Universidade de Aveiro
Departamento de Electrónica, Telecomunicações e Informática,
2014

João Manuel Leite da Silva

Perceção e arquitectura de software para robótica móvel

Perception and software architecture for mobile robotics

Programa de Doutoramento em Informática
das Universidades do Minho, Aveiro e Porto

João Manuel Leite da Silva

Perceção e arquitectura de software para robótica móvel

Perception and software architecture for mobile robotics

Tese apresentada às Universidades de Minho, Aveiro e Porto para cumprimento dos requisitos necessários à obtenção do grau de Doutor no âmbito do doutoramento conjunto MAP-i, realizada sob a orientação científica de Prof. Doutor Nuno Lau e Prof. Doutor António Neves, professores auxiliares do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro.

o júri

presidente

Prof. Doutor Helmuth Robert Malonek
professor catedrático da Universidade de Aveiro

Prof. Doutor Joaquim Arnaldo Carvalho Martins
professor catedrático da Universidade de Aveiro

Prof. Doutor Jorge Manuel Miranda Dias
professor associado com agregação da Universidade de Coimbra

Prof. Doutor Luís Soares Barbosa
professor associado da Universidade do Minho

Prof. Doutor Eduardo Alexandre Pereira da Silva
professor adjunto do Instituto Superior de Engenharia do Porto

Prof. Doutor José Nuno Panelas Nunes Lau
professor auxiliar da Universidade de Aveiro

acknowledgements / agradecimentos

All the work developed during this PhD was possible only thanks to the support of several people that, directly or indirectly, helped me along the journey. Being this work included in a group, a team, a thanks is due to all the elements of the research group for the opportunity and working environment they have always provided. A special thanks is due to my supervisors, Nuno Lau and António Neves, for accepting this challenge with me and for accompanying me all the way.
To my friend, to whom I haven't always payed the deserved attention, a thank you for being there for me anyway. To my parents, a huge thank you for providing all the opportunities that made this objective possible. To Élia, a special thanks for joining me during this journey and for moving on with me.

Todo o trabalho desenvolvido ao longo deste doutoramento foi possível graças ao apoio de diversas pessoas que, direta ou indiretamente, me ajudaram ao longo de todo o percurso. Sendo um trabalho inserido num grupo, numa equipa, um agradecimento a todos os elementos do grupo de investigação pela oportunidade e pelo ambiente de trabalho que sempre me proporcionaram. Um agradecimento especial aos meus orientadores, Nuno Lau e António Neves, por terem aceite este desafio comigo e me acompanharem durante todo o percurso.
Aos meus amigos, a quem nem sempre terei dado a atenção merecida, um obrigado por estarem presentes de qualquer maneira. Aos meus pais, um enorme obrigado por me terem proporcionado todas as oportunidades que tornaram possível este objetivo. À Élia, um obrigado especial por se ter juntado a mim durante esta jornada e seguir caminho comigo.

Abstract

When developing software for autonomous mobile robots, one has to inevitably tackle some kind of perception. Moreover, when dealing with agents that possess some level of reasoning for executing their actions, there is the need to model the environment and the robot internal state in a way that it represents the scenario in which the robot operates.
Inserted in the ATRI group, part of the IEETA research unit at Aveiro University, this work uses two of the projects of the group as test bed, particularly in the scenario of robotic soccer with real robots. With the main objective of developing algorithms for sensor and information fusion that could be used effectively on these teams, several state of the art approaches were studied, implemented and adapted to each of the robot types.
Within the MSL RoboCup team CAMBADA, the main focus was the perception of ball and obstacles, with the creation of models capable of providing extended information so that the reasoning of the robot can be ever more effective. To achieve it, several methodologies were analyzed, implemented, compared and improved.
Concerning the ball, an analysis of filtering methodologies for stabilization of its position and estimation of its velocity was performed. Also, with the goal keeper in mind, work has been done to provide it with information of aerial balls.
As for obstacles, a new definition of the way they are perceived by the vision and the type of information provided was created, as well as a methodology for identifying which of the obstacles are team mates. Also, a tracking algorithm was developed, which ultimately assigned each of the obstacles a unique identifier. Associated with the improvement of the obstacles perception, a new algorithm of estimating reactive obstacle avoidance was created.
In the context of the SPL RoboCup team Portuguese Team, besides the inevitable adaptation of many of the algorithms already developed for sensor and information fusion and considering that it was recently created, the objective was to create a sustainable software architecture that could be the base for future modular development.
The software architecture created is based on a series of different processes and the means of communication among them. All processes were created or adapted for the new architecture and a base set of roles and behaviors was defined during this work to achieve a base functional framework.
In terms of perception, the main focus was to define a projection model and camera pose extraction that could provide information in metric coordinates. The second main objective was to adapt the CAMBADA localization algorithm to work on the NAO robots, considering all the limitations it presents when comparing to the MSL team, especially in terms of computational resources.
A set of support tools were developed or improved in order to support the test and development in both teams.
In general, the work developed during this thesis improved the performance of the teams during play and also the effectiveness of the developers team when in development and test phases.

Resumo

Durante o desenvolvimento de software para robôs autónomos móveis, é inevitavelmente necessário lidar com algum tipo de perceção. Além disso, ao lidar com agentes que possuem algum tipo de raciocínio para executar as suas ações, há a necessidade de modelar o ambiente e o estado interno do robô de forma a representar o cenário onde o robô opera.

Inserido no grupo ATRI, integrado na unidade de investigação IEETA da Universidade de Aveiro, este trabalho usa dois dos projetos do grupo como plataformas de teste, particularmente no cenário de futebol robótico com robôs reais. Com o principal objetivo de desenvolver algoritmos para fusão sensorial e de informação que possam ser usados eficazmente nestas equipas, várias abordagens de estado da arte foram estudadas, implementadas e adaptadas para cada tipo de robôs.

No âmbito da equipa de RoboCup MSL, CAMBADA, o principal foco foi a perceção da bola e obstáculos, com a criação de modelos capazes de providenciar informação estendida para que o raciocínio do robô possa ser cada vez mais eficaz. Para o alcançar, várias metodologias foram analisadas, implementadas, comparadas e melhoradas.

Em relação à bola, foi efetuada uma análise de metodologias de filtragem para estabilização da sua posição e estimação da sua velocidade. Tendo o guarda-redes em mente, foi também realizado trabalho para providenciar informação de bolas no ar.

Quanto aos obstáculos, foi criada uma nova definição para a forma como são detetados pela visão e para o tipo de informação fornecida, bem como uma metodologia para identificar quais dos ostáculos são colegas de equipa. Além disso foi desenvolvido um algoritmo de rastreamento que, no final, atribui um identificador único a cada obstáculo. Associado à melhoria na perceção dos obstáculos foi criado um novo algoritmo para realizar desvio reativo de obstáculos.

No contexto da equipa de RoboCup SPL, Portuguese Team, além da inevitável adaptação de vários dos algoritmos já desenvolvidos para fusão sensorial e de informação, tendo em conta que foi recentemente criada, o objetivo foi criar uma arquitetura sustentável de software que possa ser a base para futuro desenvolvimento modular.

A arquitetura de software criada é baseada numa série de processos diferentes e métodos de comunicação entre eles. Todos os processos foram criados ou adaptados para a nova arquitetura e um conjunto base de papéis e comportamentos foi definido para obter uma framework funcional base.

Em termos de perceção, o principal foco foi a definição de um modelo de projeção e extração de pose da câmara que consiga providenciar informação em coordenadas métricas. O segundo objetivo principal era adaptar o algoritmo de localização da CAMBADA para funcionar nos robôs NAO, considerando todas as limitações apresentadas quando comparando com a equipa MSL, principalmente em termos de recursos computacionais.

Um conjunto de ferramentas de suporte foram desenvolvidas ou melhoradas para auxiliar o teste e desenvolvimento em ambas as equipas.

Em geral, o trabalho desenvolvido durante esta tese melhorou o desempenho da equipas durante os jogos e também a eficácia da equipa de programação durante as fases de desenvolvimento e teste.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Robotics is nowadays a trendy topic throughout the world and is becoming more and more implanted in our society. There are a number of particular areas of robotics that are already implemented in our daily lives, specially in industry production lines where robots are already frequently used.

There are several categories of robots, both in terms of shape and type of control. In terms of control, there are typically three types of machines: remote controlled, autonomous or a fusion of both methods. In the research group where this work is included, the focus is mainly on autonomous mobile robots. This means that the robots being developed need to execute their tasks without human intervention.

1.1 Motivation

In many application scenarios (particularly those in which it is not possible or desirable to simply react to the environment conditions) for a robot to be able to autonomously move, it needs to have an internal representation of both its own body and abilities and of the ambient around it. **We will call this internal representation world model or worldstate.**

Thus, one of the challenges of mobile robotics is defining, managing and updating the robot internal world model. Sensor and information fusion techniques are widely used for these tasks.

Generally, robots have access to partial and uncertain information of the environment through a set of sensors. In dynamic environments, information fusion (of historical information and information coming from different sensors) is essential for the world model to be as precise as possible. Information fusion is usually addressed through the use of probabilistic techniques such as Kalman or particle filters, sometimes conjugated with maximum likelihood techniques.

Besides self localization, object localization and tracking, it is usually also important for an autonomous mobile robot to be able to execute a given task. For both self and

object localization, one needs to join information, not only from the surroundings but also proprioceptive information about the robot state. This is particularly important for robots which do not have a single shape, such as humanoid robots, since the positions of the sensors relative to the world can vary significantly.

Within our research group, one of the scenarios used for development is robotic soccer. This is a highly dynamic environment, with objects moving at high speed and several robots interacting with each other, both cooperatively and competitively, creating a series of different cases of perception particularities, such as occlusion or partial occlusion and discrimination of objects.

The main question that motivates the research of this PhD is: how complete and reliable can become the world model of an autonomous mobile soccer robot?

1.2 Objectives

The main objective of this thesis, which falls within the research area of robotics and artificial intelligence, is to develop fast and effective methodologies for building representations of the environment of multi-robot teams and make use of the representation of the state of the world to infer knowledge useful for decision processes.

The objectives of the work are the following:

- Study and test of existing information fusion methodologies, both for single and multi-robot team information fusion.
- Development of extensions for existing methodologies and creation of new ones, to improve the performance of the integration of information.
- Development of information fusion methodologies capable of providing good information about the ball in a robotic soccer scenario.
- Creation of a methodology for detection and recovery of robot lost situations.
- Development of information fusion methodologies that are able to provide information about the other robots present on the field during a soccer game.
- Development of a world model that enables the mapping of obstacles as team mates or opponents and tracking of the later, allowing high level coordination methodologies that use, for example, opponent covering.
- Improvement of the CAMBADA reactive obstacle avoidance algorithm.
- Deployment of all the developed solutions on the code used for official competitions, guaranteeing their full integration and execution within the existent restrictions.

- Creation of a modular software architecture for the Portuguese Team.

- Definition of a methodology for mapping pixel coordinates provided by a humanoid robot camera into robot centered metric coordinates.

The developed methodologies will be integrated and tested in the context of RoboCup, mainly in the CAMBADA Middle Size League soccer team and also in the Portuguese Team Standard Platform League team.

The proposed research work is in the area of robotics and artificial intelligence and focuses in the construction/definition of the representation of the world state, information fusion and low-level behaviors and software architecture for multi-robot platforms.

1.3 Contributions

During this PhD, several methodologies were developed and implemented to allow the definition of a world model for robotic soccer as complete and precise as possible. The work accomplished covers several topics and technologies used to estimate useful information about all the elements that are part of a world model for robotic soccer. The main contributions of this work are:

- A set of methods and techniques used to perform information fusion and build a world model in the scenario of robotic soccer. These methods are balanced between precision and computation time in order to respect the strict existent time constraints.

- An integrated methodology for estimation of information concerning the ball projection on the ground which includes a filtering methodology, based on Kalman filter to provide noise reduction and stabilize the position estimation while keeping it precise and a linear regression approach for further estimation of the ball velocity. This integrated solution includes a filter reset feature that is capable of detecting the degradation of the estimations and perform a reset to both the position and the velocity filtering.

- A set of methodologies for detecting the ball when it is airborne, estimate its position and integrate this information with the existing one on the ground plane. Three approaches were explored, starting with a single RGB camera approach based solely on color detection. Following this approach, a new one was proposed that combined the color detection with a circularity shape classifier, based on the analysis of a polar histogram of image regions containing the ball candidates. Finally, a third approach was proposed that uses a Kinect sensor, taking advantage of its depth sensor for detection of objects directly in the 3D space rather than just using projected information of RGB cameras.

- A methodology for detection of degradation of robot localization based on heading discrepancy detection. The solution is configurable in terms of necessary fitness and is sensor independent.

- A solution for merging visual information of individual points detected on obstacles surfaces to infer the size and position that characterizes each of them.

- An integrated solution for identifying and characterizing all the robots on the field. The solution includes a team mate matching methodology based on Euclidean distance for unequivocally identifying each robot that is a team mate with the correspondent robot identifier. The remaining obstacles are identified and tracked by means of a multiple hypothesis tracking approach, working over a Kalman filter implementation. Each tracked obstacle is unequivocally identified, allowing high level decision processes to, for instance, cover a given opponent.

- An approach on reactive obstacle avoidance based on the decision of a new free direction that the robot should take. This approach follows a methodology that uses virtual sonars, by iteratively searching around the robot for a free direction. This search, however, is not necessarily a pure angular search as is the case of the sonar, but follows a set of heuristics to prioritize the selection of the most useful directions. The developed sonar solution is highly configurable to allow easy parameterizations for different constraints, such as moving free or with the ball.

- An heuristic for generating robot trajectories that take the robot from an initial position and velocity to a final position and velocity considering the robot's acceleration constraints. This is achieved by generating a sequence of points representing the positions where the robot should be in each control cycle. Additional parameters necessary are the maximum allowed velocity and accelerations, which are defined by the robot model or can be provided by the user if lower values are desired.

- A base modular software architecture for the Portuguese Team, that allows further development to be more organized and effective, as well as more easily manageable.

- The application and evaluation of an adapted tribots localization algorithm to the Standard Platform League scenario.

- A methodology for estimating camera angles, relative to the ground, based on cross product analysis of a set of base reference vectors. The step-by-step analysis allows an understanding of each involved component which promotes the detection of possible errors in a specific component estimation.

- A set of new and improved tools to help the management and development on the robotic teams. The tools include remote control, configuration and monitoring applications, as

well as applications for off line data analysis. Resulting from the analysis made possible by one of the applications, a new methodology for estimating kick power was designed.

1.4 Publications

- João Silva, Nuno Lau, António J. R. Neves, João Rodrigues, and José Luís Azevedo. Obstacle detection, identification and sharing on a robotic soccer team. In *Portuguese Conference on Artificial Intelligence (EPIA)*, volume 5816 of *LNAI*, pages 350-360, Aveiro, Portugal, October 2009.

This paper presents a first approach for defining obstacle limits based on visually perceived points on their surface and the general methodology for identifying team mate robots among the resulting obstacles.

- João Silva, António J. R. Neves, and Nuno Lau. Identifying obstacles in RoboCup Middle Size League. In *Proc RecPad2009*, Aveiro, Portugal, November 2009.

This paper is a short paper that provides a summary of the obstacle detection and team mate identification solution.

- João Silva, Nuno Lau, João Rodrigues, José Luís Azevedo, and António J. R. Neves. Sensor and information fusion applied to a Robotic Soccer Team. In *RoboCup 2009: Robot Soccer World Cup XIII*, volume 5949 of *LNAI*, pages 366-377, Graz, Austria, February 2010.

This paper provides a description of several sensor and information fusion methodologies applied to the CAMBADA robotic soccer team. It presents a general overview of robot localization and ball integration in terms of position and velocity estimation and information sharing among the team. It further presents preliminary results on the team mate obstacle identification matter.

- António J. R. Neves, José Luís Azevedo, Bernardo Cunha, Nuno Lau, João Silva, Frederico Santos, Gustavo Corrente, Daniel A. Martins, Nuno Figueiredo, Artur Pereira, Luís Almeida, Luís Seabra Lopes, Armando J. Pinho, João Rodrigues, and Paulo Pedreiras. CAMBADA soccer team: from robot architecture to multiagent coordination. In Vladan Papic, editor, *Robot Soccer*, pages 19-45. INTECH, January 2010.

This book chapter presents a global overview of the CAMBADA robots, covering hardware architecture and control and software, from visual detection of objects to high level coordination and communications, including an overview of sensor fusion for world modelling.

- João Silva, Nuno Lau, and António J. R. Neves. Ball identification in the RoboCup Middle Size League. In *Proc RecPad2010*, Vila Real, Portugal, October 2010.

This short paper presents the first approach for detecting airborne balls in the Middle Size League by using a pure color identification methodology over the data from a RGB frontal camera.

- 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, March 2011.

This article provides a summary of the world modelling of the CAMBADA team, concerning ball position and velocity estimation based on the omni directional camera, wrong localization recovery and team mate obstacle identification.

- Pedro Fonseca, António J. R. Neves, José Luís Azevedo, and João Silva. An heuristic for trajectory generation in mobile robotics. In *Emerging Technologies & Factory Automation (ETFA)*, pages 1-4, Toulose, France, September 2011.

This paper presents the solution designed for generating trajectories defined by sequences of points for a CAMBADA robot to follow.

- João Silva, Mário Antunes, Nuno Lau, António J. R. Neves, and Luís Seabra Lopes. Aerial ball detection in RoboCup Middle Size League using polar histograms. In *Proc RecPad2011*, Porto, Portugal, October 2011.

This short paper presents a second solution for detecting airborne balls in the Middle Size League by applying a color/shape hybrid approach that measures the circularity of an objects by analyzing the statistics of a polar histogram over the object image.

- João Silva, Nuno Lau, and António J. R. Neves. Estimating world coordinates in perspective vision systems for humanoid robots. In *Proc RecPad2012*, Coimbra, Portugal, October 2012.

This short paper presents a model for estimating metric coordinates of a given point in an image provided by a humanoid robot's camera through its the projection from a camera pinhole model. A methodology for calibration of camera mounting angles and validation of the whole model is included.

- João Silva, Nuno Lau, and António J. R. Neves. Cooperative detection and identification of obstacles in a robotic soccer team. In Calin Ciufudean and Lino García, editors, *Advances in Robotics - Modeling, Control and Applications*, pages 219-235. iConcept Press, January 2013.

This book chapter presents a second approach defining obstacle limits based on visually perceived points on their surface and the general methodology for identifying team mate robots among the resulting obstacles, as well as the methodology used for sharing obstacle information among team mates.

- João Silva, Mário Antunes, Nuno Lau, António J. R. Neves, and Luís Seabra Lopes. Aerial ball perception based on the use of a single perspective camera. In *16th Portuguese Conference on Artificial Intelligence (EPIA 2013)*, volume 8154 of *LNAI*, pages 235-249, Angra do Heroísmo, Açores, Portugal, September 2013.

This paper presents a global analysis and overview about the use of a single RGB frontal camera to detect airborne balls by using a color/shape hybrid approach for detection and validation of candidates and the methodology for estimating the ball position based on the image detection as well as the integration of this information in the world model.

- Paulo Dias, João Silva, Rafael Castro, and António J. R. Neves. Detection of aerial balls using a Kinect sensor. In *RoboCup 2014: Robot Soccer World Cup XVIII*, LNAI, pages in-press, Springer, 2014.

This paper presents a third approach for detecting airborne balls in the Middle Size League which makes use of a Kinect sensor to provide depth information, rather than using single RGB camera information and projection. The presented work includes the detection of the ball in 3D and the estimation of its trajectory.

1.5 Thesis structure

The remainder of this document is structured in 6 chapters. Chapter 2 presents the theme of sensor and information fusion, as well as a review of some existent work in the area.

Chapter 3 presents a general overview of the RoboCup and two of its robotic soccer leagues, both in terms of environment as in terms of common approaches used by the teams. An overview of the robotic platforms used during the development of the presented work is given, being one of them fully developed and constructed by the research group. Given the application scenario used, this chapter also provides an insight about what we consider necessary to be modeled and represented in the worldstate of a robot that plays soccer, as well as a description of common points and modules developed during this work for both presented leagues.

Chapter 4 then describes the work performed in the context of the MSL CAMBADA team, with the analysis of the several different elements of their integration in the worldstate. The elements analyzed include the ball which needs to have its visual raw information integrated in the worldstate, whether the visual information comes from the omni directional main camera or from an "extra" frontal sensor. The other robots on the field are also subject to analysis and the integration of their information on the worldstate is presented, both in terms of identification of team mates and tracking of opponents. A reactive obstacle avoidance methodology is also presented, as well as an heuristic for generating trajectories for robot movement based on sequences of points on the field.

In chapter 5 the work performed in the context of the SPL Portuguese Team is described and presented, including the proposed software architecture for the NAO robots, including the definition of used processes and protocols and architecture of the agent. A projection model for estimating metric coordinates from the pixel information provided by the camera is presented, making use of the estimated robot body pose. The application of the tribots localization algorithm in the context of the SPL is also presented.

For developing many of the proposed methodologies, a set of tools was created to assist the developers. Chapter 6 present the new and improved developed applications, both in the context of the MSL and in the context of SPL.

Finally, in chapter 7, a summary and discussion of the general results obtained during this PhD is presented, as well as a small discussion and proposal of future work for the related topics.

Chapter 2

Sensor and information fusion

Sensor and information fusion is defined as the act of combining sensory data, or data derived from sensory data, providing a resulting information that is in some sense better than would be possible when the sources were used individually [1]. In information fusion, better can mean more accurate, more complete, more dependable, or refer to the result of an emerging view or combination of sources, such as the stereoscopic vision disparity image for 3D estimation.

A general overview of different methods of multi-sensor and information fusion is given by Luo *et al.* [2], also with a brief description of application areas, such as robotics, military, biomedical and transportation areas. The 25^{th} chapter of the "Springer Handbook of Robotics" [3] provides a general overview of methods and architectures for multi-sensor data fusion [4].

In the context of this thesis, sensor fusion will be used to build a representation of the real world surrounding a team of soccer robots. The main challenges will be the estimation of the robot's own position and velocity as well as the position and velocity of the ball (either on the 2D or 3D space) and of the other robots on the field. In the case of the other robots, the information available must be merged to deduce which of these should be identified as team mates or opponents. These tasks must be explored both for single robot and for multi-robot information.

The integration of information over time in order to filter sensor noise is essential to get better estimates. This type of integration is mostly performed using Kalman filter based approaches, Monte-Carlo methods or Markov approaches. Generally, Monte-Carlo [5] approaches have better performance in cases where great discontinuities of the output values are expected, as the assumption of Gaussian probability density functions of the Kalman filter [6] is less accurate. However, Kalman filtering is a very effective method if the assumptions of Gaussian noise can be met and the system can be linearized. In this case, when the system can be linearized, other common approaches are the use of the Extended and Unscented Kalman filters [7], at the cost of more computational weight.

When a mobile robot is autonomously performing its tasks, it gathers information with its own noisy sensors and estimates its state and the state of the environment. This estimation does not necessarily correspond to the true state of the robot and the environment. The state that it keeps is thus called *belief*, as the robot believes it is in a particular state, but there is no guarantee that it is really correct; it is an estimate. This belief, $bel(X)$, can be represented as a probability distribution function (in the discrete case) or a probability density function (in the continuous case), where X is the estimated set of values for the state variables of interest, independently of the type of data considered.

At some moments of the run, the robot gets information from sensors that are at its disposal. These sensors can get information about the surroundings or the robot internal state and that information has some degree of accuracy about the own and environment states at that instant. A robot can be equipped with a variety of sensors to help its purposes and the sensors provide measurements, Z, that are also not 100% accurate, as they have some noise associated. The observations may be described by an observation model written as a probability function, $p(Z|X)$.

When using these kind of algorithms, we are facing mostly a probabilistic problem. We need a state transition model which defines how the state evolves through time and is capable of providing the beliefs for the robot. These beliefs can then be reinforced or hindered by sensor measurements. We also usually have some kind of control over the evolution of the state by actions that can be executed. These controls are denoted as U, and affect the output of the system evolution from the state transition model.

The probabilistic algorithms are usually two step processes: first a forecast of the state is made based on the evolution of the noisy dynamic system, an *a priori* estimate, and then measurements from the robot sensors are combined with the forecast in order to produce a final *a posteriori* probabilistic estimate of the system's state, the belief. In some situations, measurements may be unavailable. If no sensor readings are available, the belief generated by the transition model tends to become more uncertain, because it relies only on the forecasts of the state until a new measure can help correcting it.

2.1 Filters

There are several filtering techniques available on the literature, which can be used for virtually any purpose one can imagine and need. In this section a small overview of a few of the most popular filtering techniques is presented.

2.1.1 Bayes filter

Based on Bayes conditional probability rule, the Bayes filter is a recursive algorithm to estimate the belief distribution from measurements and control data. Being recursive, it

estimates the belief for current time t from the belief of last time $t-1$.

On the first phase, the prediction phase, it calculates a belief for state X based on the prior $bel(X)_{t-1}$ and the current control U_t. The value assigned to $\hat{bel}(X)_t$ is obtained by the integral (or sum in the discrete case) of two distributions: the distribution of $bel(X)_{t-1}$ and the probability with which control U_t creates a transition from previous state X' to current state X. This belief is a temporary value obtained on this first phase. The final belief will be updated afterwards based on this initial approach

$$\hat{bel}(X)_t = \int p(X|U_t, X')bel(X')_{t-1}dX'. \tag{2.1}$$

On the second phase, the state belief is updated based on the available measurements. The transitional belief is multiplied by the probability that the measurement Z_t may have been observed for every hypothetical posterior state X

$$bel(X)_t = \eta\ p(Z_t|X)\ \hat{bel}(X)_t,$$

where η is a normalization constant to guarantee a probability value.

2.1.2 Kalman filter

Presented by Rudolf E. Kalman in his 1960 paper [6], the Kalman filter is an *optimal recursive data processing algorithm* that estimates the state of a dynamic system from a series of noisy measurements [8, 9]. One aspect of this optimality is that the Kalman filter processes all information that can be provided to it, regardless of its precision, to estimate the current value of the variables of interest. It uses the knowledge of the system and measurement devices dynamics, the statistical description of the system noises, the measurement errors, the uncertainty in the dynamics model and any available information on initial conditions of the variables of interest [9].

Kalman filters are based on linear dynamical systems discretized in time. It is assumed that the system and the measures are affected by *White Gaussian* noise, meaning that the noise is not correlated in time, and thus we can assume that at each discrete time, the noises affecting the system and measures follow a Gaussian distribution and are independent of past or future values [8].

The Kalman filter model assumes that the belief at time t is evolved from the state at *(t-1)* according to the model of the system evolution, the *action model*:

$$X_t = F_t X_{t-1} + B_t U_t + w_t,$$

where

- X_t is the state at time t.

- F_t is the state transition model applied to the previous state X_{t-1}.
- B_t is the control-input model which is applied to the control vector U_t.
- U_t is the control vector, over which the control-input model B_t is applied.
- w_t is the process noise assumed to be a zero mean white Gaussian with covariance Q_t and independent of X

  $w_t \sim N(0, Q_t)$.

At time t, a measurement Z_t of the state X_t is made according to a model of the system observation, the *observation model*,

$$Z_t = H_t X_t + v_t,$$

where

- H_t is the observation model that defines how the measurement variables are mapped into the state variables.
- v_t is the observation noise assumed to be zero mean white Gaussian noise with covariance R_t

  $v_t \sim N(0, R_t)$.

Being a recursive estimator, the filter only needs the estimated state and correspondent covariance from the previous time step and the current measurement to compute the estimate of the current state, no other history of measurements or estimates are required. The filter state is represented by two variables:

- $\hat{X}_t$, the estimate of the state at time t, the mean of the state values.
- $\hat{P}_t$, the error covariance matrix, an estimation of how accurate is the estimate state.

In the first step of the filter, the forecast, predictions of the two variables are made, based on the previous estimated state, given the expressions

$$\text{Predicted state:} \qquad \hat{X}_{t_{predicted}} = F_t \hat{X}_{t-1} + B_t U_t \ ; \tag{2.2}$$

$$\text{Predicted estimate covariance:} \qquad \hat{P}_{t_{predicted}} = F_t \hat{P}_{t-1} F_t^T + Q_{t-1}.$$

The second step updates the forecast by taking the current time step measure into account. For that purpose, the measurement *innovation* or *residual* is calculated

$$Y_t = Z_t - H_t \hat{X}_{t_{predicted}}.$$

The residual reflects the discrepancy between the predicted measure and the actual measure. A zero residual means the prediction and the actual measures have the same values. Having the residual and the forecast, we need to estimate the Kalman gain at time t K_t.
The computation of the Kalman gain depends on the predicted covariance and the respective residual:

$$K_t = \hat{P}_{t_{predicted}} H_t^T S_t^{-1},$$

where S_t is the *residual* of the covariance matrix, which takes into account the measurement noise covariance matrix R_t:

$$S_t = H_t \hat{P}_{t_{predicted}} H_t^T + R_t.$$

The updated state can then be computed by:

$$\hat{X}_t = \hat{X}_{t_{predicted}} + K_t Y_t,$$

$$\hat{P}_t = (I - K_t H_t)\hat{P}_{t_{predicted}}.$$

The new updated variables represent the state as a belief $bel(X)_t$ that is a Gaussian function represented by its mean ($\hat{X}_t$) and covariance ($\hat{P}_t$).

2.1.3 Monte Carlo (Particle) filter

The idea of using factored sampling to estimate a set of variable states was introduced by Isard & Blake [10], with application to computer vision contour tracking.

Particle filter represents the belief $bel(X)_t$ by a set of M samples $\mathcal{X}_t = x_t^{[1]}, x_t^{[2]}, ..., x_t^{[M]}$ with associated weights. Each sample is one possible state of the system at the current time t and the the computed estimations of the state, the final outputs of the system, are based on these samples and weights.

In a first step, the previous state needs to be updated according to the system evolution model imposed by the controls applied to the object. Each of the particles/samples in M is propagated according to the defined system evolution model.

$$\begin{aligned}&\text{For all } m \in [1..M]\\ &\quad x_t^{[m]} \Leftarrow \arg\max_{x_t} p(x_t|U_t, x_{t-1}^{[m]}).\end{aligned}$$

In a second step, and for each sample, its weight is calculated. The weights are used to incorporate the measurement Z_t into the particle set and they are interpreted as the probability of the observation Z_t under the state represented by the particle $x_t^{[m]}$. All the particles are added into a "temporary set" with weighted samples.

$$\begin{aligned}&\text{For all } m \in [1..M]\\ &\quad w_t^{[m]} = p(Z_t|x_t^{[m]})\\ &\quad \hat{\mathcal{X}}_t = \hat{\mathcal{X}}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle.\end{aligned}$$

Finally, a *resampling* is made over the temporary set $\hat{\mathcal{X}}_t$. This step draws the particles of $\hat{\mathcal{X}}_t$ with a probability proportional to their previously assigned weight and adds them to a new set $\mathcal{X}_t$, thus creating the new set with the same number of particles but with a different particle distribution, by focusing the particle set to regions of the state space with higher probability.

$$\begin{aligned}&\text{For all } m \in [1..M]\\ &\quad \text{draw } i \text{ with probability } \propto w_t^{[i]}\\ &\quad \text{add } x_t^{[i]} \text{ to } \mathcal{X}_t\end{aligned}$$

The final representation of the state $bel(X)_t$ is usually estimated from the set of particles according to their weights and the estimation method depends on the application. Laue *et al.* [11] present an analysis of some possible methods. As an example, one can simply accept the particle with higher weight as the best estimate of the true state or, more commonly, use a weighted average of the particles.

A common extension to the Monte Carlo (MC) algorithm is the addition of an heuristic to add random particles to the particle set. This is done because the particles at places other than the most likely state gradually disappear. Since at some point they only "survive" near a single state, there is no way to recover if this state is incorrect. This kind of approach enables the MC algorithm to cope with sudden significant changes of state.

2.2 Localization problem

Self-localization and mapping are classic problems of intelligent mobile robotics, over which research is still extremely active. These are part of the more general challenge of defining, managing and updating the robot internal world model.

Given the abundance of published work that focus on this problem, most of the available literature and examples concerning the presented filtering methods are presented within this context, as well as variants and improvements for the base algorithms.

Thus, this section appears as context and as general presentation and exemplification of the described methods, despite the fact that this work does not focus on self-localization. Many of the concepts, however, are generalizable and are applied to the application of the methods presented on this work.

When working with mobile autonomous robots, there are typically two scenarios. The environment can be known or partially known and the robot needs to localize itself, or the environment is unknown and the robot needs to build the map as it runs. The latest is addressed as Simultaneous Localization And Mapping (SLAM) and it is another common application of sensor fusion techniques [12, 3].

The problem of mobile robots localization is to identify where a robot is, given a map of the environment around it. The localization of a robot is usually defined as a pose, which contains a position (given in some coordinate system) and an orientation (relative to the defined coordinate system).

In this case the state of the robot, X, is its pose, either represented in 2D or 3D space. When in a 2D scenario, the pose of the robot is given by its 2D ground projection (X,Y) coordinates and its orientation, which results in a three variable state. In a 3D scenario, usually we want to keep information about a specific point on the robot body, like its head, and the pose is given by a six variable state, which are the robot reference point coordinates (X,Y,Z) and the robot body orientation (or orientation of the body part containing the reference point) given by the angles around each of the (X,Y,Z) axis, usually referred as roll, pitch and yaw.

The localization problem is mostly a probabilistic problem. Robots must have models for their movement (*motion models*) that are capable of providing the beliefs. These beliefs can then be updated considering sensor measurements or, when they are not available, updated based only on the motion model.

The localization problems are usually divided considering different aspects that are not equally difficult to solve. There are four main aspects to consider [12]:

- **Local versus global localization**
  This characterizes the problem by the type of knowledge available initially and at run-time. Three different problems are distinguished, each with increasing difficulty.
  - **Position tracking**. Position tracking assumes that the initial pose of the robot is previously well estimated. That means that the localization algorithm has to estimate the new pose based on the last one, which usually has a small error that can be accommodated in the motion and sensor models with good results.
  - **Global localization**. In this case, the initial pose is unknown. The robot is placed somewhere in the environment and thus no assumptions on the limits of the pose error can be made. Global localization includes the position tracking problem.
  - **Kidnapped robot problem**. This is a variant of global localization with an added difficulty: during operation, the robot can be kidnapped and teleported to other location. The robot might believe that it knows where it is while it does not, leading to subsequent wrong pose estimations due to false initial pose knowledge. This leads to a new question of how can the pose estimation be validated and how can a wrong pose be detected. In the global localization problem, this question is not relevant, as the robot knows that it does not know where it is.
- **Static versus dynamic environments**
  The environment dynamics also causes a substantial impact on localization difficulty.

The environment is typically classified in two classes.

- **Static environment**. In this kind of environments, the robot is the only element with motion, meaning the only state variable is the pose of the robot. All the other objects in the environment remain at the same location forever.
- **Dynamic environment**. These environments have objects other than the robot whose position and configuration may change over time. Some examples of more significant changes are people, movable furniture, doors, or even light conditions (daylight, night).

Working with dynamic environments creates more difficulties than working with static ones. There are two main approaches for accommodating dynamics: one is to include the dynamic entities in the model state vector. This approach also maps the environment, but it comes with a high burden of computational and model complexity. Another approach is to filter the sensor data to correct the damage caused by unmodeled dynamics [12].

- **Passive versus active approaches**
  This aspect of localization problem characterization pertains to the fact of whether or not the localization algorithm can influence the control over the robot.

  - **Passive localization**. The localization module only observes as the robot operates. The robot movement is defined and oriented to perform the robot given task, not considering localization as a goal in its decisions.
  - **Active localization**. On active approaches, the localization algorithms have direct influence over the robot's control and attempt to move it to a more favorable position to reduce the error and obtain a better pose.

  Active approaches tend to yield better and quicker results than passive ones since the goal of the robot is to localize and thus it will search for the best information possible for that task. An example is a robot located in a symmetric corridor where the global localization can easily enter an ambiguous state (figure 2.1). The local symmetry makes it impossible to localize the robot while in the corridor. It will only be able to eliminate the ambiguity and determine its pose if it moves into a room (or if it gains access to some other disambiguation source, like a true heading, for instance).

  However, in practice, an active localization technique tends to be insufficient if applied on its own. Most of the times, the robot has to be able to execute other tasks besides localization. A common approach is to merge the localization goals with the task goals. An example would be a robot moving from a point A to B taking a longer path, but ensuring that the robot kept itself well localized along all the way.

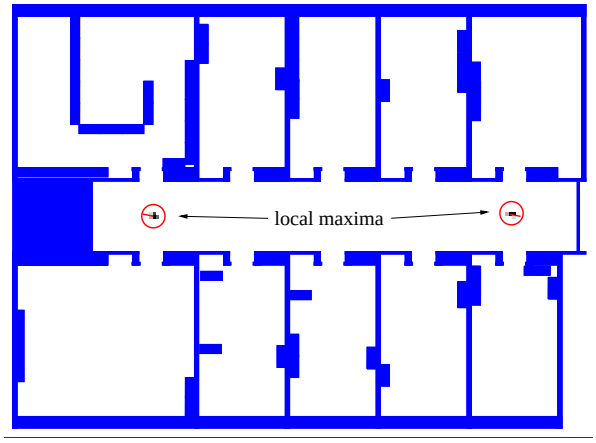


Figure 2.1: Example situation with two possible ambiguous poses. The robot would need extra information or to enter one of the rooms to determine its true location. Image from [12].

- **Single-robot versus multi-robot**
  The fourth aspect of the localization problem is related to the number of robots involved
  - **Single-robot**. This is the most typically studied approach, dealing with only one robot. It is convenient that all the data comes from the same platform, and no communication issues have to be dealt with.
  - **Multi-robot**. When working with a team of robots, the problem can be addressed as several single-robot localization problems and be solved in the same manner. However, if the robots have the ability to detect each other, one robot's beliefs can be used to validate other robot's beliefs if the relative location of both is available.

2.2.1 Localization algorithms

As localization is a probabilistic problem, most of the algorithms for mobile robot localization are based on Bayesian rules. Some algorithms will be briefly presented in this section.

2.2.1.1 Markov localization

Markov localization is the straightforward application of a Bayes filter (section 2.1.1) to the localization problem. It requires a map as input for the measurement model and often the map is also incorporated in the motion model. It mainly transforms a probabilistic belief at time *t-1* into a belief at time *t*. Being a probabilistic belief at each instant, it maintains the probability for every possible pose on the state space.

Markov localization addresses the global localization problem, tracking problem and the kidnapped robot problem in static environments.

Consider a scenario of a hallway with three identical doors and the robot is only able to move along the hallway without rotating. It is a one-dimensional example (figure 2.2).

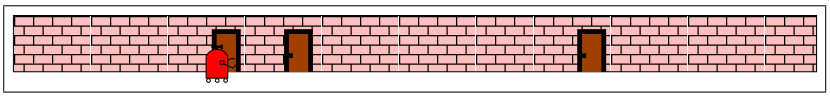


Figure 2.2: One-dimensional scenario for robot localization. Image from [12]

Figure 2.3 illustrates this one-dimensional example (presented by Thrun *et al.* [12]), where a robot moves along a corridor with three identical doors. In an initial state, the belief $bel(X) = bel(X_0)$ is uniform for all poses along the corridor (figure 2.3(a)).

In figure 2.3(b), the robot queries its sensors (modeled by a function $p(Z|X)$) and detects that it is in front of a door. The belief is updated with that information, resulting in the belief in the same image.

In a third moment, the robot moves right, and the model convolution (expression 2.1) results in the belief depicted in figure 2.3(c), flattened by the assumptions of the motion model uncertainty.

A new sensor query is made in figure 2.3(d) and the resulting belief now has a well defined peak focused on the correct pose. At this point, the robot is quite confident that it has localized itself. Figure 2.3(e) illustrates the robot's belief after a new move to the right.

This illustration refers to the global localization problem. In a position tracking problem, the initial position would be known and thus the initial state would be something like the belief in figure 2.3(d).

A Markov localization approach has been explored in robotic museum tour guides [13]. The work described by Fox *et al.* [13] was tailored for dynamic environments and was developed to support both global localization and kidnapped robot problems (recovery from location failures).

2.2.1.2 Kalman filter localization

Kalman filter localization is the application of the Kalman filter (section 2.1.2) to the localization problem and typically requires that the starting position of the robot is known. Thus, in its essence, it addresses the position tracking problem. Unlike Markov localization, it maintains a belief $bel(X_t)$ that is a Gaussian function and thus can be represented by its mean and covariance.

In the corridor example of Thrun *et al.* [12] (depicted in figure 2.4), it is assumed that the map is represented by a collection of features (where each one is identified by a *correspondence variable*) and each feature identifier is known, thus, the doors of the hallway have now a unique *correspondence variable* (1, 2 and 3). The need for these constraints is that the filter works

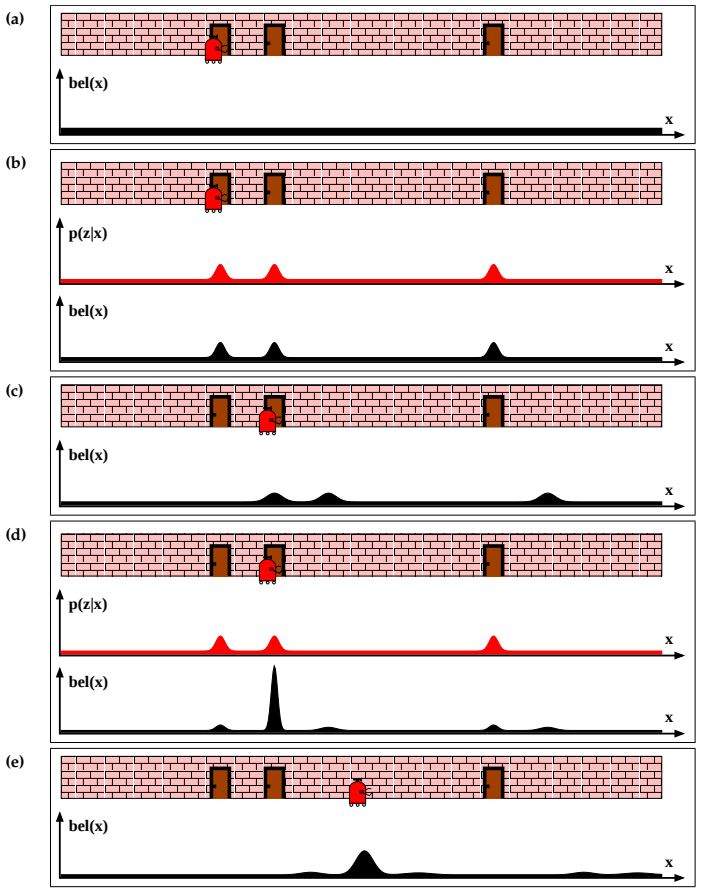


Figure 2.3: Illustration of Markov localization algorithm. In each picture, the belief $bel(X)$ function is represented. In (b) and (d), also the observation model $p(Z_t|X_t)$ is represented, describing the probability of observing a door at the different locations in the hallway. Image from [12].

on the assumption of Gaussian measurements and for this requirement to be met, each door must be unequivocally identified by the observation model $p(Z|X)$

A second assumption, needed for the algorithm, is that the initial position is relatively well known (the initial belief $bel(X_0)$ is represented by the Gaussian distribution shown in figure 2.4(a), near door 1 and with a Gaussian uncertainty).

As the robot moves right (figure 2.4(b)), its belief is convolved with the Gaussian motion model (expression 2.2), resulting in an increase of the Gaussian width, as uncertainty increases.

In another instant, the robot detects that it is in front of door number 2. In figure 2.4(c), the observation function $p(Z|X)$ is used to update the estimated pose, and the resulting belief is presented in the same picture. The variance of the resulting belief is smaller than both the previous belief and measurement variances, thus integrating two independent estimates should make the robot more certain than each of the estimations separately.

As the robot continues to move along the hallway, the uncertainty in its pose increases

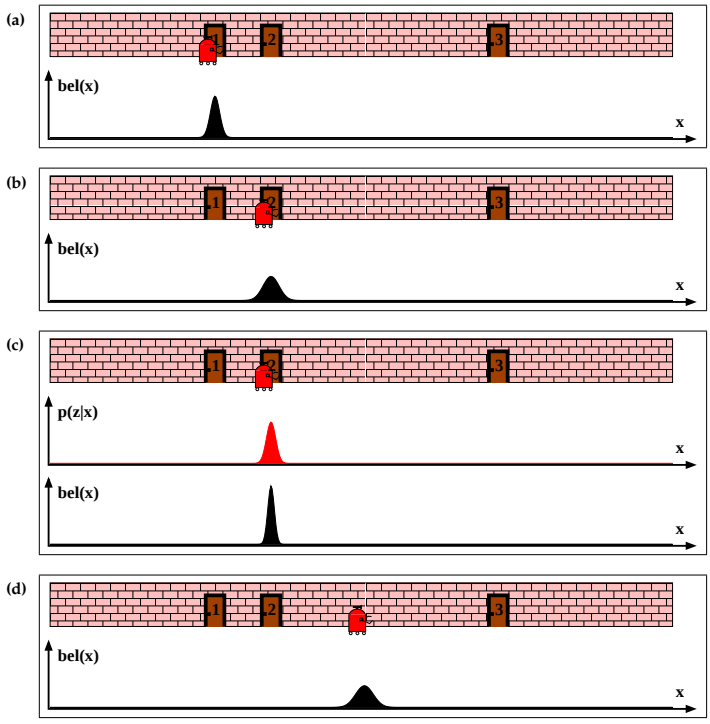


Figure 2.4: Illustration of Kalman filter localization algorithm. In each picture, the belief $bel(X)$ function is represented. In (c) also the observation model $p(Z_t|X_t)$ is represented. All densities are represented by uni-modal Gaussians. Image from [12].

again, since the filter motion model continues to incorporate the system uncertainty in the belief (figure 2.4(d)).

When the correspondence variables are unknown, the identity of the landmarks has to be determined during localization. One of the most simple and used strategies is *maximum likelihood* [14] correspondence, in which one first determines the most likely value of the correspondence variable and then applies the filter as described, using the estimated correspondence as granted. With this kind of approximations, the Kalman filter localization can be extended for the global localization problem.

There are a number of works that try to use Kalman filters in various ways, to somehow improve it so it can be adapted for some particular situations. One of them, a combination of Bayesian estimation with Kalman filtering for localization purposes is presented by Roumeliotis *et al.* [15], allowing a better performance of the system by relaxing the Kalman assumptions about the measurements noise.

A work described by Bekey *et al.* [16] applies the Kalman filter basis divided in several smaller communication Kalman filters. Each of the robots has its own filter to process the data from its own sensors. Information exchange between two individual filters is only necessary when two robots detect each other and measure their relative position. In this case, a more

comprehensive Kalman filter is used, capable of taking the new information into account.

In the work by Koyasu *et al.* [17], the authors present the use of Kalman filter to detect and track moving obstacles.

Leonard *et al.* [18] make use of an extended Kalman filter to localize a mobile robot based on environment landmarks that can be reliably observed and accurately described by geometric parameterization (referred by the authors as geometric beacons).

2.2.1.3 Monte Carlo localization (MCL)

This is another popular algorithm for localization, which represents the belief $bel(X_t)$ by a set of M particles $X_t = x_t^{[1]}, x_t^{[2]}, ..., x_t^{[M]}$. It is the application of a particle filter (described in section 2.1.3) to the localization problem. The initial belief $bel(X_0)$ is obtained by randomly generating M pose particles from the prior distribution $p(X_0)$ (usually uniformly over the entire pose space) and assigning each of them a uniform importance factor of M^{-1}.

This algorithm can address both position tracking and global localization problems.

Figure 2.5 illustrates the MCL in the one-dimensional hallway example. The initial belief is a set of random pose particles uniformly generated for the entire pose space (figure 2.5(a)). When the sensors are queried and a door is sensed, the MCL algorithm analyses each particle and assigns it an importance factor considering the measurement model, which takes into account the estimated particle, the current observation and the map.

In figure 2.5(b), the resulting particle set is shown, with the height of the particle representing its importance factor. At this stage, the set of particles is identical to figure 2.5(a), being the importance the only modification. This importance is then used for the re-sampling process, that, based on the set of figure 2.5(b), generates new particles more concentrated around the most likely positions. These newly generated particles are once again assigned the same importance factor, as each of them is basically one possible state. Figure 2.5(c) shows the new set of particles, also after incorporating a motion.

Again, the new measurement assigns non-uniform importance weights to the particle set, as depicted in figure 2.5(d). At this point, most of the cumulative probability mass is centered on the second door, which is also the most likely location (and the correct one).

After a new re-sampling phase which creates a new set, a new motion updates the particles again and we obtain the set represented in figure 2.5(e). Each motion step without measurements tends to disperse the particles, but each motion step merged with measurements tends to concentrate the particles around the correct pose.

MCL has the advantage of not being bound to assumptions about the system and measurement noise (as is the case of Kalman filter, which assumes that the noise is Gaussian). The accuracy of the estimated pose increases with the size of the particle set M, although this parameter imposes a trade off between accuracy and computational weight.

As stated in the particle filter description, the addition of random particles into the

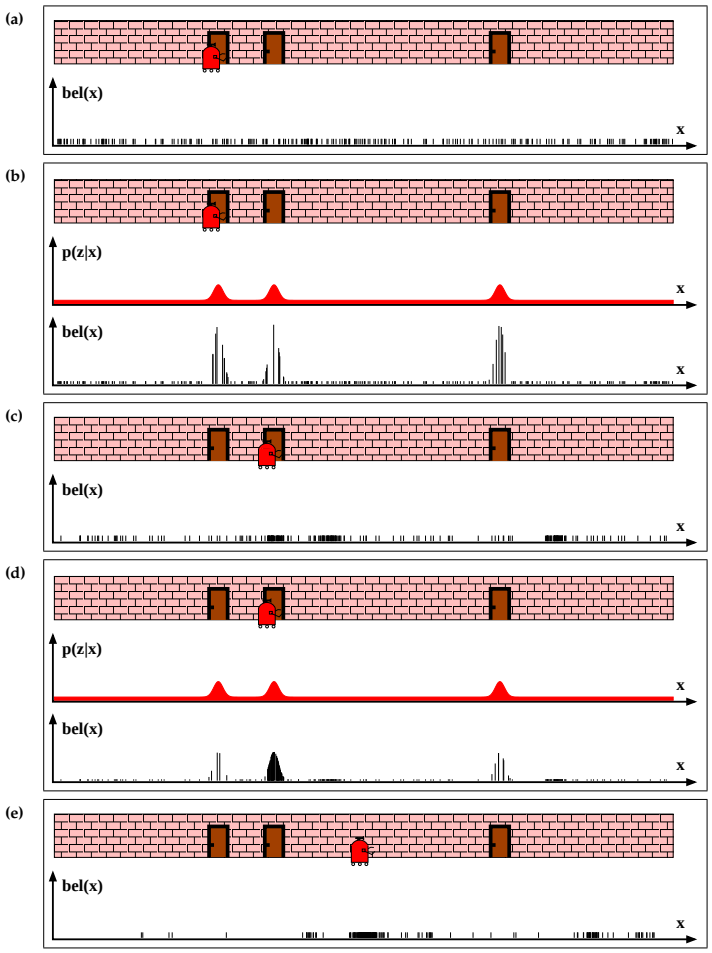


Figure 2.5: Illustration of Monte Carlo localization algorithm. In each picture, the belief $bel(X)$ function is represented. In (b) and (d) also the observation model $p(Z_t|X_t)$ is represented. Image from [12].

particle set is a common, if not mandatory feature when applying it to localization. Without the random particles spread around to act as seeds for new particles during eventual re-localizations, the MCL could not effectively solve the kidnapped robot problem, as it would keep focusing the particles on the old outdated pose and would most likely take a very long time to converge to the new pose. With the random particles around, they usually do not affect the global estimation when it is focused on the correct pose (as these stray particles weight is usually negligible) but can be crucial for the algorithm in case the robot suddenly moves near one of them.

The use of a Monte Carlo localization algorithm with a variable number of particles is presented by Heinemann *et al.* [19], in an attempt to get a good (accuracy / time and computational cost) balance of MCL, for use in a demanding environment like robotic soccer.

Montesano *et al.* [20] present a study of how vision-based bearings and motion can be

used for pairs of robots to localize themselves using each other as landmarks. They try an extended version of Kalman filter, a particle filter approach (MCL) and a combination of both. They observed that the particle filtering approach tends to be more robust than the Kalman filter to estimate the initial location. Once the filters converge to the true location, all methods provide similar results. The combination of both seems to provide the best compromise between robustness and efficiency.

Gutmann and Fox [21] present a comparison between Kalman filtering, Markov localization, Monte Carlo localization and combinations of them in a landmark based scenario, and present some results and comments on several strong and weak points of each approach. In a general way, the combined methods yield better results than the standard versions.

Dellaert *et al.* present the use of MC algorithm for localization of mobile robots [22] performing tasks as museum tour guides. In the work by Cabecinhas *et al.* [23], the authors compare the utilization of both a Kalman filter and a MC method for self localization of a soccer robot by fusing odometry and visual information and find them both suitable for their application, both with improvement potential for the integration of more sensors.

2.2.1.4 Tribots localization

The Tribots algorithm was created in the scope of robotic soccer and is based on guided update steps modeling the localization problem as an error minimization task [24]. However, its application can be more general (an example is the work presented by Gouveia *et al.* [25]). The robotic soccer scenario is composed by a soccer field with well known lines marking the field. These include straight lines for the bounding boxes of the field and areas and curved lines for the center circle and corner arcs. These lines define the map where the robot needs to localize itself and are landmarks that can be detected by visual sensors.

The algorithm was created to work on environments where there are no uniquely identifiable landmarks and relies on a Look Up Table (LUT) built over the map which, for each position, keeps the minimum distance to the closer landmark.

The algorithm starts by assuming a given pose as true and estimates the error of that pose related to the measured distances (distances of the considered pose to the landmarks represented by the observation model, in the robotic soccer case, points on the field lines) by comparing with the LUT distances to the existing landmarks (known distances of each position to the defined map landmarks, which are, in the robotic soccer case, points on the complete field lines). The objective is to maximize the fitness (minimize the error) of the match between the detected landmarks and the known landmarks (from the map) by gradually correcting the estimation. The pose that will be tested in the next step of the process is given by applying a displacement to the current pose based on the gradient of the current error, using an algorithm named RPROP [26].

Consider the example of its main application, localization of a soccer robot. The land-

marks used on the soccer field are the white lines, which are known *a priori*, as they are defined in the rules. The LUT of the map is then based on a representation of the distance of each position to the closer line/landmark (figure 2.6).

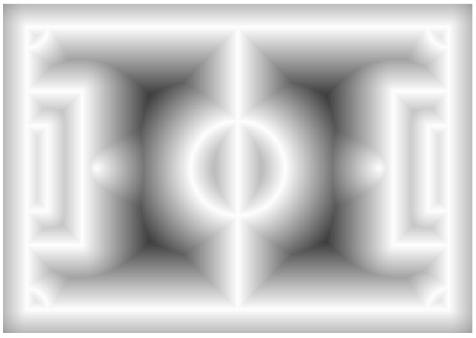


Figure 2.6: Illustration of a LUT map used by the algorithm. Darker areas indicate positions where the distance to the nearest line is large, while brighter areas indicate positions with smaller distances.

In this case, the sensor information is visual. At a given instant, the robot sees a set of points over the field lines. These points (with known distance and position relatively to the robot itself) are the landmarks used in the minimization error function. With a set of line points as in figure 2.7a, the error function estimates a position by estimating the error and error gradient on the current position according to the information on the LUT and iteratively try new positions in the direction with the least error gradient until a position with error low enough is found. Figure 2.7b plots the error function as gray levels, being dark areas the zones where the error is larger and brighter areas the zones of smaller error. The most probable position estimate (less error) is indicated by the black circle.

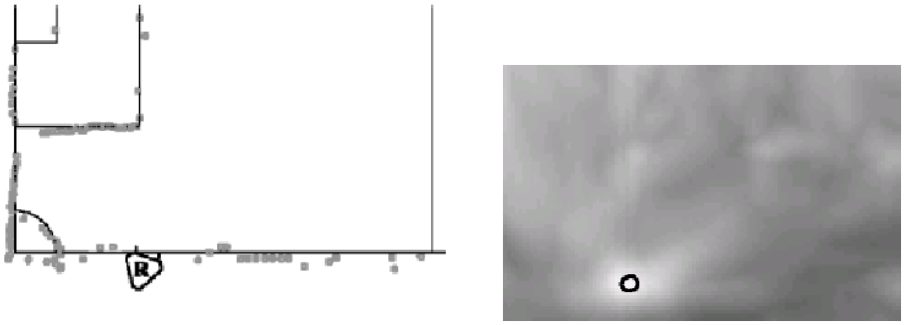
Filliat *et al.* [27] presents a review of localization strategies for map-based navigation, as well as several mapping methodologies.

2.2.2 Simultaneous Localization And Mapping (SLAM)

In plain localization problems, a map of the environment is available for the robot to observe and discover where it is. In SLAM problems, neither the map of the environment or the robot pose are available. In this case, only measurements $Z_{1:t}$ and controls $U_{1:t}$ are available and both the localization and the map need to be derived. It is a significantly more difficult problem as the robot has to acquire a map of the environment while simultaneously localizing itself within this map.

Mapping is the way the robot builds its representation of the world, its map. Typically, the construction of these maps are based on two methods [28]:

Metric or grid-based mapping. It is a representation based on the measures of the space



a) **b)**

Figure 2.7: Left **a)**: The set of line points (gray circles) and the field markings (solid lines) for a localization estimate. The estimated pose for the robot is indicated by the "R" object; Right **b)**: A gray level plot of the error function considering the pose estimation of the left-hand figure. Dark areas indicate positions with large error, bright areas indicate positions with small errors. Image from [24].

they map. In an indoor metric map, the information included could be the length of wall sections, widths of hallways, distances between intersections, and so on (figure 2.8). With this approach, the robot has to be equipped with sensors capable of estimating the distance to each detected object and it is usual to represent the mapped space with an evenly spaced grid. This is a quantitative approach and a path plan that could be applied over this kind of map could be "advance for X meters, turn θ degrees and advance other Y meters";

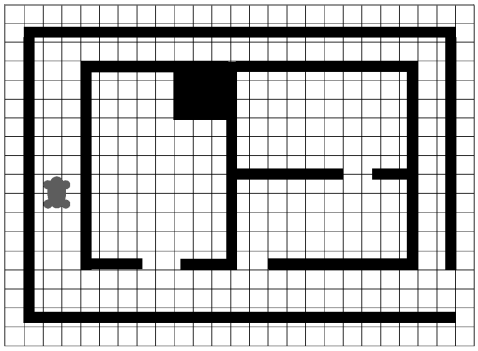


Figure 2.8: Illustration of a metric or grid-based map. All the known environment objects have known measures and positions.

Topological mapping. The representation of the space is not based on precise measurements but rather on landmarks (figure 2.9). In an indoor topological map, the information could include doors, hallway intersections, or T-junctions of hallways. To build a map based on landmarks, the robot would need to be equipped with sensors that could

identify them, typically vision systems. This is the most intuitive approach for humans, as we typically use path plans like "go straight till you see *this landmark*, turn left after it and follow until you reach *that landmark*";

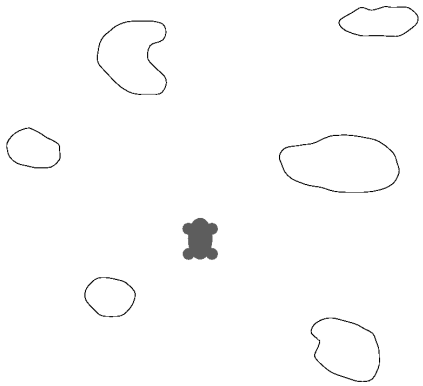


Figure 2.9: Illustration of a landmark scenario that could be used to build a topological map. The known landmarks are positioned relatively to each other, but no precise measures are known, navigation is only possible by sighting of the landmarks.

From a probabilistic perspective, the SLAM problem has two main forms.

One is known as *online SLAM*, which consists of estimating the posterior probability over the current pose along with the map: $p(X_t, m_t|Z_{1:t}, U_{1:t})$. It involves the estimation of the variables that persist at time t. It is common that online SLAM algorithms discard past measurements and controls once they have been processed.

The other main form is *full SLAM*, which consists of estimating the posterior over the entire path $X_{1:t}$ along with the map, instead of just the current pose X_t: $p(X_{1:t}, m_{1:t}|Z_{1:t}, U_{1:t})$.

Historically, the earliest SLAM algorithm designed is based on the extended Kalman filter (EKF) [12]. The EKF SLAM algorithm applies the EKF for online SLAM by using maximum likelihood data association. It is an approach that imposes a number of limiting assumptions: (1) the map has to be composed of point landmarks, preferably with as little ambiguity as possible, which may require significant attention to feature detectors; (2) as other Kalman algorithms, the assumption of Gaussian noise for both the robot motion and perception models needs to be met (or acceptable); (3) the algorithm can only process positive sightings of landmarks, no processing can be made based on the absence of landmarks.

Another example of approach for SLAM presented by Thrun *et al.* [12] is the Extended Information Form (EIF) algorithm. This EIF SLAM algorithm has in common with EKF SLAM the fact that it represents the posterior estimation by a Gaussian. However, unlike EKF SLAM, EIF SLAM solves the full SLAM problem. Thus, instead of defining the posteriors over the map and the most recent pose, it defines the posteriors over the map and the entire robot path. The EKF SLAM is therefore an incremental approach, it enables the robot to update its map without memory concerns, while the EIF SLAM is best suited for problems

where we want a map from an *a priori* known set size and we can afford to hold the data in memory up to the time where the map is built.

Several other algorithms are currently available, like fastSLAM [29], Postponement [30], relative map representations [31], submap methods [32] and Covariance Intersection (CI) based methods [33].

Madhavan *et al.* [34] use a distributed extended version of Kalman filter for cooperative multi-robot localization and mapping in outdoor environments, providing better estimates for each robot localization based on relative information of other robots poses as well as increased mapping capabilities since by sharing the own information, each robot can get unknown pieces of the map to fit into its own map and, in case of overlapping pieces, merge the information to get better estimates.

Some other algorithms and approaches for multi-robot simultaneous localization and mapping are presented by Rekleitis *et al.* [35], Mourikis *et al.* [36] and Thrun [37]. An application of multi-robot localization but applied to object localization, rather than self localization is presented by Stroupe *et al.* [38].

Julier *et al.* propose a hybrid approach on SLAM [39], with several filters implementing their own SLAM algorithm and with the ability of exchanging information between them, before generating the output.

2.3 Tracking problem

It is not uncommon in robotics to have the need to pinpoint and track a given mobile object on the scenario, and even keep track of it for as long and as precise as possible when it becomes undetected. This is usually addressed as a tracking problem.

Thus, using filtering techniques to track the state of a given object is a common application. However, given the uncertainty associated to this problem, the possibility of similar object states which can be ambiguous or even the need to track the state of several objects, there may be a need to extend the state representation. Several techniques are described by Bar-Shalom *et al.* [40] to address this problem.

2.3.1 Multiple Hypothesis Tracking

Multiple hypothesis tracking is explained by Bar-Shalom *et al.* [40] as the optimal solution for the Multitarget Multisensor tracking problem, which consists on tracking various objects states through the use of a set of sensors and dynamic models.

Being the objective of any filter to keep the internal state of one single given object, the use of a single filter is not feasible for keeping the state of several objects. Thus, the idea of multiple hypothesis tracking is to apply a framework of individual filters which keep several hypothesis for the system state, according to the defined focus. Even though multiple hypoth-

esis tracking is particularly fit for multitarget tracking, we may want to keep several possible ambiguous states for a single object, for instance, until there is a better degree of certainty available to disambiguate. Hypothesis can be either created or destroyed dynamically as new sensor information becomes available that supports the operations.

This approach has been used successfully for mobile robot localization, by keeping track of several possible poses for the robot while there is not enough data to understand which one approximates the real robot pose [15, 41]. Multiple hypothesis tracking has also been applied in human figure tracking by Cham *et al.* [42]. An application of multiple target tracking is presented by Arambel *et al.* [43], where several ground targets must be tracked from a video based sensor on board of an Unmanned Aerial Vehicle.

2.3.2 Multiple Model Tracking

Multiple model approaches are based on the fact that the behavior of a target varies through time and thus cannot be characterized by a single model. However, one can have different models to characterize the object in several situations. The idea of these approaches is to possess a bank of different models of the object dynamics for different regimes and an underlying switching function. The main difference to the multiple hypothesis methods is that, in the later we have several instances of a same filter with the same models, both in terms of system evolution model and sensor model. The multiple model approaches do not have this limitation and allow the use of distinct modeled filters for distinct situations, at the cost of a higher complexity for guaranteeing that each model is used in the correct situation so that the output data is kept in coherent states.

Interacting Multiple Models (IMM), originally proposed by Blom *et al.* [44] is one such algorithm, which achieves a good compromise between performance and complexity with a Markovian switching function [45]. An application example of this algorithm is presented by Vacher *et al.* for an aircraft control application [46], where it was proven to be a suitable solution. Semerdjiev *et al.* test implementations of IMM for tracking of a maneuvering ship, for both adaptive and non-adaptive switching functions [47]. Ceranka [48] analyses the performance of an IMM in a pedestrian location problem, comparing it to an Extended Kalman filter and an extension of that filter for colored noise. Furthermore, Civera [49] successfully applied an Interacting Multiple Model to the Simultaneous Localization And Mapping (SLAM) problem using a monocular vision system .

Although usually associated with Kalman filter and variants or Probabilistic Data Association [50], Wang *et al.* [51] presented an application of IMM using particle filters as the algorithm building blocks. According to their results, IMM particle filtering can achieve even better results than classical particle filter when visually tracking an object moving with very distinct movement types through time.

2.4 Fusing visual and inertial sensors

One recurrent problem is the fusion of visual and inertial sensors, where some results have demonstrated that the visual tracking of objects may work at higher velocities and be more robust if combined with information coming from inertial sensors [52] and also that ego-motion estimation can be more precise and navigation more robust using these approaches [53]. This is due to the fact that visual tracking can be very effective when moving at low speed, but degrades for scenarios with increased velocity. Inertial sensors on the other hand, are not effective for measuring movement at low speed, but can get accurate results at relatively high speeds. Thus, the combination of inertial sensors with a visual system can improve the visual identification, by keeping a better insight of how the system is moving and adapting the visual algorithms accordingly.

Lobo and Dias [54] present a framework for combining visual and inertial sensor information, by considering visual and inertial vertical features and combining their information. Several papers of their work have been published [55, 56, 57] demonstrating several stages, results and approaches. Furthermore, Corke *et al.* present a description of the fundamentals of inertial and visual perception as two important senses for biological and robotic systems [58]. For both, they present the principles, capacity and limitations as well as an analysis of how both senses can complement themselves and the principles for their fusion.

Mirisola and Dias [59] describe the tracking of a ground target from an unmanned airship equipped with a camera, an Inertial Measurement Unit and a GPS. They refer the application of Kalman filter to the recovered trajectories of both the camera and the target.

2.5 Summary

In this chapter, an overview of several methodologies and algorithms for sensor and information fusion were presented. Each presented filtering algorithm has examples of use within several problems where sensor and information fusion is applied. Some of the presented problems were tackled during this PhD within specific robotic scenarios and the concepts will be referred when needed.

Chapter 3

Robotic soccer and worldstate

The test scenario chosen to implement and test the objectives of this PhD work is robotic soccer, which is aligned with the development of teams of robots in our group. Particularly, the initiative where we work to accomplish this is the RoboCup initiative.

Robotic soccer appears as a complex scenario for study and development of both software architecture and sensor and information fusion, since one has to cope with a team of individual mobile robots that autonomously need to build a model of their own world which should be similar to the models of the team mate robots for them to work properly as a team.

In this chapter, a general overview of the target RoboCup environments is presented, as well as an overview of the used robots. Focusing on the world model, an overview of general approaches used by other teams is also presented, as well as an insight on the structure that we find mot useful for scenarios of robotic soccer.

3.1 Robocup

The RoboCup is an international joint project intended to be used as a vehicle to promote robotics and AI research, by offering a publicly appealing, but formidable challenge. One of the effective ways to promote engineering research, apart from specific application developments, is to set a significant long term goal.

The ultimate goal of the RoboCup Initiative is:

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

Although this goal may sound overly ambitious given the state of the art technology today, the definition of an ambitious or even overly ambitious goal is common in several projects, to foster the development of subgoals and technologies that, while not directly related with the ultimate goal, will nevertheless be crucial for reaching it.

Still within the RoboCup objectives, the initiative also aims to be a standard problem for evaluation of various theories, algorithms, and architectures. The typical example provided by RoboCup federation is the chess problem. It has been used for the last decades as a standard problem to evaluate several algorithms and theories. Having reached its ultimate goal of defeating a human champion of chess, this challenge is close to a finale. RoboCup project aims to provide a new challenge to, such as chess did before, foster the development of next generation technologies.

Despite the initial scenario being soccer, RoboCup currently counts with leagues in several areas, such as search & rescue, domestic or factory leagues. RoboCup's rules change in order to promote advances in the science and technology of robots and to make the league challenges closer to real world, rather than to impose artificial setups to improve league specific performance. They are reviewed annually by each league technical committee, and discussed with participants and other knowledgeable researchers in the field to draft out new rules.

Even within the soccer domain, there are several leagues, each of them with the objective of allowing development in more specific areas. Simulation leagues, for instance, are most used to develop team play while the real robots are still not physically stable enough to explore team play at its finest. The main leagues are currently the following:

- RoboCupSoccer
  - Simulation League
  - Small-Size League
  - Middle-Size League
  - Standard Platform League
  - Humanoid League
- RoboCupRescue
  - Rescue Simulation League
  - Rescue Robot League
- RoboCup@Home
- RoboCupJunior

Most of the work presented in this document is applied within the RoboCup scenario, particularly the Middle Size League and Standard Platform League.

3.1.1 Middle Size League (MSL)

In this league, middle-sized robots with a maximum size of 52x52x80 cm play soccer in teams of up to five robots with a regular size five FIFA soccer ball on a field similar to a scaled human soccer field with 18x12meters (figure 3.1).

Figure 3.1: Picture of a MSL game in IranOpen2014.

All sensors are on-board and robots can use wireless networking to communicate among them and with an external agent acting as a coach running on a base station PC, within strict limitations on traffic volume. The coach, however, does not possess any type of sensor, it may perform tasks and decisions but only based on the information provided by the field robots. The referee team has a PC running a *referee box* application which is used to send the game signals to the robots through the network. These signals, however, are not sent directly to the robots, they are instead sent to the external PC of each team, the same that can run the coach and is wired to the field network, and each team is responsible for relaying the information to their robots. Additionally, there can be no ad hoc wireless networking, all communications between robots and between robots and the base station PC must be established through one of the field access points (figure 3.2).

The field is composed of several elements of known color. The floor is green, the lines are white, the goals frames are also white and the robots must have a black band around them at their lower level. The ball color may vary in each tournament, being defined a-priori by the organization, with the restriction of having a saturated main color which is different from the rest of the elements on the field.

The official field markers are presented in figure 3.3, drawn with official proportions for all the dimensions, in meters.

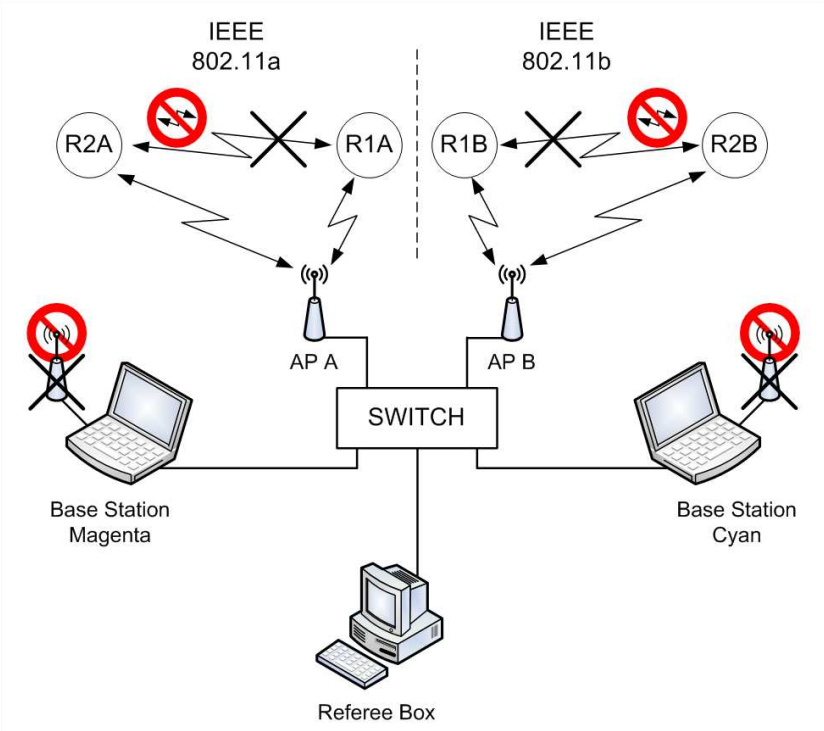


Figure 3.2: MSL networking diagram

The research focus is on full autonomy and cooperation at plan and perception levels, generally using wheeled robots.

3.1.2 Standard Platform League (SPL)

In this league and for the rules of 2012, small humanoid robots play soccer in teams of up to 4 robots with a street hockey red ball on a field similar to a scaled human soccer field with 6x4 meters (figure 3.4).

As in MSL, all sensors are on board of the robot and they also can communicate among them through a wireless network but no outside agent can communicate with the field robots. All communications between robots must be established through the field access point and no ad hoc connections are allowed. The refereeing team has a PC connected to the field network running an application referred to as *GameController* which sends the referee commands directly to the robots.

The floor is also green and the lines are white. The ball is red and the goal frames are yellow.

The official field markers (2012 edition) are presented in figure 3.5, drawn with official proportions for all the dimensions, in meters.

All teams use identical (i.e. standard) robots. Therefore the teams concentrate on software

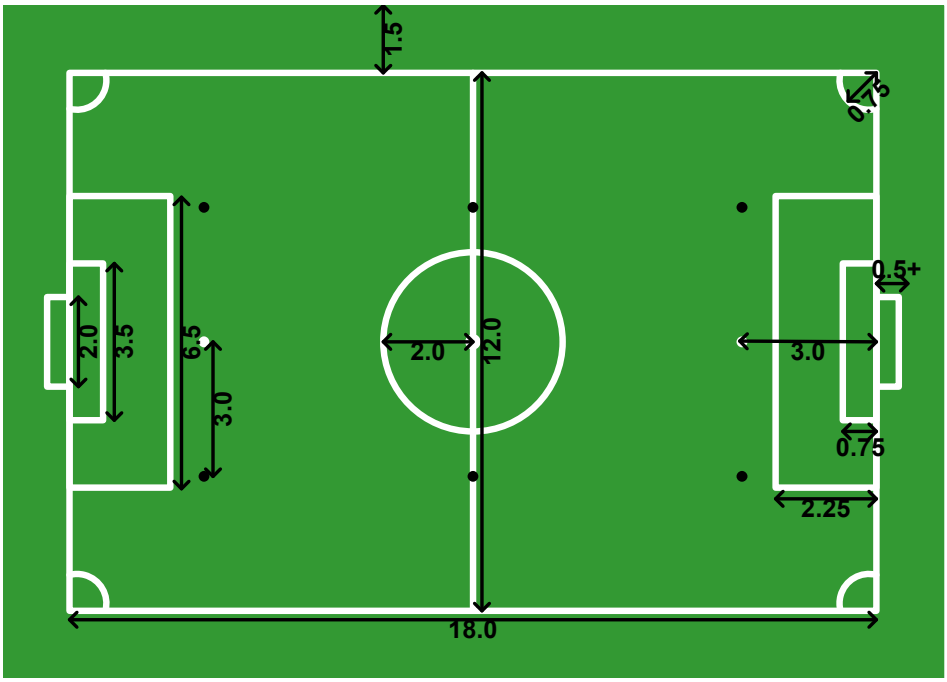


Figure 3.3: Official MSL field marker distances. All the presented values are in meters. The field representation is with official proportions.

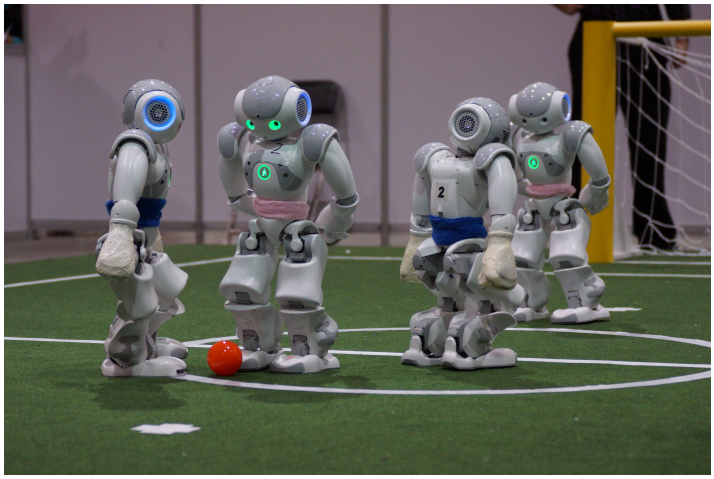


Figure 3.4: Picture of a SPL game in RoboCup2012.

development only, while still using state-of-the-art robots. Omni directional vision is not allowed, forcing decision-making to manage vision resources between self-localization, ball localization and general object localization (other robots, goals...).

Given the current state of technology concerning humanoid robots, the research focus of this league is still mainly in body control and development of methodologies to allow the robots to be able to move in a stable way in any action they take. The adoption of other league's

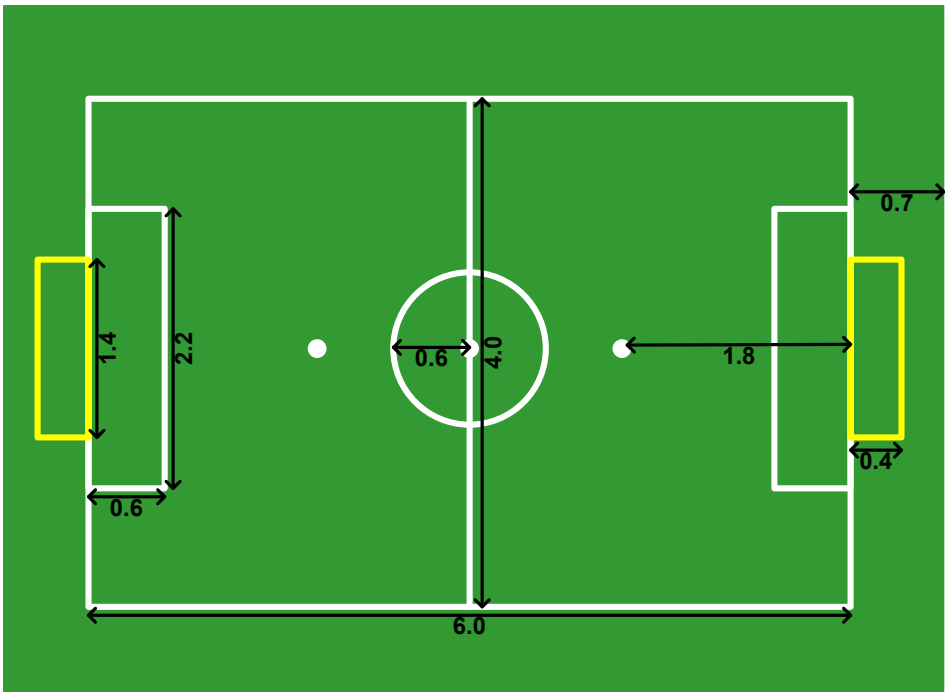


Figure 3.5: Official SPL field marker distances for 2012 edition. All the presented values are in meters. The field representation is with official proportions.

cooperation methodologies has already occurred, as it is our own case, but the effectiveness of such methodologies is understandably lower, due to the instability of the physical platform.

In 2008, the league replaced the highly successful Four-Legged League, based on Sony's AIBO dog robots, and is now based on Aldebaran's Nao humanoids.

The dimensions and general regulations presented are in its 2012 edition version, since that was the last participation of the Portuguese Team. In the 2013 edition there were changes relative to the size of the field and the number of robots in a game.

3.2 Description of used robotic platforms

The work developed was mainly within the presented leagues of RoboCup, which use different robots. The used platforms for both leagues are presented in some detail, with the exposition of the more relevant characteristics, both in terms of hardware or software.

3.2.1 CAMBADA

CAMBADA, acronym for *Cooperative Autonomous Mobile roBots with Advanced Distributed Architecture*, is a Middle Size League Robotic Soccer team created by the ATRI[1]

[1] Actividade Transversal em Robótica Inteligente - Transverse Activity on Intelligent Robotics

group, of the IEETA[2]. Created in 2003, it has been a project that includes many people's efforts in several areas, for building the mechanical structure of the robot, its hardware architecture and controllers and the software development in areas such as image analysis and processing, sensor and information fusion, reasoning and control. Operating in the RoboCup competitive environment, several concepts on robot construction (at all levels: mechanic, hardware and software) have been seen and discussed. This discussion of ideas promotes the development of the involved teams. The CAMBADA robots have obviously evolved over the years and for RoboCup2013 a new platform was built. Most of the work presented in this document was accomplished mainly over the previous platform. Also, most of it is not dependent on being run on the old or the new platform except when concerning to execution time and processing performance, since the new platform has a more powerful processing unit. A brief description of the general platform [60] is presented in this section, with some of the differences between both platforms identified when relevant.

Some common structural approaches of MSL teams: The mechanical structure of the MSL teams[3] is variable. The shape itself is different among teams, as there are some using robots with triangular [61, 62, 63, 64], circular [65] or squared [66, 67] shapes, but they can assume any shape (as long as they stay within the allowed size).

The used locomotion method is one of the most important issues in this kind of robot. Some teams use a differential movement approach while most use a holonomic approach. The latter one has the great advantage of allowing the robot to move in any direction with any orientation.

The kicking system is also known to have different approaches, either pneumatic [61], electromagnetic [63] or electro mechanic (with springs) [62].

The vision system has several implementations as well, being the most popular and adopted by most teams an omni directional system [61, 63, 67], implemented in several different ways but commonly with a camera pointed up to a convex mirror on the top of the robot.

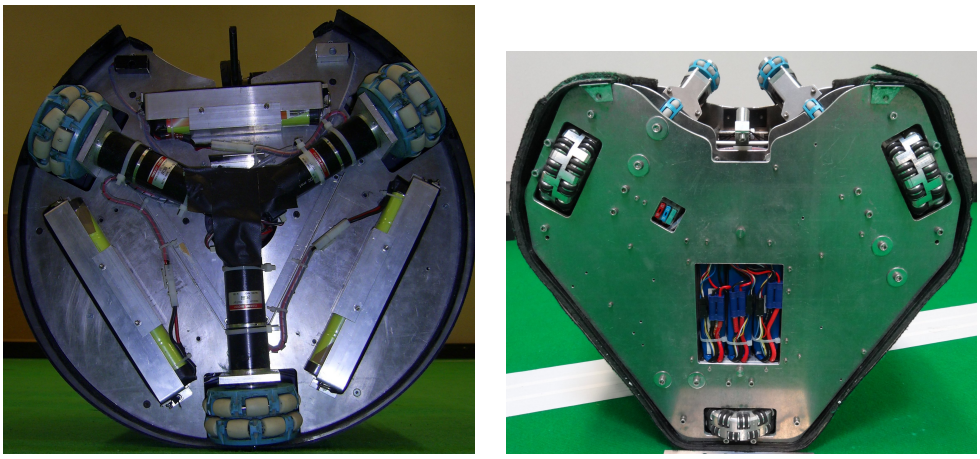
3.2.1.1 Physical structure

The CAMBADA robots were designed and completely built in-house. The base structure of the robots is not very different from one platform to the other. With the old platform, each robot was built upon a circular aluminium chassis (with roughly 485 mm diameter),

[2]Instituto de Engenharia Electrónica e Telemática de Aveiro - Aveiro's Institute of Electronic and Telematic Engineering

[3]MSL team description papers (TDP) describing their robots can be found in each of the MSL pre-registration material, with pointers in:
http://wiki.robocup.org/wiki/Middle_Size_League/2014_Qualification_Results

which supported three independent motors (one for each holonomic wheel, allowing for omni directional motion), an electromagnetic kicking device and three NiMH batteries (figure 3.6a). The new platform has chassis based on a triangular shape where the motors and wheels structure is supported by two aluminium plates, being the wheels now completely disconnected from the motor axis. This "extra" layer allows a decoupling of the locomotion mechanics and electronics from the rest of the mechanics and control electronics, which makes the new platform even more modular. The main structure of the robot is basically the same, with all the mechanics and control electronics inside the bottom base. The batteries are still three but are now LiPo technology (figure 3.6b).



a) **b)**

Figure 3.6: Pictures of **a)**: the old CAMBADA robot base, where the holonomic wheels, motors and batteries are visible. **b)**: the new CAMBADA robot base, where the motors are now protected inside an intermediate layer and uncoupled from the wheels.

The remaining parts of the robot were placed in three higher layers, namely

- The first layer upon the chassis is used to place most of the electronic modules such as kicker controller and the kicker and grabber mechanisms (figures 3.7a,b).

- The second layer contains the main computation unit, which was a PC, 13" notebook based on an Intel Core2Duo processor with 2GHz clock speed, 1GB RAM that recently changed to a notebook PC, 12" Intel Core i5 with a 2.7 GHz quad core processor, 8GB RAM and the support tripod for the third layer (laptops visible in figure 3.8).

- On the top of the robots stands an omni directional vision system consisting of a camera pointing to an hyperbolic mirror and an inertial unit, currently used for localization purposes (figures 3.7c,d). In the old version there was also a frontal vision system, consisting of a camera pointing forward.

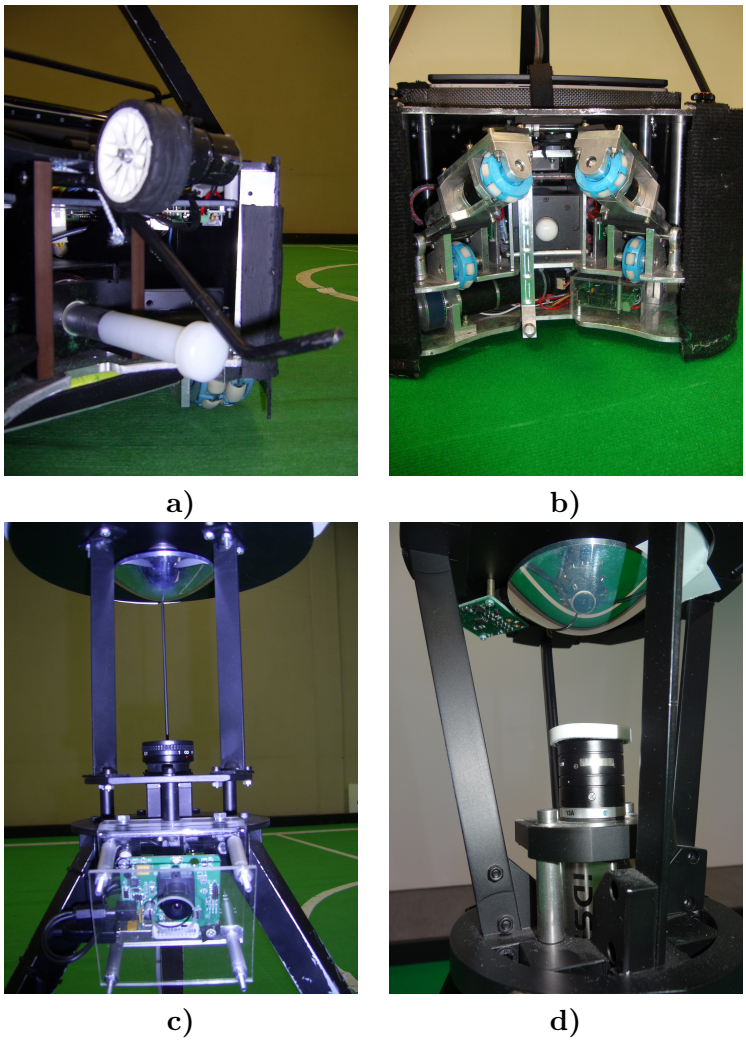


Figure 3.7: Pictures of **a)**: the old version kicker and grabber systems. **b)**: the new version kicker and grabber systems. **c)**: the vision systems of the old version, with both the omni and a frontal camera. The compass is on the back of the frontal camera support. **d)**: the vision system of the new version, with the inertial unit board visible on the left of the mirror. In both pictures, the camera and the hyperbolic mirror it points to are visible.

The mechanical structure of the robot is highly modular and was designed to facilitate maintenance. It is mainly composed of two tiers: 1) the mechanical section that includes the major mechanical parts attached to the aluminium plate (e.g. motors, kicker, batteries); 2) the electronic section that includes control modules, the PC and the vision systems. These two sections could be easily separated from each other, allowing an easy access both to the mechanical components and to the electronic modules [68]. On the new version of the robots, we can consider that the "mechanical layer" is further divided in two layers, separating the locomotion hardware from the ball handling hardware (kicker and grabber), since the maintenance interventions usually only target one type of components.

A general overall picture of both platforms is presented in figure 3.8.

a) **b)**

Figure 3.8: The CAMBADA robots. **a)** The old platform. **b)** The new platform.

3.2.1.2 General architecture

The general architecture of the CAMBADA robots has been described in [69, 70, 71]. Basically, the robots architecture is based on a main processing unit that controls the higher-level coordination, i.e. the coordination layer. A PC is used as this main processing unit, that processes visual information gathered from the vision systems, executes high-level control and coordination functions and communicates with the other robots. This unit also gets the sensing information and sends actuating commands to control the robot actions using a distributed lowlevel sensing/actuating system. The PC runs the Linux operating system. The communication among team robots is guaranteed by an adaptive TDMA transmission protocol [72, 73] on top of IEEE 802.11, built to reduce the probability of packet collisions between team mates, becoming more efficient. The protocol is used to support a Real Time DataBase (RTDB) that allows team mates to share information on some variables of interest [74].

Hardware control The low-level sensing/actuation system of the CAMBADA robots (figure 3.9) is implemented through a set of microcontrollers interconnected by a network. Controller Area Network (CAN) [75], a real-time fieldbus typical in distributed embedded systems, has been chosen for the task.

The low-level sensing/actuation system executes five main functions, namely, Motion control, Kicking, Grabbing control, System monitoring and Inertial unit. These functions are distributed over a series of controller boards that may be dedicated for a single task or not. The Motion control function provides holonomic motion using 3 DC motors actuating over

holonomic wheels. The Kick function performs the control of an electromagnetic kicker and the Grabbing control is responsible for a ball handler to dribble the ball. The System monitor function monitors the robot batteries as well as the state of all nodes in the low-level layer. Finally, the Inertial unit function reads the angle from magnetic north given by the electronic compass and the accelerations and angular velocities provided by the accelerometers and gyroscopes and estimates the robot orientation. The low-level control layer communicates with the coordination layer through a gateway, which relays the messages between the several modules to the PC [74].

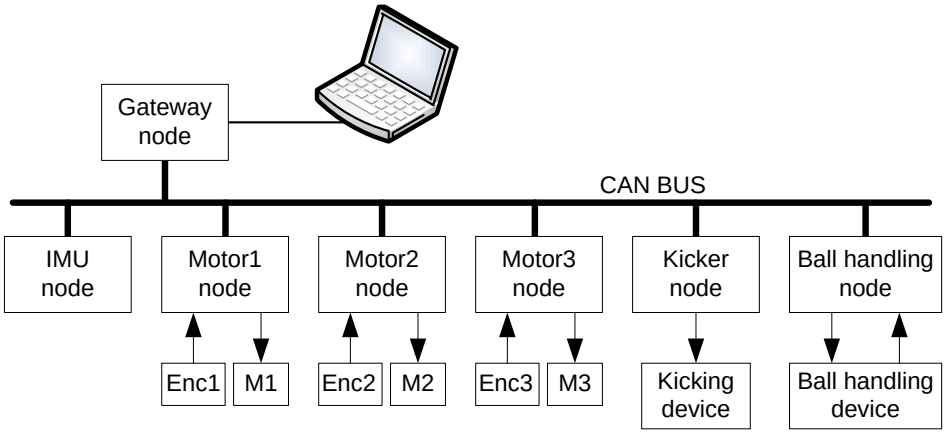


Figure 3.9: Hardware control architecture diagram.

Software The software system in each robot is distributed among the various computational units. High level functions are executed on the PC, while low level functions are executed on the micro controller network described before. A cooperative sensing approach based on a Real-Time Database (RTDB) [69, 72, 76] has been adopted. The RTDB is a data structure where the robots share their world models. It is updated and replicated in all players in real-time. The high-level processing loop starts by integrating perception information gathered locally by the robot, namely, information coming from the vision system and information coming from the low-level layer. The available information (both the one directly gathered and the one estimated through processing) is afterwards stored in the local area of the RTDB. The next step is to integrate the robot local information with the information shared by team-mates, disseminated through the RTDB. The RTDB is then used by another set of processes that define the specific robot behaviour for each instant, generating commands that are sent down to the low-level control layer [74]. Several processes are running on the PC, each with a given responsibility. These processes communicate among them by using specific RTDB data structures in a black board approach, and are depicted in figure 3.10.

The processes interacting with the RTDB are:

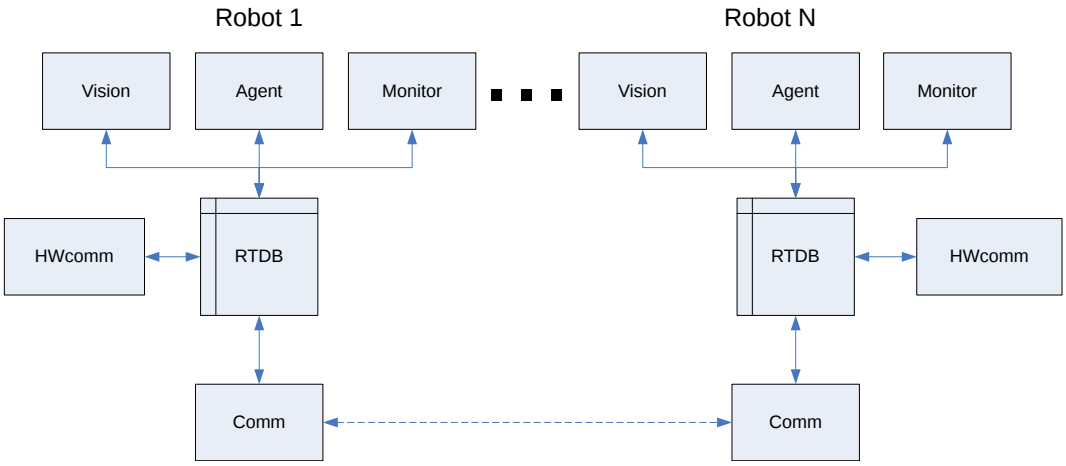


Figure 3.10: Running processes diagram.

- **vision** is the first one to be executed at each cycle. It is actually the process that defines the cycle time, as it is limited to the omni directional camera frame rate. It is responsible for retrieving the cameras frames and make all the image processing. The process has several steps, from color segmentation to object identification and relative position estimation. The processed values are then injected in the RTDB.

- **hwcomm** is the hardware comunication process. It runs with two threads, one to get information from the hardware micro controllers and one to send commands to those micro controllers so they execute them through the actuators in the low-level.

- **comm** is the wireless communication process, responsible for transmitting the RTBD information relevant for and from team mates. It also runs with two threads, one for sending information, one for receiving.

- **agent**, control process that integrates the information and decides which actions to take and which orders to send to the actuators.

- **monitor** is a monitoring process running with a lower cycle and is constantly verifying if all the other processes are still running, both by checking if the processes themselves exist and by verifying if they have been active writing on the RTDB. In case any of the processes have failed, it starts them again so that the whole system keeps working. This process is used during the games only, when it is of best interest that the robots keep working without having to take them out for manual restart.

3.2.1.3 CAMBADA vision overview

The *vision* process analyzes the current image and fills its RTDB structures with the positions of the detected objects relative to the robot. Since the analysis is made for 3D object on a 2D image, a projection of the object center to the ground plane must be considered. This process is responsible for handling the omni directional camera, which is the main sensor of the robots. Although the *vision* process is a well defined piece of existent software, it will be subject to further analysis in section 4.1, as some changes to information provided by this process were introduced during this PhD.

Examples of frames by both the omni directional and frontal vision systems are presented in figure 3.11.

a) **b)**

Figure 3.11: Vision frames **a)**: Omni directional vision frame; **b)**: Frontal vision frame.

The frontal vision system is a "secondary" system, used only for checking for the ball when it is not visible by the omni directional system, particularly when it is not on the ground. A *frontVision* process is responsible for using the extra camera, which runs on best effort over the base processes. Thus, this process is not represented in the base diagram, as it is completely optional. The *frontVision* process is analyzed in section 4.3.3 since it is being introduced on the team as part of this PhD work.

3.2.1.4 CAMBADA agent overview

The CAMBADA agent is the process that, at each cycle, is responsible for the high-level control, which is divided in several stages. The first stage is the sensor fusion, executed by the *Integrator*, with the objective of gathering the noisy information from the sensors and from its team mates and update the state of the world that will be used by the high-level decision and coordination.

The strategic positioning on the field is then defined, in order for the robots to be able to maintain a given formation.

The third stage is for the agent to decide how to act given the state of the world he built. At the higher level, it assumes a *Role*, and operates on the field with a given attitude, for example, as role Striker. The actions it can take are defined by lower level *Behaviors*, which define the orders to be sent down to the actuators in order to fulfill a task, for example, a Move behavior, to get to a given point on the field.

All the measures of the field, controller parameters on the behaviors and several configuration values are available in a file *cambada.conf.xml*. This allows a quick, easy and organized way to reconfigure many aspects on the agent, for example, the field size.

The movement controller is an implementation of a PID controller. However, some behaviors may introduce some restraints to the output of the PID, to guarantee a movement as desired, in the form of acceleration and deceleration ramps. The restrictions are mainly applied in movements in which the robot needs to control the ball and thus cannot turn too fast or brake too abruptly, for example, so that it won't loose control over the ball.

3.2.1.5 Basestation

In a game of the Middle Size League, besides the agents of the 5 robots on the field, there is an additional agent outside the field which is connected to the referee box application and is responsible for managing the team as a whole. It is responsibility of this agent to relay the signals sent by the referee to the team robots. Additionally, a coach agent may also run on this agent and take decisions and communicate with the field agents, but it can only work with information gathered and sent by the field agents, it cannot possess any type of sensor.

In our team, we refer to the PC that is connected to the referee box as basestation. However, this PC runs three or four processes during a game, which are:

- **comm** is the same process that runs on the field robots but in this PC using the wired interface.

- **coach**, the process that performs an analysis of all the information provided by the field robots. Currently, the coach estimates the cover positions for all the players during the execution of set pieces from the opponent team.

- **basestation** is a control, configuration and debug graphical tool for the team. Further details will follow.

- **logger** is a process that reads and saves all the information gathered by the basestation PC. Tipically it is used to log the games for further analysis but, being a completely independent software, it can be run at anytime to save the information received.

From the processes previously described, the *basestation* is worth a further mention and description, given the importance it plays on the team.

The *basestation* is a graphical application created with the purpose of being the main interface of the team (figure 3.12). It is a tool used both in game and during development, as it is a necessary piece to configure and activate the robots. The definition of the team color and goal side are the main configuration parameters that must be provided by the *basestation*, as well as a "virtual ON switch". The team strategic positioning can also be defined manually through this application, from the set of formations available at the time.

Figure 3.12: The *basestation* application.

Additionally, and during development and testing, we can define a fixed role for each robot and simulate the referee signals to test and evaluate all the possible game situations.

In any of the situations, the *basestation* is constantly displaying the information exchanged by the robots in a visually clear way, by drawing all the relevant information of the robots on a field and also by displaying the individual information of each robot, so that the users can detect any anomaly in their states.

3.2.2 NAO

NAO is a programmable humanoid robot developed by Aldebaran Robotics, a French robotics company headquartered in Paris. The robot's development began with the launch of Project Nao in 2004, but it was in 2007 that Aldebaran won the contest for providing

the robots for RoboCup Standard Platform League teams. The NAO robots replaced the previous discontinued Sony AIBO platform in this league.

Since 2008, year of the public release of the robots, it has been used by the SPL teams in a robotic soccer scenario and by research units all around the world. Aldebaran produces several versions of these robots, some only with torso and arms, for interaction purposes. The Nao was used in RoboCup 2008 and 2009, and the NaoV3 was chosen as the platform for the SPL at RoboCup 2010. The Nao Academics Edition was developed for universities and laboratories for research and education purposes. It was released to institutions in 2008, and was made publicly available by 2011. In December 2011, Aldebaran released the Nao Next Gen, featuring enhanced software, a more powerful CPU and HD cameras.

3.2.2.1 Physical structure

The version of the NAO robots over which the work described in this document was done is version 3. Its body has a total weight of 4.5 kg and a total height of 0.57 m and uses a Lithium-ion battery as power source. It is equipped with a series of sensors and actuators which features:

- 21 DoF
- 2 VGA cameras
- 4 microphones
- 2 speakers
- 2 sonar sensors
- 1 inertial unit (3 accel, 2 gyro)
- 2 bumpers (1 each foot)
- 8 pressure sensors (4 each foot)
- A processing unit placed on the robot's head with a 500MHz geode processor and 256 MB RAM, running a Linux based Operating System from a USB memory stick.

The previously listed features are the ones from the RoboCup version of the NAO, which is not a "full" version, since the Academic Edition has a few more sensors and actuators.

A general overall picture of a NAO robot is presented in figure 3.13a, with some of its features highlighted, and a schematic of its kinematic structure in figure 3.13b

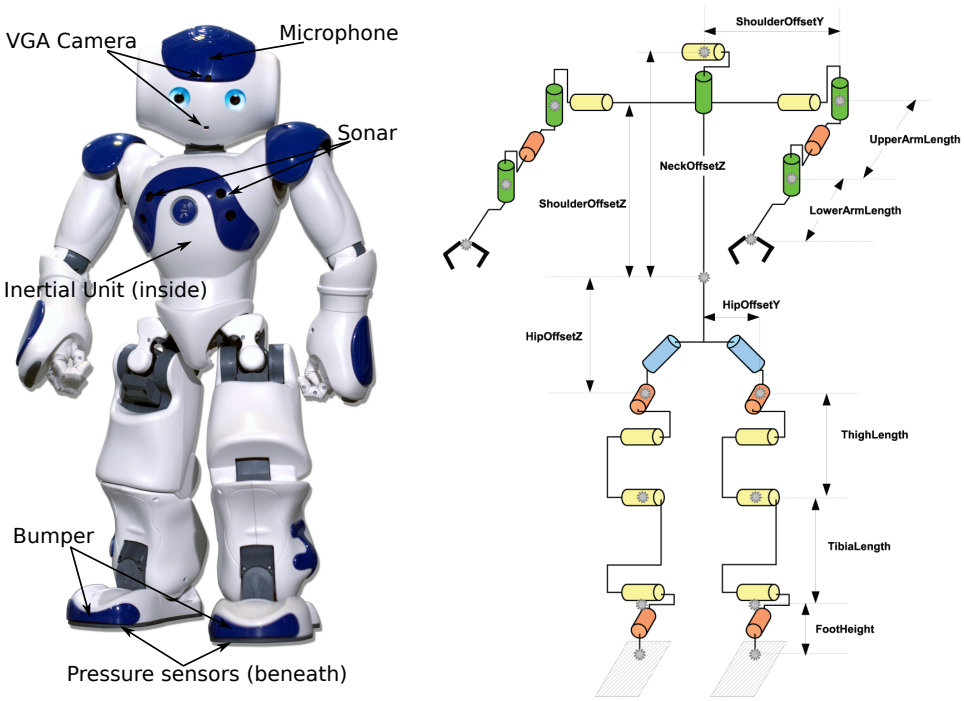


Figure 3.13: In **a)** a picture of a NAO robot with some highlighted features. In **b)** a schematic of the kinematic structure of the motor actuators of the NAO. Image from [77].

3.2.2.2 General architecture

The general architecture of the NAO robots is based on a main processing unit located on the head that, like in the CAMBADA, processes visual information gathered from the vision systems, executes high-level control and coordination functions and communicates with the other robots. It also gets sensing information and sends actuating commands to the robot through a lower level sensing/actuating system.

Hardware control The control layer of the NAO is based on an ARM7-60MHz, located in the torso, that distributes information to all actuator modules through a RS485 bus. The actuator module microcontrollers are based on Microchip 16 bit dsPICs and are divided into two buses, one for the upper part of the body and one for the lower part. The ARM7 controller communicates with the main CPU through a USB-2 bus [77] (figure 3.14).

The actuator modules are responsible for servo control, bus control, sensor management and power converters. Each circuit can drive up to two actuators and each actuator is equipped with a magnetic rotary encoders (MRE) that yield absolute outputs. Besides the main servo motors, there are some more dsPIC controllers managing other elements of the robot. There is also a pair of them on the RS485 bus controlling the feet LEDs and acquiring

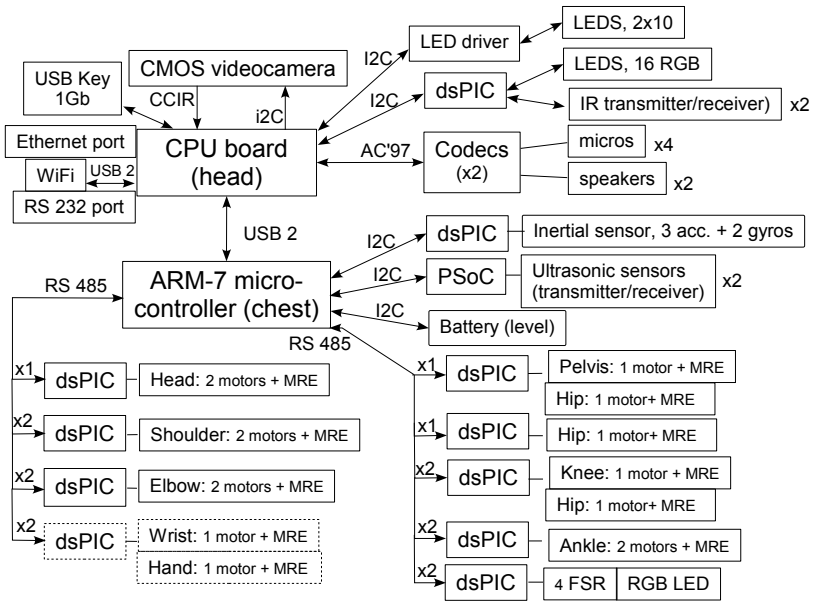


Figure 3.14: NAO hardware control architecture. MRE stands for Magnetic Rotary Encoder. Image from [77].

the data from the force. Finally there is one, connected to the ARM7 board through I2C bus, which is devoted to the signal acquisition from two gyrometers and three accelerometers and another one manages an infrared transmitter/receiver and a series of LEDS.

Software The software architecture of the NAO robots is based on an underlying embedded software framework, the NAOqi. It is a modular and distributed environment that can deal with a variable number of executable binaries, depending on the users architectural choices, and is event-driven [77].

This framework is running on the Operating System and includes a whole set of modules provided by Aldebaran to control and interact with several features of the robot. These modules are divided in several categories like Core, Motion, Audio, Vision, Sensors, Trackers, DCM, among others.

Like many of the teams which aim to research in several topics of humanoid robotics, we have custom software for most of the needed tasks. In our work version most of the modules from NAOqi are not used, only Core NAOqi and some monitoring modules which need to be running just to allow interaction through the power button and guarantee some security for the motors. The main module used is the Device Communication Manager (DCM) which is a NAO real time module that is part of the NaoQi system, and is linked as a library. It is in charge of the communication with the electronic devices of the robot except sound sensors and video cameras.

The DCM and other modules/processes deliver information to each other through the *ALMemory*, a shared memory of the NAOqi. The actuating commands are placed on the shared memory with an absolute time value based on the system time and the DCM will synchronously compute the appropriate command to send to each actuator using linear interpolation. The sensor information is, likewise, periodically placed in the shared memory by the DCM for the remaining modules to access. The period of information update of these modules is 10 ms.

3.2.2.3 Vision overview

The architecture of the vision system can be divided into three main parts: access to the device and image acquisition, calibration of the camera parameters and object detection and classification [78]. Moreover, apart from these modules, two applications have also been developed either for calibrating the colors of interest (*NaoCalib*) or for debugging purposes (*NaoViewer*). These two applications run on an external computer and communicate with the robot through a TCP module of the client-server type that we have developed. The current version of the vision system represents the best trade-off that the team was able to accomplish between processing requirements and the hardware available in order to attain reliable results in real time.

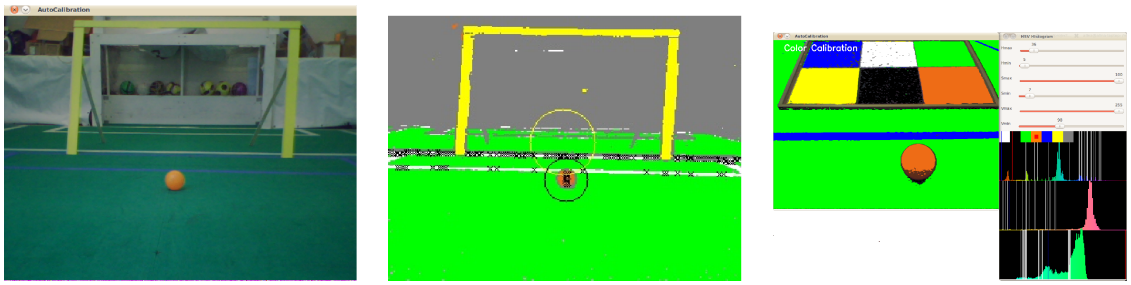
Having the possibility of running the *vision* module as a server, the two applications that we have developed, *NaoCalib* and *NaoViewer* can act as clients that can receive, display and manipulate the data coming from the robot. Thus, *NaoViewer* is a graphical application that allows the display both of the original image as well as the corresponding index image containing the validation marks for each object of interest that was found. This application is essential in terms of understanding what the robot "sees" since NAO does not have any interface to allow the connection of a monitor, all display related operations must be performed remotely by a PC with graphical processors and monitor. *NaoCalib* is a very helpful application that we developed for the calibration of the colors of interest (figure 3.15).

The *vision* final results are the positions of the several items of interest (ball center, points on field lines, goal posts) in pixel coordinates on the acquired image. This information is then used by the *agent*.

3.2.2.4 Agent overview

Prior to this PhD work, the agent architecture was a monolithic approach, with a set of functions. It would start by getting the information provided by the vision and start a decision function that would execute a series of actions.

The behaviors executed were only relative to the ball, using directly its pixel coordinates, by trying to move in a direction that kept the horizontal coordinate at half the image size.



a) **b)** **c)**

Figure 3.15: In **a)**, an image captured by the NAO camera. In **b)**, the same image with the colors of interest classified. The marks over the color blobs represent the detected objects. In **c)**, an example of the classification of the colors of interest, by means of the *NaoCalib* application.

The agent process and architecture and the general software architecture, were object of development during this PhD and thus will be analyzed and described in detail in section 5.1.

3.3 Worldstate for robotic soccer

A representation of the world state is a need in almost every scenario with autonomous robots that do not work based on a reactive architecture. In robotic soccer, as in many other scenarios involving any kind of object manipulation, the need for a deliberative architecture is evident.

When building a representation of the surroundings of a robot in the scenario of robotic soccer, and although the work described in this document was applied to different leagues, there are some basic entities that will exist and must be considered for the robots to be able to play.

3.3.1 Common approaches

Different leagues may have different typical approaches, on world modeling methods. We can, however, find a limited number of base techniques for the several world modeling problems, being localization the most documented of these problems.

Many teams use Kalman filter, Particle filter and their common variations as base for their work and implement some specific tweaks in order to obtain better results. By better, in this case, we do not mean necessarily more precise, but usually with a better precision/runtime relation.

In the MSL, concerning localization, several teams use the tribot algorithm as a base, as reported in their Team Description Papers [79, 80]. There are some teams that use different approaches, such as a combination of Monte Carlo Localization (through the use of a

Particle filter) and dead reckoning used by the Hibikino team [81]. Mu Penguins state that they use Hough transform and pattern matching to perform localization [82]. The SocRob team [65] proposed a method to cooperatively perform localization based on visually shared obstacles [83] that have recently been extended by applying the Mixture Proposal Distribution [84].

In terms of modeling the moving objects of the game, some examples of methodologies are presented by several teams:

ISePorto states that they combine two methods: the Maximum Likelihood method which provides them with an initial estimate of the trajectory and the Extended Kalman filter which is able to recursively update the trajectory using new information [85]. Aiming to make use of the information that should be available and shared through the team robots, SocRob presented a method to perform cooperative tracking of the ball based on a particle filter [86].

Furthermore, the MRL team state in their 2013 TDP that they have implemented a Kalman filter followed by a linear regression to filter ball positions and estimate its velocity [87], based on our own work [88].

In the SPL, despite the significantly higher number of teams, we find a number of base methodologies falling in the class of particle based filters and Kalman type filters including its extended and unscented versions.

We can also verify that it is currently not possible to state which approach is definitely better, as it is common for teams to change from one kind of approach to another in a balanced relation.

Concerning the localization problem, Molen presents a summary of the used algorithms by each SPL team in the 2011 edition [89].

Through the analysis of the 2013 team description papers, there are several teams that report to be using particle filter approaches, most of them reporting also some kind of extra modules.

B-Human team uses a particle filter approach [90, 91], complemented with an unscented Kalman filter module [92]. The Cerberus team opted to extend the particle filter with a x-mcl approach [93]. Furthermore, the Dutch Team [94] applies the same method but keeps both short and long term averages to improve their results [95]. SPQR team tries to apply the method in a distributed way [96].

Both Uchile and NorthernBites teams report to have changed from an extended Kalman filter to a particle filter approach for localization [97, 98] and team Dainamite changed from a multi model Kalman filter to a particle filter approach [99].

Another set of teams report to solve the same localization problem by using Kalman filter approaches, in several of their most common versions.

AustinVilla was one of the teams reporting to change from the particle filter approach to an unscented Kalman filter [100, 101] and Edinferno and Kouretes also changed to a multiple

hypothesis extended Kalman filter [102, 103]. Bembelbots [104] report to also use a multiple hypothesis approach, but using a pure Kalman filter instead of its extended version [105].

The RoboEireann team [106] base their localization algorithm on a single model unscented Kalman filter, but changed it to include a covariance intersection (CI) version to remove the traditional independent noise restriction, in an effort to achieve better robustness to correlated measurement errors [107, 105].

The rUNSWift team [108] have been evolving their Kalman filter over the years, and currently report a multi model version of the filter, with distributed data fusion between the team robots. The latest innovation was to include a variation of the Iterative Closest Point (ICP) algorithm to the matching of not only field lines but also other visual features and objects [109].

The Humboldt team [110] tries to move away from classical Bayesian approaches and invest on constraint based techniques as alternative [111, 112]. They claim that these techniques are computationally cheap and, despite having to handle inconsistent data, they can be advantageous when ambiguous data is available (as is the case, due to the field symmetry).

The team NTURoboPAL [113], although not describing the core methodology underneath the approach, report to use a simultaneous localization and tracking (SLAT) algorithm [114] that integrates information from multiple team mate robots and provides the estimates of the team mate robot poses and the opponent robot positions simultaneously.

The probabilistic approaches are also commonly used for improving the world model estimations of the game objects, such as the ball or opponent robots.

In the SPL, teams Cerberus and Kouretes report to use Kalman filter and extended Kalman filter, respectively, to estimate the state/belief of the world model objects [115, 103]. Several teams report to use a multiple model Kalman filter to have a better estimation of the ball state, such as B-Human [116], NorthernBites [98] and Dainamite [101].

3.3.2 General overview of worldstate entities

Within a soccer scenario, some elements are part of the game and can be a part of or are necessary for building the world representation, either directly perceived or estimated by each robot:

- **Ball:** For a team of agents to be able to play soccer, the most important entity that they need to know about is probably the ball. It is the main object of the game, through which the main objective of scoring goals is accomplished.

- **Goals:** The two goals present on the field are also important entities to know about, since they are the structures where the robots need to "place" the ball to score and defend against opponent shots.

- **Players:** Since soccer is a team game, there are several robots involved and inside the field during a game. These players can be divided in two categories: team mates and opponents. Since the teams are usually allowed to use communication between the robots, much team mate information is commonly acquired through direct sharing of data.
- **Field:** The field of play, defined typically by white lines on a green surface, is an important aspect of the worldstate, as they define a known layout that the robots must respect and take into consideration while playing.
- **Internal state:** For a robot to be able to perform any task, it should be aware of its own internal state so that its decisions and outputs make sense according to its task. This state can include basic information such as battery levels or configuration values in cases where those are available (such as team definition in the robotic soccer scenario).

3.3.3 Our worldstate structure

In the robotic soccer teams used as test beds for the work developed during this thesis period, we try to have a base structure for representing the necessary information. Of course that in both MSL and SPL leagues, being so different from each other, there are some aspects that may be particular for one or the other. Most of these aspects will be referred to in the respective item:

- **Robot** is a data structure that includes all the necessary data for characterizing the robot in the game. It includes most of its perception and coordination data necessary for the correct working of the team. This includes:
  - **ID** is the number that identifies a given robot.
  - **Role** is the role that the robot chose to perform.
  - **Behavior** is the current action that the role decided to take.
  - **Colors**, both team and goal (since we use a color name to distinguish the field side).
  - **Pose** which is the 2D ground position of the robot represented on the field axis and the angle it has, measured from the opponent goal.
  - **Velocity** is a representation of the robot linear velocity on the ground plane and its angular velocity, measured around itself.
  - **Internal info** is a set of flags and values used for decision making and management of the robot internal state, such as flags that mark the robot as running, having a role remotely defined or being coached. As for values, usually some coordination values and points are present in the robot structure, to allow communication of decisions that need cooperation.

  - **Ball** is a data structure with some basic attributes for the ball, which include its position on the ground both relative to the robot and in the field absolute coordinates, its velocity and a boolean flag of visibility. In the MSL context, since there is the concept of ball engaged on the robot, there is also a flag to signal this.
  - **Obstacles** is a sublist of obstacles visually detected by the robot. Each obstacle is represented by its position on the ground and an ID, which is the team mate ID if it was identified as a team mate or another value outside the team IDs.

- **List of Robot** is a list with the Robot information of all the agents in the game, updated regularly via wifi.

- **Coach** is a structure with coaching information, which is defined by a separate coach agent, which is not a robot. The information it uses is the one gathered by the robots and its data is shared to all other agents.

- **Obstacle** is a list of all the obstacles that the robot is considering. This list may have obstacles not directly seen by the robot.

- **Field** is a class that keeps all the measures of the field of play, as well as methods to extract landmark information of the field, such as the metric position of specific lines or goals.

3.3.3.1 Integration processes and phases

On the software architecture of both leagues, the first step to take on each cycle is the integration of all the available information and manage it so that we can have a representation of the robot and surroundings as complete and precise as possible. The integration process executes several tasks with the information gathered from several sources. We will call the executors of the tasks as modules, whether it is an individual process or a block of the integrator. Figure 3.16 is a schematic of these modules, represented by rectangular boxes. The ellipsoid shapes represent the information that is provided or consumed by the modules. The modules presented in figure 3.16 are briefly explained.

- **Proprioception**

  - **Body Joints Sensors:** Specifically on the NAO robot, each of the joints has a position sensor. This module is the one responsible to read the sensor data of the robot body and provide normalized angles of the joints.
  - **Kinematic analysis:** Again on the NAO robot, a kinematic model is needed to take the angles of the joints and estimate the complete pose of the robot. This pose allows to know the position and orientation of any part of the body with

relation to another part, usually the head, which is the body part that contains the camera.

- **Camera Pose Analysis:** For the robot to be able to transform its perception from pixel coordinates to metric coordinates, it has to know what is the pose of the camera in relation to the ground plane, where all the objects of the game are standing on. In the CAMBADA robot case, since the robot body is not articulated, the camera pose relative to the vertical is assumed to be always the same. In the NAO robot, this module uses the body pose and torso angles estimated by the inertial unit and estimates the necessary transformations.
- **Inertial Unit:** The robots of both teams are equipped with an inertial unit composed of accelerometers and gyroscopes. The CAMBADA robot inertial unit has also a compass which can measure the orientation of the robot. The inertial measurements are used, in the NAO case, mainly to extract the angles of the robot torso relative to the vertical, which are necessary for estimating the camera pose relative to the vertical. In the CAMBADA robot, it is used mainly to measure displacements of the robot, based on the analysis of the accelerometers.
- **Odometry Estimator:** The odometry module is responsible for "measuring" how much the robot should have moved based on the movements performed by its actuators. Since this is an internal analysis, it is prone to errors on the long run. The external interactions with the environment, such as slips and hits, deviate the supposed trajectory estimated by the module.

- **Perception**
  - **Image Analysis:** The perception of the robots is achieved through the use of cameras which are managed by a dedicated vision process, responsible for performing the image analysis and produce information about the objects of interest on the image, represented in terms of pixel coordinates on the image.
  - **Robot-centered Coordinate Estimator:** After having the pixel coordinates, we need to estimate the correspondence of those coordinates in the real world, relative to the robot point of view. The estimation of the real world metric coordinates is done by trigonometric projection calculations which are based on the known pose of the camera relative to the vertical and the known parameters of the camera hardware. By the end of this task, all the objects of interest are represented in metric positions relative to the robot center.
  - **Robot Localization:** After having the metric information of points over the field white lines and the odometry estimation of the last movement, the first task that the robot needs to perform is the localization. This step is crucial, since most of

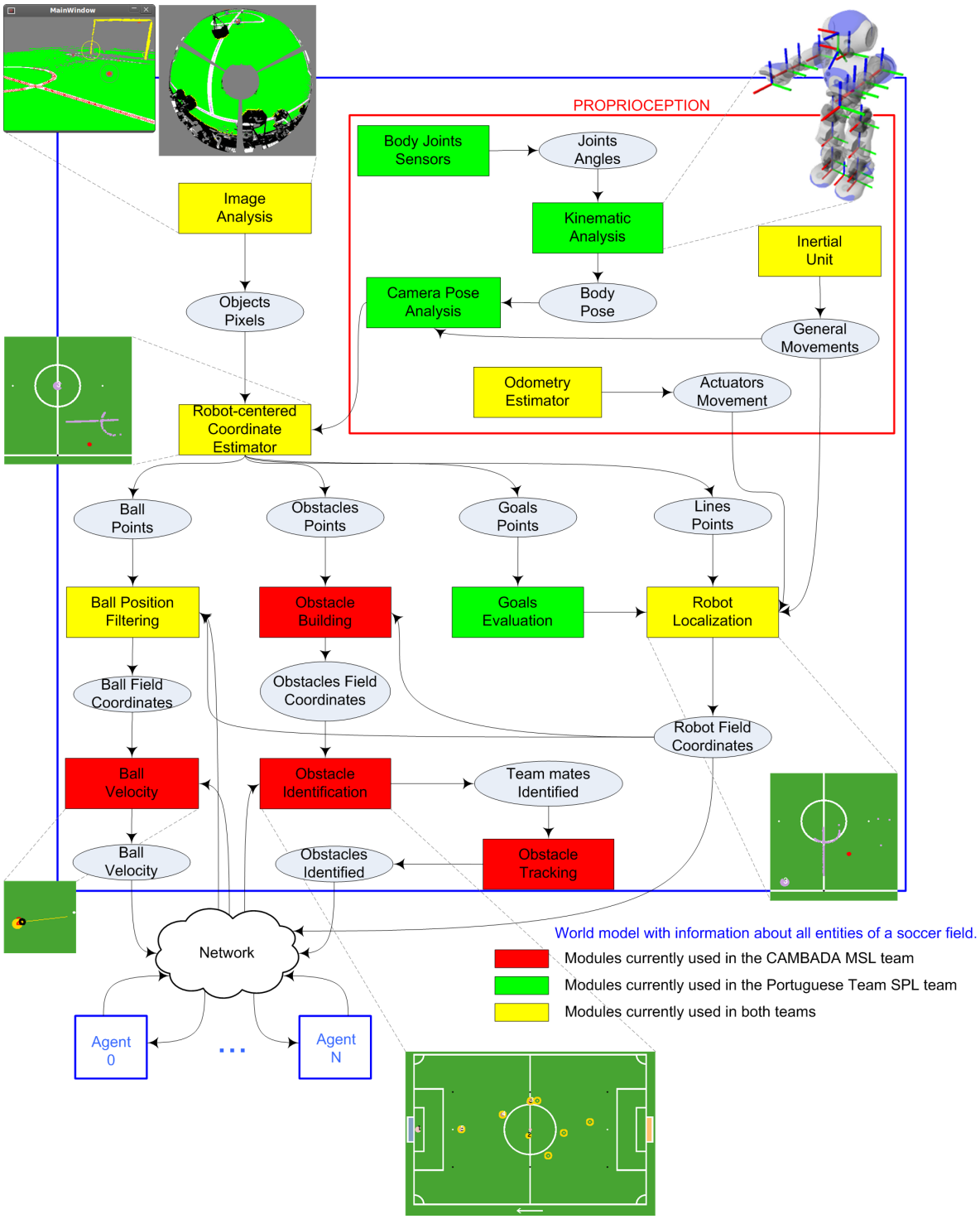


Figure 3.16: Diagram of information fusion tasks performed by the soccer agent on CAMBADA and NAO robots.

the representation of the world will need to be related to the field of play, rather than relative to the robot. The localization on both robots is similar, it uses mainly points over the white lines of the field seen by the robot to match with the known layout and thus estimate the robot pose on the field, by using adapted versions of the tribots algorithm [24].

- **Goals Evaluation:** In the case of the NAO robot, and since it is not equipped with any absolute sensor capable of estimating an orientation, such as the compass of the CAMBADA inertial unit, the goal evaluation module is used to visually search for the goals use also this information for localization purposes. This can be done due to strict rules of robot placement which enable estimation of the robot orientation given the relative distance of the goals.
- **Obstacle Building:** The visual information of the obstacles is a set of points on the periphery of the robots that are, on this phase, a set of metric coordinates. This module is responsible for clustering the points for each object, creating a list of obstacles represented by its limit points and center point. It is also responsible for dividing visually large objects, which occur frequently when several robots are clustering together, and create a separate obstacle for each of the robots present.
- **Obstacle Identification:** Of all the visually detected robots, some of them may be the other CAMBADA robots on the field. This module is responsible for crossing the information shared by the team mates with the obstacles just created and classify each of them as an opponent or a particular team mate. This information may be used by the decision layer to make different approaches depending on the obstacle being a team mate or an opponent.
- **Obstacle Tracking:** After ruling out which obstacles are team mate robots and which are not, the opponents that are visible may have just entered the robot field of view or they may already have been visible before. This module is responsible for keeping track of each opponent robot by keeping historical information about each of them. This module will keep an identifier of each obstacle as long as it is visible and within the robot field of view, thus allowing the decision layers to unequivocally identify each robot on the field, being it CAMBADA or opponent.
- **Ball Position Filtering:** Given the noisy measurements available, the ball is an object that deserves a specific and individual treatment, since it is the main object of the game. After having the estimation of its position on the current cycle, this integration module is responsible for applying a filter to the ball position, taking into account its history and the new measurement, as well as the uncertainties involved in both the measuring and the historical information.
- **Ball Velocity:** After having a more stable and reliable position for the ball, it is

necessary to estimate its velocity. This step takes into account the history of the ball positions over time to estimate its velocity.

3.4 Summary

This chapter addressed the practical scenarios that were used during the work of this PhD and presented the two leagues where the work over robotic soccer was directly applied, introducing the context that may be necessary to grasp some of the work descriptions.

The robots used in the two leagues are completely different and thus some details about each platform were presented. These concepts about the robots are important for the produced work, since the advantages and limitations that the robots present directly condition the methodologies that need to be implemented.

Finally, and since most work is focused on perception, an overview of the concept of world model or worldstate for robotic soccer is presented, with the description of entities that we consider important to exist and a general overview of the modules that are implemented on the robotic soccer teams covered by this work.

Chapter 4

World model for CAMBADA

During a game of robotic soccer in the MSL each robot, as previously described in section 3.2.1, is running a series of processes with different responsibilities. The main focus of this PhD work is related with the *agent* process and its relation to the *vision* process. The agent can be considered the main process in the sense that it is the one that gathers all the information available, builds the world model and then acts according to it.

The omni directional vision process, *vision*, is the main source of individual information available to the robot, since it is through the omni directional camera that the main elements of the field are "seen".

In this chapter we will present a general overview of the omni directional vision process as well as an overview of the CAMBADA team world model. The work developed during this PhD is presented in several sections referring to each element and approach in particular.

The ball is one of the elements addressed and, being it the best reference object in a soccer scenario due to the fact that there should be only one in the field, it is used as reference for analyzing and estimating the precision of the visual information. One of the addressed tasks related to the ball is the analysis of the uncertainty/noise involved in its detection by the omni directional vision and development of effective methodologies to provide a more stable and precise representation of both position and velocity. Another task addressed concerning ball is the development of new vision processes to provide reliable information of the ball position when it is in the air and thus unseen by the main omni directional vision.

The localization algorithm is explained and the improvements made for detecting lost situations are presented. The method for detecting a lost situation is independent of the used sensor and a comparison of the results obtained with two different sensors is presented, which represent a hardware change to the CAMBADA robots in order to achieve more reliable information.

The detection and identification of other robots on the field is another of the addressed issues. For this issue, the work developed includes the redefinition of the raw information provided by the vision, in order to obtain better results in the individualization of each

and every robot, as well as methodologies for unequivocally identifying each of those robots, specially concerning the differentiation between team mates and opponents.

The improvements on the several elements of the worldstate allowed to extend its functionalities and provide it with new and improved features and methods to provide information to the decision layer. One of the features improved during this PhD was a new approach to estimate a new direction to move the robot to reactively avoid collisions.

4.1 The omni directional camera

The main source of information for a CAMBADA robot is its omni directional camera. The *vision* process is responsible for processing the images from the camera and for detecting all the objects of interest which are currently the ball, the other robots and the lines.

All these objects start as blobs of pixels at some position on the 2D image and the center of each of those blobs represent its coordinates on the image, represented in pixels. The image coordinates of each element are converted to real world metric coordinates relative to the robot center through a mapping of each pixel to a given metric coordinate at ground level [117].

Several objects of interest on the field have specific colors: the green field, the white lines, the black robots and the previously defined colored ball. Besides the color differences, they are also within a known context at the ground level and the detection of each of them takes that context into account.

The *vision* process is thus divided in several stages:

- A color segmentation is performed, searching for all the calibrated colors and storing the information of each one as binary information using an 8 bit "image". Each color is associated to one bit and the bit is true or false either if the respective color is present on that pixel or not. This allows a pixel to be associated with several colors simultaneously.

- Points over the white lines are detected by searching for green→white→green transitions.

- Points over the robots are detected by searching for green→black transitions.

- Blobs of the ball color are constructed by clustering together pixels that are segmented as having the ball color.

- Each blob is analyzed to select which of them can actually be ball candidates. The analysis includes width and height proportions, solidity of the blob, general size of the blob according to its coordinates, among others.

None of the operations previously described are performed over all the image pixels. This is done both to improve the performance of the algorithms and to avoid processing pixels with no useful information.

The first method to select which pixels to ignore and which should be processed is an overlapping of a binary mask. This mask marks which ones should be processed and which should be ignored. There are two sets of pixels which obviously should be ignored since they do not possess any information about the robot surroundings:

- Pixels that are outside the mirror. These pixels are the excess of the square image that captures the reflection of the surroundings on the round mirror and represent the bottom part of the robot's top plate.

- Pixels on the mirror image that reflect the robot's own body. These pixels do not possess any information about the surroundings and are thus ignored. The reflected body parts are the camera and robot base itself and the support bars on the top of the robot, which connect this part of the mask to the outside part of the mask covering the non mirror zone.

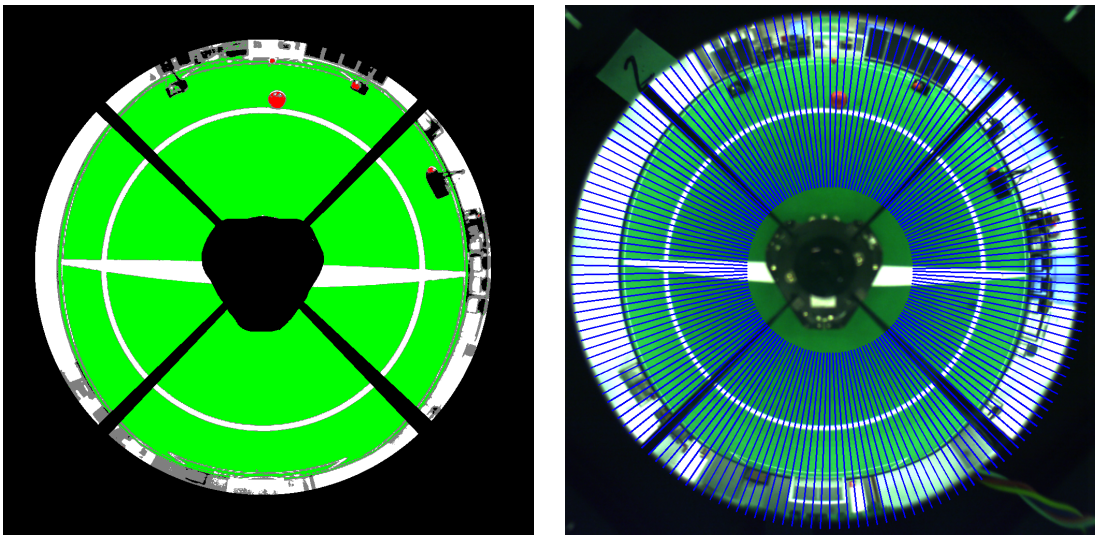
This method is designed to avoid processing pixels that possess no useful information, as is the case of the robot's own body. While this obviously reduces the processing weight by reducing the number of processed pixels, the number of remaining pixels is still high and can be further reduced without compromising the precision of the detections.

A second method for reducing the number of processed pixels is based on analyzing the pixels over a defined direction. From the center of the robot on the image, previously estimated by a configuration tool, radial "sensors" are created around the robot, each sensor represented by a line with a given angle. These lines are called *scanlines* [118] and the transitions detection and clustering operations previously described are executed only over these lines. Figure 4.1a exemplifies a small subset of the radial *scanlines* overlapped with the camera image. Currently some auxiliary circular *scanlines* are also used along with the main radial ones.

The transitions detection over a given *scanline* yields a point in pixel coordinates which will then be transformed into metric coordinates.

In the end, the blob detection algorithm also yields a point in metric coordinates. However, the point is the mass center of the given blob. To build a blob, the *scanlines* are sequentially analyzed and the colored pixels of a given *scanline* are clustered with the ones from neighboring *scanlines* according to a set of heuristics. These heuristics include analysis both in term of angular limits and separation (for blocks of pixels of neighboring *scanlines*) and limits and separation in terms of length from the scan center (for blocks of pixels in the same *scanline*). Details of this algorithm can be found in Alina *et al.* work [119].

Since the size of the objects on the image vary with the distance to the robot, it is advantageous to know the relation of that variation. The omni vision system is a non-SVP



a) **b)**

Figure 4.1: **a)** Picture representing the binary mask which excludes pixels with no useful information from processing. The colors on the remaining image are already segmented. **b)** Picture representing a small subset of *scanlines*, which work as a mask overlapped with the segmented camera image. Only the pixels on that mask will be analyzed.

hyperbolic catadioptric system. Through an inverse distance map calculation, by exploring a back-propagation ray-tracing approach and the physical properties of the mirror surface [117], the relation between the coordinates of each pixel and the coordinates of the ground points that each pixel represent in the real world, relative to the physical robot center, is known (figure 4.2).

Thus, the estimation of the objects positions on the field of play is obtained by retrieving the value from a Look Up Table (LUT), which is pre estimated and to which we call Distance Map. This is done because in the CAMBADA robots the camera is fixed on the robot body and always in the same position relative to the ground.

However, this Distance Map is created under specific conditions on the robot physical state. Since the structure is not completely rigid, small deviations on the camera and the mirror occur, some caused by collisions, others caused by trepidation as a consequence of having the robot running around the field. Due to this fact, one must keep in mind that the Distance Map is a source of uncertainty for the objects positions calculated by the vision. Just by looking at figure 4.2, it is obvious that farther distances are evaluated worse, since the distance represented by each pixel increases, non-linearly, as the pixel is farther from the center of the image.

An analysis of the uncertainty involved in the visual detection is presented in the ball analysis section.

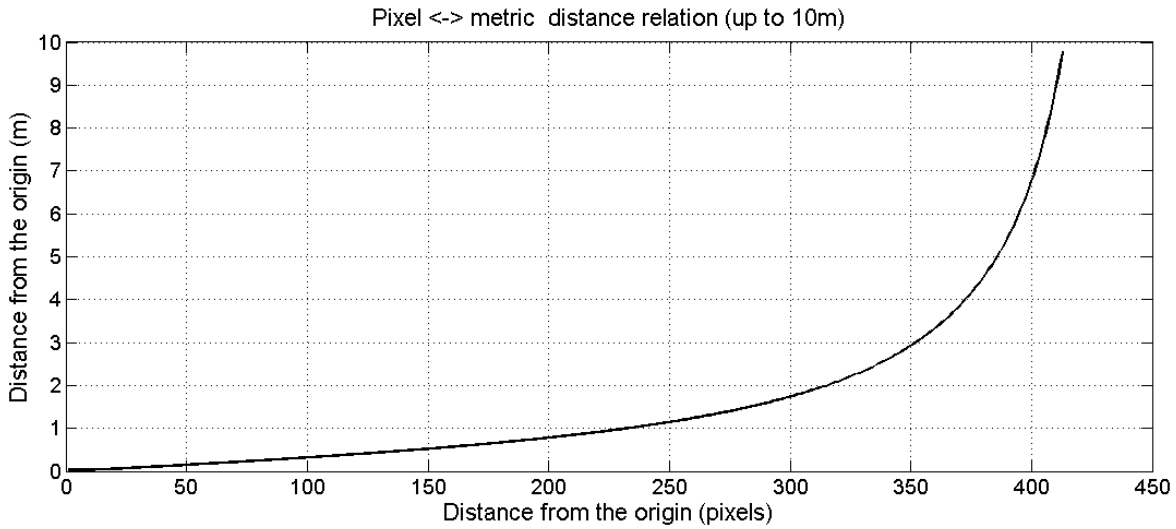


Figure 4.2: Relation between pixels and metric distances of the omni directional vision system. The center of the robot is considered the origin and the metric distances are considered on the ground plane.

4.2 CAMBADA worldstate

In the CAMBADA team, the worldstate structure is updated every cycle of the agent with any new information that becomes available. This structure keeps information about each element of the game with as much information as possible. Among the properties of each object, the most important one is probably the position.

Each point in the field of play is thus well defined and represented by a Cartesian coordinate system with origin on the field center and with the YY axis positive in the direction of the opponent goal (figure 4.3). This coordinate system is referred as *absolute*, since it is a static referential common to all team robots.

On the other hand, the visual perception of a robot is based on information gathered by the omni directional camera on the robot body. This implies that the information can only be represented from the robot's own point of view. The metric coordinates at ground plane estimated by the vision process, as referred in section 4.1 are thus represented in what we call the *relative* coordinate system. This is also a Cartesian coordinate system which is centered on the robot center and with the YY axis positive in the direction of the robot front (figure 4.4).

The *Integrator* module is responsible for updating the worldstate information, by performing a series of operations with all the available data. The integration itself is divided into several tasks executed in a specific order. For most of the tasks, the execution order is important, as later tasks depend on data produced by earlier tasks. A brief description of these tasks is presented next and represented in figure 4.5.

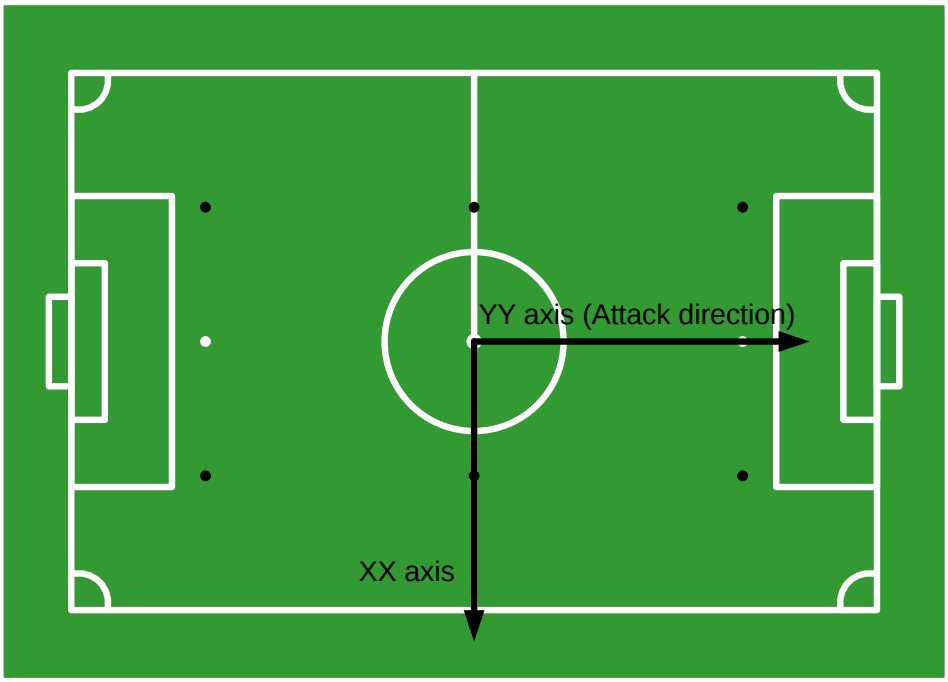


Figure 4.3: CAMBADA field absolute coordinate system.

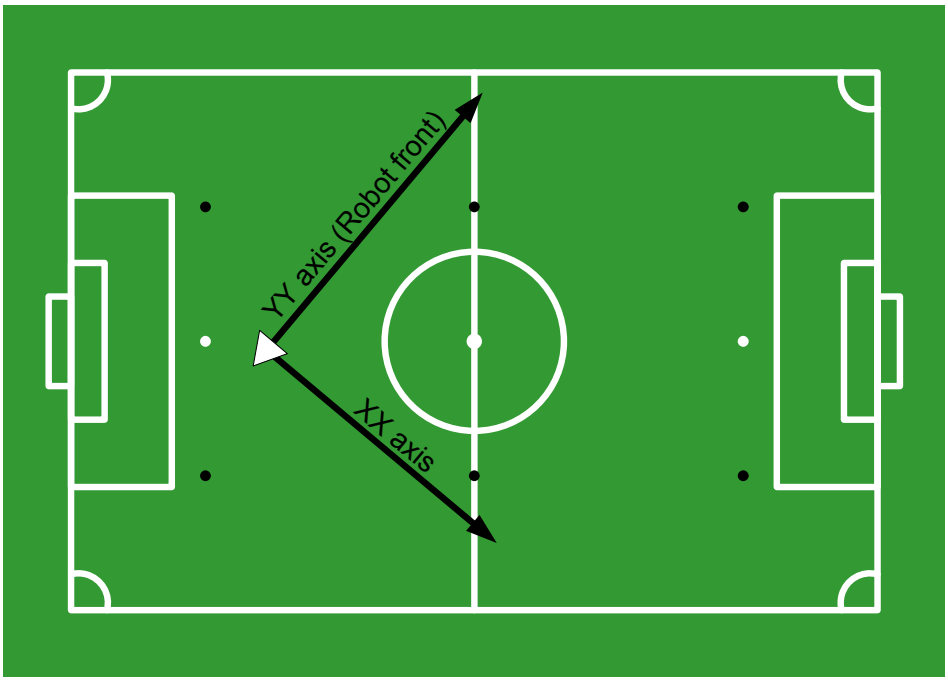


Figure 4.4: CAMBADA robot centered relative coordinate system.

- Read the information from the several available data fields of the RTDB, both the local data from the sensors (vision information and robot platform information) and shared

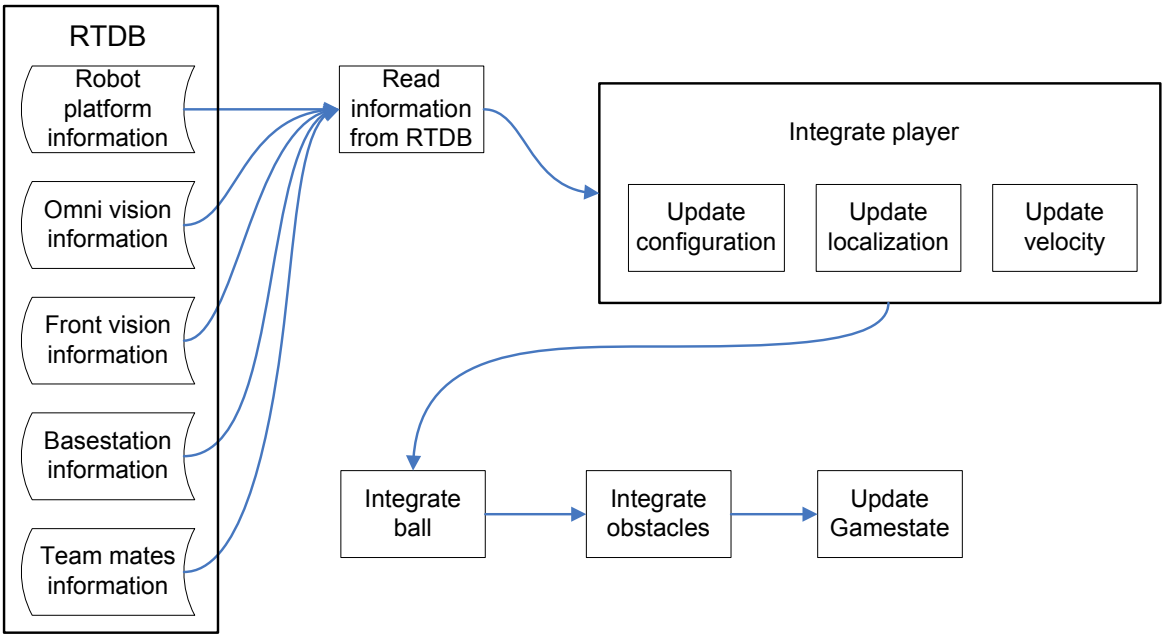


Figure 4.5: Integration tasks sequence diagram.

data from the coach and other team mate robots.

- Update player configuration values defied by the basestation. These configuration values include the current team and goal, the role that the robot should assume (if we want to force it for testing purposes) and an activation flag that defines the agent as running or not.
- The first task is to use the white points over the field lines that were detected by the vision process and the heading information provided by the robot platform. Both data are used by the localization algorithm to estimate the pose of the robot on the field. This pose is composed by a 2D Cartesian coordinate and an angle, both relative to the field coordinate system, thus representing the absolute position of the robot.
- Now that the current position in the world is estimated, the robot uses the historical information of its positions to estimate its velocity. The process used for this estimation is an egomotion algorithm, presented by Lauer [120].
- After knowing its own position in the world and the correspondent velocity, it is now time to estimate the positions of the field objects. These objects are the ball and the obstacles. The update methods for these objects are subject of further analysis in the next subsections.
- The game state is another important update to execute, which is basically an analysis of the history of the referee signals and detection of changes to the state of the world

according to the rules. As an example, the update game state code is responsible for detecting that the ball has moved in an opponent set piece and update the state to free play so that the robot can act accordingly.

4.3 Ball perception

The ball is one of the most important objects on a soccer game. In this scenario, we aim to have a precise estimation of the ball position and velocity inside the field, especially at ground level. To be able to achieve that, we need to filter the ball position perceived by the vision process, considering the inherent noise, and contextualize the data according to the newly available information.

Each agent cycle, following one vision frame, we have a list of possible candidates for the ball. The reason for having several candidates is that, in a scenario such as the MSL, we must assume that false positives can exist, since the environment around the field is completely uncontrolled and several balls, or objects that can be perceived as ball, can be around. The *vision* process itself should not discard an object that is successfully evaluated as a ball considering its heuristics briefly presented in section 4.1, since there is no context information on that level. As described in the previous section, when the ball integration is performed, the robot/player information has already been updated, which means that the current pose and velocity of the robot is already estimated, being these the most important values for subsequent analysis.

The information provided by the *vision* about the ball comes in the form of an array of 2D positions, representing the ground coordinates of each ball relative to the robot center.

Thus, when updating the worldstate ball information, the first step is to verify which of the ball candidates must be used. The main concern is to discard any ball that is outside of the field, since this is a recurrent situation during competitions.

The measurements of the ball position provided by the robot camera are noisy, due to the nature of the sensors and dynamics of the environment. In order to achieve better performance on the posterior decision of the robot movement, we need to apply a filter to the ball position to achieve a more precise position estimation and thus a more steady trajectory when it is moving, but also allow reactivity to abrupt direction changes.

4.3.1 Ball filtering

The information of the ball state (position and velocity) is, perhaps, the most important, as it is the main object of the game and it is the base over which most decisions are taken. Thus, its integration has to be as reliable as possible. To accomplish this, a Kalman filter implementation was created to filter the estimated ball position given by the visual information, and a linear regression was applied over filtered positions to estimate its velocity [121, 122, 88].

4.3.1.1 Ball position

It is assumed that the ball velocity is constant between cycles. Although that is not true, due to the short time variations between cycles, around 40 milliseconds, and given the noisy environment and measurement errors, it is a quite acceptable model for the ball movement. Thus, no friction is considered to affect the ball, and the model doesn't include any kind of control over the ball. Therefore, given the Kalman filter formulation (described by Welch *et al.* [8]), the assumed state transition model is given by

$$X_k = \begin{bmatrix} 1 & \Delta T \\ 0 & 1 \end{bmatrix} X_{k-1}$$

where $X_k = \begin{bmatrix} Pos \\ Vel \end{bmatrix}$ is the state vector containing the position and velocity of the ball in a given direction component. The state is kept for both direction components, (x,y) coordinates. This velocity is only internally estimated by the filter, as the robot sensors can only take measurements on the ball position. After defining the state transition model based on the ball movement assumptions described above and the observation model, the description of the measurements and process noises are important issues to attend. The measurements noise can be statistically estimated by taking measurements of a static ball position at known distances.

In practice, measurements of the static ball were taken while the robot was rotating around its vertical axis and this was done with the ball placed at several distances, measured with metric tape. Although real game conditions are probably more adverse, we lack the means to externally know the position of the elements on the field. For that reason, to know the real distance between the robot and the ball, we opted to use the described setup. Captures were made with the ball at 1, 2 and 3 meters, measured by tape, with the robots rotating at 90º/s and at 4 meters rotating at 45º/s since the robot could not detect the ball at 4 meters when rotating faster than 45º/s. At 5 meters the robots could not detect the ball under any conditions. Some of the results are illustrated in figure 4.6.

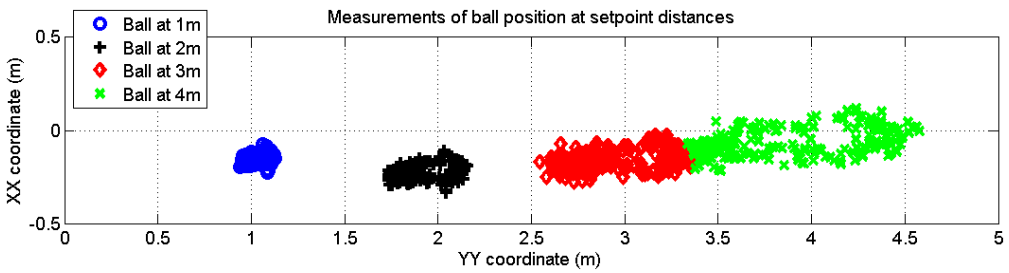


Figure 4.6: Noisy position of a static ball taken from a rotating robot.

The standard deviation of those measurements can be used to calculate the variance and thus define the measurements noise parameter.

A relation between the distance of the ball to the robot and the measurements standard deviation can be modeled by a 2nd degree polynomial best fitting the data set in a least-squares sense. Depending on the available data, a polynomial of another degree could be used, but we should always keep in mind the computational weight of increasing complexity.

As for the process noise, this is not trivial to estimate, since there is no way to take independent measurements of the process to estimate its standard deviation. The process noise is represented by a matrix containing the covariances correspondent to the state variable vector.

Based on the Kalman filter functioning, one can verify that forcing a near null process noise causes the filter to practically ignore the read measures, leading the filter to emphasize the model prediction. This makes it too smooth and therefore inappropriate. On the other hand, if it is too high, the read measures are taken too much into account and the filter returns the measures themselves.

To face this situation, one has to find a compromise between stability and reaction. Since we assume an uniform movement for the ball, there are no frictions or other external forces considered. This means that accelerations are not considered in our model and thus, the position and velocity components are quite independent of each other. Since acceleration is the main element of relation between position and velocity, we considered that the errors associated to the process position and velocity estimations do not correlate.

Because we assume an uniform movement model that we know is not the true nature of the system, we know that the speed calculation of the model is not very accurate. A process noise covariance matrix was empirically estimated, based on several tests, so that a good smoothness/reactivity relationship was kept. These empirically estimated values were made dependent on the measurement noise so that the Kalman filter predictions are also less accurate when the distance to the ball is too large. This was done so that the filter does not smooth the positions too much.

In practice, the developers team observed that using this approach improved the estimation of the ball position. Since we do not possess the means to externally know the positions of the elements on the field, a capture was made with the ball fixed at a known position on the field $(0.0, 2.0)$ (measured with metric tape). The robot was moving around the ball with a speed of 1.3 ± 0.5 m/s and the ball position measured at each moment was recorded. The ball position measured by the robot was $(-0.01, 2.03) \pm (0.05, 0.06)$ meters. Figure 4.7 illustrates the capture results.

This experiment gives an idea of the noise associated with the ball position detection. Note that during the experiment the distance between the robot and the ball is around 2 meters. Comparing the ball position cloud with the one obtained at 2 m in figure 4.6 one can verify

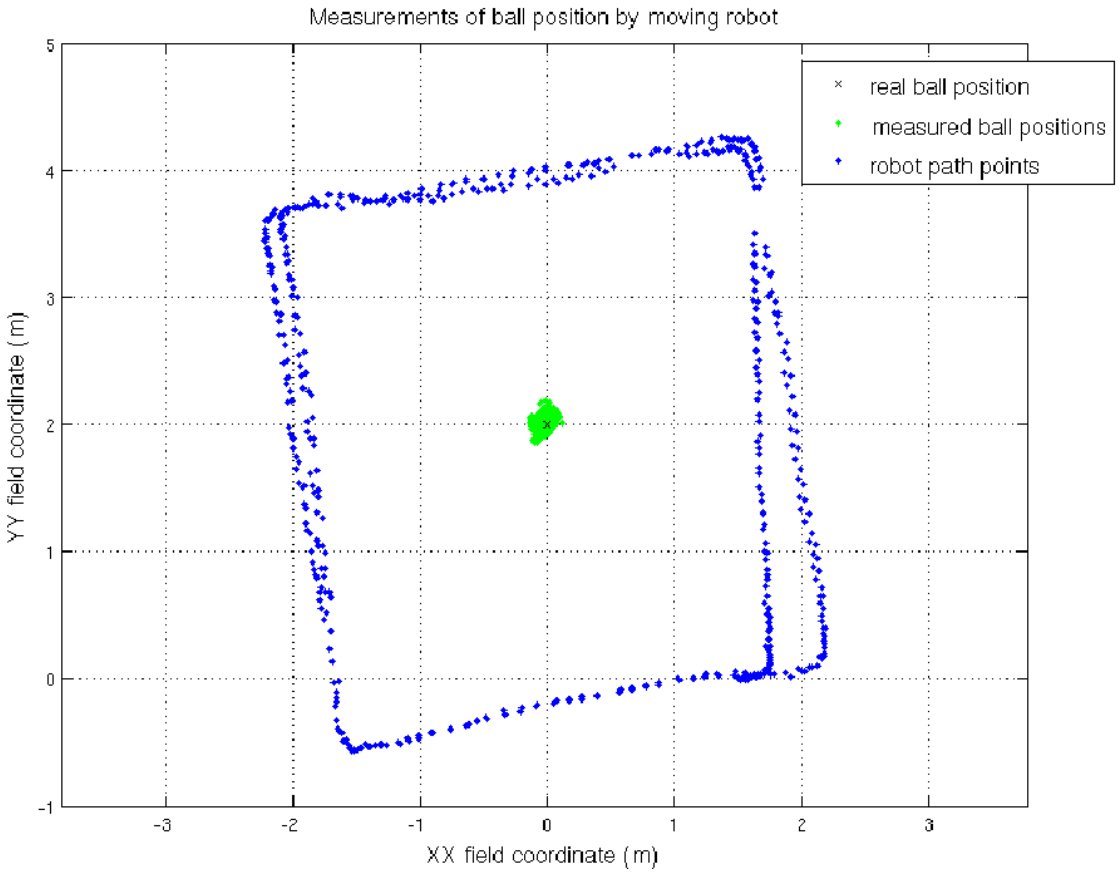


Figure 4.7: Plot of a robot movement around a fixed ball position. The ball positions measured by the moving robot form a cloud of points (green) in the area of the real ball position (black X) near coordinates (0.0, 2.0).

that they are similar, which is consistent with the previous experiment setup to simulate robot movement by rotation on the spot and thus, this static ball with rotating robot setup became the standard setup for estimating the visual noise of the CAMBADA robots.

With the presented setup experiments, the existence of noise in ball measurements became clear. With that existent noise in mind, several tests were made to validate the use of the Kalman filter to reduce it. Figure 4.8 represents a capture of one of those tests, a ball movement, where the black dots are the ball positions measured by the robot visual sensors and thus are unfiltered. Red stars represent the position estimations after applying the Kalman filter. The robot position is represented by the black star in its center and its respective radius. The ball was thrown against the robot and deviated accordingly. It is easily perceptible that the unfiltered positions are affected by much noise and the path of the ball after the collision is composed of positions that do not make much physical sense. Although we lack the means to externally provide a ground truth for the ball position during its movements, the filtered positions seem to give a much better approximation to the real

path taken by the ball, as they provide a path that physically makes more sense.

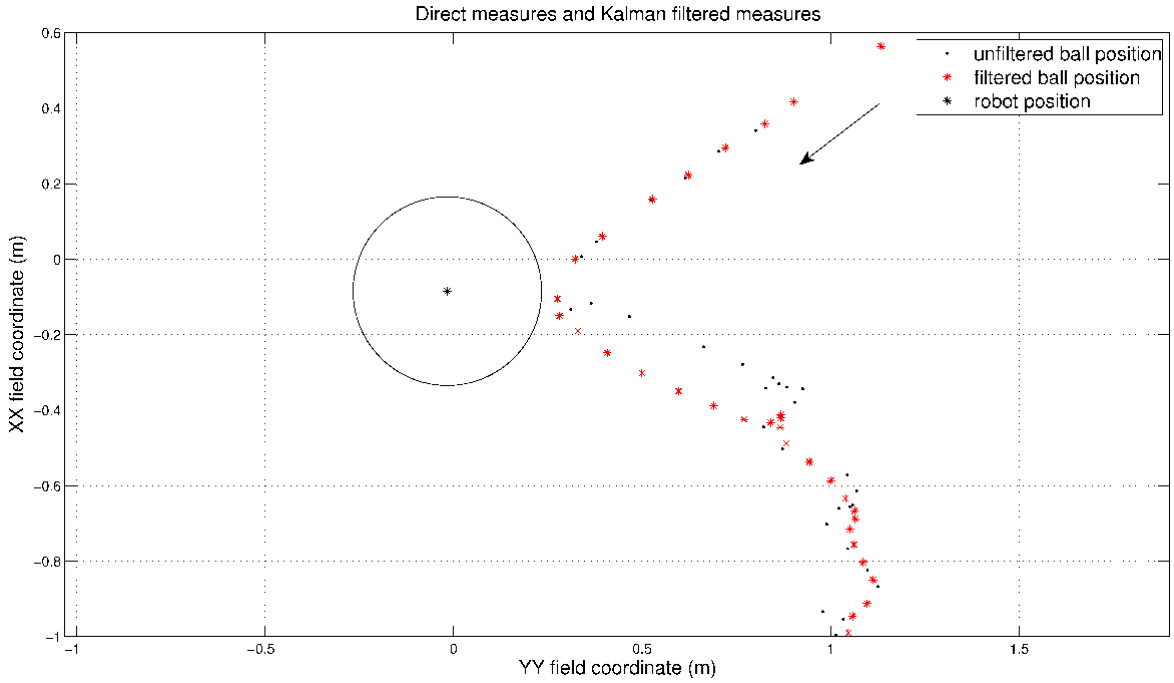


Figure 4.8: Plot of a ball movement situation.

After producing the *a-priori* estimation of the ball position, this estimation is compared with the read measure to detect if the variation between them is too great. If the difference between them is greater than a given threshold (estimated empirically) for more than a few cycles (currently 3 cycles), the filter can indicate that the ball suffered a hard deviation (figure 4.9 illustrates this concept).

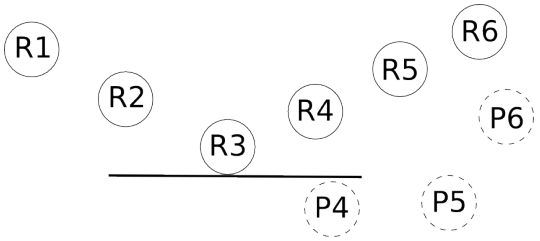


Figure 4.9: Situation where a hard deviation would be detected by the filter. Positions R4,5,6, are the measured positions after the ball hits an obstacle, P4,5,6 are the predicted filtered estimations, which did not consider that something might alter the ball path.

Although hard deviations are not a serious problem for the filter (as it quickly converges to the new positions), they are used for velocity convergence (as described in the next subsection).

4.3.1.2 Ball velocity

The calculation of the ball velocity is a feature becoming more and more important over time. It allows that better decisions can be implemented based on the ball speed value and direction. Assuming the same ball movement model described before, constant ball velocity between cycles and no friction considered, one could theoretically calculate the ball velocity by simple instantaneous velocity of the ball with the first order derivative of each component $\frac{\Delta D}{\Delta T}$, being ΔD the displacement on consecutive measures and ΔT the time interval between consecutive measures. However, given the noisy environment, it is also predictable that this approach would be greatly affected by that noise and thus its results would not be satisfactory.

Figure 4.10 shows a ball movement capture where the ball was moving from left to right, as indicated by the arrow in the top of the figure, and was then deviated into a downward movement near the "1st deviation" tag. While moving downward, the ball was deviated again near the "2nd deviation" tag and started to move from right to left. Finally, in the end of the capture, a new deviation occurred near tag "3rd deviation" where the ball started to move upward. The estimated ball positions are represented by the blue dots. Red lines represent the velocity vectors estimated based on consecutive positions displacement. It is clear that the velocity estimates hardly give an acceptable insight of the ball movement.

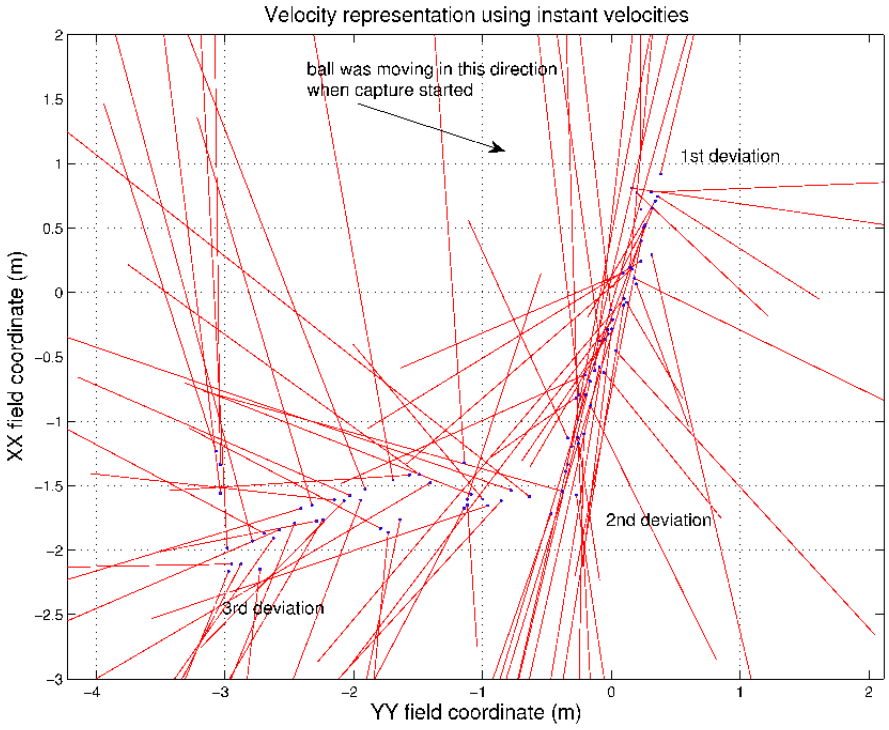


Figure 4.10: Velocity representation using consecutive measurements displacement.

To keep a calculation of the object velocity consistent with its displacement, an imple-

mentation of a linear regression algorithm was chosen. This approach based on linear regression [123] is similar to the velocity estimation described by Lauer *et al.* [124]. By keeping a buffer of the last m measures of the object position and sampling instant (in this case buffers of 9 samples were used), one can calculate a regression line to fit the positions of the object. Since the object position is composed by two coordinates *(x,y)*, we actually have two linear regression calculations, one for each dimension. This is made in a transparent way, so the description is presented generally, as if only one dimension was considered.

When applied over the positions estimation, the linear regression velocity estimations are much more accurate than the instant velocities calculated by $\frac{\Delta D}{\Delta T}$, and allow a better insight of the ball movement. The same ball movement capture described earlier is represented in figure 4.11, this time with the velocity vectors estimated by the linear regression applied over the position estimations provided by the Kalman filter.

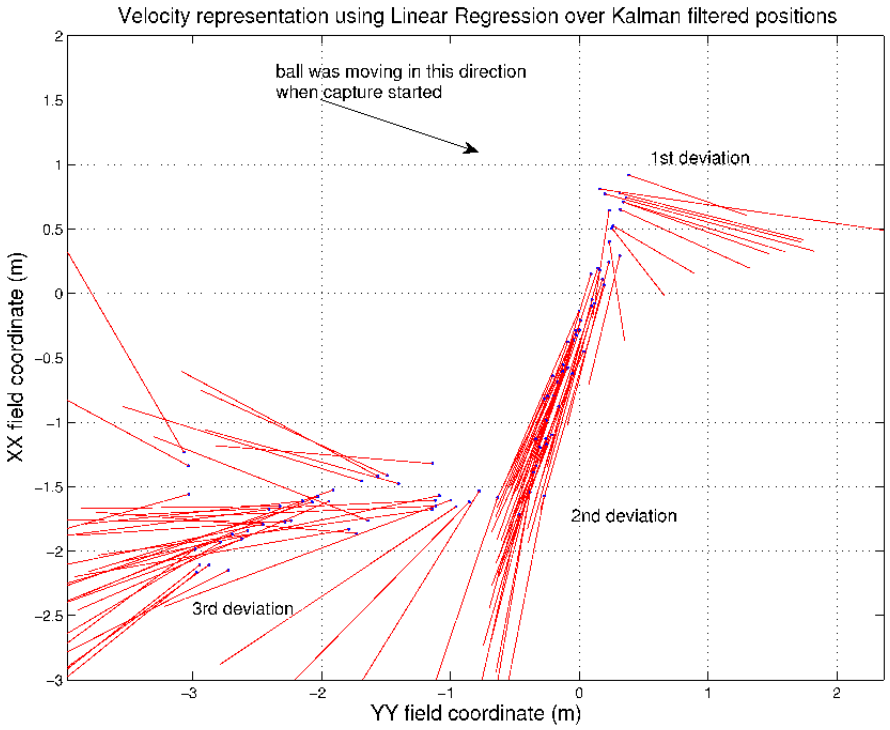


Figure 4.11: Velocity representation using linear regression over Kalman filtered positions.

In order to try to make the regression converge more quickly on deviations of the ball path, a reset feature was implemented, an idea similar to the one presented by Veloso *et. al.* [125]. This allows deletion of the older values, keeping only the n most recent ones, and provides control of the buffer size. By keeping the most recent values after a hard deviation, we reduce outliers of the previous path, thus promoting faster convergence. This reset results from the interaction with the Kalman filter described earlier by querying it for the existence of a hard deviation on the ball path.

The obtained values were tested to confirm if the linear regression of the ball positions was more precise and would converge faster than the internal velocity estimated by the Kalman filter. Tests showed that the velocity estimated by the Kalman filter has a slower response than the linear regression estimation when deviations occur. Given this, the linear regression was used to estimate the velocity because quickness of convergence was preferred over the slightly smoother approximation of the Kalman filter in the steady state. That is because in the game environment the ball is very dynamic, it constantly changes its direction and thus a convergence in less than half the cycles is much preferred. Figure 4.12 shows the results for a theoretical velocity scenario where the ball was moving at a constant speed of 2m/s and suddenly dropped to a constant 1m/s speed. The speed estimated by the Kalman filter and the one estimated by the linear regression are presented.

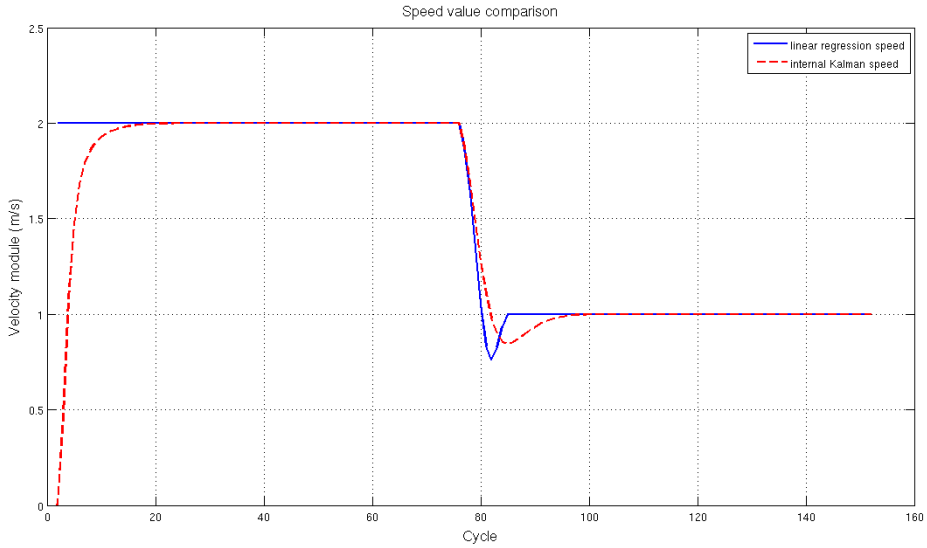


Figure 4.12: Comparison between the velocity estimated by the linear regression (blue solid line, faster convergence) and internally by the Kalman filter (red dashed line, smoother, but of slow convergence).

4.3.2 Noise analysis

Over time, as the algorithms of the team change and evolve, some aspects of the models also need to be updated. The noise associated with the positions measured by the omni directional camera is one of those aspects.

The noise level of the ball position, at the time of the writing of this document, is measured through a series of experiments where the ball is standing still and the robot is moving. These still ball scenarios are used due to our impossibility to measure the position of a moving ball without any noise.

In order to reflect the improvements that the vision hardware and algorithms have been suffering over the years on the robot's perception, the capture scenario presented before in section 4.3.1.1 was repeated using the new robots and the new algorithms. The same captures were made with the ball at 1, 2, 3 and 4 meters, measured by tape, with the robots rotating at 90°/s. While currently the robots can detect the ball farther than before, the capture scenario was kept almost under the same conditions, so that the results comparison is as direct as possible. The only difference is that, for the new robots, the captures were made all with a rotation speed of 90°/s, even at 4 meters, for simplification of the capture process and because the ball can now be detected at that distance and angular velocity.

The measurements of the relative ball position for each of the robots at each of the distances, while rotating at 90°/s is presented in figure 4.13.

To have an idea of the noise associated with a ball position measured by the vision, which is represented in relative coordinates, we use the length of the vector between the robot (origin of the axis) and the ball position. By estimating the standard deviation of the measured distances, we get an approximation of the uncertainty of those measures and consider this uncertainty value as the measurement noise associated with the given distance. Table 4.1 presents the results for the *distance* → *standard deviation* relation measured for the new robots.

Distance (m)	Robot 2 StdDev (m)	Robot 3 StdDev (m)	Robot 4 StdDev (m)	Robot 5 StdDev (m)	Mean StdDev (m)
1.0	0.0179	0.0172	0.0154	0.0180	0.0171
2.0	0.0418	0.0277	0.0265	0.0299	0.0315
3.0	0.0600	0.0528	0.0428	0.0539	0.0524
4.0	0.0881	0.0814	0.0803	0.0781	0.0820

Table 4.1: Standard deviation results for each robot.

In the first experiments with the old robots, the option taken at the time was to estimate a mean of standard deviation of all the robots and estimate a single fitting function for the measured set points. The obtained function was estimated using the Matlab *polyfit* function, which estimates the best fit over a least-squared distance method. The estimated relation was given by the mean of all the standard deviations for the distance d between the robot and the ball, $\hat{\sigma_1}$.

$$\hat{\sigma_1} = 0.04d^2 - 0.02d + 0.016 \tag{4.1}$$

Prior to RoboCup2012, still with the old robots but with changes to the algorithms of the vision process, a new analysis was made and the noise function was updated to $\hat{\sigma_2}$.

$$\hat{\sigma_2} = 0.01d^2 - 0.01d + 0.039 \tag{4.2}$$

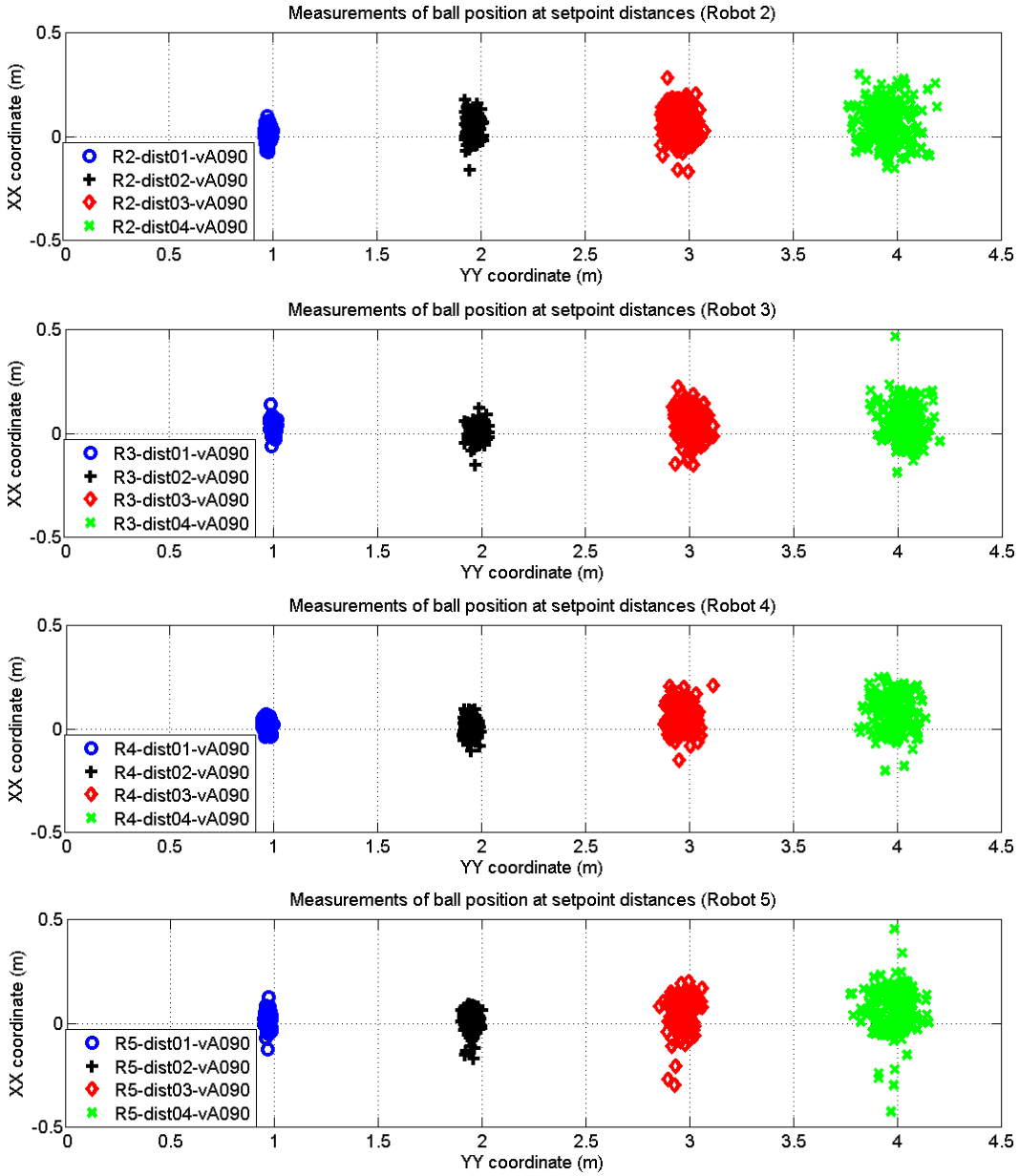


Figure 4.13: Noisy position of a static ball taken from rotating robots.

With the more recent data of table 4.1 from the capture scenario described before, a new noise function was estimated, $\hat{\sigma_3}$.

$$\hat{\sigma_3} = 0.0038d^2 - 0.0026d + 0.0109 \tag{4.3}$$

The three noise functions are represented in figure 4.14 and from the visualization of the curves, the effectiveness of the improvements becomes clear.

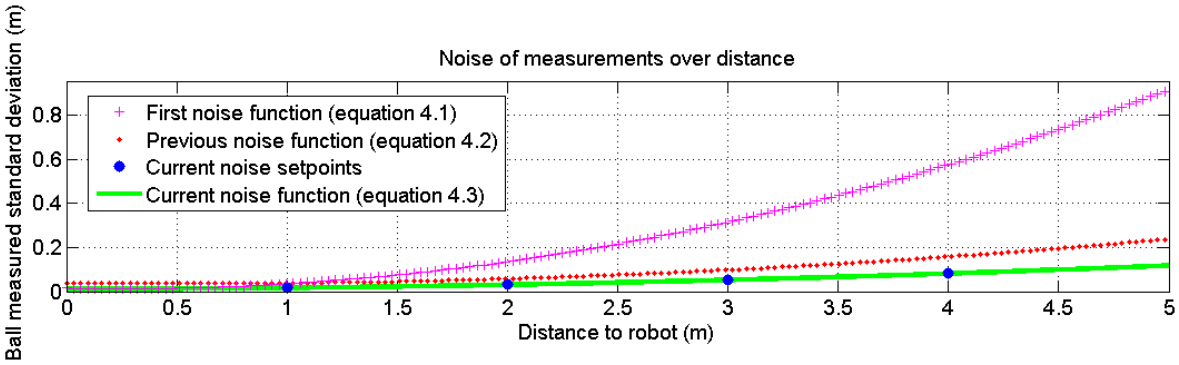


Figure 4.14: Evolution of the polynomial functions used for estimation of the sensor noise present on the visual measurements.

From the analysis of figure 4.14, one can easily verify that, although the evolution of the vision software greatly improved the precision of the detection even with the old platform, the current vision software on the new platform further improved the precision.

4.3.3 Detection of airborne balls

An overview of how the ball is perceived by the omni directional vision when it is on the ground has been presented and we have methods to estimate both its position and velocity with satisfactory reliability. However, another problem arises when the ball is kicked off the ground. In this situation, the omni directional vision is unable to detect it, since the ball on the image is projected to positions that are not in a useful part of the image, such as being above the horizon, and eventually disappears completely, when it becomes higher than the robot height and thus outside of the area perceived by the convex mirror of the omni directional vision. This problem is of significant interest in the MSL because most teams kick the ball through the air, since aerial balls are generally harder for goal keepers to defend. The detection of airborne balls is thus particularly important for goal keepers. To tackle this problem on the CAMBADA team, we installed a frontal camera on the goal keeper robot.

Since the ball in the MSL has a known predominant color for each tournament, most teams take advantage of this restriction while the ball is on the ground, since in that case, the environment is relatively controlled (green floor with some white lines and black robots) and the ball candidates can be expected to be surrounded by this reduced set of colors. In the air, these restrictions are completely lost and thus the approach should be more shape based. The existing shape based approaches on our team are mainly aiming at detecting a ball through shape on the omni directional camera [126], specially because they are dependent on the knowing the size that the ball should have when its center is in any given pixel of the image, which is a relation known only at the ground plane through the already referred distance map (section 4.1).

Several teams already presented preliminary work on 3D ball detection using information from several sources, either two cameras or other robots information [127, 128]. However, these approaches rely on the ball being visible by more than one source at the same time, either two cameras with overlapping visible areas or two robots, and then triangulate it. This is not possible if the ball is above the robot omni directional camera.

The detection of balls in the air is usually very aligned with the problem of detecting unknown color balls, or balls with an arbitrary color, which implies that they are detected mostly through shape analysis. Regarding the detection of arbitrary balls, the MSL league is the most advanced one. Many of the algorithms proposed during previous research work showed promising results but, unfortunately, in some of them, the processing time does not allow its use during a game, being in some cases over one second per video frame [129].

Hanek *et al.* [130] proposed a Contracting Curve Density algorithm to recognize the ball without color labeling. This algorithm fits parametric curve models to the image data by using local criteria based on local image statistics to separate adjacent regions. The author claims that this method can extract the contour of the ball even in cluttered environments under different illumination, but the vague position of the ball should be known in advance. The global detection cannot be realized by this method.

Treptow *et al.* [131] proposed a method for detecting and tracking the ball in a RoboCup scenario without the need for color information. It uses Haar-like features trained by an Adaboost algorithm to get a colorless representation of the ball. Tracking is performed by a particle filter. The author claims that the algorithm is able to track the ball with 25 fps using images with a resolution of 320×240 pixels. However, the training process is too complex and the algorithm cannot perform in real time for images with higher resolutions. Moreover, the results still show the detection of some false positives.

Mitri *et al.* [132] presented a scheme for color invariant ball detection, in which the edged filtered images serve as the input of an Adaboost learning procedure that constructs a cascade of classification and regression trees. This method can detect different soccer balls in different environments, but the false positive rate is considered too high by the authors for the problem at hand, and although it is supposedly real-time, execution times are never presented or analyzed.

Lu *et al.* [133] considered that the ball on the field can be approximated by an ellipse. They scan the color variation to search for the possible major and minor axes of the ellipse, using radial and rotary scanning, respectively. A ball is considered if the middle points of a possible major axis and a possible minor axis are very close to each other in the image. However, this method has a processing time that can achieve 150 ms if the tracking algorithm fails.

More recently, Neves *et al.* [60] proposed an algorithm based on the use of an edge detector, followed by the circular Hough transform and a validation algorithm. The average processing

time of this approach was approximately 15 ms. However, to use this approach in real time it is necessary to know the size of the ball along the image, which is simple when considering the ground plane in a omni directional vision system. This is not the case when the ball is in the air, in a completely unknown environment without a reference over which the estimation of the ball size on the image can be calculated.

4.3.3.1 Frontal RGB camera

The used camera contains a CCD of 6.26×5.01 mm and pixel size 3.75 μm. The maximum resolution is 1296×964 and the used lens has a 4 mm focal length. The camera is fixed to the robot in such a way that the axis normal to the CCD plane is parallel to the ground (figure 4.15).

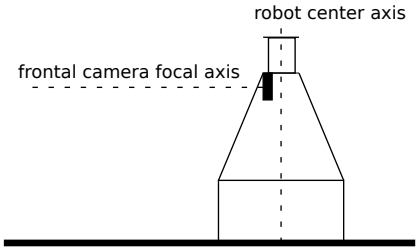


Figure 4.15: Illustration of the frontal camera positioning on the robot. It is placed in such a way that the camera focal axis is parallel to the ground and the focal point is slightly displaced from the robot center axis.

Based on the CCD size and the focal length, we can derive the opening angle, both along the horizontal and vertical axis. We will use α to identify the horizontal opening angle and β to identify the vertical opening angle.

Given α and β, we can also estimate the theoretical relation of the field of view (FOV) of the camera at different distances (these calculations do not take into account any distortion that may be caused by the lens). For a given distance, we can then estimate the width of the plane parallel to the camera CCD (as illustrated in figure 4.16) by:

$$tan(\alpha) = \frac{1}{2}\frac{hFOV}{Y} \quad \Rightarrow \quad hFOV = 2 \times Y \times tan(\alpha) \tag{4.4}$$

where:

- α is the horizontal opening angle of the vision system;
- hFOV is the width of the FOV plane at a given distance;
- Y is the distance of the considered FOV plane to the camera focal point.

For the height of the same plane, the analysis is similar in every way, now considering the CCD height and a β angle.

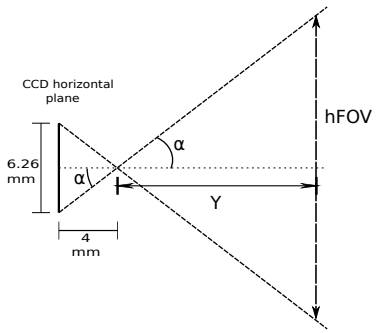


Figure 4.16: Scheme of the relation between the CCD width and focal length with the opening angle. The geometric relation is used for estimating the FOV at a given distance.

The positioning of the camera relative to the robot allows a direct application of this analysis of the FOV and the associated geometric characteristics. This happens because there is no angle distorting the FOV relative to the front of the robot, which means that the planes of detection remain parallel to the camera CCD. The idea of this application is to provide an approximation of the ball position mainly when it is in the air. The camera captures the images using format7, which allows the camera to capture the images using a specific ROI within full resolution of the camera. For the objectives of the described work, we opted to use the full horizontal size of the image, while the vertical size was cropped to the first 500 lines. This value was defined to cope with the choice that the camera is used to detect aerial balls and thus it only needs the image above the horizon line. Since the maximum vertical resolution is 964, we use the top of the image, with a small margin.

4.3.3.2 Ball visual detection and validation using a RGB camera

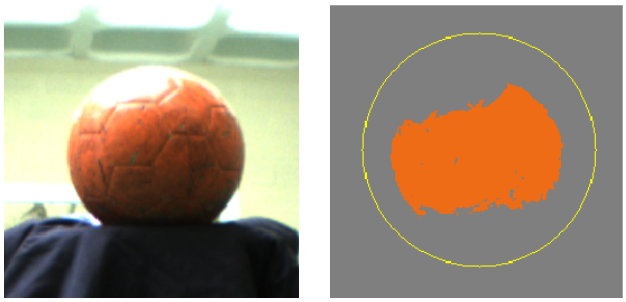
The first step that must be performed is the detection of the ball in the camera images. This task was tackled in two phases. On a first phase, the used detection approach was purely color segmentation based [134]. After preliminary results and analysis of existing scenarios, a second approach was developed and tested [135, 136].

Initial approach

It is visible and easily understandable that the ball on camera images does not appear as a homogeneous blob of the given color, but has three distinct regions. The central region of the ball is the one closer to the expected aspect. The bottom part of the ball is usually darker than the expected color and the top part is brighter (figure 4.17). This is due to the fact

that light comes from above, thus brightening the top of the ball while keeping its bottom in shadow.

The method to calibrate the colors and camera parameters is the same as the one used for the team omni directional camera and is described by Neves *et al.* [137].



a) **b)**

Figure 4.17: Screenshot of a frontal camera frame showing a ball in typical conditions. Left **a)**: original image; Right **b)**: Segmented image.

The algorithm to identify the ball is based on the analysis of blobs of the defined ball color. After segmenting the image searching for the defined ball color, several blobs can exist and each of them is a potential ball. There is the need to induce which blobs can effectively be a ball.

The algorithm (algorithm 1) uses horizontal *scanlines* and each time a ball color pixel is intersected the number of consecutive horizontal pixels of that color is incremented.

All the *scanlines*, represented by their start and end points in pixel coordinates and their length, are kept in a list which is fed to the blob clustering algorithm. The creation of blobs from the *scanlines* is performed by a clustering methodology based on euclidean distance between the pixels classified with ball color.

Each of the resulting blobs is then analyzed to verify if they have the necessary characteristics of a ball:

- The blob must have a minimum global size to be consistent with the size of a ball on the image. We consider 200 pixels as the minimum size, which was verified through testing that it is the average size of the ball at around 7 meters.

- The solidity of the blob must also have a minimum value. We consider 20% as the minimum solidity value to accept a ball. There are two main reasons for the use of a low threshold. The first one is to relax the need for a precise color calibration since the ball in the air is subject to different lighting and a calibration that is more "open" has better chance to detect some parts of the ball in the several conditions. The second reason is that the balls being used in the RoboCup always have some pattern or letters with colors other than the main one, which inevitably reduce the segmented area.

Algorithm 1 Algorithm used for detecting ball candidate based on the search over horizontal scanlines and blob analysis.

Input: segmented image
Output: ballCandidates → list to hold the ball candidates

```
1: scanlineList.clear()
2: for row = 0 to maxImageRow do
3:    for col = 0 to maxImageColumn do
4:       if pixel(row, col) not "ballColor" then
5:          skip to next pixel
6:       end if
7:       nPixels = 0
8:       while pixel(row, col) has "ballColor" do
9:          nPixels ++
10:         col ++
11:      end while
12:      if nPixels > 2 then
13:         currentScanline = computeScanlineLimits(row, col, nPixels)
14:         scanlineList.add(currentScanline)
15:      end if
16:   end for
17: end for
18: blobList = estimateBlobs(scanlineList)
19: ballCandidates.clear()
20: for idx = 0 to blobList.size() do
21:    currentBlob = blobList[idx]
22:    if size(currentBlob) > minSize and solidity(currentBlob) > minSolidity and minWHRel < WHRel < maxWHRel and colorGradient < 0 then
23:       currentBall = computeBallFromBlob(currentBlob)
24:       ballCandidates.add(currentBall)
25:    end if
26: end for
```

- The relation between width and height must be within a set of limits, since the ball is a proportional object. However, given the characteristics of a typical segmentation (figure 4.17b) the relation is usually not as proportional as a ball should be and thus we accept blobs with a relation of up to 4 times the width over the height. To cope with ball dragging on the image, which creates vertical artifacts, we also accept a minimum relation of 0.75.

- Finally, given the characteristics of lighting from above, the color gradient of the blob is also analyzed and only blobs with a zero or less than zero gradient are considered.

The blobs that fulfill the criteria are added to a list of ball candidates, represented by their pixel coordinates and the width and height.

This approach for detection works on a very controlled environment, where the background

of the ball on the image did not possess any element of the same color as the ball. However, it proved to be unreliable when several objects of the same color as the ball are present in the image (figure 4.18). During games, the surroundings of the field are completely uncontrolled and there is a high probability that elements of the same color of the ball are present.

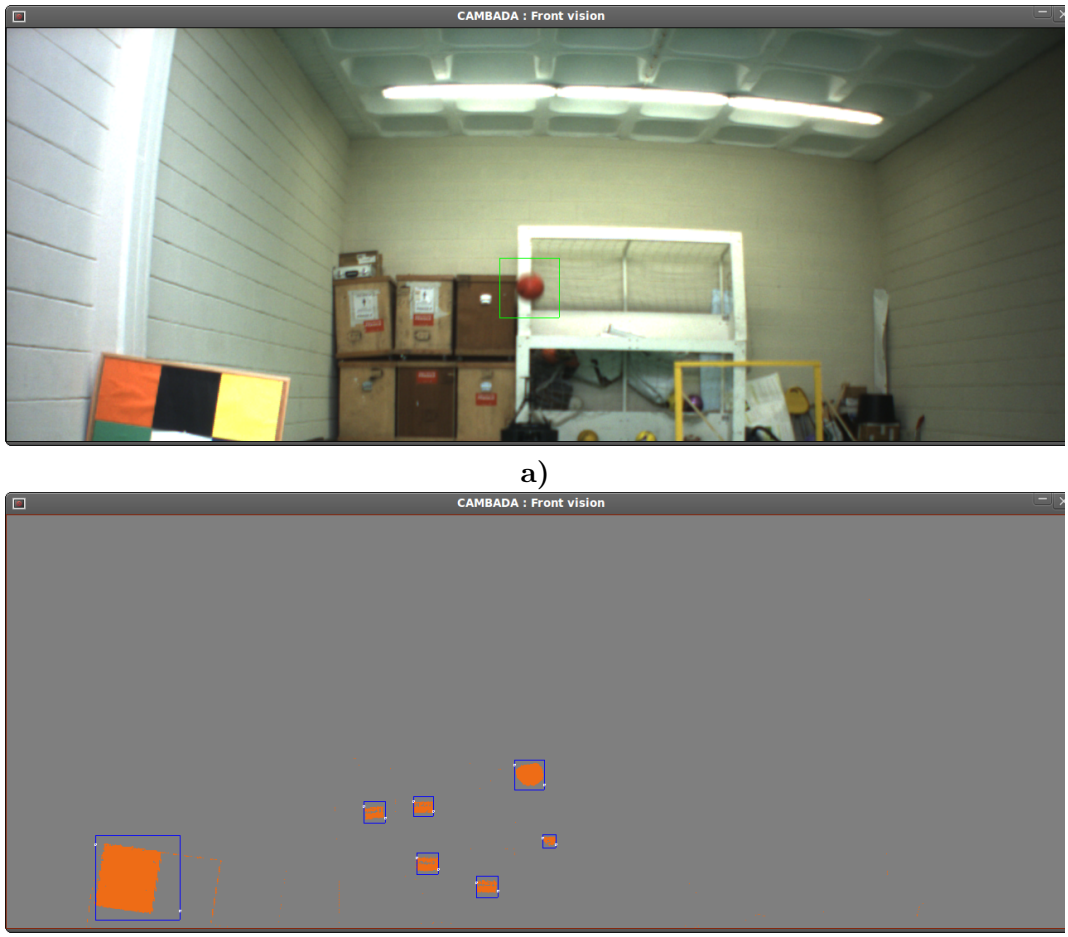


Figure 4.18: Results of the initial purely color approach. All the segmented objects selected in **b)** by blue bounding boxes fulfill the criteria previously described and thus would be candidates passed to the agent.

Second approach

To try to generalize the air ball detection for uncontrolled environments, we use a hybrid approach of color segmentation and statistical analysis of a global shape context histogram.

On a first phase, the color approach described before is applied, although the color range is relaxed so that the color classification also takes into account the lighter and darker areas of the ball.

Each of the blobs that pass the criteria previously defined is used to select a ROI in the original image which contains the possible candidate (figure 4.19b).

These images are then analyzed by a modified global shape context classifier [138]. Each image is pre-processed with an edge detector and a polar histogram is created. This histogram is then statistically analyzed and returns a measure of circularity of the candidate. The edges image is divided in n layers and m angles, creating the polar histogram, as defined by Mario *et al.* [139]. The result of this division creates several areas on the image, as exemplified in figure 4.19c, which are called slices.

The analysis of the histogram is made layer by layer, covering all the angles. An estimation of the average number of edge points on each slice and its standard deviation allows to discriminate between circular and non-circular contours, as exemplified in figure 4.19c. A ratio of edge points is also part of the statistics of the candidate.

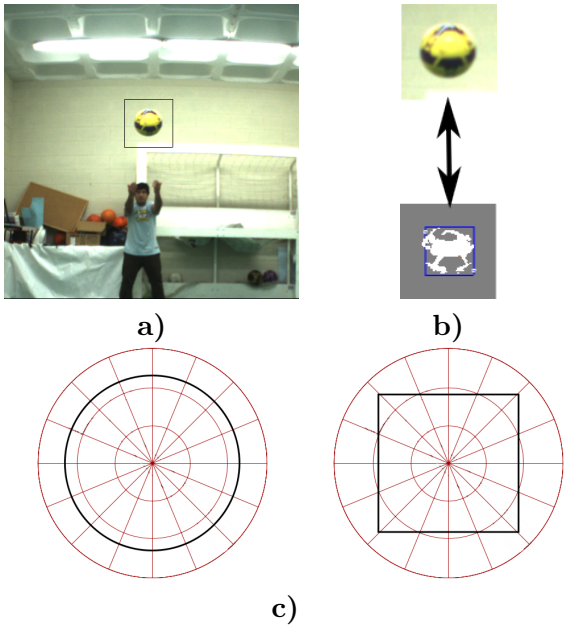


Figure 4.19: **a)** image from the camera with a ROI around the ball, which is the portion of the image used for the histogram analysis; **b)** the ROI created based on the correspondent color blob; **c)** rough representation of the polar histogram. The histogram creation fits its radius with the outer points of the edge image, which is not fully represented in these pictures. *Left*: A perfect circular edge on the polar histogram would look something like this. All the edge points are on the same layer and each of its slices have a similar number of points; *Right*: A square on the polar histogram. Due to the fitting properties of the histogram itself, the square edge points should be divided in more than one layer, which would not yield good statistics as circles.

The statistics of a layer are the average value of contour pixels per slice, the standard deviation of that same count and the ratio of total edge points on the layer relative to the area of the image. The histogram analysis always returns the layer with the best statistics,

which is currently considered the layer that has higher average of contour pixels per slice with minimum standard deviation. This should represent the layer with the most consistent number of edge pixels and thus should be the rounder layer. The next step must then select which of these statistics make sense. Currently, three characteristics are analyzed:

- The ratio of edge points on the layer must be within a given range. This range was empirically estimated through testing of several examples of ball and no ball candidates.
- The order of magnitude of the mean should be greater than or equal to the order of magnitude of the standard deviation.
- The candidate diameter estimated by color segmentation and the diameter estimated by the classifier must be coherent. Since the radius of the histogram is dependent on the number of layers, the coherence between the measures is dependent on an error margin based on the histogram radius.

Experimental results

Several experiments were performed on which the robot was stationary observing the ball with the frontal camera. The experiments consisted on throwing the ball through the air from a position approximately 6 meters away from the camera and in its direction. In the acquired videos the ball is always present and performs a single lob shot.

As expected, since the ball is moving almost directly to the camera, the variation of the ball center column on the image is very small (figure 4.20). The row of the ball center, however, was expected to vary. Initially the ball was being held low on the image (meaning the row of the image was a high value) and as it was thrown, it was expected that it went up on the image, then down again. Figure 4.20 allows us to verify that this behavior was observed as expected.

On the other hand, since the ball is coming closer to the camera every frame, it was also expectable that its size on the image would be constantly growing. The correct evaluation of the width of the ball is important for the position estimation described in the next section.

The main contribution of this color/shape hybrid approach, however, is the reliability of the acquired data, respecting the real time constraints of the application. A test scenario was created, where the ball was thrown in such a way that it was visible in every frame of the videos, and several runs were made. Although the results are strongly affected by the environment around the ball, we obtained promising results with relatively high precision, even if the recall has shown low results. Table 4.2 presents a summary of the detection results for two different configuration parameters:

- **Center** is the coordinates used to consider the center of the polar histogram. The two possibilities are to use the center of the image or the center of the contour detected by

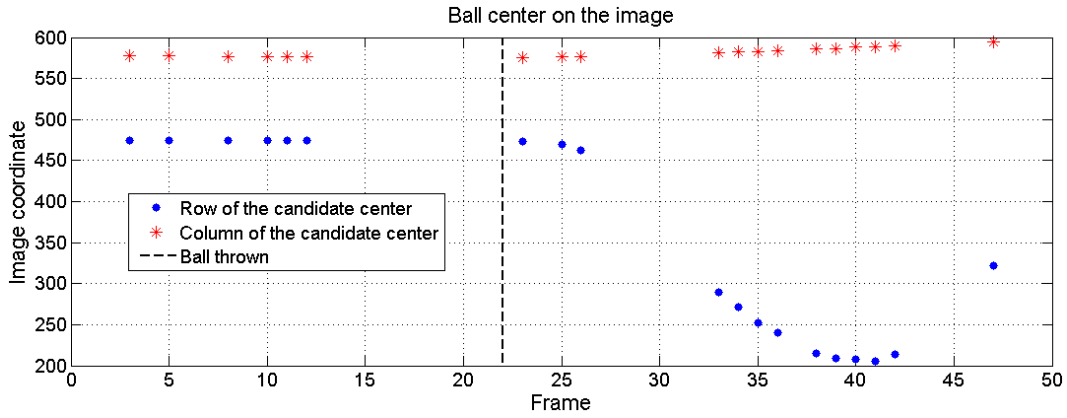


Figure 4.20: Results from the image analysis of a ball being thrown in the camera direction: values of the row (Blue dots) and column (Red stars) where the ball center was detected on the image.

the shape analysis prior to histogram creation.

- **Layers** is the number of layers used for building the polar histogram. The two presented values are 2 and 3 layers.

The results correspond to the analysis of six different ball launches saved to video. The videos names are on the top column of each table section, "Kick<BallColor>N", followed by the total number of frames of the video. The numbers of true positives, false positives and false negatives are presented, as well as the number of frames where there was no ball on the image. Finally, the precision and recall achieved for each situation is presented. Red and green numbers indicate worst and better results, respectively. The decision of the configurations to use were based on the generally higher recall between the 2 layer configurations, since the 3 layer configuration has a huge amount of false positives. Excluding one situation, using the contour center provided lower recall values and fewer true identifications, thus the image center was chosen.

A pure color approach yielded better results in a very controlled environment, but in a more uncontrolled environment we obtained a very low precision. We consider that having a higher precision is more advantageous, even when facing the loss of some recall. The processing time for the algorithm was 11.9 ± 2.8 ms which is still within our time restrictions. The frontal vision process runs on a best effort processor scheduler and the used laptop has two processors. Despite the fact that the remaining processes run with top priority, only a small part of the vision process is divided into the two processors and thus the frontal vision process runs with little delay.

On the same frame presented in figure 4.18, this second approach based on circularity only validated one of all the candidates produced by the initial color approach. In this case,

	KickRed01(16)				KickRed02(28)				KickRed03(28)			
Center:	**image**		**contour**		**image**		**contour**		**image**		**contour**	
Layers	2	3	2	3	2	3	2	3	2	3	2	3
True pos	6	9	3	6	11	13	3	8	7	16	7	9
False pos	0	0	0	0	1	0	0	2	0	0	0	2
no ball	0	0	0	0	0	0	0	0	0	0	0	0
False neg	10	7	13	10	17	15	25	20	21	12	21	19
Precision	1	1	1	1	0.92	1	1	0.8	1	1	1	0.82
Recall	**0.38**	0.56	0.19	0.38	**0.39**	0.46	0.11	0.29	**0.25**	0.57	0.25	0.32

	KickYellow01(26)				KickYellow02(33)				KickYellow03(31)			
Center:	**image**		**contour**		**image**		**contour**		**image**		**contour**	
Layers	2	3	2	3	2	3	2	3	2	3	2	3
True pos	7	9	3	9	2	8	7	11	4	6	2	7
False pos	3	*31*	1	*26*	6	*38*	3	*25*	7	*35*	0	*32*
no ball	9	9	9	9	14	14	14	14	13	13	13	13
False neg	10	8	14	8	17	11	12	8	14	12	16	11
Precision	0.7	0.23	0.75	0.26	0.25	0.17	0.7	0.31	0.36	0.15	1	0.18
Recall	**0.41**	0.53	0.18	0.53	**0.11**	0.42	0.37	0.58	**0.22**	0.33	0.11	0.39

Table 4.2: Color/shape hybrid detection algorithm results. The green values on the *Recall* lines indicate the chosen parameterization (histogram with 2 layers centered on the image). See text for details.

presented in figure 4.21, only that one candidate (surrounded in the image by the yellow circle) would be sent to the agent for processing.

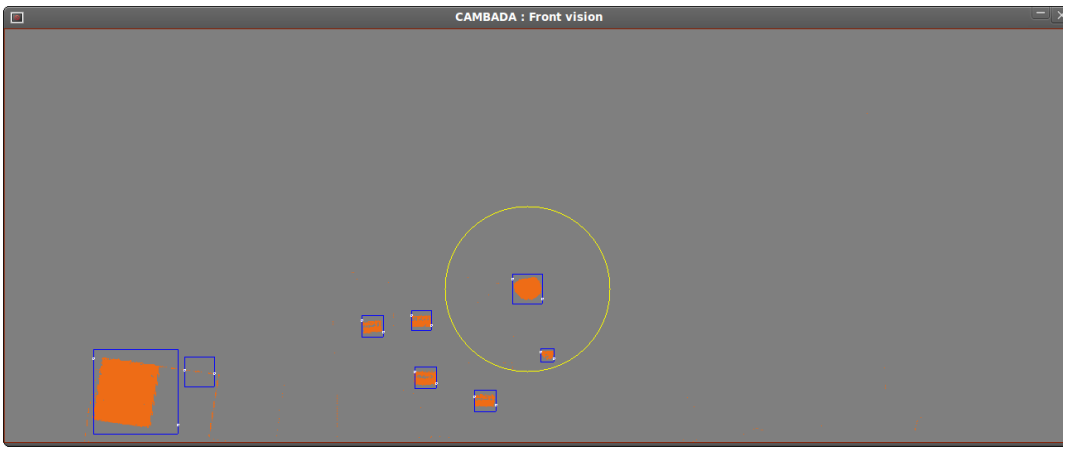


Figure 4.21: Results of the color/shape hybrid detection algorithm. In this case, the shape step selected a single color candidate from the several possibilities.

4.3.3.3 Ball position estimation

After having the candidates selected as balls, there is the need to estimate their position. To accomplish that, we first analyze the candidate radius in pixels. The size that each pixel represents at each distance increases with distance to the camera focal point. This is due to the fact that the resolution is constant but the horizontal FOV at any given distance is not. We can estimate the theoretical horizontal FOV at any given distance through expression 4.4.

Table 4.3 presents the expected FOV at several distances, from one to nine meters, in steps of 1 meter.

Distance to camera	1	2	3	4	5	6	7	8	9
Expected width	1.56	3.13	4.69	6.26	7.82	9.39	10.95	12.52	14.08
Expected Height	1.25	2.50	3.75	5.01	6.26	7.51	8.76	10.02	11.27

Table 4.3: Table with the theoretical Field Of View at some setpoint distances. Values are in meters.

With the FOV width relation, we can estimate the size that each pixel represents at each given distance, by a relation of the estimated distance and the horizontal resolution

$$pS = \frac{hFOV}{hR} \tag{4.5}$$

with:

- pS is the pixel size in the plane with the defined horizontal FOV ($hFOV$);
- hR is the CCD horizontal resolution;

and thus, since we know that the ball has 0.22 m, we can estimate the number of pixel expected for the blob width at a given distance:

$$pW = \frac{0.22}{pS} \tag{4.6}$$

where:

- pW is the expected ball width, in pixels, for the plane with the given pixel size.

For the same setpoint distances, from 1 to 9 meters, the ball width in pixels was estimated. Table 4.4 presents those results.

From this analysis, Equation 4.6 can be developed using Equations 4.5 and 4.4, resulting in an inverse relation function of pixel width, pW, and distance to camera focal point, Y.

$$pW = \frac{0.22 \times hR}{hFOV} = \frac{0.22 \times hR}{2 \times Y \times tan(\alpha)} \tag{4.7}$$

Distance to camera	1	2	3	4	5	6	7	8	9
Expected ball width	182	91	61	46	36	30	26	23	20

Table 4.4: Table with the theoretical ball width at several distances. Distances are in meters, ball widths are in pixels.

Given the known distance of the ball candidate and the linear relation of the pixel size, we can estimate the XX coordinate. This is true due to the camera positioning on the robot that, besides having the focal axis parallel to the ground, is also coincident with the robot YY axis. To accomplish the estimation of the XX coordinate we have to analyze the horizontal pixel coordinate of the ball center from the image center and apply the calculated pixel size.

We can thus obtain the XX and YY coordinates of the ball on the ground plane, relative to the frontal camera, from which we know the relative coordinates from the center of the robot. Since the camera is aligned with the YY robot axis, the XX coordinate is direct, while the YY coordinate is a simple displacement of +5 cm, since the camera is just this distance away from the robot center (figure 4.15).

Experimental results

An experimental analysis was performed to verify the relation between the detected ball width in pixels and the distance it is from the camera.

Unfortunately, it was verified that the expected theoretical values of the ball pixel width according to the distance was not verified in practice. Through the experimental setup, a more appropriate relation was found by taking measurements.

The experiment was performed by placing the robot such that the camera had its axis along a field line and placing the ball in front of it. The ball was on top of a support, which maintained ball height, and was iteratively placed at several setpoint distances from the camera (from one to nine meters). These distances were measured with tape and used as ground truth data. The ball pixel width was measured by manually stopping the video at frames corresponding to the setpoint distances and verifying the estimate width of the detected ball. The results are presented in Table 4.5.

Distance to camera	1	2	3	4	5	6	7	8	9
Measured ball width	200	110	72	52	42	32	28	22	26

Table 4.5: Table with the measured ball width at several distances. Distances are in meters, ball width are in pixels.

Based on the values for the measured relation between the ball pixel width and the distance to camera, a 3rd degree polynomial function was found to better fit most of the setpoints

The function input is a ball blob's width and the result is the estimation of its distance to camera (the YY coordinate). This relation is different from the theoretical relation presented in Equation 4.7, maybe due to factors like lens distortion that were not accounted in the previous analysis, misinformation about the hardware characteristics or even assembly factors that may affect the output image, even if the hardware characteristics are exactly the ones described in the products data sheet.

The estimation of the distance of a ball candidate to the camera is then achieved by a function that, given a pixel width of a given candidate, returns the distance at which that candidate is from the camera.

To achieve the desired polynomial function, several were tested to visually verify the fitness of each of them. Figure 4.22a shows the results for the second, third and fourth degree polynomial functions. It is observable that the second degree polynomial clearly unfit for the setpoints. The third and fourth degree functions both have a satisfactory fit for most of the points, becoming unusable for the last two setpoints on the XX axis (larger width at shorter distances). Given the need for a different approach for these shorter distances, we opted to use the third degree function, for the sake of a lower complexity.

For shorter distances, a linear approximation was found to fit the two points and thus was used to "complete" the distance estimation function (figure 4.22b). The chosen separation point was the ball width of 100, which is the lowest point of the used decreasing part of the third degree function.

In the same experiment of throwing the ball from 6 meters away from the camera, described in Section 4.3.3.2, the results of the positions evaluated by the previously described algorithm were captured. Figure 4.23 depicts these results. The path formed by the estimated XY positions approximates the path of the ball arc through the air, projected on the ground. This data allows a robot equipped with such camera to estimate the path of the incoming airborne ball and place itself in front of the ball, for intercepting it with its body and defend from a possible goal, in the case of the goal keeper. Given the nature of this main goal keeping task, we can cope with just a general direction of the ball, even if the position estimation does not have a very high precision. However, it is clearly visible in the image that the detection and position estimation result in a straight line, as expected.

The frontal vision process, from capture to the production of the XX and YY coordinates took an average time of around 12.5 ms to execute in a computer with an Intel Core 2 duo at 2.0 GHz. The tests were made for the frontal camera process running standalone at 30 frames per second. These times should be enough to obtain readings from the frontal camera with minimum loss of camera frames due to the fact that currently only the vision process uses separate threads and thus the frontal vision process should have a free processor to use for more than 20 ms since the omni vision process takes a mean of 12 ms to run.

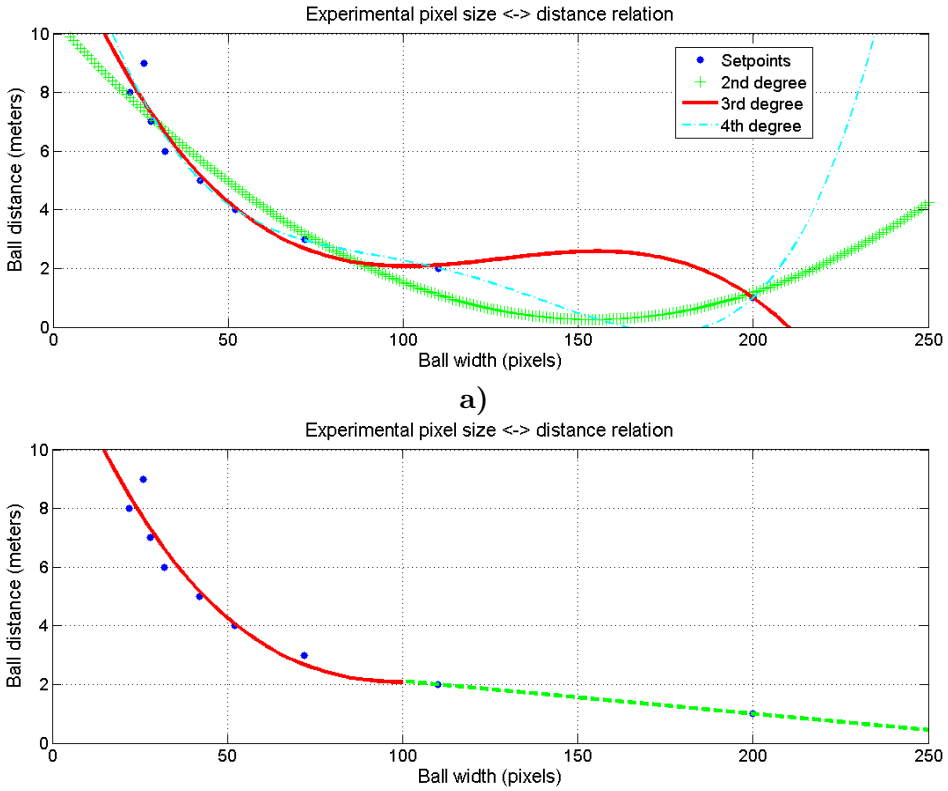


Figure 4.22: Left **a)**: Several degree polynomials fitting the setpoints. It is clear that the 2^{nd} degree does not fit properly any point. The 3^{rd} and 4^{th} fit similarly below 100 pixels width; Right **b)**: Third degree polynomial function (red line) fitting the defined experimental setpoints (blue dots) and linear polynomial function (green dashed) for the sizes corresponding to closer distances.

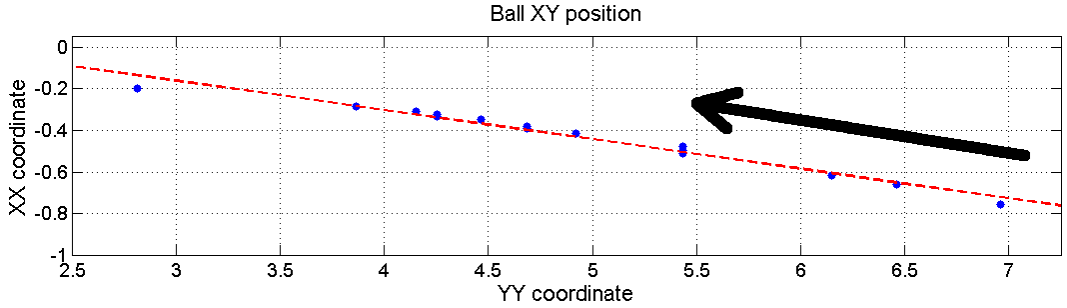


Figure 4.23: Picture of a data capture of a ball kicking test. The ball was thrown by the air from a position around (-0.5, 6.0) in the approximate direction of the camera (which is the origin of the referential). The blue dots represent the estimated ball positions, the red dashed line is the linear approximation obtained with the measured points. The kick direction is indicated by the black arrow.

4.3.3.4 Frontal camera ball integration

Being the ball the main element of a soccer game, its information is very important and needs to be as precise as possible. Failure on its detection can have very negative impact on the team performance. Probably even worse than failing to detect the ball (situation on which the robots can strategically move in search for it) is the identification of false positives on the ball. This can deviate the attention of a robot or even the entire team from the real ball, which can be catastrophic. To avoid false positives and keep coherence on the information of the ball, several contextual details are taken into account.

Given the several possible sources of information, the priority source for ball position is the position given by the omni directional camera. Details about the visual ball detection and validation on the omni directional camera can be found in [137]. The next source to use is the shared information between team mates, because if they see the ball on the ground, there is no need to check the frontal camera information. Finally, the agent tries to fit the information from the frontal camera into the worldstate (figure 4.24).

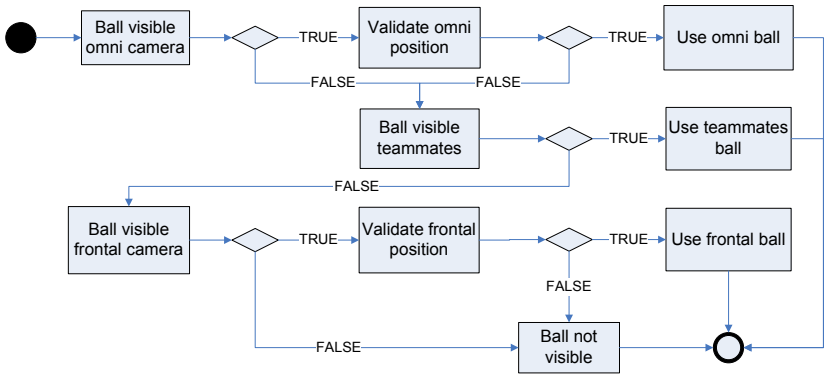


Figure 4.24: Ball integration diagram when using a frontal RGB camera.

The integration of the perspective camera data must take into account some historical and context information. The integration process, responsible for handling and filtering the ball information as presented before, analyzes the list of candidates available by the perspective vision process. Some validations are performed to filter the information and select a suitable candidate from the perspective balls list.

- The first analysis to make is whether the ball candidate is inside the field or not. If it is outside the field of play, there is no need to use it, as even if it is truly the ball, the game would be stopped.

- To maintain coherence between the perspective vision ball and the omni vision ball, an analysis of the last omni camera ball positions is made and a perspective ball candidate is considered only if that candidate position is within a given vicinity of the last omni

camera ball position. When the ball rises from the ground, the omni camera position is affected, as the projection on the ground will evaluate the ball further than it is in reality. For that reason the angular difference is the measure considered for defining the vicinity, since the direction of the omni detection should not be affected by the projection effect. A candidate from the perspective camera is only accepted if the difference to the last omni camera ball position is bellow a few degrees. This validation is performed on the first detections by the perspective camera, when the ball has also just became or is becoming not visible for the omni directional camera.

- After "choosing" the perspective camera ball candidate, it should be visible in the next few frames. Thus, after fixing on a candidate, further frames are analyzed based on that candidate last position, meaning that the choice of a candidate is made based on a vicinity of the last perspective camera ball position. This vicinity, contrary to the omni camera one, is based on distance. The reason for tracking the same perspective camera ball is that, when detected by the perspective camera, the ball should very soon become invisible to the omni camera. The cameras work in an almost exclusive way, since the omni directional camera is very limited in terms of ball height, not being able to detect any ball that is above 50 to 60 cm. However, the ball position from the omni directional camera has some persistence that is useful for many decision aspects and to guarantee that a few frames without detecting the ball will not cause a gap in the robot's knowledge and decision. Frames without detecting the ball easily happen because of, for instance, a photography flash or the ball passing behind a robot. Thus, when the ball becomes visible by the perspective process, it is still persistent on the omni directional information.

- Another filtering performed is an analysis of the number of cycles with the ball visible on the perspective camera. Again, the objective of the perspective camera is to detect aerial balls. During the game, the ball leaves the ground only when it is kicked by a robot and the ball trajectory takes no more of no more than 2 seconds. When a kick raises the ball, it will inevitably be in the air for only those few instants, which can be periodic if the ball bounces several times, but still the appearances are short. A ball constantly detected for more than a pair of seconds is then discarded, since it is probably a false positive or, for instance, a stop game situation and the referee is holding the ball on his hands.

4.3.3.5 Ball visual detection and validation using a Kinect sensor

The analysis of the performance of the previously described approach showed that there are some limitations to its use on a real game scenario, mainly due to the fact that the ball, moves at a very high speed. In many frames of a video capture, it is verified that the distortion

blur is very high and thus, the shape analysis is compromised, forcing us to wide the range of detection, thus lowering the effectiveness of the algorithm. Also, as visible in table 4.2, the recall and in some situations the precision of the approach were rather low.

In an effort to achieve a more effective solution, a different approach was designed [140]. Given the restrictions of an RGB camera within the "ball on any background" scenario and considering the extremely difficult task of getting a good shape detection within useful processing times, we are now using a depth sensor instead of a color camera.

As 3D sensor, a Kinect was chosen given its low price, its ability to directly provide 3D depth information and its refresh rate of 30 fps similar to most of the RGB cameras available.

Besides the sensor availability factor, the software availability was also a decisive factor. The Kinect sensor counts with a wide variety of software drivers and utilities for various platform, provided by the huge community working with it. The driver used by our approach is the *libfreenect*, which is one of the most commonly used drivers for using Kinect sensors. In our application, the acquisition of the 3D data from the Kinect is done using a C++ wrapper for *libfreenect* that transforms depth images into and OpenCV matrix. The transformation from raw data to metric is done using a formula found in an online manual [141]. The type of data that we get with a point cloud is represented in figure 4.25, in this case with the RGB camera pixel data mapped to the visual model.

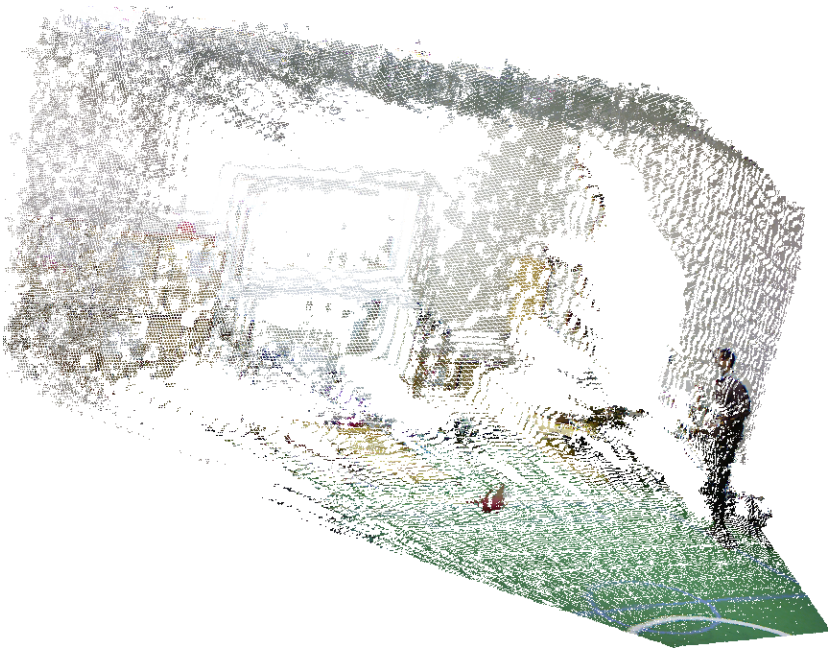


Figure 4.25: Representation of a kinect point cloud captured by the depth sensor with the RGB camera pixel data mapped into the cloud.

Ball detection algorithm

Despite the limitations inherent from the discretization of the space, more visible at higher distances, the use of this information still seems to be a good option.

The approach we opted to use for detecting aerial balls using the Kinect is to detect flying objects within the Kinect field of view. To achieve this, given the properties of a flying ball in the MSL environment, we decided to voxelize the space to work in an occupancy voxel space rather than considering the whole cloud of points.

This step, besides allowing an increase of the process speed by reducing computation, also allows us to define a flying object as an object that occupies a given number of voxels with a minimum number of points and whose surrounding voxels are empty. It can be thought as a 3D mask with the inside voxels non empty (containing a minimum number of points) and the outside voxels empty. Figure 4.26 presents an example of such a mask.

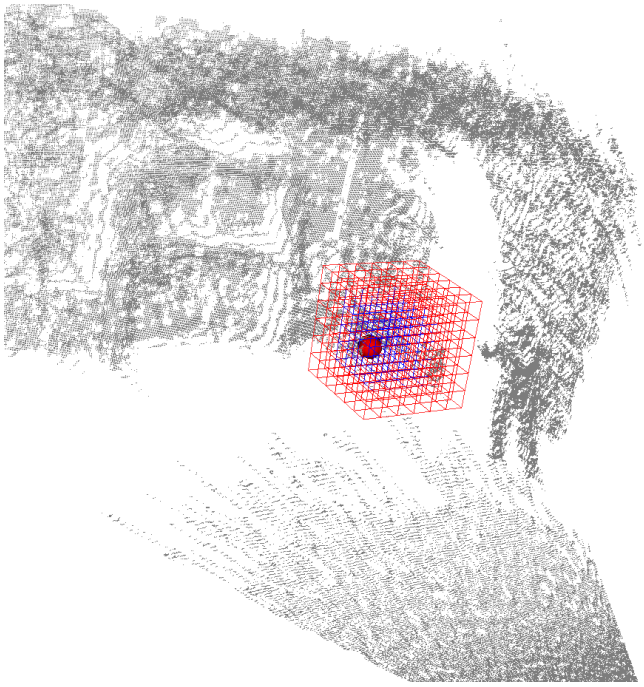


Figure 4.26: Example of 3D Mask used for flying object identification.

The values used for the grid and the mask obviously depend on the size of the ball to detect. However, they have to be defined taking into consideration two main aspects: *1)* the grid size must be small enough to allow that a real flying ball, when voxelized, does not become smaller than the space between any two planes, which could make us loose the information of the ball. This issue becomes more hazardous at farther distances; *2)* the grid mask must be large enough to accommodate a volume larger than the ball, since some blurring is inevitable due to the high speeds achieved by the ball.

With this mask approach, we rule out false positives from any good detection of other objects on the field of play, since all other artifacts during a game can only be a robot or a human. Since all of them have a clear "connection" with the ground, the mask will not allow a valid detection. Of course false positives can still happen due to noise on the Kinect sensor detection.

Also, any object that could effectively be identified, at this point, as a ball by the mask can be outside the field of play. Since this is a complementary vision system for our robots and since they know their position inside the field of play, further integration steps will be responsible for handling these possible false positives.

Detection results

To validate the detection methodology and estimate its effectiveness, a series of captures were performed on a static setup, so that all the distances and positions evaluated were known through measurements with a tape.

The setup was composed by a kinect sensor with its center at 0.65 m and with its focal axis parallel to the ground and a ball fixed by a thin wire at a height of 1.2 m. The kinect vision process, in each cycle, registered the 3D position of the detected ball to a file and one file was created for each distance from 2 m to 7 m in steps of 1 m. The kinect measurements are based on an axis where the YY is the distance to the sensor along its focal axis, XX is to the right of the sensor and ZZ is to the top of the sensor.

Each capture has some hundreds of instances and, given the static scenario, it is known that the ball was present in every frame of the capture. The detection rate obtained was good for the several distances involved up to 6 m. At 7 m, a lower detection rate was verified which makes us assume it to be the limit distance to use in the detection algorithm. During these captures, there was a single false positive in the 7 m capture. Table 4.6 shows a summary of the experiment, with the mean position measured at each distance and the standard deviation of those measurements, as well as the detection rates and false positives at each distance.

Setup position			Mean			Std			Detection rate	False pos
XX	YY	ZZ	XX	YY	ZZ	XX	YY	ZZ		
0.0	2.0	1.2	0.04	1.98	1.18	0.001	0.004	0.001	100%	0
0.0	3.0	1.2	0.06	3.00	1.19	0.004	0.005	0.001	100%	0
0.0	4.0	1.2	0.06	3.96	1.20	0.004	0.009	0.002	100%	0
0.0	5.0	1.2	0.05	4.94	1.20	0.010	0.012	0.004	97%	0
0.0	6.0	1.2	0.00	5.90	1.19	0.004	0.019	0.005	86%	0
0.0	7.0	1.2	0.03	6.82	1.21	0.007	0.039	0.006	17%	1

Table 4.6: Summary of the ball detection algorithm for static ball. All values are represented in meters.

The measured height (ZZ coordinate) is the total height of the ball from the ground, meaning it is compensated with the height of the sensor to the ground, which is, as already mentioned, 0.65 m.

Figure 4.27 illustrates the results of these captures. For simplification of the perception of the 3D positions, two separate views are presented. A side view is presented, with the projection of the ball positions on the YoZ plane, which is the distance of the ball to the sensor and the height of the ball to the ground. Also a top view is presented with the projection of the ball positions on the XoY plane, which is the distance of the ball to the sensor and the sideways distance of the ball to the center of the sensor focal axis.

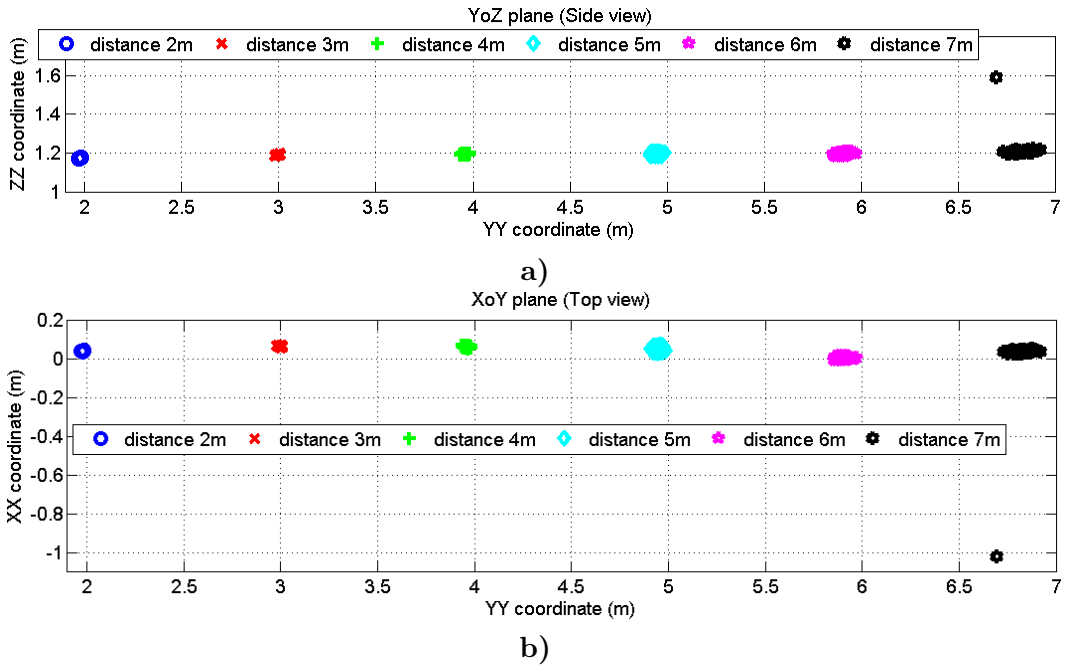


Figure 4.27: Results of the detection of a static ball with the Kinect. **a)** Coordinates of the ball projected on the YoZ plane. **b)** Coordinates of the ball projected on the XoY plane.

Given the results obtained, specially when taking into account the low deviation of the measurements, we consider the approach fit for use. One must also consider that the mean and standard deviation for the 7 m distance was estimated without considering the false positive. The reason for this was that we consider that a false positive (which is more than 1 m away) is detectable by the integration module and thus is not considered as a noisy position and would be discarded.

Ball trajectory estimation

Ball detection allows us to have information about its whereabouts in any given instant,

but we must use this raw information to improve our robot perception. In robotic soccer for example this algorithm might be useful as long as the goalkeeper can estimate the trajectory of the ball to move in a position to prevent a goal.

For flying objects, and considering that air resistance is negligible, the trajectory can be approximated by a simple ballistic trajectory. To perform this evaluation, we keep trace of the last ball positions. Since during a kick the ball is on the air for about 1 second and given the obtained detection results with a mean of 10 detections per kick, we are currently keeping a history of the previous 10 positions for estimating the trajectory.

Each time a new ball is detected, we verify the Euclidean distance between the current point and the existing trajectory. If the point supports the trajectory by being close enough (currently 0.25 m, which means closer than the distance equivalent to one ball) a new trajectory is estimated by using the last supporting ball positions, including the one that was just added to the list.

The general algorithm for managing the estimated trajectory is presented in algorithm 2.

Algorithm 2 Algorithm used for managing trajectory estimation.

Input: ballPos → position of the candidate
positionsList → circular buffer list of last positions detected by the Kinect partOfTrajectoryList → circular buffer list to flag the correspondent positions as part of trajectory
Output: trajectory → an updated or empty trajectory

```
1:  if ballPos.exists() then
2:    positionsList.add(ballPos)
3:    if trajectory.exists() then
4:      if trajectory.distanceTo(ballPos) < minDistance then
5:        partOfTrajectoryList(ballPos) = true
6:      else
7:        partOfTrajectoryList(ballPos) = false
8:      end if
9:    else
10:     partOfTrajectoryList(ballPos) = false
11:   end if
12: end if
13: if not partOfTrajectoryList(currentCycle) and not partOfTrajectoryList(lastCycle) then
14:   trajectory.reset()
15:   trajectory.initializeFromLastPoints()
16: else
17:   if trajectory.exists() then
18:     trajectory.refineTrajectory()
19:   end if
20: end if
```

Figure 4.28 shows the results of the trajectory estimation for a flying ball. The history of balls used for the computation is presented as large red spheres and the parabolic trajectory

estimated is represented by the small blue spheres.

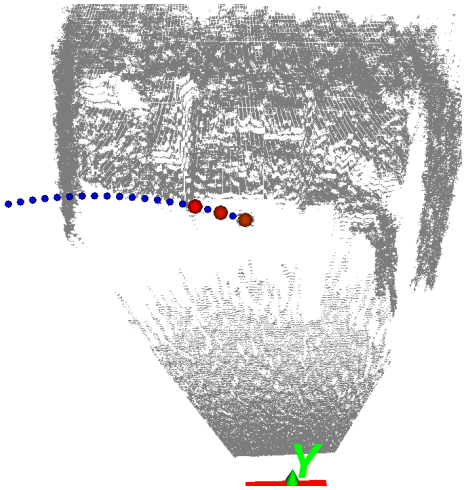


Figure 4.28: Example of trajectory estimation where the small blue spheres represent the trajectory computed from the balls in the history (large red spheres).

With the trajectory estimated, we can easily give the agent data about the current projection of the ball position on the ground and even the predicted touchdown point of the ball.

The integration of this information is similar to the previously described in section 4.3.3.4 for a RGB perspective camera. However, given the obtained results, we have a higher confidence in the detections by this new sensor and thus most cross validations performed should be simplified and the priority of the integration sources should change (figure 4.29).

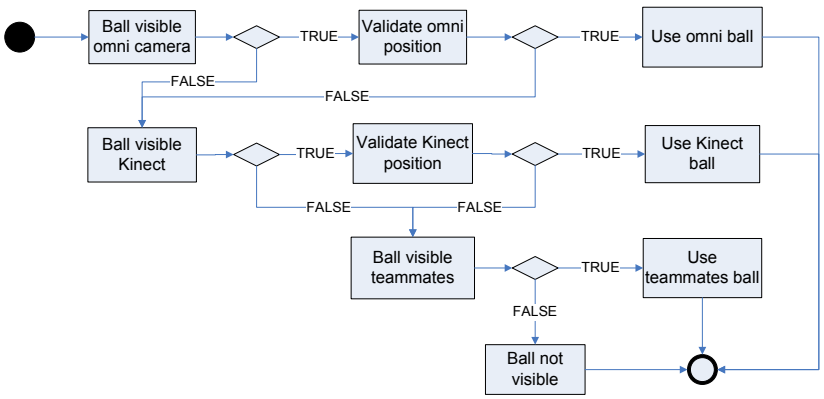


Figure 4.29: Ball integration diagram when using a Kinect sensor.

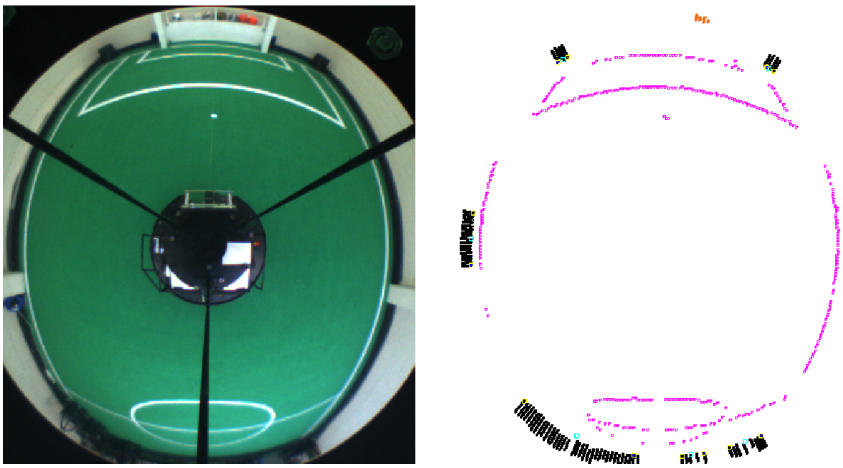
4.4 Localization

Self-localization of the agent is an important issue for a soccer team, as strategic moves and positioning must be defined by positions on the field. In the MSL, the environment is partially known, as every agent knows exactly the layout of the game field but does not know the position of any other elements, either itself, other robots or the ball. Given the known map, the agent has then to locate itself.

The CAMBADA team localization algorithm is based on the detected field lines, with fusion of information from the odometry sensors and an electronic orientation sensor [122]. It is based on the approach described by Lauer *et al.* [24] and presented in section 2.2.1.4, with some adaptations. It can be seen as an error minimization task, with a derived measure of reliability of the calculated position so that a stochastic sensor fusion process can be applied to increase the estimation accuracy [24].

From the center of the robot in the image, radial sensors are created around the robot, each one represented by a line with a given angle. These are called *scanlines*. The image processing, in each cycle, returns a list of positions relative to the robot where the *scanlines* intercept the field line markings [118].

The idea is to analyze the detected line points, estimating a position and, through an error function, describe the fitness of the estimation. This is done by reducing the error of the matching between the detected lines and the known field lines (figure 4.30). The error function must be defined considering the substantial amount of noise that affects the detected line points which would distort the representation estimation [24].



a) **b)**

Figure 4.30: Captures of an image acquired by the robot camera and processed by the vision algorithms. **a)** the image acquired by the camera. **b)** the same image after processing with magenta dots over the detected field lines.

In normal operation mode, the localization is done over a limited set of base positions from which tracking is maintained. Since it is an algorithm based on optimization and since there are many local minima, the tracking only works satisfactorily if the estimations are near the solution. In situations where the robot does not possess a valid estimation, a global localization algorithm estimates the robot position on the field using a much wider set of initial estimations over which the already referred error minimization process for optimization is applied. However, this global localization algorithm is computationally heavy and time consuming. For that reason, after having an initial position, the simpler tracking localization handles the cyclic relocation, performing the error minimization process over a base pose, predicted by applying current cycle odometry to the last pose, and another four poses around the base one in four opposing directions, at distances equivalent to one robot.

Although the odometry measurement quality quickly degrades with time, within the reduced cycle times achieved in the application, consecutive readings produce acceptable results and thus they are used to produce an initial approximation of the new position of the robot. The fusion is based on the application of the odometry displacements directly to the previous robot pose thus providing a prior estimation of the current pose which will then be updated by the visual information. The minimization performed over the odometry estimated pose will generally converge quicker, since this is closer to the current pose than the previous cycle pose. This approach also allows the agent to estimate its position even if no visual information is available. However, it is not reliable to use only odometry values to estimate the position for more than a few cycles, as sliding and frictions on the wheels produce large errors on the estimations in short time.

Due to the nature of the approach, this algorithm works acceptably with a relatively low number of points over white lines, like a few tens of points, as long as they are representative of the surroundings. Consider the case of matching a 90 degrees corner. If the algorithm had access to 200 points all over the same line, it would not be capable of matching the corner. On the other hand, with only 20 or 30 points scattered over both lines, the algorithm would be capable of detecting the match. Even in situations where the points are over the same line, the merging with odometry and position tracking provide a good robustness to the algorithm [24], as long as the situation is temporary, which is usually the case.

The visually estimated orientation can be ambiguous, i.e. each point on the soccer field has a symmetric position, relatively to the field center, where the robot detects exactly the same field lines. To disambiguate the symmetry problem and to detect wrong estimations, an electronic orientation sensor is used [88]. The orientation estimated by the localization algorithm using the visually detected line points is compared to the orientation given by the sensor and if the error between them is larger than a predefined threshold, actions are taken. If the error is really large (i.e. around $\pm$ 180 degrees), it means that the robot estimated orientation is symmetric to the real one, so it should assume the mirror position. On the

other hand, if the error is larger than the acceptance threshold (i.e. a 90 degrees acceptable area), a counter is incremented (figure 4.31).

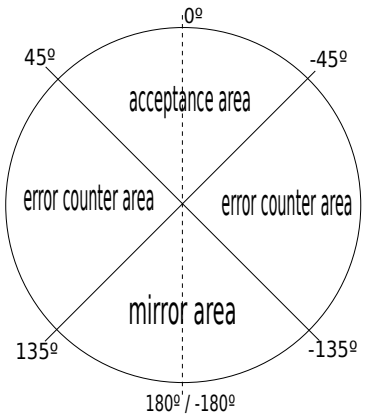


Figure 4.31: Illustration of the compass error angle intervals.

This counter will be incremented every cycle in which the error is greater than the threshold. If a given number of consecutive cycles with high errors is reached (i.e. the counter reaches a given number, currently 10), the robot considers itself "lost", meaning that it will not continue to track its position but will instead consider the initial situation, with no a-priori knowledge and thus executes the global localization algorithm. Figure 4.32 shows situations where the threshold was reached and relocation was forced after some cycles.

4.4.1 Robot heading sensor

When the field became symmetric the need for a heading sensor to facilitate agent initialization and recovering from drift situations became clear. The use of this heading sensor has already been presented in section 4.4 and its effectiveness shown. The sensor, however, has not been the same since the work started, as new issues emerged over time.

Initially, the used sensor was a single electronic compass placed near the top of the robot that provided an angular measurement. This approach proved to be effective and robust and, with it, the previously defined localization recovering algorithm was implemented. However, given the variety of fields and venues where the team plays, it was detected that, in some of the competition sites, the presence of magnetic fields around the field of play make the use of the compass impossible, since the values returned by the sensor are not coherent. This eventually became a problem and the sensor was changed.

Currently, the robot is equipped with an Inertial Measurement Unit (IMU) composed by accelerometers, gyroscopes and a compass. The current approach for measuring the heading of the robot is to use the yaw value of the IMU. This value tends to be more steady than the compass value, even when there are no magnetic fields hindering its measurements.

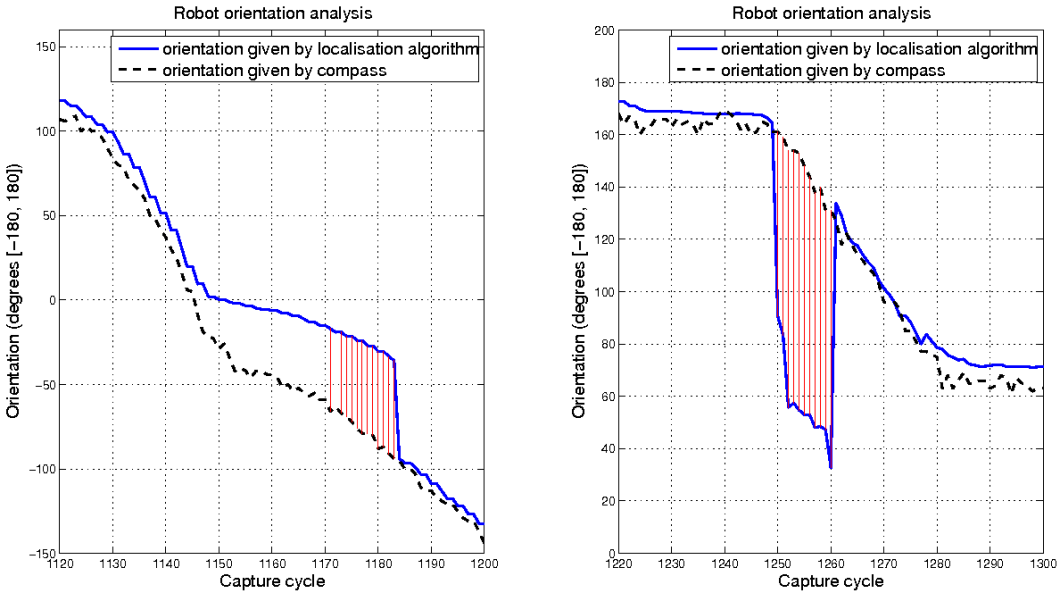


Figure 4.32: Illustration of two situations where relocation was forced. Dashed line represents the angle given by the orientation sensor, solid line represents the angle estimated by the localization algorithm, red lines represent the cycles on which the error between the two angles is greater than the threshold. **a)** the camera was covered while the robot moved. The estimated orientation error degrades progressively and after getting higher than a threshold, the cycle count starts and forces relocation. **b)** the robot tilted. The estimated orientation error is immediately affected by more than a threshold and the cycle count starts and forces relocation.

To understand the real effect of the magnetic fields on the compass and how the IMU yaw value can be more stable than the compass, a test scenario was created. In this scenario, the robot moves around the field while keeping its orientation fixed. During the movement, the robot localization algorithm was running normally and the heading sensors were not taken into account so that there was no chance of a re-localization to occur. The robot orientation was visually confirmed to be the expected one and constant throughout the movement.

This allows us to estimate the variation that the orientation sensors measure in several places around the field. The trajectory performed by the robot is represented in figure 4.33 and the complete tour took a total of 70 seconds.

The captures were made in our lab field, which turned out to be one of the fields where the effect of magnetic fields on the compass is more substantial. In figure 4.34a, the raw values of the compass along the capture tour are presented. It is clearly visible that the value of the robot orientation is completely unreliable throughout the field, with huge variations that even include more than 360º in two occasions. Considering the algorithm presented before in section 4.4, even if the acceptance margin of figure 4.31 was greatly increased, the algorithm would not work and would consider the robot lost in many occasions.

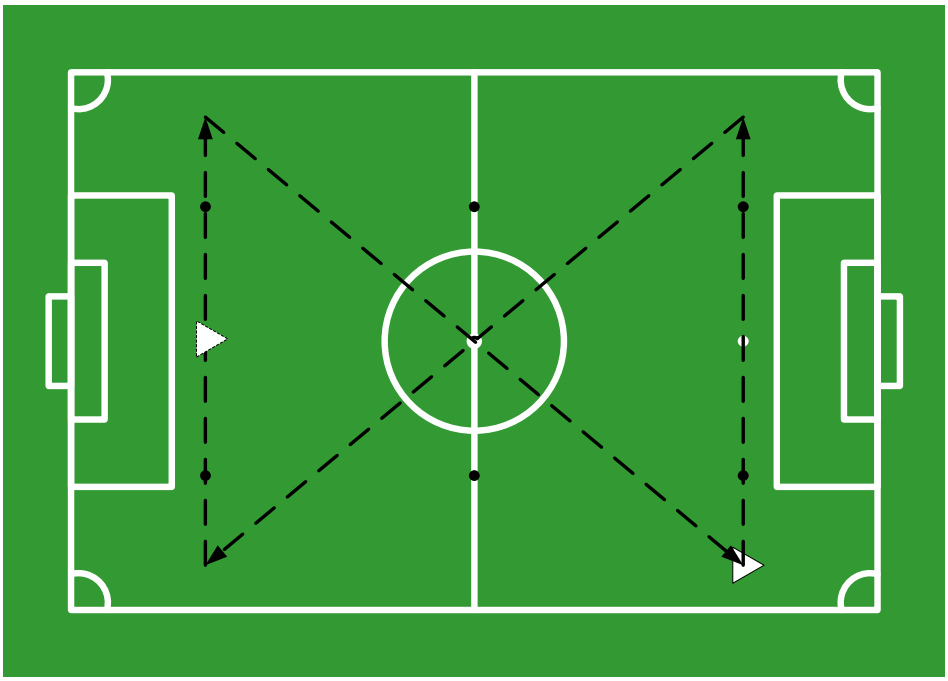


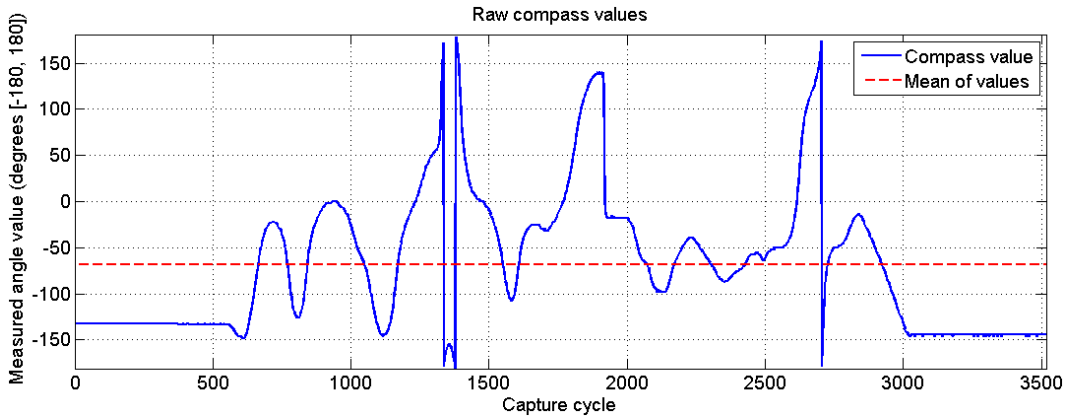
Figure 4.33: Tour performed by the robot while capturing the raw values of both the electronic compass and the IMU yaw angle. The orientation of the robot during all the capture was facing the left goal, as shown in the figure. The represented tour was performed in 70 seconds.

On the other hand, in figure 4.34b we have the representation of the raw values of the yaw measured by the IMU. By looking at the figure, we can easily verify that the variation of the robot orientation is very stable throughout the field. With these values, we can estimate a meaningful mean which is, in this case, -6.29 ± 1.92. With the IMU yaw, we get a maximum variation of less than 10^{o} from the mean value, which is well within the acceptance margins of the robot orientation.

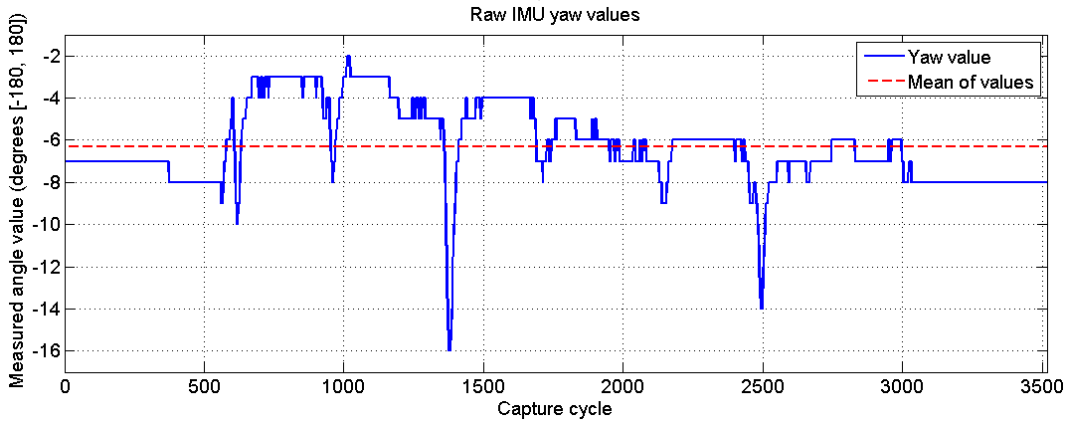
Thus, the current heading sensor software takes advantage of the stability of the IMU yaw and uses this value as orientation reference. However, the compass is still an important piece for the system, as it must be used as reference for calibrating the yaw sensor, so that the values measured are the angle relative to our north, which is one of the field goals.

The procedure takes into consideration that, during a match, the robots are turned on and initialized in a preferential area of the field, which is near the center line and near the side line of the team base stations. Given this procedure, it is important that the compass raw value is stable and measurable in that area of the field so that, when the IMU board is turned on, the reference value of the compass is a good value.

In an attempt to reduce the drift effect of the IMU yaw angle measurement, a recalibration of the imu during robot operation is possible, under specific conditions. Each agent cycle, the robot's orientation estimated by the localization algorithm is stored on the RTDB and sent to



a)



b)

Figure 4.34: Robot orientation measured by both **a)** the compass and **b)** the IMU yaw during the tour represented in figure 4.33. The dashed red line is the mean value of the data.

the IMU micro controller. When a discrepancy occurs (i.e. when the angle difference between localization and heading sensor does not fall under the acceptance margin of figure 4.31), a preset value is sent to signal the situation.

The IMU micro controller software is implemented in such a way that, when the orientation of the robot estimated by the localization algorithm is coherent with the compass values for a long time (currently 10 seconds is the minimum matching time), then it readjusts its calibration based on the current compass value.

4.5 Obstacle detection and identification

In the CAMBADA team, visual information about the obstacles is gathered by the omni directional camera, which is the only sensor that all robots have to gather information about their surroundings. According to RoboCup rules, the robots are mainly black. Since in a

game robots play autonomously, all obstacles in the field are the robots themselves (and occasionally the referee, who is recommended to have black/dark pants). The CAMBADA vision algorithm takes advantage of this fact and detects the obstacles by evaluating blobs of black color inside the field of play [142], by detecting transitions from green to black over the *scanlines*.

After identifying the points on the image where such transitions occur, the way to deal with them has been changed over the course of this PhD work.

4.5.1 Previous definition

In a first approach on this PhD work, the black points on the image were clustered into blobs and an analysis similar to the ball analysis was performed, to estimate the obstacle blobs limits and lengths. To define the limits of the clustering, a pair of thresholds were used to decide when the space between black points was enough to consider another obstacle or to consider the point as part of a given existing cluster [143, 144].

Since each black point is the point on the image where the first transition from green to black occurred over a *scanline* and since the used *scanlines* are the main radial ones, the points are analyzed angularly around the image center.

The pair of thresholds that were verified were the number of *scanlines* that separated two consecutive black points and the length from the center of each of the consecutive black points. At this stage of the vision process, all the information and thresholds are defined in pixels and pixel coordinates.

The separation offsets of a blob close to the robot were bigger than the ones at a high distance, to maintain coherent precision, estimated by a function with a curve similar to the distance map relation of figure 4.2. The angular separation offset was considered for situations where robots were side-by-side, at the same distance, but there was no visual contact between each blob; the length separation offset was checked for situations where, on sequential *scanlines*, there were blobs with visual contact but the robots were actually at different distances, meaning the considered point was at a different length on the *scanline*. Figure 4.35 presents an example of several situations that we wish to obtain, using only a very small number of *scanlines* and examples of 1 and 2 *scanlines* separation thresholds which is obviously just a crudely approximated example for illustration of the problem.

The results of this approach were very dependent on a polynomial function for defining the pixel variation on the image. In practice, it was verified that at higher distances (corresponding in the image to some tens of pixels from the mirror border), as the distance difference between pixels tended to very high values, the offset definition in pixels was not accurate.

Since the offsets cannot be fixed and need to depend on the inter-pixel distance, when maintaining an acceptably accurate threshold relation for shorter distances, the offset at longer distances would not be applicable. This caused the merging of blobs which were clearly part

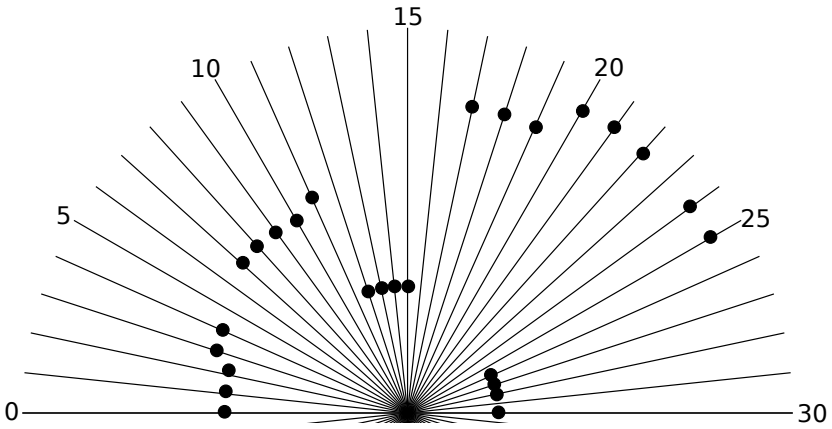


Figure 4.35: Illustration of black points detected on *scanlines*. Some limit situations for clustering the points on the image are presented:
- Situation 1: The distance from center of the point on *scanline* 2 and *scanline* 3 should be enough to consider that the later is a point on a different obstacle.
- Situation 2: Contrary to the previous situation and given the general distance of the points to the center, the *scanline* 11 should be considered as part of the same obstacle of *scanline* 10.
- Situation 3: From *scanline* 19 to 20, although the distance from the center is high, the points are far enough to be separated in different obstacles.
- Situation 4: Although there is only one *scanline* separating 22 from 24, they are farther from the center and thus the radial distance should be enough for separating them.
- Situation 5: Contrary to the previous situation, the radial distance from *scanline* 28 to 30 should be small enough to consider both points on the same obstacle.

of different objects (figure 4.36) or in some situations, even well segmented obstacles would be discarded at longer distances, when segmented by few *scanlines*, because the separation threshold was too high.

Moreover, in the vision process, there is no information about the position of the robot and thus, from the vision process point of view, any black blob should be considered. However, we do not need to consider any obstacle that is outside the field, as it will not interfere with the game.

4.5.2 New definition

To avoid the situation previously described, a new way of determining the obstacles center and limits was implemented [145]. Thanks to the known relation between pixel and distance, the relative coordinates of each point is known and can be made available for the integration process. Changes to the vision algorithm were made so that the relative position of each detected black point (the first black point on each *scanline*) is passed over to the integration process, instead of a visual estimation of the obstacle center and limits. The passed points are angularly ordered according to the *scanlines* angular definition.

The integration process takes the points in metric coordinates and builds the obstacles.

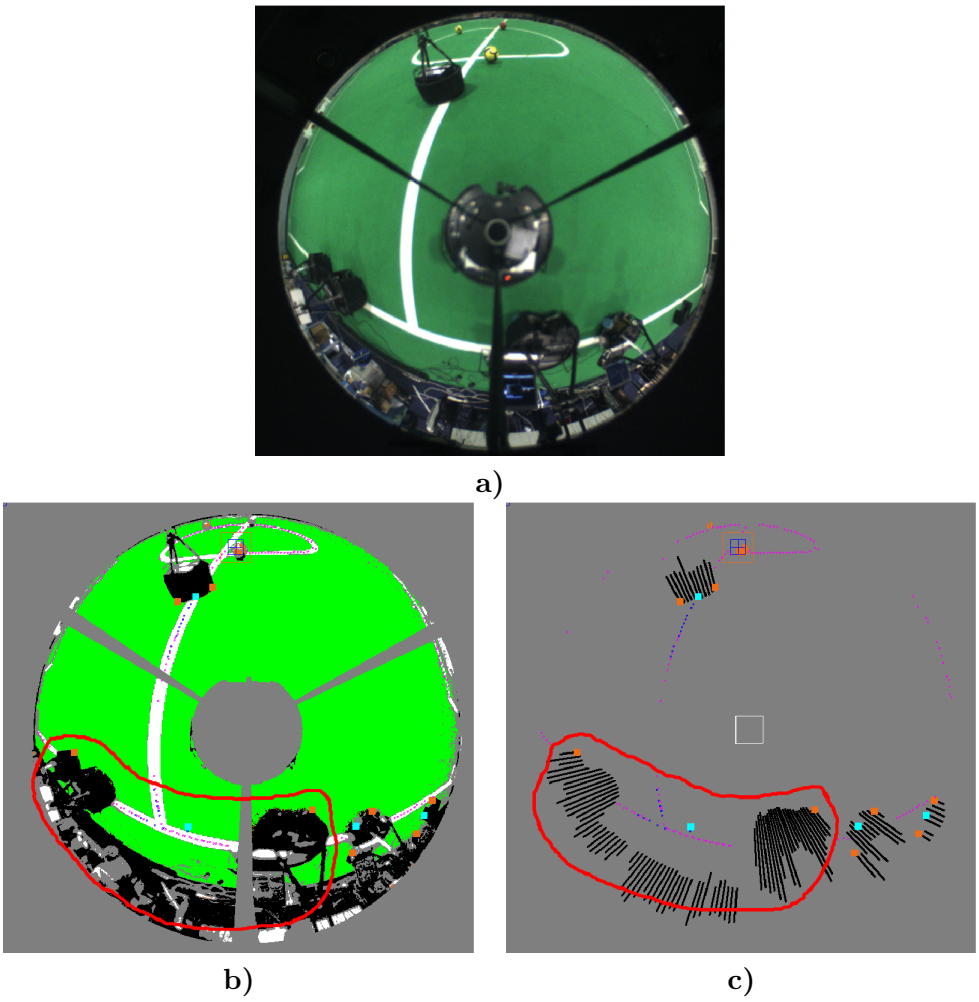


Figure 4.36: Capture of a camera frame with an incorrect visual obstacle detection as described. Top **a)** Original frame. **b)** Color segmented image. **c)** Blob image obtained from segmentation. Obstacles are represented by the estimated visual limits (brown squares) and by its center (cyan square). The bottom left part of the image is merged within a single obstacle (surrounded by the red line which was drawn over the images to highlight the detail).

This is made by an iterative process which starts by acquiring the first black point of the list (corresponding to the radial *scanlines* zero angle) and define it as the limits of an obstacle. Iteratively, each black point is tested to be within a given neighborhood of the previous one. If it is within a given threshold metric distance, it is added to the currently obstacle being built and is assumed as the left limit. The neighborhood value is redefined by a mean of distances between all the points of the current obstacle under construction. When a black point appears that does not belong to the neighborhood of the previous, the current obstacle is put on the obstacle list and a new obstacle is considered, starting with the current black point. After iterating all the points, the first and last obstacles limits are tested. If they are closer than the initial neighborhood value, they are considered as part of the same obstacle,

which was divided by the start *scanline* of the search algorithm and thus they are merged (see algorithm 3).

Algorithm 3 Algorithm for building obstacles from the visual points.

Input: points → list of black points
Output: obstacleList → list of obstacle objects

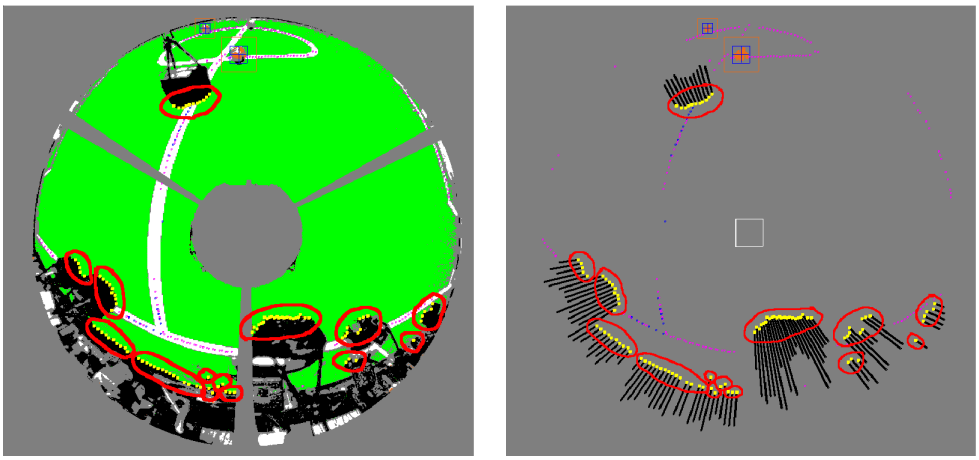
```
1: numberOfCreatedObstacles = 0
2: obstacleList.clear()
3: temporaryObstacle.leftLimit = points[0]
4: temporaryObstacle.rightLimit = points[0]
5: nPointsOnCurrent = 1
6: threshold = defaultMaxSeparationOffset
7: for p = 1 to totalNumberOfPoints.size() do
8:    if distance(points[p], points[p-1]) < threshold then
9:       temporaryObstacle.leftLimit = points[p]
10:      nPointsOnCurrent ++
11:      threshold = averageCurrentPoints(points[p], nPointsOnCurrent)
12:   else
13:      obstacleList.add(temporaryObstacles)
14:      temporaryObstacle.leftLimit = points[p]
15:      temporaryObstacle.rightLimit = points[p]
16:      nPointsOnCurrent = 1
17:      numberOfCreatedObstacles ++
18:   end if
19: end for
20: last=numberOfCreatedObstacles
21: if distance(obstacle[0].rightPoint, obstacle[last].leftPoint) < threshold then
22:    merge(obstacle[0], obstacle[last])
23: end if
```

This algorithm allows that obstacles at higher distances are evaluated in the same way as obstacles near the robot, without the need to estimate any additional relation function as it was the case of the previous definition, since the general distance map used by the vision is the one used to project, at ground plane, each one of the detected black points.

Additionally, since at the integration level the robot already knows its location, we can easily eliminate the black field borders or any other obstacle on the field vicinity by ignoring the obstacle points that are outside the field by more than a given distance threshold. This field context information allows us to, in algorithm 3, consider that a point too far from the field border lines is not in the neighborhood of the currently obstacle being built.

In the same situation depicted in figure 4.36, this new approach would visually provide all the points represented in figure 4.37a),b) by the yellow squares and the result of the algorithm on the integration process would consider different obstacles, surrounded in the images by red lines. This separation of obstacles is more accurate than the previous pixel separation, as the integration process knows the context of the black points and the individual Cartesian

coordinates of each one. In our case, the Cartesian distances are easier and more accurate to evaluate than pixel distances.



a) **b)**

Figure 4.37: Capture of a camera frame with the new visual information provided by the vision process. Left **a)**: Color segmented image; Right **b)**: Blob image obtained from segmentation. All the detected valid black points are marked by yellow squares. All these points are provided to the integration which considers several obstacles (the red lines surround the obstacles created based on the described algorithm.

Having a precision as high as possible on the obstacle positions is very important for the next step of obstacle treatment, which is their selection and classification as team mates or opponents.

With the objective of refining the information of the obstacles, and have more meaningful and human readable information, the obstacles are selected and a matching is attempted, in order to try to identify them as team mates or opponents [122, 88].

Due to the weak precision at long distances, a first selection of the obstacles is made by selecting only the obstacles closer than a given distance as available for identification (currently 6 meters). Also, obstacles that are smaller than 10 centimeters wide or outside the field of play margin are ignored. This is done because the MSL robots are rather big, and in game situations small obstacles are not present inside the field. Also, it would be pointless to pay attention to obstacles that are outside the field of play, since the surrounding environment is completely ignorable for the game development.

To be able to distinguish obstacles, identifying which of them are team mates and which are opponent robots, a fusion between the own visual information of the obstacles and the shared team mates positions is made. By creating a circle around the team mate positions with the robot radius (considered 22 cm plus an error margin defined by the visual noise relation, equation 4.3), a matching of the estimated center of visible obstacle area is made (figure 4.38), and the obstacle is identified as the corresponding team mate in case of a positive

matching (figures 4.39c) and 4.40c)). This matching consists on the existence of interception points between the team mate circle and the obstacle circle or if the obstacle center is inside the team mate circle (the obstacle circle can be smaller, and thus no interception points would exist).

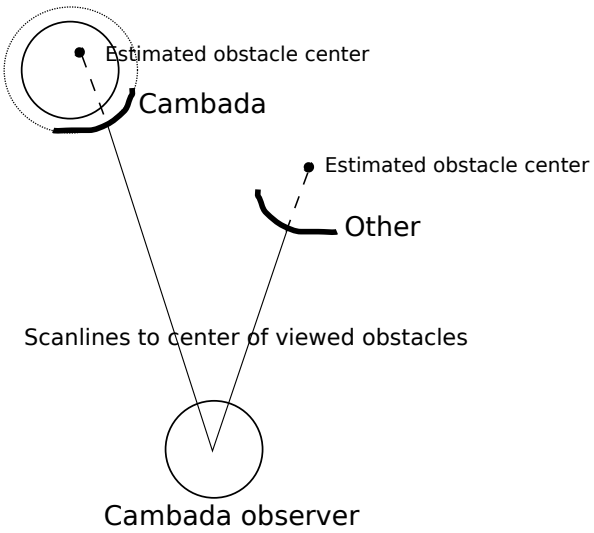


Figure 4.38: When a CAMBADA robot is on, the estimated centers of the detected obstacles are compared with the known position of the team mates and tested; the left obstacle is within the CAMBADA acceptance radius, the right one is not.

Since the detected obstacles can be large blobs, the above described identification algorithm cannot be applied directly to the visually detected obstacles. If the detected obstacle fulfills the minimum size requisites already described, it is selected as candidate for being a robot obstacle. Its size is evaluated and classified as robot if it does not exceed the maximum size allowed for MSL robots (figures 4.39a) and 4.39b)).

If the obstacle exceeds the maximum size of an MSL robot, a division of the obstacle is made, by analyzing its total size which is used to estimate how many robots are in that obstacle. This may be a common situation, robots clashing together and thus creating a compact black blob, originating a big obstacle if they are sufficiently lined up (figures 4.40a) and 4.40b)).

4.5.3 Experimental results

To assert the effectiveness and precision of the proposed algorithms, a series of tests were performed. The laboratory used for the tests receives natural light which can affect the vision processing algorithms. The presented results are not treated in any way to diminish the effects of natural light, as we are interested in understanding if the algorithms can cope with those conditions which can be found in real situations.

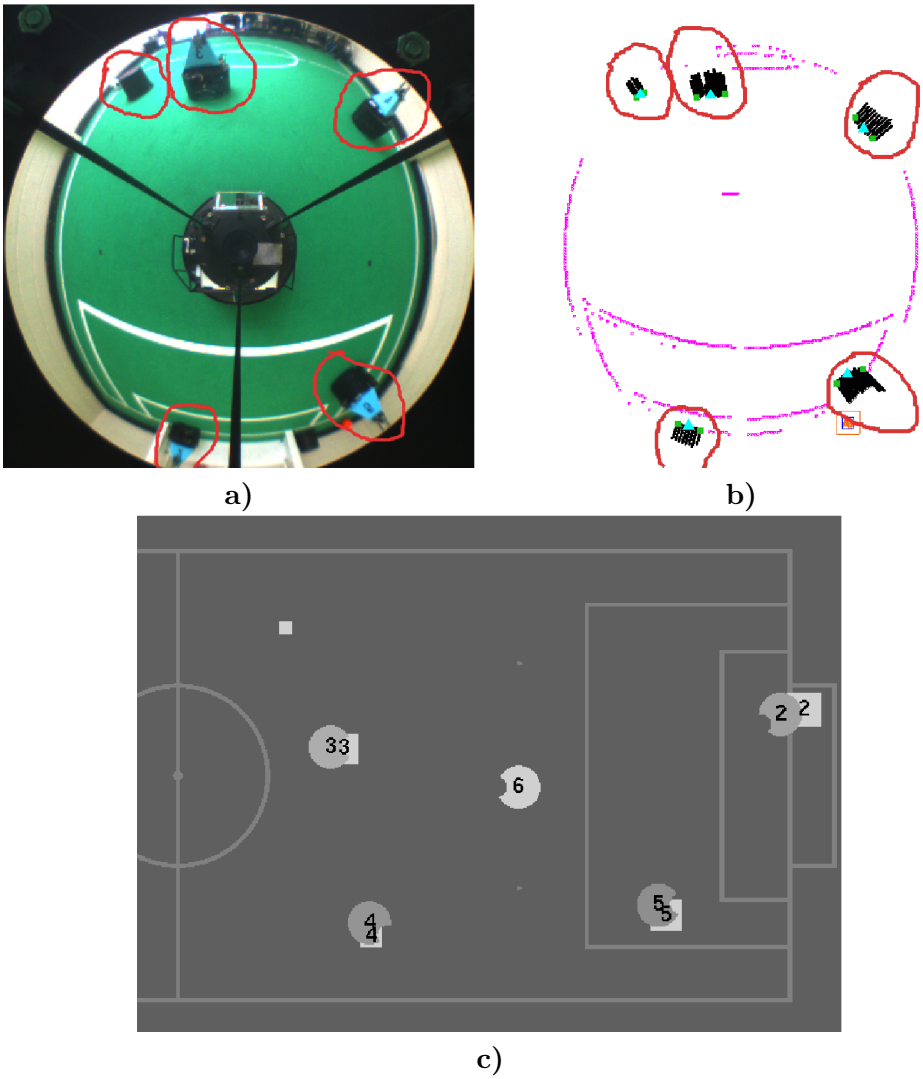


Figure 4.39: Illustration of single obstacles identification. In **a)** image acquired from the robot camera (obstacles for identification are marked); In **b)** the same image after processing; In **c)** image of the control station. Each robot represents itself and robot 6 (the lighter gray) draws all the 5 obstacles evaluated (squares with the same gray scale as itself). All team mates were correctly identified (marked by its corresponding number over the obstacle square) and the opponent is also represented with no number.

In the first test situation, a robot was positioned on the field at a stationary position. It was running the normal processes and thus localizing by itself on the field, while sharing its information as it would in a game situation. The position estimated by the localization algorithms was stable at $(-0.05, 1.88)$ and that position was shared with any other team mate. This robot will be referred to as *pivot*. Another robot was moving on a rectangular path around the pivot, and a capture of its data was done. This robot will be referred to as *observer*. This scenario is intended to give some insight about the performance of the

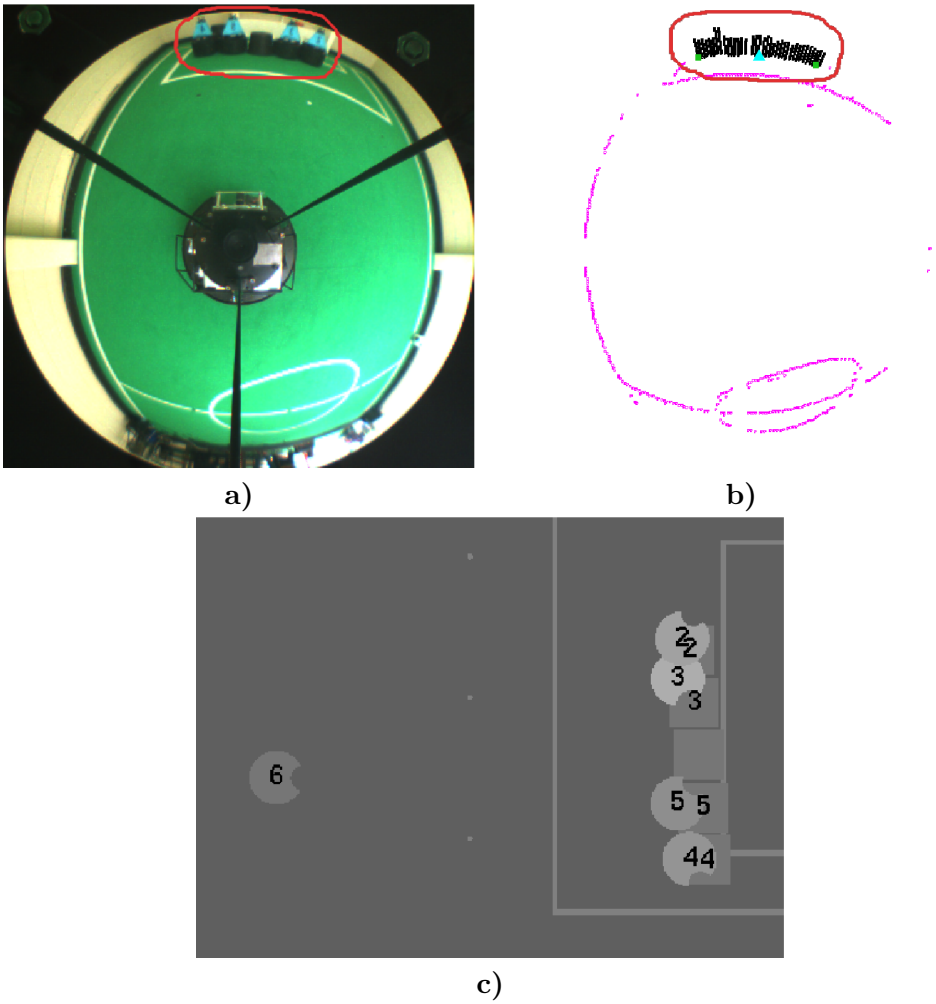


Figure 4.40: Illustration of multiple obstacles identification. In **a)**: image acquired from the robot camera (obstacle for identification marked); In **b)**: the same image after processing. Visually, the aligned robots are only one large obstacle; In **c)**: image of the control station. Each robot represents itself and robot 6 (the darker gray) draws all the 5 obstacles (squares with the same gray scale as itself). The visual obstacle was successfully separated into the several composing obstacles, and all of them were correctly identified as the correspondent team mate (marked by its corresponding number over the obstacle square) and the opponent is also represented with no number.

identification when the team mates are static or nearly static, as is the case of set plays during the games in which it is important to analyze passing lines. Figure 4.41 is a graphic representation of the acquired data, with the pivot represented in black in the position that it estimates by itself. The blue dots are the positions over the path taken by the observer, estimated by its own localization, which covers the rectangular path for 3 times. In each cycle, the center of the obstacle perceived by the observer is represented by a red 'x'.

It is visible that, as expected, the obstacle position perceived by the observer is not exactly

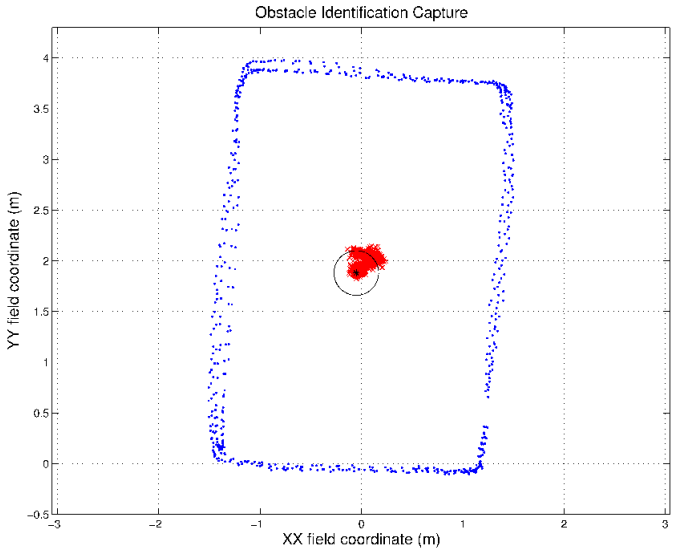


Figure 4.41: Representation of a capture of the obstacle identification algorithm results. The path taken by the observer is represented by blue dots in the rectangular path taken. Near the center, the pivot shared position is represented by the black star and its limits by the black circle. The center of the perceived obstacle in each cycle is represented by a red cross. Given the hundreds of measurements taken, they overlap and form the blob of red around the obstacle center.

the pivot position. The capture in question is composed of 677 cycles. The identification of the obstacle as the correspondent team mate failed to succeed in only one cycle, which corresponds to a 99.85% success rate.

Considering that the pivot has 22 cm radius (although it is slightly bigger), the mean of the centers of the perceived obstacle is within the real area occupied by the pivot, at nearly 16 cm with a standard deviation of 10 cm (Table 4.7).

Perceived obstacle		
	X	**Y**
Mean	0.05	2.01
Std	0.08	0.07

$$|\vec{Real} - \vec{Perceived}| = 0.16$$
$$|Std| = 0.10$$

Table 4.7: Table with the mean and standard deviation of the capture perceived obstacle position

One must keep in mind that, in the experimental setup, the observer CAMBADA robot was navigating around autonomously. It estimates its pose through the localization algorithm

and estimates the perceived obstacles based on its perceived pose. This means that the noise present on the presented scenario is not exclusively associated with visual perception uncertainty of the black points but also has a factor associated with the uncertainty of the localization algorithm. Unfortunately, given the absence of an external monitoring system capable of providing a global position of the elements on the field, we have no means to discriminate between the localization noise and the obstacle perception noise.

On the other hand, we do not consider this a critical problem, since we have a global idea of the noise associated, which is a good approximation of a real game situation, because the localization error will always be there and always affect every aspect of the perception.

Another test scenario was considered for evaluation of the algorithm performance for moving obstacles. Several captures were performed to evaluate the performance of the algorithm when identifying a moving team mate. This set of six captures consisted on a robot observing a team mate moving around and registering the data about the obstacles. The path taken by the moving team mate is represented in figure 4.42.

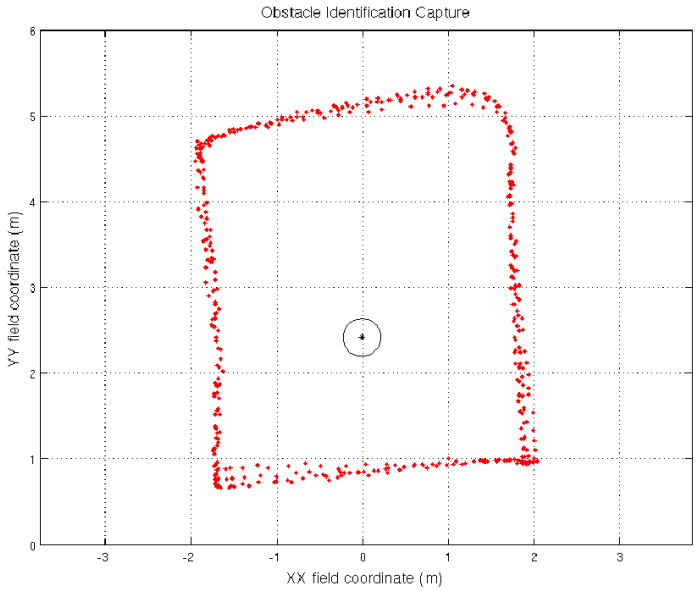


Figure 4.42: Representation of the path taken by the team mate to identify (the red dots represent each communicated position). The observer position is represented by the isolated black circle with the star in the middle.

The captures were performed throughout the day, with different lighting conditions but with the same robot calibration, without any automatic camera parameter adjustment software. Table 4.8 summarizes this set of captures, which revealed a total mean identification ratio of approximately 71%.

During the execution of the captures, and in a general way, the number of failed identi-

	Total cycles	Successes	%
Capture 1	1798	1319	73
Capture 2	1065	748	70
Capture 3	1528	1332	87
Capture 4	1162	769	66
Capture 5	1935	1278	66
Capture 6	2152	1411	66

Table 4.8: Table with the individual ratio of successful identification of the moving team mate for the several captures performed.

fications was greater when the moving robot was farther from the observer, as expected due to the noisy nature of the measurements. Also expected due to the noise was the existence of some outliers to this tendency. Figure 4.43 shows the summary of the six captures performed. The distance from the observer to the moving team mate to be identified was separated into five categories: <=2 m,]2, 3] m,]3, 4] m,]4, 5] m and >5 m and the percentage of successful identifications for each category calculated.

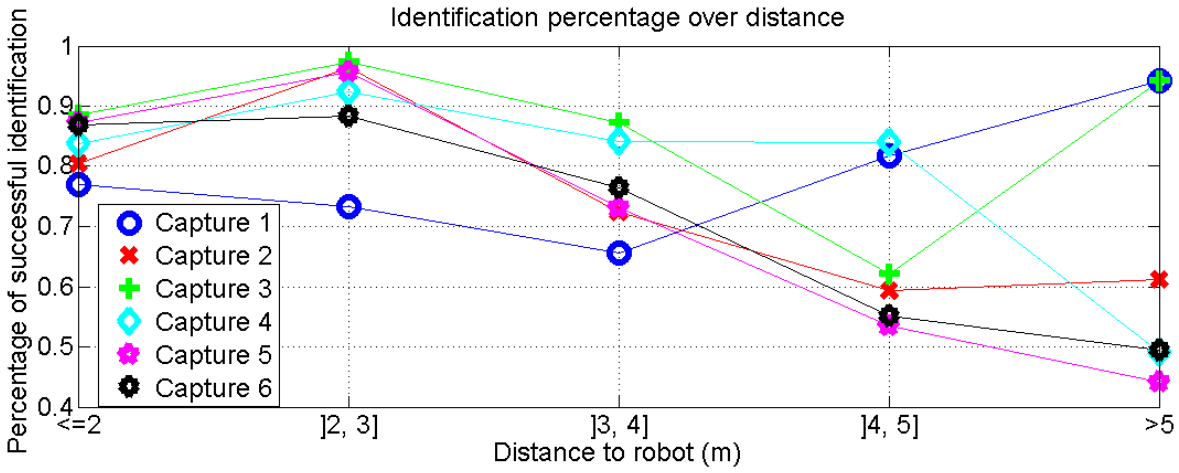


Figure 4.43: Summary of success rate for the identification algorithm for each of the six captures presented.

The results obtained by the team mate identification algorithm have a good detection rate in general and are fit for use in the team architecture. One should also keep in mind that the vision process usually runs with automatic camera parameter adjustment software and thus the provided data is usually more stable than the one used for testing the algorithm, which were a worst case scenario.

Currently, the identification of team mates plays a critical role in the coordination of passes among the robots, since the knowledge that a given seen obstacle is actually a team mate allows the evaluation of the passing conditions to be different when analyzing a team

mate. Since a team mate will not try to intercept the ball during a pass of its our own team, the pass can be made with a tighter corridor meaning the ball can pass closer to the team mate than it would be required for an opponent. This may even allow the creation of strategies to use team mates as cover for the ball path when performing a pass to another team mate.

4.5.4 Tracking

After identifying which obstacles are team mates and which ones are not, the next step that we aim for is to track each of the visually detected opponent, so that it can be identified unequivocally over time. To achieve this in the CAMBADA team, a multiple hypothesis tracking approach is used, based on the use of a Kalman filter.

The idea is to keep a list of tracks, each element corresponding to one of the detected opponents. Each track's state keeps the Kalman filter state, which includes position and velocity, the identification of the given obstacle, a counting variable to register for how long the track has not been fed a new measurement, a lifetime counter and all the internal Kalman values necessary for the filter.

The algorithm starts by updating the last state of each track, by executing the first phase of the Kalman filter, the prediction phase. The currently used state transition model is based on a uniform linear movement equation. In this phase, any track that has been evolving based only on prediction for more than 100 ms is erased, because tracks that do not have visual support would introduce false positives on the world model (algorithm 4).

The second step of the algorithm is to evaluate and match each of the visually detected obstacles with the existing tracks. For each of the existing tracks, the detected obstacle is compared with the tracked position, to verify if they are in proximity. When that is the case, the track filter is updated according to the measured position and associated noise and the detected obstacle is marked as used.

When a detected obstacle does not match any of the existing tracks or if there are no tracks available, the obstacle initiates a new track, which is initialized both in terms of the obstacle position and obstacle identification. The management of the identifiers of each obstacle track is kept as a global feature, so that each identifier is unique in each moment.

The identifier feature of the obstacle tracking is currently well explored, mainly on opponent set piece situations, where an opponent team robot has to pass the ball to another team robot. One strategy adopted by our team is to cover potential receiver opponents, so that they are no longer a useful option and, if the pass is made to them anyway, we already have a robot ready to intercept the ball.

During opponent set pieces, our robots may choose to cover an opponent robot. Since it is usually not standing still, as it tries to find an empty space by running away from our robot, our robot needs to follow the opponent wherever it goes. The identification of the opponent

Algorithm 4 Algorithm for using a Kalman filter multiple hypothesis tracking for opponent tracking.

Input: opponentList → list of visually detected opponent obstacles
tracksList → list of existing opponent tracks
Output: tracksList → an updated list of opponent tracks

```
1: for currentTrack = tracksList.first() to tracksList.last() do
2:   if currentTrack.onlyPredictionCount > predictionLimit then
3:     tracksList.erase(currentTrack)
4:   else
5:     currentTrack.predictionPhase(currentTime)
6:   end if
7: end for
8: for o = 0 to opponentList.size() do
9:   observationUsed = false
10:  if tracksList.size() > 0 then
11:    for currentTrack = tracksList.first() to tracksList.last() do
12:      trackPosition = currentTrack.getPosition()
13:      currOppPosition = opponentList.getPosition(o)
14:      if distance(trackPosition, currOppPosition) < proximityLimit then
15:        currentTrack.setNoise( distanceTo( currOppPosition ) )
16:        currentTrack.updatePhase(currOppPosition)
17:        observationUsed = true
18:      end if
19:    end for
20:    if not(observationUsed) then
21:      newTrack = createNewTrack()
22:      newTrack.predictionPhase(currentTime)
23:      newTrack.updatePhase(currOppPosition)
24:      newTrack.estimateID()
25:      tracksList.add(newTrack)
26:    end if
27:  else
28:    newTrack = createNewTrack()
29:    newTrack.predictionPhase(currentTime)
30:    newTrack.updatePhase(currOppPosition)
31:    newTrack.estimateID()
32:    tracksList.add(newTrack)
33:  end if
34: end for
```

is crucial for this task, to guarantee that our robot does not change its opponent target.

In figure 4.44, a game situation is presented, where the CAMBADA robot number 3, on an opponent set piece situation, is seeing a set of five obstacles. One of them is identified as CAMBADA robot number 4, while the others are assigned a unique identifier.

In the depicted situation CAMBADA robot number 3 decided, in the beginning of the

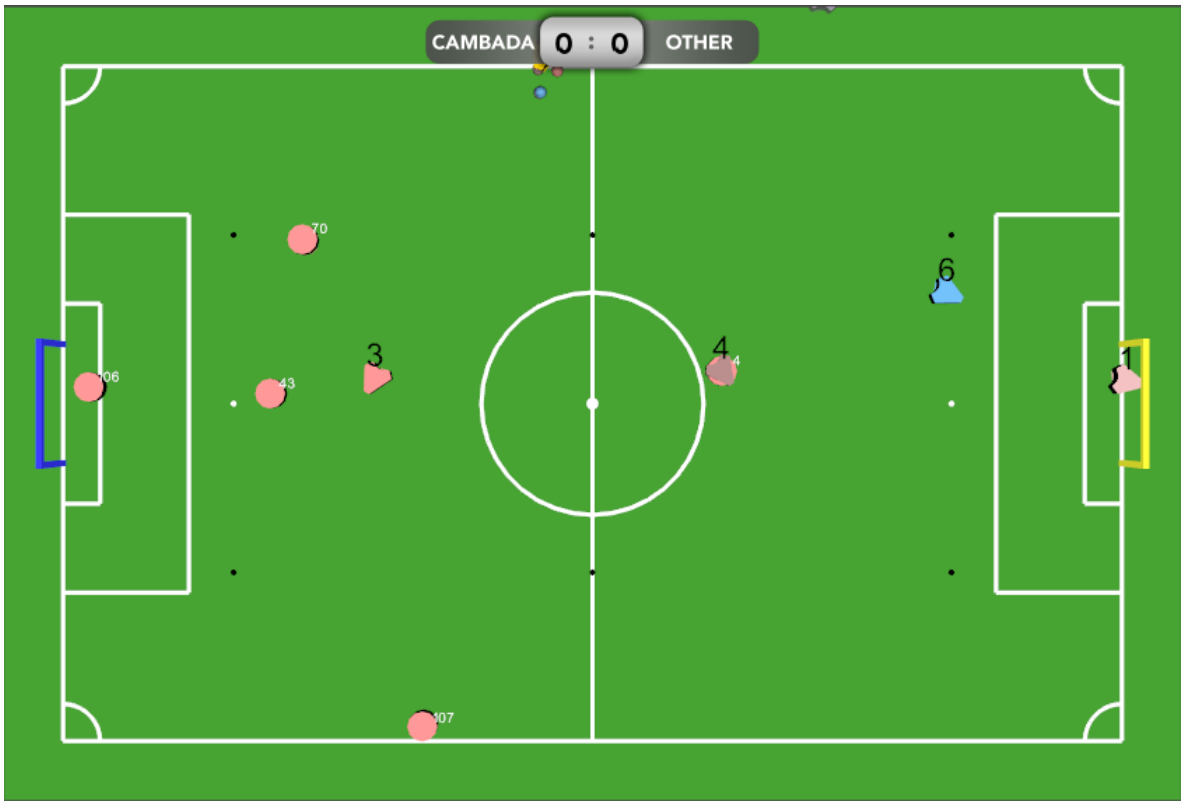


Figure 4.44: Result of obstacle tracking during a game on the RoboCup2014. In this opponent setpiece situation, CAMBADA robot 3 was seeing five obstacles and identified one of them as team mate number 4. The opponent identified as 43 was in this case chosen to cover and was followed through the remaining of the play.

play, to cover the opponent that is identified as 43. Throughout the play, it followed the opponent to cover its direct path to the ball, in order to hinder the possibility of that robot to receive a pass.

4.6 Obstacle avoidance

The perception of obstacles in a soccer robot is an important feature to allow the reasoning layer to execute its tasks effectively. One of the issues addressed during this work was a way to perceive the obstacles and react to their presence by avoiding them effectively. This reactive layer is a last resource estimation of what would be the best direction to take according to the current near field occupancy. We named this approach as "sonar", given the similarity of the representation of the obstacles in areas angularly distributes around the robot.

The base idea behind this reactive layer is to angularly search for the least angular deviation relative to a straight line to the target, taking into account the configurable constraints imposed by the obstacles around the robot. The angular step is configured *a-priori* and each

"direction" is identified by an ID, zero being always the ID of the direction straight to the target.

4.6.1 Previous definition

The reactive avoidance algorithm previously implemented allowed the robots to perform avoidance tasks reasonably. However, over the time and with the natural evolution of the league, some issues became more evident and eventually limiting.

The analysis of the surrounding space was performed in N simple slices originating from the robot center, as represented in figure 4.45. The main slice (zero) is the slice that point directly to the desired target position and a maximum distance at which avoid actions are taken is parameterizable, being a slice considered as "free" if obstacles are farther than the maximum distance.

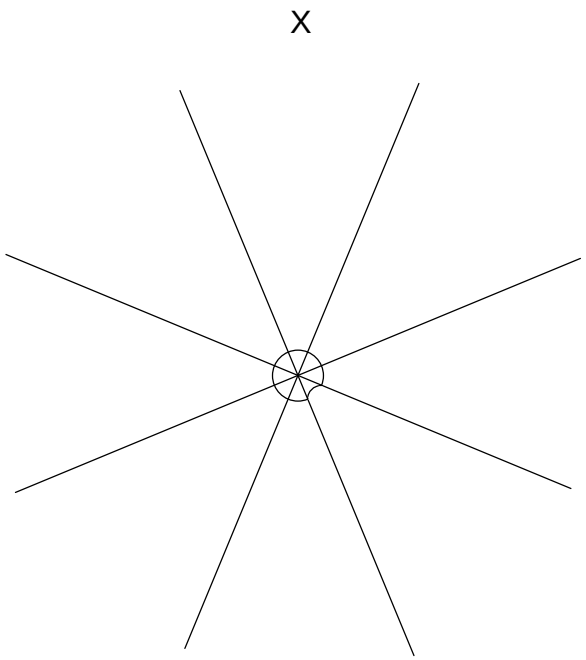


Figure 4.45: Representation of the old obstacle avoidance model.

Given the list of obstacles, each of them is tested to be within any of the slices and through this iterative process, each slice is "marked" with the distance of the closest obstacle. The least deviation slice is then searched (being the least deviation slice the first free slice encountered on the "search" cycle) and the slice direction is the new direction that the robot should take. The search for the deviation slice is performed in alternate directions until the first "free" slice is found (algorithm 5).

This approach had a reasonable performance when the robot would have an obstacle to avoid that made it deviate a lot from the original trajectory, because that way, when the

Algorithm 5 Choice of the free direction based on the previously defined slice model. The algorithm finishes as soon as it finds the first free direction.

Input: obstacleList
Output: deviationAngle → angle of the chosen free direction

```
1: nSlices = readFromConfig()
2: sliceMaxDist = readFromConfig()
3: deviationAngle = 0
4: for currSlice = 0 to nSlices do
5:    minDistCurrSlice = MAXDISTANCE
6:    for o = 0 to obstacleList.size() do
7:       if  currSlice.isInside(obstacleList[o]) and dist(obstacleList[o]) < minDistCurrSlice
   then
8:          minDistCurrSlice = dist(obstacleList[o])
9:       end if
10:   end for
11:   if minDistCurrSlice < sliceMaxDist then
12:      currSlice.setNotFree()
13:   end if
14: end for
15: for currSlice = 0 to nSlices/2 do
16:   if currSlice.isFree() then
17:      deviationAngle = currSlice.angle()
18:      FINISH
19:   end if
20:   if (nSlices - currSlice).isFree() then
21:      deviationAngle = (nSlices - currSlice).angle()
22:      FINISH
23:   end if
24: end for
```

obstacle was not occupying the main slice, it would have some space to return to the trajectory without hitting the obstacle. When it was not the case, however, the robot would easily hit the obstacle when it evaluated the main slice as free, since the obstacle on a neighboring slice could be too close to allow the body of the robot to pass.

Another issue that came up over the time was due to the use of simple slices. Since they keep opening, at higher distances, the distance necessary for the robot to consider the slice free was too large, which would cause the robot to go around situations where a quite large corridor was free.

On the other hand, the choice part of the algorithm would also easily create situations of indecisions due to the lack of historical context, making the robot choose to deviate left and right continuously when the obstacle would be on a slice limit situation.

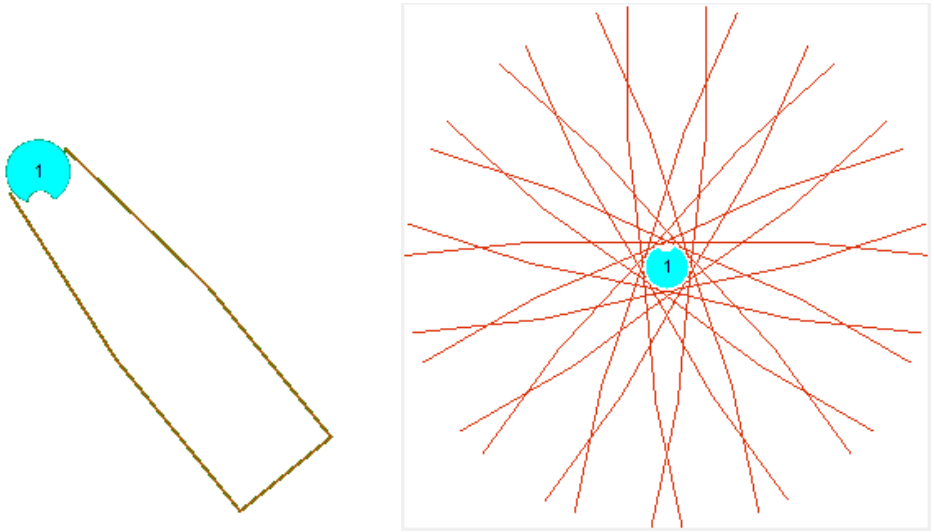
4.6.2 New approach

With the previously defined issues in mind, a new approach was designed and implemented to cope with them.

In this new approach, the first thing to abandon was the simple slice model. We designed the new model to be a section composed by two parts: a triangle not centered on the robot and a rectangle, as in a corridor (figure 4.46a). This means that each sonar direction starts by a triangle, opening from the robot's border until a defined width, and then keeps that width along the remaining distance, creating a corridor. This allows the sonar to cope with larger distances without requiring ever larger width of free space for the robot to pass. The triangle area that covers the robot area from its center to the direction opposed to the rectangular corridor is not considered, as that area is in "the back" of the section.

Furthermore, by starting the opening of the corridor on the robot's surface, we make sure that the occupied direction remains occupied until the robot body has passed the obstacle, instead of changing the obstacle from slice too soon as happened with the previous model. A sonar is defined by a given number of sections with the described characteristics (figure 4.46b), which should overlap to cover most of the area around the robot. The number of sections for the sonar can be seen as the resolution of the avoidance direction.

The main direction of the model (zero) is, as before, pointing directly to the target position.



a) **b)**

Figure 4.46: Representation of the new obstacle avoidance model, the sonar.

The sonar model is defined by a set of parameters that allow to configure each instance as required. The main configuration parameters are the following:

- **maxSonarOpening** defines the maximum width of a section, which is basically the

width of the straight corridor part of the section.

- **maxSonarDistance** is the maximum length of the sections of the sonar, defining the maximum distance at which avoid actions are taken.
- **thresholdDistance** defines the distance at which the section ceases to be triangular and becomes a straight corridor.
- **numberOfSections** is the total number of sections that compose the sonar model.
- **bodyOversize** is a factor applied to the robot radius to define the opening of the section at its source, which is the line that is perpendicular to the section direction and passes through the center of the robot.

With this design, we tackled the two first problems identified before, in terms of space requirements in both ends of the model. The next step was to create a decision model that would perform better than the previous one.

First of all, we made the last direction index available as a state of the sonar class. When there is no deviation action in course, the choice is similar to the previous model, by searching the sections on alternate sides of main target section and choosing the first free (algorithm 6).

In case there is a deviation action in course, there are two considered cases which use a different approach in an attempt to prioritize the search of a free section in the direction of the target while maintaining the general direction of the deviation to avoid indecisions.

When the last deviation action was still in the general direction of the target (meaning that the absolute value of the deviation angle relative to the target was not bigger than 90°) the search for the free section to use is done first up to twice the last deviation target. From there on and until we reach half circle (the section that is more opposite to the target) the search is performed alternating between those remaining sections and the first sections on the opposite side of the target. When the robot would start to deviate so much it would go basically opposite to the target, we find preferable to try to avoid the obstacle from the other side instead of inverting the movement. Finally, if there is still no free section, the remaining sections of the opposite side of the last deviation are tested, which would result in moving away from the target and in the opposite direction (algorithm 6).

When the last deviation was already moving away from the target (more than 90°), the first search range starts at 0 and goes up to the last deviation target. Afterwards, and since the robot is currently moving away from the target, the search is performed on the opposite side of the last deviation and up to half its value. Then the remaining sections between the last deviation and half circle are searched while alternating with the equivalent number of sections on the opposite side (from the equivalent to half last deviation, where the algorithm stopped before). Finally, the remaining sections on the opposite side of last deviation are tested (algorithm 6).

4.6.2.1 Sonar analysis

To test and validate the use of the new avoidance model, the sonar, a graphical tool was created that simulates the agents movements and possesses a graphical interface to define the configuration of a given agent so that different parameterizations can be conceptually tested prior to try them on the real robots. The configuration parameters include some physical characteristics of the robot, like its diameter, motor and transmission parameters and dynamic behavior limits, and the configuration of the sonar parameters (figure 4.47).

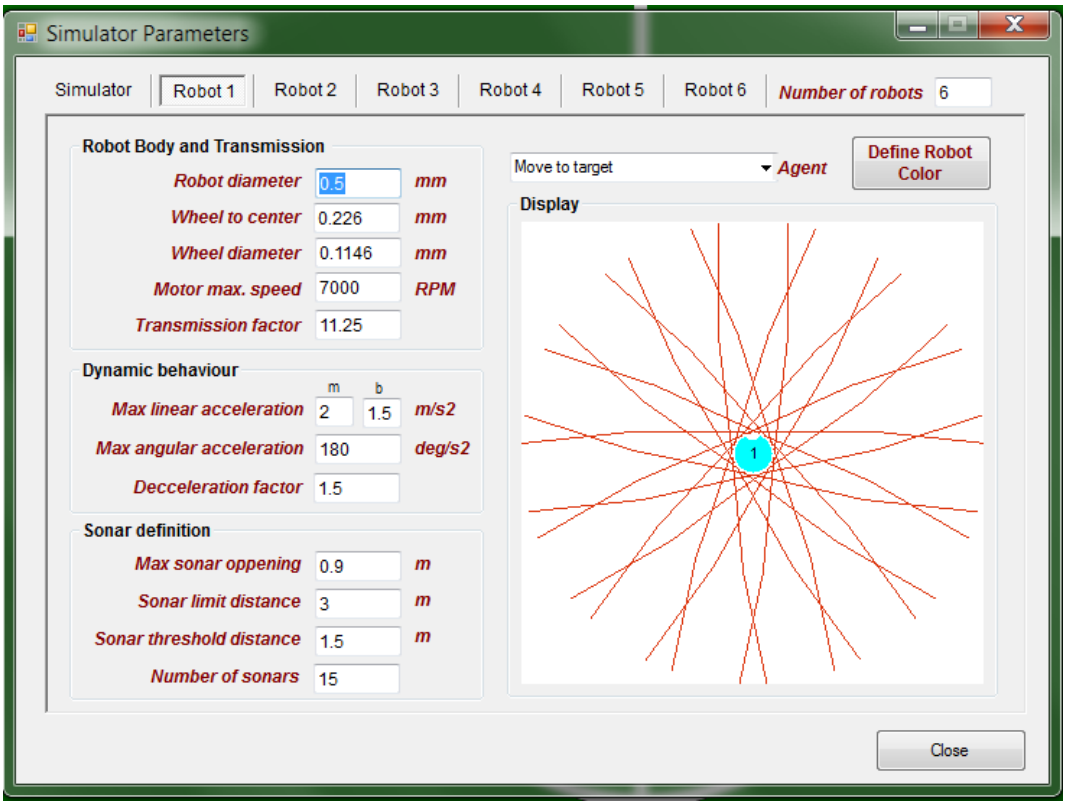


Figure 4.47: Configuration interface of the sonar simulation application.

Algorithm 6 Algorithm used for selection of the direction the robot should take to avoid obstacles. The algorithm finishes as soon as it finds the first free direction.

Input: obstacleList
Output: deviationAngle → angle of the chosen free direction

```
1: lastIndex = getLastDirectionIndex()
2: nSlices = readFromConfig()
3: deviationAngle = 0
4: if lastIndex ~= 0 then
5:   if abs(angle(lastIndex)) < 90° then
6:     for currSlice = 0 to 2*lastIndex do
7:       if currSlice.isFree(obstacleList) then
8:         deviationAngle = currSlice.angle()
9:         FINISH
10:      end if
11:    end for
12:    beginIdx = 2*lastIndex
13:    nIterations = 1
14:    for currSlice = beginIdx to nSlices/2 do
15:      if currSlice.isFree(obstacleList) then
16:        deviationAngle = currSlice.angle()
17:        FINISH
18:      end if
19:      if sliceOnOppositeSide[nIterations].isFree(obstacleList) then
20:        deviationAngle = sliceOnOppositeSide[nIterations].angle()
21:        FINISH
22:      end if
23:      nIterations ++
24:    end for
25:    for currSlice = sliceOnOppositeSide[nIterations] to nSlices/2 do
26:      if currSlice.isFree(obstacleList) then
27:        deviationAngle = currSlice.angle()
28:        FINISH
29:      end if
30:    end for
31:  else
32:    for currSlice = 0 to lastIndex do
33:      if currSlice.isFree(obstacleList) then
34:        deviationAngle = currSlice.angle()
35:        FINISH
36:      end if
37:    end for
38:    for currSlice = 0 to sliceOnOppositeSide[lastIndex/2] do
39:      if currSlice.isFree(obstacleList) then
40:        deviationAngle = currSlice.angle()
41:        FINISH
42:      end if
43:    end for
```

```
44:       nIterations = 1
45:       for currSlice = lastIndex to nSlices/2 do
46:         if currSlice.isFree(obstacleList) then
47:           deviationAngle = currSlice.angle()
48:           FINISH
49:         end if
50:         if sliceOnOppositeSide[nIterations].isFree(obstacleList) then
51:           deviationAngle = sliceOnOppositeSide[nIterations].angle()
52:           FINISH
53:         end if
54:         nIterations ++
55:       end for
56:       for currSlice = sliceOnOppositeSide[nIterations] to nSlices/2 do
57:         if currSlice.isFree(obstacleList) then
58:           deviationAngle = currSlice.angle()
59:           FINISH
60:         end if
61:       end for
62:     end if
63: else
64:     for currSlice = 0 to nSlices/2 do
65:       if currSlice.isFree(obstacleList) then
66:         deviationAngle = currSlice.angle()
67:         FINISH
68:       end if
69:       if sliceOnOppositeSide[currSlice].isFree(obstacleList) then
70:         deviationAngle = sliceOnOppositeSide[currSlice].angle()
71:         FINISH
72:       end if
73:     end for
74: end if
```

With this tool, a series of tests were performed using several configurations to analyze how the configuration of the sonar affects the behavior of a robot. Three scenarios were run, each one with three different sonar configurations. The scenarios were:

- Scenario **one**: Robot moves to a point on the opposite side of the field with a static obstacle on its direct path (figure 4.48).

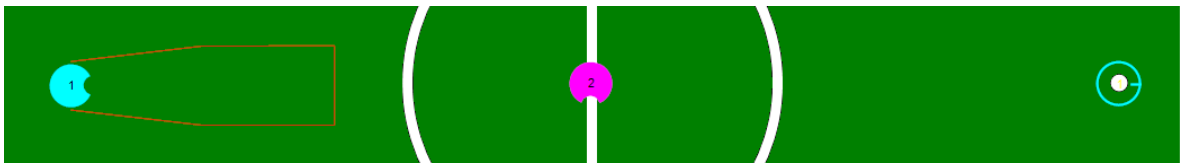


Figure 4.48: Avoid test scenario 1.

- Scenario **two**: Robot moves to a point on the opposite side of the field with two static obstacles forming a roughly 1 m wide corridor on its direct path (figure 4.49).

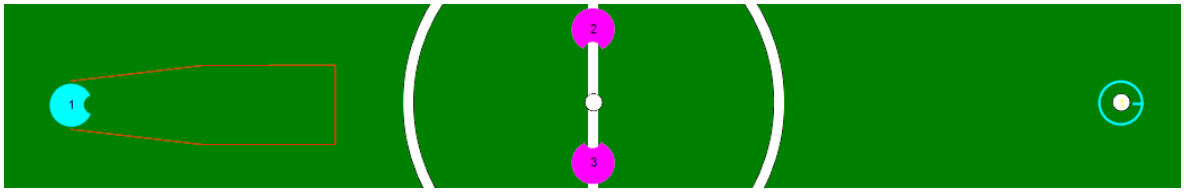


Figure 4.49: Avoid test scenario 2.

- Scenario **three**: Robot moves to a point on the opposite side of the field while an obstacle makes a symmetric movement in a straight line, without any attempt of avoidance (figure 4.50).

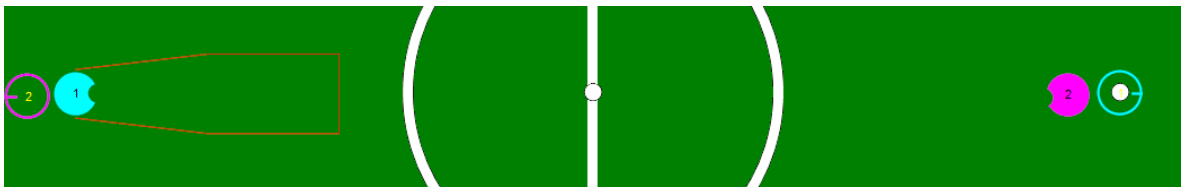


Figure 4.50: Avoid test scenario 3.

All the scenarios were tested with three different sonar configuration, varying two of the sonar properties: the number of sonar sections and the maximum opening of each section:

- Configuration **one** uses 15 sonar sections, 0.9 m maximum sonar opening.
- Configuration **two** uses 15 sonar sections, 1.5 m maximum sonar opening.

- Configuration **three** uses 30 sonar sections, 1.5 m maximum sonar opening.

The remaining properties were the same for all the runs:

- maxSonarDistance was set to 3 m, which means that the robot will try to avoid obstacles as soon as they are closer than 3 m within a given sonar section.
- thresholdDistance was set to 1.5 m, meaning the section becomes a corridor of the defined width at 1.5 m from the robot.
- bodyOversize was set to 1.1 meaning that the minimum distance taken into account for the robot to pass beside an obstacle is 10% more than its radius.

The moving robot was configured to move at a maximum speed of 3.0 m/s, as well as the moving obstacle from scenario 3. The distances involved guarantee that the robot is moving at maximum speed when the avoidance evaluation starts. The period of the simulation is 20 ms and the perception of the simulated robot includes Gaussian noise of 0.05 m in terms of translation of the perceived points and 3º in terms of rotation of perceived points.

Scenario one

Considering the first scenario, figure 4.51 displays the complete trajectory of the robot to avoid the obstacle. Figure 4.51a, b and c represent the captures with sonar configurations one, two and three, respectively. The robot outline is represented in the point on the robot trajectory where it was closer to the obstacle, to have an idea of the clearance margin during the run.

In table 4.9 and for scenario one, the total time of the robot movement and the minimum distance of the robot to the obstacle during the run are presented.

	Min distance (m)	**Time (seconds)**
Sonar config 1	0.36	4.60
Sonar config 2	0.54	4.64
Sonar config 3	0.42	4.62

Table 4.9: Minimum obstacle distance and time results for scenario one.

Considering the results of the several configurations and in scenarios of avoiding one static obstacle, there is a small advantage on using the first sonar configuration, which causes the robot to deviate less and thus have a marginally better time of execution.

Scenario two

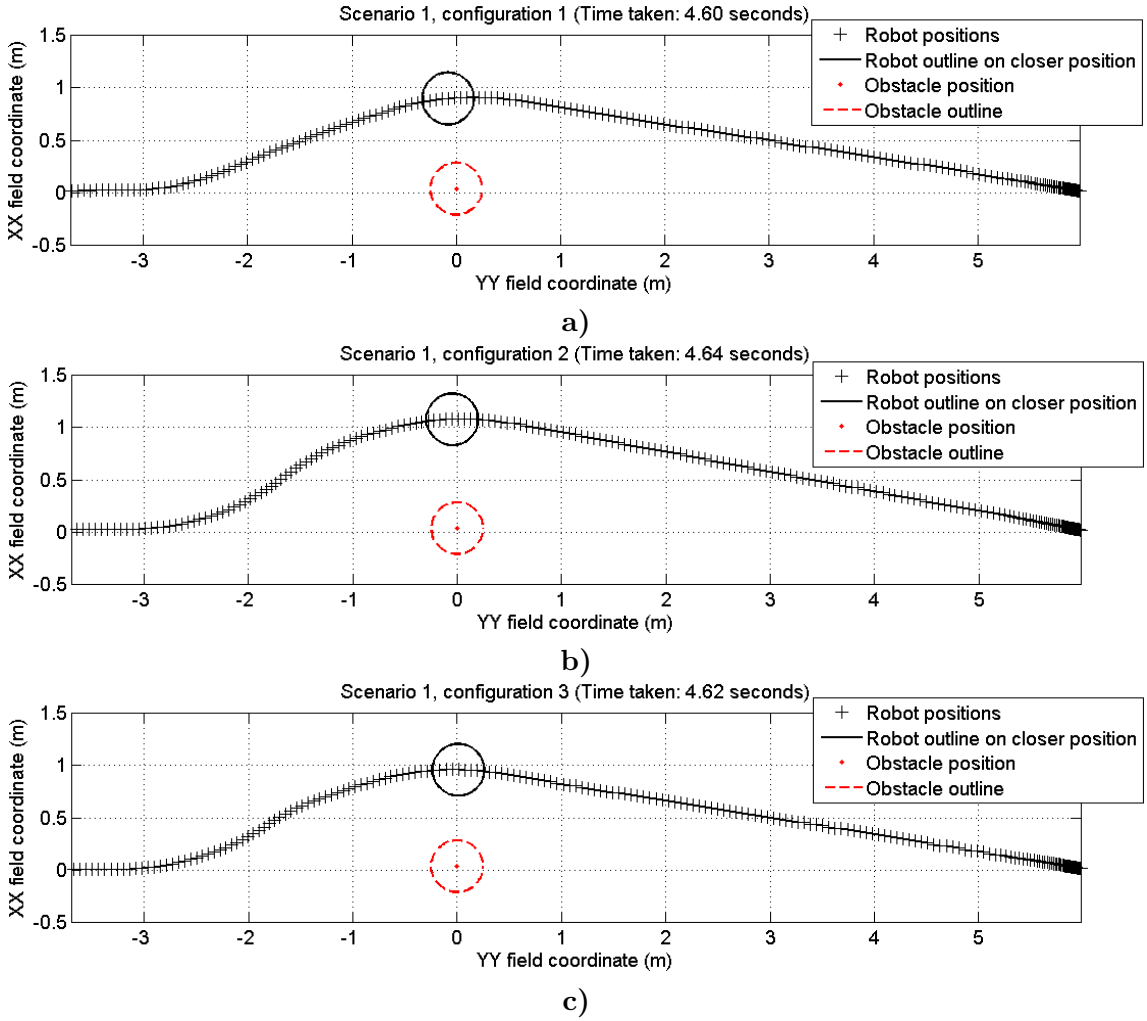
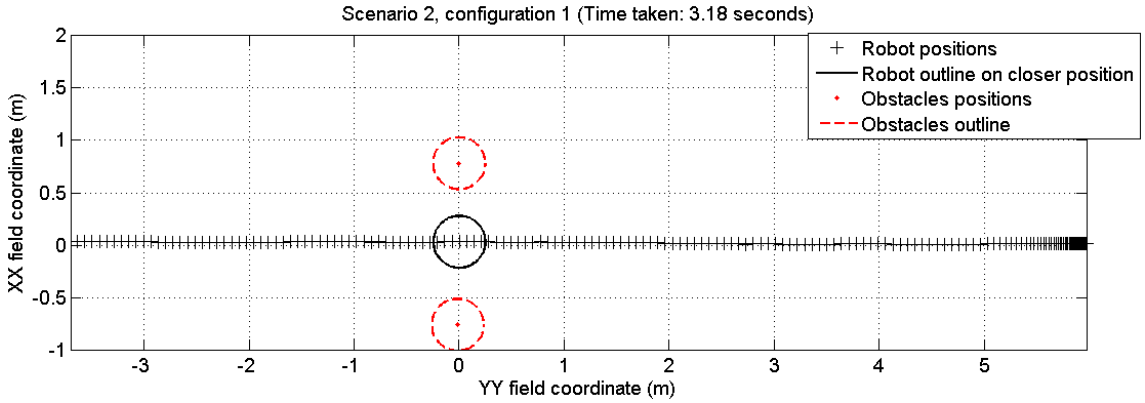


Figure 4.51: Visual representation of the robot path in avoidance scenario 1. Both the robot and avoided obstacle are represented by their outlines in the path position where they became closer to each other.

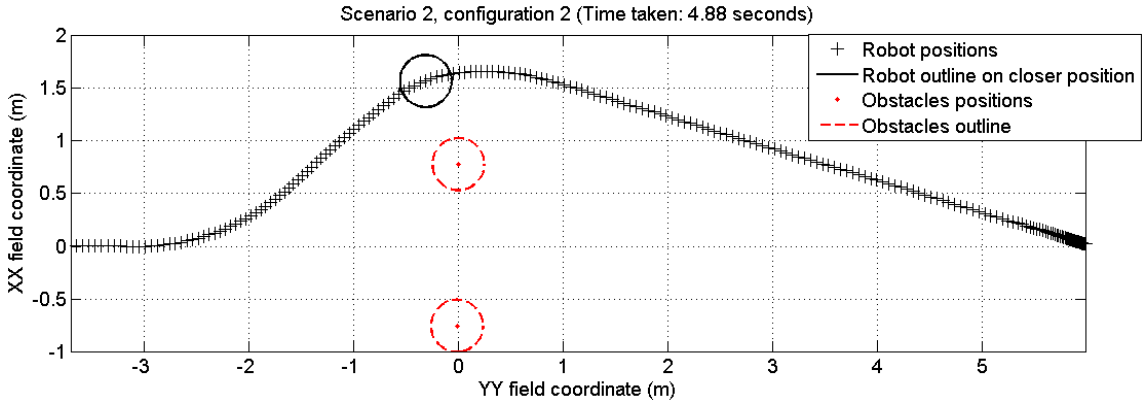
Considering the second scenario, figure 4.52 displays the complete trajectory of the robot to avoid the obstacles. Figure 4.52a, b and c represent the captures with sonar configurations one, two and three, respectively. The robot outline is represented in the point on the robot trajectory where it was closer to the obstacles, to have an idea of the clearance margin during the run. In this case one the the sonar configuration allows the robot to move through the corridor formed by the obstacles, without the need to avoid them.

In table 4.10 and for scenario two, the total time of the robot movement and the minimum distance of the robot to the obstacles during the run are presented.

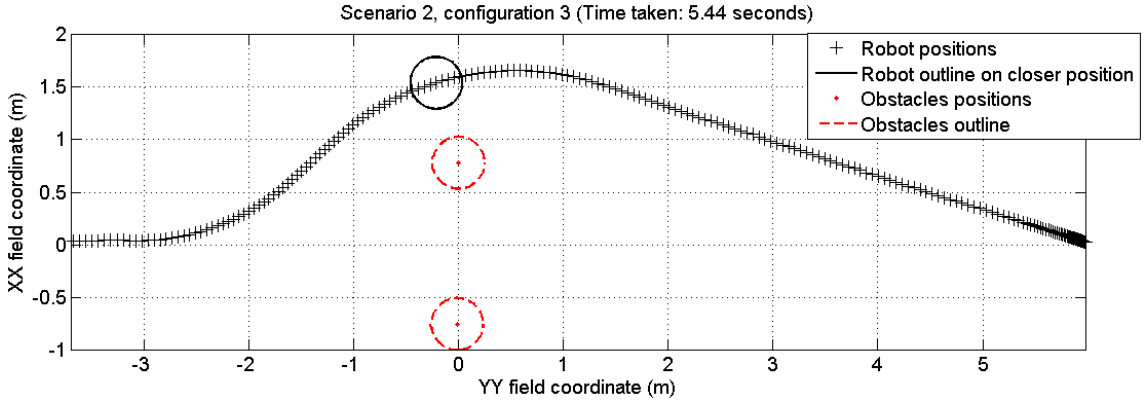
In this case, the time of the first sonar configuration is obviously much smaller, since the robot has enough space to pass through the obstacles and does not need to avoid any of them.



a)



b)



c)

Figure 4.52: Visual representation of the robot path in avoidance scenario 2. Both the robot and the two avoided obstacles are represented by their outlines in the path position where they became closer to each other.

The 0.9 m corridor of the first sonar configuration provides a good margin for our robots to go through a pair of obstacles while maintaining a comfortable distance from them. When requiring a larger passing corridor of 1.5 m still it seems preferable to maintain the number

	Min distance (m)	Time (seconds)
Sonar config 1	0.25	3.18
Sonar config 2	0.35	4.88
Sonar config 3	0.29	5.44

Table 4.10: Minimum obstacle distance and time results for scenario two.

of sonars relatively low, given the faster convergence of the robot movement.

Scenario three

Considering the third scenario, figure 4.53 displays the complete trajectory of the robot to avoid the obstacle that, in this case, is also moving in the opposite direction. Figure 4.53a, b and c represent the captures with sonar configurations one, two and three, respectively. The robot and objects outlines are represented in the point on their trajectories where they were closer to each other, to have an idea of the clearance margin during the run. In this case, 8 runs were performed for each sonar configuration and an additional statistic is presented in table 4.11.

In table 4.11 and for scenario three, the total time of the robot movement and the minimum distance of the robot to the obstacle during the run are presented. Additionally, the collision rate is presented, with the number of avoidances where a tangential collision occurred over the total number of runs.

	Min distance (m)	Time (seconds)	Hits/Runs	%
Sonar config 1	0.03	4.72	5 / 8	62.5
Sonar config 2	0.03	4.76	0 / 4	0
Sonar config 3	0.05	4.70	3 / 8	37.5

Table 4.11: Minimum obstacle distance (when no collisions occur) and time results for scenario three.

In this case, a larger corridor with a relatively low number of sonars has the tendency to provide a clearer passage, even though the clearance margin is very small. In this case, we consider that none of the configurations is completely satisfactory and thus the general sonar configuration chosen is currently the first one, with 15 sections and a corridor of 0.9 m, given the need to pass through relatively small corridors during a Middle Size League game. Some future work on this matter is discussed in section **future work**.

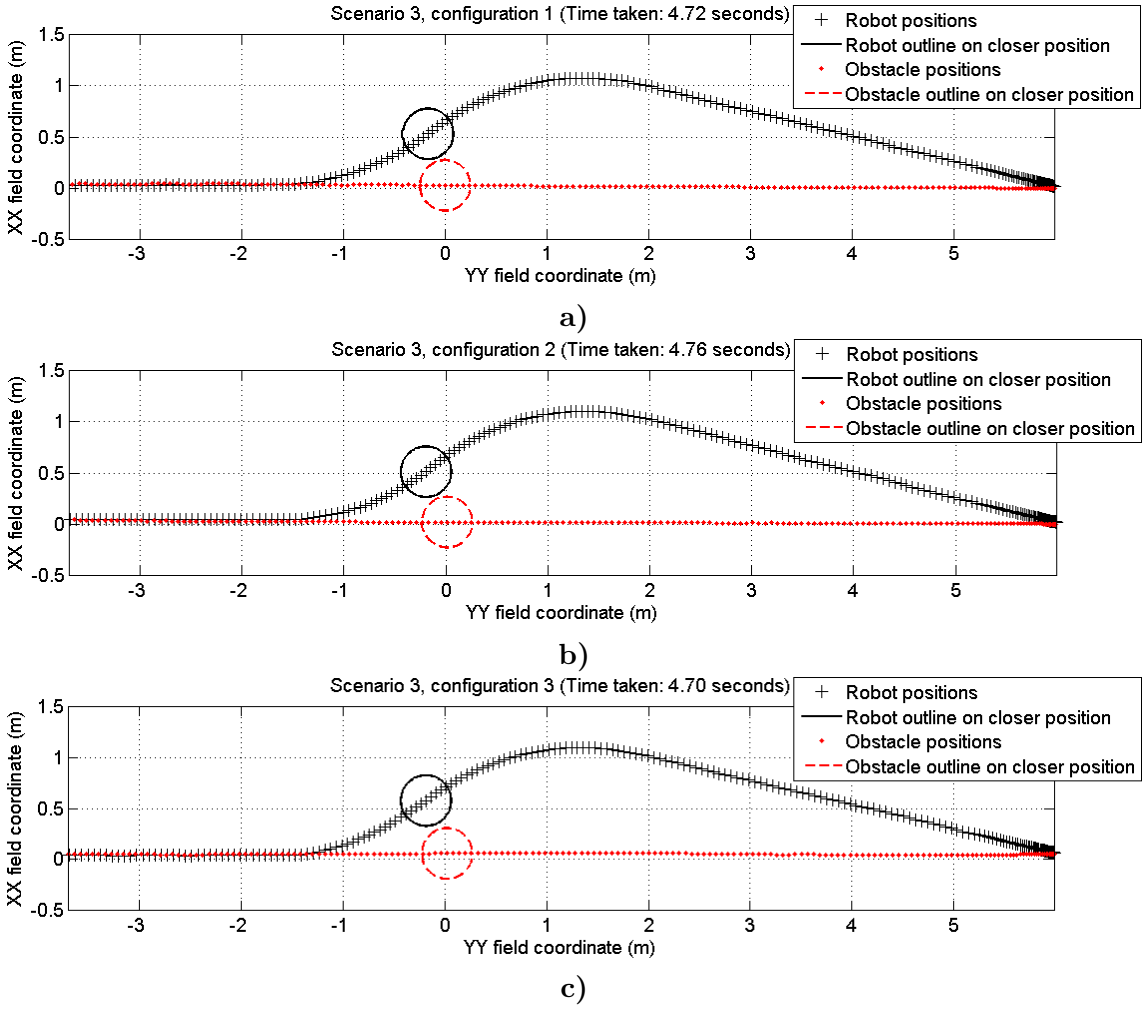


Figure 4.53: Visual representation of the robot path in avoidance scenario 3. Both the robot and the avoided obstacle are represented by their outlines in the path position where they became closer to each other. In this scenario, the path of the moving obstacle is also represented and the movement direction was perfectly opposed to the robot movement.

4.7 Trajectory generator

In mobile wheeled robots, the problem of generating a trajectory for the robot to follow is a major task to perform. In some cases, it is acceptable to coarsely define a path, simply by means of a target point (a destination), where the robot moves to, while avoiding obstacles in the way. No special constraints are posed to the robot movement during displacement. In some other situations a more finely grained definition would be desired. This occurs in the MSL, for instance, when the players try to perform dribbling: maneuvering the ball in its possession while progressing towards the goal and avoiding the opponents. This action requires a definition of the robot movement as precise and fine-grained as possible. The CAMBADA

team is at a point where a more sophisticated way of generating the robot trajectories is desirable. For that, we developed an heuristic for generating the robot trajectories, defined as a sequence of points in space that define the robot position at regularly spaced, consecutive time instants, as well as the acceleration and velocity vectors in each of those points. In the current CAMBADA control architecture, only the positions are fed to the robot control system at a fixed rate (currently, every 20 ms) [146]. As the only constraints posed on the robot path are those defined by the robot's dynamics, the proposed heuristics are directed at omnidirectional holonomic robots.

4.7.1 Assumptions

We assume that the robot is capable of correctly estimating its position, orientation and velocity in an absolute coordinate system. The algorithm computes and provides the robot a sequence of timed spatial positions as points in absolute coordinates that the robot should follow. It also computes the acceleration and velocity vectors in each of those positions, wherein these have some constant values in some phases of the trajectory, as will be further described.

Since the robots are omnidirectional, this allows us to consider separately the computation of the robot localization and orientation and to decouple the problem of trajectory estimation in two separate problems: the computation of the trajectory in the XY plane and the computation of the orientation Θ. This decoupling cannot be made without considering that there are acceleration constraints imposed by the robot mechanics, meaning that the decoupling is not possible if the estimated values for any of the components is near the limit. We will still require these two movements (position in 2D and orientation in 1D) to be synchronized, meaning that they start and finish at the same time.

The trajectories generated by the algorithm are a variant of the well known and widely used trapezoidal velocity profile: an initial acceleration phase (phase 1), a phase of traveling at constant speed (phase 2) and a final deceleration phase (phase 3). The movement to be computed is defined by:

- the initial conditions: position, p_i, and velocity, v_i
- the target or final conditions: position, p_f, and velocity, v_f
- the absolute value, or norm, for the plateau speed, v_p^*
- the absolute value, or norm, for the accelerations during phase 1, a_1^*, and phase 3, a_3^*

We will use the convention of noting with an asterisk $*$ the symbols for quantities expressed as the absolute value or the norm of the corresponding physical quantity. For instance, the plateau speed is specified as v_p^* (a real, positive scalar); the algorithm will compute the plateau velocity v_p, such that $|v_p| = v_p^*$. We will also note unit vectors with a hat $\hat{}$ symbol.

We assume that the accelerations in phases 1, a_1^*, and 3, a_3^*, can be different. This can correspond, for instance, to the fact that friction acts in opposition to the force exerted by the motors when the robot increases speed and acts reinforcing this force when it decreases speed.

The problem to be solved is thus, given the initial and final values for position and velocity, p_i, v_i, p_f, v_f; given the scalar values for the maximum plateau speed, v_p^*, and for the maximum accelerations a_1^* and a_3^*, compute a sequence $p_0, p_1, \ldots, p_n$ of $n+1$ points in the XY space, equally spaced in time and corresponding to instants separated by a time interval T_a, such that:

- $p_0 = p_i$: start point for the movement
- $p_n = p_f$: end point for the movement
- $p_1 - p_o = v_i.T_a$: initial speed
- $p_n - p_{n-1} = v_f.T_a$: final speed
- $|v_{k+1} - v_k| = a_1^*.T_a$, for $k \leq k_1$: acceleration during phase 1 (k_1 is the last sampling moment of phase 1)
- $|v_{k+1} - v_k| = a_3^*.T_a$, for $k \geq k_2$: acceleration during phase 3 (k_2 is the last sampling moment of phase 2)
- $|p_{k+1} - p_k| = v_p^*.T_a$ for $k_1 < k < k_2$: plateau speed

The robot will attempt to travel the distance s in a movement in three phases. During phase 1, it will change its velocity to align with a plateau speed, subject to an acceleration a_1^*. In phase 2, it will travel at constant plateau speed and in phase 3, it will change its velocity from plateau velocity to the target velocity, subject to an acceleration a_3^*.

When phase 3 ends, the robot velocity is the target velocity and the robot position is the target destination.

4.7.2 Heuristic for path computing

The main point in the heuristic is to find the plateau velocity vector, v_p. Consider p_α as the point where the robot enters phase 2 and p_ω the point where robot leaves phase 2. In between these two points, the robot travels in a straight line, at constant speed $v = v_p$. It is easy to see that v_p is aligned with $p_\omega - p_\alpha$.

The problem here lies in the fact that v_p is defined by p_α and p_ω, which, in turn, depend on v_p (different values of v_p will yield different values of p_α for the same v_i; the same applies to p_ω).

To solve this interdependency, an algorithm (*mov2d*, presented in algorithm 7) was devised that works iteratively to find a solution for v_p. A first estimation for v_p is to align it with $p_f - p_i$, resulting in v_p^0. From p_i, v_i, p_f, v_f and v_p^0, the algorithm computes p_α and p_ω. These later p_α and p_ω now yield v_π, an estimation for v_p, such that $v_\pi \parallel (p_\alpha - p_\omega)$ and $|v_\pi| = v_p^*$. The later v_π is combined with the previous estimation using a convergence function h (discussed later), and the process continues until convergence. In this algorithm, the counter i can be used to terminate the algorithm if it does not converge within an acceptable number of iterations.

Algorithm 7 *mov2d*: Heuristic for 2D movement

Input: $p_i, v_i \rightarrow$ initial trajectory point and velocity
$p_f, v_f \rightarrow$ final trajectory point and velocity
$v_p^* \rightarrow$ maximum norm allowed for the plateau velocity
$a_1^*, a_3^* \rightarrow$ norm of the accelerations allowed for phases 1 and 3
Output: $v_p \rightarrow$ plateau velocity for the movement phase 2
$p_\alpha \rightarrow$ point of transition from phase 1 to 2
$p_\omega \rightarrow$ point of transition from phase 2 to 3

1: $s = p_f - p_i$
2: $v_p^0 = \hat{s}.v_p^*$
3: continue = TRUE
4: **while** continue **do**
   {Estimate p_α}
5: $\Delta v_1 = v_p^i - v_i$
6: $a_1 = \hat{\Delta v_1}.a_1^*$
7: $s_1 = \frac{\Delta v_1}{a_1}.\frac{v_i + v_p}{2}$
8: $p_\alpha = p_i + s_1$
   {Estimate p_ω}
9: $\Delta v_3 = v_f - v_p^i$
10: $a_3 = \hat{\Delta v_3}.a_3^*$
11: $s_3 = \frac{\Delta v_3}{a_3}.\frac{v_p + v_f}{2}$
12: $p_\omega = p_f - s_3$
   {Estimate displacement at plateau speed}
13: $s_\pi = p_\omega - p_\alpha$
14: $v_\pi = \hat{s_\pi}.v_p^*$
15: **if** $\phi(v_p^{(i-1)}, v_\pi, s_\pi)$ **then** {ϕ defines convergence criteria}
   {Final value reached}
16: continue = FALSE
17: **else**
   {Update estimation}
18: $v_p^i = h(s, v_p^{(i-1)}, s_\pi, v_\pi)$
19: **end if**
20: i++
21: **end while**

Convergence criteria is defined by function ϕ. This function evaluates to *TRUE* when the values of v_p, p_α and p_ω provide a "sufficiently good" path for the robot to follow. One possible definition for the convergence criteria function ϕ is

$$\textbf{\textit{returns}}\ |v_p^{i-1} - v_\pi|.\frac{|s_\pi|}{|v_\pi|} < \epsilon$$

where the function computes an estimate for Δs, the difference between the traveling distance in phase 2 for the velocities $v_p^{(i-1)}$ (the estimation for plateau velocity in the previous iteration) and v_π (the estimation in the current iteration), and compares it to a given threshold, using the fact that the triangle with side $v_p^{(i-1)}$ and v_π is similar to the triangle with sides $s_p^{(i-1)}$ and s_π.

One possible implementation for h, the function that computes the new estimation for v_p based on the previous estimation and the temporary estimation v_π is

$$k = \frac{s.s_\pi}{|s|^2}$$
$$\textbf{\textit{returns}}\ k.v_\pi + (1-k).v_p^{(i-1)}$$

The value k can be read as: "the ratio of the length of the projection of s_π in s to the length of s". The role of factor k is to reduce the weight of the vector v_π when the distance traveled in the direction of the overall movement is short. In these situations, the direction of v_π is very sensitive to small changes in p_α and p_ω, increasing instability and preventing convergence of the algorithm.

Although there is so far no proof of convergence, the algorithm was tested and the results showed that it converged in all situations where:

- the condition $s.v_f \geq 0$ holds, or, in other words, v_f is not in a direction opposite to the movement direction;
- p_i and p_f are sufficiently apart and the robot has enough space to change direction at both ends of the movement (an analytical expression for the minimum distance between p_i and p_f has not yet been determined)

The case where p_i and p_f are not sufficiently separated is detected by the condition $s.s_\pi < 0$, which means that after computing the estimations for p_α and p_ω, the projection of vector $s_\pi = p_\omega - p_\alpha$ in the direction of $s = p_f - p_i$ points in the direction opposite to s. This situation, as is, has no solution by our algorithm. In this case reducing the value of v_p^*, and thus reducing the length of vector v_π, has shown to allow for convergence of the algorithm. In the actual implementation, the code in algorithm 8 is inserted in algorithm 7 immediately after computing the value of v_π.

Algorithm 7 computes the parameters that define the movement, namely p_α, p_ω and v_p.

Algorithm 8 Heuristic for degenerate movement

Input: $s \rightarrow$ distance between the initial and end points of the movement
$s_\pi \rightarrow$ distance between the plateau transition points
$v_p^* \rightarrow$ current norm allowed for the plateau velocity
Output: $v_p^* \rightarrow$ redefined norm allowed for the plateau velocity to allow convergence

```
1: if s.s_π < 0 then
2:    v_p^* = λ.v_p^* {0 < λ < 1}
3:    if v_p^* < v_{p,min}^* then
         {Could not find a viable value for v_p^*}
4:       return fail code
5:    end if
6:    v_p^i = ŝ.v_p^*
7:    i++
8:    continue {Restart loop with new value of v_p^*}
9: end if
```

The actual computation of the points in the trajectory is performed by the algorithm *motion comp*, described in algorithm 9.

At the end of *motion comp*, *pp* is a list that contains all points in the trajectory.

Figure 4.54a to d represent the major elements that define the movement: p_i, v_i, p_f and v_f, that were input to algorithm 7; the figures represent several iterations of the values estimated by algorithm 7 for p_α, p_ω and v_p. Figure 4.54e corresponds to the same trajectory, after superimposing the points computed by algorithm 9.

This approach is still under development, as some issues became evident during the development. The main issue is the perception of the robot's own velocity, which is currently affected by a relatively high noise.

The variation of the robot velocity estimations tends to provoke discontinuities on the generated trajectories that affect the behavior of the robot. During the movement, it tends to make sudden stops, especially near transition points, as the target point changes rapidly.

The approach has, however, worked with good results in situations where the robot initial and final velocities are zero, leading us to the conclusion that further work is required to improve the algorithm robustness to the sudden variations of target and velocities that occur in the MSL.

4.8 Summary

The Middle Size League is a fast paced scenario for teams of robots to operate effectively with challenges in both cooperative and adversarial scopes. Several of the world model entities presented in section 3.3 were addressed in this chapter as part of this PhD.

An overview of the omni directional vision process was given as it is the main source of information available to the robot. The advantages and limitations of this system were

Algorithm 9 *motion comp*: Compute points in a trajectory

Input: $p_i, v_i \rightarrow$ initial trajectory point and velocity
$p_f, v_f \rightarrow$ final trajectory point and velocity
$a_1^*, a_3^* \rightarrow$ norm of the accelerations allowed for phases 1 and 3
$v_p \rightarrow$ plateau velocity for the movement phase 2
$p_\alpha \rightarrow$ point of transition from phase 1 to 2
$p_\omega \rightarrow$ point of transition from phase 2 to 3
Output: $pp \rightarrow$ list of trajectory points for the robot to follow

1: $\Delta v_1 = v_p - v_i$ {Δv in phase 1}
2: $t_1 = \frac{|\Delta v_1|}{a_1^*}$ {Time to complete phase 1}
3: $n_1 = \lceil \frac{t_1}{T_a} \rceil$ {Number of cycles in phase 1}
4: $a_1 = \hat{\Delta v_1}.a_1^*$
5: $\Delta v_3 = v_f - v_p$ {Δv in phase 3}
6: $t_3 = \frac{|\Delta v_3|}{a_3^*}$ {Time to complete phase 3}
7: $n_3 = \lceil \frac{t_3}{T_a} \rceil$ {Number of cycles in phase 3}
8: $a_3 = \hat{\Delta v_3}.a_3^*$
{Start computing the points}
9: $p = p_i; v = v_i$ {Initial values}
{Phase 1}
10: **for** $n = 1$ **to** n_1 **do**
11: $v = v + a_1.T_a$
12: $p = p + v.T_a$
13: pp.add(p)
14: **end for**
{Phase 2}
15: $p = p_\alpha; v = v_p$ {Initial values for phase 2}
16: pp.add(p)
17: **for** $n = 1$ **to** n_2 **do**
18: $p = p + v.T_a$
19: pp.add(p)
20: **end for**
{Phase 3}
21: $p = p_\omega$ {Initial point for phase 3}
22: pp.add(p)
23: **for** $n = 1$ **to** n_3 **do**
24: $v = v + a_3.T_a$
25: $p = p + v.T_a$
26: pp.add(p)
27: **end for**
28: $p = p_f$ {Last point in the movement}
29: pp.add(p)

exposed and should help in the understanding of some approaches and decisions during the development of further work.

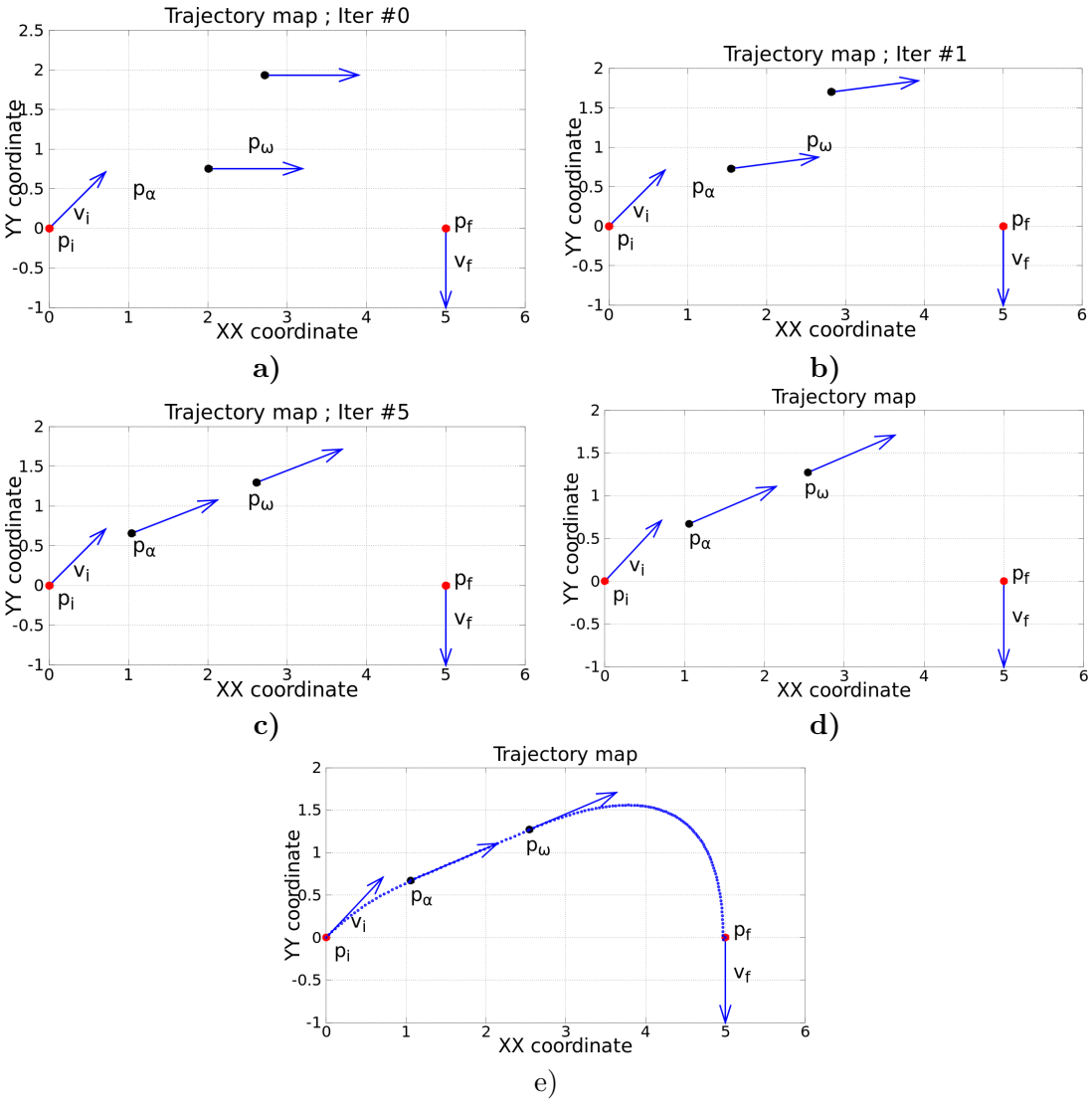


Figure 4.54: Visual representation of an example of a trajectory. In **a)** to **d)** the elements that define the movement, both inputs and outputs of algorithm 7. Each image is a different iteration of the algorithm with the evolution of the output values p_α, p_ω and v_p, which is the vector represented in both p_α and p_ω. In **e)** the points computed by algorithm 9 that define the targets for the robot are overlapped with the movement definition elements.

The ball was object of analysis in terms of uncertainty on its position estimated by the vision process. The raw information available from the vision along with the model of the noise that affects that information are used on a Kalman filter to obtain a more precise and stable position, which is then used to infer the velocity of the ball, by means of a linear regression. Results from both the filtering and the linear regression for velocity were presented.

Despite the effectiveness of the perception of the ball by using the information from the omni directional camera, this sensor has an inherent limitation of ball visibility when the ball

is kicked off the ground. To tackle this problem, several approaches were made by using an additional sensor to detect the ball in the air. Initially a single RGB camera was used, with some promising results but eventually became not good enough for the desired performance. With the advent of the Kinect sensor a new approach was developed using such sensor. The results are more precise than the RGB camera approach and there is now room for progress concerning ball detection and tracking.

Robot localization has been refined by the introduction of a heading sensor that allows to detect situations where the robot estimated orientation has degraded or inverted. By detecting these situations, actions can be taken in order to correct such errors by re-localizing the robot. The heading sensor has been analyzed and changes were made to both hardware and software so that the reliability of the heading sensor could be improved.

The robots on the field are an integral part of a soccer game. From a single player perspective, there are other entities on the field that are visually detectable and the need to identify them is great, so that the decisions are taken based on as much information as possible. The type of information about visually detected robots/obstacles provided by the vision has been changed in order to achieve a better discrimination of each among them. With this improved raw perception and through information crossing, a robot is capable of identifying which of the robots/obstacles are team mates and which are not. Furthermore, from the list of obstacles that are not team mates, keeping track of each of them is advantageous for decision processes, which currently make use of this information for pass coordination and opponent pass covering purposes.

Some of the improved information achieved concerning precision of obstacles motivated the development of an avoidance methodology important for the CAMBADA robots to play more effectively. This methodology was presented and analyzed, although the improvements it brought were mostly observed during real matches by the development team.

Having a last resource approach for avoiding obstacles reactively, a methodology to control the robot movement by defining a set of finely tuned way points is desirable. A method to estimate a set of finely grained way points over a trajectory was devised, that allows the configuration of a given movement by defining the initial and final position and velocity. With the sequencing of movements that have final and initial velocities synchronized, we should be able to create complex movements that do not require the robot to stop at each way point, as is the current case.

Chapter 5

Perception and software architecture for SPL

The Portuguese Team is a robotic soccer team that participates in the Standard Platform League of the RoboCup. In this league, all the teams use same robots, the NAO robots, and cannot make any hardware change.

The Portuguese Team appears as a way for the IEETA robotics group to start working in the area of applications with real humanoid robots. The team is a mixed team, with collaboration of elements from the FC Portugal team (2d & 3D simulation), CAMBADA and 5DPO teams (Middle Size League), from universities of Aveiro, Porto and Minho. The 3D simulation league uses simulated NAO humanoid robots and thus the know how of the FC Portugal 3D team allowed to create the body movement control layer. In the beginning of the project, also the vision system for the NAO camera was developed. In the end of a first "phase" of the project the NAO could move and could extract pixel coordinate information from the camera. The work accomplished during this PhD over this platform comes after the referred modules were created. The need for a worldstate representation was evident and indispensable. Also, the general architecture of the code needed to be revised and organized. This chapter presents the general software architecture created for the Portuguese Team and the work developed in terms of perception.

5.1 Software architecture

The software developed for the Portuguese Team robots uses a similar distributed architecture as the one used in the CAMBADA robots [60]. This distributed architecture is based on several processes running on each robot which use a shared memory to interact, particularly the Real-time database (RTDB).

Following the CAMBADA software approach, the software used in the robot is also distributed. Therefore, five different processes are executed concurrently. All the processes run

on the robot's processing unit in Linux. Furthermore, a NAOqi module was also implemented, to handle the communication between the agent process and the robot Device Communication Manager (DCM) (figure 5.1).

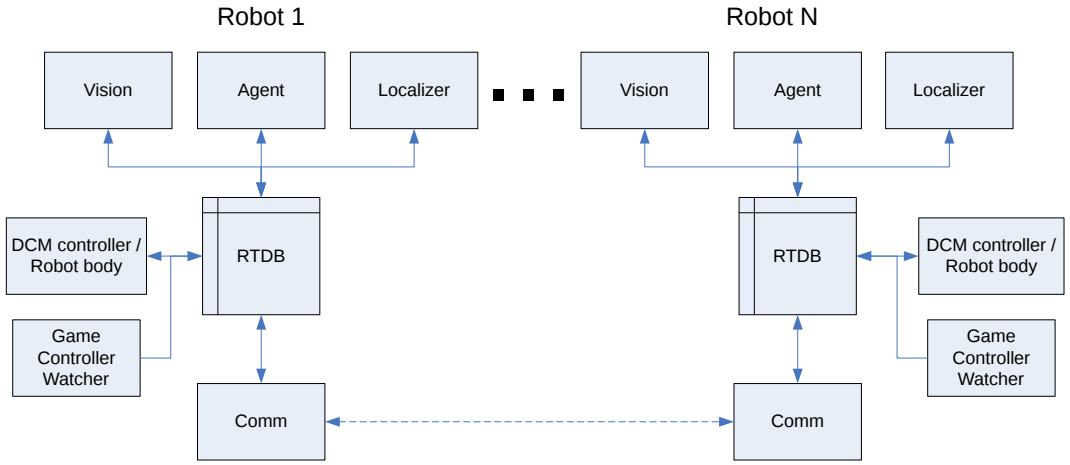


Figure 5.1: Diagram of the software running on the robots.

Inter-process communication is handled by means of a RTDB, implemented as a block of shared memory. The RTDB is divided into two regions, the local and the shared one. The local region allows communication between processes running on the robot. The shared region implements a Blackboard communication paradigm and allows communication between processes running on different robots. All shared sections in the RTDB are kept updated by an adaptive broadcasting mechanism that minimizes delay and packet collisions.

The processes composing the Portuguese Team robot software are:

- **Vision**: is responsible for acquiring the visual data from the robot cameras.
- **Agent**: is the process that integrates the sensor information and constructs the robot's worldstate, taking the decisions based on this information.
- **Comm**: handles the inter-robot communications, receiving the information shared by other robots and transmitting the data from the shared section of the RTDB.
- **GCWatcher**: listens to the game controller and writes all the referee information on the RTDB.
- **Localizer**: is responsible for running the localization algorithm whenever the agent requires it, through RTDB communication.
- **DCM controller**: this is a NAOqi module which uses the internal API to interact with the robot sensors and actuators, reading the values from the sensors and making

them available to the agent, and reading the agent commands and sending them to the actuators.

Since the Agent process is the one that defines the body movements, in order to maintain a control as smooth as possible, this process, along with the Naoqi which actually executes the movements, are set to run under a linux FIFO scheduler with the top priority definable. Since all other processes run under the normal best effort scheduler that has lower priority than the FIFO scheduler, these two processes run with top priority of access to the processor. This means that these processes will not lose control of the processor for any other process (except system processes necessary for the operating system and over which we have no control). The remaining processes run on a best effort basis scheduler. This is a need that rises from the little computational power available for all the computational heavy tasks that the software must execute.

Since the DCM period is defined by Aldebaran to 10 milliseconds, the agent process, which is responsible for estimating the motor positions, was built over that constraint and thus runs with a cycle time of 10 milliseconds.

5.1.1 Agent

The desired software architecture for our agent should follow a well defined task sequence while allowing several modules and tasks to be taken out or inserted in the loop. It is also desired that several methodologies and code structures can be updated or replaced. Most of the modules described next have a specific API to interact with and are confined within their own execution block. Thus, they can be replaced by other methods, if so desired, with little amount of effort.

5.1.1.1 Cycle loop

The agent process, being responsible for all the decisions throughout a run, is a cyclic process that keeps updating all the information available, both from other processes to it and from it to other processes.

The agent process loop is organized in a sequence of tasks that need to be executed in the correct order, as they have dependencies from the previous one. The general steps performed by the agent loop follow the sequence presented next (figure 5.2).

- Proprioception update: this first step on each cycle of the agent must be for it to know what is the current state of its own body. This task is the one that actually defines the cycle time, since the method to get the values of all the body sensors from shared memory is blocking until the DCM Controller process that is filling that memory allows access. After having the data, the first step of proprioception is to update the body model, particularly the kinematic model which in the end provides the relative

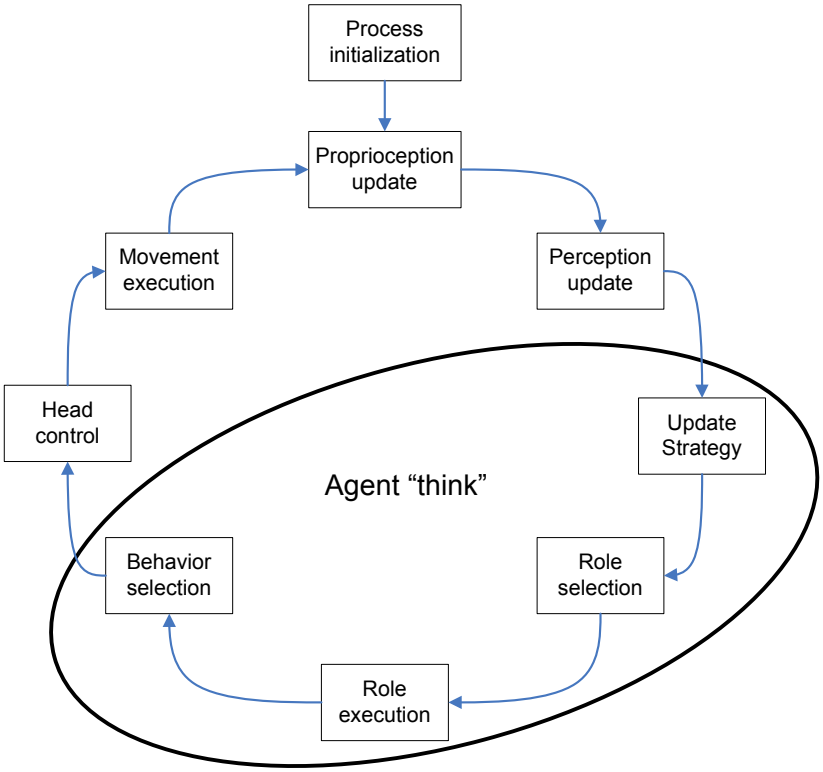


Figure 5.2: Tasks executed by the agent process cycle loop.

positions of all the body parts in relation to the robot head, in the form of a set of transformation matrices. The reason to use the head as base for the representation of the transformation matrices is that the camera is placed there and thus the posterior perception module uses them directly. The model used for defining the kinematic chain of the robot on the Portuguese Team is a dynamic and configurable model. The robot model is defined by a configuration file in XML format which enumerates a list of body parts, characterized by a name and a mass, and a list of joints, characterized by name, identifier of perceptor and effector objects, rotational axis, limit values and the two anchor points from the body parts list, which include the translations between each of them. The model allows that changes to the robot physical structure can be easily introduced, by simply updating the configuration file with the new values. An example of configuration file is visible in figure 5.3.

- Perception update: this step is responsible to create the world model of the robot and update all relevant information. It starts by updating the game time and the gamestate sent by the refereeing entity. It then gets the latest visual information made available by the vision process and the latest worldstate information made available by the team mates. It then updates the robot localization, either based on new visual information or

```
<!-- Nao Masses and Joints for hardware version 3.3 -->
- <agentmodel name="nao" type="humanoid" rsgfile="nao/nao.rsg">
    <bodypart name="head" mass="0.52065"/>
    <bodypart name="neck" mass="0.05930"/>
                (...)
    <!-- HeadYaw -->
  - <joint name="head1" perceptor="hj1" effector="he1" xaxis="0" yaxis="0" zaxis="-1" min="-119.5" max="119.5">
      <anchor index="0" part="neck" x="0" y="0" z="0"/>
      <!-- 0, 0, NeckOffsetZ -->
      <anchor index="1" part="torso" x="0" y="0" z="-0.12650"/>
    </joint>
    <!-- HeadPitch -->
  - <joint name="head2" perceptor="hj2" effector="he2" xaxis="0" yaxis="-1" zaxis="0" min="-38.5" max="29.5">
      <anchor index="0" part="head" x="0" y="0" z="-0.06790"/>
      <!-- 0, 0, CameraTop -->
      <anchor index="1" part="neck" x="0" y="0" z="0"/>
    </joint>
                (...)
```

Figure 5.3: Excerpt from the xml configuration file with values for the NAO robot.

solely on odometry update and finally it updates all the information about the objects of interest (ball, goals, other players).

- Agent "think" process: after having the most recent information about itself and the worldstate, the agent can finally analyze and decide how to act:

  - Update strategy: the first thing to do, now that it has its information and the information from the team mates, is to update the strategic model so that its decisions will accomplish the desired outcome in terms of general field placement.
  - Role selection: based on the defined strategy, the agent will then have to decide which role to take on the game. This role defines the general attitude that the player will take. An agent with the role Striker, for instance, will actively go for the ball, while a midfielder will keep its position within the defined adaptive formation, without directly going for the ball.
  - Role execution: the chosen role will then be executed. For simplicity sake on the development of roles, the general execution starts by automatically guaranteeing that the robot is not fallen and also check for forced relocalization commands. If everything is fine up to this point, then the specific role tasks go into action.
  - Behavior selection: each role has specific tasks to accomplish and wants the robot to execute different types of actions/movements. The role is responsible for, usually within a defined set of states, define what is the behavior that the robot must execute. These behaviors define how the robot arms and legs must move to achieve a defined action. Given the several possibilities of walking algorithms and parameterizations, an API was created to allow a role developer to execute actions like walk forward, walk right, curve left, etc, without having to explicitly set the

walking parameters or even implementation at the role level. This allows the role developer, usually focused on higher level decision, to abstract himself from the specific implementation/parameterization of how to achieve a forward walk or any other base movement.

- Head control: generally the roles define how they want the robot to move around the field of play. The head however, is treated independently so that the perception layer can be responsible for controlling it according to a given profile. The most used are to control the head to keep the ball in sight, and to sweep around in search for visual landmarks (lines and goals).

- Movement execution: finally the agent can set all the values of the actuators previously defined and update these values on the shared memory so that the DCM controller can execute them on the robot body in the end of the cycle.

5.1.1.2 Roles & Behaviors

The decision process of the agent is organized in several different stages. These stages are organized in roles and behavior.

Roles The decision first step is to choose a role to execute at the moment. This decision can be a simple static role assignment that only depends on a very small set of conditions (as the penalized or the goalie roles, for instance, which only depend on game Controller state and/or robot number) to a more complex assignment which depends on the conditions of the robot and the world around, including strategic issues.

The role is then responsible for a more detailed analysis of the conditions and for taking actions to achieve its goals. These actions are the behaviors available to the robot.

The assignment of the roles is made dynamically, so each robot can assume any given role during a game, dependent on the game conditions. These conditions are mainly dependent on the ball position on the field, which is the main game object. All the robots are assigned specific roles according to their position relative to the ball, i.e. a robot that is near the ball will be assigned a role to try to move to it, while robots farther from the ball will be assigned other types of roles which do not try to move to the ball but rather to a position on the field that is advantageous, either for defense purposes or attack purposes.

Each role and behavior is an instance that can be used at any time. Currently, the main roles are (figure 5.4):

- **Goalie**: this role implements a state machine for the behavior of the goalie robot. It tries to keep itself inside the goal area, tracking the ball trajectory and trying to adjust its position so that it intercepts the ball trajectory to the goal. When the ball comes close to the goal, it tries to lay down to defend it.

- **Striker**: this is the main role during free play. The Striker is the robot that more actively goes for the ball. It actively tries to get to the ball, dribble it in the opponent goal direction and kick to the goal.

- **Midfielder**: this role is a supporting role that exists during free play. Apart from the Goalie and the Striker, all the other robots assume this role. Each of them keeps a strategic positioning, estimated by this role, to keep the team coherent and cooperative. Any Midfielder that reaches a situation more advantageous than the Striker will be reassigned as Striker, and the former one will fall back to the strategic positioning as a Midfielder.

- **Penalized**: this is the role assigned to robots to keep them inactive when the referee penalizes them for any reason. In order to deactivate the robot motors, so that they are not kept under strain during all the penalization time, the robot autonomously moves to a crouched position that is stable even when the motors are off. That way, during penalization with the motors off, the robot saves battery power and releases strain from the motor so that they do not heat.

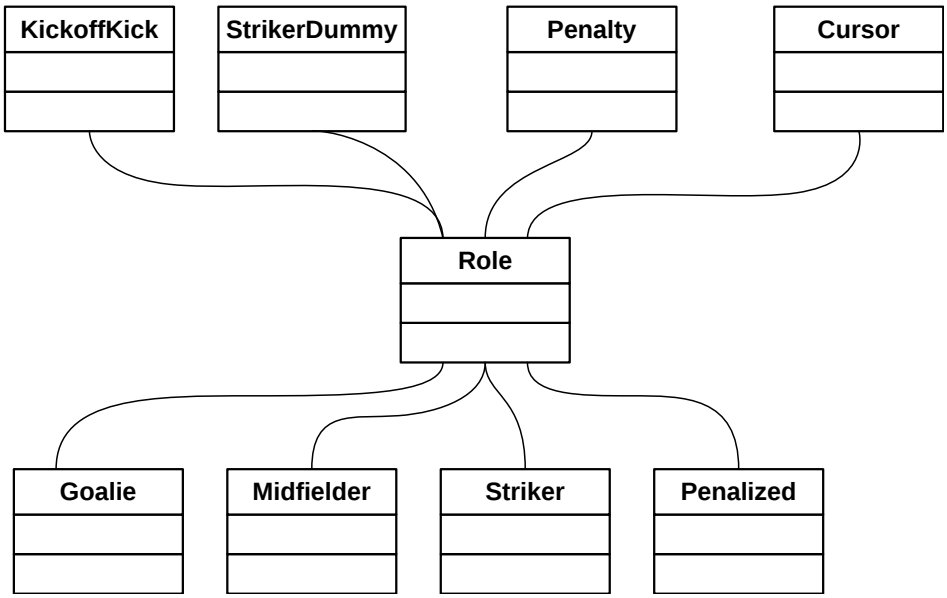


Figure 5.4: Role structure diagram.

These roles are used during a game on free play. However, a set of other roles are implemented, both for development and non free play and fall-back situations, like manual game controller mode or unavailability of localization. Some examples are (figure 5.4):

- **Cursor**: this is a role that allows remote control of the robot movements. This role is used by the cursor tool (section 6.2.4) and works almost as a gateway by reading the

movement commands sent from the cursor tool via a shared memory slot of the RTDB. The cursor itself allows to test behaviors without the need to implement a whole role and decision processes to activate it, which is important during the behavior development phase. It also allows the parameterization of an omni directional walk methodology in terms of X, Y and Theta velocities, which allow for quick testing on any new surface to discover which parameters would be better for that surface.

- **Penalty**: the role penalty is used for penalty shootout situations which occur in the end of games that cannot end with a draw.

- **StrikerDummy**: this is a role that makes the robot search and go for the ball without using localization. In this case, it will just try to go for the ball directly and dribble it in the same direction from where it approached the ball. This is done with the objective of keeping the ball moving so the game won't be stuck, because there is no way for the robot to know and choose a direction for dribbling. Even if it would see a goal frame, since they are the same color, it would not know if it is the opponent goal or not, thus the approach of just keeping the ball moving and try to avoid own goals.

- **KickoffKick**: when localization is not available, this role is a special role used in a kickoff in our favor. In this situation, and despite there is no localization and the robots do not know where they are, we know that the robots are all facing the opponent goal, due to the rules of manual placement, which define well known poses for each of the team robots. Thus, the robot that is closer to the ball goes to it and kicks the ball forward before falling back to StrikerDummy.

Behaviors In the Portuguese team context, the behaviors are the low level methods for controlling the robot body, the entities that calculate the angles for all the body joints in order to obtain a sequence of body movements that create the desired effect.

Several walking behaviors developed and optimized for the FC Portugal 3D simulation team [147] have been adapted and tested [148]. Currently, the existing behavioral structure is as depicted in figure 5.5, being the main behavior models the Slot and the CPG behavior models [149] when performing general movements like getting up, swinging the legs for kicking and other generally "on the spot" movements. As for the walk, the TFS walk engine is the one currently more stable and faster on the field. Additionally, we also make use of the Aldebaran walking engine for tests that do not require much speed but require stability in a less controlled ground.

Abstraction layer While the instances of the behaviors are platform dependent, most of the software architecture from the roles to the point prior to the estimation of the motor setpoints is generic. The roles, on the other hand, should be as independent of the behavior

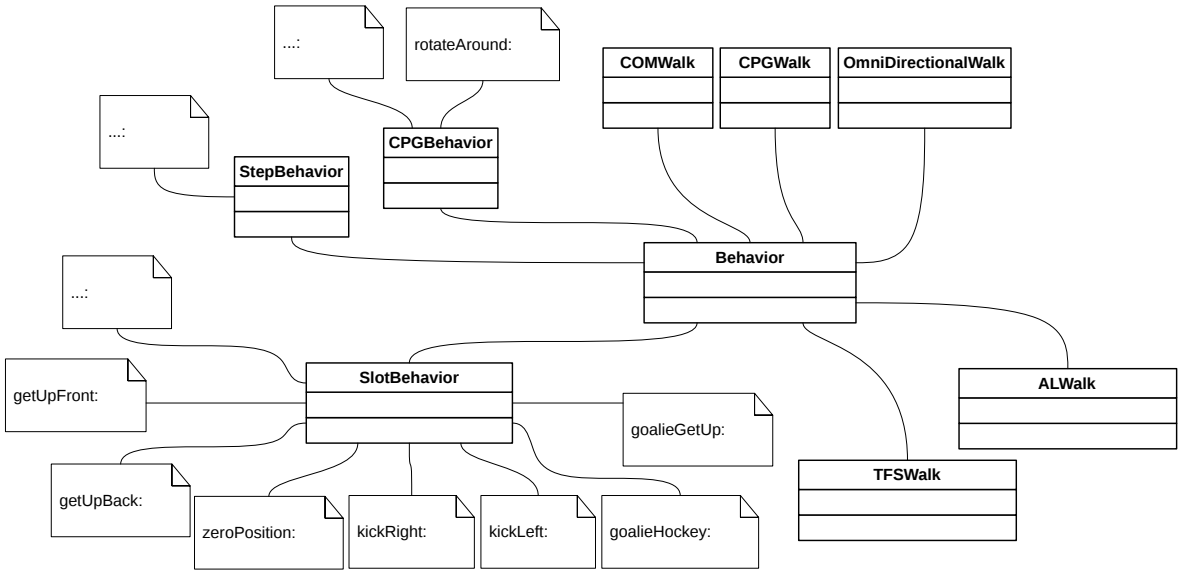


Figure 5.5: Behavior structure diagram.

implementation as possible. Thus an abstraction layer emerges at role level, which allows the role creators to abstract themselves from the specific behavior chaining and parameterization and develop in terms of more general ideas. This separation is particularly important in terms of walking movements, since the walking methods are still under heavy development and change rapidly.

This abstraction layer provides an API for role development that includes a set of movements that, when combined, should provide all the navigation actions necessary for moving the robot around the field with any necessary objective. The implementation of the actions on this layer will be the only place where there is the need to initialize and configure the walking behavior. This allows that when there is the need to change the implementation of the walk to another type of behavior or to change the parameterization of the current walking behavior, this is the only block in the code structure that needs to be changed.

The main general actions of the soccer robot are explained next (figure 5.6). The presented sequence follows a top down sequence, and thus, in each action, references to still not presented actions may occur.

- **moveToBall**: This is the most high level action of the abstraction layer, where the role can simply define that it wants to go to the ball, wherever it is on the field. The only parameter needed is the final heading desired when reaching the ball, since the ball position is known and available in the world model. This action will make use of two different actions, depending on how to approach the ball. Currently this action moves directly to the ball, without taking into account the final heading desired, and thus only uses one of the possible action chaining, the *moveToBallRel* action, presented further

bellow.

- **moveToPoint**: This action intends to move the robot to a specific point on the field. The action will make use of the *moveToPointRel* action (presented bellow) by updating it every cycle. It will also be the one responsible for planning the necessary checkpoints on the movement to guarantee the arrival at the destination with a given heading. The interface of this action includes two parameters which are the field absolute position to move to and the heading that the robot should have when reaching the target position.

- **rotate**: The rotate action is currently responsible for turning the robot to a given heading available as a parameter for the action. The turn is performed on the spot in the direction closer to the final heading. It is also intended to allow the user to provide a radius over which the robot should turn, making the rotation not on the spot but around a spot, like rotating around the ball for alignment. For this second option, the radius must be passed as a parameter and the movement will be composed by both rotation and lateral linear speeds in order to obtain a movement over a circle perimeter.

- **moveToBallRel**: This action is a specialization of the move to a relative point action (presented next). It was created due to the fact that moving to the ball is one of the most common needs, which justifies a dedicated action available for it. It does not take any parameter, since it moves directly to the ball position, available in the world model. The ball relative position is used for the invocation of the *moveToPointRel* action.

- **moveToPointRel**: This action is the one responsible for heading to a given point, relative to the robot. The relative target to move to is available as a parameter. Each cycle the direction of the point is evaluated and the action decides to move forward, move while turning to a given direction to correct small misalignments or, if the misalignment with the destination is very high, rotate in place before proceeding. Of course that, also every cycle, the point to move must be updated. This action is one of the lower level actions available on this abstraction layer.

- **rotateRel**: This action makes the robot rotate around itself in a specific direction that arrives as a function parameter. Additionally, a second parameter is available to allow rotation around a point, defined by a radius of rotation. As stated before, in action rotate, this must be achieved by combination of rotation and lateral movement. This action is paired with *moveToPointRel* as the lower level actions, meaning that all the other actions are composed by combinations of these lower level ones.

The actions previously described (represented to the left of the dashed line in figure 5.6) are still completely behavior independent actions since they are used to define high level commands for moving or rotating to points, which allow an abstraction of thought when creating a role.

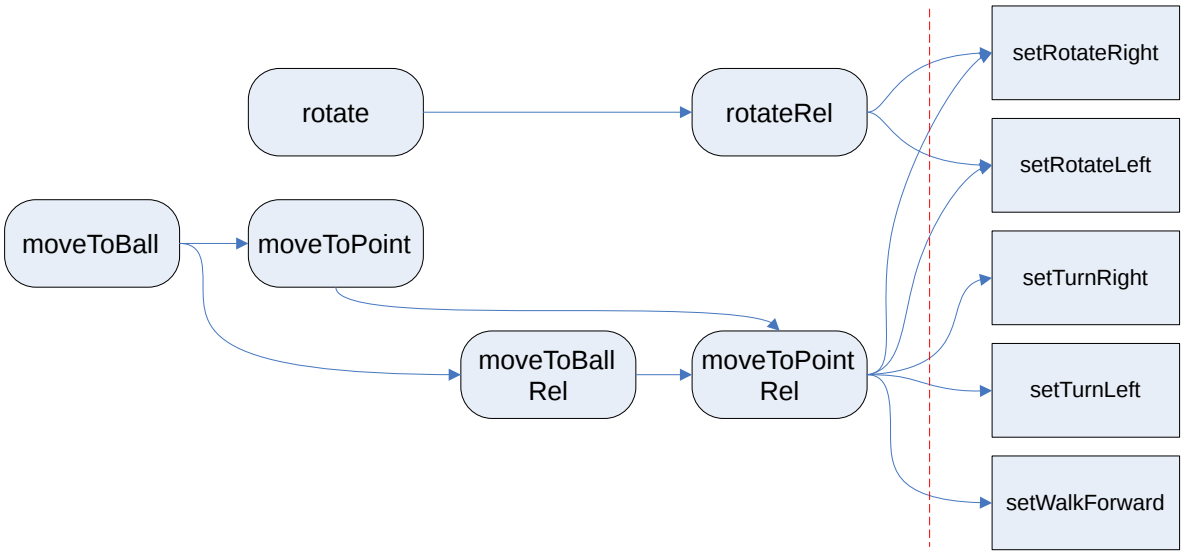


Figure 5.6: Diagram of the existent actions on the Role↔Behavior abstraction layer.

At role level there is another abstraction layer, which is the one responsible for actually defining the behaviors needed to accomplish the basic actions. Currently this layer has 5 basic actions (represented to the right of the dashed line in figure 5.6). These basic actions are the ones which declare and parameterize the default walking method. These 5 actions, implemented as 5 functions, are the only functions where the development team need to make changes when they want to change the default walking method.

- **setWalkForward**: This action provides the implementation and parameterization of the default walking method. It will internally define all the parameters necessary for the walking method to move the robot straight forward at a speed that is stable. Depending on the used walking method, the speed can be fixed or it can be read from a globally available variable that is dynamically defined by the agent at each cycle.

- **setTurnLeft/Right**: These actions configure the current default walking method to make the robot execute a curving movement to the left or to the right, respectively. Just like walking forward, the linear and angular speed for obtaining the curve can be fixed or dynamically defined by the agent, depending on the current walking method.

- **setRotateLeft/Right**: These actions use the current default walking method to make the robot rotate around itself to the left or to the right, respectively. Once again, the angular speed for the rotation can be either fixed or dynamically defined, depending on the used walking method.

5.2 Perception on SPL

The perception of the NAO robots is based on the use of cameras placed on the robot's head. The vision module/process is responsible for detecting the objects of interest and estimate the coordinates of the objects in the image.

Contrary to the CAMBADA case, where the camera is fixed on the robot body and provides images on a plane parallel to the ground, the NAO camera provides images in very different planes, depending on the position of the head and body. While in CAMBADA the vision process directly provides object coordinates in meters because of the use of a fixed pixel-distance mapping that does not depend on any other process, in the NAO, the vision process is only capable of providing pixel coordinates on the image, as the mapping to distances cannot be determined by the vision process. The agent is the entity that has access to the robot model and motor positions and thus can estimate metric coordinates from the pixel coordinates.

For the robots to be able to move and execute tasks on the environment making use of localization and moving to specific positions, there is the need to know where the objects are in terms of metric distances. This section presents several steps needed to achieve a metric representation of the objects on the field, describing the worldstate of the robots.

5.2.1 Projection model

To accomplish the ground point projection, we use the pinhole camera model to analyze the geometrical relations of coordinates in 3D space and their projection into the camera 2D CCD [150]. The first step was to make a static analysis of a vision system, in a way similar to the one presented for the CAMBADA team by Martins *et al.* [151], with the necessary adaptations for a more generic camera pose relative to the ground, presented in this section. Figure 5.7 presents two schematics of a camera system, side view and a perspective view.

The initial analysis was based on a system with a fixed camera which, similar to the top scheme of figure 5.7, was placed at a fixed height, pointing towards the ground at a given angle and with no other rotation affecting it. The several measurements involved in this static analysis include parameters related both to the camera intrinsic data and measurements extrinsic to the camera. The intrinsic camera values *focal length*, *pixel size*, $pixel_n$ and $pixel_m$ are equivalent to the front camera analysis of section 4.3.3.1 and thus are not visually represented here.

- *focal length* - distance between lens and CCD.
- *pixel size* - the physical size of each pixel on the CCD.
- $pixel_n$ - number of pixels (n) along a CCD column.
- $pixel_m$ - number of pixels (m) along a CCD row.

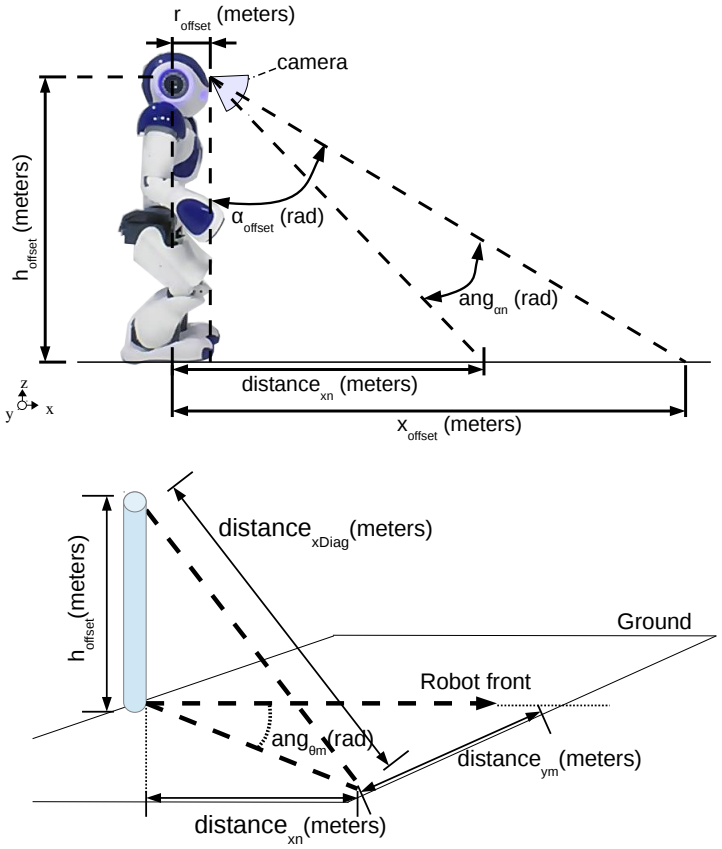


Figure 5.7: On top, side view schematic of the vision system. On bottom, a perspective view schematic of the vision system.

- h_{offset} - height of the camera relative to the ground.
- r_{offset} - radial distance from the camera to the robot center.
- α_{offset} - angular offset from the vertical axis to the camera focal axis.
- x_{offset} - distance from the projection of the center of the robot on the ground to the point in the center of the image, projected on the ground.
- $ang_{\alpha \mathrm{n}}$ - angle measured between the camera focal axis and the line between the CCD center and the projection of $pixel_n$ on the ground.
- $ang_{\theta \mathrm{m}}$ - angle measured between the robot frontal axis and the line between the camera center projected on the ground and the $pixel_m$ ground projection.
- $distance_{xn}$ - distance from the projection of the center of the robot on the ground, measured along the robot front axis, to the intersection of a line perpendicular to robot front which passes through $pixel_n$ projected on the ground.

- $distance_{xDiag}$ - distance from the camera focal point to $pixel_n$ projected on the ground.

- $distance_{ym}$ - distance from the projection of the center of the robot on the ground, measured along the robot side axis, to the intersection of a line parallel to robot front which passes through $pixel_m$ projected on the ground.

From the camera focal point, which is considered the reference of the vision system, we can easily verify that:

$$\alpha_{offset} = atan\left(\frac{x_{offset} - r_{offset}}{h_{offset}}\right) \tag{5.1}$$

From expression 5.1 we can generalize to an angle $ang_{\alpha n}$,

$$ang_{\alpha n} = \alpha_{offset} - atan\left(\frac{distance_{xn} - r_{offset}}{h_{offset}}\right) \tag{5.2}$$

Given the geometric properties of the camera vision system, we can relate an angle $ang_{\alpha n}$ and a pixel along a vertical column of the CCD, assuming that the CCD is not rotated over the robot front axis, i.e. it does not have a roll angle.

$$pixel_n = round\left(\frac{tan(ang_{\alpha n}) \times focal\ length}{pixel\ size}\right) \tag{5.3}$$

From expressions 5.2 and 5.3, we obtain

$$pixel_n = tan\left[\alpha_{offset} - atan\left(\frac{distance_{xn} - r_{offset}}{h_{offset}}\right)\right] \times \frac{focal\ length}{pixel\ size} \tag{5.4}$$

We can isolate α_{offset} from expression 5.4,

$$\alpha_{offset} = atan\left(\frac{pixel_n \times pixel\ size}{focal\ length}\right) + atan\left(\frac{distance_{xn} - r_{offset}}{h_{offset}}\right) \tag{5.5}$$

With some manipulation of expression 5.4, the distance corresponding to each pixel can be found by

$$distance_{xn} = r_{offset} + h_{offset} \times tan\left[\alpha_{offset} - atan\left(\frac{pixel_n \times pixel\ size}{focal\ length}\right)\right] \tag{5.6}$$

We can thus obtain the distance, from the robot center, of any point on the image, using expression 5.6.

Following a similar analysis and based on the schematic of figure 5.7, we know relation between $ang_{\theta m}$ and distances $distance_{xn}$ and $distance_{ym}$ projected on the ground:

$$ang_{\theta m} = atan\left(\frac{distance_{ym}}{distance_{xn} - r_{offset}}\right) \tag{5.7}$$

Since we have a height associated, and the horizontal angle has a relation with both the distances (XX and YY), we can derive the following

$$\begin{aligned} distance_{ym} =& pixel_m \times distance_{xDiag} \times \frac{pixel\ size}{focal\ length} \\ & \times cos\left[\alpha - atan\left(\frac{distance_{xn} - r_{offset}}{h_{offset}}\right)\right] \end{aligned} \tag{5.8}$$

The result of a point in pixel coordinates, $(pixel_n, pixel_m)$, transformed through the described projection model is a point in metric coordinates, $(distance_{xn}, distance_{ym})$ which is a point measured relative to the robot center projection on the ground, with the XX axis being the front of the robot and the YY axis being the left side of the robot.

5.2.1.1 NAO vision system

While in the initial analysis performed, the camera was fixed with a constant relation to the ground and with a fixed rotation in only one axis, in the case at hand with the NAO robot, the camera is placed on the head, which is a mobile part of the robot. Thus, in this case, we have to consider that the camera has rotation on all three axis XX, YY and ZZ, commonly known as roll, pitch and yaw respectively.

Looking at a camera in space, and considering XX the camera focal axis with the YY/ZZ plane on the CCD, we can see that if we apply a roll angle to the camera, we have a rotation of the pixels. The first step of the coordinates estimation is to rotate the given pixel by the roll angle of the camera to correct that distortion.

The pitch analysis basically results in the α_{offset} that allows us to get the coordinates relative to the current image. Through expressions 5.6 and 5.8 we estimate the coordinates of the given pixel considering that the front of the robot is the current direction of the camera.

Finally, knowing the yaw of the camera, we can rotate the given point, already in ground plane coordinates, and obtain the final coordinates, relative to the robot front.

5.2.1.2 Extracting camera placement angles

To get the roll, pitch and yaw angles of the camera on the space, we make use of the robot kinematics, combined with the use of the inertial unit present on the robot. Moreover, each camera of each robot has different angles relative to the head, since the construction of the robot does not guarantee precise values. Thus, we need to check the cameras roll, pitch and yaw angles relative to the head, which are fixed angles that result from the robot assembly process and will add to the correspondent angles estimated from the kinematics analysis.

To help with the estimation of these parameters and to visually confirm both the parameters and the results of the presented estimations, a visual tool was created.

The tool allows us to define and test a set of parameters necessary for estimating the projection of a pixel on the ground. The parameters include the used camera intrinsic parameters *pixel size* and *focal length*, as well as the possibility of defining the pixel coordinates of the focal center on the CCD, which can be applied as an offset to all pixel coordinates in case the camera lens focal point is not aligned with the center of the CCD. The other definable parameters are the roll pitch and yaw angles of the camera on the head which will always be added to the correspondent estimated angles.

The application can run from a live feed from both the robot camera and the robot position sensors, which allows us to verify the kinematic model estimations when using static well controlled scenarios.

In order to allow the application users to easily confirm the effects of the parameterization, we solved expressions 5.6 and 5.8 in relation to $pixel_n$ and $pixel_m$, so that we can project any point with metric coordinates over the image, in pixel coordinates.

The developed tool was a need that came up when starting to extract the pixel ground projection information. The main need was initially to help the development and eventual corrections of the expressions, based on the knowledge of what the grid should be like over the image of a known environment. It is also a tool to try and test the angular parameters of the camera, also allowing to test the intrinsic parameters focal length, pixel size and CCD center.

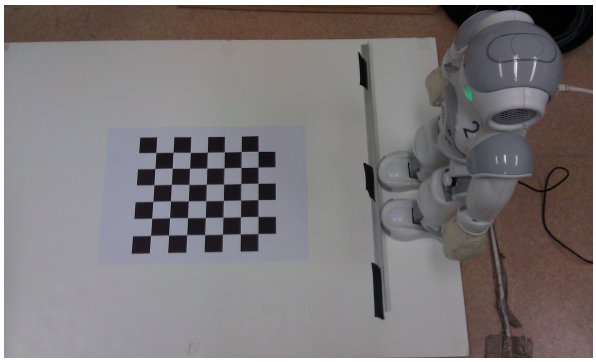
Figure 5.8 presents some screen shots of the application with a grid of several points on the ground plane drawn on the pixel coordinates estimated according the inverse of the projection model. In the example, the squares of the chessboard have a 36 mm side and the bottom line is at 311 mm from the robot center projection on the ground, manually measured with tape. The robot is placed in such a way that it is parallel to the chessboard. The "virtual" grid is also configured with a 36 mm side.

This tool and setup allows us to determine the parameters already described for each of the cameras of each of the robots. It is necessary to calibrate all of them independently because, by construction, the camera placement is not physically guaranteed to be the same. Also, when looking at the estimations, one must keep in mind that the extraction of body pose is also subject to error. This means that a set of parameters estimated for a robot's camera may become ineffective as the robot joints may become loosen and readings less accurate.

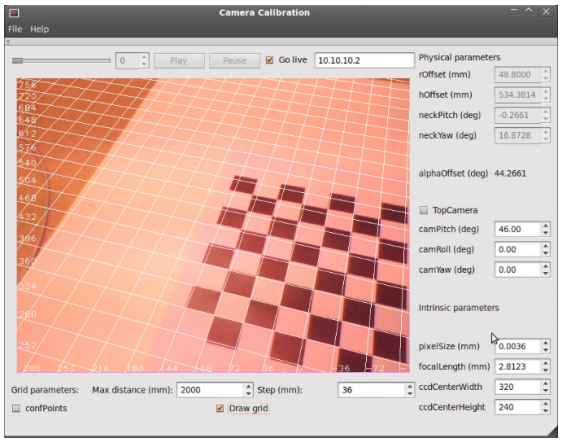
Details on how to extract the camera CCD pose at a given instant are described later in section 5.2.2.

5.2.2 Camera pose extraction

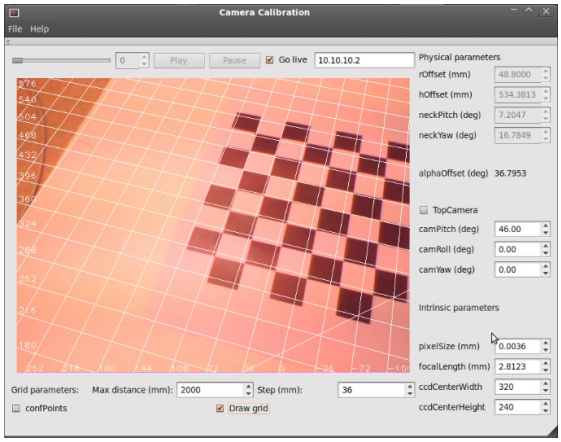
One of the requisites for projecting the pixel coordinates into the world ground plane is to know the pose of the camera relative to the ground. To estimate this pose, we make use of the kinematic model of the robot and the inertial unit measurements.



a)



b)



c)

Figure 5.8: In **a)**, the setup with known measures. In **b)** and **c)**, a 36mm grid projected on the image, with some pitch and yaw applied to the robot head.

The main assumption made for this analysis is that the robot has at least one foot in full contact with the ground. This is assumed because the current walking engines guarantee that and, when the robot is off balance, the decision layer will trigger a series of routines to manage the fall and the posterior get up. These procedures have priority over any other, since a fallen robot will not be useful and thus, during their execution, the information provided by the camera is ignored until the robot is up again, fulfilling the requisite of having a foot on the ground whenever vision information is being used.

Every agent cycle, when updating the body model, the camera pose data necessary for projecting the coordinates are estimated. The values needed are the height, the yaw angle of the neck and the general orientation of the camera CCD (angles over XX, YY and ZZ) relative to the ground plane. Also, a timestamp of the instant of each specific pose is kept, for further use.

Throughout this section, XX axes will be represented with red tones, YY axes with green and ZZ axes with blue. The definition of the angles will be according to the most standard

definition of roll, pitch and yaw being rotations around XX, YY and ZZ, respectively.

5.2.2.1 CCD angles extraction

Since we have the pitch and roll angle of the torso relative to gravity, thanks to the inertial unit, we use that reference and the analysis of the kinematic model to extract the CCD angles relative to the ground. The estimation is not direct and takes a series of steps to accomplish, being the first ones to extract the camera homogeneous transformation relative to the vertical axis defined by gravity.

The example in figure 5.9 has the robot torso with some pitch, which is measured by the inertial unit, and the head has some pitch and yaw around the neck moving point. The small axis system on the surface of the head is the camera referential. All the axis in this section use RGB color code to XYZ respectively and additionally the black and pink axis are the reference vertical (gravity) axis and robot front axis respectively.

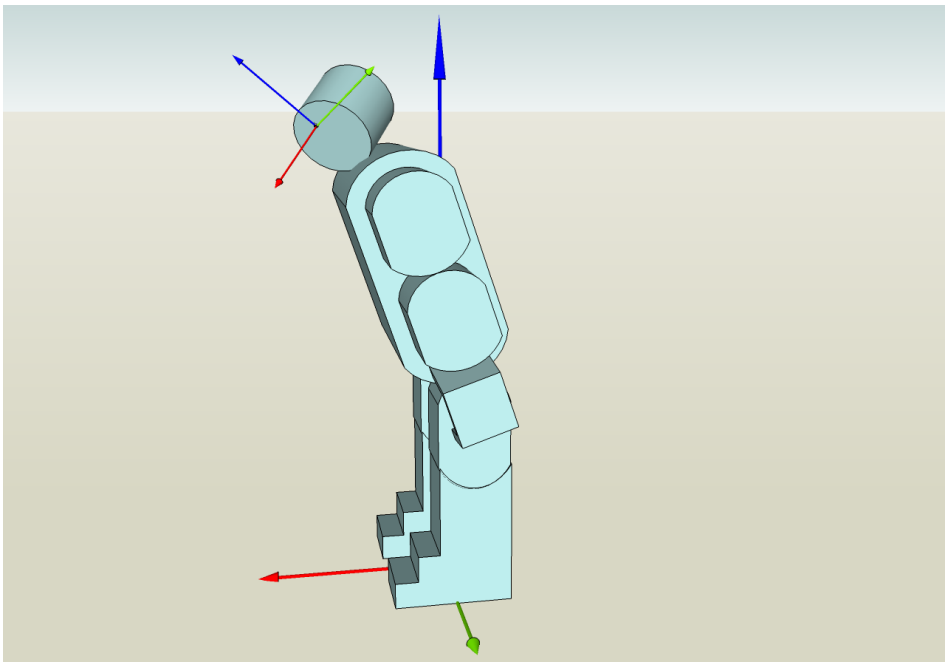


Figure 5.9: Diagram of the camera placement on the robot with correspondent general axes.

- Based on the homogeneous transformation of the head relative to the torso (kinematic model calculated simply by the head joints values) and of the torso relative to the vertical (roll and pitch angles given by inertial unit), we get the head reference relative to the gravity.
- Applying the camera configured position and angles on the head, we get the camera reference relative to the vertical gravitational axis. The homogeneous transformation

obtained has a base representation of the orientation of the CCD which we explore further to extract each individual angle relative to the ground.

We then need to, step by step, estimate each individual angle, measured by using the inner product between the CCD axes and two reference axes, the gravity axis, $\overrightarrow{refZZ}$ (which is a unit vector over ZZ axis), and the robot front axis, $\overrightarrow{refXX}$. The first angle we estimate is the camera CCD pitch angle:

- With the camera reference transformation based on the gravity previously estimated, and since we consider the XX axis the focal axis of the camera, we can get a reference for the XX of the CCD by applying the camera transformation based on gravity to the unit vector XX (figure 5.10):

$$\overrightarrow{xCCD} = [R] * \overrightarrow{refXX}$$

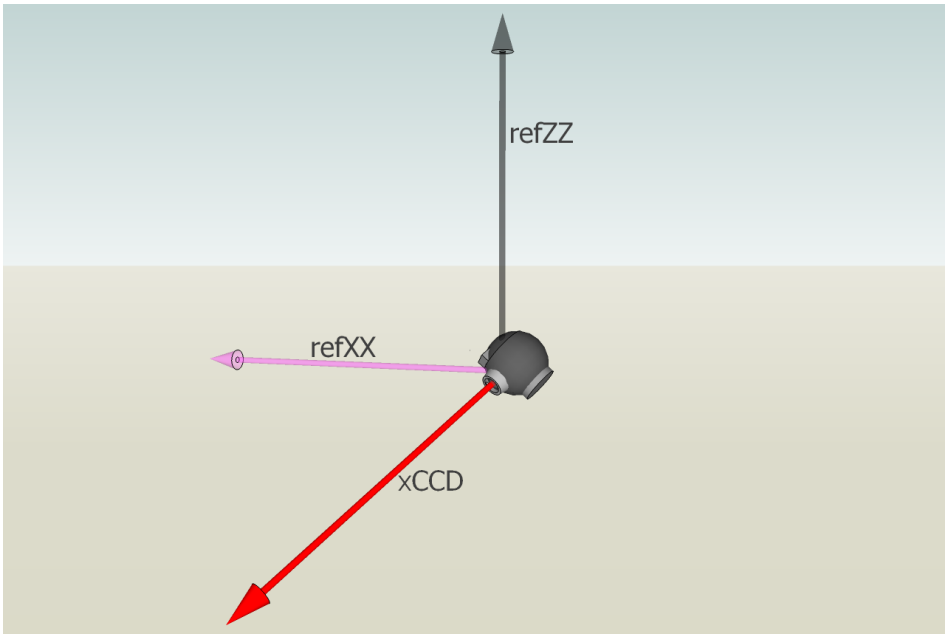


Figure 5.10: Diagram of the first step to estimate the pitch angle of the CCD.

- The next step is to get a YY vector perpendicular to the plane formed by $\overrightarrow{xCCD}$ and $\overrightarrow{refZZ}$ and parallel to the ground (figure 5.11):

$$\overrightarrow{yVertCCD} = \overrightarrow{refZZ} \times \overrightarrow{xCCD}$$

- Finally, given the XX vector of the CCD, $\overrightarrow{xCCD}$, and the YY vector on the CCD that is parallel to the ground, $\overrightarrow{yVertCCD}$, we can estimate a ZZ vector on the CCD alignment,

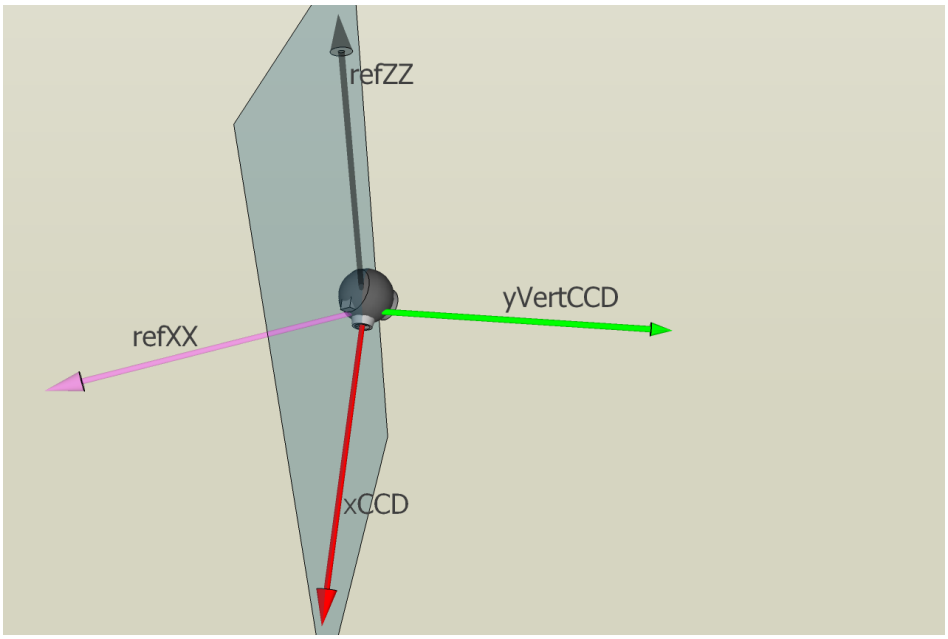


Figure 5.11: Diagram of the second step to estimate the pitch angle of the CCD.

$\overrightarrow{zVertCCD}$. We finally can estimate the pitch angle of the CCD relative to the ground by measuring the angle between the vertical ZZ vector, $\overrightarrow{refZ}$, and this last ZZ CCD reference, $\overrightarrow{zVertCCD}$ (figure 5.12):

$$\overrightarrow{zVertCCD} = \overrightarrow{xCCD} \times \overrightarrow{yVertCCD}$$
$$pitchCCD = acos(\overrightarrow{zVertCCD} \cdot \overrightarrow{refZZ})$$

To get the roll angle, we can get the ZZ vector of the CCD, $\overrightarrow{zCCD}$, aligned with the top of the CCD, by applying the transformation of the camera relative to the gravity to the unit ZZ vector, $\overrightarrow{refZZ}$. The roll angle can be estimated by measuring the angle between the vertical axis, $\overrightarrow{refZZ}$ and the CCD ZZ axis, $\overrightarrow{zCCD}$ (figure 5.13):

$$\overrightarrow{zCCD} = [R] * \overrightarrow{refZZ}$$
$$rollCCD = acos(\overrightarrow{zVertCCD} \cdot \overrightarrow{zCCD})$$

For measuring the yaw, we first extract an XX CCD axis, $\overrightarrow{xVertRot}$, that is parallel to the ground, using the $\overrightarrow{yVertCCD}$ and the $\overrightarrow{refZZ}$. The yaw angle is the angle measured between the XX axis parallel to the ground, $\overrightarrow{xVertRot}$ and the reference XX unit vector,

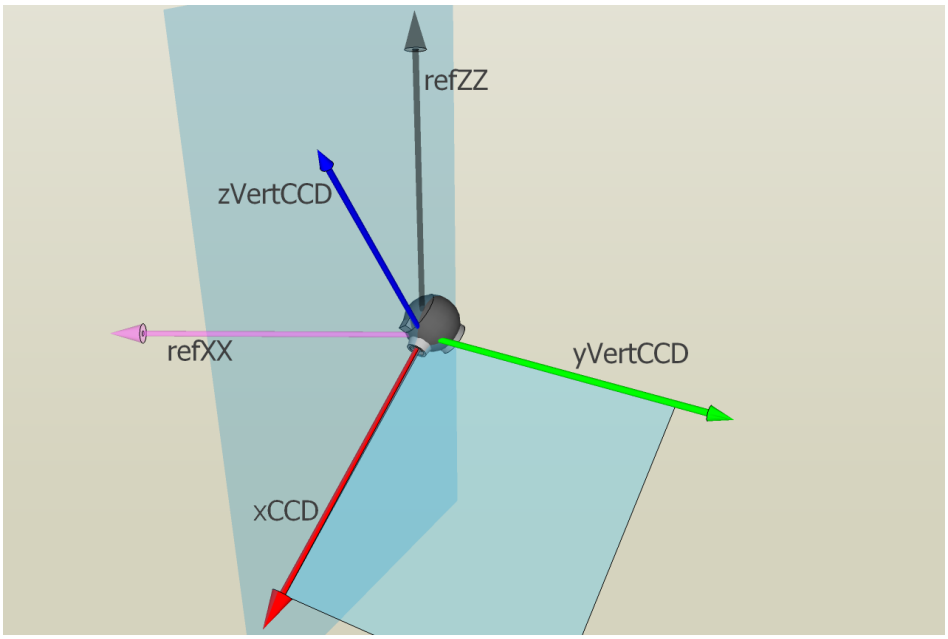


Figure 5.12: Diagram of the third step to estimate the pitch angle of the CCD.

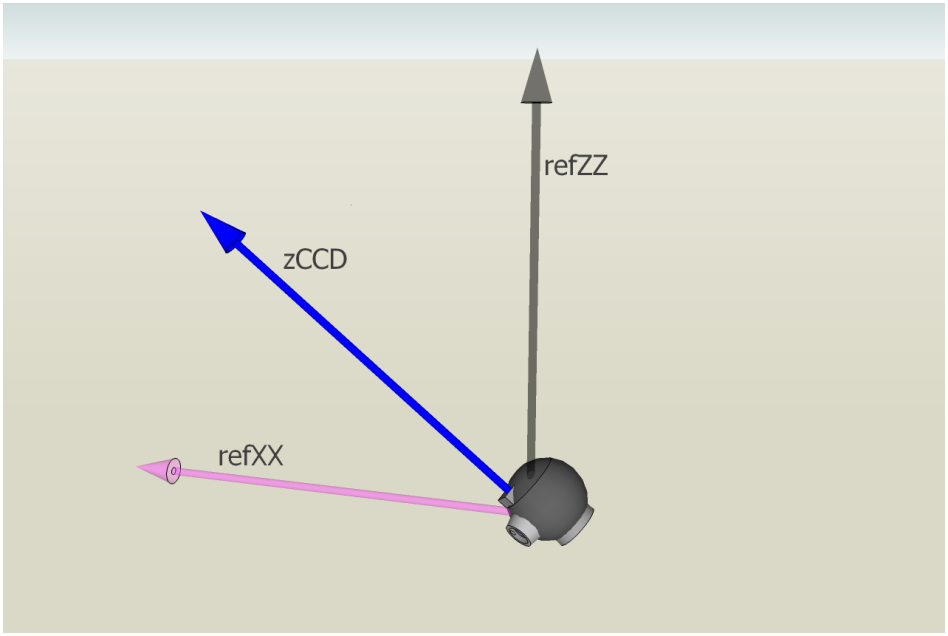


Figure 5.13: Diagram of the estimation of the roll angle of the CCD.

$\overrightarrow{refXX}$ (figure 5.14):

$$\overrightarrow{xVertRot} = \overrightarrow{yVertCCD} \times \overrightarrow{refZZ}$$
$$yawCCD = acos(\overrightarrow{xVertRot} \cdot \overrightarrow{refXX})$$

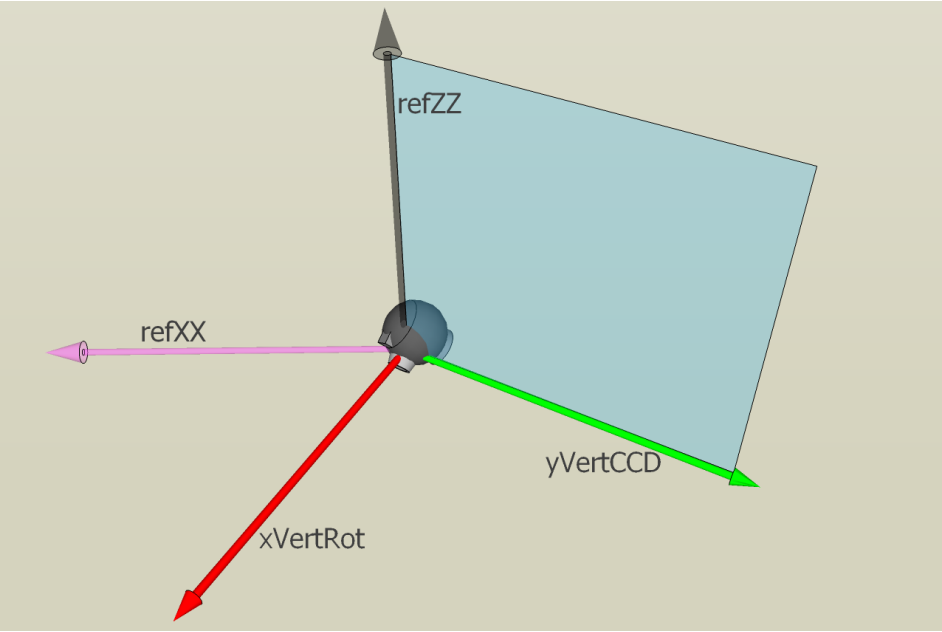


Figure 5.14: Diagram of the estimation of the yaw angle of the CCD.

5.2.2.2 Camera pose matching

As stated before, in section 5.1, the agent process that extracts the camera pose is not synchronous with the vision process that evaluates the visible artifacts. The vision process is configured to use the camera at 15 frames per second, which would theoretically result in cycles of 66 ms. However, and since this process is computationally heavy and due to the fact that it runs in best effort scheduling on the NAO single processor, the average cycle times are larger than the theoretical 66 ms. The vision process time was measured and the obtained mean cycle time for the process is 137.8 ± 68.2 ms. Moreover, the agent process runs with a high priority scheduler and has a cycle time of 10 ms.

These characteristics of the agent and vision processes imply that we have measurements of the camera pose, explained in the previous section 5.2.2.1, each 10 ms. On the other hand, the visual information in pixel coordinates that needs to be transformed to metric coordinates relative to the robot center projection on the ground is only available at a rate more than ten times lower than the camera pose estimations.

This means that, when new information is available from the vision process, the camera pose on the current agent cycle is many cycles ahead of the camera pose at the moment of the frame grabbing. Due to this lack of synchronization, the pixel coordinates of all the objects detected by the vision process that are based on that frame, the tendency is that the transformation of those coordinates into the metric coordinates will not produce truthful results, unless the robot has not moved and the body pose is the same as more than 100 ms

before.

This phenomena led us to create a history of the camera pose data alongside a timestamp that can allow us to "synchronize" the pose with the visual information (algorithm 10).

The vision process cycle starts by pinpointing the time of image capture by the camera in a timestamp just after the frame grabbing which, regardless of the time taken to analyze the image, is the most approximate time of the capture moment.

This timestamp is compared with the timestamps of each of the camera poses in the history in order to discover which pose the camera had at the time that the image was grabbed.

Algorithm 10 Algorithm used to choose the camera pose correspondent to the moment of the last image captured and processed by the vision process.

Input: camPoseList → list of the last N camera poses
vinfo → visual information which includes the frame capture timestamp
Output: synchronizedPose → camera pose closer the the image capture time

```
1: minTimeLapse = 10000 ms
2: poseIdx = -1
3: currentCamera = projectionModel.getActiveCamera()
4: for t = 0 to camPoseList(currentCamera).size() do
5:    currTimeLaspe = camPoseList(currentCamera)[t].timeStamp - vinfo.timeStamp
6:    if abs(currTimeLapse) < minTimeLapse  then
7:       poseIdx = t
8:       minTimeLapse = abs(currTimeLapse)
9:    end if
10: end for
11: synchronizedPose = camPoseList(currentCamera)[minTimeLapse]
```

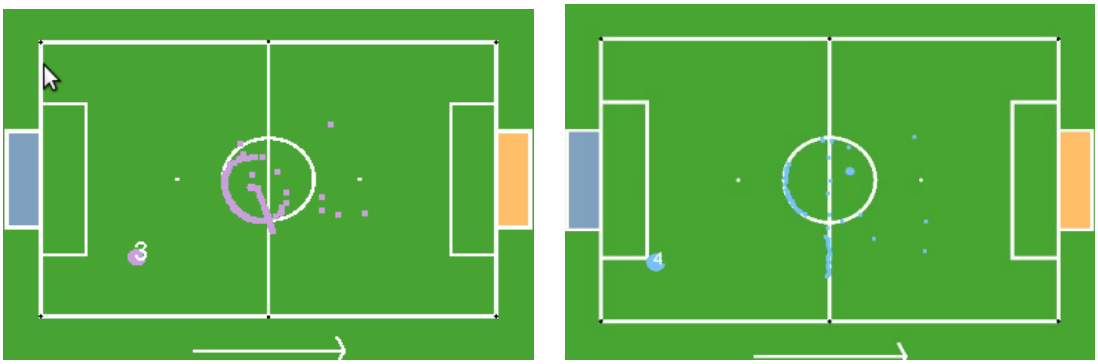
In figure 5.15 we can visualize the projection of visible white points over the center circle of the soccer field when directly applying the projection expressions presented in section 5.2.1 over the theoretical camera placement on the robot head and when using the correction estimated both through the use of the camera calibration method presented in 5.2.1.2 and camera pose extraction and synchronization methods presented in section 5.2.2.

5.2.3 Localization

Providing the NAO robots with a localization method was a need that became obvious from the moment that the field goals changed from multicolored to single colored, i. e. they ceased to be one blue and one yellow to become two yellow goals.

Given the experience with the CAMBADA MSL team robots using the adapted tribots localization algorithm, and considering the objective of sharing methodologies, the same algorithm is used in the NAO robots.

To use the algorithm on the NAOs, however, and given their reduced field of view, some aspects have to be considered.



a) **b)**

Figure 5.15: **a)** the projection of points over the center circle using the projection without camera angle corrections. **b)** the projection of a similar set of points now using the correction of the camera angles extracted through camera calibration tool.

We know from experience with the algorithm that, whilst not requiring a huge amount of points to work, it does require the points to be representative of the surroundings. The most recurrent example is a scenario where the robot can detect hundreds of points over a single line of the field (figure 5.16a): despite having a huge amount of points to feed the algorithm, they are not representative of any of the field regions. However, if the robot could see some tens of points over a set of representative lines (figure 5.16b), these few points would be enough for the algorithm to provide a good estimation of the pose.

In the CAMBADA robots, this scenario is not a problem due to the fact that the robot has a reliable omni directional vision system with a relatively high range. Wherever it is on the field, the vision is capable of detecting enough white points over at least two distinct lines for the localization algorithm to work. The decision layer does not take into account localization needs when planning the robot movements.

However, in the NAO humanoid robots, the scenario is very different. These robots have a very directional vision system with a very limited range. A NAO robot is easily in a situation where it is facing a side line, thus falling in the scenario presented before where it would see points over a single line.

Also, there is generally little amount of information during most of the cycles. In section 4.4, it was explained that the error minimization task of the algorithm is performed every cycle effectively because there are always good points over the white lines available and the prior pose was known and close to the current pose, which allows the algorithm to rapidly converge. It was also explained that, at the beginning, the algorithm performs the initial localization by exhaustively testing many poses and choosing the one with less error.

In the NAO robots, we opted not to perform the minimization task every cycle, because there is no significant visual information available every cycle, since the robot's vision is very limited both in terms of range and field of view and thus the results obtained by the algorithm

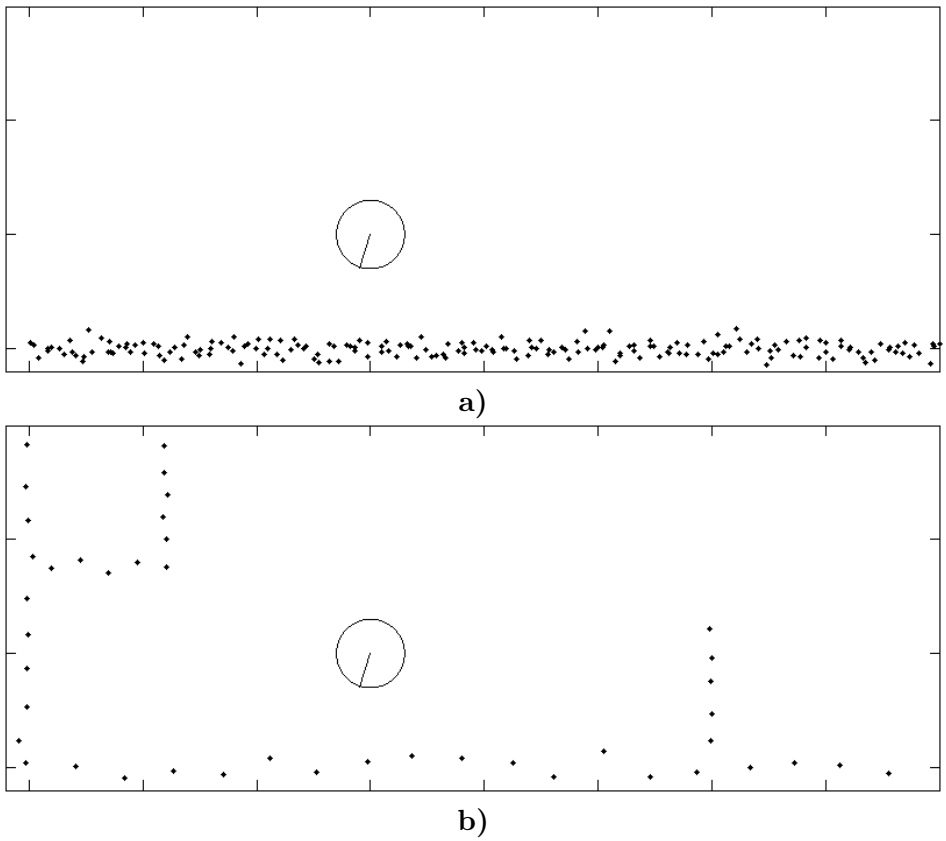


Figure 5.16: Example of two sets of points to use in the localization algorithm. In **a)**, despite having 200 visible points, the robot pose could not be well estimated. In **b)**,exactly in the same pose and with only 44 points visible, the localization estimation would be quite precise, since those points are scattered over several lines of the field.

would degrade due to the lack of information.

This means that in this league, our approach to localization needs to be more pro-active, meaning that the robot needs to actively search for points over white lines in order to obtain information to use in the localization algorithm.

Thus, the idea for applying the localization algorithm in the NAO robots is to perform the error minimization task with a relatively low period, but trying to guarantee that the set of points over the lines are significant and can provide a good estimation. The convergence of the process is quite slow, both because the processing unit is slow and also because the prior pose will be outdated and farther from the current pose.

As presented in section 5.1, the localization runs as a separate process. This process is controlled by the agent process, which is the one that decides when to localize. The two processes make use of a set of values on the RTDB to communicate and synchronize. Whenever the agent decides to localize, it prepares the necessary visual information, writes it on the RTDB and signals the localizer process to start the localization algorithm using those

data, by activating a flag also through the RTBD.

The localizer process is periodically checking the starting flag filled by the agent to know if it is time for it to execute the algorithm. The current period is one second. When the flag is active, it proceeds to read all the information required to run. After estimating the robot pose from the data set by the agent, the localizer process writes back the results on the RTDB, making them available for the agent to retrieve and fills an ending flag for the agent to know that the localizer process has finished the task.

On the side of the agent process, when it needs to take a localization action, there are several constrained tasks that need to be executed step by step:

- First, a localization action needs to be triggered, by either a role decision or as a priority task triggered by some event, like achieving a low threshold of confidence on the current pose. Whichever entity starts the process, is responsible for waiting for its conclusion before continuing.

- When the localization action is triggered, the robot is stopped and stands still to perform the next steps.

- The localization execution on the agent side moves the head to look in several directions, covering a very big angular range around the robot. Given the maximum turning angle of the head and the angular opening of the camera lens, we can cover everything but the 90^{o} centered on the robot's back.

- At each of the configured head positions, the agent takes the points over white lines given by the vision process, converts them to metric coordinates using the projection model and gathers them. When all the head positions have been covered, all the points are made available to the localizer process and the flag to start the estimation of the pose is activated.

- The agent then waits until it is flagged back by the localizer process, meaning that it has finished the processing and new information about the robot pose is available. When the agent gets the information, it updates the player pose and considers that the localization action is finished.

Obviously, the whole process must be performed when the robot is standing still so that the points of all the perspectives correspond to the same pose on the field.

Figure 5.17 presents two sets of points perceived by a robot with two different head angles. The perceived points are represented in coordinates relative to the center of the robot projection on the ground.

These two views, taken and merged through the described process, would result in a single overlapped set of points with more complete information (figure 5.18) to feed the localization

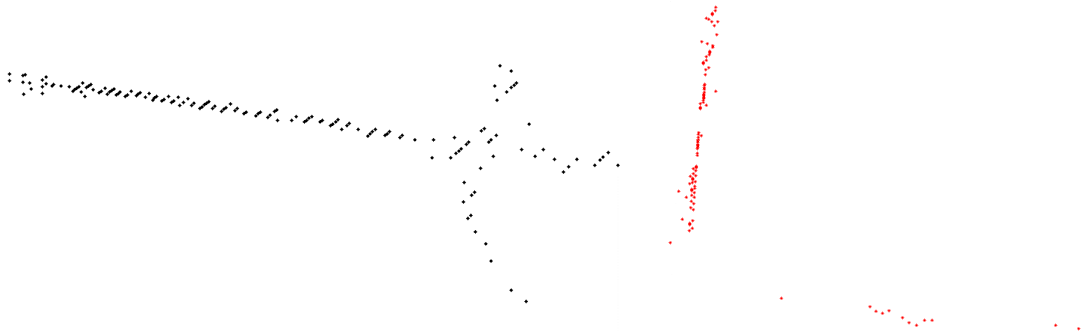


Figure 5.17: Two sets of points detected by a robot with two different head angles while "looking around". These representations are independently placed side by side.

algorithm. The separate sets of points are merged through matching of each visual set with the pose of the robot at each given set.

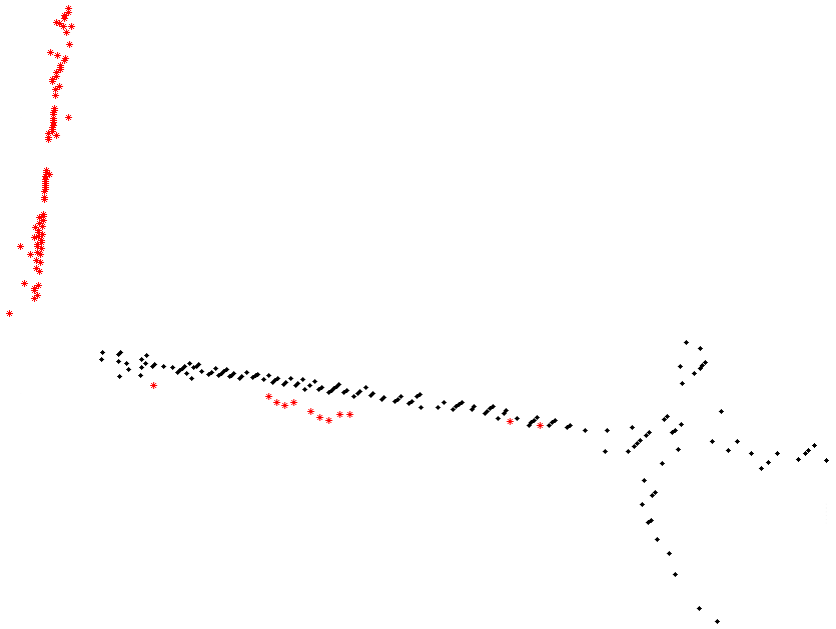


Figure 5.18: The two sets of points of figure 5.17 merged into a single set after the transformations according to the correspondent poses.

Another example of merging of points on two distinct view angles is presented in figure 5.19, where the robot has localized itself based on those points. In this particular case, the estimated pose is affected with some error, since the perceived points are distorted.

The example of the projected points presented in figure 5.15 is also exemplifying the results of the localization algorithm, where the robot was placed on the goal area corner facing the

center of the field. In figure 5.15a, with badly perceived points, the robot has localized in a wrong position, while in figure 5.15b, with good points, the localization algorithm provided the real pose of the robot.

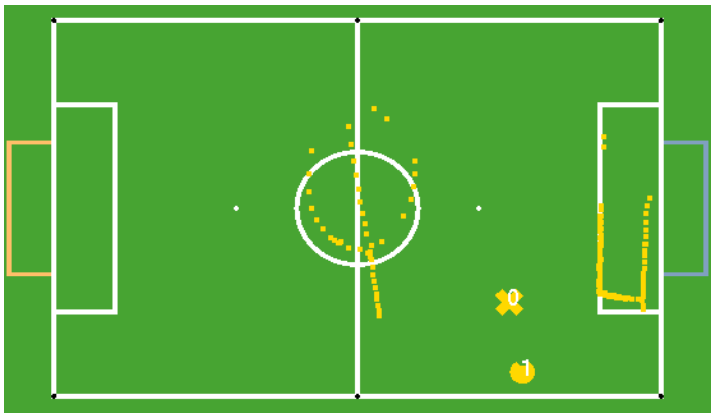


Figure 5.19: Two sets of points in a single representation from a live run of a robot.

Finally, in figure 5.20 screenshots from the video of a trial run are presented. The robot was placed over a side line of the field, facing into the field and to the left of the field center line (figure 5.20a). The initial belief of the robot pose is simply a pose with position and orientation set to zero, which corresponds to the middle of the field. The visible points over the center field line before the robot localizes itself are (figure 5.20b) are obviously misplaced on the field, since they are represented relative to the robot and the robot is not in the correct pose. After finishing the localization procedure, the robot pose is the correct one and the matching of the visible points with the real center line is clear (figure 5.20c).

5.3 Summary

The work achieved during this PhD over the Standard Platform League robotic team appears on a starting phase of development, where several modules were functional but the team architecture was still not well defined. Thus, one of the objectives achieved during the work was the definition of a software architecture for the SPL team that allows the development of software to be modular and organized enough for the team developers to focus on specific modules that can be easily introduced or updated in the code structure.

The vision of the NAO robots of this team was built to detect the objects of interest on the image and provide pixel coordinates of each of the points of interest. Early approaches of the agent would simply try to "follow" the ball on the image, by moving the robot and its head in an attempt to center the ball on the image. For the robots to be able to play soccer, they need to perceive more than pixel coordinates and thus a projection model was designed to allow us to transform the pixel coordinates provided by the vision into metric coordinates

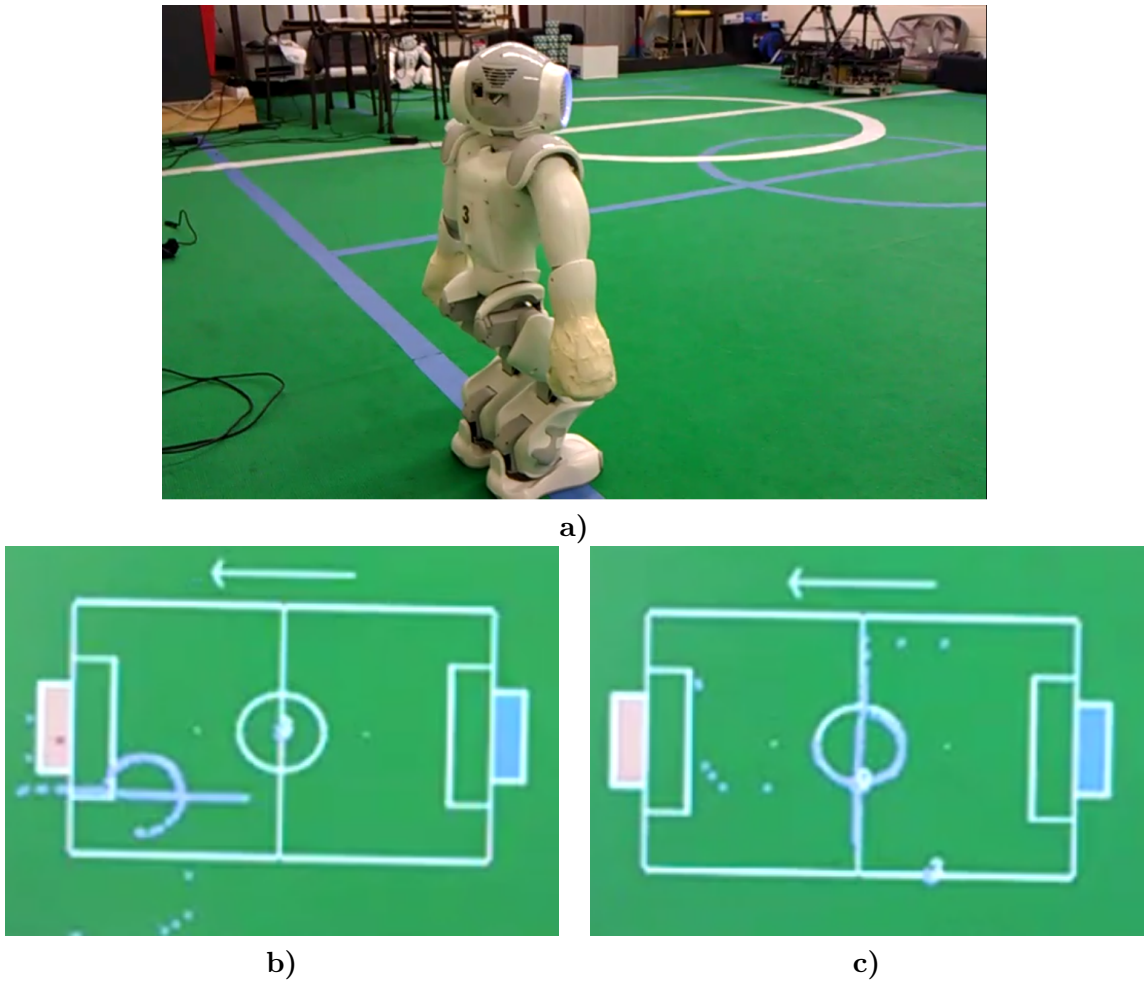


Figure 5.20: Field trial of the localization algorithm. In **a)** the real robot place on the field. The field used by the NAOs is the one marked with blue lines. In **b)** the representation of the visible points over the field center line and circle, represented relative to robot pose before localization. In **c)** the representation of the same points after a successful localization.

centered on the robot ground projection, so that all the detected points (ball, lines, goals, other robots) can be part of a world model represented in meters.

With the representation of the points in metric units, localization of the robot on the field can be performed, so that the team robots can cooperate according to a given strategy and play soccer more effectively. To achieve localization, the tribots localization algorithm was used. Due to the restrictions imposed by the robot platform, the use of the algorithm is different from the use in the CAMBADA team and some adaptations were necessary, specially concerning the algorithm data input, which needs to be carefully prepared. For gathering points over the white lines to achieve a successful localization, a localization procedure was created and was described with examples.

Chapter 6

Tools

While developing within projects such as the CAMBADA or Portuguese teams, we always need to have tools to work with, either for manual control and test, debug, monitor, development, among other uses. In this chapter, some of the tools used in both scenarios are presented, particularly tools that were created or enhanced during the development of this work because there was a need for them at some point.

6.1 Tools for MSL

In this section we present some of the tools used within the CAMBADA project that were either created or altered during this work.

6.1.1 Basestation

The basestation graphical tool is the main team interface for configuration, debug, visualization and control. It is a fundamental piece that provides a visualization of the data present on the shared memory, corresponding to all the agents of the team. During this PhD, it has been enhanced with new functionalities to provide even clearer information to the users both during development and competition.

6.1.1.1 The add-ons

During a typical game, a lot of set piece situations occur where the robots need to perform a synchronized team action. In situations like kickoff, throw in, goal kick, free kick or corner kick, the team needs to execute a cooperative play to put the ball back in game. The play may include one or several passes between two robots. The two robots involved in a pass on a set piece assume specific roles. The one that will first touch the ball and put it in play after the referee whistle is called the *replacer*. Any of the other robots that are potential receivers of the pass are called exactly that, *receiver*. To achieve synchronization on this task,

the robots use a set of coordination flags to signal their state during the several stages of a set piece, including signaling that they are OK to receive a pass or not according to the field occupancy by the opponent robots.

For some time, when an anomalous situation occurred in a set piece during a game, we could only guess what went wrong. A clear example was a replacer making a pass to a robot that did not possess a pass line instead of passing to one that clearly had a free pass line.

One of the upgrades implemented on the basestation was to visually represent the coordination flags during set pieces. The visual indicators are a set of dashed lines with a green or red color code from each of the possible receivers to the ball, indicating if they evaluate a free or occupied pass line (figure 6.1a). After the decision is taken, a solid line is represented to show the final pass decision (figure 6.1b).

With this upgrade, we can easily understand if the problem is in the receivers that evaluated wrongly the status of the pass line or if the replacer is the one that made a wrong decision.

Of course that, during a game, we can instantly detect some problems and even write them down for further analysis, but there are some details that easily go unnoticed in real time visualization and easily forgotten after a few days, when the programmers can finally sit down to evaluate the problems. In order to tackle this problem, we have built a logging protocol and tool to save the information of the several agents present on the RTDB (as perceived by the basestation agent) so that we could later analyze the general game situations. The information of the RTDB is saved to the log every 100 ms.

However, the tool is manually managed and the logging needs to be started and stopped (at the beginning and end of the game, or even paused on half time) and also saved to file by the users at the end, since it is kept in memory during the capture. Unfortunately, during competition we verified that the starting, stopping and saving of logging failed too many times and some important logs were lost.

Thus, an auto logging feature was added to the basestation, which is completely independent from the manual logging tool.

The idea is to create a history of the information, perceived for the last hour (approximately), that is kept in memory, in a circular buffer approach that, after reaching the buffer limit, removes the oldest information to input the newest. By keeping this history of the last hour in memory, it allows the users to quickly analyze immediate situations during development, since the last events up to one hour are immediately available in the log playing mode of the basestation (figure 6.2), and then seamlessly continue with the tests. This is achieved by changing the source of information to fill the basestation from the direct RTDB readings to the data array of information kept in memory.

The information kept in memory is periodically flushed out to temporary files of approximately one minute of events, also using a circular buffer approach by replacing the older

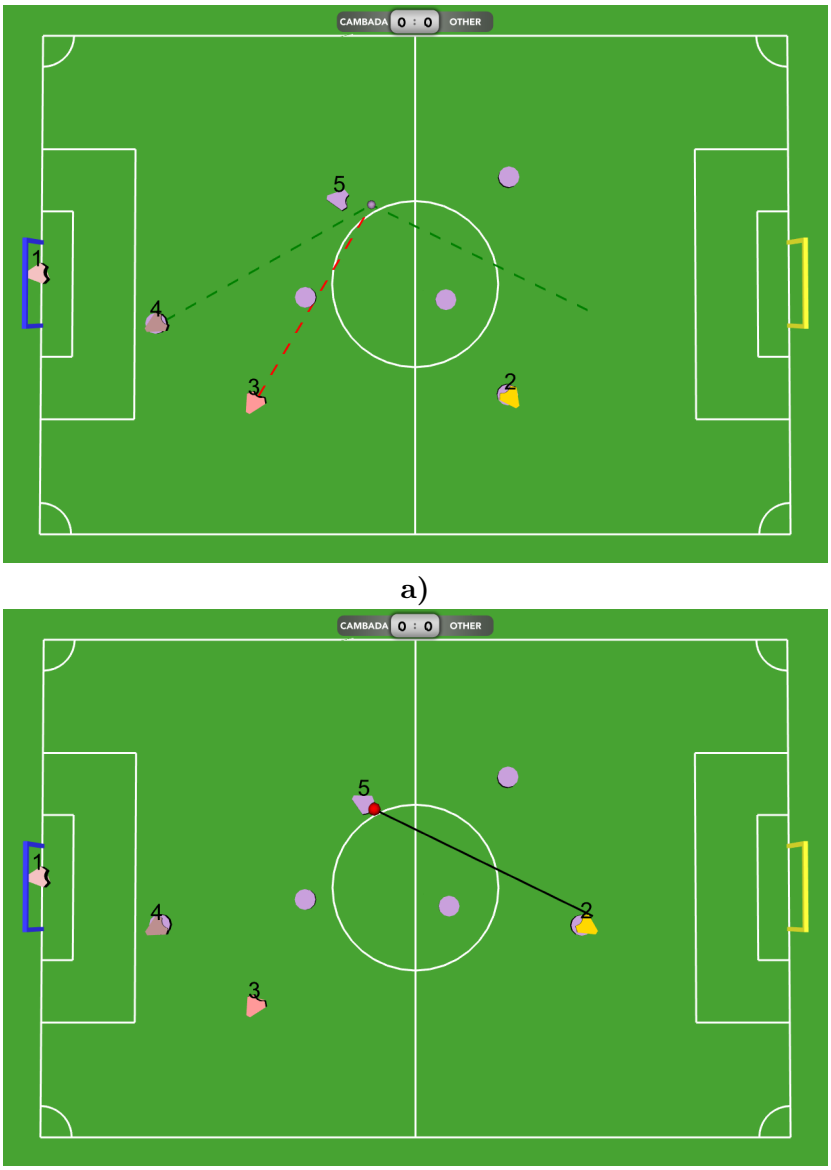


Figure 6.1: Basestation pass visualization. In **a)** the evaluation of the pass lines to each receiver. Robot 3 signals that it cannot find a free line of pass, robot 4 signals a direct line of pass and robot 2 signals that it has a free line of pass to a nearby position that it will use to get away from the opponents. In **b)** the chosen action is represented, in this case robot 2 is just getting to the arranged position as robot 5 is performing the pass.

information. These files are not readable by the log player, since they do not possess the initializers and terminators of the log format, but can be used to manually build a log file in case we need the log of a run where the basestation crashed for some reason. When the basestation is closed normally, a single complete log file is created through the concatenation of all the temporary files.

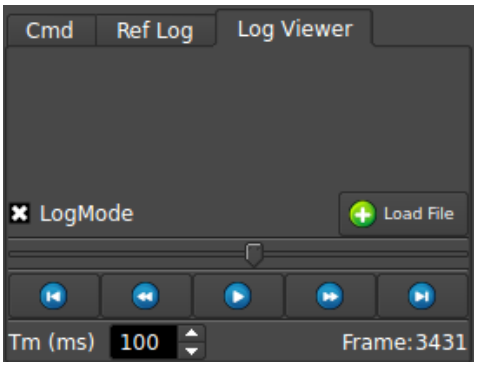


Figure 6.2: The log player widget of the basestation, which allows to playback a log from a file or, by default, from the last events up to one hour. It can be easily played frame by frame (100 ms is the time between frames) to detect small state changes.

This methodology allows the team to have a backup of a game log in case the proceedings with the logging tool failed. The choice of keeping the information for the last hour is precisely to guarantee that a whole game is covered, since the 30 minutes game time plus 10 minutes of half-time plus delays are normally under one hour.

6.1.2 Cursor graphical user interface

One of the tools used for testing and calibration of the robot hardware platform, both in terms of wheel movement and kicker test and manual calibration, is our *cursor* application.

This application allows direct control over the robot platform and direct access to the robot body status information. To run the *cursor* the only need is to launch also the *hwcomm* process to send and receive messages from the micro controller network.

The functionalities of the cursor include sending movement orders for the motors, in the form of a linear velocity with component over XX and YY and an angular velocity. It allows also to send kicking and passing orders, with manually defined power values and turning the grabber on and off.

On the other hand, the *cursor* allows us to visualize the information provided by the platform, namely the batteries statuses, the values of the orientation sensor, the values of the odometry in each component XX, YY and angular, the power of the kicking device and its status on charging and readiness and the value of what we called the *barrier*, which detects if something is in the front gap of the robot.

6.1.2.1 The new interface

The *cursor* application was previously a console interface driven by key press, meaning that any information would only be displayed when a given key was pressed. This interface became obsolete when the need to constantly verify some sensors values came up.

The application was given a graphical user interface that now displays all the relevant sensor information, as well as the currently defined velocities (figure 6.3). Several new features were added when the robot platform changed, since it has new sensors that need their values displayed. Some of the main features are used for robot configuration very frequently, such as each time we arrive at a new field of play or whenever there is an intervention to the robot platform for repairing or upgrading the micro controllers software.

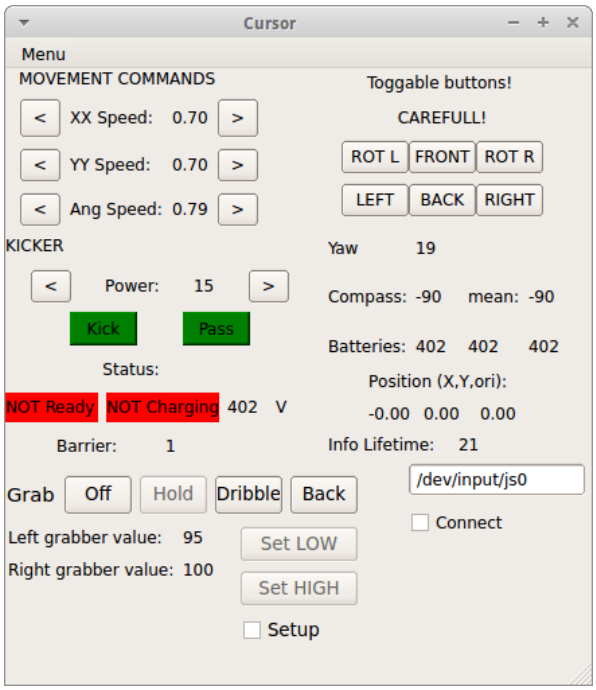


Figure 6.3: The cursor graphical interface

- When manually configuring or testing the kick device, the kicking action must be taken only when the capacitors are already charged, which is our reference. With a color code, the user can easily verify when it is OK to make the kick and read the voltage power of the kick capacitors. The interface keeps and displays the value of kick power to be used and two buttons/actions are available: kick and pass. Since a kicking action is only possible when the ball is engaged on the robot front gap (this constraint is directly in the kicking device micro controller for security reasons), the kick and pass actions are only available when the barrier is active.

- While the previous robots possessed an infra red sensor on the ball gap in its front that allowed to know if the ball was engaged (or at least if an object was there), the new robots do not possess that sensor. The grabbing system was greatly enhanced and is now composed by two motors with Swedish wheels that pull the ball towards the robot. These motors are mounted on rotation axis so that their height changes when the ball

is placed inside the robot front gap, becoming higher as the ball is more inside the gap. The motors positions are now the feedback of the grabbing system, which means that the *barrier* is now active when both motor are above 75% of their total course to guarantee that the ball is being hold. The raw values of each o the motors of the grabbing system were also added to the *cursor*. The interface has a set of buttons to activate the grabber motors in one of the several existent modes and continuously displays the position of each motor. Furthermore, the grabbing system needs to be calibrated to define the low and high position of each motor. The low position is calibrated when the motors are in rest and the high position is calibrated when the ball is completely inside the mouth. A protocol was defined so that these two values can be sent to the grabber micro controller and the cursor interface setup buttons to perform that task.

- The yaw, compass and mean values are other frequently used features, for verification and calibration of the magnetic reference. The reference used is one of the goals of the field and the process of getting the necessary value is to manually (using the *cursor*) move the robot around the field while keeping it always directly toward the goal line of the chosen side. The mean value of the compass is used to read the angle of the chosen side to later include it on the configuration file. During the process, we verify that the angle coherent throughout the field. If that is not the case, we probably will not be able to use the compass values.

6.1.3 Video editor

Given the need for development on all the areas of the soccer team, our vision system allows the recording of videos from a robot camera feed for offline analysis and development. When developing the front vision with the perspective camera, presented in section 4.3.3, and for validation of the algorithms, we felt the need to edit the saved videos and cut specific pieces of it to, for example, have a video where the ball was present in every frame.

A new tool was created for that purpose, with a simple interface that allows to "bookmark" the desired initial frame and end frame, play the result and save it to a new video (figure 6.4).

6.1.4 Laser range lob analyzer

Over the years, the kicking device of the robots has been configured individually for each robot with a methodology of kicking from several setpoint distances with different kick powers and then matching a polynomial function to estimate the necessary kick power according to the distance to the goal.

Particularly in the old platform, we always verified that there were differences between robots and also differences within the same robot. However, visually registering the point where the ball hit the ground was the only way to try to evaluate the ball trajectory.

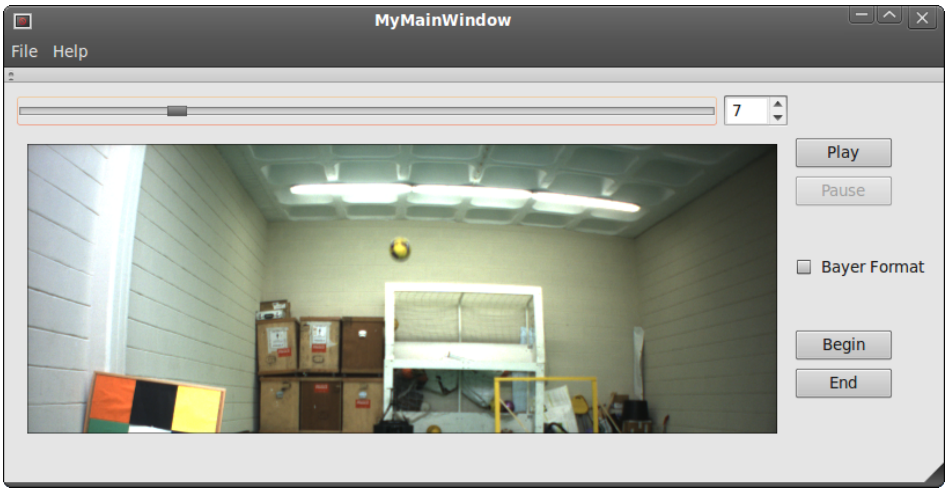


Figure 6.4: Screenshot of the video editing tool.

At some point, we felt the need to have some precise tool to analyze the ball trajectory so a battery of tests could be created in order to try to, one by one, isolate the factors that could provoke such differences. When a laser range finder became available, the idea to use it to get precise measurements of the ball immediately came up.

This tool represents the data from a laser range finder sensor in Cartesian coordinates and is used to extract the ball trajectory. The plane of the lob shot and the laser plane need to be aligned, which is achieved by positioning a robot and a laser in a straight line and the ball is shot on that plane (figure 6.5). The visual application starts by gathering the points for a given number of cycles and extracts a "background" based on them. This implies that, before kicking the ball, we need to keep the kicking plane clean of moving objects.

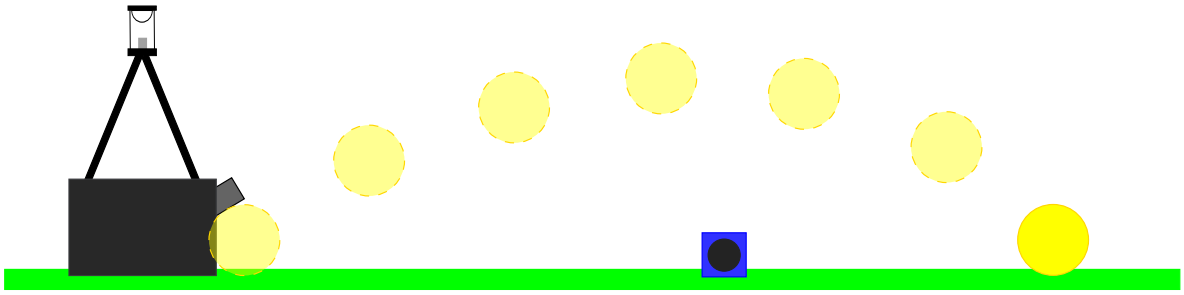


Figure 6.5: Illustration of the setup for measuring the ball kick trajectory.

After extracting the background, the points detected by the laser range that are not part of it are considered the ball. This means that the captures must be performed in a controlled way to avoid anyone or anything besides the ball to cross the laser capture plane. Taking that into account and given the small noise, a simple distance clustering is performed over

the points extracted and the limits of the cluster are estimated. Since we consider the ball as a single point on its center, and given the limits of the cluster, we project the middle point of the cluster by a ball radius (11 cm) in the direction angularly opposing the laser range, which is the observation point. This projection is exemplified in figure 6.6

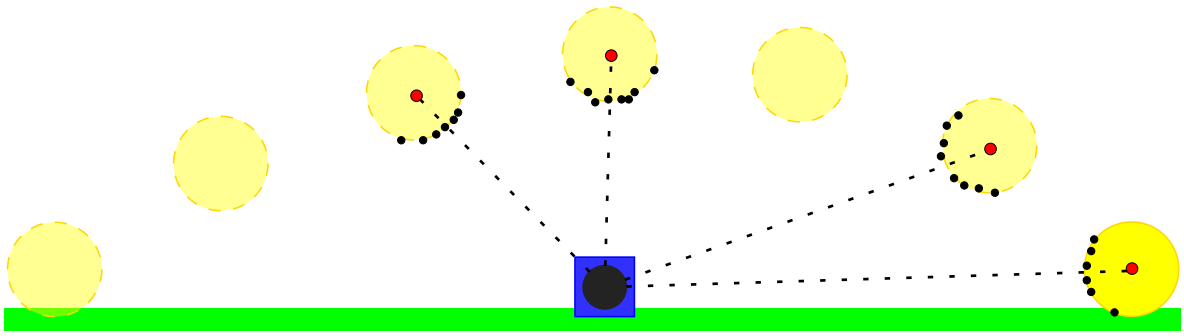


Figure 6.6: Illustration of the projection of the ball center based on points perceived by the laser range. Several examples over the trajectory are illustrated.

In figure 6.7 we have a screenshot of the application, where the background data is represented in green and the current object in black. The estimated ball center is the red dot and the laser, which is the origin of the Cartesian system, is the blue dot.

When creating the new platform, the kicking device was designed and implemented such that its repeatability was as good as possible. For verifying this assumption, we used the data provided by this tool for a set of kick powers on each robot. The obtained results confirmed a quite acceptable repeatability on each robot (figure 6.8). With these results, we were also able to extract the kick parabola for each of the used setpoint kick powers and introduce an analytical configuration of the kicker on our robots.

6.1.5 Kick power estimation

The previous kicker approach was based on a table relating distance with kick power, extracted by having the robot kick with several powers at predefined distances until we visually obtained an acceptable height of the ball on the goal line. The main problem with this approach was that the kick configuration would work rather well when the robot was stopped but not so well when it had movement. Also, being the configuration dependent on a human observation of the ball height, the configuration could easily be very different depending on the observer, since the acceptable height was measured through instantly observing if the ball crossed the goal line at a good height, and "good height" have different meanings to different observers.

In this new approach, we analytically estimate a relation of initial speed with kick power. The advantage of this approach is that, when deciding to kick, we can directly affect the kick power with the current robot speed, since we are directly working on the same physical units.

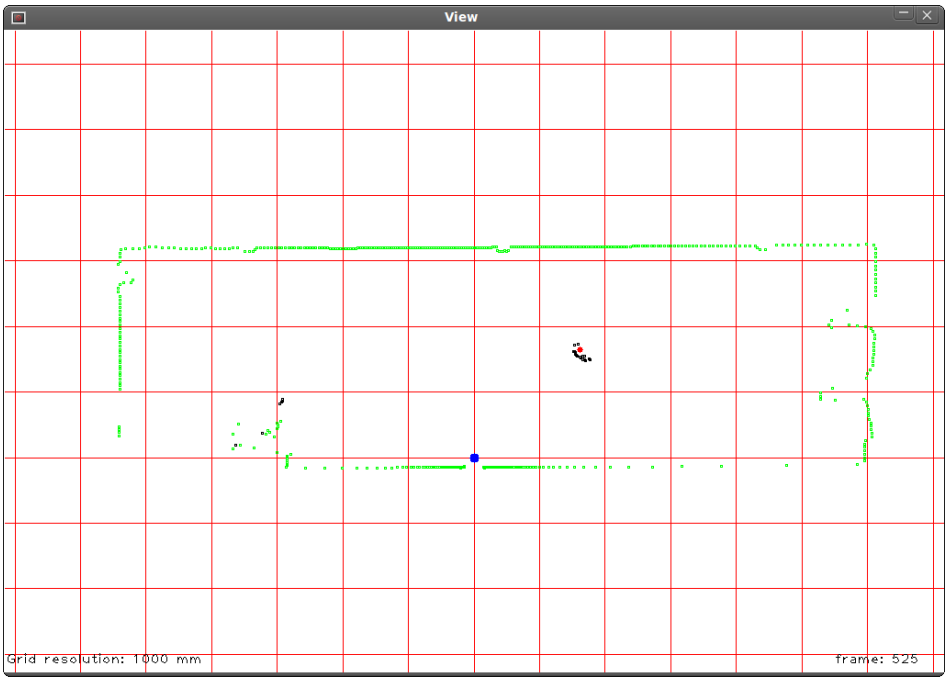


Figure 6.7: Screenshot of the laser range tool. The green dots are points detected as background, which are most of the points detected by the laser range finder. Black points are points that are part of moving objects, the bigger blue point centered horizontally is the laser range position and the red point among the black points isolated to the right of the laser range is the estimated center of the ball points.

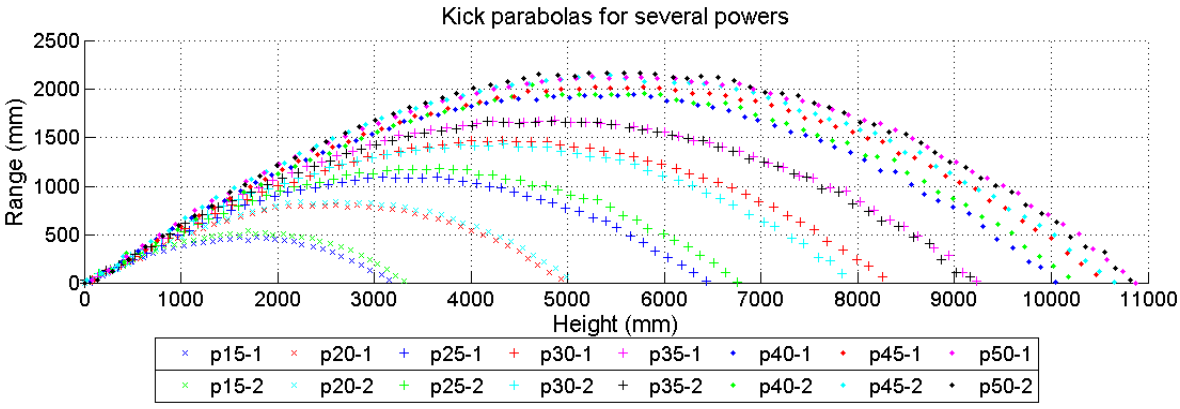


Figure 6.8: Representation of the kicking parabolas of a robot for different kicking powers and trials, represented as pNN-X where NN is the used kicking power and X is the trial.

Using the laser range data, we can easily estimate the maximum distance and height of each parabola. The defined procedure implies that two kicks are performed and captured for each of the setpoint kick powers. With the maximum parabola distance, D, and the maximum parabola height, H, we extract the exit angle, θ, and the exit ball velocity, V_0.

$$D = \frac{V_0^2 \times sin(2\theta)}{g} \qquad H = \frac{V_0^2 \times sin^2(\theta)}{2g}$$

$$\frac{H}{D} = \frac{sin^2(\theta)}{2 \times sin(2\theta)} = \frac{sin^2(\theta)}{4 \times sin(\theta) \times cos(\theta)} = \frac{tan(\theta)}{4}$$

$$\theta = tan^{-1}\left(\frac{4 \times H}{D}\right) \qquad V_0 = \sqrt{\frac{D \times g}{sin(2\theta)}}$$

Considering that the angle, θ, is constant because it is only related to the shape of the kicker and the point where it hits the ball, we consider a mean of the angles of the several parabolas of each robot as the value of θ. A kicking table is created with the exit velocities for each of the setpoint kick powers and those values are used as reference when the robot estimates the power of a kick.

For any given distance, we can estimate the exit velocity of the ball with $V_0 = \sqrt{\frac{D \times g}{sin(2\theta)}}$, which is considered when the robot is stopped. However, since many of the kicks are made when the robot is moving, we have to consider that the robot frontal velocity component will affect the ball exit velocity on its XX component. Thus the velocity considered for the kicking order is $V_{kick} = V_0 - V_{robot} \times cos(\theta)$. One of the configurations of this kicking power estimation method is a constant that affects how much of the robot velocity is transfered to the ball exit velocity, since we noticed that different floors affect this velocity differently. Particularly, as the floor becomes more slippery, the effect of the robot velocity on the ball exit velocity is more complete. The limit situation would be a floor with no friction, which would correspond to a 100% velocity transfer.

On the other hand, when estimating the necessary kick power, we use the exit velocities and kicking setpoints as reference to make a piecewise linear approximation of curve corresponding to those setpoint pairs of values. Thus, the kicking power for any given distance and robot velocity is given by the linear equation between the kicking velocity setpoint immediately lower than V_{kick} and the one immediately higher than V_{kick}.

In table 6.1 a summary of the kick effectiveness in terms of height on the goal line is presented. These results were obtained in the final games of two competitions of 2014, IranOpen2014 where this approach was not implemented and Robotica2014, where this approach was implemented.

We thus obtained a significant improvement on the accuracy of the estimation of the kick power necessary to shoot the ball at a useful height, and not shoot it too high to hit the bar or go over it. An efficiency of 50% was obtained during the IranOpen, while in the Robotica a much better efficiency of 83% was achieved. The described method increased the efficiency

	On height	Too high	Efficiency
Previous method	5	5	50%
Current method	10	2	83%

Table 6.1: Comparison of number of shots with good height (i.e. good kick power) and shots too high with the previous approach and the new proposed approach.

of the estimated kick power by more than 60%.

6.2 Tools for SPL

In a way similar to the MSL, in the SPL we also have the need for graphical tools to help in the development and debug of the team code. Some of the tools presented in the MSL section were inevitably ported and adapted for the SPL, while other specific tools were created.

6.2.1 Direct joint control

One of the possible types of behaviors used in the Portuguese Team is the slot behavior. This type of behavior is based on the sequencing of static body poses over time, thus creating the desired movement. To create such movements, one has to write the positions of every motor of the robot body for each movement pose.

The direct joint control tool appears as an easy to use interface to facilitate the creation of poses for slot behaviors. It allows the user to puppet the robot body and fix it in given poses, make small corrections and export the list of values in the format used on the slot behaviors. Currently the export action is triggered by a single shortcut key and writes the data in a default named file, meaning that between each export, the user must copy the information.

It allows us to remotely control each of the servo motors of the robot, by defining both the stiffness level and the position angle. When remotely connected to a robot, we continually have the information of the current position of each of the motors, as well as the roll and pitch angles provided by the inertial unit (figure 6.9).

The interface is composed by three different sets of values:

- On the left we have a set of text labels with all the motors current position and the roll and pitch angles. These values are read from the robot sensors and thus are presented with simple text labels, i. e., they are read only.

- In the center we have the angle interface for each of the motors. Here we can define, either by manually typing the values or using the spinbox interacts, the angular value for any given motor.

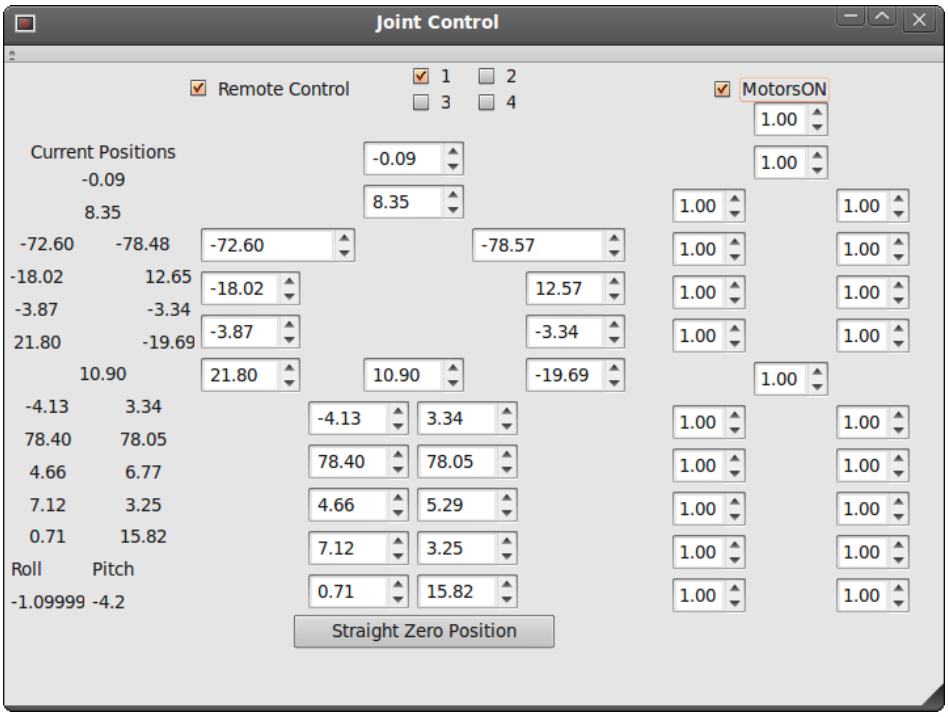


Figure 6.9: Direct joint control tool.

- On the right we have the stiffness interface, which allows the user to define the stiffness value for any motor. This value is defined as a percentage between 0 and 1, 0 meaning motor disabled and 1 meaning motor in full force.

The check box on top left opens or closes a connection to the selected robot on the check boxes immediately to its right. These check boxes are mutually exclusive and thus selecting a different robot with a connection active will close the current connection and try to connect to the new robot. The rightmost check box enables or disables the motors. Even when enabled, the user can keep some motors free by defining their stiffness value to 0.

The main advantage for defining the poses is that we can disable the motors and puppeteer the robot. By enabling the motors we can fix them at the desired pose, evaluate it and do minor corrections by moving any motor through the spinbox interface.

6.2.2 Camera parameters configuration

In section 5.2.1.2 a graphical tool is referred to help in the estimation of the camera parameters, as well as in the verification of the projection model.

This tool is composed by a main display widget where a video is played and virtual elements are overlapped according to the configured projection model. The video can either be a loaded file or a live feed from a NAO robot's camera. In video mode, the controls on the

top left of the main window (figure 6.10) are composed by a slider and spinbox that can be used to navigate to any given frame or the video can be played normally. When the option to go live is used, an IP address must be provided in the correspondent text box.

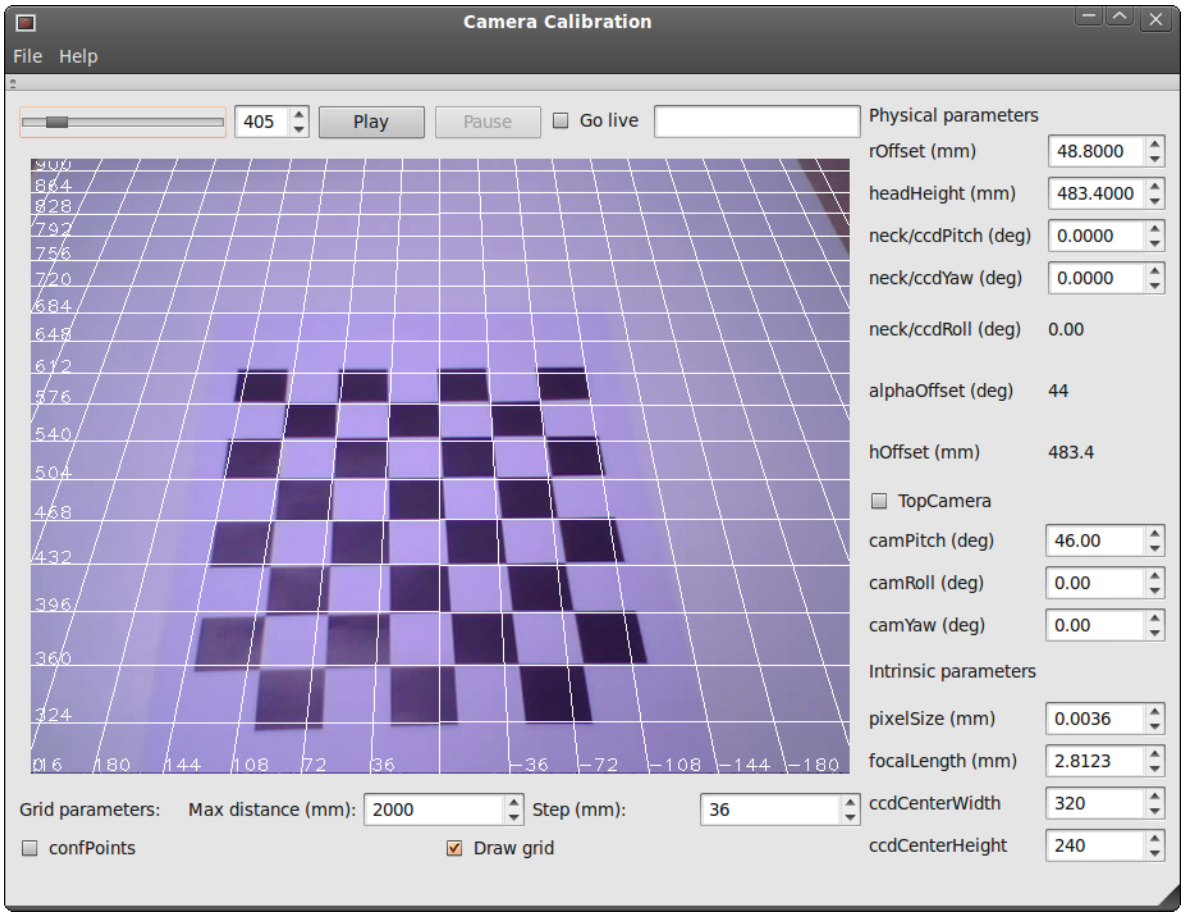


Figure 6.10: Example of the perspective camera calibration tool. It allows to lively test several configuration parameters of both intrinsic and extrinsic camera parameters and allows to visually confirm the projection model by comparison of known elements (like a chessboard) with an overlapped grid on the image.

On the right side of the application window are a set of parameters used by the projection model. These parameters are divided in three categories:

- On top, we have four values that describe the distances of some reference points (total height of the head and offset from head center to camera CCD in the front) and the angles of the neck actuators. These values are either manually defined when in video mode or provided by the agent process through the RTDB when in live mode. Below the first values, editable in video mode, some processed values are presented that are estimated trough combination of the first ones.
- A second set of parameters are the camera angles related to the head axis. These values

are editable and allow us to, through trial and error, define the angles of the camera placement, which are not physically guaranteed by the robot assembly.

- The last set of values allows us to define some intrinsic parameters of the camera, which are not provided by the robot company and thus need to be "discovered" through testing, mainly the focal length of the lenses.

The projection algorithm used by the robots is the one used by the tool, which takes all the defined/read values into account (as explained in section 5.2.1.2) and projects a virtual grid with user defined size over the real image, allowing the user to visually access the fit of the projection over the ground. Typically this verification is done by verifying the fit of the virtual grid over the field lines which are well known or, as seen in figure 6.10, over a well known printed grid.

6.2.3 Basestation

A basestation application is an important feature to any league of robotic soccer, since it is a way (usually the only way) to have an idea of what is happening with the robots in real time.

A basestation application similar to the CAMBADA one was created for the NAOs data during the course of this work. Since the work included the definition of the code structure, the adaptations were embedded in the basestation directly.

The general look of the application is a 2D version of the CAMBADA, as well as the general information provided by the robots, since the scenario is the same.

The main feature that was added, mainly due to the lack of graphical capabilities on the NAO robot processing board, is the possibility to visualize the white points over the lines detected by a robot (figure 6.11).

This feature was added so that, during development, we could more easily have a notion of what are the white points over the lines perceived by the robot. It is also very useful to visually verify if the localization algorithm is providing a satisfactory estimation.

6.2.4 Cursor

To assist the development of code for the SPL team, the creation of a remote control application with which we could drive the agent around while gathering sensor data to perform tests was important. The cursor tool is used for that purpose, in this case by making use of the Aldebaran walk engine, but prepared to use any omni directional walk that is defined by separated speeds for each velocity component. The tool also became an important feature to test the walk on different surfaces and infer a satisfactory set of walk parameters to promote stability of the walk.

Figure 6.11: The SPL basestation field with the representation of the relative positions of the detected white points.

With this tool, the user can set the parameters of the walking speed as defined by the Aldebaran, with a percentage of XX axis linear speed, YY axis linear speed and angular speed. Also a frequency of execution can be defined (figure 6.12).

In the case of the Aldebaran omni directional walk, the relative "speed" is basically a percentage of the maximum feet opening in the respective direction, while the frequency is the actual frequency of the movement.

However, the cursor can be applied to any omni directional walk that is parameterized in each direction. The agent implementation is the one that defines how the parameters will be used.

One other important aspect to consider is that the NAO cursor application is not standalone like the CAMBADA cursor application. This means that, while in the CAMBADA, the cursor is executed directly on the robot PC, in the NAOs, that is not possible, due to the lack of graphical capabilities on their on board processor. The cursor application is executed on the control PC and the commands redirected through the basestation.

Additionally, the tool structure allows to bind slot behaviors to different buttons to allow the user to use the cursor to test these type of behaviors. The current version of the cursor has two buttons binded to the kick with left leg and kick with right leg slot behaviors.

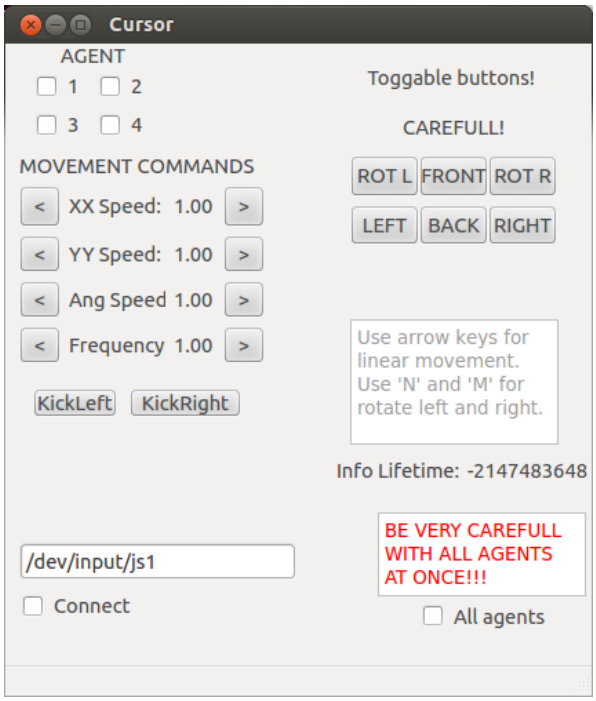


Figure 6.12: The NAO GUI cursor application

6.3 Summary

During the development of projects such as robotic soccer teams there are lots of tasks that require testing, visualization, validation and configuration of parameters and results. To facilitate the daily work on development and testing of these projects, some tools were created.

Concerning the CAMBADA team and the Middle Size League, several tools were improved or created that were described in this chapter. The basestation, being the main team interface, needs to provide functionalities that allow the developers team to understand what is happening with the robots during their operation. Some important functionalities were added and were described.

Remote control of a robot, while being completely against the league and team objectives, is an important feature for test and development. The cursor remote control tool of the CAMBADA robots was developed with a graphical interface that allows both testing of low level aspects, like holonomic platform control, and configuration of some team aspects, such as the robots grabbing devices and the reference heading on each field of play.

During the development of vision algorithms, it is common to make use of the vision save feature that saves the camera video feed. These videos are important features for testing and developing the vision algorithms and sometimes specific clips from large vision captures are desired. A small graphical application was developed to allow cropping videos taken from the

CAMBADA robot cameras.

A laser range finder is a high precision sensor that is useful for many situations. In the CAMBADA team, it allowed the development of an application to analyze a robot's kick trajectory in a way that was not possible before. With a well defined setup and testing, we were able to better understand the robot kick and develop a new, more efficient approach for estimation of the kick power that the robot needs to use in different situations. This approach was explained and its results presented.

Concerning the Standard Platform League team and given the need to create behaviors with pose sequencing, a direct joint control tool was created to allow an easy and quick to use interface to puppeteer a robot into a given pose, fix it, make small adjustments and generally testing the pose.

For assisting in the development and test of the projection model and all the configuration of all the necessary parameters used by the model, an application was developed. The application allows to immediately verify the results of the parameterization defined by the user through the drawing of a grid with known size over the real image, allowing the user to compare the virtual grid with real artifacts that should be present on the image of a defined setup.

Similar to the MSL team, a basestation application was created to allow configuration and visualization of the team robots performance. This basestation is presented with the possibility to visualize the white points seen by a robot, since it is not possible to visualize them on the robot due to lack of graphical capabilities of its processor.

Finally, a remote control application was presented, which is an important tool mainly for testing parameterization of walking parameters.

Chapter 7

Conclusion and future work

This document presented several solutions and analysis of several different aspects regarding world modeling of a soccer robot. In this final chapter, a brief conclusion and discussion of their effects and effectiveness is presented, as well as a small discussion of some future work on some of the covered topics.

7.1 Conclusions and discussion

The robotic soccer scenario is a worldwide used test bed to promote development of autonomous multi agent robotics because of the challenges it poses for the several sub areas of the multi disciplinary robotics area. Modeling the robot's own state and the state of its surroundings is one of those areas.

The accomplished work focus mainly on the construction of the world model representation of a soccer robot by gathering the raw information available through all the sensors of a robot and merging all the information into a structured repository of information that contains extended knowledge about each element of the world, inferred through information fusion methodologies. This structured repository is the world model or worldstate.

Considering the worldstate of the Middle Size League robots in the context of this PhD, the several tackled problems were solved through a set of solutions that, in a general way, provided better results and are thus fully integrated in the team official competition code.

The knowledge about the ball is mainly composed by its position and velocity. The proposed solution for estimation of this knowledge is based on a Kalman filter with reset capabilities for estimation and refinement of the position and the estimation of the velocity is achieved by a linear regression interlaced with the position filter and reseting feature. Despite the unavailability of a global detection system that could provide a ground truth for analyzing the results of this approach, the possible analysis made (by taking manual measurements of

controlled test scenarios and observing the general robot behavior when interacting with the ball) shows a significant improvement on robot performance, mainly when considering stability of the movements and coherence of decisions.

To further extend the knowledge about the ball, one limitation imposed by robot construction had to be overcome: the inability to detect the ball when it is airborne. To overcome this limitation, a RGB camera was installed on the goal keeper robot and a solution for detecting and estimating the ball position when airborne was designed. This solution was subject to several steps and progressed to a state where the results obtained were good enough to apply in the official competition code. The information resulting from this approach provided information to the integration module so that it could fill the "gaps" about the ball whereabouts when it was kicked high. Although the amount of information achieved by this approach was used and provided a slight improvement of robot performance, a solution that provides more complete information was desirable. The methodology for the detection of an airborne ball was subject to a new approach when the Kinect depth sensor became widely available. By exploring the capabilities of this sensor, we are able to obtain more reliable results is terms of ball detection and candidate validation, as well as precision of the estimation of its position. During the RoboCup2014, the use of the information provided by the airborne ball detection, using Kinect, by the goal keeper showed good results in terms of robot behavior, allowing it to intercept some kicks that were not detectable before.

The results obtained in terms of successfully detecting robot lost situations and start a relocalization process have been observed over the years and proved to work in most situations. The solution, however, has a limitation to its use in the situations where the field of play is strongly affected by magnetic fields, since the values measured by the electronic compass are completely incoherent.

When inferring knowledge about the visually detected obstacles on the field it is important to characterize each of them so that a distinction between team mates and opponents is provided. An integrated solution was designed for creating the notion of obstacle from visually independent points and identifying the obstacles visible on the positions of team robots as the correspondent team mate. This solution is successfully working on the official competition code and provides a high success rate. The team mate identification provided by this solution is currently underneath the pass decision algorithms that evaluate the pass lines and pass receiver robot based on this information. Further developments concerning obstacle characterization led to the creation of an obstacle tracking methodology, based on a multiple hypothesis tracking over a Kalman filter implementation, that allows each opponent robot to be unequivocally identified and tracked around. This solution is also fully integrated in the official competition code and has already been explored by the decision layer in the implementation of the opponent covering capabilities, which are used during the game to cover pass lines for opponent in every setpiece situations of the opponent team. Given the improved

results on obstacle detection, which include more stability and precision of the obstacles, an obstacle avoidance methodology was implemented that takes advantage of this stability for improved coherence of the avoidance decision. The avoidance algorithm is being used on the robots and provided an increase of successfully avoided collisions.

Concerning the solution presented for generating points on a trajectory for the robot to follow, it has been observed that the robot performance through this method is similar when performing simple linear movements where the robot starts and ends the movement with zero velocity. However, it was verified that the solution is unstable when the robot target changes rapidly, which is currently an issue since the Middle Size League is a fast paced scenario. Further development is due to tackle a solution that can cope with this issue.

In the Standard Platform League scenario, the developed work included the design and implementation of a base software architecture for the NAO robots. The proposed architecture, as well as the base implementation of its several modules are being used as the current architecture for the Portuguese Team.

For allowing the robot to build its world model in terms of metric units for position and velocities of the game entities, a method to transform the pixel coordinates of artifacts provided by the vision into metric coordinates on the ground plane, relative to the robot was proposed. This projection model has been defined, implemented and used on the NAO robots during the official competitions.

To tackle the localization problem on the SPL, an adaptation of the tribots localization algorithm was explored. This approach provided good results when the available data provides discriminate information. To achieve good results in the localization, and considering that the NAO field of view is very reduced, the proposed solution is for the robot to follow an active localization methodology and, from time to time, focus its behavior on gathering information of points over field lines around itself, by stopping and looking around in distinct directions. This solution has been used and provided good results for the localization problem.

During the development of solutions for the robotic soccer scenarios several tools were used to aid testing, configuration and debugging, both for the work accomplished in the MSL and in the SPL contexts. Most of the used tools were improved or fully developed in the context of this PhD work and were a crucial aid in the development of the solutions presented on this work.

One particular tool developed in the MSL context allowed the development of a new approach for the CAMBADA robots that critically increased team performance. The data resulting from the laser range lob analyzer tool provided a depth insight about the kicking capabilities of the CAMBADA robots and thanks to it, a new solution for estimating the kick power necessary for a robot to kick the ball was designed. This solution greatly increased

the precision of the CAMBADA robots kick in terms of height, which was evident during the games due to the drastic reduction of balls shot too high (on the goal top bar or over the goal) and consequent increase of balls reaching the goal line with the desired height.

7.2 Future work

During the development of the several tasks of this work, some development directions were clearly identified and are presented next:

- Thinking about the airborne ball detection using the Kinect sensor, one should note that currently the process returns the position of the center of the point cloud blob which is evaluated as possessing the higher mass. We expect that, even in the false positive situation of the captures, the ball was detected on the point cloud, but with a mass smaller than the returned false positive. A near future improvement of this process should be to, instead of returning the thicker mass as the only candidate, return a list of the several masses detected that has a minimum mass to be a ball. The choice of which information to use is then handled by the integrator module on the *agent* process, which can try to contextualize each candidate with both the surroundings and historical information.

- As future work on the matter of the IMU, the most coveted one is the use of the other IMU measurements, both in terms of accelerations and rotations to improve the robot proprioception. The merging and filtering of the three accelerations and three angular speeds provided by the IMU can result in a way to detect if the robot is stuck, if a collision occurred and if the current acceleration is the supposed one or the robot grip is not as expected. Of course that after detecting any of the situations, several behavioral approaches must be implemented to cope with any given situation.

  Another desired feature that can be achieved by using the IMU, and given its position on the top of the robot and fixed with the vision system, is to estimate the angles of the camera relative to the ground plane, by using the gravity as reference. This could allow the distance relation of pixel coordinates to ground coordinates to be adjusted in real time according to the camera pose. This would provide better estimations of the visible points of interest and thus a smaller displacement error than the current one.

- Concerning the obstacle avoidance algorithm, there are two main improvements that can and should be implemented in a near future, both concerning the free sonar search algorithm.

  One of them is to avoid general search directions in the first cycles of the avoidance algorithm and take into account the position of the robot on the field. We have verified

that in some situations, when the robot chooses to avoid obstacles in a direction that takes it to the field line and since we do not allow the robot to go out of the field, mostly for safety reasons, the behavior on those limit situations becomes unstable. Thus, one can take the position into account and make the search for a free sonar to the inside of the field, even if it takes the robot back, which we consider more useful and safer than forcing it to the outside of the field.

The second main improvement would be to analyze the free directions in different ways for different types of obstacles, team mate or opponent. We believe that, given the shared information of the team mates, which include the velocity vector, the avoidance can be more adaptive in such way that an avoidance decision can be almost considered as a shared decision. Moreover, the idea came up to share the target of each robot in its information so that its general direction can be determined and used to improve the avoidance decision.

Bibliography

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

[2] Ren C. Luo, Chih-Chen Yih, and Kuo L. Su. Multisensor fusion and integration: Approaches, applications, and future research directions. *IEEE Sensors Journal*, 2(2):107–119, April 2002.

[3] Bruno Siciliano and Oussama Khatib, editors. *Springer Handbook of Robotics*. Springer, 2008.

[4] Hugh Durrant-Whyte and Tom Henderson. Multisensor data fusion. In Siciliano and Khatib [3], pages 585–613.

[5] Nicholas Metropolis and S. Ulam. The Monte Carlo method. *Journal of American Statistical Association*, 44(247):335–341, 1949.

[6] R.E. Kalman. A New Approach to Linear Filtering and Prediction Problems. *Journal of Basic Engineering*, 82(1):35–45, 1960.

[7] Eric A. Wan and Rudolph van der Merwe. The unscented Kalman filter for nonlinear estimation. In *IEEE Adaptive Systems for Signal Processing, Communications, and Control Symposium*, pages 153–158, Alberta, Canada, October 2000.

[8] Greg Welch and Gary Bishop. An Introduction to the Kalman Filter. In *Proc of SIGGRAPH, Course 8*, Chapel Hill, NC, USA, 2001.

[9] Peter S. Maybeck. *Stochastic Models, Estimation, and Control*, volume 1, chapter 1. Academic Press, 1979.

[10] Michael Isard and Andrew Blake. Condensation—conditional density propagation for visual tracking. *International Journal of Computer Vision*, 29(1):5–28, 1998.

[11] Tim Laue and Thomas Röfer. Pose extraction from sample sets in robot self-localization - a comparison and a novel approach. In Ivan Petrovic and Achim J. Lilienthal, editors,

Proceedings of the 4th European Conference on Mobile Robots - ECMR'09, pages 283–288, Mlini/Dubrovnik, Croatia, 2009.

[12] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. The MIT Press, 2005.

[13] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. Markov localization for mobile robots in dynamic environments. *Journal of Artificial Intelligence Research*, 11:391–427, 1999.

[14] Andrew Howard, Maja J. Mataric, and Gaurav S. Sukhatme. Localization for mobile robot teams using maximum likelihood estimation. In *IEEE International Conference on Intelligent Robots and Systems*, pages 434–459, Lausanne, Switzerland, September 2002.

[15] Stergios I. Roumeliotis and George A. Bekey. Bayesian estimation and Kalman filtering: A unified framework for mobile robot localization. In *IEEE International Conference on Robotics and Automation*, volume 3, pages 2985–2992, San Fransisco, USA, April 2000.

[16] Stergios I. Roumeliotis and George A. Bekey. Distributed multirobot localization. *IEEE Transactions on Robotics and Automation*, 18(5):781–795, 2002.

[17] Hiroshi Koyasu, Jun Miura, and Yoshiaki Shirai. Realtime omnidirectional stereo for obstacle detection and tracking in dynamic environments. *IEEE International Conference on Intelligent Robots and Systems*, pages 31–36, October-November 2001.

[18] John J. Leonard and Hugh F. Durrant-Whyte. Mobile robot localization by tracking geometric beacons. *IEEE Transactions on Robotics and Automation*, 7(3):376–382, 1991.

[19] Patrick Heinemann, Juergen Haase, and Andreas Zell. A combined monte-carlo localization and tracking algorithm for robocup. In *IEEE International conference on Intelligent Robots and Systems*, pages 1535–1540, Beijing, China, October 2006.

[20] Luis Montesano, José Gaspar, José Santos-Victor, and Luis Montano. Cooperative localization by fusing vision-based bearing measurements and motion. In *IEEE International Conference on Intelligent Robots and Systems*, pages 2333–2338, Edmont, Canada, August 2005.

[21] J.-S. Gutmann and D. Fox. An experimental comparison of localization methods continued. In *IEEE International conference on Intelligent Robots and Systems*, volume 1, pages 454–459, Lausanne, Switzerland, September 2002.

[22] Frank Dellaert, Dieter Fox, Wolfram Burgard, and Sebastian Thrun. Monte Carlo localization for mobile robots. In *IEEE International Conference on Robotics and Automation*, volume 2, pages 1322–1328, Detroit, USA, May 1999.

[23] David Cabecinhas, João Nascimento, João Ferreira, Paulo Rosa, and Pedro Lima. Self-localization based on kalman filter and monte carlo fusion of odometry and visual information. In *Proceedings of Robotica 2006*, 2006.

[24] M. Lauer, S. Lange, and M. 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*, volume 4020 of *Lecture Notes in Computer Science*, pages 142–153. Springer, 2006.

[25] Manuel Gouveia, António Paulo Moreira, Paulo Costa, Luís Paulo Reis, and Marcos Ferreira. Robustness and precision analysis in map-matching based mobile robot self-localization. In *New Trends in Artificial Intelligence: 14th Portuguese Conference on Artificial Intelligence*, pages 243–253, 2009.

[26] M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: the RPROP algorithm. In *IEEE International Conference on Neural Networks*, volume 1, pages 586–591, 1993.

[27] David Filliat and Jean arcady Meyer. Map-based navigation in mobile robots: I. A review of localization strategies. *Cognitive Systems Research*, 4(4):243–282, December 2003.

[28] George A. Bekey. *Autonomous Robots: From Biological Inspiration to Implementation and Control*. The MIT Press, 2005.

[29] Michael Montemerlo, Sebastian Thrun, Daphne Koller, and Ben Wegbreit. FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *AAAI National Conference on Artificial Intelligence*, pages 593–598. AAAI, 2002.

[30] Andrew J. Davison. *Mobile robot navigation using active vision*. PhD thesis, University of Oxford, 1998.

[31] Steven Holmes, Gabe Sibley, Georg Klein, and David W. Murray. A relative frame representation for fixed-time bundle adjustment in SFM. In *IEEE International Conference on Robotics and Automation*, pages 2631–2636, Kobe, Japan, 2009. IEEE Press.

[32] Kai Ni and Frank Dellaert. Multi-level submap based slam using nested dissection. In *IEEE International conference on Intelligent Robots and Systems*, pages 2558–2565, Taipei, Taiwan, October 2010.

[33] Simon J. Julier and Jeffrey K. Uhlmann. Using covariance intersection for {SLAM}. *Robotics and Autonomous Systems*, 55(1):3–20, 2007.

[34] Raj Madhavan, Kingsley Fregene, and Lynne E. Parker. Distributed cooperative outdoor multirobot localization and mapping. *Autonomous Robots*, 17(1):23–39, July 2004.

[35] Ioannis M. Rekleitis, Gregory Dudek, and Evangelos E. Milios. Multi-robot cooperative localization: A study of trade-offs between efficiency and accuracy. In *IEEE International Conference on Intelligent Robots and Systems*, pages 2690–2695, Lausanne, Switzerland, September 2002.

[36] Anastasios I. Mourikis and Stergios I. Roumeliotis. Performance analysis of multirobot cooperative localization. *IEEE Transaction on Robotics*, 22(4):666–681, August 2006.

[37] Sebastian Thrun. A probabilistic online mapping algorithm for teams of mobile robots. *International Journal of Robotics Research*, 20(5):335–378, 2001.

[38] Ashley W. Stroupe, Martin C. Martin, and Tucker Balch. Merging gaussian distributions for object localization in multi-robot systems. In *Seventh International Symposium on Experimental Robotics*, volume 271, pages 343–352. Springer, 2000.

[39] Simon J. Julier and Jeffrey K. Uhlmann. Using multiple SLAM algorithms. In *IEEE International Conference on Intelligent Robots and Systems*, volume 1, pages 200–205, Las Vegas, USA, October 2003.

[40] Yaakov Bar-Shalom and Xiao-Rong Li. *Multitarget-Multisensor Tracking: Principles and Techniques*. YBS Publishing, 1995.

[41] Patric Jensfelt and Steen Kristensen. Active global localization for a mobile robot using multiple hypothesis tracking. *IEEE Transactions on Robotics and Automation*, 17(5):748–760, October 2001.

[42] Tat jen Cham and James M. Rehg. A multiple hypothesis approach to figure tracking. In *IEEE Computer Vision and Pattern Recognition*, pages 239–245, FortCollins, USA, June 1999.

[43] Pablo Arambel, Matthew Antone, Constantino Rago, Herbert Landau, and Thomas Strat. A multiple-hypothesis tracking of multiple ground targets from aerial video with dynamic sensor control. In *Proc International Conference on Information Fusion*, pages 1080–1087, Stockholm, Sweden, June-July 2004.

[44] Henk A.P. Blom. An efficient filter for abruptly changing systems. In *IEEE International Conference on Decision and Control*, volume 23, 1984.

[45] Henk A.P. Blom and Yaakov Bar-Shalom. The interacting multiple model algorithm for systems with Markovian switching coefficients. *IEEE Transactions on Automatic Control*, 33(8):780–783, 1988.

[46] P. Vacher, I. Barret, and M. Gauvrit. Design of a tracking algorithm for an advanced ATC system. In Y. Bar-Shalom, editor, *Multitarget-multisensor tracking: applications and advances*, volume 2, pages 1–29. Artech House, 1992.

[47] Emil Semerdjiev and Ludmila Mihaylova. Adaptive interacting multiple model algorithm for manoeuvring ship tracking. In *Proc International Conference on Information Fusion*, pages 974–979, 1998.

[48] Szymon Ceranka. Sensor Fusion Algorithms for Pedestrian Location. In *International Conference on Methods and Models in Automation and Robotics*, volume 2, pages 1343–1348, Szczecin, Poland, 2002.

[49] Javier Civera, Andrew J. Davison, and J.M.M. Montiel. Interacting multiple model monocular SLAM. In *IEEE International Conference on Robotics and Automation*, pages 3704–3709, Pasadena, USA, May 2008.

[50] E. Mazor, A. Averbuch, Y. Bar-Shalom, and J. Dayan. Interacting multiple models methods in multitarget-multisensor tracking: Survey. In *IEEE Transactions on Aerospace and Electronic Systems*, volume 34, pages 103–123, January 1998.

[51] Jianyu Wang, Debin Zhao, Wen Gao, and Shiguang Shan. Interacting multiple model particle filter to adaptive visual tracking. In *Proc International Conference on Image and Graphics*, pages 568–571, Hong Kong, December 2004.

[52] Guillem Alenyà, Elisa Martínez, and Carme Torras. Fusing visual and inertial sensing to recover robot ego-motion. *Journal of Robotic Systems*, 21(1):23–32, January 2004.

[53] S.G. Chroust and M. Vincze. Fusion of vision and inertial data for motion and structure estimation. *Journal of Robotic Systems*, 21(2):73–83, February 2004.

[54] Jorge Lobo and Jorge Dias. Inertial sensed ego-motion for 3D vision. *Journal of Robotic Systems*, 21:3–12, 2004.

[55] João Alves, Jorge Lobo, and Jorge Dias. Camera-inertial sensor modelling and alignment for visual navigation. In *Proc 11th International Conference on Advanced Robotics*, pages 1693–1698, Coimbra, Portugal, 2003.

[56] Jorge Lobo and Jorge Dias. Vision and inertial sensor cooperation using gravity as a vertical reference. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(12):1597–1608, 2003.

[57] Jorge Lobo and Jorge Dias. Relative pose calibration between visual and inertial sensors. *International Journal of Robotics Research*, 26(6):561–576, 2007.

[58] Peter Corke, Jorge Lobo, and Jorge Dias. An introduction to inertial and visual sensing. *International Journal on Robotics Research*, 26(6):519–535, 2007.

[59] Luiz G. B. Mirisola and Jorge Dias. Tracking from a moving camera with attitude estimates. In *ICR 08 - The 2nd Israeli Conference on Robotics*, Herzlia, Israel, November 2008.

[60] António J. R. Neves, José Luís Azevedo, Bernardo Cunha, Nuno Lau, João Silva, Frederico Santos, Gustavo Corrente, Daniel A. Martins, Nuno Figueiredo, Artur Pereira, Luís Almeida, Luís Seabra Lopes, Armando J. Pinho, João Rodrigues, and Paulo Pedreiras. CAMBADA soccer team: from robot architecture to multiagent coordination. In Vladan Papic, editor, *Robot Soccer*, pages 19–45. INTECH, January 2010.

[61] R. Hafner, S. Lange, M. Lauer, and M. Riedmiller. Brainstormers Tribots Team Description, 2008.

[62] Amir A.F. Nassiraei, Shuichi Ishida, Noriyuki Shinpuku, Miyuki Hayashi, Naoya Hirao, Kazunori Fujimoto, Kazutaka Fukuda, Kazutomo Takanaka, Ivan Godler, Kazuo Ishii, and Hiroyuki Miyamoto. Hibikino-Musashi Team Description Paper, 2014.

[63] Cesar Lopez Martinez, Ferry Schoenmakers, Gerrit Naus, Koen Meessen, Yanick Douven, Harrie van de Loo, Dennis Bruijnen, Wouter Aangenent, Joost Groenen, Bob van Ninhuijs, Matthias Briegel, Rob Hoogendijk, Patrick van Brakel, Rob van den Berg, Okke Hendriks, René Arts, Frank Botden, Wouter Houtman, Marjon van 't Klooster, Jeroen van der Velden, Camiel Beeren, Lotte de Koning, Olaf Klooster, Robin Soetens, and René van de Molengraft. Tech United Eindhoven Team Description 2014, 2014.

[64] Xueyan Wang, Yong Zhao, Song Chen, Xiaoming Liu, Wanjie Zhang, Xinxin xu, Ye Lu, and Ye Tian. Water Team Description 2014, 2014.

[65] João Messias, Aamir Ahmad, João Reis, Miguel Serafim, and Pedro Lima. SocRob 2013 Team Description Paper, 2013.

[66] T. Amma, J. Beifuß, Z. Bozic, M. Bui, F. Gawora, T. Haque, K. Geihs, S. Jakob, D. Kirchner, N. Kubitza, K. Liebscher, S. Opfer, D. Saur, T. Schaake, T. Schluter, S. Triller, and A. Witsch. Carpe Noctem 2013, 2013.

[67] M.Gholipour, H.Rasam Fard, M.Montazeri, F.Fathalibeyglou, M.Rasam Fard, E.Saeedi Kamal, B.Tajshafaghi, M.Farsi, Sh.Kazemi, M.Hoshyari, S.Ziyadloo, and Sh.Arash. MRL Middle Size Team: RoboCup 2014 Team Description Paper, 2014.

[68] J. L. Azevedo, N. Lau, G. Corrente, A. Neves, M. B. Cunha, F. Santos, A. Pereira, L. Almeida, L. S. Lopes, P. Pedreiras, J. Vieira, P. Fonseca, D. Martins, N. Figueiredo, J. Puga, and J. Taborda. CAMBADA'2007: Team Description Paper, 2007.

[69] 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 ISCIS*. Springer, 2004.

[70] V. Silva, R. Marau, L. Almeida, J. Ferreira, M. Calha, P. Pedreiras, and J. Fonseca. Implementing a distributed sensing and actuation system: The CAMBADA robots case study. In *Proc. of the 10th IEEE Conference on Emerging Technologies and Factory Automation, ETFA 2005*, volume 2, 2005.

[71] J.L. Azevedo, B. Cunha, and L. Almeida. Hierarchical distributed architectures for autonomous mobile robots: A case study. In *Proc. of the 12th IEEE Conference on Emerging Technologies and Factory Automation, ETFA 2007*, pages 973–980, 2007.

[72] F. Santos, L. Almeida, P. Pedreiras, L.S. Lopes, and T. Facchinetti. An Adaptive TDMA Protocol for Soft Real-Time Wireless Communication among Mobile Autonomous Agents. In *Proc. of the Int. Workshop on Architecture for Cooperative Embedded Real-Time Systems, WACERTS 2004*, 2004.

[73] F. Santos, G. Corrente, L. Almeida, N. Lau, and L.S. Lopes. Selfconfiguration of an Adaptive TDMA wireless communication protocol for teams of mobile robots. In *Proc. of the 13th Portuguese Conference on Artificial Intelligence, EPIA 2007*, 2007.

[74] J. L. Azevedo, N. Lau, G. Corrente, A. Neves, M. B. Cunha, F. Santos, A. Pereira, L. Almeida, L. S. Lopes, P. Pedreiras, J. Vieira, D. Martins, N. Figueiredo, J. Silva, N. Filipe, and I. Pinheiro. CAMBADA'2008: Team Description Paper, 2008.

[75] R. Bosch. CAN Specification Version 2.0. Technical report, Stuttgart, Germany: Robert Bosch GmbH, 1991.

[76] P. Pedreiras, F. Teixeira, N. Ferreira, L. Almeida, A. Pinho, and F. Santos. Enhancing the Reactivity of the Vision Subsystem in Autonomous Mobile Robots Using Real-Time Techniques. In *RoboCup 2005: Robot Soccer World Cup IX*, volume 4020 of *Lecture Notes in Computer Science*, pages 371–383. Springer, 2006.

[77] David Gouaillier, Vincent Hugel, Pierre Blazevic, Chris Kilner, Jérôme Monceaux, Pascal Lafourcade, Brice Marnier, Julien Serre, and Bruno Maisonnier. The nao humanoid: a combination of performance and affordability. *CoRR*, abs/0807.3223, 2008.

[78] Alina Trifan, António J. R. Neves, Nuno Lau, and Bernardo Cunha. A modular real-time vision system for humanoid robots. In *Proc. of IS&T/SPIE Electronic Imaging 2012*, San Francisco, USA, January 2012.

[79] Zhiwen Zeng, Dan Xiong, Qinghua Yu, Kaihong Huang, Shuai Cheng, Huimin Lu, Xiangke Wang, Hui Zhang, Xun Li, and Zhiqiang Zheng. NuBot Team Description Paper 2013, 2013.

[80] R. Dias, F. Amaral, J. L. Azevedo, R. Castro, B. Cunha, J. Cunha, P. Dias, N. Lau, C. Magalhães, A. J. R. Neves, A. Nunes, E. Pedrosa, A. Pereira, J. Santos, J. Silva, and A. Trifan. CAMBADA'2014: Team Description Paper, 2014.

[81] Amir A.F. Nassiraei, Shuichi Ishida, Noriyuki Shinpuku, Miyuki Hayashi, Naoya Hirao, Kazunori Fujimoto, Kazutaka Fukuda, Kazutomo Takanaka, Ivan Godler, Kazuo Ishii, and Hiroyuki Miyamoto. Hibikino-Musashi Team Description Paper, 2013.

[82] David Jahshan. MU-Penguins 2013 Team Description, 2013.

[83] Pedro U. Lima, Pedro Santos, Ricardo Oliveira, Aamir Ahmad, and João Santos. Cooperative localization based on visually shared objects. In Javier Ruiz-del Solar, Eric Chown, and PaulG. Plöger, editors, *RoboCup 2010: Robot Soccer World Cup XIV*, volume 6556 of *Lecture Notes in Computer Science*, pages 350–361. Springer, 2011.

[84] Sebastian Thrun, Dieter Fox, Wolfram Burgard, and Frank Dellaert. Robust monte carlo localization for mobile robots. *Artificial Intelligence*, 128(1-2):99–141, 2000.

[85] José Almeida, Alfredo Martins, André Dias, Hugo Silva, Carlos Almeida, Nuno Dias, Luís Lima, Guilherme Silva, and Eduardo Silva. ISePorto Robotic Soccer Team for Robocup 2013: Improving Team Perception, 2013.

[86] Aamir Ahmad and Pedro Lima. Multi-robot cooperative object tracking based on particle filters. In *Proc. of the European Conference on Mobile Robots (ECMR 2011)*, Örebro, Sweden, September 2011.

[87] M.Montazeri, B.Eskandariun, M.Mahmoodi, F.Fathalibeyglou, M.Ghafouri Tabrizi, R.Fathi Rad, H.Amiri, M.Hajikarimian, M.Rasam Fard, E.Saeedi Kamal, M.Noori, B.Tajshafaghi, M.Farsi, V.Houshangi, E.Jazeb Nikoo, F.Jafari, and M.A.Ghoochani. MRL Middle Size Team: RoboCup 2013 Team Description Paper, 2013.

[88] 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, March 2011.

[89] Hessel van der Molen. Self-Localization in the RoboCup Soccer Standard Platform League with the use of a Dynamic Tree. Master's thesis, University Of Amsterdam, 2011.

[90] Dieter Fox, Wolfram Burgard, Frank Dellaert, and Sebastian Thrun. Monte carlo localization: Efficient position estimation for mobile robots. In *Proc. of the National Conference on Artificial Intelligence (AAAI*, pages 343–349, 1999.

[91] Tim Laue and Thomas Röfer. Particle filter-based state estimation in a competitive and uncertain environment. In *Proc. of the 6th International Workshop on Embedded Systems*. VAMK, VAMK, University of Applied Sciences, 2007.

[92] Simon J. Julier and Jeffrey K. Uhlmann. A new extension of the kalman filter to nonlinear systems. In *Proc. of AeroSense: The 11th International Symposium on Aerospace/Defense Sensing*, Orlando, FL, USA, April.

[93] Kemal Kaplan, Buluç Çelik, Tekin Meriçli, Çetin Meriçli, and H. Levent Akın. Practical extensions to vision-based monte carlo localization methods for robot soccer domain. In Ansgar Bredenfeld, Adam Jacoff, Itsuki Noda, and Yasutake Takahashi, editors, *RoboCup 2005: Robot Soccer World Cup IX*, volume 4020 of *Lecture Notes in Computer Science*, pages 624–631. Springer, 2006.

[94] Patrick de Kok, Nicolò Girardi, Amogh Gudi, Chiel Kooijman, Georgios Methenitis, Sebastien Negrijn, Nikolaas Steenbergen, Duncan ten Velthuis, Camiel Verschoor, Auke Wiggers, and Arnoud Visser. Dutch Nao Team Team Description for RoboCup 2013 in Eindhoven, the Netherlands, 2013.

[95] Georgios K. Methenitis Amogh Gudi, Patrick de Kok and Nikolaas Steenbergen. Feature detection and localization for the robocup soccer spl. Technical report, Universiteit van Amsterdam, 2013.

[96] F. Previtali, G. Gemignani, L. Iocchi, and D. Nardi. SPQR RoboCup 2013 Standard Platform League Team Description Paper, 2013.

[97] J. M. Yáñez, L. Leottau, P. Cano, M. Mattamala, W. Celedón, M. Silva, C. Silva, P. Saavedra, P. Miranda, Y. Tsutsumi, and J. Ruiz del Solar. UChile Robotics Team, Team Description Paper RoboCup 2013 - Standard Platform League, 2013.

[98] Eric Chown, Elizabeth Mamantov, Wils Dawson, Edward Googins, Benjamin Mende, Josh Zalinger, Josh Imhoff, Brian Jacobel, Ellis Ratner, and Daniel Zeller. The Northern Bites 2013 Standard Platform League Team, 2013.

[99] Axel Heßler, Yuan Xu, Erdene-Ochir Tuguldur, and Martin Berger. DAInamite Team Description for RoboCup 2013, 2013.

[100] Gregor Jochmann, Sören Kerner, Stefan Tasse, and Oliver Urbann. Efficient multi-hypotheses unscented kalman filtering for robust localization. In Thomas Röfer, N. Michael Mayer, Jesus Savage, and Uluç Saranlı, editors, *RoboCup 2011: Robot Soccer World Cup XV*, volume 7416 of *Lecture Notes in Computer Science*, pages 222–233. Springer, 2012.

[101] Michael J. Quinlan and Richard H. Middleton. Multiple model kalman filters: A localization technique for robocup soccer. In Jacky Baltes, Michail G. Lagoudakis, Tadashi Naruse, and SaeedShiry Ghidary, editors, *RoboCup 2009: Robot Soccer World Cup XIII*, volume 5949 of *Lecture Notes in Computer Science*, pages 276–287. Springer, 2010.

[102] Aris Valtazanos, Efstathios Vafeias, Alejandro Bordallo Mico, Daniel Mankowitz, Nantas Nardelli, Subramanian Ramamoorthy, and Sethu Vijayakumar. Team Edinferno Description Paper for RoboCup 2013 SPL, 2013.

[103] Nikolaos Kargas, Nikolaos Kofinas, Evangelos Michelioudakis, Nikolaos Pavlakis, Stylianos Piperakis, Nikolaos I. Spanoudakis, and Michail G. Lagoudakis. Kouretes 2013 SPL Team Description Paper, 2013.

[104] Markus Meissner, Holger Friedrich, Andreas Fürtig, Tobias Weis, Jens-Michael Siegl, Christian Becker, Vincent Michalski, Gerhard Ruscher, Andreas Kehlenbach, Anna Reckers, Konrad Zacharias, Eric Hanssen, and Alexander Heun. Bembelbots Frankfurt Team Description for RoboCup 2013, 2013.

[105] Thomas Whelan, Sonja Stüdli, John McDonald, and Richard H. Middleton. Line point registration: A technique for enhancing robot localization in a soccer environment. In Thomas Röfer, N. Michael Mayer, Jesus Savage, and Uluç Saranlı, editors, *RoboCup 2011: Robot Soccer World Cup XV*, volume 7416 of *Lecture Notes in Computer Science*, pages 258–269. Springer, 2012.

[106] Thomas Whelan. RoboEireann Team Description for RoboCup Standard Platform League 2013, 2013.

[107] Thomas Whelan, Sonja Stüdli, John McDonald, and Richard H. Middleton. Efficient localization for robot soccer using pattern matching. In Reiner Hähnle, Jens Knoop, Tiziana Margaria, Dietmar Schreiner, and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification, and Validation*, Communications in Computer and Information Science, pages 16–30. Springer, 2012.

[108] Beth Crane, Richard Hua, Jack Murray, Dan Padiha, Stephen Sheratt, Calvin Tam, Alexander Whillas, Jayen Ashar, Sean Harris, Brad Hall, Bernhard Hengst, Maurice Pagnucco, and Claude Sammut. Team rUNSWift RoboCup 2013 Standard Platform League, 2013.

[109] Peter Anderson, Youssef Hunter, and Bernhard Hengst. An icp inspired inverse sensor model with unknown data association. In *IEEE International Conference on Robotics and Automation*, pages 2713–2718, Karlsruhe, May 2013. IEEE.

[110] Verena Hafner, Hans-Dieter Burkhard, Heinrich Mellmann, Thomas Krause, Marcus Scheunemann, Claas-Norman Ritter, and Paul Schütte. Berlin United - NaoTH 2013, 2013.

[111] Heinrich Mellmann. Active landmark selection for vision-based self-localization. In *Proceedings of the Workshop on Concurrency, Specification, and Programming CS&P 2009*, volume 2, pages 398–405, 2009.

[112] H. Mellmann, M. Jüngel, and M. Spranger. Using reference objects to improve vision-based bearing measurements. In *IEEE International conference on Intelligent Robots and Systems*, pages 3939–3945, Nice, France, September 2008.

[113] Chieh-Chih Wang, Chun-Hua Chang, Chun-Kai Chang, Bang-Cheng Wang, and Kun-Li Lin. NTU RoboPAL Team Description for RoboCup 2013, 2013.

[114] Chun-Hua Chang, Shao-Chen Wang, and Chieh-Chih Wang. Vision-based cooperative simultaneous localization and tracking. In *IEEE International Conference on Robotics and Automation*, pages 5191–5197, Shangai, May 2011. IEEE.

[115] H. Levent Akın, Barıs Gökçe, Ergin Özkucur, and Okan Asık. Cerberus'13 Team Description Paper, 2013.

[116] Thomas Röfer, Tim Laue, Judith Müller, Alexander Fabisch, Fynn Feldpausch, Katharina Gillmann, Colin Graf, Thijs Jeffry de Haas, Alexander Härtl, Arne Humann, Daniel Honsel, Philipp Kastner, Tobias Kastner, Carsten Könemann, Benjamin Markowsky, Ole Jan Lars Rieman, and Felix Wenk. B-human team report and code release 2011. Technical report, Universität Bremen, 2011.

[117] B. Cunha, J.L. Azevedo, N. Lau, and L. 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 Artificial Intelligence*, pages 417–424. Springer, 2008.

[118] A.J.R. Neves, D.A. Martins, and A.J. Pinho. A hybrid vision system for soccer robots using radial search lines. In Luís Seabra Lopes, Filipe Silva, and Vítor Santos, editors, *Proc. of the 8th Conference on Autonomous Robot Systems and Competitions, Portuguese Robotics Open - Robótica 2008*, pages 51–55, Aveiro, Portugal, April 2008.

[119] Alina Trifan, António J. R. Neves, Bernardo Cunha, and José Luís Azevedo. UAVision: A modular time-constrained vision library for soccer robots. In *RoboCup 2014: Robot Soccer World Cup XVIII*, Lecture Notes in Artificial Intelligence, pages in–press. Springer, 2014.

[120] Martin Lauer. Ego-motion estimation and collision detection for omnidirectional robots. In Gerhard Lakemeyer, Elizabeth Sklar, Domenico G. Sorrenti, and Tomoichi Takahashi, editors, *RoboCup 2006: Robot Soccer World Cup X*, volume 4434 of *Lecture Notes in Computer Science*, pages 466–473. Springer, 2006.

[121] João Silva, Nuno Lau, João Rodrigues, and José Luís Azevedo. Ball sensor fusion and ball interception behaviours for a robotic soccer team. In *IROBOT 2008 3rd International Workshop on Intelligent Robotics*, pages 25–36, Lisbon, Portugal, October 2008.

[122] João Silva, Nuno Lau, João Rodrigues, José Luís Azevedo, and António J. R. Neves. Sensor and information fusion applied to a robotic soccer team. In *RoboCup 2009: Robot Soccer World Cup XIII*, volume 5949 of *LNAI*, pages 366–377, Graz, Austria, February 2010. Springer.

[123] H.J. Motulsky and A. Christopoulos. *Fitting models to biological data using linear and nonlinear regression.* GraphPad Software Inc., 2003.

[124] Martin Lauer, Sascha Lange, and Martin Riedmiller. Modeling moving objects in a dynamically changing robot application. In Ulrich Furbach, editor, *KI 2005: Advances in Artificial Intelligence*, volume 3698 of *Lecture Notes in Computer Science*, pages 291–303. Springer, 2005.

[125] Scott Lenser and Manuela Veloso. Sensor resetting localization for poorly modelled mobile robots. In *IEEE International Conference on Robotics and Automation*, volume 2, pages 1225–1232, San Francisco, CA, April 2000. IEEE.

[126] D. A. Martins, A. J. R. Neves, and A. J. Pinho. Real-time generic ball recognition in RoboCup domain. In *Proc. of the 3rd International Workshop on Intelligent Robotics, IROBOT 2008*, pages 37–48, Lisbon, Portugal, October 2008.

[127] A. Voigtlrande, S. Lange, M. Lauer, and M. Riedmiller. Real-time 3D ball recognition using perspective and catadioptric cameras. In *Proc. of the 3rd European Conference on Mobile Robots*, Freiburg, Germany, September 2007.

[128] Avinash Burla. 3D ball detection in robocup. Master's thesis, University of Stuttgart, 2007.

[129] S. Mitri, S. Frintrop, K. Pervolz, H. Surmann, and A. Nuchter. Robust object detection at regions of interest with an application in ball recognition. In *Proc. of the 2005 IEEE*

International Conference on Robotics and Automation, ICRA 2005, pages 125–130, Barcelona, Spain, April 2005.

[130] R. Hanek, T. Schmitt, and S. Buck. Fast image-based object localization in natural scenes. In *Proc. of the 2002 IEEE/RSJ Int. Conference on Intelligent Robotics and Systems*, pages 116–122, Lausanne, Switzerland, October 2002.

[131] A. Treptow and A. Zell. Real-time object tracking for soccer-robots without color information. *Robotics and Autonomous Systems*, 48(1):41–48, August 2004.

[132] S. Mitri, K. Pervolz, H. Surmann, and A. Nuchter. Fast color independent ball detection for mobile robots. In *Proc. of the 2004 IEEE Int. Conference on Mechatronics and Robotics*, pages 900–905, Aechen, Germany, September 2004.

[133] H. Lu, Z. Zheng, F. Liu, and X. Wang. A robust object recognition method for soccer robots. In *Proc. of the 7th World Congress on Intelligent Control and Automation*, Chongqing, China, June 2008.

[134] João Silva, Nuno Lau, and António J. R. Neves. Ball identification in the robocup middle size league. In *Proc RecPad2010*, Vila Real, Portugal, October 2010.

[135] João Silva, Mário Antunes, Nuno Lau, António J. R. Neves, and Luís Seabra Lopes. Aerial ball detection in robocup middle size league using polar histograms. In *Proc RecPad2011*, Porto, Portugal, October 2011.

[136] João Silva, Mário Antunes, Nuno Lau, António J. R. Neves, and Luís Seabra Lopes. Aerial ball perception based on the use of a single perspective camera. In *16th Portuguese Conference on Artificial Intelligence (EPIA 2013)*, volume 8154 of *LNAI*, pages 235–249, Angra do Heroísmo, Açores, Portugal, September 2013. Springer.

[137] António J.R. Neves, Armando J. Pinho, Daniel A. Martins, and Bernardo Cunha. An efficient omnidirectional vision system for soccer robots: From calibration to object detection. *Mechatronics*, 21(2):399–410, March 2011.

[138] Rui Pereira and Luís Seabra Lopes. Learning visual object categories with global descriptors and local features. In *Progress in Artificial Intelligence*, volume 5816 of *Lecture Notes in Artificial Intelligence*, pages 225–236, Aveiro, Portugal, October 2009.

[139] Mário Luís Pinto Antunes. Semantic vision agent for robotics. Master's thesis, University of Aveiro, 2011.

[140] Paulo Dias, João Silva, Rafael Castro, and António J. R. Neves. Detection of aerial balls using a kinect sensor. In *RoboCup 2014: Robot Soccer World Cup XVIII*, LNAI, pages in–press. Springer, 2014.

[141] Nicolas Burrus. Kinect calibration, consulted in 2013/2014.

[142] A.J.R. Neves, G. Corrente, and A.J. Pinho. An omnidirectional vision system for soccer robots. In José Neves, Manuel Filipe Santos, and José Manuel Machado, editors, *Progress in Artificial Intelligence*, volume 4874 of *Lecture Notes in Artificial Intelligence*, pages 499–507. Springer, 2007.

[143] João Silva, Nuno Lau, António J. R. Neves, João Rodrigues, and José Luís Azevedo. Obstacle detection, identification and sharing on a robotic soccer team. In *Portuguese Conference on Artificial Intelligence (EPIA)*, volume 5816 of *LNAI*, pages 350–360, Aveiro, Portugal, October 2009. Springer.

[144] João Silva, António J. R. Neves, and Nuno Lau. Identifying obstacles in RoboCup Middle Size League. In *Proc RecPad2009*, Aveiro, Portugal, November 2009.

[145] João Silva, Nuno Lau, and António J. R. Neves. Cooperative detection and identification of obstacles in a robotic soccer team. In Calin Ciufudean and Lino García, editors, *Advances in Robotics - Modeling, Control and Applications*, pages 219–235. iConcept Press, January 2013.

[146] Pedro Fonseca, António J. R. Neves, José Luís Azevedo, and João Silva. An heuristic for trajectory generation in mobile robotics. In *Emerging Technologies & Factory Automation (ETFA)*, pages 1–4, Toulose, France, September 2011. IEEE.

[147] Nima Shafii, Luís Paulo Reis, and Nuno Lau. Biped walking using coronal and sagittal movements based on truncated fourier series. In Javier Ruiz-del Solar, Eric Chown, and PaulG. Plöger, editors, *RoboCup 2010: Robot Soccer World Cup XIV*, volume 6556 of *Lecture Notes in Computer Science*, pages 324–335. Springer, 2011.

[148] Edgar Domingues, Nuno Lau, Bruno Pimentel, Nima Shafii, Luís Paulo Reis, and António J. R. Neves. Humanoid behaviors: From simulation to a real robot. In Luis Antunes and H. Sofia Pinto, editors, *Portuguese Conference on Artificial Intelligence (EPIA)*, volume 7026 of *Lecture Notes in Artificial Intelligence*, pages 352–364. Springer, 2011.

[149] Hugo Picado, Marcos Gestal, Nuno Lau, Luis Reis, and Ana Tomé. Automatic generation of biped walk behavior using genetic algorithms. In Joan Cabestany, Francisco Sandoval, Alberto Prieto, and Juan Corchado, editors, *Bio-Inspired Systems: Computational and Ambient Intelligence*, volume 5517 of *Lecture Notes in Computer Science*, pages 805–812. Springer Berlin / Heidelberg, 2009.

[150] João Silva, Nuno Lau, and António J. R. Neves. Estimating world coordinates in perspective vision systems for humanoid robots. In *Proc RecPad2012*, Coimbra, Portugal, October 2012.

[151] D. A. Martins, A. J. Neves, and A. J. Pinho. Obtaining the distance map for perspective vision systems. In *Proc. of the 2nd ECCOMAS Thematic Conference on Computational Vision and Medical Image Processing, VipIMAGE 2009*, Porto, Portugal, 2009.