32

RtDB Realtime Data Base. v, 8, 9, 56, 72

Chapter 1

Introduction

1.1 Multi-Agent Robotics Systems

A **software agent** is a persistent and goal-oriented computer program capable of running without continuous direct supervision. Based on sensed events, it decides and executes one or more actions to achieve a stated goal. Agents may also be able to communicate with other agents or people in order to achieve this goal.

One of the most famous and well-accepted definition of **agent** comes from Russell and Norvig [Russell and Norvig, 2020]:

> "An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators."

Autonomous **robotic systems** are systems equipped with sensors, a processing unit capable of running a software agent and physical actuators. This set of capabilities allows the robot not only to sense, but also to act on the physical environment, with the capacity of changing it. Dealing with physical objects actually creates a new set of challenges that need to be taken into account when designing the software agent.

Usually, when designing agent-based systems for a certain target application, there are two aspects that need to be balanced: **reactive action** and **deliberative decision**. Purely reactive agents generate control signals directly in response to sensor measurements [Brooks, 1989], therefore the performance of these systems tends to degrade when the difficulty of the task increases. On the other hand, deliberative agents maintain an internal state and typically search through a space of possibilities (actions/behaviours), predicting the outcome of each possible action. However, this search space can be so exhaustive that it takes too much time to be used in some **real-time** applications.

As an example, a robotic arm in a factory may be able to behave in a deliberative way, if it can stop to plan its motion and then execute it, but a soccer-playing robot must be able to react to sudden changes in the environment and may not have so much time available to

plan an action. Once the target application requirements are settled, the trade-off between optimisation and performance naturally arises - and this explains why most real-time systems are designed with time efficiency as a critical aspect, sometimes overlapping complex or more demanding algorithms in terms of priority.

In any case, to reach a near-optimal decision for an action, a mobile robotic system must maintain a consistent internal representation of its environment, usually called the **World Model**. It contains not only information about the robot self state (usually position, orientation, linear and angular velocities, etc.), but also relevant data about the robot surroundings. This World Model building process is commonly called *integration* and consists in taking not only raw and noisy data (usually coming from sensors), but also historical data and turning them into higher order useful and filtered information. In real-world deployments of mobile robotics applications, agents have to handle uncertainty, which can arise from different sources. First of all, the environment is only partially observable and usually also non-deterministic, continuous and dynamic. Sensors have associated noise and physical limitations. Moreover, robots are real-time systems, so most algorithms and models used during computation have to comply with time restrictions, sacrificing accuracy, if needed, and, finally, the actuators that directly interact with the environment can also be inaccurate.

Coping with all these restrictions while creating an accurate *World Model* representation poses a problem, since computational resources are naturally limited in all real applications.

In a **Multi-Agent System** (MAS) [Wooldridge, 2009], a group of autonomous agents interact with each other via a number of different interaction models for coordination [Jennings, 1993, Durfee, 1999], collaboration [Levesque et al., 1990, Grosz and Kraus, 1996] and negotiation [Davis and Smith, 1983, Parker, 1998]. The group activity should not be seen only as a result of simultaneous individual behaviours, since the group goal, if it exists, should be taken into account by each member [Cohen and Levesque, 1991].

It has already been demonstrated that distributed sensor fusion can enhance the belief by synergistically merging data from different agents [Merino et al., 2005, Sun et al., 2017, Battistelli and Chisci, 2016] to derive a better approximation of the World Model than would be possible with each one individually [Elmenreich, 2002].

A remaining open challenge is how to efficiently make a team of autonomous agents cooperate in a responsive and coherent way in a highly dynamic and stochastic environment. Agents are expected to react to rapid changes in the domain and therefore robustness is required against not only sensorial noise, but also individual agent or sensor failures and unreliable communication means.

1.2 The RoboCup Initiative

RoboCup [Kitano et al., 1997] is an enormous commitment from scientific and research groups from all around the world on developing robots that one day will help humans in their daily lives. The RoboCup event is hosted by a different country every year[1] since 1997 with a series of competitions. This challenging environment not only gives teams extra motivation to keep participating every year, showing new skills and other improvements, but also provides a discussion forum for researchers in this robotics domain, the Symposium, in which teams have the chance to share ideas, knowledge and implementations, in order to make scientific process in the field of mobile autonomous robotics. By organising an annual event, RoboCup creates opportunities for researchers to test their work and exchange technical and scientific information, while entertaining and educating the audience at the same time, making the general public more aware of the state of the art in robotics and its current problems.

Moreover, with the ambition of keeping a high level of interest and competitiveness, the RoboCup Federation has created a challenging milestone for the RoboCup Soccer leagues:

> *"By the middle of the 21st century, a team of fully autonomous humanoid robot soccer players shall win a soccer game, complying with the official rules of FIFA, against the winner of the most recent World Cup."*

We are still far from this goal, but a significant progress has been made so far. The competition is divided into several leagues, each one tackling different problems, whether restricted to the software level only, or on both hardware and software, in a multi-agent or single-agent environments, cooperative and/or competitive.

> *"RoboCup (Soccer) is an attempt to foster AI and intelligent robotics research by providing a standard problem where wide range of technologies can be integrated and examined. In order for a robot team to actually perform a soccer game, various technologies must be incorporated including: design principles of autonomous agents, multi agent collaboration, strategy acquisition, real-time reasoning, robotics, and sensor-fusion."* [Kitano et al., 1997]

Just like in human soccer, robot players must work as a team in order to score and defend their own goal at the same time, while adapting to new situations and different opponents. This makes soccer a rich domain for fundamental exploration and development of multi-agent solutions related to perception, sensor fusion, world model representation, high-level decision and coordination and communication.

[1] Except for 2020 - the event was cancelled due to the COVID-19 pandemic.

Figure 1.1: The Middle-Size League final match in RoboCup 2013.

From simulation to real robots, from anthropomorphic to wheeled robots, there are several different soccer leagues in RoboCup, one of which is the Middle-Size League (MSL) - Figure 1.1.

In this league, robots play soccer autonomously in a $22m \times 14m$ field with a standard Size-5 FIFA ball. The rules[2] of the matches are based on the official FIFA rules, with a few changes to adapt for the playing robots. While having a huge potential for a variety of applications, MAS are extensively tested and benchmarked in RoboCup and this league provides an excellent testbed for autonomous robotic teams in stochastic and highly dynamic environments. A soccer match cannot be overlooked as a testbed, since it resembles more the real world (complex and semi-structured) than a research lab. Agents must be resilient to unexpected situations, since the opponent team actions can only be predicted up to a certain point. Furthermore, the state of the ball and the opponent robots can change expeditiously at any time.

Specifically in the RoboCup Middle-Size League (MSL) and regarding sensor fusion for world modelling [Silva et al., 2011], teams have been applying Kalman Filters for the ball position estimation [Lauer et al., 2006, Y. et al., 2006]. More recently, teams have also been tackling problems such as visual information fusion from multiple cameras to get a better estimation of the ball trajectory in the air [Neves et al., 2015], while integrating the Inertial Measurement Unit (IMU) information to optimize the localisation process.

Regarding coordination, some interesting work has also been done on utility-maps based approaches [Neves et al., 2015], coordination-aware architectures for multi-agents systems [Skubch et al., 2009] and multi-robot cooperative perception for object tracking [Ahmad and Lima, 2013]. Team coordination paradigms have been applied also in robotic soccer to select roles in the game and define strategies [Spaan and Groen, 2003b, Lau et al., 2011].

[2] `https://msl.robocup.org/rules`

Multiple centralised direct supervision and decentralised mutual adjustment coordination approaches have been proposed within the context of RoboCup leagues [Paquet et al., 2004]. Among these approaches, the decentralised approach is more flexible. These algorithms have mainly been used in high-level decisions, to select roles or plan tasks to be performed by each team member to achieve a goal. However, they can also be used when integrating information from several agents [Olfati-Saber et al., 2007], in which consensus problems are applied to distributed sensor fusion [Olfati-Saber and Shamma, 2005] and belief propagation [Stroupe et al., 2001], for instance.

1.3 CAMBADA

Cooperative Autonomous Mobile roBots with Advanced Distributed Architecture (CAMBADA) [Neves et al., 2010] is the RoboCup Middle-Size League robotic soccer team from the University of Aveiro, coordinated by the IEETA IRIS (Intelligent Robotics and Intelligent Systems) group. The project involves people working on several different areas: mechanical structure of the robot, hardware architecture and electronics, software on image analysis and processing, sensor fusion and decision and cooperation architecture on the high-level software.

Figure 1.2: The CAMBADA team robots.

The team development started in 2003 and a steady progress was observed since then, with the participation in several competitions, both national and international, including RoboCup World Championships and the annual Portuguese Open Robotics Festival. The CAMBADA team was world champion in 2008 and is the national champion since 2007, and, overall, has achieved **9 1st places**, **9 2nd places** and **9 3rd places** in all competition tournaments it has participated in.

Apart from the soccer tournament, there are two additional challenges in RoboCup with special requirements: the **Scientific** and **Technical Challenges**.

The **Scientific Challenge** requires teams to do a small pitch about relevant developed work in the context of MSL for other teams to evaluate several criteria: presentation, novelty, interest for the league, scientific/technical complexity, importance of experimental results and relevance of the published results presented as a support for this challenge. In the end, all weighted averages given by other teams are summed up and the score ranking determines the places in this challenge.

On the other hand, the **Technical Challenge** has been an opportunity for teams to test, improve and show their coordination abilities. With some variations over the years, this challenge has been focused on a sequence of tasks that require multiple robots.

The CAMBADA team won 3 Technical Challenges (also, 1 second place) and 3 Scientific Challenges (also, 3 second places and 1 third place).

1.3.1 Distributed and Modular Architecture

Throughout the years, the CAMBADA robot hardware platform has suffered changes to accommodate the demands. The latest major redesign was performed in 2013, keeping the positive aspects of the previous version, but with special detail to modularity. As the team name conveys, the hardware architecture is distributed, meaning that there is a dedicated controller for each component of the robot, as depicted in Figure 1.3.

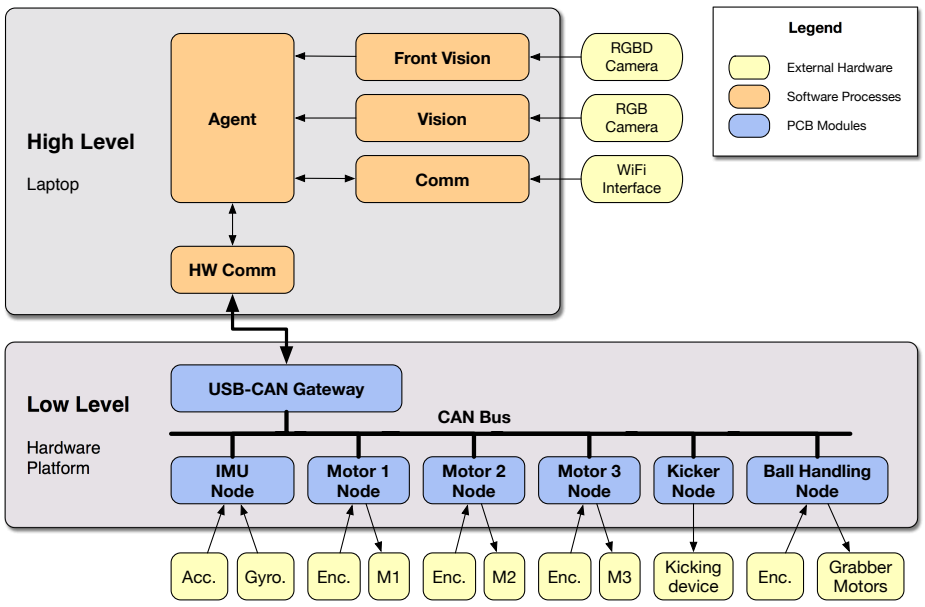


Figure 1.3: The general architecture of a CAMBADA robot.

The main sensor of these robots is the camera - a catadioptric vision system composed

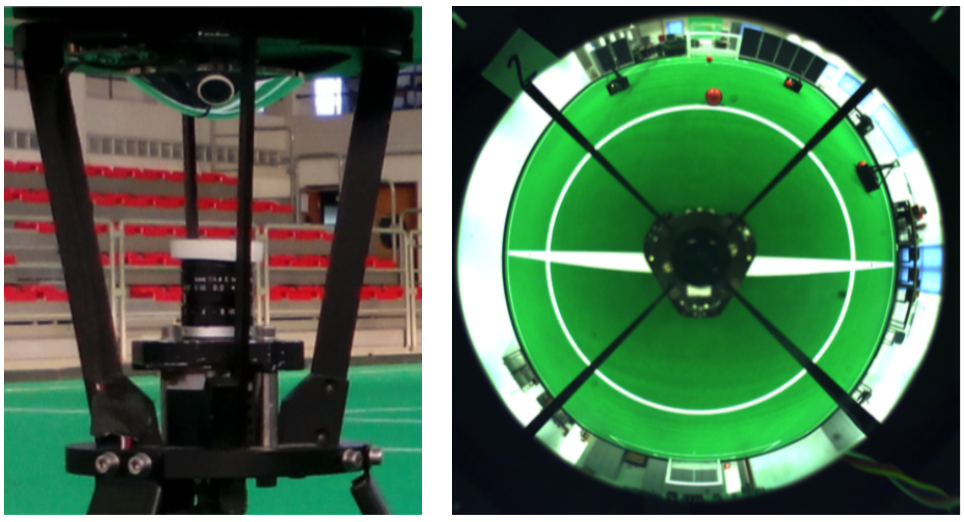
of an industrial camera and a mirror, which allows the robot to see 360 degrees around it. The main processing unit (a 12-inch laptop) is used to process the images coming from the camera at 50 Hz rate (`Vision process`), to receive information from team-mate agents and also from sensors installed in the robot platform via the USB-CAN gateway.

The `Agent` process uses sensor fusion techniques to continuously integrate all the aforementioned input information on a world model representation, which is then used to make a decision on the next action.

The `HWComm` process is responsible for sending the targets calculated by the `Agent` to the low-level hardware platform for execution, as well as for getting sensorial information from the physical platform. `HWComm` communicates with a gateway [Pinto, 2020] that translates the messages carrying the actions to CAN-bus messages, and also messages in the opposite direction (from sensors to the laptop).

Lastly, the electronic modules listen to the respective CAN messages and execute the actions accordingly - the Kicker board actuates on the solenoid that is used to kick the ball and controls the energy storage in the 450V capacitor, Motor boards set the velocities for the motors and provide feedback regarding odometry info, the Grabber board sets the ball handling mechanism behaviour and also provides feedback on the current grabber "arms" angle, etc.

1.3.2 The Vision System



(a) Catadioptric vision system (b) Example of an acquired frame

Figure 1.4: CAMBADA vision system and frame example.

With the technological improvements and increase in the affordability of image sensors and video cameras, these are nowadays capable of acquiring high-resolution images at high frame

rates. Hence, this generates a great amount of visual information that has to be processed in a well defined period of time. These real-time constraints pose a problem of efficiency, which should have a minimum impact on accuracy.

The challenge of playing soccer autonomously requires real-time perception of the overall environment in order to allow self localisation, team-mate and opponent position estimation and, of course, determination of the ball position and its velocity. To achieve this, most robots in the MSL use a catadioptric omni-directional vision system set composed of a regular video camera pointed at a hyperbolic mirror. An image of the physical set-up of the digital camera and the mirror on a CAMBADA robot can be seen in Figure 1.4a. The camera installed on the robots can provide images with a resolution of 1280×1024 pixels at 50 FPS (Figure 1.4b). This type of setup ensures an integrated perception of all major features of interest on the robot surrounding area in each frame.

1.3.3 Communication Model

As the information must flow between these processes within a single robot, as well as some information must be shared between robots, the need for a communication model naturally follows. For inter-robot and inter-process communication, CAMBADA developed an open-source[3] middleware called Realtime Data Base (RtDB) [Almeida et al., 2004]. This library provides a seamless access to the complete team state using a distributed database, partially replicated to all team members, as depicted on Figure 1.5.

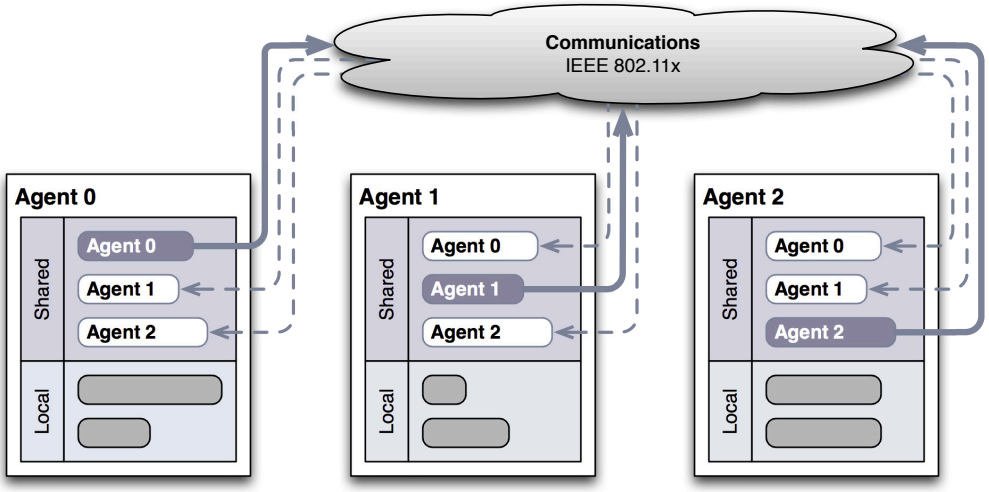


Figure 1.5: Example of RtDB functionality with 3 agents. RtDB items can be shared (broadcasted to all team members) or local (can be used in processes running in the local agent).

In a multi-agent architecture perspective, information such as the pose of the players, their velocity, intention, ball position, ball velocity and other relevant information for the

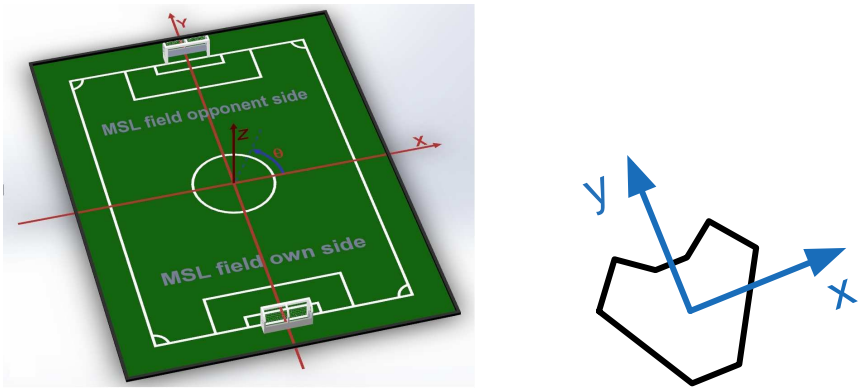
[3] `https://github.com/CAMBADA/cambada-rtdb2`

team are included in the RtDB. This is particularly important, for example, when a certain robot does not see the ball, as then it can use a team-mate's information as a guide. The modular structure of the RtDB (particularly of its second generation) [Silva, 2017] allows to easily add, remove and modify the shared items dynamically at runtime.

As previously stated, this library is also intensively used as an inter-process communication mean. This is achieved by defining RtDB local items that are not broadcasted to other robots, but are still accessible by other processes running on the same agent environment. This local information may include sensorial data from the vision system and the hardware platform and also commands that are sent to the low-level control layer through a hardware gateway.

A software process called `Comm`, which is included in the RtDB suite, is responsible for handling the Wi-Fi communication. It sends shared RtDB items to the team multicast group and also receives data from other agents and updates the appropriate shared RtDB sections [Santos et al., 2010]. Together, RtDB and `Comm` are used by CAMBADA (and several other teams in MSL) to solve the issues of intra-robot and inter-robot communication at the same time, in a flexible and modular way.

1.3.4 Coordinate Systems Definition



(a) Absolute (world) coordinates (b) Relative (robot) coordinates

Figure 1.6: Absolute and relative coordinates used in CAMBADA, as defined in the MSL standard definition document

The coordinate system adopted by the CAMBADA team follows the MSL standard definition [MSL Technical Committee, 2016] for the two types of coordinate systems mentioned throughout this document: robot coordinates (Figure 1.6b) and world coordinates (Figure 1.6a). Robot coordinates (or relative coordinates) are defined with origin centred with the robot, the y-axis pointing to the front of the robot and x-axis pointing to the right side of the robot. When referencing world coordinates (or absolute coordinates), the origin lies in the field center, and the system follows the right-hand rule with the y-axis points in the direction of attack (to the opponent goal) and the z-axis point up to the ceiling.

1.4 Project Motivation

One of the main research goals on distributed autonomous agents in a Multi-Agent System (MAS) is the development of mechanisms to improve cooperation and coordination techniques in order to achieve a common goal.

When compared to centralised approaches, distributed systems present a major advantage: they are usually more resilient to failures. However, it is much more difficult to implement a real fully distributed system, when comparing with a centralised approach. In a centralised approach, one "master" agent has control over one or more "slaves" and by giving direct or indirect orders to them, based on a strategy, it is able to achieve the goal. The distributed approach is different in the sense that there is usually no "master" controlling the process. Agents taking part in a MAS must have some sort of communication and/or negotiation skills, whether it is explicit or implicit. The so-called team gain should not be merely seen as the result of the combination of individual behaviours, since teamwork and the aforementioned skills must be put into practice to efficiently achieve a common objective.

A software architecture for MAS should encompass a communication model that supports agreement from World Model integration up to higher level team coordination. In fact, the software architecture plays a major role in guaranteeing that the solution is robust, easy to understand, to change and to tweak without compromising the final result.

While having a huge potential for a variety of applications, MAS are extensively tested and benchmarked in robotic soccer competitions, and in RoboCup in particular. The RoboCup Middle-Size League provides an excellent test-bed for autonomous robotic teams in stochastic and highly dynamic environments. A soccer match cannot be overlooked as a test-bed, since it resembles more the real world (complex and semi-unstructured) than a research lab. Therefore, agents must be resilient to unexpected situations, since the opponent team actions can only be predicted up to a certain point.

In this league, high-level coordination has been used for some years as an essential tool for selecting which role each robot should play - being an attacker, defender, goal-keeper, etc. However, being a networked Multi-Agent System, a robotic team can also perform a much more efficient data integration if information from several agents is taken into account. In order to achieve that, agents must communicate, not only to share their information, but also to achieve an agreement regarding the World Model. As an example, if multiple ball candidates are detected by robots, which one should be considered the "real" ball of the game?

Some major challenges on this league are the real-time constraints, the need for accurate vision calibration, dealing with outliers and accounting for perception and process noises. When trying to merge information from different robots, new problems arise: sensorial data may be collected at different moments by the multiple robots and they must deal with communication problems such as delay and the possibility of intermittent connection.

1.5 Objectives

This research project aims at developing new methodologies for data fusion in a distributed approach and its evaluation in the real robots of the robotics soccer team CAMBADA [Neves et al., 2010] (acronym of Cooperative Autonomous Mobile roBots with Advanced Distributed Architecture), from University of Aveiro. Moreover, the acquired know-how and a better World Model representation will allow the exploitation of new solutions for team coordination and cooperation to enable teams of autonomous robots to accomplish complex collective tasks.

This research project aims at developing new solutions for data fusion in a distributed approach in a real robotics soccer team, as well as improving the current high-level coordination strategy with novel methodologies. Therefore, two main objectives were set:

1. Development of a new integration module, by applying cooperative sensor fusion techniques to enhance the global perception of the environment;

2. Improvement of the high-level coordination strategy of the team.

While most of the literature presents simulated results, this project was developed over a team of real robots and tested in a highly-dynamic, stochastic and partially observable environment such as the one provided by the RoboCup soccer competitions.

This is particularly important because any proposed solutions must effectively and efficiently work within the existing real scenario and not a conceptual (sometimes unrealistic) simulated environment. Although not particularly detailed in this document, the fact that the application is in actual robots (with inherent hardware specificities, handicaps and occasional failures) posed some extra challenges throughout the development of this project. The nature of the benchmark scenario (robotic soccer tournaments) in some cases did not allow the extraction of extremely thorough results, nor the re-iteration of test scenarios with different algorithms for comprehensive comparison.

1.6 Thesis Structure

This document is structured in six chapters, being the first this Introduction, which includes the motivation for this work and states the context and the objectives of the thesis.

Chapter 2 makes an overview of the most common methods for state estimation and introduces the concept of sensor fusion, where a review over distributed sensor fusion methods is also made. The process synchronization in CAMBADA is detailed and some improvements on the robots pose and velocity estimation are performed, as well as improvements on the robot motion control.

Chapter 3 expands on the problem of estimating the state of multiple objects to a distributed perspective, in which information from different agents is aggregated to improve the global world model.

In Chapter 4, the main focus is on Multi-Robot Systems Coordination methods, including review of existing models for strategic position and role assignment. A new method is proposed for leader election that overcomes two limitations found on existing consensus algorithms.

Following, Chapter 5 introduces a new Coordination framework that makes use of all the work devised in the previous chapters, resulting in a powerful yet easy-to-use setplay engine, fully integrated with the existing software architecture, overcoming a number of identified drawbacks of the existing solutions.

Finally, the last chapter, Chapter 6 closes the document with the major contributions of this project and some final considerations.

Chapter 2

State Estimation and Control

In order for a robotic system to perform its tasks towards the predefined goal, it needs information about itself and also its environment and, therefore, state estimation emerges naturally as a common issue for agents running on robotics applications.

A *state* holds all information of the robot and its environment that was deemed to be relevant for the agent to make a decision, hence influencing the future state. While there are static state variables (variables that never change during the robot operation lifetime), in real robotics applications the majority of the variables are prone to change over time for two reasons: on one hand, the agent is expected to act on the environment, on the other hand, the environment is usually dynamic itself. The *state* may comprise information regarding the robot pose, velocity, hardware status, etc. but also information about the robot surrounding environment.

In real mobile robotics applications, agents have to handle **uncertainty**, which can arise from different sources. First of all, the **environment** is partially observable and usually also non-deterministic, continuous and dynamic. Any **sensor** has an associated noise and physical limitations and since robots are real-time systems, most algorithms and models used during **computation** have to comply with time restrictions, sacrificing the accuracy, if needed. Finally, the **actuators** that directly interact with the environment can also be inaccurate, depending on many different factors.

The challenge of understanding the environment in the face of uncertainty is very important in the field of robotics. The ability to cope with these uncertainties is critical to build robots that can achieve their task in unpredictable environments. This was the motivation for a new approach to robotics called **probabilistic robotics** [Thrun, 2000], which makes use of probability theory to explicitly handle uncertainty. Probabilistic models provide a powerful and consistent means of describing uncertainty and lead naturally into ideas of information fusion. Information is represented by probability distributions over a space of possible hypotheses. The major advantage of probabilistic approaches is that they are typically more resilient to uncertainty [Thrun et al., 1998].

"A robot that carries a notion of its own uncertainty and that acts accordingly is superior to one that does not."

– Sebastian Thrun [Thrun, 2000]

2.1 Common State Estimation Methodologies

In this section, an overview of the state-of-the-art in state estimation is performed, focusing on methods widely used and validated in robotics applications.

2.1.1 Discrete Kalman Filter

The **Kalman Filter** [Kalman, 1960] (KF) is probably the most studied and implemented technique for optimal state estimation in linear Gaussian systems, by modelling the state variables with a mean x and a covariance P. Being a recursive filter, it is computationally efficient and also supports estimation of states in a way that minimizes the mean of the squared error (hence, it is called optimal). This filter is useful when the measurement can be modelled as a linear combination of the state (with added Gaussian noise).

The Kalman Filter represents beliefs by a mean and a covariance and works under the following assumptions:

- The state transition probability is linear in their arguments and with added Gaussian noise:

$$x_k = A_k x_{k-1} + \varepsilon, \text{where } \varepsilon \sim \mathcal{N}(0, Q) \tag{2.1}$$

- The measurement probability is also linear and with added Gaussian noise, with the following observation model:

$$z_k = H x_k + \delta, \text{where } \delta \sim \mathcal{N}(0, R) \tag{2.2}$$

- The initial belief must be normally distributed with some mean x_0 and covariance P_0.

At the core of the KF lies a form of feedback control with two main types of operations: **predict** and **update** (Figure 2.1). The *predict* step projects the current state estimate ahead in time and the *update* adjusts the projected estimate by an actual (noisy) measurement at that time.

Predict Step

The predict step models how the system evolves through a known and pre-defined linear model, before incorporating the actual measurements.

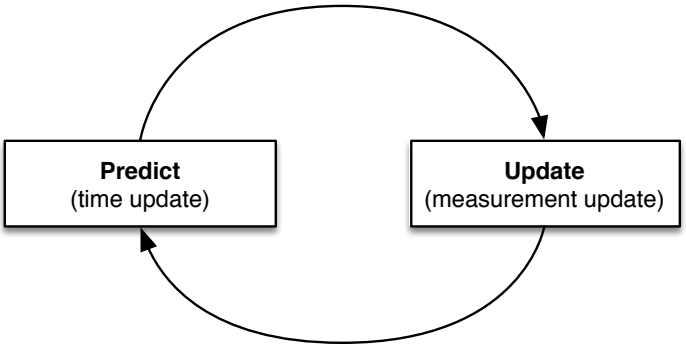


Figure 2.1: The discrete Kalman Filter cycle.

$$x'_k = A_k x_{k-1} + B_k u_k \tag{2.3}$$

$$P'_k = A_k P_{k-1} A_k^T + Q_k \tag{2.4}$$

In equation 2.3, the matrix A_k models the transition of the state between steps, while B_k relates the new state with the control vector u_k. P_k represents the covariance matrix that models the state estimation uncertainty, which is affected by the state transition itself (equation 2.4), with Q_k representing the process noise covariance.

Note that the state transition probability is *linear Gaussian* since it is a linear function with added Gaussian noise, in which x_t and x_{t-1} are state vectors and u_t is the control vector at time t.

Update Step

For the update step, equation 2.2 is put in place, in which the H matrix models the linear relation between a measurement and the state vector. The following equations rule the update step of the KF:

$$K_k = P'_k H^T (H P'_k H^T + R_k)^{-1} \tag{2.5}$$

$$x_k = x'_k + K_k(z_k - H x'_k) \tag{2.6}$$

$$P_k = (I - K_k H) P'_k \tag{2.7}$$

In the update stage, noisy measurements z_k are used to update the state x_k. The first step is to compute the *Kalman gain* K_k (equation 2.5), taking the measurement noise covariance R_k into account, and then generate *a posteriori* state estimate by incorporating the measurement z_k (equation 2.6).

K_k is used to determine how much of the measurement goes into the new state estimation, by multiplying it with the *innovation* - the difference between the actual measurement and the expected measurement. Finally, the error covariance estimate is obtained (equation 2.7).

Practical Considerations in Robotics Applications

It is possible to conclude that the Kalman Filter method provides a very convenient way for it to be applied to any kind of online real-time processing application. However, applying a Kalman Filter to a real robot theoretically requires accurate knowledge about its sensors and actuators models - it is often impossible to obtain a model of these entities to which the KF can be applied directly. Due to this recurring gap between theoretical models and practical applications, the current section describes the method used to tune the Kalman Filter used throughout the following chapters.

Prior to the operation of the Kalman Filter, the measurement noise R can easily be measured, since the process must be measurable in some way for the filter to be applied later. It is then generally practical to take some off-line sample measurements in order to determine the variance of the measurement noise.

In contrast, the process noise covariance Q is usually more difficult to obtain, since in real applications, it is typically hard to directly observe the process of which state is to be estimated.

In robotics applications, it is frequent to have a dynamic measurement error covariance R, i.e. one that is not constant over time. For example, for a soccer playing robot, one of the measurements taken might be the robot position on the field, given by the detection of the field lines. When the robot is stopped this information is more reliable, but it tends to degrade with the increase of robot velocity due to vibrations on the camera that occur when the robot is moving. This extra noise might be integrated on the measurement covariance R.

When information is available from more than one sensor, after a single predict step, several update steps can be performed, one for each sensor, with the respective measurement noise covariances or, alternatively, a single update step with all measurements and an appropriate observation model matrix H. When a particular sensor is able to get information for a sub-set of the variables that constitute the state, then a high (ideally infinite) measurement covariance is given for any remaining variables.

In a final note, after performing off-line sampling on the process in order to reach an approximation for the covariance matrix R, it is usually a good practice to use a groundtruth system (whenever possible) to compare the state estimate with the "real" state to validate any necessary fine-tuning on matrices R and Q.

2.1.2 Non-Linear Kalman Filters

In robotics applications, some systems do not meet the linear assumptions - their measurements are not linear functions of the state and the next state is rarely a linear function of the previous state. In fact, when the dynamics of the system are too complex to model using linear assumptions, other filters can perform better.

The **Extended Kalman Filter** (EKF) and the **Unscented Kalman Filter** (UKF)

[Wan and Van Der Merwe, 2000] are other common approaches developed to deal with non-linear systems but at higher computational costs.

The EKF, also called the First-Order Filter, uses the Taylor Series expansion to linearise the model functions under certain conditions. However, it is only reliable for systems that are almost linear on the time scale of the updates, as according to [Julier and Uhlmann, 2004]. This led the author to propose the UKF [Julier and Uhlmann, 1997] in 1997, that is also a nonlinear Kalman filter which approximates the probability density function using a deterministic sampling of points (called *sigma points*) around the mean.

Another strategy is to change the dimension of the state vector, based on changes of the target object dynamics - the **Variable State Dimension Filter** (VSDF) [Bar-Shalom and Birmiwal, 1982]. In this variation of the Kalman Filter, the size of the state vector estimated by the filter is allowed to vary in an arbitrary way. For example, local states may be added to or removed from the state vector at any time.

Interacting Multiple Models (IMM) filter [Bar-Shalom et al., 2002, 2005] can also be used to mix different process models. The complex dynamics of the system is modeled using a weighted blending of distinct models.

Multiple Hypothesis Tracking (MHT) [Reid, 1979] is a different approach, in which multiple independent state estimators are used in parallel. Afterwards, a decision process selects the most relevant, usually based on specific external domain knowledge.

2.1.3 Particle Filters

Monte-Carlo [Metropolis and Ulam, 1949] based approaches are currently the most used when Kalman Filter assumptions are not met or when major discontinuities in the output values are expected. These filters are usually called **Particle Filters** [Del Moral, 1996] and use a genetic type mutation-selection sampling approach, in which each particle has a certain likelihood weight that represents the probability of that particle being sampled from the probability density function. In the resampling step, the particles with negligible weights are eventually replaced by new particles in the proximity of the particles with higher weights.

In recent formulations [Cunningham et al., 2020], it is a common approach to duplicate some particles and remove others, i.e. the same particle is selected more than once for the new population.

This type of filters has been extensively used in robot localisation problems when the map of the environment is previously known [Potthast et al., 2019, Doucet et al., 2001]. The essence of this filter is that particles that are more consistent with the measurements are more likely to survive, when compared to other particles. As a result, when applied to a localisation problem, places of high probability will collect more particles and therefore will be more representative of the robot posterior belief.

2.2 Perception and Sensor Fusion

Gathering information from the environment is one of the most important tasks for mobile robots. Various measurements are taken from different sensors which can later be transformed into useful information.

Siegwart defined two functional axes for sensors [Siegwart and Nourbakhsh, 2004]. In one of the dimensions, *proprioceptive* (measure values from the robot itself) and *exteroceptive* (gather data from the environment) sensors. On the other hand, sensors can be also classified as *active* (these work by exciting the environment in some way in order to measure the reaction) or *passive* (simply measure ambient energy entering the sensor).

The degree to which sensors can contribute to the **World Model** should not be overlooked. Every sensor reading is influenced by some sort of noise or error and this is why, whenever possible, multiple sensors are taken into account, their measurements are merged using **Sensor Fusion** techniques through an **Integration** process.

> "Data fusion is the process by which data from a multitude of sensors is used to yield an optimal estimate of a specified state vector pertaining to the observed system."
>
> – Richardson and Marsh [Richardson and Marsh, 1988]

Common Integration methodologies often use not only the most recent data from the sensors, but also take history into account.

2.2.1 Distributed Sensor Fusion

In an attempt to improve world modelling on MAS, sensor fusion techniques have been applied to multi-agent scenarios as well, usually by merging raw data or information gathered by multiple agents.

The environment provided by the RoboCup competitions actively encourages the developments of distributed sensing solutions, once the robotic team is considered a distributed sensor processing network. Fusion of information that comes from different sources and transmitted through a WiFi network creates a challenge that starts with data timestamp synchronization.

A software agent for robotics applications must comply with strict time demands in a potentially highly dynamic environment, therefore the data shared between robots must be reduced to a minimum as well as the computational demands to combine their observations. For example, sending every raw camera image frame to every other agent is an unfeasible solution because it is not scalable and spreads the complexity to the neighbour agents, defeating the purpose of having a distributed system (with distributed local processing). In an ideal scenario, each agent should pre-process their raw sensor data in order to share only relevant information extracted from it to its agent teammates.

These techniques have been already investigated over two main areas of interest in robotics:

- **Object tracking**:

  The objective of object tracking is to pursue an object of interest with very little to no information about its future move. In order to comply with the timing demands, point-based object tracking is usually applied, while still using the term object tracking to refer to it. As already stated in a previous section, Kalman Filters (and its extensions) have been extensively applied to object tracking problems. A few examples include real-time camera pose tracking [Wang et al., 1998, Erdem and Ercan, 2015], tracking of objects for grasping [Petryk and Buehler, 1997, Ilonen et al., 2013] and prediction of future object locations [Rembold et al., 1998].

  Because they provide a good alternative for tracked objects of which motion either has major discontinuities, is too random/unpredictable or can not be modelled at all, particle filters have also been applied to this type of problems [Kim and Sim, 2010].

  More recently, distributed object tracking techniques were also applied on wireless sensor networks [Chen et al., 2016], with results showing improvements on the overall accuracy, while reducing communications and the energy required when compared to a centralized solution.

- **Localisation and Mapping**:

  Kalman Filters have also been used to localise robots within mapped environments. This approach is applied when several position estimates are available from multiple sources (either synchronous or asynchronous) [Gutmann et al., 1999, Larsen et al., 1999, Moreno et al., 1999, Sasiadek and Hartana, 2000].

  While some approaches rely on having a map of the environment, some research has been ongoing in SLAM (Simultaneous Localization and Mapping), not only in single-agent scenarios [Thrun et al., 1998, Brown and Donald, 2000], but on MAS as well [Binns et al., 2002, Cunningham et al., 2010].

2.3 Process Synchronization In CAMBADA

The efficiency of the state estimation algorithms is directly related to the quality of the data provided for the filters to integrate. In this section, a review over the process execution pipeline in CAMBADA is made and a proposed modification is presented, resulting in a reduction of the time period jitter on the communications with the low-level/hardware, which in turn results in a positive impact on the Integration process.

2.3.1 Design Choice Considerations

In CAMBADA there are mainly 3 processes running in sequence every cycle (at a frequency of $50Hz$): `Vision`, `Agent` and `HWComm`, in this order - in general terms, these processes mirror the three common steps of robotic agents: perception, decision and action. Once a precedent process finishes its task for that cycle, the subsequent process is awaken. This synchronisation is achieved using the `PMan` (Process Manager) library [Pedreiras and Almeida, 2007].

Each process has a task to accomplish:

- `Vision`: communication with the camera and computer vision processing
- `Agent`: main decision process
- `HWComm`: communication with the hardware/low-level to send new orders for the actuators and collect hardware sensor data

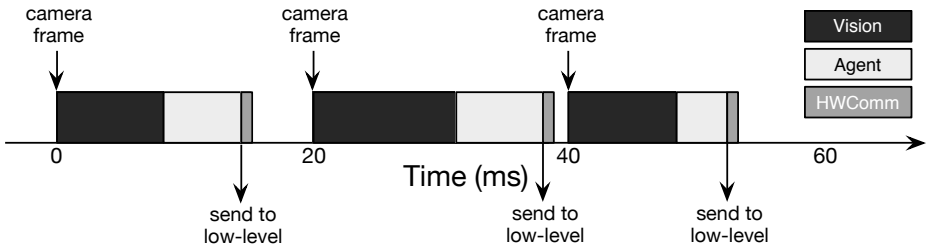


Figure 2.2: The initial order of processes

When this work started, the processes ran in the sequence represented in Figure 2.2. In this pipe-lined execution chain, the `HWComm` process execution timing is naturally conditioned by the sum of all processing times of all previous processes (`Vision` and `Agent`) - and these are not fixed. Therefore, this results in a temporal jitter on the execution of the low-level communication step, which has severe impact on the information reported by the hardware that is integrated during the period between consecutive executions (such as wheel odometry measurements).

Although this results in a relatively low impact for the orders sent to the low level (velocity set-points, for example), the case of the information gathered from sensors is more critical, because the introduced jitter results in a dynamic time span (and thus, impacting the precision) of the low-level measurement. As an example, with this kind of jitter, the information can be integrated over a longer period of time in one particular cycle than on the consecutive one. This has the potential to negatively affect the upper layers of high-level motion control.

A proposed solution to this problem is presented in Figure 2.3. The `HWComm` cycle runs each time a new frame is received from the camera, an event that occurs each 20 ± 2 ms. The values sent to the low-level are the ones calculated by the `Agent` in the previous cycle. In

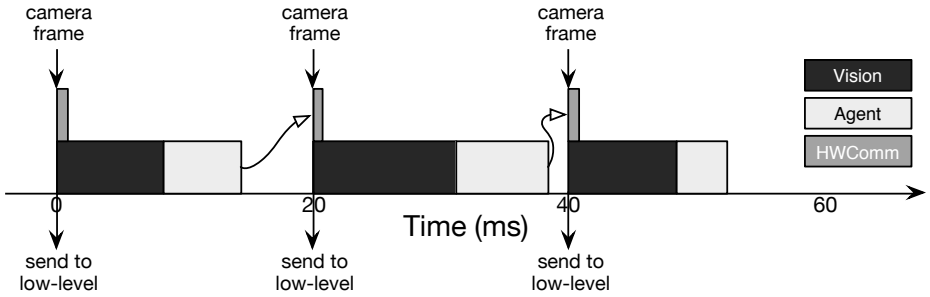
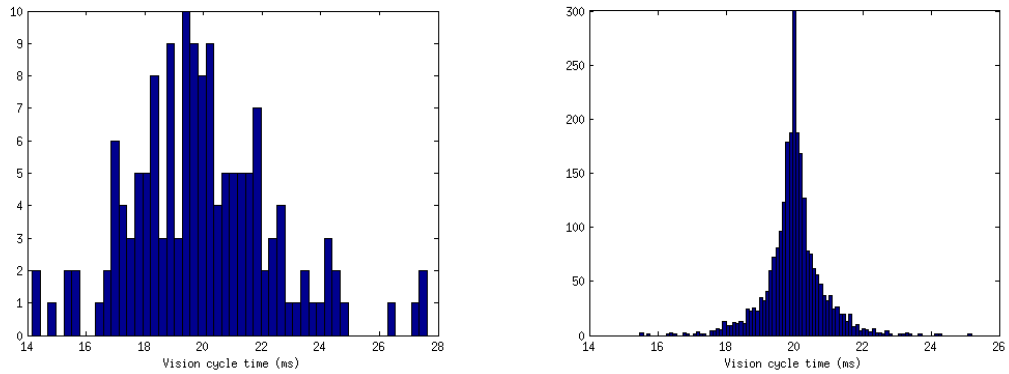


Figure 2.3: The proposed order of process execution

parallel, the `Vision` process could potentially take control of a different CPU core and start analysing and processing the new image frame.

However, one constraint is introduced when compared with the previous solution: the total time of `Vision` and `Agent` together should not exceed the cycle period in order for the `HWComm` to send new values for the low-level hardware every cycle, whereas in the pipeline approach it was possible to have temporal overlapping on the processes, i.e. a new frame could arrive while the `Agent` was still processing the previous frame.



(a) Before changing the order of processes.

(b) After changing the order of processes.

Figure 2.4: Results of the process cycle re-ordering.

Figure 2.4 shows a comparison of the vision cycle time (used as the main cycle time) before (Figure 2.4a) and after (Figure 2.4b) the proposed change. By comparing both figures, it is possible to conclude the time jitter has been significantly reduced, which resulted in a great positive impact on the robot high-level control.

2.4 Robot Position and Velocity Estimation

In CAMBADA, position and velocity estimations are achieved using Linear Kalman Filters, integrating both previous orders sent to the low-level/hardware and sensor measurements (namely odometry data from wheel encoders and information processed from the image captured by the video camera). In this work, motion data from the Inertial Measurement Unit (IMU) was also integrated to improve the estimation.

2.4.1 Localisation Algorithm

Although not part of the work developed in this project, before the position estimation integration process is described, it is important to understand how a position measurement is generated. Using computer vision techniques and a previous calibration process, the `Vision` process is able to produce a set of points that lie at the center of the field white lines (in a Cartesian System in robot coordinates).

Inferring the global robot position in field coordinates from relative line points is possible through an optimisation process originally created by a former Middle-Size League team Brainstormer Tribots, called "Perfect Match" [Lauer et al., 2006], which uses the RPROP algorithm [Riedmiller and Braun, 1993] to solve the minimisation task.

This algorithm is able to provide a measurement vector $\boldsymbol{z_{v\_loc}}$ that includes the estimated pose of the robot (position in 2D absolute coordinates x_{v_loc} and y_{v_loc}, and orientation θ_{v_loc}) based on this optimization process ran against the detected white lines.

$$\boldsymbol{z_{v\_loc}} = [x_{v\_loc}, y_{v\_loc}, \theta_{v\_loc}] \tag{2.8}$$

2.4.2 Robot Pose Estimation

In order to estimate the pose of the robot, a sensor fusion step is performed using a linear Kalman Filter (KF). The state vector $\boldsymbol{x_{pose}}$ is defined as in Equation 2.9 below:

$$\boldsymbol{x_{pose}} = [x, y, \theta] \tag{2.9}$$

In order to estimate the ($\boldsymbol{x_{pose}}$) state, the odometry sensors delta data (change since the last cycle, projected in absolute coordinates) is used as the control vector u_k for the KF predict step. These sensors provide information for x, y and θ. For the update step, the aforementioned vision localisation optimisation estimation $\boldsymbol{z_{v\_loc}}$ is included - used as a measurement vector here.

2.4.3 Robot Velocity Estimation

A similar approach is used for the robot velocity estimation, also making use of a Linear Kalman Filter, but incorporating extra measurements that are available from the hardware platform. The robot velocity state vector is defined as in Equation 2.10 below:

$$\boldsymbol{x_{vel}} = [v_x, v_y, w_\theta] \tag{2.10}$$

For the predict step, the velocities previously sent to the platform are used, taking into account the action-delay of the platform. From the specification of the CAMBADA hardware platform, there is a delay between a command sent by the agent and its actual execution of $80ms$, which, taking the cycle time of $20ms$ into account, translates into a buffer of 4 cycles. So, in practice, the predict step control vector are the target velocities calculated 4 cycles before the current cycle.

On the KF update step, several sensors on the robot provide relevant information for the velocity estimation, namely:

- Odometry deltas: contributes to v_x, v_y and w_θ;
- Vision flow: contributes to v_x, v_y and w_θ;
- Gyroscope angular velocity: contributes directly to w_θ;
- Gyroscope yaw estimation deltas: contribute to w_θ.

The KF update step is repeated for each sensor above, using a measurement noise covariance matrix tuned according to the considerations on Section 2.1.1.

2.5 Improving High Level Motion Control

The robots high-level motion control is based on Proportional–Integral–Derivative (PID) controllers - one for the linear velocity, and a different one for the angular velocity, as depicted in Figure 2.5.

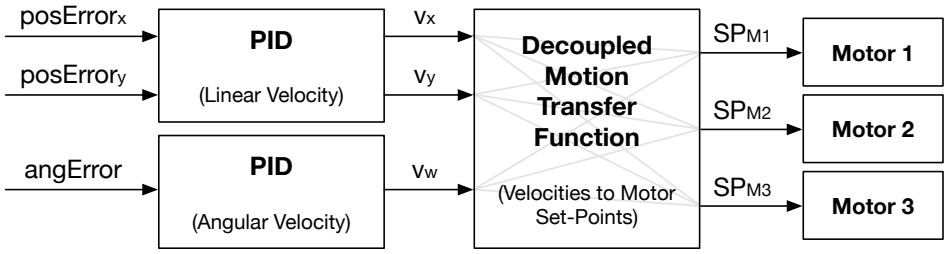


Figure 2.5: High-level motion controller.

The problem with this approach is that due to the inter-dependency of the different motor set-points with the requested linear and angular velocities and with the added acceleration constraints, sometimes the tuple $\{v_x, v_y, w_\theta\}$ calculated by the controllers is impossible to achieve, resulting in an undefined sub-optimal final robot motion.

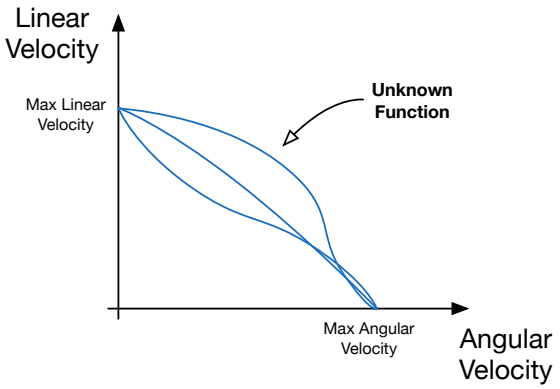


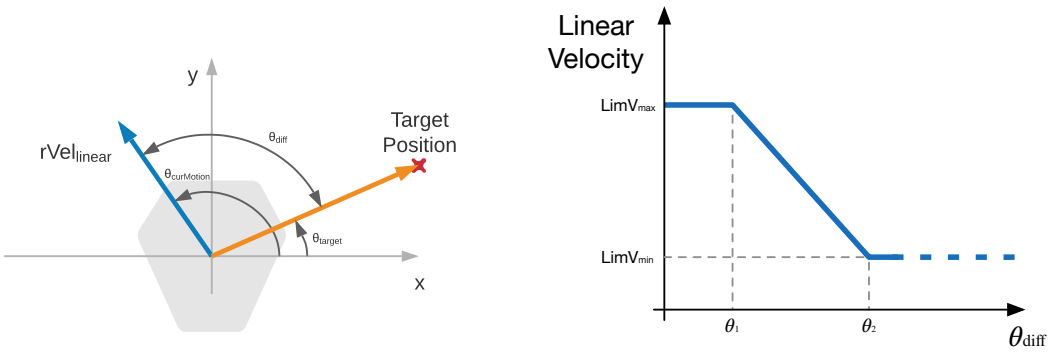
Figure 2.6: Trade-off between maximum linear velocity and maximum angular velocity.

Therefore, these limitations need to be taken into account to achieve the desired motion path. Due to the nature of the holonomic motion system, it is (empirically) known from the experience with the platform that when the robot is doing a rotation around its axis with the motors at their maximum speed, there is no margin left to add linear velocity since we reached the saturation level. A similar situation occurs when the robot is driving linearly at maximum speed - there is no extra speed left for the motors to add rotation on top of that (Figure 2.6). Modelling that function is a challenging problem - it depends on factors such as the static and dynamic friction with the field surface, which changes with the robot rigid body momentum and even with the relative direction to which the robot is moving (due to the way the wheels are disposed, there are several variables that effectively depend on the linear motion relative angle).

Although several attempts have been made to model this function, none has achieved the desired results, since the resulting linear velocity was too conservative for the application, which limits the robot to slow speeds that are unpractical for a dynamic competition environment. The main conclusion of those attempts was that it was possible to achieve predictable motion paths, provided that the saturation point is not reached. The process is explained in the following sections.

2.5.1 Proposed Solution

Although it might not obvious, depending on the desired direction and robot transient orientation, motors can quickly transition from a fully saturated state to a complete stop. These extreme situations happen frequently in the MSL matches, as the ball changes direction regularly, shifting the focus and positioning of the robots.



(a) Representation of some relevant angles.

(b) Limitation applied to the linear velocity.

Figure 2.7: Proposed solution based on the angle difference between the relative target position and the estimated robot linear velocity in robot coordinates.

As part of this project, a new solution has been proposed that takes into account the current robot motion and uses the angle difference $\theta_{\text{diff}} = abs(\theta_{\text{target}} - \theta_{\text{curMotion}})$ between the relative target position and the estimated robot linear velocity in robot coordinates (Figure 2.7a).

The maximum linear speed is then modelled as a clipped linear function of θ_{diff}. The smaller this value, the higher the allowed linear speed, and vice-versa, as depicted in Figure 2.7b. This approach forces the rotation to take precedence over linear motion, by limiting the latter if the robot is not moving towards the target.

Figure 2.8 shows the updated version of the controller with the limiter block.

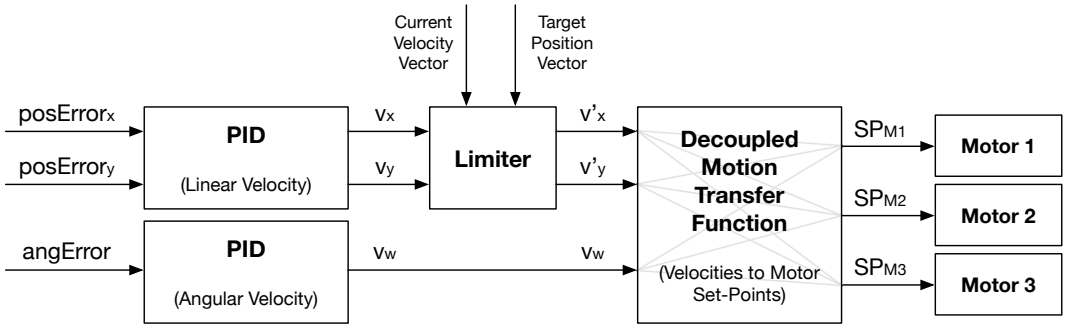


Figure 2.8: Proposed high-level motion controller.

2.5.2 Experimental Setup

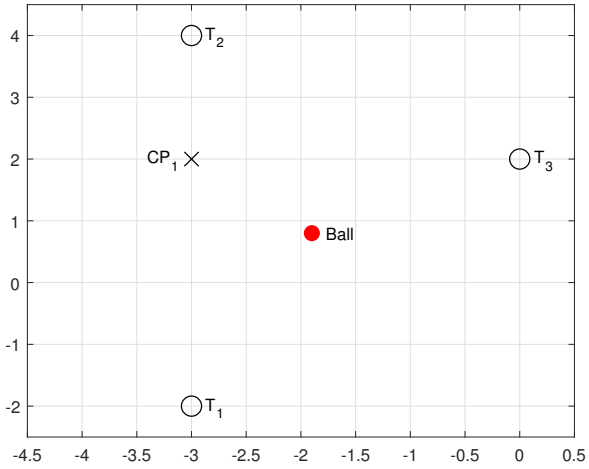


Figure 2.9: Setup used to test the proposed high-level control solution.

A setup according to Figure 2.9 has been devised to test the proposed high-level control solution. The robot will move through a set of predefined way-points, starting on point T_1, moving to T_2 and quickly changing to T_3 as soon as it crosses CP_1 - this ensures that a sudden target change occurs while the robot is close to its maximum linear velocity, which is a common situation during soccer matches.

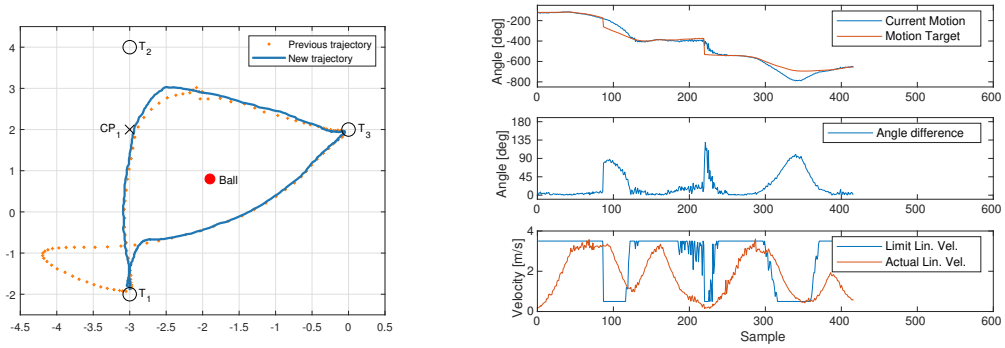
The ball is placed in the middle of the way-points and is used as the robot orientation target.

To run the solution validation tests, the following parameters have been used, as a result of some on background knowledge, but also after some manual tuning with a trial-and-error approach:

$$\begin{aligned} \mathrm{LimV}_{\mathrm{min}} &= 0.5\ m/s \\ \mathrm{LimV}_{\mathrm{max}} &= 3.5\ m/s \\ \theta_1 &= 15\ deg \\ \theta_2 &= 45\ deg \end{aligned}$$

2.5.3 Results Discussion

For the sake of visualization, this discussion will focus on a single round-trip in one of the tests (Figure 2.10a).



(a) Comparison with the previous solution.

(b) Time-series representation of angles and velocities in the test.

Figure 2.10: Results of the proposed high-level control solution based on linear velocity imposed limitation (new trajectory).

The dotted line represents the trajectory of the robot with the previous method, and the solid line the new trajectory. Note that the "overshoot" that can be seen in the bottom-left corner of the figure on the "Previous Trajectory" is not due to an improperly tuned high-level control (such as PID). In reality, it is the result of the acceleration limits imposed by the low-level control at the Motor level, beyond the control of the high level software.

On the other hand, using the same PID and the same acceleration and speed limits on the motor-level, the new solution is able to control the robot position to a much more desirable and expected path, by preventing motor saturation. Moreover, the new solution reduced the total round-trip time from $8.8s$ to $8.3s$, which is a notable time gain.

Figure 2.10b shows a time-series representation of the angles θ_{diff}, θ_{target} and $\theta_{\text{curMotion}}$, as well as the velocities (actual velocity and the output of the velocity limiter, which is based on θ_{diff}). This figure also shows the moments where the velocity limit is applied, as well as the natural delay between the application of the limit and the actual velocity drop perception.

Chapter 3

Multi-Object Tracking With Distributed Sensing

An accurate representation of the environment is very important for any robotic system that has to interact with its surroundings. Detection and tracking of multiple moving objects are particularly important if the environment is highly dynamic: the ability to correctly estimate the state of these objects around the robot can aid in the anticipation of its actions, by planning ahead and thus improving the overall robot efficiency towards the goal.

One of the main challenges concerns a trade-off between accuracy and performance, which is automatically imposed when dealing with applications defined by real-time constraints. Furthermore, when dealing with Multi-Agent Systems, the development of mechanisms to form a better world model by merging information from different agents is also a relevant research topic.

This chapter presents a real-time compliant multi-object tracking solution for multi-agent systems that operate on stochastic and highly dynamic environments, using information gathered by various agents over time. The proposed solution is detailed from the detection and feature extraction phase to the general problem of efficiently tracking objects in the environment.

In Chapter 1, section 1.1, some of the major aspects about sensor fusion in MAS were already introduced and in section 1.4, the motivation for a distributed approach was also described.

When addressing distributed sensing in MAS, much of the literature focuses on area coverage optimisation through active sensing using gradient based methods [Kantaros et al., 2015] or Voronoi cells [Abbasi et al., 2017]. In contrast, in the method we propose, the main focus was on passive sensing of multiple moving objects that may be seen by more than one sensing agent, which poses a data association problem [Kamal et al., 2016]. However, some well-known distributed estimation approaches are usually designed based on an implicit assumption of unlimited computation resources, non-delayed sensing, unlimited bandwidth and/or perfect

communication environments. To cope with the communication problems, some solutions resort to a moving-horizon estimation [Ahmad and Bülthoff, 2016], but still require system clock synchronisation which would potentially fail under unreliable communication channels.

The solution presented in this chapter was specially designed for applications with real-time constraints and sub-optimal communication conditions - being able to cope with constrained resources is fundamental to actually implement a solution in a real robotics environment.

The major contributions of this approach are as follows:

- Cost-effective (in terms of performance vs. resources used) solution for detection of objects of interest using visual information;
- Novel solution for real-time multiple object tracking, which takes background knowledge into account;
- Integration of multiple local tracking to generate a unified multi-object tracking mechanism using agents as sensorial network;
- Implementation, testing and benchmarking of this solution in a real application environment, which is highly stochastic and dynamic, showing the compliance of this solution with strict real-time constraints.

Taking a robotic soccer competition as an example application and benchmarking scenario, the process taken will be detailed with the sensorial hardware used, how feature detection and local tracking were achieved, the inter-robot communication framework and finally how a distributed sensing approach enabled any agent (even agents without sensors) to get an overview of moving opponent robots (or obstacles for navigation) spread around the field.

An important note is that although robotic soccer is presented as an example application domain, the main ideas expressed in this chapter are not limited to this application area.

3.1 Perception

In Section 1.3.2, the hardware of the CAMBADA vision system was described, upon which the perception process relies. It starts with the detection of the objects of interest that need to be tracked. One important issue is that the quality of information provided by the perception system directly impacts the tracking performance.

In our target application, one of the main problems is that those robots can drive at top-speeds that can go as high as 4 m/s. This creates a mechanical shaking effect in the vision system, which induces a lot of noise in the observations performed by the robot, and therefore results in several false-positive observations as well.

However, as in every other application domain, there are specific conditions that can be taken into account to enhance the perception of the environment. For this particular application, the objects of interest are the **ball** and the **obstacles** (which can either be teammate robots or opponent robots). This section describes the vision processing algorithms that are used to detect these two different types of objects. Each frame captured by the camera is processed independently, generating a list of observations. The *integrator* (Section 3.2) is then responsible to correlate these observations with previous ones.

3.1.1 Ball Detection

At the time the work was conducted, in the MSL, the ball has a preponderant well-defined color, so robots can take advantage of color segmentation to detect the ball. However, supporting ball detection solely on color segmentation and heuristics for blob validation (based on the blob size on the 2D image) has obvious shortcomings: low resilience to changes in ambient lighting and easy failure of heuristics when the ball is partially occluded - the blob size is reduced, producing an offset in the center of the ball with respect to the center of the blob.

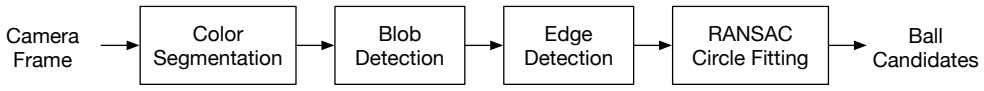


Figure 3.1: Ball Detection Fluxogram.

Therefore, two extra steps were introduced to further improve the perception of the ball: **Edge Detection** and **RANSAC Circle Fitting**

Edge Detection

To detect the edges of the ball in the 2D image, a set of radial scan-lines is dynamically created in the center of each detected blob. Each scan-line is analysed individually, starting on the center of the blob, and moving outwards, while a set of heuristics is used to select the edge point: it starts by finding the ball color on that line and then the edge point is defined as the transition of the ball color to green, white or black.

RANSAC Circle fitting

The points (pixels on the image) extracted in the previous step are then used by a RANSAC algorithm to fit a circle, in a similar fashion as in [Mironov, 2017]. At each iteration i, the RANSAC algorithm works by fitting a circle c_i that is created by randomly sampling 3 edge points.

Afterwards, the sum of all Euclidian distances (function D) between the remaining edge points p_k and the circle candidate c_i is used as the score (s_i), as in equation 3.1.

$$s_i = \frac{1}{\sum_{k=1}^{N} D(p_k, c_i)} \tag{3.1}$$

Finally, the highest scoring circle candidate is picked as the circle that best fits the ball position in the image.

(a) Raw image frame. (b) Detected Edges. (c) RANSAC Fitting.

Figure 3.2: Partial ball occlusion example.

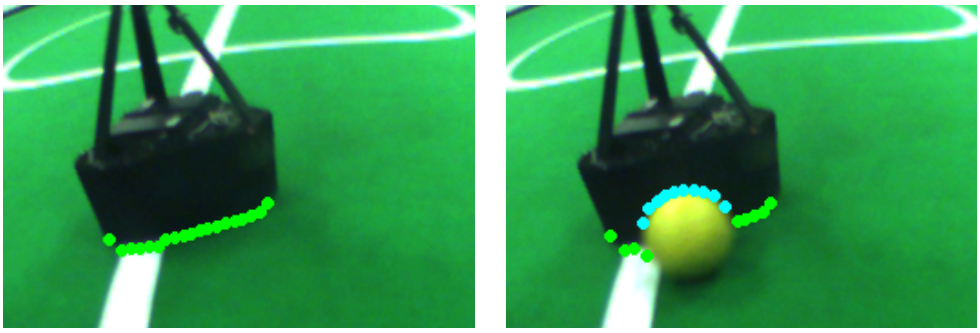
Figure 3.2 shows a situation where a partially occluded ball is detected with a promising accuracy using this method. The circle fitting using RANSAC performs well when there are ball edge outliers, like in the example above.

The center of the circle is then assumed to be the center of the ball in the 2D image. An offline-created look-up table is next used to efficiently convert distances from pixels to meters, based on the configuration of the catadioptric system and the characteristics of the mirror [Cunha et al., 2008]. This ball detection process generates a list of ball observation candidates, that is sent to the integrator process (Section 3.2).

3.1.2 Obstacles

Reliable and fast detection of obstacles is also a critical issue to ensure safety and integrity of robots and humans. By correctly identifying obstacles around it, each robot is able to plan a trajectory and move around the field while avoiding any physical contact.

To cope with the highly dynamic scenario provided by soccer, robots need to perceive the other robots around them, both opponents and team-mates. This will allow them to move around the field, either for re-positioning or dribbling the ball, while avoiding contact with any other obstacle. Furthermore, from a tactical and strategic point of view, knowing the opponents position can provide an additional advantage in terms of planning.



(a) Detected obstacle edges. (b) Partial occlusion by the ball.

Figure 3.3: Obstacle detection with partial ball occlusion.

Obstacle detection is based on radial scan-lines, starting in the center of the robot (near the center of the image) to the outer part of the mirror. Since the bottom part of the robots must be dark, color segmentation is used to detect transitions such as "green-black" or "white-black" - Figure 3.3a. These detected points are assumed to be on the ground, and are then transformed from pixels to the local coordinate system of the robot (in meters), using the same look-up table discussed in the previous section.

However, when the ball passes in front of an obstacle, it occludes the lower part of the obstacle and the detected edge obstacle points are above the ground (Figure 3.3b), which violates the previous assumption. The (undesired) result is that after clustering all points, the obstacle position is erratically projected further away from the observer robot, inducing an error in the position estimation. This can be minimised if this particular situation is taken into consideration. In order to deal with this special case, the points that are detected above a ball are marked by the detection algorithm and the relative distance of the obstacle is adjusted according to the median of the distances of the other points (not above the ball).

At the detection stage, no differentiation is made between opponent robots and team-mates, since all of them must comply with the previously described rule. The decision as to whether an obstacle is a team-mate or not, is left to the Integrator module (Section 3.2). The detection result is a set of points belonging to the periphery of the obstacle.

To aggregate points that belong to the same obstacle, a clustering algorithm was implemented, based on the *COP-Kmeans* method [Wagstaff et al., 2001], to take advantage of the background knowledge, that can be expressed as a set of instance-level constraints on the clustering process. Namely, in this clustering algorithm implementation, instances that are too close to each other (in this implementation, closer than 0.4 m) are merged into a single instance and instances that are too big (of which farthest points are distanced more than 0.7 m in this case) are split into two instances. These values were selected sensibly considering the size of the robots, which is around 0.5 m wide.

3.2 The Integrator

Values provided by any sensor are inevitably noisy and usually not very usable in their raw form. This is why the integration of the information available to the robot is a crucial step.

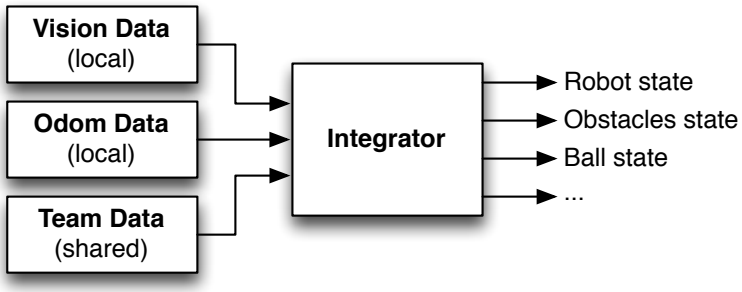


Figure 3.4: Illustration of the agent integrator module.

In the case of the CAMBADA software architecture, this is the function of the Integrator module, which takes values from different sources (camera, wheel encoders, team-mates shared info, etc.) and properly handles them so that a relevant and trustworthy database is built to represent the current World Model (Figure 3.4).

The ball is one of the most important objects on the field, since the objective in the considered scenario is to play soccer. Just like in the case of obstacles, it is also common to have spurious ball detections for various reasons, including occlusions by other robots, lighting conditions, high speeds (causing motion blur on the image), self occlusion while dribbling, etc.

An accurate estimation of the ball state (position and velocity) is crucial to be able to intercept a pass, to catch the ball, etc. Additionally, if a correct representation of the opponent obstacles can be achieved, then the team can evolve from simple reactive obstacle avoidance to more complex strategies such as opponent modelling and coercive behaviour [Biswas et al., 2014].

For the case of state estimation of the ball and obstacles around the robot, the new Multi-Object Tracking module (Section 3.3) was used. One of the main problems is the fact that robots move very fast (up to 4 meters per second), which introduces a lot of noise in its measurements, thus inducing a lot of false detections. Therefore, a set of heuristics are applied to validate these detections.

Taking into account that the current robots in this league have omni-directional drive, it is not useful to estimate the obstacles orientation. On the other hand, in some situations, knowing the opponent positioning may allow the team to act quicker, anticipate their actions and even prevent dangerous situations like forward passes by covering an opponent from the ball perspective. Therefore, it is crucial that this information is as accurate as possible.

3.3 Local Multi-Object Tracking

In this Section, the novel Object Tracking software module is introduced and its applicability in both obstacle tracking and ball tracking is also discussed.

3.3.1 Tracklet Definition

Tracklets are essentially paths along which a particular object is perceived to have travelled and constitute the core of most object tracking techniques.

In this work, the tracklets are assumed to be consistent with a Gaussian process model in the 2D space, therefore each one is associated with a Kalman Filter (KF) with 4 state variables: x, y, $\dot{x}$, $\dot{y}$ - position and velocity in x and y axis. As previously mentioned, the orientation was not considered, because in this league omni-directional motion capabilities are widely used, although it may be included in the future if needed.

Furthermore, some extra properties were included to allow the tracker to monitor each tracklet consistency and to purge invalid tracklets:

- `age` - integer counter that starts at zero and is incremented at each agent cycle;
- `visibleCount` - total number of cycles this tracklet has been visible (an observation was matched) since it was created;
- `invisibleCount` - how many consecutive cycles this tracklet has not been matched with an observation.

3.3.2 General Algorithm

For each new agent cycle, the object tracker algorithm is updated with the observations on that cycle. This algorithm can be summarized in 7 steps, as shown in Algortihm 1.

Algorithm 1: Object tracker update algorithm

1. Predict new tracklet positions;
2. Assign observations to tracklets;
3. Update assigned tracklets;
4. Update unassigned tracklets;
5. Purge old tracklets;
6. Create new tracklets;
7. Sorting;

Step 1 - Predict new tracklet positions

This step consists in running the predict step in each tracklet's KF. It will not only update the covariance matrices, but also the state estimate, predicting the position of each object one cycle ahead, using the estimated velocity and assuming uniform motion.

Step 2 - Assign observations to tracklets

This step represents a data association problem, since some of the observations of the present cycle belong to objects which are already being tracked, others are newly detected objects that will originate the creation of new tracklets and finally some tracklets might not match with any observation.

To solve this pairwise association problem, the Hungarian method [Kuhn and Yaw, 1955] was used, that finds an optimal assignment for a given cost matrix C' in polynomial time, if we assume that only one observation exists per object.

The cost matrix that was used is a padded square matrix - it has additional columns/rows to account for the possibility of not matching any of the observations with the current tracklets. If there are T tracklets and N observations, the dimension of the cost matrix C' will be $(T+N) \times (N+T)$. The rows of this matrix represent tracklets and the columns represent observations, hence the C matrix has the dimension $(T \times N)$. The δ factor represents a bias factor in the same order of magnitude as the cost matrix C. For example, if the cost matrix C is calculated with euclidian distances, in meters, between trackets and observations, δ should be sensibly assigned a value for a distance that we want to consider a non-match (lower values will create more new tracklets, while higher values will tend to match existing tracklets as much as possible).

$$C' = \left(\begin{array}{cccc:cccc} & & & & \delta & \infty & \cdots & \infty \\ & & & & \infty & \delta & & \\ & C & & & \vdots & & \ddots & \\ & & & & \infty & & & \delta \\ \hdashline \delta & \infty & \cdots & \infty & & & & \\ \infty & \delta & & & & & 0 & \\ \vdots & & \ddots & & & & & \\ \infty & & & \delta & & & & \end{array}\right)$$

In the C' matrix above, the horizontal dashed line appears after the T tracklets, and the vertical dashed line appears after the N observations.

After running the Hungarian algorithm over the cost matrix C', there will be three main groups in the posteriori logical matrix C'': matched tracklets (M), unmatched tracklets (U) and unmatched detections (V). The lower-right part of the matrix C'' is ignored.

$$C'' = \left(\begin{array}{c:c} \text{M} & \text{U} \\ \hdashline \text{V} & 0 \end{array}\right)$$

Step 3 - Update assigned tracklets

For each match (sub-matrix M), the tracklet's KF is updated with the respective observation, `age` and `visibleCount` counters are incremented and `invisibleCount` is reset.

Step 4 - Update unassigned tracklets and purge old tracklets

For the unassigned tracklets (sub-matrix U) The `age` and `invisibleCount` counters are incremented and a factor $f < 1.0$ is applied to the velocities, to avoid that tracklets which will remain unassigned for some cycles (and eventually removed later) remain with the same speed during that period. These factors are adjusted based on the dynamics of the opponent team and our team frame-rate. Example values for this particular application are presented along with the results in section 3.5.

Step 5 - Purge old tracklets

A validation is performed to check if the tracklet should remain or be removed. Here, two heuristics were applied: a tracklet is deleted if either the $inv_{max} =$ `invisibleCount` counter is higher than a threshold or the visibility ratio ($v =$ `visibleCount`/`age`) goes below a certain threshold. The first heuristic purges tracklets that are not observed for a period of time. The second one allows spurious detections to be deleted on the next cycle (even if the invisible threshold was still not reached).

Step 6 - Create new tracklets

A new tracklet is created for each unassigned detection (sub-matrix V), which represent observations not matched with any prior tracklet.

Step 7 - Sorting

The sorting criteria depends on the object to be tracked and objective of the tracker in each case.

In the case of obstacles, the sorting is done by descending visibility - meaning that tracklets with higher visibility ratio will be first on the list. This criteria is important for the decision of which tracklets should be shared with the team-mates, since there is a limit on the number of tracklets that can be shared, imposed by MSL rules as a bandwidth usage limit.

Ball tracking differs from obstacle tracking in the fact that there should be just one ball inside the field, but the agent should be able to handle situations where it perceives more than one ball candidate. Therefore, the agent keeps track of multiple balls, using this method, but sorting is done by age and the oldest tracklet is selected.

3.3.3 Tracklet Sharing Criteria within a MAS

Agents should integrate the maximum available observations (local and shared by their team-mates), so that their World Model is as rich as possible. An agent shared information comes as a result of its local observations. However, these are never free of noise and can possibly include false positives. Therefore, knowing that a team-mate agent is likely to incorporate one's shared information, the sharing criteria of each agent should aim at maximizing the information other agents can extract from what is being shared. Ideally, these should not include false-positives and should be as accurate as possible.

Therefore, a set of rules were defined to decide whether a particular tracklet should be shared with the team-mates:

- The maximum number of shared tracklets must be pre-defined sensibly and according to the target application and its specificities;
- Obstacle tracklets with higher visibility have a higher priority to be shared;
- Tracklets must have a pre-defined minimum `age`;
- Tracklets must have a pre-defined minimum `visibleCount` and visibility ration;
- Tracklets distance to the observing agent must not exceed a pre-defined threshold (to prevent false positive detections on large distances);
- Shared tracklets can only be un-shared if they are explicitly deleted or if the position lies outside a pre-defined region-of-interest.

3.4 Distributed Sensing

In this Section, the methodology used to merge information from multiple agents to generate a unified representation of the obstacles spread around the field is presented.

In the context of the Middle-Size League, although also useful for the robots, this is particularly important for the coach, which is a computer allowed to communicate with the robots, but not allowed to have any attached sensors. The objective of the coach is to give high-level coordination instructions for the team - strategy, formation, attitude, etc. Using the information shared by the robots, the coach is able to create a representation of the opponents positions, which can be used to anticipate opponent gameplay, such as forward passes and set-plays.

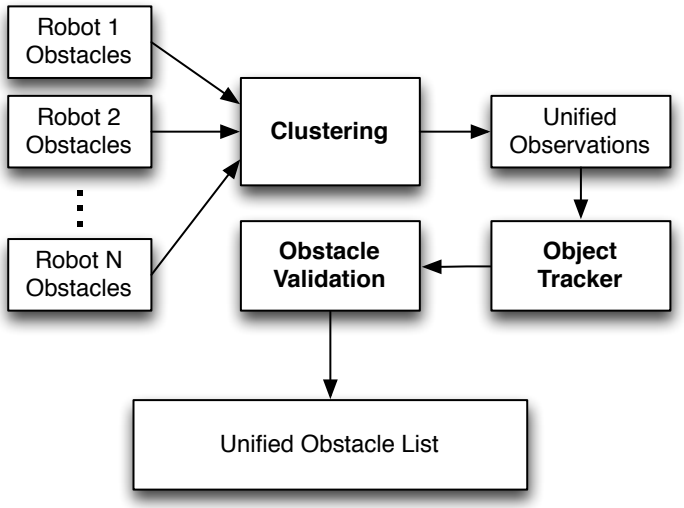


Figure 3.5: Summary of the tracking system using multiple agent shared observations, adapted from [Dias et al., 2016].

Figure 3.5 shows an overview of the global tracker. It considers the various agents shared obstacle tracks as observations. Although this shared information is not made of raw observations, but rather processed observations that have been associated together as a track and met the previously discussed criteria to be shared among the team, they can be considered observations for the purpose of creating this unified obstacle list.

3.4.1 Observation Clustering

In the last Section, we already demonstrated how the Hungarian Algorithm can be used to match observations with tracks. However, by solving a global minimisation problem, it can not account for situations where there are multiple observations for the same object. Therefore, a clustering algorithm was implemented, based on the *COP-Kmeans* method [Wagstaff et al., 2001], to take advantage of the background knowledge, that can be expressed as a set of instance-level constraints on the clustering process.

In the case of the MSL, the maximum size of the obstacles is limited by the rules (50×50cm), which implies that an obstacle that occupies the maximum allowed size can be perceived as a 70.7cm-wide obstacle (when seen from the diagonal). Despite this theoretical value, on this league, at the time of writing this thesis, most teams opt for a triangular configuration on their platforms, meaning that such a size is not reached. Moreover, their sides do not measure less than 30cm.

Using this background knowledge, the *COP-Kmeans* method has been applied with the following constraints:

- if $width(centroid_i) > 0.7$m, split in two centroids;
- if $distance(centroid_i, centroid_j) < 0.3$m, merge $centroid_i$ with $centroid_j$.

3.4.2 Applying the Object Tracker

The output of the previous clustering stage is a unified observation list, which is the input for the distributed object tracker module. Its implementation was already described in Section 3.3.2.

3.4.3 Obstacle Validation

To avoid sharing unreliable information with the team-mates, the unified tracklets are subject to validation and once a tracklet has been validated, it will remain valid until it disappears.

Essentially, the relative position from track to the team robots defines three different zones:

- **Zone 1** - any position closer than d_1 from a team-mate. In this zone, only tracks observed by the closest team-mate robot can be validated. It is such a small distance that the closest robot must be able to see it directly. This prevents using detections of spurious obstacles close to other team-mate positions;
- **Zone 2** - any position closer than d_2 and further than d_1 from a team-mate. Any track lying on one of these zones is validated;
- **Zone 3** - any position closer than d_3 and further than d_2 from a team-mate. A track lying on this zone requires at least two observing robots to be validated;
- Any tracked obstacle for which the distance from the closest team-mate is more than d_3 is not validated, since it is outside the maximum detection distance boundary.

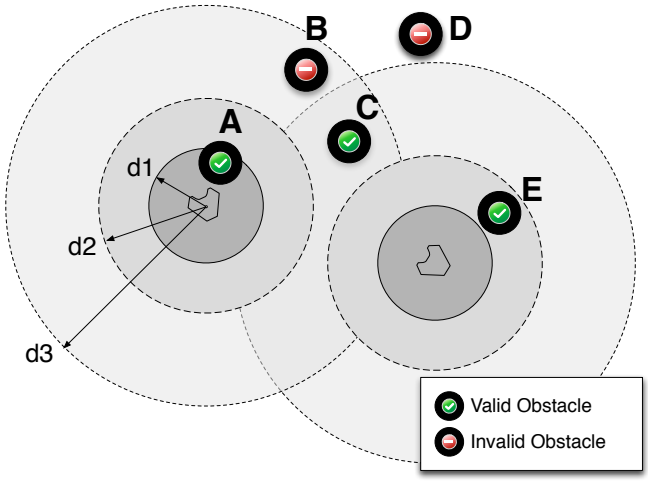


Figure 3.6: Example of the validation criteria using two robots, adapted from [Dias et al., 2016]. Here, three observations are validated and two do not pass the validation criteria. The image is not in scale.

Figure 3.6 shows an illustration of the implemented validation methodology.
In this example, there are 5 obstacles:

- **Obstacle A - Valid** - its position lies inside **Zone 1** of the robot on the left;
- **Obstacle B - Not Valid** - the obstacle is inside **Zone 3** of one robot, but not the other robot;
- **Obstacle C - Valid** - it is inside **Zone 3** of both robots;
- **Obstacle D - Not Valid** - It is outside all zones of all robots;
- **Obstacle E - Valid** - it is inside **Zone 2** of a team-mate.

These criteria are used to validate new obstacle observations into the unified obstacles list. Once a track has been validated it will remain valid in that list until it is cleared: for example, a track may be validated in zone 3 for being seen by two different robots and then leave this zone (even going further than d_3), it will remain in the obstacles list during all this period.

3.5 Results and Discussion

To benchmark the performance of this solution, we used four different metrics:

- **Precision**: ratio between the number of correctly identified obstacles and the number of detections in each frame;
- **Recall**: ratio between the number of correctly identified obstacles and the number of obstacles in groundtruth in each frame;
- **False-Positive Rate** (FPR): ratio between the number of outliers and the number of obstacles in groundtruth in each frame;
- **Position Accuracy Gain** (PAG): ratio gain between the individual observations position error and the merged obstacles position error.

To test the Multi-Object Tracking techinque described in section 3.3, the experiments ran with the following set of parameters:

- Zone 1 distance $d_1 = 1.0m$
- Zone 2 distance $d_2 = 2.5m$
- Zone 3 distance $d_3 = 5.0m$
- Unassigned track velocity factor $f = 0.8$
- Bias for cost matrix C', $\delta = 30.0m$
- Max. number of invisible frames $inv_{max} = 20frames$
- Visibility ratio $v = 0.2$

In the following sections, the results will be presented for both lab and competition environments.

3.5.1 Lab Experiment

To benchmark this solution in the lab, a setup has been devised according to Figure 3.7: in the middle (in black), four obstacles have been placed on the field - three of which are static throughout the experiment, while a fourth is constantly moving in a square shape.

Figure 3.7: Lab setup used to benchmark the solution.

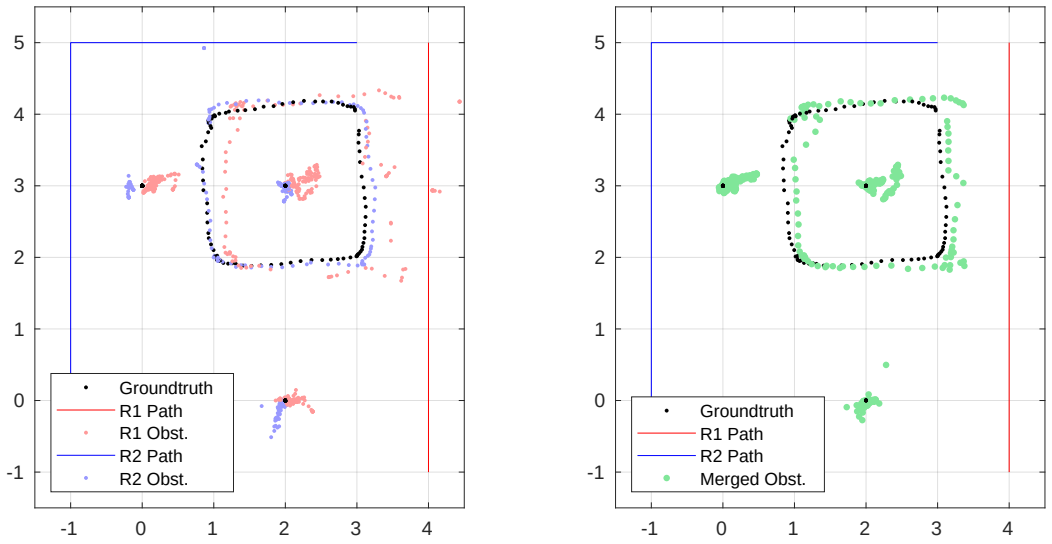
Two teammate robots (in blue) are asynchronously following their respective paths around this set of obstacles, while observing and tracking them in real-time (one following an L-shape path and the other a linear path).

A total of 3 runs were conducted in the lab and in order to test some hypothesis, the linear velocity of the robots was increased in the second test and a constant angular velocity was introduced in the third test, as in Table 3.1.

	Linear Vel. [m/s]	**Angular Vel.** [rad/s]
Lab Run 1	1.5	0.0
Lab Run 2	2.5	0.0
Lab Run 3	1.5	2.0

Table 3.1: Lab tests conditions for each run.

With the lack of a better groundtruth system in place, the result of the localization process of the moving robot obstacle (in black) was used as the groundtruth. The static obstacles were manually placed in the pre-defined depicted positions.



(a) The obstacles observations by the two robots. (b) Merged obstacles in the corresponding samples.

Figure 3.8: Representation of a subset of frames in the third lab experiment.

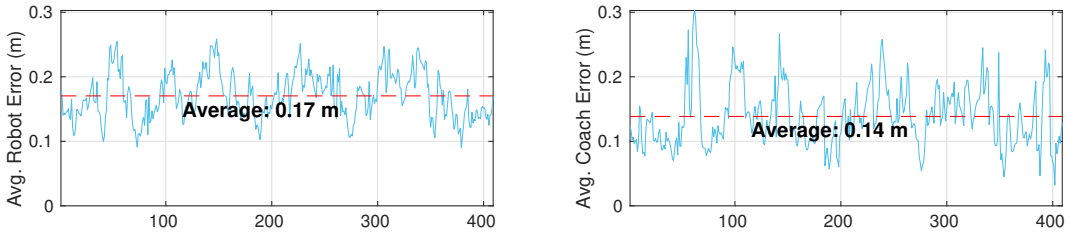


Figure 3.9: Comparison of position error on the first lab experiment.

	Linear Vel. [m/s]	**Angular Vel.** [rad/s]	**Avg. Precision** [%]	**Avg. Recall** [%]	**Avg. FPR** [%]	**Avg. Accuracy Gain** [%]
Lab Run 1	1.5	0.0	98.11	98.11	1.89	17.65
Lab Run 2	2.5	0.0	95.72	96.84	4.76	17.65
Lab Run 3	1.5	2.0	97.37	98.03	3.05	6.25

Table 3.2: Results of the tests conducted on the lab.

Figure 3.8 shows a small subset of frames from the third lab experiment. Figure 3.8a shows the obstacle position as reported by each robot individually after running the Obstacle Tracking algorithm locally. This is the information that is then combined using the method described in Section 3.4.1 that results in the merged obstacles depicted in Figure 3.8b.

Figure 3.9 shows a comparison between the position error (in meters) of individual observations (on the left) and the merged observations (on the right) in the first lab experiment. The plot shows the absolute value for each cycle as well as the average for the whole experiment run.

As can be seen on Figure 3.9 and Table 3.2, not only was the average error reduced considerably, but very interesting results were also achieved for the Precision, Recall and FPR.

3.5.2 Competition Environment Experiment

In the Middle-Size League, since 2016, teams are encouraged to supply, their *world model* information during the matches (only for logging purposes, this information cannot be used by the opponent team during the match). The objective is to provide the other team a reference to benchmark solutions after the match ends. In this case, knowing the reported location of the opponents after the match, allows us to use it as a "groundtruth" to match with our obstacle detection and tracking algorithm. Here we are assuming the opponent is able to have a better estimate of its position than the one we are able to detect, which is generally the case for the top-teams in this league. Unfortunately, due to restrictions imposed by the MSL rules, there is no groundtruth available for the ball position in the competition dataset.

The proposed solution was tested during the first half (15 minutes) of the final match of CAMBADA in the RoboCup 2016 World Championship (3rd/4th place match). The obstacle detection and tracking was analysed offline against the "groundtruth" (provided by the other team). For this purpose, only free-play situations considered (because in other situations, humans may be inside the field repositioning the ball, and therefore inducing extra obstacles in the perception of the robots). Under these conditions, a dataset with **3245 frames** sampled at **10 Hz** was considered. This dataset includes data that allows the calculation of average precision, recall and false-positive rate.

The results of all tests are presented in Table 3.3 and prove that most spurious detections occur when the robots are moving faster (in competition, robots can reach velocities of 4 meters per second) and the distance between the obstacles and the perceiving robot increases - this explains the drop in precision and recall and increase of FPR in competition. One idea for a future improvement is to model the threshold distances as a function of the robot velocity.

	Linear Vel. [m/s]	**Angular Vel.** [rad/s]	**Avg. Precision** [%]	**Avg. Recall** [%]	**Avg. FPR** [%]	**Avg. Accuracy Gain** [%]
Lab Run 1	1.5	0.0	98.11	98.11	1.89	17.65
Lab Run 2	2.5	0.0	95.72	96.84	4.76	17.65
Lab Run 3	1.5	2.0	97.37	98.03	3.05	6.25
Competition	N/A	N/A	92.39	86.82	7.61	N/A

Table 3.3: Results of the tests conducted on the competition (lab results conveniently included to help comparison).

Moreover, by visual inspection we also noticed that sometimes the false-positive detections occur near our robots in the competition experiment. Taking our validation criteria into account, this means that the robots sometimes wrongly identify obstacles in their vicinity. These situations can occur, for example, after the robot vision system is hit by a ball and needs re-calibration or in cluttering situations, where strong shadows appear on the field, because our obstacle detection relies mostly on color - the robots design is required to be mostly black, which can easily be confused with a dark shadow on the green field.

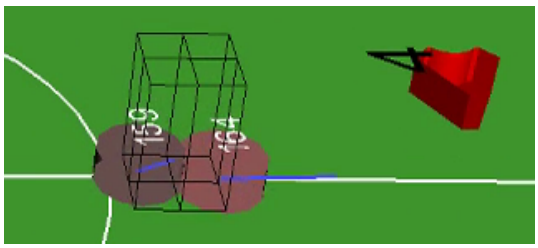


Figure 3.10: Merged obstacle (black wireframe) from two individual observations (cylinders)

A Position Accuracy Gain (PAG) is achieved by clustering multiple observations of the same obstacle, as an example on Figure 3.10. By taking two averages - the agents observations error and the merged obstacles error with respect to the groundtruth - it is possible to calculate a ratio and the results show that the merged position was always better in all the tests.

As an example, Figure 3.9 shows a comparison between average position error of the individual obstacle perception of the robots (on the left) and the error of merged obstacles (on the right), during the second lab experiment.

A thorough study on the false-positive occurrences per frame was performed in the competition dataset (Figure 3.11): a maximum of 7 false-positive obstacles were detected, but in the majority of the frames the number does not go above 2, which is acceptable, given the environment characteristics.

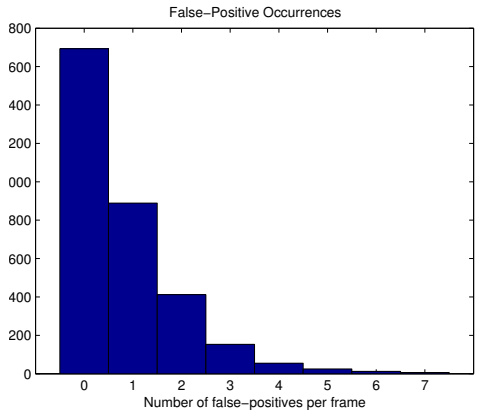


Figure 3.11: Closer study on false-positive occurrences on the competition test

3.6 Summary

In this section, a new solution for real-time multi-object tracking with distributed sensing on a stochastic and highly dynamic environment was presented. The results on the testbed show the efficiency of the solution and its compliance with the real-time constraints of the robotic soccer context under benchmark. This solution was implemented on a RoboCup Middle-Size League agent integrator and also tested during the RoboCup 2016 competitions, with the results shown for the last match in that year's World Championship competition in Leipzig, Germany and it is still the method currently in use to track obstacles by CAMBADA.

Unfortunately, no quantitative direct comparison can be made with other techniques proposed in the literature, as they were not implemented in our experimental setup. Although some approaches [Ahmad and Bülthoff, 2016] present solutions for the Middle-Size League, they do not show any results on actual real-world scenarios or do not perform under the $20ms$ real-time constraint with on-board processing, while others present results under lab conditions or based on offline processing [Ahmad and Lima, 2013].

Chapter 4

Coordination in Robotic Soccer

In Multi-Agent Systems, several interacting intelligent agents pursue a common goal by performing a set of tasks and one of the most important aspects is the ability for the agents to select and initiate behaviours in a given context in order to help the team achieve a common goal. To do so, they need to share information and coordinate to maximize the group gain. Among different topics that orbit around multi-agent coordination [Reis, 2003], a major one is task assignment. In particular, when dealing with homogeneous teams, where agents with similar characteristics can assume any available role, fixing the role per agent may not be the wisest decision, since it would limit the ability to dynamically change roles between team members, therefore decreasing the overall performance. On the other hand, dynamically assigning roles in a team can have some associated costs, namely processing time and computing power, as well as the potential for conflicting assignments (for example, exclusive roles with double assignments, essential roles without assignment, etc.), which are usually a consequence of dynamic and distributed assignment and the fact that each agent can have a slightly different world model from the rest. Although the team attitude could potentially be seen as the mere conjunction of all individual behaviours, more advanced coordination techniques can improve the overall team performance as will be seen in this Chapter.

4.1 Agents Communication

Team coordination becomes impossible without some sort of **communication** - either implicit (little or no communication) or explicit.

In a stochastic, dynamic and partially observable environment such as the one in the RoboCup Middle-Size League, team communication plays a major role maintaining an accurate World Model representation across all team members, supported by the field players beliefs, since, as previously stated, all sensors must be installed on-board of the robots. There are, for example, several situations where the ball can be occluded from one robot at a given

time, but the ball position and velocity information can be transmitted from another teammate that is able to see it. This constitutes one of the simplest, yet effective, forms of communication among a robotic soccer team.

The 2019 Middle-Size League rules enforce that teams communicate through an IEEE 802.11a/b/n network link via unicast or multicast - an official Access Point is provided for each field during competitions. Communication is allowed between the player robots and an additional computer (often called basestation or coach computer), but there are bandwidth and transmission power restrictions to ensure a fair use of the network to promote favourable conditions during matches - as of today, $2.2 Mbit/sec$ for each team and $-40 dBm$ at $9 meter$ distance.

These restrictions also exist to serve as an incentive to implement fully distributed solutions and prevent teams from using the basestation computer as a centralised "master", directly controlling the "slave" robots. Furthermore, the basestation computer is not allowed to have any sensors, so all information regarding the world model must be collected from team players. If a team wants to use a coach in the MSL in their basestation computer, in order to make a decision, it needs information that can only be communicated by the players on the field.

Additionally, as mentioned in the previous Chapter, since 2016, MSL teams are encouraged to supply their worldstate information during the matches for offline-analysis purposes. This is particularly useful to cross-compare different solutions for, for example, obstacle tracking among different teams, as localization data from one team can serve as groundtruth for analysing the other team's obstacle tracking performance.

In the case of the RoboCup-Soccer 2D Simulation league, there are strict bandwidth limitations [Almeida et al., 2010], which lead teams to develop solutions for smarter information sharing, such as the Advanced Communications framework [Reis, 2003], in which each player maintains a separate world model from its perceived one. By comparing both states, the agent selects the most useful items to share, based on an assessment of the estimated interest by its teammates.

Motivated by the same restrictions, other solutions that require little or no communication were proposed based on assigned roles [Kok et al., 2005], player's beliefs [Isik et al., 2007] and prediction models trained offline [Stulp et al., 2006]. Simulation challenges also demonstrated how the combination of implicit coordination with beliefs exchange can perform better than explicit communication [Isik et al., 2007].

4.2 Strategic Positioning

As teams began solving basic tasks such as localization, locomotion and ball handling, the need for a strategy quickly arose and this explains why strategic positioning is a relevant topic within the RoboCup community. Generally, the target position of an agent is called its

strategic position and in order to play soccer in an effective way, agents have to coordinate their positioning on the field and their actions. A good strategic positioning can make a difference, both in defensive situations by blocking passes from the opponent, and in attacking situations to create opportunities to pass the ball and progress on the field towards the opponent goal.

4.2.1 Base Player Formation

One of the first efforts to achieve coordination in multi-agent soccer was Strategic Positioning with Attraction and Repulsion (SPAR) [Stone, 2000]. This method takes into account the positions of other agents and the ball. Then, some attraction and repulsion forces are evaluated regarding opponents, team-mate robots, the ball and the goal to decide the target position for each agent.

One of the most popular strategic method for positioning in Robocup Soccer Simulation is the Situation-Based Strategic Positioning (SBSP) [Reis et al., 2001, Lau et al., 2008], which can be used to calculate target actions for agents on the field. The positioning of an agent takes into consideration the current team formation, tactic, game situation (attacking, defending, etc.) and ball position.

A method for dynamic positioning based on Voronoi Cells (DPVC) [Dashti et al., 2006] was also proposed, which places robotics agents based on attraction vectors. These vectors represent attraction forces towards objects and take into account each agent role and the current game state.

Delaunay Triangulation (DT) formations [Akiyama and Noda, 2008] split the soccer pitch into several triangles, of which vertexes correspond to ball positions on the field and are associated with a set of target agent positions. In runtime, the estimated ball position is used to calculate the target positioning of the agents, based on the given training data.

Grid-based approaches have also been applied to robotics to represent and manipulate different kinds of spacial information, providing a simplified way of perception and modelling [Rosenblatt, 1998]. This type of representation is often oriented to a specific goal, such as identifying cells that are occupied by other robots to plan a motion trajectory.

Utility functions (also commonly called utility maps) have been presented [Chaimowicz et al., 2004, Spaan and Groen, 2003a] as a tool for role selection within multi-agents system and have recently been introduced in the MSL [Neves et al., 2015] as a method for positioning of soccer robots in defensive and offensive set pieces, as well as in freeplay pass situations. The use of utility maps is extremely relevant in the context of this league, specially due to its unpredictable nature and the need to react to opponent attempts to create opportunities to pass the ball to a more advantageous position.

4.3 Distributed vs. Centralized Assignment

4.3.1 Distributed Assignment

In multi-agent systems, a common distributed approach is to dynamically assign roles locally (on each agent) based on a set of pre-defined policies that depend on their world model *belief*. These policies must be defined in a way that guarantees convergence and avoid conflicts between intentions and/or actions [Groen et al., 2007]. This is often achieved using policy reconstruction methods, which make explicit predictions about an agent action, by explicitly running the decision-making algorithm of that agent, using shared plans [Grosz et al., 1999] or by using a learned model of the other agent behaviour [Hsieh and Sun, 2008, Farouk et al., 2017]. Some related work can achieve coordination when in low-communication and time-critical environments, provided that agents can periodically have full connectivity [Stone and Veloso, 1999].

However, each agent has a slightly different world model *belief* at a given time. It was possible to, in a real application, take an instant snapshot of all agents *beliefs* simultaneously, when inspeciting them we would find (slight) differences. This is due to the fact that the information residing on the agent world model is affected by many disturbances (starting on the measurement itself, partial observance, unideal modelling, noise, integration errors and even network delays). Therefore, the obvious drawback of the distributed approach is that agents are basing their decisions upon different *beliefs*, which can easily lead to lack of consensus and conflicting decisions. Most distributed policies designed to reach a consensus on task assignment are based on the fact that there is a common world model *belief* for all agents, which in real time-constrained highly dynamic applications is rare.

To overcome this problem the agents can, instead of deciding locally and instantly committing to a given decision, broadcast the intention and then use distributed negotiation algorithms to deal with any conflicts [de Weerd et al., 2017]. However, this approach is dependent on the network conditions and negotiation is not practical when the application demands a high level of responsiveness/reactivity.

4.3.2 Centralised Assignment

As opposed to fully distributed approaches, centralised architectures rely on a single agent to control and monitor the action plans of all other agents. Since this coordinator agent gets to decide on the final plan (which includes all agents partial plans), any conflicts between agents' plans can be taken into account during the planning process.

Centralising the decision to achieve consensus has some advantages over a distributed approach:

- Centrally solves the problem of *tightly-coupled coordination* that arises whenever there are one or more actions of one agent that affect the optimal action choice of another

team member, since the decisions are based on a single belief of the world model.

- From a software architecture point of view, it is simpler to implement and maintain.

Albeit these positive aspects, a completely centralised approach has three main drawbacks with respect to decentralised solutions:

- **No redundancy**. The coordinator agent constitutes a single point of failure - if it fails, the whole team may fail due to lack of coordination.
- **Network delays can propagate to actions**. Decisions need to be communicated to the agents, which take time that may be critical in some applications, depending on the authority level of the coordinator agent and the network delays.
- **Limited scalability**. A higher number of agents will require higher computational power on the coordinator agent to process and to devise a plan for the complete team of agents.

4.3.3 Centralised Assignment With Leader Election

When scalability is not a priority, some systems rely on centralised decision taken by one of the participating agents. However, network delays make it unfeasible for the leader to make realtime decisions when the environment requires a high level of reactivity. Therefore, the leader has to provide the team high-level coaching hints that will work towards a group consensus, while leaving low-level decisions (the fast-paced action that needs to be taken locally) to the other team-members.

The election of the coordinator agent is a fundamental part of this type of architecture to overcome a possible faulty coordinator. In case the coordinator fails, the agents need to recognise that failure and then communicate among them to find consensus on which agent should become the next leader.

4.4 The Consensus Problem

The consensus algorithms described below were designed to achieve consensus between processes or between server clusters, but in this section the more inclusive term *node* will be used to refer to either an *agent* or a *server* in a cluster. Consensus is a general term used to describe a state where participating nodes on a system agree on something, bound under certain conditions.

When applied to Multi-Agent Systems, consensus algorithms allow a group of agents to work coherently, enabling the system as a whole to survive in the event of sporadic failures of one or more of its members. For example, consensus algorithms have been successfully implemented for distributed storage on server clusters using log replication [Oki and Liskov, 1988], as well as in robotic networks [Montijano and Sagüés, 2015].

4.4.1 Paxos Algorithm

Over the last decade, Paxos [Lamport, 1998] has dominated the subject of consensus algorithms for software systems. Paxos has either been applied to or influenced many systems to solve a consensus problem on *nodes* clusters. Multi-Paxos [Van Renesse and Altinbuken, 2015] is also proposed in the literature, as an optimisation to Paxos - it essentially skips one step, which has no impact on coherence, for scenarios where the leader remains the same and online for a long period of time.

However, the (Multi-)Paxos algorithm, which was conceived upon a complex theoretical model, makes it less convenient to implement in real-world systems: in order to properly run it on practical systems, significant changes to its architecture are required [Chandra et al., 2007].

4.4.2 Raft Algorithm

As a response to the concerns mentioned above, Diego Ongaro and John Ousterhout have developed the Raft algorithm [Ongaro and Ousterhout, 2014], with understandability and implementability as a primary goal, but without compromising the correctness or efficiency of the (Multi-)Paxos. Using techniques such as decomposition and state space reduction, the authors have not only been able to separate leader election, log replication and safety, but also reduce the possible states of the protocol to a minimal functional subset.

Assumptions

The following three assumptions are made by Raft:

- **Machines run asynchronously**: there is no clock synchronisation between different systems and there are no upper bounds on message delays or computation times.
- **Unreliable links**: possibility of indefinite networks delays, packet loss, partitions, reordering, or duplication of messages.
- **Unreliable nodes**: processes may crash, may eventually recover, and in that case rejoin the protocol. Byzantine failures [Lamport et al., 1982] are assumed not to occur.

Consensus by Strong Leadership

Raft achieves the consensus by a strong leadership approach. In steady-state, a node in a Raft cluster is either a *LEADER* or a *FOLLOWER*. There can only be one leader in the cluster and when the leader becomes unavailable, an election occurs, and nodes can become *CANDIDATE*s.

In the original Raft system applied to log replication, the *LEADER* is fully responsible for managing log replication to the followers (the remaining nodes) and regularly informs the

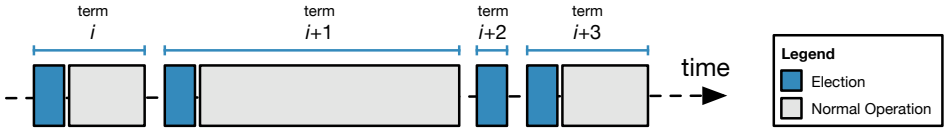


Figure 4.1: Timeline of example execution of Raft, adapted from [Ongaro and Ousterhout, 2014]. In three of the presented *terms*, a leader was elected after an election period. Only in *term* $i+2$, consensus was not reached during the election process, so a new election stars.

---

followers of its existence by sending heartbeat messages. Upon receiving this heartbeat, a *FOLLOWER* node resets a timer - whenever this timer reaches a timeout value the node can initiate a new election by becoming a *CANDIDATE*.

Each leader is elected for a *term* - a discrete temporal identifier (counter). At most, one leader can be elected in a given *term* and the event of a new election marks the start of a new *term*, as depicted in Figure 4.1. When a node enters the *CANDIDATE* state, it sends a packet to the other nodes informing them about the new term ID and its intention to become the leader - the other nodes then vote for or against it.

During an election, three situations can occur:

1. The majority of the nodes vote for the *CANDIDATE*, meaning this node can switch to the *LEADER* state and start sending heartbeat messages to others in the cluster to establish authority.

2. If a *CANDIDATE* receives any packet with higher *term* number than its own, they fall back to *FOLLOWER* state and vote for the sender candidate. When the *term* number is smaller than its own, the packet is ignored the node remains a *CANDIDATE*.

3. The *CANDIDATE* neither loses nor wins. If more than one node becomes a *CANDIDATE* at the same time, the vote can be split with no clear majority. In this case, a new election begins after one of the *CANDIDATE*s times out.

Limitations

Two main limitations have been identified in the Raft protocol:

- **1 and 2 active nodes corner-cases**: when there are less than 3 nodes available, Raft will fail to elect a leader, because it is impossible to achieve the majority of votes in either of the cases.

- **No prioritisation**: nodes are equally probable of becoming the leader. In some heterogeneous clusters, the user might want to defer the leadership to a node that has more computing power available.

4.5 Proposed Solution For Leader Election

In this work, a new leader election solution is proposed, based on the ideas of Raft algorithm for that purpose, but including some adaptations to overcome the identified and aforementioned limitations. Furthermore, we have integrated it in the RtDB middle-ware as an asynchronous service. By doing so, the information from the current leader is available for all agents at any time with no need for application-level re-configuration.

4.5.1 Timing Parameters

Three crucial aspects to consider when implementing this solution are the parameterisation of the sending frequency of heartbeat packets ($f_{\mathrm{HB}} = 1/\Delta T_{\mathrm{HB}}$), the heartbeat timeout ($T_{max,\mathrm{HB}}$) and the election timeout ($T_{max,E}$). Despite Raft originally suggesting times in the order of tens or hundreds of milliseconds, the selection of these times depends a lot on the application, fail-frequency and the communication medium between nodes.

In most mobile robotic teams, the robots communicate with each other in one or more of the many different available forms of radio communication. In this particular application, robots are using the Wi-Fi (IEEE 802.11a standard) in a spectrally dense environment, with strict bandwidth limitations (currently $2.2 Mbit/s$).

To select the heartbeat frequency f_{HB}, a trade-off between delay in the start of a new election and bandwidth expense has to be considered, while accounting for the actual role of the leader and the frequency at which he produces new information for the team. This is important because the robots will follow the latest available order while a new leader is being elected.

The heartbeat timeout $T_{max,\mathrm{HB}}$ should be selected in line with the packet loss experienced in the test-bed environment. For example, when selecting an heartbeat timeout that is more than twice the maximum heartbeat packet period, then a new election will occur when two consecutive heartbeats are not received from the leader.

The election timeout $T_{max,E}$ accounts for the time we allow the exchange of vote packets and is important whenever no majority of votes is achieved by any of the candidates.

Based on these assumptions, the values were empirically selected as follows based on background knowledge about the communication problems, with the ranges defining the limits of random uniform distributions.

$$40ms \leq \Delta T_{\mathrm{HB}} \leq 60ms \tag{4.1}$$

$$250ms \leq T_{max,\mathrm{HB}} \leq 400ms \tag{4.2}$$

$$T_{max,E} = 100ms \tag{4.3}$$

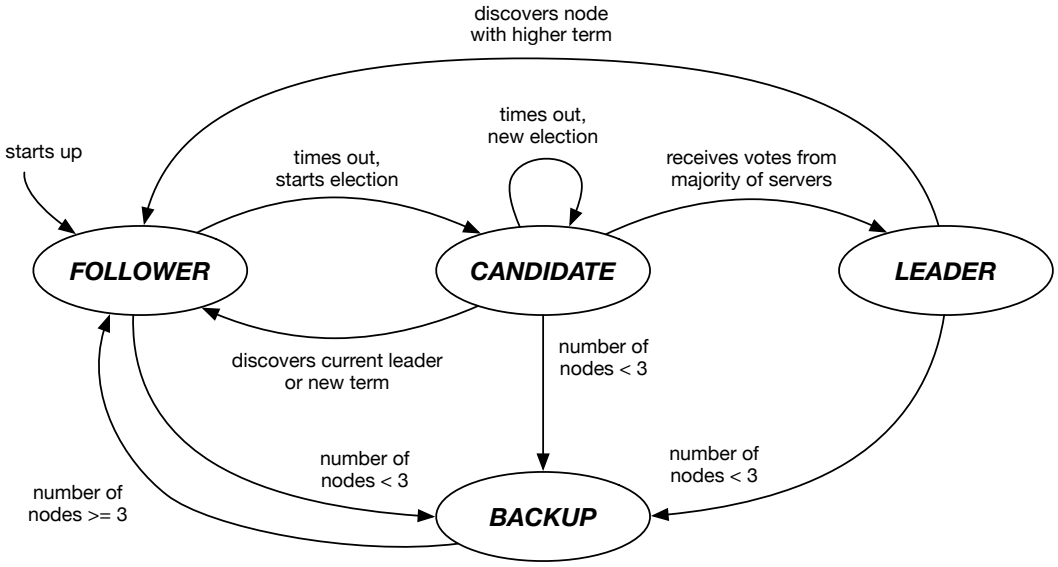


Figure 4.2: State Machine of The Proposed Solution

4.5.2 The Backup State

To tackle the first Raft limitation identified in section 4.4.2 (failure to achieve majority in an election with less than 3 active nodes), apart from *LEADER*, *FOLLOWER* and *CANDIDATE* states in the original Raft algorithm, the *BACKUP* state was introduced, which is triggered whenever there is only 1 or 2 active nodes in the system. The complete state machine is presented in Fig. 4.2.

Each node maintains a dynamic dictionary of timers, indexed by the peer ID. The timer belonging to a peer node is reset whenever a packet is received. The active nodes list is determined by the peer IDs in the dictionary that have an elapsed timer lower than the heartbeat timeout. The *LEADER*, *FOLLOWER* and *CANDIDATE* states indirectly force the nodes to send packets periodically (either actively or as a reply/vote), but not the *BACKUP*. Therefore, a new type of packet (keepalive) was introduced, which is sent periodically by nodes in the *BACKUP* state to keep the active nodes list up-to-date.

It was also defined that when in a *BACKUP* state, the leader is determined by the lowest ID among the active nodes.

4.5.3 Preferred Leader Agent

In our particular application, in order to free computational resources from the robots, it is wise to give preference to a coach computer to be the leader, whenever it is available, while keeping the leader election functionality active as a redundancy mechanism for failures.

In order to achieve this priority for the coach agent, while keeping harmony and consistency among the voting agents and the voting process, we skip the heartbeat timer reset on the

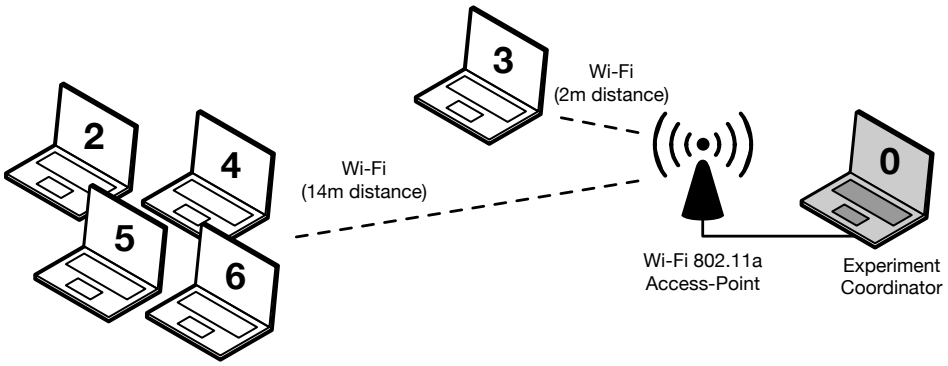


Figure 4.3: Experimental Setup

coach agent. When joining the network, the coach agent will start as a follower and will start receiving the heartbeat packets from the current leader, updating its *term* accordingly, but ignoring the heartbeat timer reset step. When reaching the heartbeat timeout, the coach agent will start a new election in a higher *term* and the previous leader will retreat. This constitutes the only situation when an agent intentionally takes over the team leadership, but a solution to solve the lack of prioritization leadership attribution of Raft.

4.5.4 Experimental Setup and Results

To test this solution, an experimental setup has been devised with 5 computers running the communication process with the leader election algorithm described in the previous section and an experiment coordinator. The coach agent was disconnected. The coordinator was impaired from becoming the leader, but participates in the voting phase and is responsible for monitoring the leader selection evolution, measuring times and forcing periodic communication failures (each 5 seconds, approximately) on the elected leader, hence triggering a new election, and logging data for offline analysis. The setup is depicted in Fig. 4.3.

The system has been setup and worked for 16 hours and 37 minutes, producing a total number of 11970 *terms*. From this dataset it is possible to statistically analyse the performance of the proposed solution with respect to *term* period, election time, occurrence of simultaneous multiple candidates in an election, leader attribution distribution and also the number of failed elections due to lack of majority in the voting process. Among all samples, the average measured *term* time was $5000.15 \pm 92.73ms$, which is consistent with our experimental setup described above.

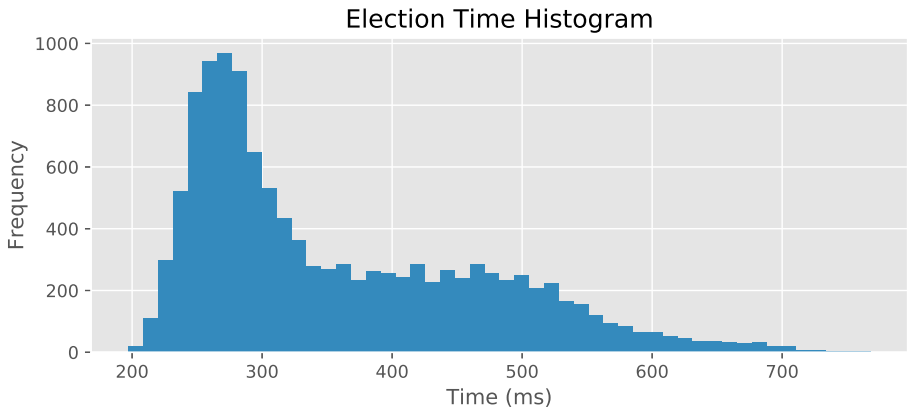


Figure 4.4: Election Time Histogram

Failed Elections

Having the coordinator participating in the voting rounds, makes it possible to tie a voting, because the total population consists of 6 agents. A failed election occurs whenever none of the candidates receives the majority of votes. From the total number of 11970 *terms*, there were 2 registered failed elections.

Election Time

Election times were inspected (Fig. 4.4) showing that they follow a distribution that is consistent with the selected heartbeat timeout time $T_{max,HB} \in [250 - 400]ms$, picked by a random uniform distribution on that range.

It is also important to mention that apart from the results presented on Fig. 4.4, there were 5 other samples with higher election times, namely: 1.1, 2.2, 2.6, 2.7 and 2.8 seconds. Because these account for 0.04% of all samples, they can safely be treated as outlier samples (of which cause can be related to external factors, such as the WiFi), therefore these were not included in the Figure to improve visibility of most samples in the plot.

Simultaneous Multiple Candidates

The occurrence of multiple candidates for election was also analysed. These results are shown in Fig. 4.5 (with the y axis in a logarithmic scale), were we can see that in 97.4% of the samples there is only one candidate, 2.5% two candidates and 0.03% (only 3 times in the whole run) three candidates, which was the maximum count for multiple candidates in a single round.

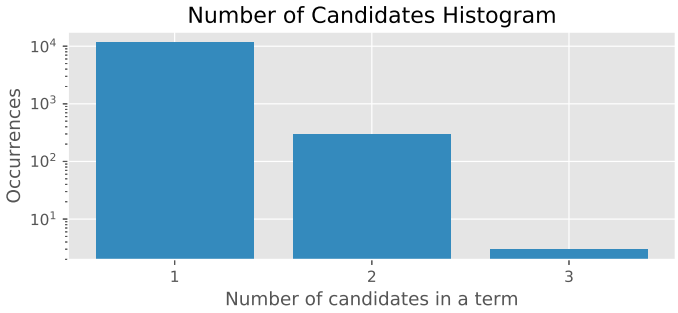
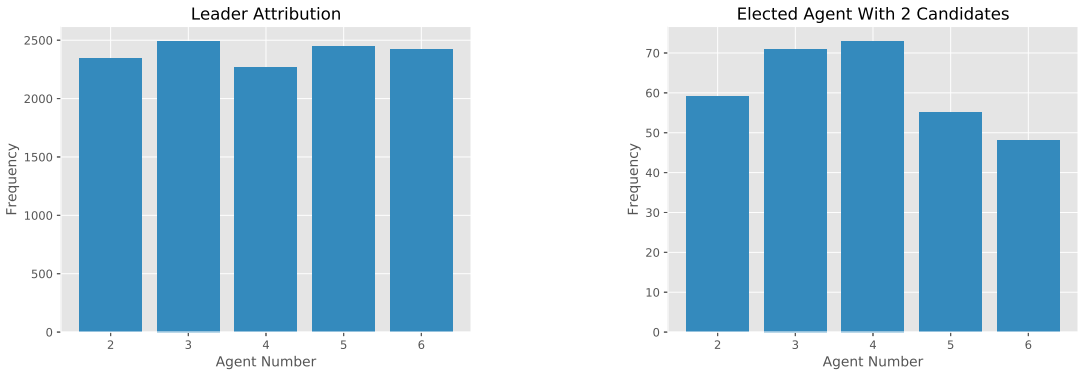


Figure 4.5: Number of Candidates Histogram - y axis in logarithmic scale

Leadership Attribution

A uniform distribution of leaders among the eligible agents was expected, however there are small differences between the agents, as shown in Fig. 4.6a. Since agent 3 (the laptop closer to the access-point) showed the maximum number of wins in elections, we wanted to investigate further if this was merely a coincidence or if the relative position to the access-point would affect the priority of being selected as a leader when there are multiple candidates.



(a) Global attribution

(b) Multiple candidates attribution

Figure 4.6: Leadership attribution analysis

To test that hypothesis, we analysed the leader attribution in the *terms* for which there were more than one candidate (Fig. 4.6b). These results do not show a clear higher chance of agent 3 to become a leader in conflict situations. Because in this setup many external factors can influence the communication medium, further tests must be performed with laptops' positions shuffled between runs.

4.6 Summary

In this section, a new solution was proposed to coordinate multiple agents in a MAS in a hybrid fashion - decisions can be centralized in an agent, but a new leader is elected using a distributed voting system based on Raft in case the current leader fails. Tests were conducted on a lab experiment that forced leader re-elections periodically.

This solution solves a corner-case of the original Raft algorithm for few agents running, while adding the possibility of having a preferred leader among the team, which is useful when resources are not evenly distributed.

The results showed the expected results and therefore, this solution has been successfully integrated and adopted into the CAMBADA agent architecture to be used as an integral part in the following Chapter 5.

Chapter 5

CR7 Setplay Engine

In the context of robotic soccer, once a solid baseline of skills is established, teams usually focus on higher-level features.

In 2014, the CAMBADA agent software architecture has been redesigned [Dias, 2014] to address this issue and turn the CAMBADA architecture into one that could be easily modified without compromising the final result - something that the previous architecture failed in providing, since it was based on overly complex state machines with manually-defined transitions. In order to achieve the best results in the competitions, it is utterly important that the software architecture is modular and flexible to allow the deployment of different strategies for different teams.

With a matured modular software architecture in place and making use of the enriched world model with the features detailed in the previous chapters, it was possible to develop a new setplay engine - the CAMBADA Cooperation fRamework 7 (CR7) engine.

5.1 Motivation

CAMBADA has been using an existing tool (Figure 5.1) to configure the robots positioning in setpiece situations (Freekick, Kick-Off, Corner, etc.). The field was divided into 10 zones and this tool allowed the user (defined as the human team member that uses the available tools) to setup a single-step (one pass) setplay for each zone. This simple, yet effective, tool provides a few features that should be highlighted:

- **Robot Availability Flexibility**: all the defined robot positions have a priority and, at run-time, robots are assigned in order to these priorities. When not all the robots are playing in the field, positions with lower precedence are not used.

- **Pass priority**: Among the available receivers, the pass-line is evaluated in an order that can be different than the receiver assignment order above. The pass will be made to the first available pass line to a receiver.

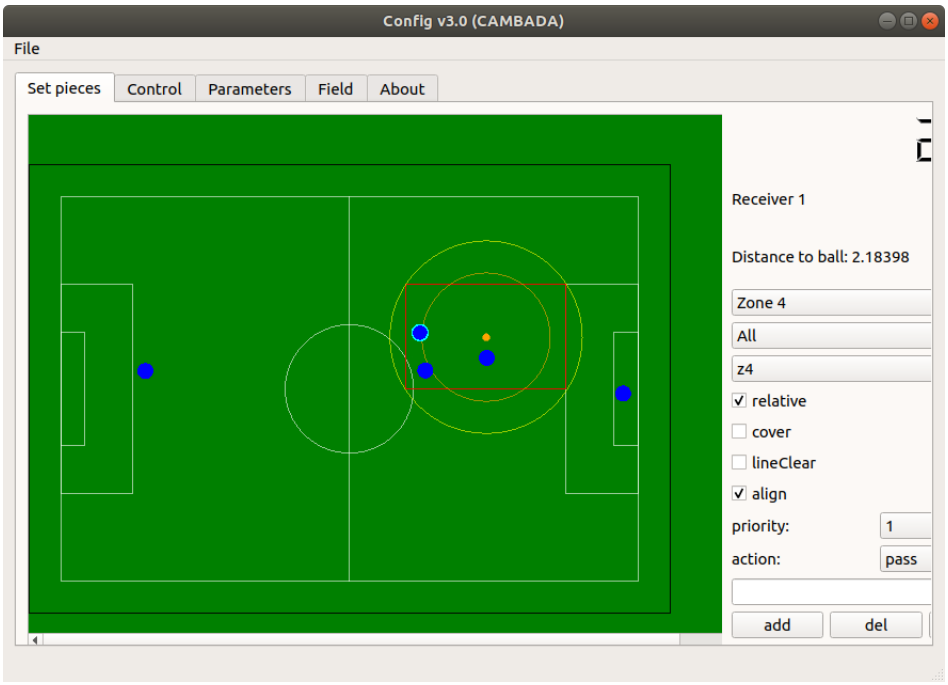


Figure 5.1: The existing CAMBADA setpieces configurator

- **Options per receiver**: There are a few check-boxes that slightly change how the robots behave when executing the set-piece. These allow to set a robot position as relative to the ball, to enter a "cover" stance (that relates to a specific CAMBADA behaviour), to enable and disable check for line-clear, to dynamically align with the opponent goal when facing the ball (more on this later in this chapter).

However, this workflow does not account for multi-step setplays nor free-play situations. One solution to overcome this could be the integration of the third-party setplay engine from the FC Portugal team (Simulation 3D league) [Mota et al., 2011] into CAMBADA, for which a couple of attempts were performed. This setplay engine is highly flexible with a few features to be highlighted:

- **Multi-Step Execution**: the setplays can have multiple available steps, which are executed in sequence;
- **User-Defined Conditions**: all the conditions to initiate a setplay and also transition from one step to another are completely defined by the user, with a set of helper conditions available to the user, which include, but are not limited to, ball possession, ball position, player position and alike;
- **Configuration UI**: The setplays and the respective steps can be visualized and edited in a configuration tool called "SPlanner".

However, a few issues were raised by the team in the latest attempt to integrate this engine in the CAMBADA architecture, namely the cost of flexibility - the FC Portugal engine makes no assumptions about any conditions: the user is therefore required to input all invocation conditions, all conditions to transition between steps and also abort conditions. Failing to input all the correct conditions would result in an unexpected behaviour or even dead-locks. Moreover, the actions to be executed during a setplay (pass, move, etc.) had to be implemented in a class that most of the times replicates the regular team play behaviours. Maintaining an extensive codebase with duplicate behaviours was also pointed out as a major drawback.

Additionally, using independent solutions for stop-game setpieces and free-play setplays would imply having different configuration tools, which is not desirable nor scalable - apart from the obvious multiple configuration tools and files, there is no relationship between the two, so there was no elegant solution to force a setplay to initiate after a setpiece situation.

The aforementioned drawbacks of the two solutions were the main motivation to create the CR7 engine. This proposed solution, built from scratch, makes use of a few of the FC Portugal engine components as baseline (mainly the configuration UI and the ideas of setplays composed of multiple steps and transitions between steps) and some concepts of the existing setpieces tool in order to provide an integrated way to configure planned setplays execution. In order to achieve that, a set of requirements was settled:

- **Valid for Free-play and Setpiece situations**: the solution must account for setplays that are initiated in the middle of a match in free-play, but also setpiece situations that must account for the status of the game in a special way (the robots should position themselves prior to a referee start);

- **Flexible Dynamic Positioning**: it is possible that, at runtime, during the match, a target position for a robot is already occupied by an opponent. However, the play should continue if the robot can find another valid position nearby the pre-defined position. Therefore, all target positions should be assigned a radial margin (which could also be infinite);

- **High-Level Abstraction**: the user should not be required to input all conditions, as most of the times the engine can infer those conditions from the defined actions. For example, for a pass action, the condition to start the transition would be a robot with the ball possession and a receiver within the target area (position + margin) with line clear for the pass. Similarly, the transition ends when the receiver grabs the ball;

- **Distributed Coordination**: the execution of a setplay usually requires a "conductor" and the previous integration of the engine in CAMBADA used the *coach* for that purpose. It is possible to eliminate this single point of failure, as setplays should continue to operate with the Leader Election mechanism (Section 4.5);

- **Tight integration with the existing architecture**: this includes reusing existing behaviours (for ball protection, pass, receive, etc.) and an execution engine that fits with the existing *BDI* architecture [Dias, 2014];
- **Drag-and-drop UI Interaction**: it should be possible to define the simplest properties of setplays actions with just the mouse in a drag and drop fashion. For example, dragging from one robot to another one should assume a pass action, or dragging to a free position in the field should assume a move action, etc. Additional fine-tuning configurations should also be possible;
- **Configurable Zones**: the zones in the previous tool were fixed and hard-coded, which sometimes was limiting. The team requires a flexible way to define field regions (called "Zones").

The CR7 software package is composed of two main components: a **library** and a **configuration tool**. Both will be detailed in the following sections 5.2 and 5.3, respectively, along with the design choices discussion.

5.2 Library Software Architecture

The library was implemented in `C++`, making extensive use of the object oriented programming paradigm.

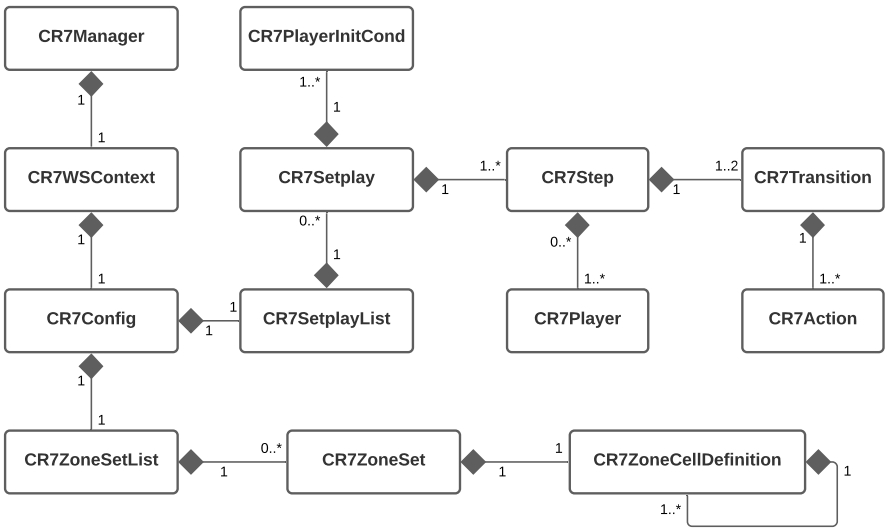


Figure 5.2: CR7 Classes Overview UML Diagram

Figure 5.2 shows an overview of the framework classes. In the following sections, the architecture will be thoroughly described along with explanations for some of the design choices. Full class documentation is available in the Appendix A.

5.2.1 Types of Setplays

The CR7 Engine divides the Setplays in 3 different types: **Freeplay**, **Kick-Off** and **Setpieces**. This separation comes from the fact that each of these three situations must be handled in a different way in terms of activation, release and transition evaluation conditions:

- **Freeplay**: setplays to be executed during regular operation, assuming one robot player has the ball possession. Initialization conditions depend on the participating players availability and their absolute positions on the field.

- **Setpieces**: setplays for setpiece situations other than Kick-offs (mainly Free-kicks, Throw-ins, Goal-Kicks and Corner-kicks). The initialization conditions depend on the participating players availability, on the game state being a setpiece one and the ball position being inside a zone in the field that is associated with the setplay.

- **Kick-Off**: special setplays tailored at kick-off situations. Initialization conditions depend on the participating players availability and on the game-state being Kick-off. This type of setpiece is isolated from the remaining ones because it has special constraints - the ball position is previously known and the robots initial positions must be constrained to the team own field.

By separating the Setplays in these three groups, the user is not required to take any special considerations about initialization conditions specificities, since the engine can assume the aforementioned conditions for each case based on the setplay type.

5.2.2 Players Definition

Team players are assumed to take roles (such as attacker, defender, midfielder, etc.) that are unique among the team and robots can change roles dynamically during the match, but each role can only be assigned to one robot at a time. The only assumption made is that the Striker (also known as "attacker") is the one that has the ball in free-play or is the one to get the ball and perform the first pass in kick-offs and other setpiece situations.

The Player IDs defined in CR7 by default are: *Striker*, *Supporter*, *Midfielder* and *Defender*. Because this has a strong correlation with the team architecture, the management of available player IDs is performed by configuring the `SETPLAYS_PLAYER` enumerator in the `CR7Defines.h` header file.

5.2.3 Actions Definition

In each transition between steps, each participant player has an associated explicit action, which can be one of the following:

- **Wait**: The player does nothing in this transition.
- **Move**: The player moves to a defined position (with margin). If the player is the ball holder, a dribble is performed.
- **Pass**: Action only available for the ball holder, make a pass to another player.
- **Get Ball**: only available in Kick-Off and SetPiece situations, until the ball is grabbed by one of the participants.

Any references to the `SETPLAYS_ACTION` enum in this chapter are related to the actions above. It is important to note that each Action above does not translate directly into an agent Behaviour because there are also implicit actions that arise from the defined interactions. For example:

- **Ball Protect**: The ball holder is expected to protect the ball from opponents while the Setplay is executing - it is not acceptable that it stays still waiting for the receiver team-mate to find a position with a clear pass-line. Therefore, when a ball holder has a Wait action, it will protect the ball from the opponents. It will also protect the ball while executing a Move action and even when its action is Pass, but it is waiting for a clear pass-line.
- **Receive Ball**: Receiving the ball is not an explicit action for a player - the receiver can either have a Wait action (and receive the ball in the position where it is already) or a Move action (and receive the ball in a different position - if the situation allows, a forward pass can be achieved). This Receive Ball action is triggered whenever another player has the Pass action for the target player of the pass.
- **Recover Ball**: Real systems fail. Failing to correctly receive the ball after a pass could lead to a Setplay abortion. However, it is sometimes possible to recover the ball within a relatively small time frame. It is probably wiser to allow the receiver to recover the ball and try to continue with the Setplay instead of aborting it. While the ball possession is evaluated to be on our team's side and the execution is within the time limit, the players will keep executing the Setplay.

5.2.4 Positioning Options

All positions defined in a Setplay have an associated margin. The Players position in the first step serve as the initialization conditions for the setplay. Freeplay Setplays, among other conditions, can be executed if, during the match, the players happen to be in the positions defined in the first step, within the given margin. For setpieces, these positions serve as hint positions for the players - the receivers use the are within the margin to position themselves in a place where a pass is possible.

When a Setplay is executing, any positions have an associated margin to allow players to dynamically react and adapt to the opponent team while thriving to finish the execution of the Setplay.

Specifically for Setpieces, it is possible to define positions in absolute field coordinates or relative to the ball position. Additionally, also exclusively for **Setpieces**, it is possible to define a boolean option to dynamically align a player with the opponent goal. It is important that both players are aligned with the goal before the initial pass, to be able to make the pass and kick as quickly as possible to the opponent goal.

5.2.5 Architecture and Design Choices

In this section, some of the most relevant classes are presented, along with their most important methods, including brief explanation of the functionality and their interaction. The main purpose of this section is to justify some design choices, and it is roughly organized in a bottom-up approach following their interaction model - refer to Figure 5.2 for an overview of the framework classes and the Appendix A for a complete class reference.

Config

Reference C++ class: `CR7Config`.

This class is responsible for reading from and writing to configuration files. The CR7 configuration file is a JSON file that follows a pre-defined schema (Appendix A.16). JSON was chosen because it is human-readable and there are already many libraries for all platforms that allow operations on JSON objects, therefore it is scalable for future updates, namely integration with other tools.

Most relevant methods:

- `void LoadConfigFile(const std::string& filepath)`
  Reads the file passed in the `filepath` argument and parses its contents into internal configuration variables. Throws an exception with a description of the failure.

- `void SaveConfigFile(const std::string& filepath)`
  Dumps the contents of the internal configuration variables into a properly formatted

JSON file in the specified `filepath`. Throws an exception with a description of the failure.

- `CR7ZoneSetList* GetZoneSetsList()`
  Returns the list of Zone Sets.

- `CR7SetplayList* GetSetplayList()`
  Returns the list of Setplays.

World State Context

Reference C++ class: `CR7WSContext`.

The World State Context class serves as the interface class for the application architecture world model. The application that is going to integrate CR7 needs to instantiate and feed a `CR7WSContext` with live data from its world model regarding the game state, the players pose and velocty, the ball position and velocity, evaluation of clear pass lines, etc.

Management of running setplays during runtime is highly dependent on this class, as it provides all the information required to trigger, release and manage setplays.

All methods

- `std::vector<SETPLAYS_PLAYER> CR7_getAvailablePlayers()`
  Returns a list of the currently available players.

- `Vec CR7_getPlayerPosition(SETPLAYS_PLAYER playerId)`
  Returns the absolute position of a player.

- `int CR7_getPlayerAgentIdx(SETPLAYS_PLAYER playerId)`
  Returns an unique ID of a player. Unlike roles, the unique ID does not change for a robot.

- `Vec CR7_getPlayerTargetPosition(SETPLAYS_PLAYER playerId)`
  Returns the absolute target position of a player (position in the field the robot is moving to).

- `bool CR7_playerHasBall(SETPLAYS_PLAYER playerId)`
  Returns true if given player has the ball, false otherwise.

- `bool CR7_playerPassedBall(SETPLAYS_PLAYER playerId)`
  Returns true if given player has passed the ball, false otherwise.

- `bool CR7_isBallVisible()`
  Returns true if the ball position is known, false otherwise.

- `Vec CR7_getBallPosition()`
  Returns the absolute ball position.

- `Vec CR7_getBallVelocity()`
  Returns the absolute ball velocity.

- `bool CR7_ballInOurPossession()`
  Returns true if the ball is in team's posession, false otherwise.

- `bool CR7_ballInTheirPossession()`
  Returns true if the ball is in opponent's posession, false otherwise.

- `bool CR7_isInsideField(Vec testPos, float outMargin)`
  Returns true if the `testPos` position lies inside the field (augmented by a margin defined by `outMargin`), false otherwise.

- `bool CR7_passLineClear(Vec p1, Vec p2)`
  Returns true if there is a clear pass line between two absolute points defined by `p1` and `p2`, false otherwise.

- `bool CR7_GameState_Changed()`
  Returns true if a game state change occurred between the last cycle and the current one, false otherwise.

- `bool CR7_GameState_Stopped()`
  Returns true if the current game state is Stop, false otherwise.

- `bool CR7_GameState_KickOff()`
  Returns true if the current game state is KickOff, false otherwise.

- `bool CR7_GameState_Setpiece()`
  Returns true if the current game state is Setpiece (Free-kick, Corner, Throw-in, etc.), false otherwise.

- `bool CR7_GameState_Freeplay()`
  Returns true if the current game state is Freeplay, false otherwise.

Manager

Reference C++ class: `CR7Manager`.

The Manager is the entity responsible for managing the execution of Setplays (Section 5.2.5). The list of available Setplays is given by a Config (Section 5.2.5) instance - the manager goes through this list to check if any Setplay execution can be initialised. The Manager constitutes the entry point for every agent that participates in the Setplays (all robots agents and also the coach). Internally, the Manager uses the Leader Election mechanism to decide if it should lead the Setplay management or follow a leader.

The setplay execution state is synchronised between all agents using a new shared item in the CAMBADA RtDB, which includes two fields: the executing (if any) *Setplay* identifier and the corresponding currently executing *Step*.

Most relevant methods

- `void update()`
  Must be called every cycle. If the caller is the current leader, it manages the setplay execution; if not, it synchronises local information with the information shared by the current leader.

- `bool isRunningSetplay()`
  Returns true if executing any Setplay, false otherwise.

- `CR7Setplay* getRunningSetplay()`
  If running a Setplay, returns a pointer to its object, or `NULL` otherwise.

Setplay

Reference C++ class: `CR7Setplay`.

This is the main container of a Setplay, with properties that include the name, Setplay type, enable status, zone definition (when type is Setpiece), maximum execution time and cooldown time (the time during which the setplay is temporarily disabled after it was executed). It also contains methods and properties for runtime operation that serve as helpers for the Manager.

Most relevant methods

- `bool IsRunning()`
  Returns true if the Setplay is running, false otherwise.

- `bool HasFinished()`
  Returns true when an end-step (a step without exit transitions) is reached, false otherwise.

- `bool CanExecute(CR7WSContext* world)`
  Returns true if all the conditions for execution are met. If the Setplay is not running, it checks if all the invocation conditions are met to start execution. If the Setplay is already running, it checks if the next transition is still valid.

- `void UpdateExecution(CR7WSContext* world)`
  This method is called by the Manager to update the execution based on an updated new World State Context. It updates internal variables and is responsible for checking if a transition has ended to move to the next step or abort the Setplay.

- `CR7Step* GetCurrentStepPtr()`
  Returns a pointer to the currently executing Step.

Step

Reference C++ class: `CR7Step`.
Step objects contain participant Players and associated Transitions.

Player

Reference C++ class: `CR7Player`.
This class holds information about a Player state, namely its configured target position and margin, the ball holding status and other flags for specific situations (such as setplays).
Most relevant methods

- `SETPLAYS_PLAYER getPlayerID()`
  Returns the player ID associated with this Player.
- `bool getBallHolder()`
  Returns true if this player is the ball holder in the respective Step.
- `Vec getPosition()`
  Returns the configured target position.
- `double getPositionMargin()`
  Returns the configured target position margin.
- `SETPLAYS_REL_TYPE getRelType()`
  Returns the type of relative positioning (if any) to be considered.
- `bool getAlignWithGoal()`
  Returns true if the player should align with the opponent goal. This flag can be used in setplays for receiving Players, to have them position themselves aligned with the ball and the opponent goal, regardless of the actual ball position.

Transition

Reference C++ class: `CR7Transition`.
A Transition is established between Steps and defines the actions for each Player.
Most relevant methods

- `CR7Step* getSource()`
  Returns a pointer to the source Step of this transition.

- `CR7Step* getDest()`
  Returns a pointer to the destination Step of this transition.
- `std::map<SETPLAYS_PLAYER, CR7Action*>& getActions()`
  Returns a map with the Actions for all Players.
- `CR7Action* getPlayerAction(SETPLAYS_PLAYER playerID)`
  Returns a pointer to the Action of the given Player.

Action

Reference C++ class: `CR7Action`.
The action holds information about an action to perform and its associated parameters.
Most relevant methods

- `SETPLAYS_ACTION getType()`
  The action type, as described in Section 5.2.3.
- `SETPLAYS_PLAYER getReceiver()`
  Returns the player ID when the type is Pass.
- `Vec getToAbsPoint()`
  Returns the target position when type is Move.
- `Vec getMargin()`
  Returns the target position margin when type is Move.
- `bool getOptPassIgnoreLineClear()`
  Returns true when the user configured this step to ignore the check for a clear pass-line when type is Pass.

Zone Sets and Zones

A Zone Set constitutes a way of dividing the field into smaller cells. In order to keep the divisions agnostic to field size variations over time, these are defined in a normalised and tree-like way. Starting with a full field, it can be divided either in the x-axis direction or the y-axis direction, with division thresholds being defined between -1 (field left or bottom) and 1 (field right or top). Then, each resulting cell of the division, can again be divided into other sub-cells with the same or a different direction, using the same normalisation technique.

After all the divisions and sub-divisions, field Zones are defined as the resulting end-cells (cells that are not sub-divided). This can also be seen as a branching model, where the Zones are defined as the leaves on this tree.

Several different ZoneSets can be included in the configuration file and be referenced by Setpiece type of Setplays.

5.3 CR7Planner - Configuration Tool

The CR7Planner is the configuration tool with a user interface to manage the CR7 configuration file in a visual way. It has been developed in C++ and Qt5, with some UI widgets and the layout adapted from the FC Portugal SPlanner [Cravo et al., 2014].

5.3.1 Extending CR7 classes for UI

One very important aspect of this configuration tool is that it uses the Config class to load and save the JSON configuration file into CR7 classes objects - this is utterly important to maintain the coherence between the JSON file generation from the CR7Planner and the parsing done by the agents.

For UI manipulation, Qt expects that each object inherits from the QGraphicsItem class. Instead of making Qt a dependency of the CR7 library, the proposed solution is to have separate classes for the CR7Planner that inherits both the QGraphicsItem and the CR7 base class, which will hereby be referenced to as "CR7 Qt classes".

A possible alternative would be to maintain two separate sets of objects (one set loaded and created by the CR7Config, and another of "CR7 Qt classes" created by the tool with a copy of the attributes from the first set) - but it would use more RAM than what is necessary and it would result in the same consistency problem that we were avoiding before by requiring the user to copy all attributes from and to the base classes for saving the config file using the CR7Config object in the end.

The proposed solution consists in making this transparent for the user, by performing a three-step operation on load:

- **Step 1. Load CR7Config**: The configuration file is loaded by the `CR7Config` and all objects are created in their CR7 base class form.
- **Step 2. Populate "CR7 Qt" objects**: The CR7Planner goes through the structure of the config (setplays, child steps, transitions, actions, players, etc.) and creates corresponding objects of the corresponding types that inherit the QGraphicsItem and the CR7 base class. The contents of the objects are copied by making use of the serialization and de-serialization methods that exist in the base classes - i.e. the objects are exported to JSON and then imported in the Qt objects.
- **Step 3. Pointer substitution**: This step consists in going through the base objects, free them and replace the pointers by the pointer to the Qt object that corresponds to the deleted base object. This works because the CR7 Qt classes extend the respective CR7 base classes (thus, they also contain the methods from the CR7 base class), they can be manipulated both by other CR7 base classes or by the configuration tool - this makes it highly transparent for the programmer.

After running this process, any "get" operation on the CR7 classes can be seamlessly upcast to a corresponding CR7 Qt class at any time in order to make use of the extended class methods.

5.3.2 Layout Overview

The CR7 Planner layout is tab-based, with two fixed main tabs: **Zones** (Figure 5.3) and **Strategy** (Figure 5.4).

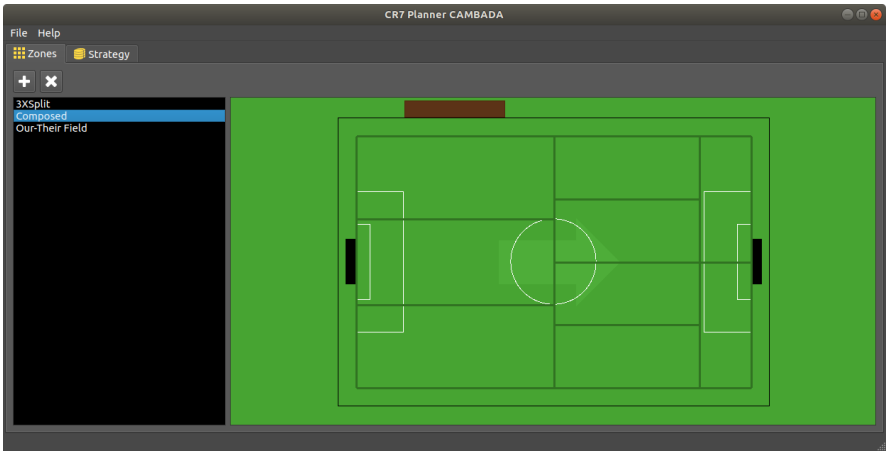


Figure 5.3: CR7 Zones Tab: shows a list of the Zone Sets and a visualisation for the selected Zone Set in the field 2D visualiser.

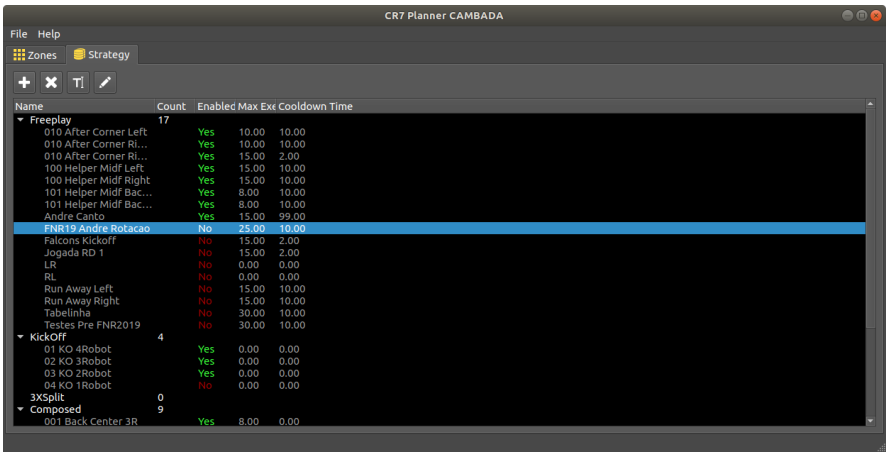


Figure 5.4: CR7 Strategy Tab: displays the complete list of Setplays, grouped by type (Freeplay, Kick-Off and the Zone Sets names).

It is possible to enable and disable a Setplay by double-clicking the "Enabled" column. The toolbar above allows the user to add and remove Setplays.

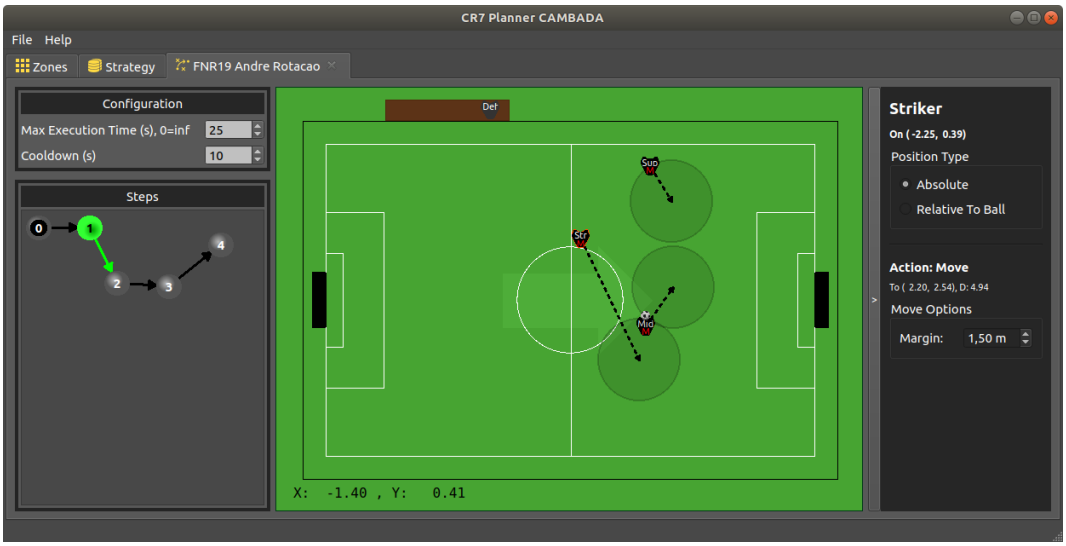


Figure 5.5: CR7 Setplay Tab screenshot.

When double-clicking on a setplay (or selecting it and clicking on the Edit button), a new (closable) editor tab opens for that Setplay. The Setplay editor tab is depicted on Figure 5.5 and its layout is divided into the following sections:

- **Top-left - Setplay global preferences**: maximum setplay execution time and cool-down time (can be expanded with further options in the future);
- **Bottom-left - Setplay Steps graph**: the Steps and their linked Transitions. It is possible to click each Step or Transition to select it for editing;
- **Center**: the field 2D visualisation with drawn robots, their actions and field Zone when applicable. Players Actions are set by interacting visually in this widget (explained in section 5.3.3).
- **Right sidebar**: properties for the selected Player. Depending on the situation, it may allow the user to set the positioning type relative to the ball instead of absolute in the field, the position margin, shows the Action data (like position and travel distance when in Move, receiver Player when in Pass, etc.) and allows to set the action options (margin, ignore line-clear flag, etc.).

5.3.3 Initialization Conditions and Player Actions

The user can edit the Setplay initialization conditions by dragging the players out of the "bench" with left mouse button, when in Step 0 (always the first step). Setting their position and respective margin defines the initialization condition based on the Players position.

Players in bench do not participate in the Setplay. For participant Players, when a margin of zero is applied to its start position, it is interpreted as infinite (i.e., the player is only required to be available, independently of its position on the field).

When editing Setplays of the Setpiece type, it is also possible to move the ball inside the selected field Zone and see the robots which position is defined as relative to the ball moving along with the ball. This is useful to check for conflicting Players positions.

Setting or editing Actions for Players can be achieved by dragging with the right mouse button starting on that Player:

- Drag to itself: sets a Wait Action (default);
- Drag to a field position: sets a Move Action to that position;
- Drag to another Player: sets a Pass Action to that Player;
- Drag to the ball: sets a Get Ball Action (only available in Kick-Off and Setpiece positions).

5.4 Unit Testing

In order to test the Setplay engine before integrating it with the CAMBADA agent, a unit testing framework has been devised using the Boost test library. A Dummy World State class has been created, which allows the user to manually set the positions for the players, their velocity, the ball position and velocity, ball possession, etc.

The test cases use a Manager that is plugged into an instance of the Dummy World State and manually force world state conditions between Manager `update()` calls and check for expected conditions. This proved to be a very useful tool to keep the coherence of the engine throughout development and has been included in the library to help on further development.

5.5 Integration in the CAMBADA Architecture

The CAMBADA software architecture is also implemented in C++ and already has a World State object, which is able to feed the CR7 World State Context object. Therefore, the integration started by making the existing World State class a child of the `CR7WSContext` class. The `CR7WSContext` class virtual methods were then implemented by calling existing methods of the CAMBADA World State.

A CR7 Manager was then added to the CAMBADA agent (which is a process running in every robot, as explained in Section 1.3) and the coach (process running in the basestation computer). Because of the way the previously presented Leader Election (Section 4.5) algorithm works in CAMBADA, the coach is selected as the leader by default and the robots will synchronise with the coach Setplay Manager execution state, as explained in Section 5.2.5

Manager subsection (page 71). The leader election algorithm ensures that if the coach process fails or is not running, one of the robots will assume the leadership and manage the Setplays for the whole team.

A Zone Set has been created that resembles the previous set of zones that CAMBADA used for the setpieces execution and multiple setplays for multiple situations have been added to the configuration file.

The CAMBADA Basestation [Figueiredo et al., 2009] tool has also been updated to display a preview of the running Setplay while the robots are executing it, showing target positions, margins, pass-line clear status and other relevant information in realtime.

5.6 Results

Following its successful integration within the CAMBADA architecture, the CR7 Framework has been thoroughly tested during an MSL competition, the Portuguese Robotics Open 2019, which follows the official RoboCup rules. Table 5.1 shows an overview of the results achieved by the CR7 framework in free-play during the tournament.

Stage	**Match**	**Executed Setplays**	**Success Rate**	**Lost Ball Possession**
Round-Robin 1	CAMBADA - Falcons	8	25%	0%
	CAMBADA - Tech	10	40%	17%
	CAMBADA - VDL	7	29%	0%
Round-Robin 2	CAMBADA - 5DPO	6	50%	0%
	CAMBADA - Falcons	6	50%	0%
	CAMBADA - Tech	9	0%	33%
	CAMBADA - VDL	6	50%	0%
Semi-Final	CAMBADA - VDL	5	40%	0%
Final	CAMBADA - Tech	3	33%	50%

Table 5.1: Results of the CR7 framework in the Portuguese Robotics Open 2019 in free-play setplays.

The **Success Rate** column represents the average success of all the setplays executed on the respective match. Considering that creating a opportunity to kick to the opponent goal is the ultimate objective of a setplay, it was considered to be successful in two situations:

- The setplay plan was executed until the last step without interruptions
- The setplay was interrupted because there was a chance to kick to the opponent goal

The **Lost Ball Possession** column represents the percentage of occurrences where our team has lost possession of the ball to the opponent team on unsuccessful setplay executions, excluding setplays that were aborted due to a game stop issued by the referee.

The actual list of configured setplays has changed multiple times throughout the competition (sometimes even between the first and second halves of the same match), while CAMBADA tried to adapt to the opponent team strategies. This also proves that the team can modify, create and disable CR7 setplays in an expeditious way.

Table 5.2 includes a summary of the causes for the setplays to finish throughout the same tournament.

	Reason to finish setplay	**Occurrences**
Successful	Kick and Goal	12
	Finished plan	5
	Kick	3
Aborted	Pass-line blocked (before pass)	21
	Lost ball control	11
	Ball intercepted by opponent after pass	5
	Game stopped by the referee	4
	Ball pass reception failed	3

Table 5.2: Reasons for the setplays to finish.

5.7 Future Work

There are a few ideas to enhance the current implementation of the CR7 engine.

The current version of the engine does not allow branching from steps (i.e. a setplay step can only transition into another step). This will require some kind of order prioritization regarding the condition evaluation of the multiple options, as well as the respective editor in the GUI tool.

One important feedback from the team was the possible need for additional options for setplays and steps. For example, an option to allow and disallow the interruption of the setplay on a certain step, due to an opportunity to kick to the goal. Another possible global setplay option would be a maximum number of executions per match/half. Options such as these are expected to be added according to the strategic needs of the team.

Another feature for future development is the ability to visually edit zones in the CR7 Planner. Currently, they have to be defined in the JSON file directly.

In terms of the strategic capabilities it provides, it would be interesting to add statistics digestion to the setplays, allowing these to be updated continuously and thoroughly. With such statistics, it would be possible to add long-time learning capabilities to the CR7 engine, training a model to maximize the success-rate based on the attempts made during the matches.

Chapter 6

Conclusion

In this Chapter, the main conclusions are drawn regarding the work devised and the resulting major contributions in the thesis topics: **perception**, **multi-object tracking**, **leader election** and finally the **CR7 setplay engine**.

6.1 Perception

This thesis work initiated with some improvements on the robots proprioceptive perception. By rearranging the main processes execution pipeline, it was possible to guarantee a significant jitter reduction in communication with the low-level. Furthermore, by incorporating extra information in the integration step (which included vision-flow, gyroscope and yaw measurements), the robot pose and velocity estimations were improved. Moreover, the high-level motion control was also improved by incorporating a new solution that takes into account some of the physical constraints related to the CAMBADA robots hardware, by using the angular difference between the relative target position and the estimated robot linear velocity. The maximum allowed linear velocity is then modulated in terms of this angular difference, which resulted in a more predictable robot path in some movements (a lot closer to the expected motion path), but also significantly optimizes the total trajectory time and thus faster motion transitions.

In terms of environment perception through vision processing, the ball detection algorithm was also improved by introducing two extra steps in the previous process: an edge detection and a RANSAC fitting, which results in a more precise estimate of the ball position, especially when the ball is partially occluded by another robot.

Finally, the obstacle identification algorithm was also improved for situations where the obstacle is partially occluded by the ball.

Although this work on perception might seem a bit off the remaining work, it provided a solid baseline for the following higher level developments.

6.2 Multi-Object Tracking

A solution for real-time multi-object tracking on a stochastic and highly dynamic environment was presented. The results on the testbed show the efficiency of the solution and its compliance with the real-time constraints of the robotic soccer context under benchmark. However, it should be stressed that the solution presented for distributed multiple object tracking is general and not limited to this specific application domain. While experiments were conducted under both lab conditions and real competitions, the lack of groundtruth data in the competition tests required the use the opponent localization as a groundtruth for the obstacles position.

This multi-object tracking solution can be applied in different scenarios which require some sort of efficient tracking of multiple objects of interest in real-time applications with strict timing constraints. With no prejudice on the general algorithm itself, the employed state estimation method (Kalman Filter (KF) in this case) can be adapted to the application whenever the KF does not provide satisfactory results, and there are computational resources available to do so - for example, the KF can be swapped by an Extended or Unscented Kalman Filter [Julier and Uhlmann, 2004], Interacting Multiple Models [Bar-Shalom et al., 2002], Multi-Hypothesis Tracking [Reid, 1979], etc. An additional important consideration is that each tracklet on the tracking system has an associated state estimator and, depending on the application, the number of tracklets can be low or high. Therefore, the selection of the state estimator should take the computational power availability into account.

This solution was implemented on a RoboCup Middle-Size League agent integrator and tested during the RoboCup 2016 competitions, with the results shown for the last match in that year's World Championship competition in Leipzig, Germany and it is still the method currently in use to track obstacles by CAMBADA.

The results show that a high detection rate can be achieved, with some room for improvement concerning the false-positive rate, and that merging obstacle observations from multiple robots improve the position accuracy when compared to the individual observations. Maximizing both precision and recall should be a priority while minimizing the false-positive rate. However, there is always a trade-off between these two metrics, since it is always possible to be less conservative in the heuristics used to validate obstacles and tracklets, but not without sacrificing the false-positive rate. The solutions presented in this thesis were specially designed for applications with real-time constraints and sub-optimal communication conditions, which proved to perform consistently under a real-world scenario within this application specifications.

In the considered scenario for the conducted experiment, cycles of up to $20ms$ are used to perceive, plan and execute an action. The high efficiency of this method is proved by showing that it is able to perform along with all the remaining tasks required by the robot to function (although those other tasks are beyond the scope of this work).

Because real data collection is performed by both teams on field and shared at the end of the competition, in the future it should be possible to use the opponent position information matched with our perception to run an optimization procedure to fine-tune the zones distances parameters that were manually set. Eventually, Machine Learning techniques could also be applied to avoid fixed zones definition, at the cost of having a more opaque implementation with small room for manual tuning during the competition.

6.3 Leader Election

After discussing the original Raft approach to achieve consensus on networked machine clusters, two main limitations have been identified - an open-issue regarding the corner-case when there are only 1 or 2 active nodes and also the lack of prioritisation among agents to become a leader. A solution that is based on the Raft leader election protocol has been proposed, described and successfully implemented to overcome the two aforementioned limitations.

An experimental setup has been created to test our leader election solution, by continuously forcing a new election by killing the running leader agent. The results obtained are in line with the timings set for the asynchronous activity of this mechanism, set accordingly to the requirements of the application - in this case, a robotic soccer team.

In a nutshell, the proposed leader election solution is suitable to select a leader among the team agents, as it accounts for the possibility of having a preferred leader agent, providing a fault-tolerant and reliable redundancy mechanism whenever the leader becomes inactive.

6.4 CR7 Setplay Engine

Taking advantage of all the previous devised work on perception, multi-object tracking and also the leader election mechanism, a setplay engine was developed and implemented into the tested application.

Taking a previous framework by FCPortugal as a baseline, the CR7 Setplay Engine uses even higher-level instructions to allow for a much easier, user-friendly and reliable configuration. The end-user can use a GUI to configure a setplay: essentially a sequence of steps and select actions (wait, pass, move, etc.) for the robots based on these steps. The team strategy is defined as a set of setplays that can be configured prior to the robotic soccer matches.

The CR7 Manager (automatically assigned using the leader election mechanism) is then responsible for assessing the current game situation in real-time and deciding whether it is possible to initiate a setplay or abort a currently running setplay. This is based on the actions defined for the players - for example, a setplay can be aborted due to invalid pass-line between two players. The pass-line evaluation is performed by the manager using a unified representation of the obstacles using the distributed multi-object tracking solution presented.

6.5 Final Considerations

In a bottom-up approach, this project has incrementally touched a wide range of areas. Starting on the individual robotic agent perception regarding its own state estimation and also the state estimation of other assets of interest around it, including the ball and other robots. Following, the main focus has shifted to the MAS component of the benchmark application (a multi-agent robotics soccer team), with a new proposed distributed sensing technique and an efficient leader election algorithm based on Raft methodologies, providing an hybrid way to achieve consensus - decisions are centralized in the leader, but the leader election process (whenever it needs to take place) is distributed. Finally, all the previous work was brought together as foundations for the proposed CR7 setplay engine, a strategic planner for the MSL that focused particularly on both usability and flexibility.

From low-level development of new integration modules, that apply distributed sensor fusion techniques to enhance the global perception of the team, to high-level strategy improvements, the solutions proposed in this thesis were implemented and evaluated in a highly-dynamic, stochastic and partially observable environment provided by the MSL RoboCup robotic soccer competitions. When comparing to the original project objectives, it is possible to say that they have been successfully achieved.

6.6 List of Contributions

Publications

Multi-Robot Fast-Paced Coordination With Leader Election
Ricardo Dias, B. Cunha, J. L. Azevedo, A. Pereira, N. Lau
RoboCup Symposium, Canada, 2018

Real-time multi-object tracking on highly dynamic environments
Ricardo Dias, B. Cunha, E. Sousa, J. L. Azevedo, J. Silva, F. Amaral, N. Lau
IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC), Coimbra, 2017

Multi-object tracking with distributed sensing
Ricardo Dias, N. Lau, J. Silva, G. H. Lim
IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI), Baden-Baden, 2016

Improving the Kicking Accuracy in a Soccer Robot
Ricardo Dias, J. Silva, J. L. Azevedo, B. Cunha, A. Neves, N. Lau
ACM Symposium on Applied Computing (SAC), Salamanca, 2015

A New Approach for Dynamic Strategic Positioning in RoboCup Middle-Size League
Antonio J.R. Neves, F. Amaral, Ricardo Dias, J. Silva, N. Lau
17th Portuguese Conference on Artificial Intelligence (EPIA), Lisbon, 2015

Content distribution emulation for vehicular networks
Goncalo Pessoa, Ricardo Dias, T. Condeixa, J. Azevedo, L. Guardalben, S. Sargento
Wireless Days (WD), Porto, 2017

Other Contributions / Dissemination

- **Nov 2018** – Organizing Comittee of the VIII Intl. MSL Workshop (Aveiro, Portugal)
- **Oct 2018** – Demo in TechDays 2018 (Aveiro, Portugal)
- **Jul 2018** – Demo in FCT Ciência 2018 (Lisbon, Portugal)
- **Mar 2018** – Presentation for MAP-i Doctoral Programme (IEETA)
- **Nov 2017** - Presentation in IEETA Symposium
- **Nov 2017** – Presentation in the VII Intl. MSL Workshop (Eindhoven, Netherlands)
- **Nov 2017** – Presentation in Data Science Portugal 18th Meetup (Altice Labs, Aveiro)
- **Oct 2017** – Demo in TechDays 2017 (Aveiro, Portugal)
- **Jul 2017** – Presentation in MSL Scientific Challenge RoboCup 2017
- **Jul 2017** – Activity Monitor for "Academia de Verão UA"
- **Jul 2017** - Demo and Presentation in FCT Ciência 2017 (Lisbon, Portugal)
- **Jun 2017** – Presentation at Students@DETI
- **Nov 2016** – Presentation at the VI Intl. MSL Workshop (Kassel, Germany)
- **Nov 2016** – Demo and Presentation at "Jornadas da Informática" (UBI, Covilhã)
- **Oct 2016** – Demo and Presentation at "ENEI" 2016 (Aveiro)
- **Oct 2016** – Demo in TechDays 2016 (Aveiro, Portugal)
- **Jul 2016** – Activity Monitor for "Academia de Verão UA"
- **Jun 2016** – Presentation in Scientific Challenge RoboCup 2016
- Other demos and presentations on IRIS-Lab/IEETA/DETI/UA for UA students, high-school students, companies and other institutional visitors
- Collaboration in supervising some Master Thesis and other DETI student projects

Bibliography

Farshid Abbasi, Afshin Mesbahi, and Javad Mohammadpour Velni. A team-based approach for coverage control of moving sensor networks. *Automatica*, 81:342 – 349, 2017. ISSN 0005-1098. doi: 10.1016/j.automatica.2017.04.019.

Aamir Ahmad and Heinrich H. Bülthoff. Moving-horizon nonlinear least squares-based multirobot cooperative perception. *Robotics and Autonomous Systems*, 83:275 – 286, 2016. ISSN 0921-8890. doi: 10.1016/j.robot.2016.06.002.

Aamir Ahmad and Pedro Lima. Multi-robot cooperative spherical-object tracking in 3D space based on particle filters. *Robotics and Autonomous Systems*, 61(10):1084 – 1093, 2013. ISSN 0921-8890. doi: 10.1016/j.robot.2012.12.008.

Hidehisa Akiyama and Itsuki Noda. Multi-agent Positioning Mechanism in the Dynamic Environment. In Ubbo Visser, Fernando Ribeiro, Takeshi Ohashi, and Frank Dellaert, editors, *RoboCup 2007: Robot Soccer World Cup XI*, volume 5001, pages 377–384, Berlin, Heidelberg, 2008. Springer Heidelberg. ISBN 978-3-540-68847-1. doi: 10.1007/978-3-540-68847-1_38.

F. Almeida, N. Lau, and L. P. Reis. A Survey on Coordination Methodologies for Simulated Robotic Soccer Teams. In Olivier Boissier, Amal El Fallah-seghrouchni, Salima Hassas, and Nicolas Maudet, editors, *CEUR Workshop Proceedings*, volume 627 of *CEUR Workshop Proceedings*, 2010.

L. Almeida, F. Santos, T. Facchinetti, P. Pedreiras, V. Silva, and L. S. Lopes. Coordinating distributed autonomous agents with a real-time database: The CAMBADA project. In *Proc. of the 19th International Symposium on Computer and Information Sciences, ISCIS 2004*, volume 3280 of *Lecture Notes in Computer Science*, pages 878–886. Springer, 2004. doi: 10.1007/978-3-540-30182-0_88.

Y. Bar-Shalom and K. Birmiwal. Variable dimension filter for maneuvering target tracking. *IEEE Transactions on Aerospace and Electronic Systems*, AES-18(5):621–629, Sep. 1982. doi: 10.1109/TAES.1982.309274.

Y. Bar-Shalom, S. Challa, and H. A. P. Blom. IMM estimator versus optimal estimator for hybrid systems. *IEEE Transactions on Aerospace and Electronic Systems*, 41(3):986–991, July 2005. ISSN 0018-9251. doi: 10.1109/TAES.2005.1541443.

Yaakov Bar-Shalom, Thiagalingam Kirubarajan, and X.-Rong Li. *Estimation with Applications to Tracking and Navigation*. John Wiley & Sons, Inc., New York, USA, 2002. ISBN 0471221279.

Giorgio Battistelli and Luigi Chisci. Stability of consensus extended kalman filter for distributed state estimation. *Automatica*, 68:169 – 178, 2016. ISSN 0005-1098. doi: 10.1016/j.automatica.2016.01.071.

Lewis A. Binns, Dimitris Valachis, Sean Anderson, David W. Gough, David Nicholson, and Phil Greenway. Distributed SLAM. In *Proc. of the XI Conference on Signal Processing, Sensor Fusion and Target Recognition*, volume 4729, pages 62–68, July 2002. doi: 10.1117/12.477628.

Joydeep Biswas, Juan Pablo Mendoza, Danny Zhu, Benjamin Choi, Steven Klee, and Manuela Veloso. Opponent-driven planning and execution for pass, attack, and defense in a multi-robot soccer team. In *Proc. of the 2014 International Conference on Autonomous Agents and Multi-agent Systems (AAMAS '14)*, pages 493–500, January 2014. ISBN 978-1-4503-2738-1.

Rodney A. Brooks. A Robot That Walks; Emergent Behaviors from a Carefully Evolved Network. *Neural Comput.*, 1(2):253–262, June 1989. ISSN 0899-7667. doi: 10.1162/neco.1989.1.2.253.

G. R. Brown and R. B. Donald. Mobile Robot Self-Localization without Explicit Landmarks. *Algorithmica*, 26(3):515–559, 2000. ISSN 1432-0541. doi: 10.1007/s004539910023.

Luiz Chaimowicz, Vijay Kumar, and Mario F. M. Campos. A Paradigm for Dynamic Coordination of Multiple Robots. *Autonomous Robots*, 17(1):7–21, 2004. ISSN 1573-7527. doi: 10.1023/B:AURO.0000032935.30271.a5.

Tushar D. Chandra, Robert Griesemer, and Joshua Redstone. Paxos made live: An engineering perspective. In *Proceedings of the Twenty-sixth Annual ACM Symposium on Principles of Distributed Computing*, PODC '07, pages 398–407, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-616-5. doi: 10.1145/1281100.1281103.

Tzung-Shi Chen, Jen-Jee Chen, and Cheng-Han Wu. Distributed object tracking using moving trajectories in wireless sensor networks. *Wireless Networks*, 22(7):2415–2437, Oct 2016. ISSN 1572-8196. doi: 10.1007/s11276-015-1107-9.

Philip R. Cohen and Hector J. Levesque. Teamwork. *Special Issue on Cognitive Science and Artificial Intelligence*, 25(4):487–512, September 1991.

João Cravo, Fernando Almeida, Pedro Henriques Abreu, Luís Paulo Reis, Nuno Lau, and Luís Mota. Strategy planner: Graphical definition of soccer set-plays. *Data & Knowledge Engineering*, 94:110 – 131, 2014. ISSN 0169-023X. doi: 10.1016/j.datak.2014.10.001.

Bernardo Cunha, José Azevedo, Nuno Lau, and Luis Almeida. Obtaining the inverse distance map from a non-svp hyperbolic catadioptric robotic vision system. In Ubbo Visser, Fernando Ribeiro, Takeshi Ohashi, and Frank Dellaert, editors, *RoboCup 2007: Robot Soccer World Cup XI*, volume 5001 of *Lecture Notes in Computer Science*, pages 417–424, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. ISBN 978-3-540-68847-1. doi: 10.1007/978-3-540-68847-1.

A. Cunningham, M. Paluri, and F. Dellaert. DDF-SAM: Fully distributed SLAM using Constrained Factor Graphs. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 3025–3030, Oct 2010. doi: 10.1109/IROS.2010.5652875.

N. Cunningham, J.E. Griffin, and D.L. Wild. ParticleMDI: particle Monte Carlo methods for the cluster analysis of multiple datasets with applications to cancer subtype identification. *Advances in Data Analysis and Classification*, 14:463–484, Jun 2020. doi: 10.1007/s11634-020-00401-y.

HesamAddin Torabi Dashti, Nima Aghaeepour, Sahar Asadi, Meysam Bastani, Zahra Delafkar, Fatemeh Miri Disfani, Serveh Mam Ghaderi, Shahin Kamali, Sepideh Pashami, and Alireza Fotuhi Siahpirani. Dynamic Positioning Based on Voronoi Cells (DPVC). In *RoboCup 2005: Robot Soccer World Cup IX*, pages 219–229. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006. ISBN 978-3-540-35438-3. doi: 10.1007/11780519_20.

Randall Davis and Reid G. Smith. Negotiation as a metaphor for distributed problem solving. *Artificial Intelligence*, 20(1):63–109, 1983. ISSN 0004-3702. doi: 10.1016/0004-3702(83)90015-2.

Harmen de Weerd, Rineke Verbrugge, and Bart Verheij. Negotiating with other minds: the role of recursive theory of mind in negotiation with incomplete information. *Autonomous Agents and Multi-Agent Systems*, 31(2):250–287, Mar 2017. ISSN 1573-7454. doi: 10.1007/s10458-015-9317-1.

Pierre Del Moral. Nonlinear Filtering: Interacting Particle Solution. *Markov Processes and Related Fields*, 2(4):555–580, 1996.

R. Dias, N. Lau, J. Silva, and G. H. Lim. Multi-object tracking with distributed sensing. In *2016 IEEE International Conference on Multisensor Fusion and Integration for Intelligent*

Systems (MFI), pages 564–569, Baden-Baden, Germany, Sept 2016. doi: 10.1109/MFI.2016.7849548.

Ricardo Dias. Agent architecture of the cambada robotics soccer team. Master's thesis, University of Aveiro, 2014.

Arnaud Doucet, Nando de Freitas, and Neil Gordon. An introduction to sequential monte carlo methods. In Arnaud Doucet, Nando de Freitas, and Neil Gordon, editors, *Sequential Monte Carlo Methods in Practice*, pages 3–14, New York, NY, 2001. Springer New York. ISBN 978-1-4757-3437-9. doi: 10.1007/978-1-4757-3437-9_1.

Edmund H Durfee. Practically coordinating. *AI Magazine*, 20(1):99–116, 1999.

Wilfried Elmenreich. *Sensor Fusion in Time-Triggered Systems*. PhD thesis, Institut fur Technische Informatik, Vienna, Austria, 2002.

A. T. Erdem and A. Ö. Ercan. Fusing inertial sensor data in an extended kalman filter for 3d camera tracking. *IEEE Transactions on Image Processing*, 24(2):538–548, 2015.

G. M. Farouk, I. F. Moawad, and M. M. Aref. A machine learning based system for mostly automating opponent modeling in real-time strategy games. In *12th International Conference on Computer Engineering and Systems (ICCES)*, pages 337–346, December 2017. doi: 10.1109/ICCES.2017.8275329.

Nuno M. Figueiredo, António J. R. Neves, Nuno Lau, Artur Pereira, and Gustavo Corrente. Control and monitoring of a robotic soccer team: The base station application. In Luís Seabra Lopes, Nuno Lau, Pedro Mariano, and Luís M. Rocha, editors, *Progress in Artificial Intelligence*, pages 299–309, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. ISBN 978-3-642-04686-5. doi: 10.1007/978-3-642-04686-5_25.

Frans C. A. Groen, Matthijs T. J. Spaan, Jelle R. Kok, and Gregor Pavlin. Real world multi-agent systems: Information sharing, coordination and planning. In Balder D. ten Cate and Henk W. Zeevat, editors, *Logic, Language, and Computation*, pages 154–165, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. ISBN 978-3-540-75144-1.

Barbara J. Grosz and Sarit Kraus. Collaborative plans for complex group action. *Artificial Intelligence*, 86(2):269–357, 1996. ISSN 0004-3702. doi: 10.1016/0004-3702(95)00103-4.

Barbara J Grosz, Luke Hunsberger, and Sarit Kraus. Planning and acting together. *AI magazine*, 20(4):23, 1999. doi: 10.1609/aimag.v20i4.1476.

Jens-Steffen Gutmann, Wolfgang Hatzack, Immanuel Herrmann, Bernhard Nebel, Frank Rittinger, Augustinus Topor, and Thilo Weigel. Reliable Self-Localization, Multirobot Sensor Integration, Accurate Path-Planning and Basic Soccer Skills: Playing an Effective Game of

Robotic Soccer. In *Proceedings of the Ninth International Conference on Advanced Robotics*, pages 289–296, 1999.

J. L. Hsieh and C. T. Sun. Building a player strategy model by analyzing replays of real-time strategy games. In *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, pages 3106–3111, June 2008. doi: 10.1109/IJCNN.2008.4634237.

J. Ilonen, J. Bohg, and V. Kyrki. Fusing visual and tactile sensing for 3-d object reconstruction while grasping. In *2013 IEEE International Conference on Robotics and Automation*, pages 3547–3554, 2013.

Michael Isik, Freek Stulp, Gerd Mayer, and Hans Utz. Coordination without negotiation in teams of heterogeneous robots. In Gerhard Lakemeyer, Elizabeth Sklar, Domenico G. Sorrenti, and Tomoichi Takahashi, editors, *RoboCup 2006: Robot Soccer World Cup X*, volume 4434 of *LNAI*, pages 355–362, Berlin, Heidelberg, 2007. Springer. ISBN 978-3-540-74024-7. doi: 10.1007/978-3-540-74024-7_33.

Nick R. Jennings. Commitments and conventions: The foundation of coordination in multi-agent systems. *The Knowledge Engineering Review*, 8(3):223–250, September 1993.

S. J. Julier and J. K. Uhlmann. Unscented filtering and nonlinear estimation. *Proceedings of the IEEE*, 92(3):401–422, Mar 2004. ISSN 0018-9219. doi: 10.1109/JPROC.2003.823141.

Simon J Julier and Jeffrey K Uhlmann. New extension of the kalman filter to nonlinear systems. In *Signal processing, sensor fusion, and target recognition VI*, volume 3068, pages 182–193. International Society for Optics and Photonics, 1997.

Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Journal of Fluids Engineering*, 82(1):35–45, 1960.

A. T. Kamal, J. H. Bappy, J. A. Farrell, and A. K. Roy-Chowdhury. Distributed multi-target tracking and data association in vision networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(7):1397–1410, July 2016. ISSN 0162-8828. doi: 10.1109/TPAMI.2015.2484339.

Yiannis Kantaros, Michalis Thanou, and Anthony Tzes. Distributed coverage control for concave areas by a heterogeneous robot–swarm with visibility sensing constraints. *Automatica*, 53:195 – 207, 2015. ISSN 0005-1098. doi: 10.1016/j.automatica.2014.12.034.

Hyung-Bok Kim and Kwee-Bo Sim. A particular object tracking in an environment of multiple moving objects. In *Control Automation and Systems (ICCAS), 2010 International Conference on*, pages 1053–1056, Oct 2010.

Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, and Eiichi Osawa. Robocup: The robot world cup initiative. In *The First International Conference on Autonomous Agent (Agents-97)*, pages 340–347, 1997. doi: 10.1.1.50.5425.

Jelle R. Kok, Matthijs T.J. Spaan, and Nikos Vlassis. Non-communicative multi-robot coordination in dynamic environments. *Robotics and Autonomous Systems*, 50(2-3):99–114, 2005. ISSN 0921-8890. doi: 10.1016/j.robot.2004.08.003. Multi-Robots in Dynamic Environments.

H. W. Kuhn and Bryn Yaw. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, pages 83–97, 1955.

Leslie Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, May 1998. ISSN 0734-2071. doi: 10.1145/279227.279229.

Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, July 1982. ISSN 0164-0925. doi: 10.1145/357172.357176.

T.D. Larsen, K.L. Hansen, N.A. Andersen, and O. Ravn. Design of Kalman filters for mobile robots; evaluation of the kinematic and odometric approach. In *Control Applications, 1999. Proceedings of the 1999 IEEE International Conference on*, volume 2, pages 1021–1026, 1999. doi: 10.1109/CCA.1999.801027.

Nuno Lau, Luis Seabra Lopes, and Gustavo Corrente. CAMBADA: information sharing and team coordination. In *Proc. of the 8th Conference on Autonomous Robot Systems and Competitions, Portuguese Robotics Open - ROBOTICA 2008*, pages 27–32, Aveiro, Portugal, April 2008.

Nuno Lau, Luis Seabra Lopes, Gustavo Corrente, Nelson Filipe, and Ricardo Sequeira. Robot team coordination using dynamic role and positioning assignment and role based setplays. *Mechatronics*, 21(2):445 – 454, 2011. ISSN 0957-4158. doi: 10.1016/j.mechatronics.2010.05.010. Special Issue on Advances in intelligent robot design for the Robocup Middle Size League.

Martin Lauer, Sascha Lange, and Martin Riedmiller. Calculating the Perfect Match: An Efficient and Accurate Approach for Robot Self-localization. In Ansgar Bredenfeld, Adam Jacoff, Itsuki Noda, and Yasutake Takahashi, editors, *RoboCup 2005: Robot Soccer World Cup IX*, pages 142–153, Berlin, Heidelberg, 07 2006. Springer Berlin Heidelberg. ISBN 978-3-540-35438-3. doi: 10.1007/11780519_13.

Hector J Levesque, Philip R Cohen, and José HT Nunes. On acting together. In *AI Magazine*, volume 90, pages 94–99, June 1990.

L. Merino, F. Caballero, J.R.M.-d. Dios, and A. Ollero. Cooperative Fire Detection using Unmanned Aerial Vehicles. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 1884–1889, April 2005. doi: 10.1109/ROBOT.2005.1570388.

Nicholas Metropolis and Stanislaw Ulam. The Monte Carlo Method. *Journal of the American Statistical Association*, 44(247):335–341, 1949.

K. Mironov. Transport by robotic throwing and catching: Accurate stereo tracking of the spherical object. In *2017 International Conference on Industrial Engineering, Applications and Manufacturing (ICIEAM)*, pages 1–6, May 2017. doi: 10.1109/ICIEAM.2017.8076490.

Eduardo Montijano and Carlos Sagüés. *Robotic Networks and the Consensus Problem*, pages 9–19. Springer International Publishing, Cham, 2015. ISBN 978-3-319-15699-6. doi: 10.1007/978-3-319-15699-6_2.

L. Moreno, J. M. Armingol, A. De La Escalera, and M. A. Salichs. Global integration of ultrasonic sensors information in mobile robot localization. In *Proc. of the Ninth International Conference on Advanced Robotics*, pages 283–288, 1999.

Luís Mota, Luís Paulo Reis, and Nuno Lau. Multi-robot coordination using setplays in the middle-size and simulation leagues. *Mechatronics*, 21(2):434 – 444, 2011. ISSN 0957-4158. doi: 10.1016/j.mechatronics.2010.05.005. Special Issue on Advances in intelligent robot design for the Robocup Middle Size League.

MSL Technical Committee. Data structure for World Model sharing in MSL. Technical report, RoboCup, Mar 2016.

A. Neves, J. Azevedo, N. Lau B. Cunha, J. Silva, F. Santos, G. Corrente, D. A. Martins, N. Figueiredo, A. Pereira, L. Almeida, L. S. Lopes, and P. Pedreiras. CAMBADA soccer team: from robot architecture to multiagent coordination. In *Robot Soccer*, pages 19–45, Vienna, Austria, January 2010. I-Tech Education and Publishing.

A. J. R. Neves, A. Trifan, P. Dias, and J. L. Azevedo. Detection of aerial balls in robotic soccer using a mixture of color and depth information. In *2015 IEEE International Conference on Autonomous Robot Systems and Competitions*, pages 227–232, 2015. doi: 10.1109/ICARSC.2015.13.

António J. R. Neves, Filipe Amaral, Ricardo Dias, João Silva, and Nuno Lau. A New Approach for Dynamic Strategic Positioning in RoboCup Middle-Size League. In Francisco Pereira, Penousal Machado, Ernesto Costa, and Amílcar Cardoso, editors, *Progress in Artificial Intelligence*, volume 9273 of *Lecture Notes in Computer Science*, pages 433–444. Springer International Publishing, 2015. ISBN 978-3-319-23484-7. doi: 10.1007/978-3-319-23485-4_43.

Brian M. Oki and Barbara H. Liskov. Viewstamped replication: A new primary copy method to support highly-available distributed systems. In *Proceedings of the Seventh Annual ACM Symposium on Principles of Distributed Computing*, PODC '88, pages 8–17, New York, NY, USA, 1988. ACM. ISBN 0-89791-277-2. doi: 10.1145/62546.62549.

R. Olfati-Saber, J.A. Fax, and R.M. Murray. Consensus and Cooperation in Networked Multi-Agent Systems. *Proceedings of the IEEE*, 95(1):215–233, January 2007. ISSN 0018-9219. doi: 10.1109/JPROC.2006.887293.

Reza Olfati-Saber and Jeff S Shamma. Consensus filters for sensor networks and distributed sensor fusion. In *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC'05. 44th IEEE Conference on*, pages 6698–6703. IEEE, 2005.

Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*, USENIX ATC'14, pages 305–320, Berkeley, CA, USA, 2014. USENIX Association. ISBN 978-1-931971-10-2.

Sébastien Paquet, Nicolas Bernier, and Brahim Chaib-draa. Comparison of different coordination strategies for the RoboCup rescue simulation. In *Int'l Conf. on Industrial & Engineering Applications*, volume LNAI 3029, pages 987–996. Springer-Verlag, 2004.

L.E. Parker. ALLIANCE: an architecture for fault tolerant multirobot cooperation. *Robotics and Automation, IEEE Transactions on*, 14(2):220–240, April 1998. ISSN 1042-296X. doi: 10.1109/70.681242.

Paulo Pedreiras and Luis Almeida. Task management for soft real-time applications based on general purpose operating systems. In Pedro Lima, editor, *Robotic Soccer*, Rijeka, 2007. IntechOpen. doi: 10.5772/5134.

G. Petryk and M. Buehler. Robust Estimation of Pre-Contact Object Trajectories. In *Robot Control*, volume 2, pages 793–799, 1997.

Francisco Pinto. Ethernet gateway design for the cambada robots. Master's thesis, University of Aveiro, 2020.

Roland Potthast, Anne Walter, and Andreas Rhodin. A localized adaptive particle filter within an operational nwp framework. *Monthly Weather Review*, 147(1):345–362, 2019.

D. Reid. An algorithm for tracking multiple targets. *IEEE Transactions on Automatic Control*, 24(6):843–854, Dec 1979. ISSN 0018-9286. doi: 10.1109/TAC.1979.1102177.

L. P. Reis. *Coordination in Multi-Agent Systems: Applications in University Management and Robotic Soccer*. PhD thesis, Faculty of Engineering of the University of Porto, July 2003.

Luis Paulo Reis, Nuno Lau, and Eugenio Costa Oliveira. Situation Based Strategic Positioning for Coordinating a Team of Homogeneous Agents. In *Balancing Reactivity and Social Deliberation in Multi-Agent Systems*, volume 2103 of *Lecture Notes in Computer Science*, pages 175–197. Springer Berlin Heidelberg, 2001.

D. Rembold, U. Zimmermann, T. Langle, and Heinz Worn. Detection and handling of moving objects. In *Proceedings of the 24th Annual Conference of the IEEE Ind. Electron. Soc., IECON '98*, volume 3, pages 1332–1337, Aug 1998. doi: 10.1109/IECON.1998.722843.

J. M. Richardson and K. A Marsh. Fusion of Multisensor Data. *Int. J. Rob. Res.*, 7(6):78–96, December 1988. ISSN 0278-3649.

M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: the rprop algorithm. In *IEEE International Conference on Neural Networks*, pages 586–591 vol.1, March 1993. doi: 10.1109/ICNN.1993.298623.

Julio K. Rosenblatt. Utility Fusion: Map-Based Planning in a Behavior-Based System. In Alexander Zelinsky, editor, *Field and Service Robotics*, pages 411–418, London, 1998. Springer London. ISBN 978-1-4471-1273-0. doi: 10.1007/978-1-4471-1273-0_62.

Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson series in artificial intelligence. Pearson, 4th edition, 2020. ISBN 9780134610993.

Frederico Santos, Luis Almeida, Luis Seabra Lopes, José Luís Azevedo, and Manuel Bernardo Cunha. Communicating among robots in the robocup middle-size league. In Jacky Baltes, Michail G. Lagoudakis, Tadashi Naruse, and Saeed Shiry Ghidary, editors, *RoboCup 2009: Robot Soccer World Cup XIII*, pages 320–331, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. ISBN 978-3-642-11876-0.

J.Z. Sasiadek and P. Hartana. Sensor data fusion using Kalman filter. In *Proc. of the Third International Conference on Information Fusion*, volume 2, pages 19–25, July 2000. doi: 10.1109/IFIC.2000.859866.

R. Siegwart and I.R. Nourbakhsh. *Introduction to Autonomous Mobile Robots*. MIT Press, 2004. ISBN 9780262195027.

Diogo Silva. RTDB2 : a flexible distributed blackboard. Master's thesis, University of Aveiro, 2017.

João Silva, Nuno Lau, António J.R. Neves, João Rodrigues, and José Luís Azevedo. World modeling on an MSL robotic soccer team. *Mechatronics*, 21(2):411–422, 2011. ISSN 0957-4158. doi: 10.1016/j.mechatronics.2010.05.011. Special Issue on Advances in intelligent robot design for the Robocup Middle Size League.

Hendrik Skubch, Michael Wagner, and Roland Reichle. A Language for Interactive Cooperative Agents. ALICA is a behaviour specification language for teams of agents. It provides modeling elements to describe team behaviours and strategies from a global perspective., February 2009.

Matthijs T. J. Spaan and Frans C. A. Groen. Team Coordination among Robotic Soccer Players. In Gal A. Kaminka, Pedro U. Lima, and Raúl Rojas, editors, *RoboCup 2002: Robot Soccer World Cup VI*, pages 409–416, Berlin, Heidelberg, 2003a. Springer Berlin Heidelberg. ISBN 978-3-540-45135-8. doi: 10.1007/978-3-540-45135-8_36.

Matthijs T. J. Spaan and Frans C. A. Groen. Team Coordination among Robotic Soccer Players. In *RoboCup 2002: Robot Soccer World Cup VI*, volume 2752 of *Lecture Notes in Computer Science*, pages 409–416. Springer Berlin Heidelberg, 2003b. ISBN 978-3-540-40666-2. doi: 10.1007/978-3-540-45135-8_36.

Peter Stone. *Layered learning in multiagent systems: A winning approach to robotic soccer*. MIT Press, 2000.

Peter Stone and Manuela Veloso. Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork. *Artificial Intelligence*, 110(2): 241 – 273, 1999. ISSN 0004-3702. doi: 10.1016/S0004-3702(99)00025-9.

Ashley W Stroupe, Martin C Martin, and Tucker Balch. Distributed sensor fusion for object position estimation by multi-robot systems. In *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, volume 2, pages 1092–1098. IEEE, 2001.

F. Stulp, M. Isik, and M. Beetz. Implicit coordination in robotic teams using learned prediction models. In *Proc. of the 2006 IEEE International Conference on Robotics and Automation (ICRA 2006)*, pages 1330–1335, May 2006. doi: 10.1109/ROBOT.2006.1641893.

Shuli Sun, Honglei Lin, Jing Ma, and Xiuying Li. Multi-sensor distributed fusion estimation with applications in networked systems: A review paper. *Information Fusion*, 38:122 – 134, 2017. ISSN 1566-2535. doi: 10.1016/j.inffus.2017.03.006.

Sebastian Thrun. Probabilistic algorithms in robotics. *Ai Magazine*, 21(4):93–109, 2000.

Sebastian Thrun, Wolfram Burgard, and Dieter Fox. A Probabilistic Approach to Concurrent Mapping and Localization for Mobile Robots. *Machine Learning*, 31(1):29–53, April 1998. ISSN 1573-0565. doi: 10.1023/A:1007436523611.

Robbert Van Renesse and Deniz Altinbuken. Paxos made moderately complex. *ACM Comput. Surv.*, 47(3):42:1–42:36, February 2015. ISSN 0360-0300. doi: 10.1145/2673577.

Kiri Wagstaff, Claire Cardie, Seth Rogers, and Stefan Schrödl. Constrained k-means clustering with background knowledge. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 577–584, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. ISBN 1-55860-778-1.

Eric A Wan and Rudolph Van Der Merwe. The unscented Kalman filter for nonlinear estimation. In *Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000. AS-SPCC. The IEEE 2000*, pages 153–158. IEEE, 2000.

Han Wang, Choon Seng Chua, and Ching Tong Sim. Real-time object tracking from corners. *Robotica*, 16:109–116, 1 1998. ISSN 1469-8668. URL `http://journals.cambridge.org/article_S0263574798000198`.

Michael Wooldridge. *An introduction to multiagent systems*. John Wiley & Sons, 2009.

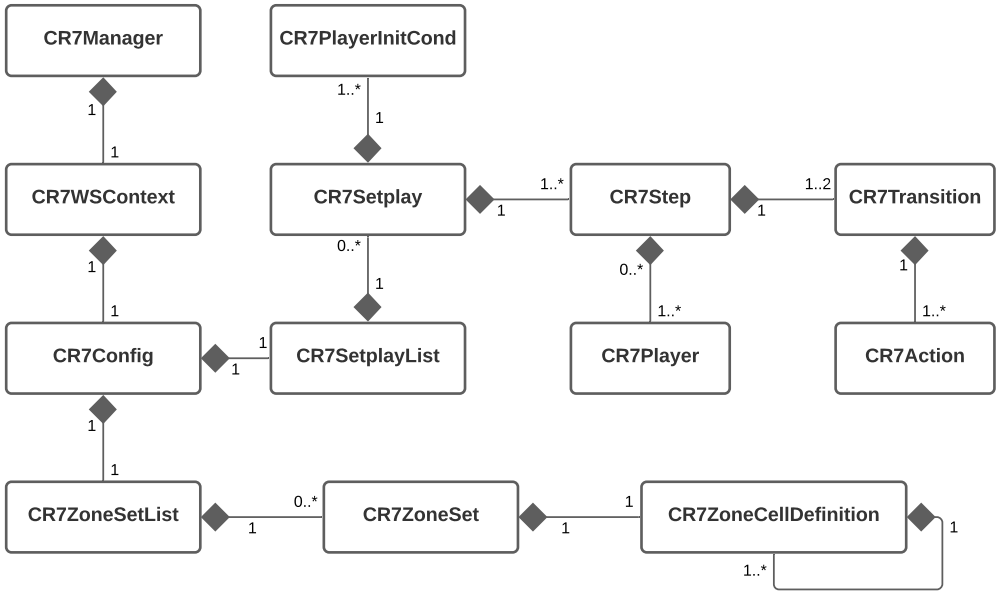
Xu Y., Jiang C., and Tan Y. SEU-3D 2006 soccer simulation team description. In *CD Proc. of RoboCup Symposium 2006*, Bremen, Germany, 2006.

Appendices

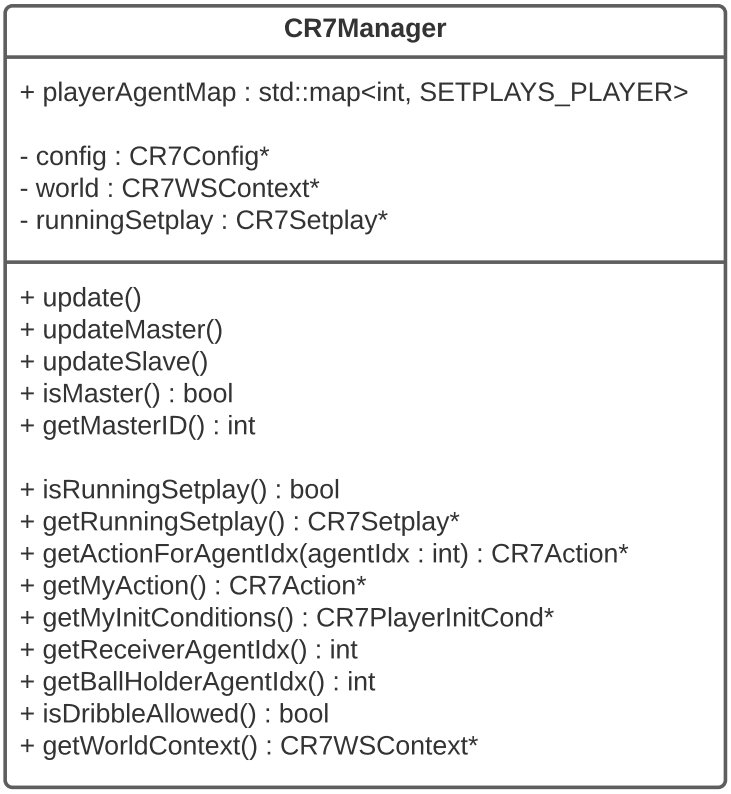
Appendix A

CR7 Documentation

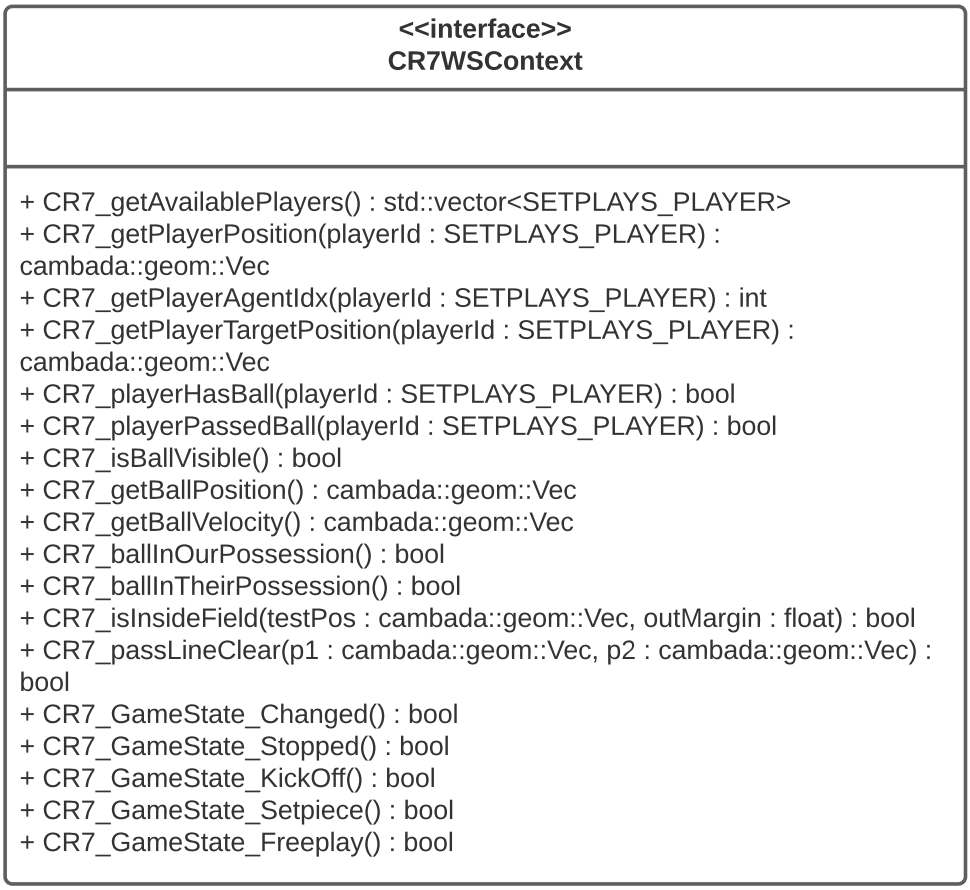
A.1 Class Diagram - Overview



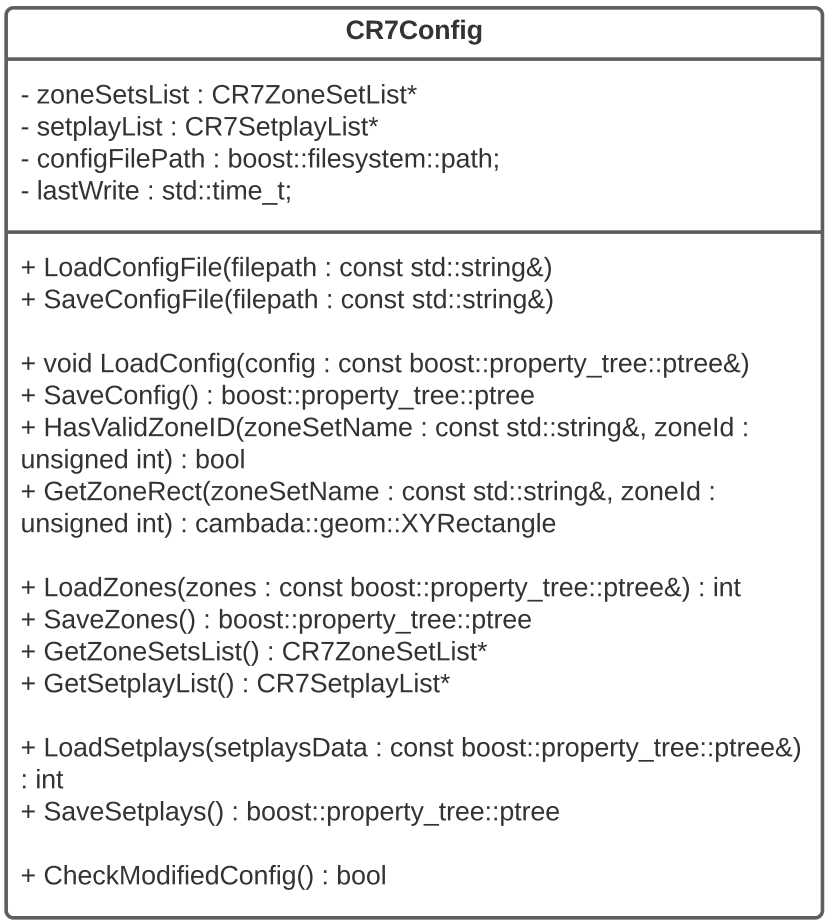
A.2 CR7Manager Class Details



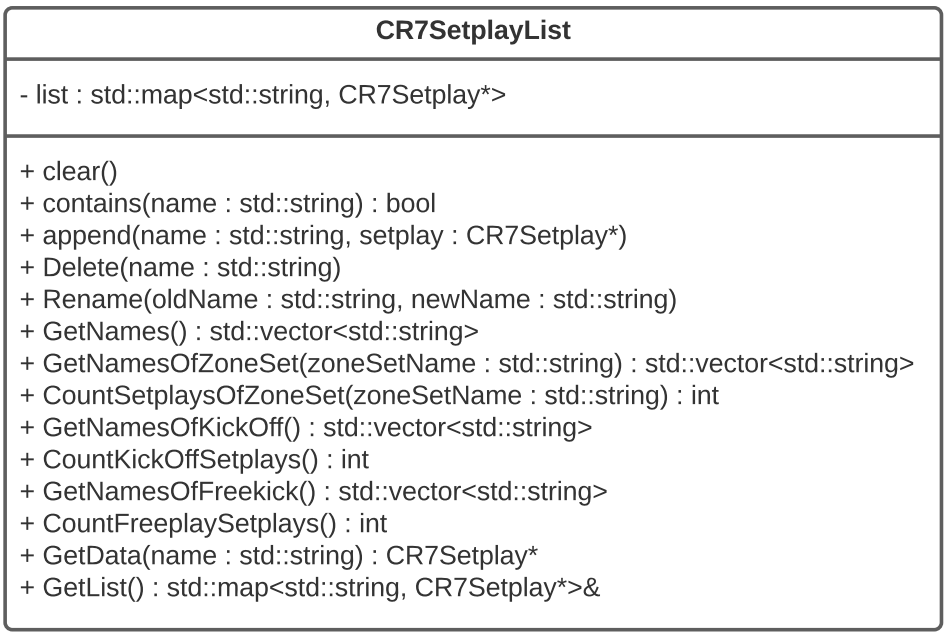
A.3 CR7WSContext Class Details



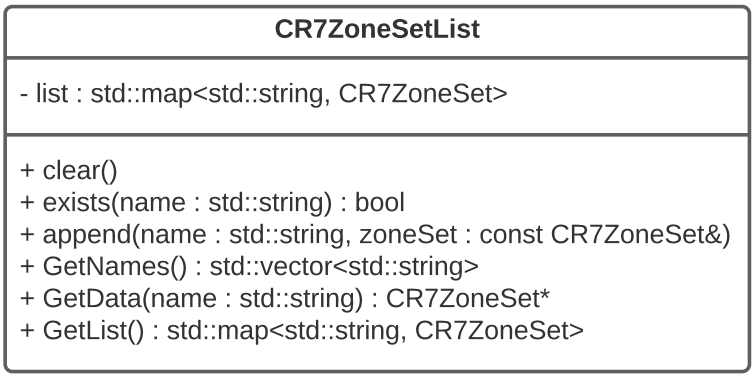
A.4 CR7Config Class Details



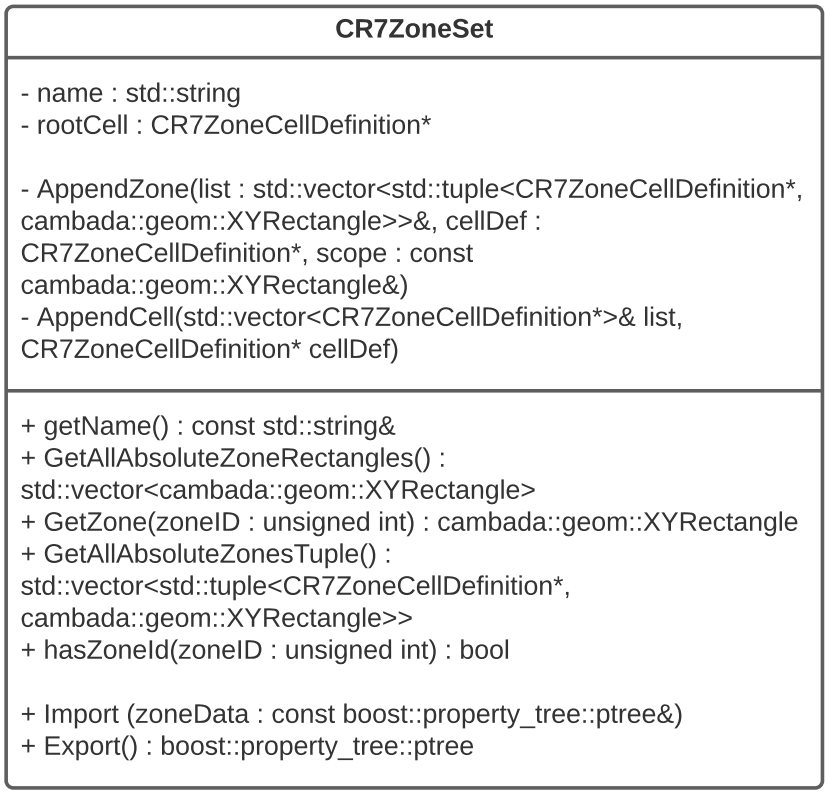
A.5 CR7SetplayList Class Details



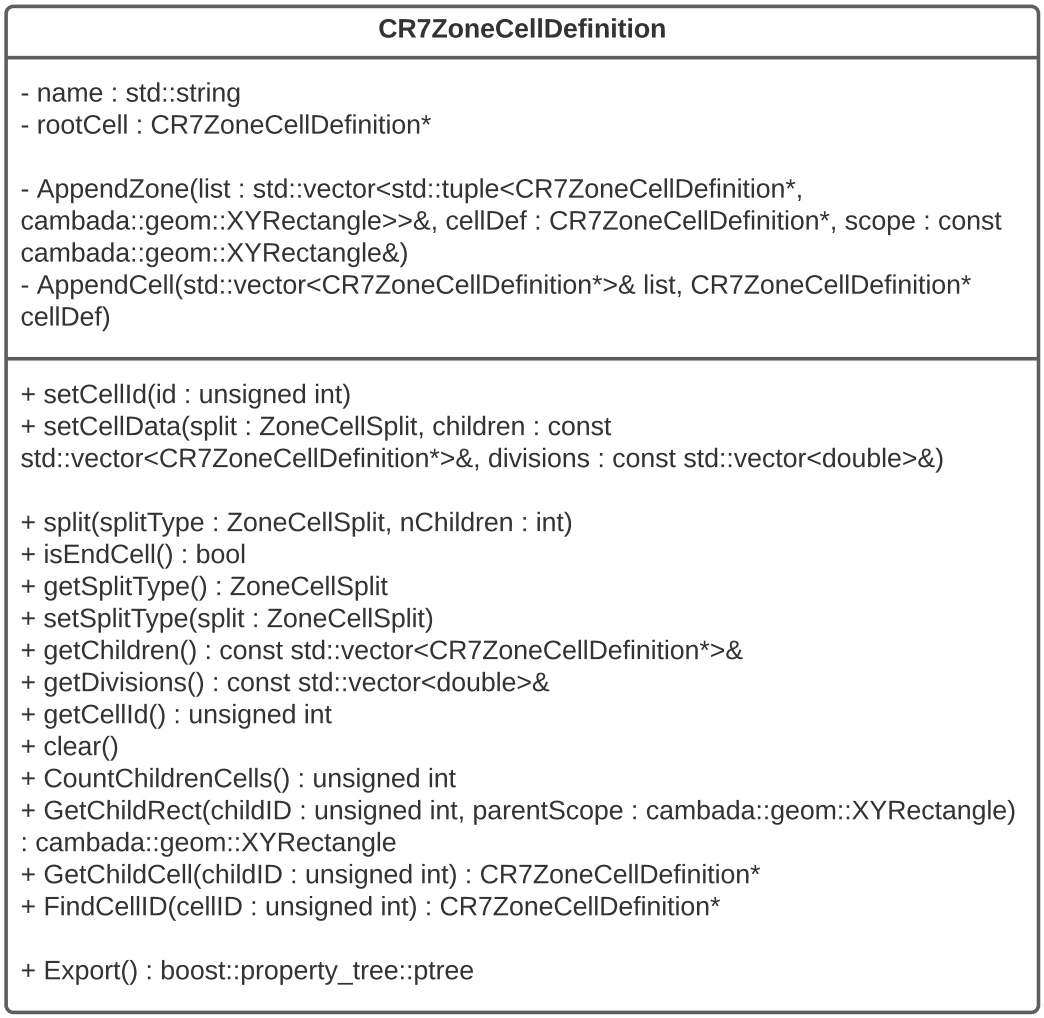
A.6 CR7ZoneSetList Class Details



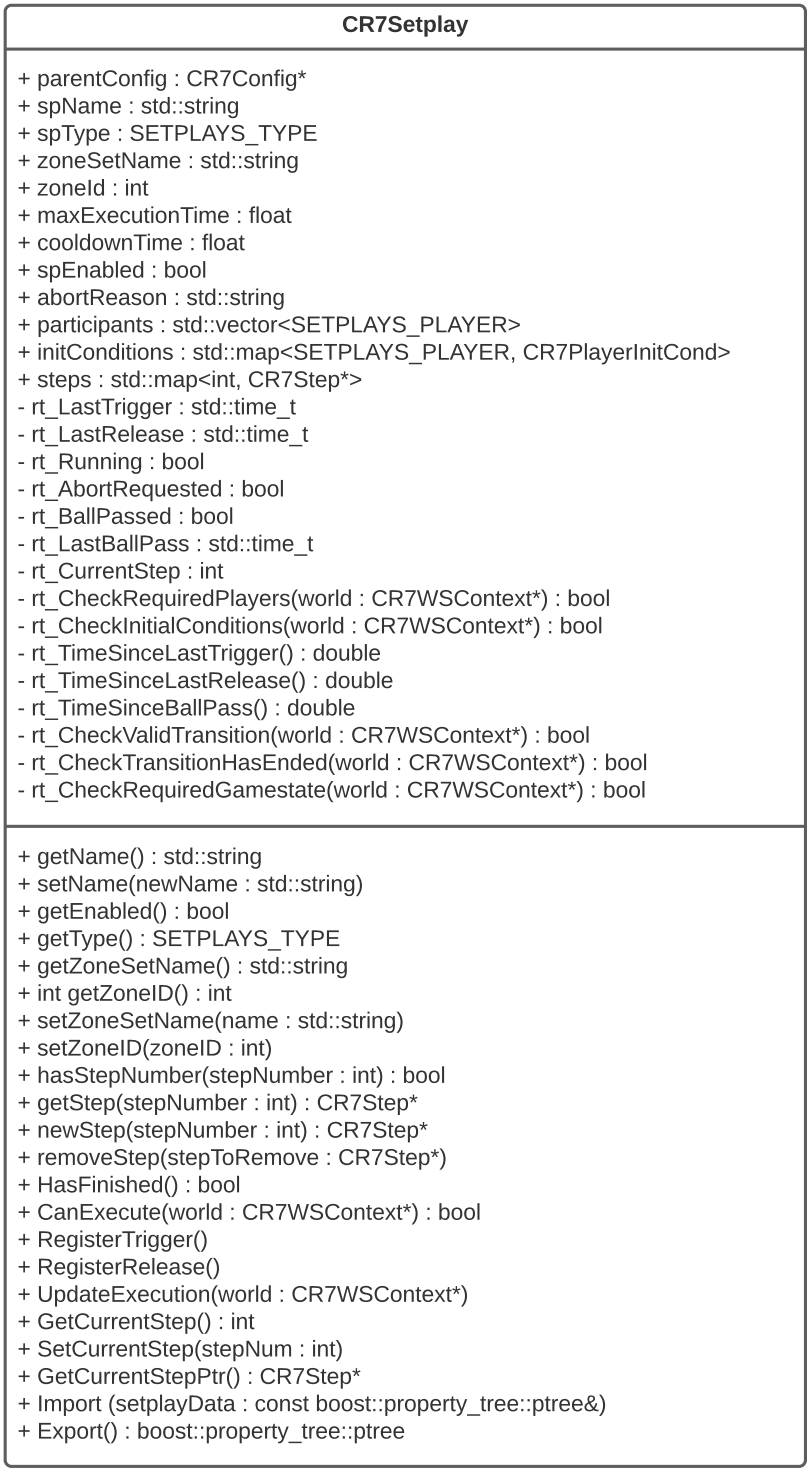
A.7 CR7ZoneSet Class Details



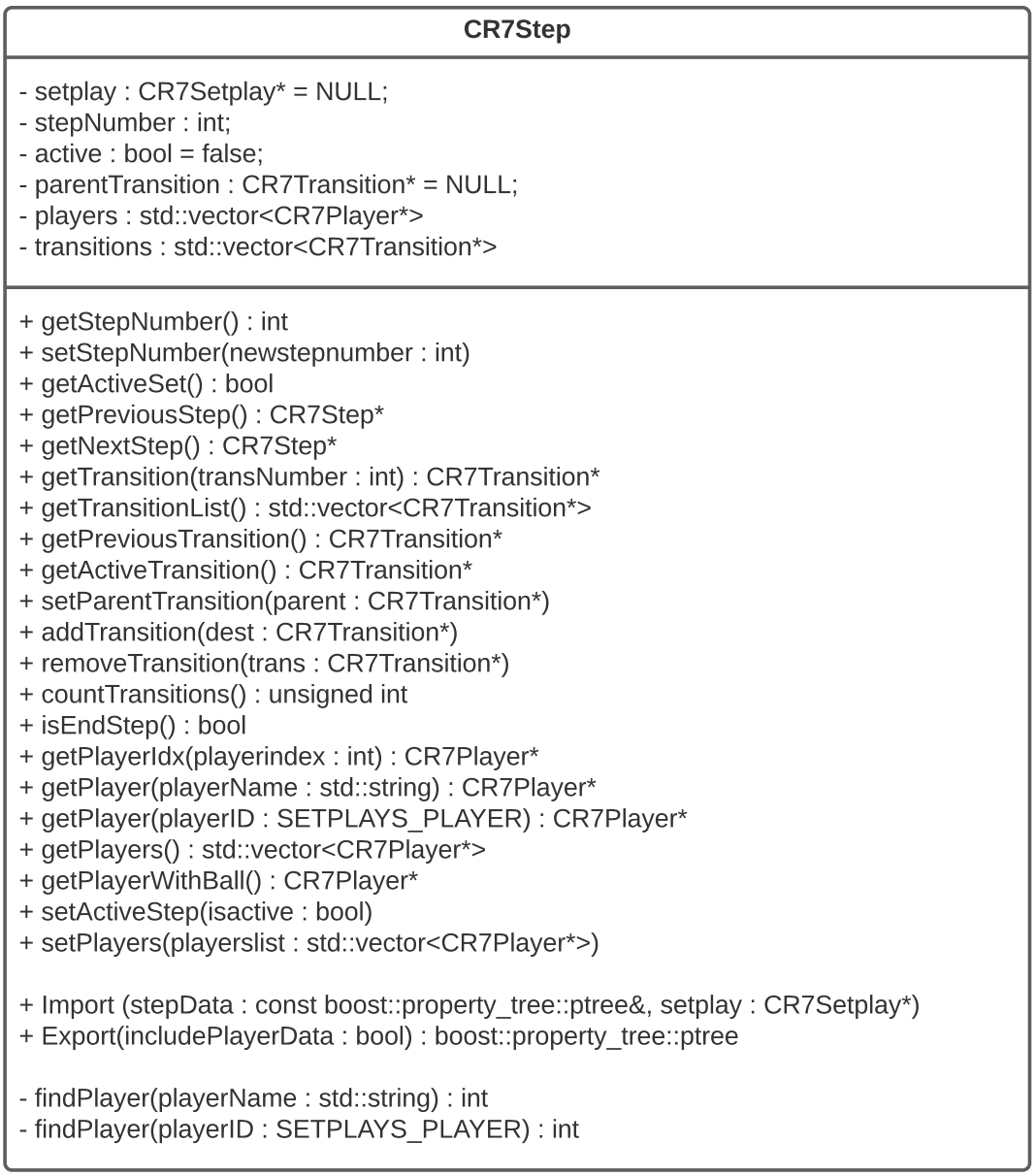
A.8 CR7ZoneCellDefinition Class Details



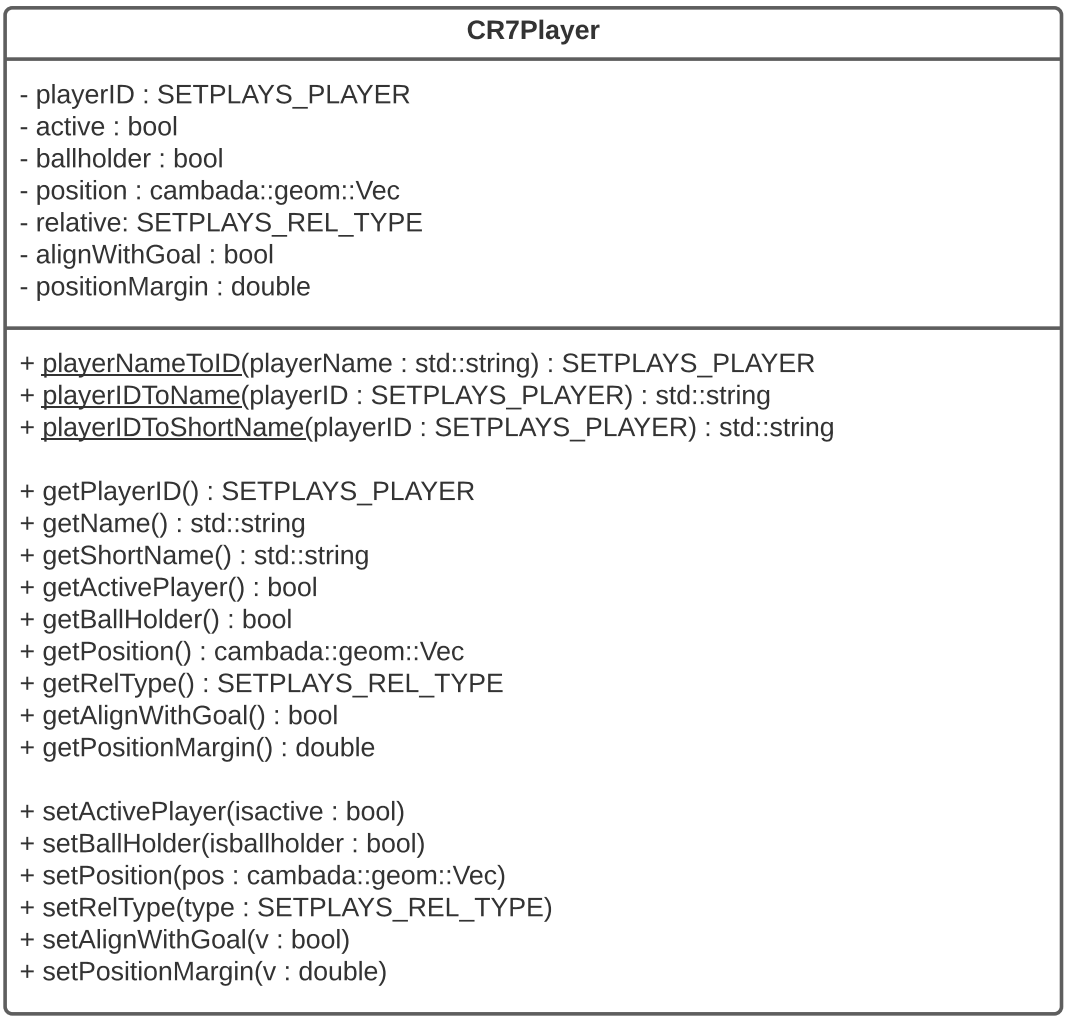
A.9 CR7Setplay Class Details



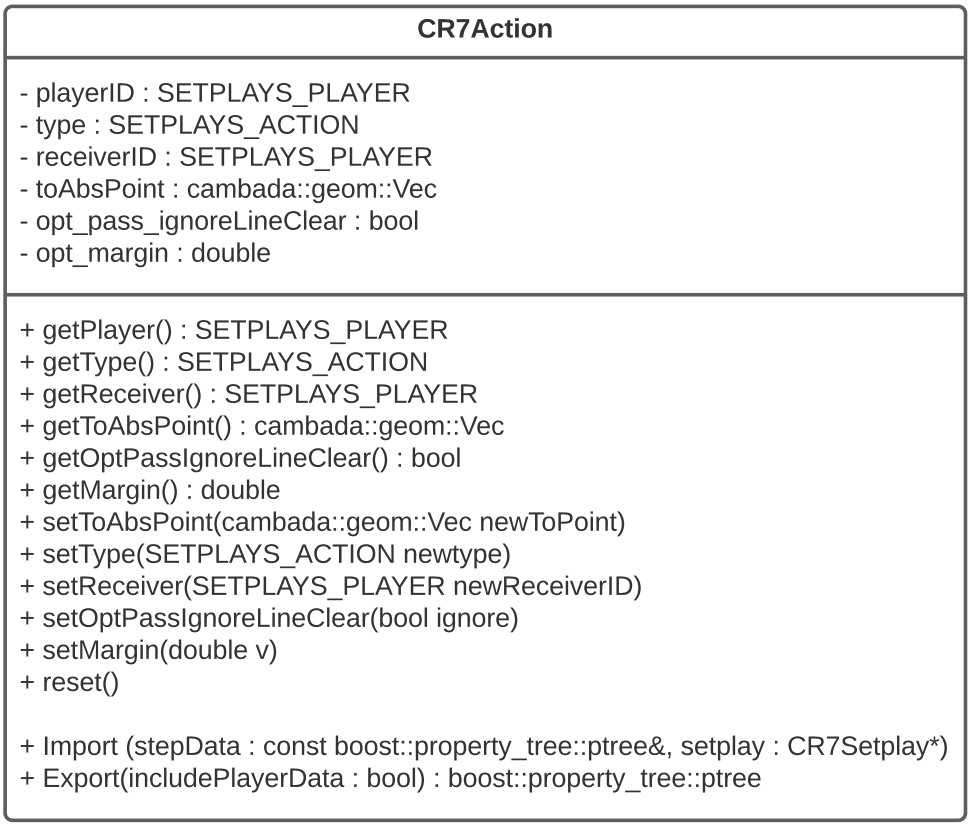
A.10 CR7Step Class Details



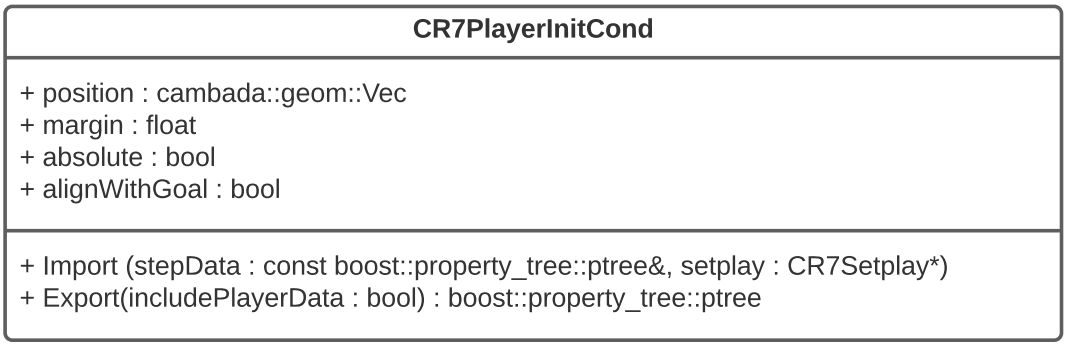
A.11 CR7Player Class Details



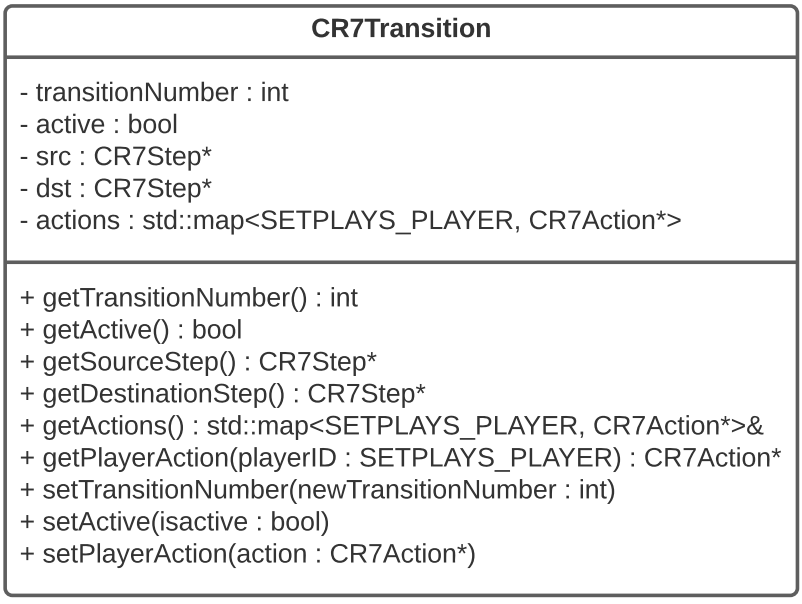
A.12 CR7Action Class Details



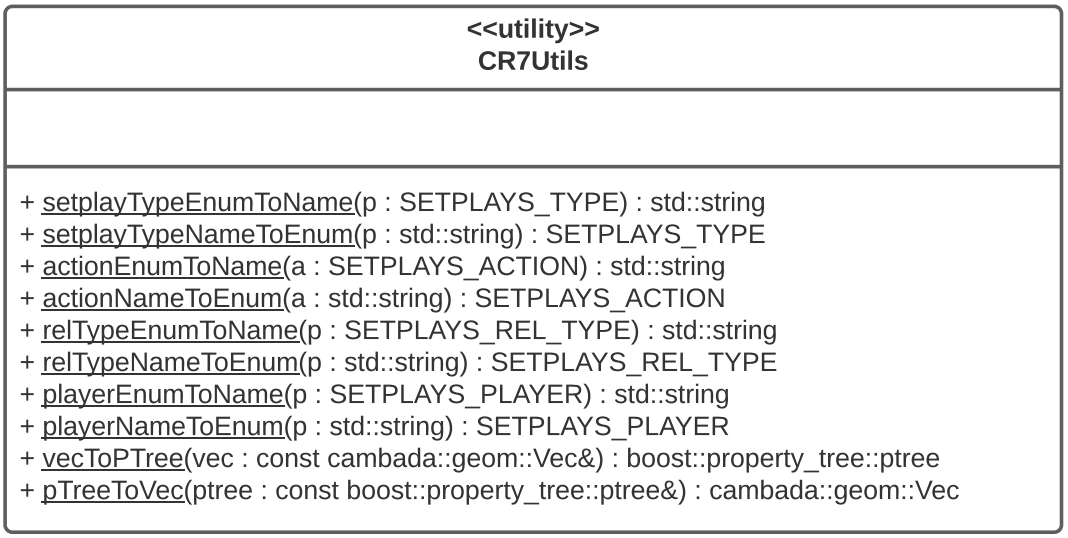
A.13 CR7PlayerInitCond Class Details



A.14 CR7Transition Class Details



A.15 CR7Utils Class Details



A.16 CR7 Config JSON Schema

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "$id": "http://robotica.ua.pt/cambada/cr7.schema.json",
  "title": "CR7 Config",
  "description": "The schema for a CR7 configuration",
  "type": "object",
  "properties": {
    "setplays": {
      "$ref": "#/definitions/setplays"
    },
    "zones": {
      "$ref": "#/definitions/zones"
    }
  },
  "required": [ "setplays", "zones" ],

  "definitions": {
    "setplays": {
      "description": "A list of setplays",
      "type": "array",
      "items": { "$ref": "#/definitions/setplay" },
    },

    "setplay": {
      "title": "Setplay",
      "description": "Definition of a setplay",
      "type": "object",
      "required": [ "name", "enabled", "max_exec_time", "cooldown_time",
      ↪  "type", "participants", "init_conditions", "steps" ],
      "properties": {
        "name": {
          "type": "string"
        },
        "enabled": {
          "type": "boolean"
        },
        "max_exec_time": {
```

```
      "type": "integer",
      "minimum": 0
    },
    "cooldown_time": {
      "type": "number",
      "minimum": 0
    },
    "type": {
      "type": "string",
      "enum": ["KickOff", "SetPiece", "Freeplay"]
    },
    "participants": {
      "type": "array",
      "items": {
        "type": "string"
      }
    },
    "init_conditions": {
      "$ref": "#/definitions/init_conditions"
    },
    "steps": {
      "$ref": "#/definitions/steps"
    }
  }
},

"init_conditions": {
  "description": "Definition of the setplay initialization conditions for
  ↪ each participant (the participant IDs should be used as keys in
  ↪ this object)",
  "patternProperties": {
    ".*": {
      "type": "object",
      "properties": {
        "absolute": {
          "type": "boolean"
        },
        "position": {
          "type": "array",
```

```
          "minItems": 2,
          "maxItems": 2
        },
        "margin": {
          "type": "number",
          "minimum": 0
        },
        "align_with_goal": {
          "type": "boolean"
        }
      }
    }
  }
},

"steps": {
  "type": "array",
  "items": { "$ref": "#/definitions/step" }
},

"step": {
  "type": "object",
  "properties": {
    "id": {
      "type": "integer",
      "minimum": 0
    },
    "end_step": {
      "type": "boolean"
    },
    "actions":  {
      "type": "array",
      "items": { "$ref": "#/definitions/actions" }
    },
    "ball_owner": {
      "type": "string"
    }
  },
  "required": ["id", "end_step", "actions", "ball_owner"]
```

```
},

"actions": {
  "description": "the participant IDs should be used as keys in this
  ↪  object to define individual actions",
  "type": "object",
  "properties": {
    "to_step": {
      "type": "integer",
      "minimum": 0
    }
  },
  "required": ["to_step"],
  "patternProperties": {
    ".*": { "$ref": "#/definitions/action" }
  }
},

"action": {
  "type": "object",
  "properties": {
    "action": {
      "type": "string",
      "enum": ["GetBall", "Wait", "Pass"]
    }
  },
  "allOf": [
    {
      "if": {
        "properties": { "action": { "const": "Pass" } }
      },
      "then": {
        "properties": {
          "params": { "$ref": "#/definitions/action_pass_params" }
        }
      }
    }
  ]
},
```

```
"action_pass_params": {
  "type": "object",
  "properties": {
    "receiver": {
      "type": "string"
    },
    "ignore_line_clear": {
      "type": "boolean"
    }
  },
  "required": ["receiver", "ignore_line_clear"]
},

"zones": {
  "type": "array",
  "items": { "$ref": "#/definitions/zone" }
},

"zone": {
  "type": "object",
  "properties": {
    "name": {
      "type": "string"
    },
    "cells": {
      "type": "array",
      "items": { "$ref": "#/definitions/zone-cell" }
    }
  },
  "required": ["name", "cells"]
},

"zone-cell": {
  "type": "object",
  "properties": {
    "id": {
      "type": "integer"
    },
```

```
        "end-cell": {
          "type": "boolean"
        },
        "split": {
          "type": "string",
          "enum": ["X", "Y"]
        },
        "divs": {
          "type": "array",
          "items": {
            "type": "number",
            "minimum": 0,
            "maximum": 1
          }
        },
        "cells": {
          "type": "array",
          "items": {
            "type": "integer"
          }
        }
      },
      "required": ["id", "end-cell"]
    }
  }
}
```