Universidade de Aveiro Departamento de Electrónica, Telecomunicações e Informática,
2008

João Manuel Leite da Silva

Fusão sensorial e comportamentos na equipa de Futebol Robótico CAMBADA

Sensor fusion and behaviours for the CAMBADA Robotic Soccer Team

Universidade de Aveiro Departamento de Electrónica, Telecomunicações e Informática,
2008

João Manuel Leite da Silva

Fusão sensorial e comportamentos na equipa de Futebol Robótico CAMBADA

Sensor fusion and behaviours for the CAMBADA Robotic Soccer Team

Dissertação apresentada à Universidade de Aveiro, para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica de Prof. Doutor Nuno Lau e Prof. Doutor João Rodrigues, professores auxiliares do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro.

o júri

presidente

Doutor António Rui de Oliveira e Silva Borges
professor associado da Universidade de Aveiro

Doutor Armando Jorge Miranda de Sousa
professor auxiliar da Faculdade de Engenharia da Universidade do Porto

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

Doutor João Manuel de Oliveira e Silva Rodrigues
professor auxiliar da Universidade de Aveiro

acknowledgements / agradecimentos

For the time working in the CAMBADA project, i thank all the team members, both students and professors, for the good working environment, support and ideas. I paticularly thank professors Nuno Lau and João Rodrigues for the provided orientation and availability whenever a doubt came up.
For all the time over the years, i thank my parents for the support and opportunities provided, my friends for all they endured from me, and Alexandra for the patience, support and inspiration.

Pelo tempo a trabalhar no projecto CAMBADA, agradeço a todos os membros da equipa, tanto alunos como professores, pelo bom ambiente de trabalho, suporte e ideias. Agradeço particularmente aos professores Nuno Lau e João Rodrigues pela orientação dada e disponibilidade sempre que uma dúvida surgia.
Por todo o tempo ao longo dos anos, agradeço aos meus pais pelo apoio e oportunidades fornecidas, aos meus amigos por tudo o que aturaram de mim, e à Alexandra, pela paciência, apoio e inspiração.

Abstract

Sensor fusion and behaviours are two of the modules required in the implementation of software agents able to play soccer. CAMBADA is the MSL RoboCup robotic soccer team created by the ATRI group, part of the IEETA research unit at Aveiro University.

This thesis provides an overview of the CAMBADA team architecture over which the described work was implemented. The motion of the robot is a low level control problem, in this case solved by a PID controller; an overview on low level control is presented in this document.

The CAMBADA team robots are completely autonomous and thus they possess their own perception sensors. The information provided by those sensors is raw and has to be processed to provide better quality information for the agent. Sensor fusion techniques provide the means to achieve this information enhancement and some are discussed in this document. An implementation of a Kalman filter was created and tested to estimate the ball position from the noisy measurements and to detect changes on the ball path, based on a comparison between predicted values and measured values. Also, a linear regression was implemented for estimation of the ball and robot velocities. Due to changes in the MSL rules, that turned the field symmetric from the vision point of view, a new electronic compass was integrated providing a means to verify the results of the position tracking algorithm and hence enhance localisation.

The information resulting from the sensor fusion is kept in a description of the state of the world used by the robot. Some developments were made in this world state representation. Algorithms were created and implemented to allow the agent to check for a set of conditions that are used by the high level decision module.

The developments on sensor fusion and world state representation supported the implementation of new behaviours. The behaviours define “reactions” to a set of conditions, that can be verified through the information of the state of the world, and are the responsibles for defining the commands to be sent down to the low level controllers of the actuators. The intelligent combination of these behaviours allows the robot to act in a defined way on the field. Work at the behaviour level provides better action capabilities, important for the development of effective game strategies. Two new interception behaviours were implemented that allow the robot to intercept the ball by reasoning over its path and the robot capabilities. Also a new algorithm for ball avoidance was developed, for situations like kickoff, throwin and other situations when the game is stopped. In these situations the robots have to reposition themselves on the field without touching the ball. The created algorithm improves the performance by reducing the necessary deviation.

The velocity estimation of the ball was greatly improved and is now much more reliable in game situations. The new behaviours brought a new dynamic to the game and the tools to manipulate the state of the world provided a simplification and improved the modularity of the high level code. In a general way, the developments achieved in the work described by the thesis have improved the overall performance of the team in competition.

Resumo

Fusão sensorial e comportamentos são dois módulos necessários à implementação de agentes capazes de jogar futebol. CAMBADA é a equipa de futebol róbotico da liga média do RoboCup criada pelo grupo ATRI, pertencente à unidade de investigação IEETA da Universidade de Aveiro.

Esta tese fornece uma visão geral da arquitectura da equipa CAMBADA sobre a qual o trabalho descrito foi implementado. O movimento dos robôs é um problema de controlo de baixo nível, resolvido por um controlador PID; uma descrição geral sobre controlo é apresentada neste documento.

Os robôs da equipa CAMBADA são completamente autónomos e portanto possuem os seus próprios meios sensoriais. A informação fornecida por esses sensores não é tratada e tem que ser processada para fornecer informação de melhor qualidade para o agente. Técnicas de fusão sensorial fornecem meios para obter esta melhoria da informação e algumas são discutidas neste documento. Uma implementação de filtro de Kalman foi criada e testada para estimar a posição da bola a partir das medidas ruidosas e para detectar mudanças no caminho da bola, baseada na comparação entre valores previstos e valores medidos. Também foi implementada uma regressão linear para estimar as velocidades da bola e robô. Devido a mudanças nas regras da liga média, que tornaram o campo visualmente simétrico, uma nova bússola electrónica foi integrada para providenciar um meio de verificar os resultados do algoritmo de estimação da posição, melhorando a localização.

A informação proveniente da fusão sensorial é mantida numa descrição do estado do mundo usada pelo robô. Alguns desenvolvimentos foram feitos nesta representação. Foram criados e implementados algoritmos que permitem ao agente testar um conjunto de condições que são usadas pelo módulo de decisão de alto nível.

Os desenvolvimentos na fusão sensorial e representação do estado do mundo permitiram a implementação de novos comportamentos. Os comportamentos definem "reacções" a um conjunto de condições, que podem ser verificadas a partir da informação do estado do mundo, e são responsáveis por definir os comandos a enviar para os controladores dos actuadores. A combinação inteligente destes comportamentos permitem ao robô agir de uma determinada forma em campo. Trabalho ao nível dos comportamentos fornecem melhores capacidades de acção, importantes para o desenvolvimento de estratégias de jogo eficazes. Dois novos comportamentos de intercepção foram implementados que permitem ao robô interceptar a bola avaliando o seu caminho e as capacidades do robô. Também foi desenvolvido um novo algoritmo para desvio da bola, para situações de pontapé de saída, lançamento e outras situações de bola parada. Nestas situações, os robôs têm que se reposicionar no campo sem tocar na bola. O algoritmo criado melhora o desempenho reduzindo o desvio necessário.

A estimativa da velocidade da bola foi muito melhorada e é agora muito mais fiável em situações de jogo. Os novos comportamentos trouxeram uma nova dinâmica ao jogo e as ferramentas de manipulação do estado do mundo simplificaram e melhoraram a modularidade do código do alto nível. De uma forma geral, os desenvolvimentos obtidos pelo trabalho descrito nesta tese melhoraram o desempenho geral da equipa em competição.

Contents

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Objectives . . . 1
- 1.3 RoboCup . . . 2
  - 1.3.1 Middle Size League (MSL) . . . 3
- 1.4 Thesis structure . . . 3

2 Low-level control **5**
- 2.1 Open-loop control . . . 5
- 2.2 Closed-loop control . . . 6
- 2.3 PID controller . . . 7
- 2.4 Methodologies and approaches on control . . . 8
- 2.5 Summary . . . 11

3 Sensor and information fusion **13**
- 3.1 Linear regression . . . 14
- 3.2 Kalman filter . . . 15
- 3.3 Particle filter . . . 17
- 3.4 Mapping and localisation . . . 18
  - 3.4.1 Mapping . . . 18
  - 3.4.2 Localisation . . . 19
- 3.5 Summary . . . 20

4 CAMBADA **21**
- 4.1 Common structural approaches of MSL teams . . . 21
- 4.2 CAMBADA Robots . . . 22
  - 4.2.1 Physical structure . . . 22
  - 4.2.2 General architecture . . . 22
- 4.3 CAMBADA vision overview . . . 25
- 4.4 CAMBADA agent overview . . . 26
- 4.5 Localisation . . . 27
- 4.6 Summary . . . 29

5 World state **31**
- 5.1 Organisation . . . 31
  - 5.1.1 Coordinate systems . . . 31
  - 5.1.2 Data structures . . . 33

5.2 Methods and algorithms 35
5.2.1 Transformation methods 35
5.2.2 Basic queries (getters) 36
5.2.3 Advanced queries and utilities 37
5.2.4 Interception point algorithms 42
5.3 Summary 42

6 Integration 43
6.1 Organisation 43
6.2 Ball position filter 45
6.2.1 Model 45
6.2.2 Initialisation 46
6.2.3 Implementation 48
6.2.4 Results 49
6.3 Velocity estimation 57
6.3.1 Model 58
6.3.2 Implementation 58
6.3.3 Results 59
6.4 Compass 62
6.4.1 Results 63
6.5 Ball position integration 64
6.5.1 Integration using self visual information 64
6.5.2 Integration using multi robot information 65
6.6 Summary 66

7 Behaviours 67
7.1 Passive interception 68
7.1.1 Results 69
7.2 Active interception 71
7.2.1 Uniform movement equations approach 71
7.2.2 Iterative approach 72
7.2.3 Interception point correction 73
7.2.4 Results 74
7.3 Velocity direction correction on interceptions 75
7.3.1 Results 77
7.4 Catchball (pass reception) 79
7.4.1 Results 79
7.5 Ball avoidance 79
7.6 Summary 81

8 Conclusion and future work 83
8.1 Conclusion 83
8.2 Future work 84

A Velocity estimation choice 85

List of Figures

1.1 Picture of a Robótica2008 game. 3
1.2 MSL field markers 4

2.1 Open-loop control system. 5
2.2 Closed-loop control system. 6
2.3 PID controller in a closed-loop process. 7

3.1 Metric map 19
3.2 Landmark map 20

4.1 Pictures of CAMBADA robot parts (1) 22
4.2 Pictures of CAMBADA robot parts (2) 23
4.3 Hardware architecture diagram. 24
4.4 Software architecture diagram. 25
4.5 Vision systems frames 26
4.6 Relation between pixel and metric distances. 27

5.1 Field coordinate system (absolute coordinates). 32
5.2 Robot coordinate system (relative coordinates). 32
5.3 Coordinate transformation illustration 33
5.4 Class diagram for the field objects. 35
5.5 *shootInGoal* illustration 38
5.6 *isBallInDangerZone* illustration 39
5.7 *approachPoint* illustration 40
5.8 *isLineClear* illustration 41
5.9 *isSliceClear* illustration 42

6.1 Integrator functionality diagram. 45
6.2 Effects of Q on Kalman filter 47
6.3 Ball path deviation illustration 50
6.4 Kalman theoretical result 50
6.5 Noisy position of a static ball taken from a static robot. 51
6.6 Noisy position of a static ball taken from rotating robots. 53
6.7 Polynomials to fit the measurements standard deviation 54
6.8 Kalman results compare (1.1) 55
6.9 Kalman results compare (1.2) 56
6.10 Kalman results compare 2 57
6.11 Velocity representation using consecutive measures displacement. 60

6.12 Velocity representation using linear regression over direct position measures. . 61
6.13 Velocity representation using linear regression over Kalman filtered positions. 61
6.14 Compass error areas 62
6.15 Orientation results 63
6.16 Orientation results with relocation 64
6.17 Ball integration activity diagram. 66

7.1 Structure of the behaviour classes. 67
7.2 Movement to ball 68
7.3 Passive interception algorithm illustration 69
7.4 Passive interception illustration 70
7.5 Iterative interception point illustration 73
7.6 Interception over line illustration 74
7.7 Active interception illustration 75
7.8 Velocity correction illustration 76
7.9 Movement of robot to a static ball without velocity correction 77
7.10 Movement of robot to a static ball with velocity correction 78
7.11 CatchBall illustration 80
7.12 Old ball avoidance algorithm 80
7.13 New ball avoidance algorithm 81

A.1 Linear regression estimated velocity 85
A.2 Internal Kalman filter estimated velocity 86
A.3 Comparison between regression and Kalman velocities 87

List of Tables

3.1 Common names for linear regression variables 14

6.1 Rotating speed - distance relationship 52
6.2 Distance-standard deviation relationship 52

Chapter 1

Introduction

1.1 Motivation

CAMBADA, *Cooperative Autonomous Mobile roBots with Advanced Distributed Architecture*, is the Middle Size League Robotic Soccer team from Aveiro University. The project started in 2003, coordinated by the IEETA[1] ATRI[2] group. It involves people working on 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.
Since its creation, the team has participated in several competitions, both national and international. Each year, new challenges are presented, and new objectives are defined, always with a better team performance in sight. After achieving the 1st place in the national competition Robótica2007[3] and the 5th place in the world championship RoboCup2007[4], the wish for even better results is more than ever present. To achieve this, along with the necessity to develop the high-level decision comes the necessity for a stable and reliable description of the state of the world, capable of integrating the available information in a structured and easy to access manner. This description includes information about positions of field objects, like the robot itself and its team mates, the ball, the goals and detected obstacles, information of the velocity of the mobile objects, robot's internal state an other field information. Also necessary for the high level improvement is the development of new behaviours implemented under the decision layer.

1.2 Objectives

The objectives of this project include refinement and correction of existing behaviours, as well as the proposal and implementations of new ones that may become necessary. All of them should be included in a library integrated in the CAMBADA code structure. Besides the work on the behaviour level, it is also an objective the refinement and implementation of information integration techniques in the worldstate, as well as the development of methods to manipulate this worldstate. All the development should be done aiming at the increase of the team performance and according to the strategic needs defined by the work team.

---

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

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

[3] http://www.ccvalg.pt/robotica2007/en/home.htm

[4] http://www.robocup-us.org/Old/robocup-2007/

1.3 RoboCup

RoboCup[5] is an international joint project to promote artificial intelligence, robotics and related fields. It is a project in constant development and each year innovations can and probably will appear. Presently, RoboCup[6] includes several leagues:

- RoboCupJunior
    - Junior Dance
    - Junior Soccer
    - Junior Rescue
    - Junior Demo
- RoboCup@Home
- RoboCupRescue
    - Rescue Simulation
    - Rescue Robot
- RoboCupSoccer
    - Simulation
    - Small Size Robot
    - Middle Size Robot
    - Standard Platform
    - Humanoid

Each existing league has a different approach, some only at software level, others at hardware, with single or multiple agents, cooperative or competitive. The CAMBADA team was created to participate in the middle size league of robotic soccer, which requires the integration of several technologies in order for the robots to play soccer, such as autonomous agents architectures, multi-agent cooperation, strategy acquisition, real-time reasoning, control and sensor fusion. The long term objective of the soccer leagues is:

> *By the year 2050, develop a team of fully autonomous humanoid robots that can win against the human world soccer champion team.*

---

[5] http://www.robocup.org/

[6] http://www.robocup-cn.org/

Figure 1.1: Picture of a Robótica2008 game.

1.3.1 Middle Size League (MSL)

In this league, each team is composed of 4 to 6 robots with maximum size of 50x50cm base, 80cm height and a maximum weight of 40Kg, playing in a field of 18x12m. The rules [1] of the game are similar to the official FIFA rules, with required changes to adapt for the playing robots. Each robot is autonomous and has its own sensorial means. They can communicate among them through a wireless network and with an external computer acting as a coach. This coach computer has no sensor of any kind, it only knows what is reported by the playing robots. The agents should be able to evaluate the state of the world and make decisions suitable to fulfil the cooperative team objective. Some of the elements in the field have well defined colour, such as the field itself that has green colour, the white coloured lines, the ball which is still orange in this 2008 edition and the robots which are mainly black. However, these restrictions tend to diminish with the advance of technologies and development of the teams, as is the case of the goal colours, which were removed since the last year's edition. The ball colour is another example of this, being the restriction of orange colour a restriction to be dropped in a near future. The official field markings are presented in figure 1.2, although the field itself is not in the official proportions.

1.4 Thesis structure

The remainder of this document is structured in other 7 chapters.
Chapter 2 gives a brief insight of existing low-level control techniques, with special emphasis on the PID controller, used by the team for the low-level control. Chapter 3 presents the definition of sensor fusion and presents some of the most used techniques, like linear regression, Kalman and particle filtering and mapping and localisation issues. In chapter 4, more specific content about the project is presented, by describing the CAMBADA robots at all levels, with brief description of existing work. Chapter 5 presents the structure of the state of the world built by the CAMBADA robots, its main characteristics and new tools and algorithms. In chapter 6 a description of the work done at the information integration level for the

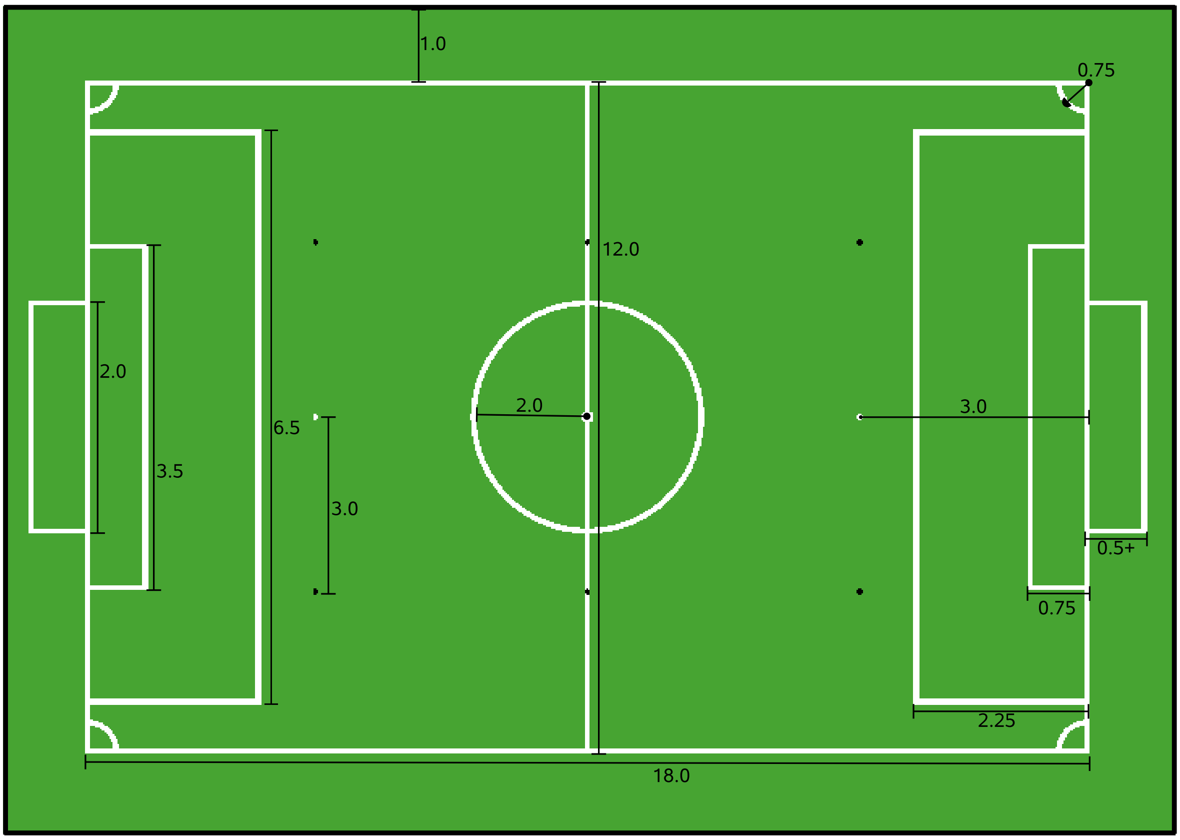


Figure 1.2: Official MSL field marker distances. All the presented values are in metes. The field representation is **not** in official proportions.

construction of the state of the world is presented. On a higher level, chapter 7 presents the developed work at the robot behaviour level, both new behaviours and refinement of existing ones, with new algorithms. Finally in chapter 8 some comments on the achieved results are presented as conclusion statements as well as an overview on pending issues and proposals for future work.

Chapter 2

Low-level control

Control is defined as the ability to change a system's status to a given value. A quotidian life example is the control of an automobile speed with the use of accelerator and brake pedals. In engineering, a control system has at least two parts. The *controller* and the object to be controlled, the *plant*. These elements can be configured with or without feedback.

2.1 Open-loop control

An open-loop control configuration has no feedback on the system.

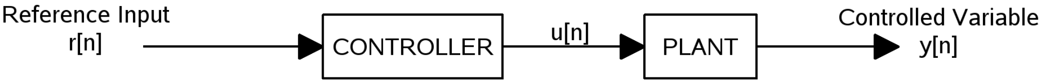


Figure 2.1: Open-loop control system.

When using this configuration, one depends on a very good knowledge and accurate models on the plant and environment. Let us consider a motor [2], with an open-loop control configuration (the motor is the plant in this case). We must have an accurate model of the motor. In this case, an increase of the voltage applied to the motor causes an increase of the its speed. The reference input is then the desired speed and the controller is the one that chooses the voltage necessary to apply to the plant.

However, an open-loop configuration is greatly affected by disturbances. In the motor case, a typical disturbance is the batteries not being fully charged. In that case, the voltage input would be affected and the plant model would not create the expected output. Given a controller gain K_c, the controller output based on the reference input $r[n]$ would be [3]

$$u[n] = K_c\ r[n] \tag{2.1}$$

and the plant output would be

$$y[n] = K_P\ u[n] \tag{2.2}$$

where K_P represents the gain of the plant. In the motor case, the controller output $u[n]$ is the to be applied to the motor and the output variable $y[n]$ is the speed of the motor. Combining 2.1 and 2.2 we get

$$y[n] = K_c\ K_P\ r[n] = K\ r[n]$$

which denotes that the controlled output is proportional to the desired value $r[n]$. If the motor properties change by 1%, we have $K' = 1.01\ K$ and

$$y'[n] = K'\ u[n] = 1.01\ y[n]$$

which shows that a 1% error in the plant characteristics produce a 1% error in the controller variable, demonstrating the need for a precise mathematical model of the plant.

2.2 Closed-loop control

Closed-loop control is another possible configuration for a controller. In this configuration, two other components are considered: a *feedback* system and an adder.

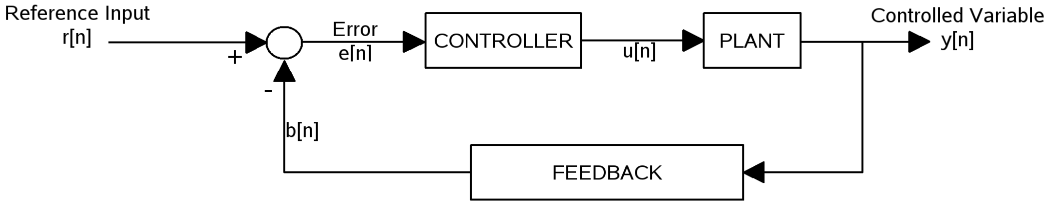


Figure 2.2: Closed-loop control system.

The controlled variable y is measured by some sensor and compared with the desired value for this variable. The difference given by the adding of the negative measurement from the desired reference value gives the *error* that, after being processed by the *controller*, originates the signal that controls the plant (hence the term "negative feedback"). In the motor example [2], a speed sensor would measure the actual speed of the motor that would be compared with the desired one. The speed would then be increased until the error reached zero.

Assuming the same models as before, the controller output would now be [3]

$$u[n] = K_c\ e[n] = K_c[r[n] - b[n]] \tag{2.3}$$

where the feedback signal $b[n]$ is related to the output by

$$b[n] = K_f\ y[n] \tag{2.4}$$

being K_f the feedback gain, that relates the measurement units with the input units. Combining 2.3, 2.4 and 2.2, we get

$$\begin{aligned} & y[n] = K_P\ K_c[r[n] - K_f\ y[n]] \\ \Leftrightarrow\quad & y[n] = \frac{K_P\ K_c}{1 + K_P\ K_c\ K_f} r[n] \end{aligned}$$

which indicates that the output in a closed-loop configuration is related with the reference input by a ratio with the numerator being the product of all the forward-path gains in figure 2.2 and denominator being one plus all the gains in the loop. We can see that the effect of a change in the plant gain on the output is now also influenced by the controller and feedback gains K_c and K_f. Assuming the same 1% change in the plant gain ($1.01K_P$) and dividing both the numerator and denominator by K_c we get

$$y[n] = \frac{1.01K_P}{\frac{1}{K_c} + 1.01K_f\ K_P} r[n].$$

If the controller gain is large, the term $\frac{1}{K_c}$ in the denominator will become negligibly small and the output becomes

$$\begin{aligned} y[n] &\approx \frac{1.01K_P}{1.01K_f\ K_P} r[n] \\ &= \frac{1}{K_f} r[n] \end{aligned}$$

This indicates that in the presence of feedback, the output is nearly independent of the plant gain K_P, the approximation depending on the controller gain. This means that a closed-loop system is *less sensitive* to plant parameter variations and disturbances (noise) than the open-loop.

Despite the need for additional components, namely, a way of measuring the output and a feedback system, this "insensitivity" of closed-loop systems is advantageous in a physical situation of robotic control, since it works whatever the disturbances are, and noise is an almost certain presence.

2.3 PID controller

PID controller is a closed-loop controll method and thus attempts to correct the error between the desired reference input value and the actual output value. It is the most used controller in industrial processes [4] and in low-level control of mobile robots motors [3]. The control output u is based on a summation of three separate components which give name to the controller: P-Proportional, I-Integral and D-Derivative.

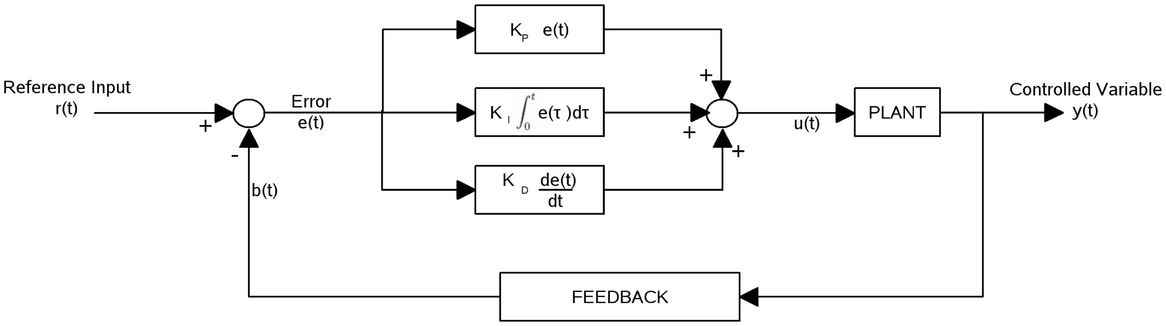


Figure 2.3: PID controller in a closed-loop process.

A standard PID controller is also known as the "three-term" controller [5] and its output is given by

$$u(t) = K_P\ e(t) + K_I \int_0^t e(\tau)\ d\tau + K_D\ \frac{de(t)}{dt}$$

where $e(t)$ is the input error signal, K_P is the proportional gain, K_I is the integral gain and K_D is the derivative gain [4]. The discrete form, usually used is given by

$$u[n] = K_P\ e[n] + K_I\ I[n] + K_D\ D[n]$$

where $e[n]$ is the discretised input error signal and K_P, K_I and K_D are the gains as in the continuous form and

$$\begin{aligned} I[n] &= I[n-1] + T\ e[n] \\ D[n] &= \frac{e[n] - e[n-1]}{T} \end{aligned}$$

being T the time interval between samples.

The proportional term provides an overall control action proportional to the error. A larger K_P means faster response since the larger the error, the larger is the proportional term compensation. However, an excessively large proportional gain would lead to instability and oscillation [5].

The integral term reduce steady-state errors through an integrator. It allows a finer approximation to the desired state with the cost of a larger overshoot.

The derivative term allows a reduction in the overshoot, but can slow down the response and may lead to instability. This term allows the controller to take the decreasing error into account and produce an output proportional to the slope (derivative with respect to time) of the error. If the error is positive but decreasing, the output of the controller will be smaller than it would be due to the proportional term alone. This term gives the controller some anticipation [3].

By adjusting the three gains K_P, K_I and K_D, one can obtain the desirable performance characteristics. In some situations, the integral or the derivative terms are omitted, resulting in a PD (proportional-derivative) or a PI (proportional-integral) controller.

There are some existing rules to define the gains of the PID controllers. Among them are the Ziegler-Nychols rules, which were derived empirically from the responses obtain in experiments of different industrial equipments. Through experimental tests of the equipments to compensate, these rules aim to adjust the system so that the maximum overshoot on the response to a step is 25% [4]. Another set of rules, based on the previously referered, was proposed by Chien, Hrones e Reswick, aiming for a performance increase [4].

2.4 Methodologies and approaches on control

Methodologies

Being control a large and complex field, there are several existing control methodologies. A brief description of some of these strategies is presented below.

- *Adaptive control* [3] is a methodology usually applied in systems operating in widely variable environments, where a single control design may not be suitable for all situations. Within adaptive control, there are also several possible approaches.
  As an example, an aircraft that has a good controller for sea level flights, but with poor results at high altitudes. One obvious solution to this problem would be to design several controllers suitable for each altitude range and switch among them based on the

altitude given by a proper sensor.
In applications where both the movements of the robot and it's operation environment are known, it is possible to program controller gains as a function of time, known as *programmed gain controllers*.

  - The gain switching approach is a way of compensating the robot's dynamic as a result of load or environment changes. This suggests that if changes in the robot's dynamic could be tracked, one could introduce compensatory changes in the controller, resulting in a "learning model" adaptive approach. The "learning model" receives both the input and the output of the plant. If it contains a mathematical model that represents the plant at initial time, then it can adjust its parameters to track changes in the plant and adjust the controller based on these parameters.
  - Another approach to obtain the information needed to adjust the controller as the environment changes is based on a "performance evaluation" measure, and adjust the controller parameters only when the changes in the environment affect the system performance.
  - A third method of adaptation is based on the idea that we often expect the system to behave in a certain way and variations on these behaviours are not acceptable. A mathematical model that produces the desirable behaviour is set as the "reference model" for the system. By applying the same inputs to both the reference model and the closed-loop system and comparing them, if the responses differ, the information can be used to adjust the controller.

- *Optimal control* is another methodology that aims to design controllers that perform in the best possible manner, with respect to a given performance criteria. This approach is based on mathematical optimisation methods for deriving control policies. The Pontryagin Maximum Principle and its variants are a base for this approach. It is usually more suitable to linear systems and quadratic criteria [3].

- *Fuzzy control* is a control methodology based on fuzzy logic. It allows that complex requirements may be implemented in simple, easily maintained and inexpensive controllers. The fuzzy technology can provide decision support and powerfull reasoning capabilities bound by a minimum of rules. They are mainly used in systems with continuous phenomena that are not easily broken down into discrete segments. Also when a mathematical model of the process does not exist or is too difficult to encode, or too complex to evaluate in usefull time, or involves too much memory on the chip architecture.
  An example might be a controller for an automobile anti-lock breaking system. The control rules could include variables like car speed, brake pressure and brake temperature. These variables are continuous and subject to interpretation by the system designer. The temparature could have a range of states such as: cold, cool, moderate, warm, hot, very hot. Yet, the change from one state to another is not precisely defined, the idea of what is cold, warn or hot is subject to interpretation by different experts at different points in the variable's domain. This subjectivity is the heart of the power and flexibility of fuzzy logic. In the case of the braking system, a fuzzy rule might be something like antilock braking system might be: *"If brake temperature is **Warm** AND speed is **Not***

***very fast**, then brake pressure is* ***Slightly decreased**.”* while a conventional controller rule would need concrete values: *“If brake temperature is greater than 280 AND speed is less than 45, then brake pressure is 190.”*.
The fuzzy control model has similar components to conventional one, differing mainly that the fuzzy systems contain “fuzzifiers”, which convert inputs into their fuzzy representations and “defuzzifiers” which converts the output of the fuzzy logic into numerically precise solution variables, that can be used on the controlled actuators [6].

- *Intelligent control* is a methodology that can be hard to define and accept by control engineers. There are several levels of intelligence and one must understand that intelligent control methodologies does not provide the controller intelligence in its fullest possible sense, but rather a quite restrictive type of intelligence.
  In [7], a definition for intelligent control methodologies states that

  > *“a control methodology is an intelligent control methodology if it uses human-, animal-, or biologically motivated techniques and procedures (e.g, forms of representation or decision making) to develop or implement a controller for a dynamical system.”*

  Hence, the fuzzy methodology described before can be seen as a special type of intelligent control, since it uses human inherited subjectivity to represent the rules.
  Other intelligent control methodologies include expert control, learning control, use of planning systems for control, neural control (which is motivated by low-level biological representations and decision making) and the use of biologically motivated genetic algorithms.
  Many intelligent control methodologies result from a synthesis of several intelligent and conventional methodologies.
  Given the interpretation ambiguity of the presented definitions, to help make the distinction, it is emphasised that in intelligent control, the focus is on *designing controllers to emulate or perform certain functions of humans, animals or biological systems to solve control problems* [7].

State space approach

While the presented models are concentrated on the frequency domain, the *state space* approach is entirely based on time domain, making use of a *state* concept. It relies on two elements: the state vector and state equations. The system’s state is a minimum number of variables that describe the behaviour of the system given its initial conditions and inputs. The idea of state stems from the mechanics, since a dynamical system motion is determined by the evaluation of the position and velocity vectors, and thus, a mechanical system’s state is represented by these two variables (position and velocity). The idea that complete system dynamics can be represented by a moving point in a plane can be extended to n-dimensional vector spaces, without the corresponding visualisation. The notion of state can be generalised to apply to abstract general dynamical systems described by differential equations, without specific reference to a physical domain. However, the state concept requires that a mathematical model or description of the system is available [3, 8, 4].

2.5 Summary

On this chapter, a basic overview on low level control was presented. The definitions of open-loop and closed-loop control, with exposure of each one's advantages and disadvantages were discussed, and some examples of control methodologies were enumerated. A slightly deeper overview of PID controller was presented, as it is the method used to control the low level layer of the CAMBADA team.

Chapter 3

Sensor and information fusion

The definitions on sensor and information fusion are sometimes used in different ways. To avoid confusion on the meaning, the term "information fusion" was used by Dasarathy as an overall term for fusion of any kind of data [9]. As stated in [10],

> **Information fusion** *encompasses theory, techniques and tools conceived and employed for exploiting the synergy in the information acquired from multiple sources (sensor, databases, information gathered by humans, etc.) such that the resulting decision or action is in some sense better (qualitatively or quantitatively, in terms of accuracy, robustness, etc.) than would be possible if any of these sources were used individually without such synergy exploitation.*

Sensor fusion is then defined as a subset of information fusion as the act of combining sensorial data or data derived from sensorial data providing a resulting information that is better than would be possible when the sources were used individually.

The sensor fusion definition above states that sensor data or data derived from sensor data has to be combined (it is not required that the inputs are produced by multiple sensors). In the integration work described in chapter 6, we fall into the information fusion field, as well as the sensor fusion, both single and multiple sensor.

There are many methods for sensor and information fusion. Regression techniques are a way of getting more accurate estimates on data sets. An introduction to linear regression is presented in 3.1. The Kalman filter [11] is currently one of the most frequently used, being able to filter, smooth and predict data states. An introduction to it is presented in 3.2. Also, a very general description on the idea of particle filters is presented, although it was not implemented, for reasons described in 3.3. Mapping and localisation issues are one of the important areas of application of sensor fusion, and thus a brief explanation on these theme is presented in 3.4.

In the next sections, the variables present in the equations have the following notation:

ˆ for estimated values;

¯ for mean values;

Uppercase letters for values usually presented as matrices or vectors;

Lowercase letters for usually simple values;

3.1 Linear regression

Linear regression analyses the relationship between two variables, x and y [12]. These two variables names vary according to the field of application being some of the most common names presented in table 3.1. The primary reasons for fitting a regression equation to a set of data is 1) to describe the data; 2) to predict the response from the independent variable.

x-axis	**y-axis**
independent	dependent
predictor	predicted
carrier	response
input	output

Table 3.1: Common names for linear regression variables, presented in [12].

It is commonly assumed that x is the independent variable and y the dependent one. Thus, given a set of points (x,y), by linear regression we can obtain an equation of the form

$$\hat{y} = \alpha + \beta x$$

where $\hat{y}$ is the predicted value of the response obtained by using the equation. The line given by the equation should give a good fit of the data set. It is created so that the values obtained by using the line should be close to the true values, that is, the residuals should be minimised. Therefore, when assessing the fit of a line, the vertical distances of the points to the lines are the only ones that matter. Perpendicular distances are not considered because only the vertical distances, the dependent variable ones, are affected by error, the independent variable is considered error free [12, 13].

The simple linear regression equation is also called the *least squares* regression. Its name refers to the criterion used to select the best fitting line, namely that the sum of the *squares* of the residual should be minimal. That is, the least squares regression equation is the line for which the sum of squared residuals $E = \sum (y_i - \hat{y}_i)^2$ is minimum. It is not necessary to make trial and error attempts to get the best fitting line for the data set, as the line equation that minimises the sum of squared residuals is algebraically deductible. The α and β values that accomplish the minimisation of E satisfy the system

$$\left\{ \begin{array}{rcl} \frac{\partial E}{\partial \alpha} & = & 0 \\ \frac{\partial E}{\partial \beta} & = & 0 \end{array} \right. \Leftrightarrow \left\{ \begin{array}{rcl} m\,\alpha + \sum x_i\,\beta & = & \sum y_i \\ \sum x_i\,\alpha + \sum x_i^2\,\beta & = & \sum x_i y_i \end{array} \right.$$

where m is the number of elements in the data set. The expressions for which the sum of squared residuals will be minimised is given by

$$\beta = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sum (x_i - \bar{x})^2} = \frac{m \sum x_i y_i - \sum x_i \sum y_i}{\Phi} \tag{3.1}$$

$$\alpha = \bar{y} - \beta \bar{x} = \frac{\sum x_i^2 \sum y_i - \sum x_i \sum x_i y_i}{\Phi} \tag{3.2}$$

where

$$\Phi = m \sum x_i^2 - \left(\sum x_i \right)^2 \tag{3.3}$$

and all the summations are from *i=1* to *i=m*.

3.2 Kalman filter

Developed by Rudolf E. Kalman in his 1960 paper [11], the Kalman filter is an *optimal recursive data processing algorithm* that estimates the state of a dynamic system from a series of noisy measurements [14, 15]. One aspect of this optimality is that the Kalman filter processes all information that can be provided to it, regardless of their 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 and the uncertainty in the dynamics model, and any available information on initial conditions of the variables of interest [15]. It is widely used in several areas, such as statistics, aeronautics, engineering and economics. A typical example of an application would be providing accurate information about the position and velocity of a moving object given only an incomplete sequence of measures of its position, each with some error associated.

Kalman filters are based on linear dynamical systems discretised in time. It is assumed that the system and the measures are affected by *White Gaussian* noise, meaning the noise is not correlated in time, and thus we can assume that at each discrete time, the noise affecting the system and measures are independent of past or future values. The filter is a two step process: First a forecast of the output is made based on the linear evolution of the noisy dynamic system, an *a priori* estimate, and then a measurement of the state is combined with the forecast in order to produce a final *a posteriori* probabilistic estimate of the system's state [14].

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

$$X_k = F_k X_{k-1} + B_k U_{k-1} + w_{k-1}$$

where

- X_k is the real state at time *k*,
- F_k is the state transition model applied to the previous state X_{k-1}
- B_k is the control-input model which is applied to the control vector U_k
- U_k is the control vector, over which the control-input model B_k is applied
- w_k is the process noise assumed to be a zero mean Gaussian white with covariance Q_k and independent of X

  $w_k \sim N(0, Q_k)$

At time *k*, a measurement Z_k of the true state X_k is made according to a model of the system observation, the *observation model*

$$Z_k = H_k X_k + v_k \tag{3.4}$$

where H_k is the observation model that defines how the measurement variables are mapped into the true state variables and v_k is the observation noise assumed to be zero mean Gaussian white noise with covariance R_k

$v_k \sim N(0, R_k)$.

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

- $\hat{X}_k$, the estimate of the state at time k;
- $\hat{P}_k$, the error covariance matrix, a measure of estimated accuracy of the state estimate.

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

$$\text{Predicted state:} \qquad \hat{X}_{k_{predicted}} = F_k \hat{X}_{k-1} + B_k U_{k-1} \tag{3.5}$$

$$\text{Predicted estimate covariance:} \qquad \hat{P}_{k_{predicted}} = F_k \hat{P}_{k-1} F_k^T + Q_{k-1}. \tag{3.6}$$

Aiming for the optimal Kalman gain, 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

$$Y_k = Z_k - H_k \hat{X}_{k_{predicted}}.$$

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, the output state of the filter is computed by:

$$\hat{X}_k = \hat{X}_{k_{predicted}} + K_k Y_k \tag{3.7}$$

where K_k is the Kalman gain at time k.
The optimal Kalman gain can be expressed in the form:

$$K_k = \hat{P}_{k_{predicted}} H_k^T S_k^{-1} \tag{3.8}$$

being S_k the *innovation* or *residual* of the covariance matrix, taking into account the measurement noise covariance matrix R_k:

$$S_k = H_k \hat{P}_{k_{predicted}} H_k^T + R_k. \tag{3.9}$$

Writing equations 3.8 and 3.9 together

$$K_k = \frac{\hat{P}_{k_{predicted}} H_k^T}{H_k \hat{P}_{k_{predicted}} H_k^T + R_k}$$

one can easily verify that as the error covariance R approaches zero,

$$\lim_{R_k \to 0} K_k = H^{-1}.$$

On the other hand, as the *a priori* estimate error covariance $\hat{P}_{k_{predicted}}$ approaches zero,

$$\lim_{\hat{P}_{k_{predicted}} \to 0} K_k = 0.$$

This means that when the measurement error covariance R approaches zero, K_k has a relatively high value and the second parcel of equation 3.7 will weight the residual Y_k more, meaning the read measurement Z_k is more “trusted” as it affects more the result, “trusting” less on the predicted measurement $H\hat{X}_{k_{predicted}}$. On the other hand, as the *a priori* estimate error covariance $\hat{P}_{k_{predicted}}$ approaches zero, K_k tends to be zero and the second parcel of equation 3.7 will weight the residual Y_k less, meaning the read measurement Z_k is less “trusted” while the predicted measurement $H\hat{X}_{k_{predicted}}$ is “trusted” more [14].

Having the output update been done with the equation 3.7, the Kalman gain is also used to update the estimated accuracy of the state estimate by updating the covariance matrix $\hat{P}_k$

$$\hat{P}_k = (I - K_k H_k)\hat{P}_{k_{predicted}}.$$

This *a posteriori* estimate of the covariance matrix is used at the time step *k+1* in equation 3.6 to calculate the *a priori* estimate of the covariance matrix $\hat{P}_{k+1_{predicted}}$.

The presented equations are valid for the optimal Kalman gain. Using other gain values is possible, however, the equations presented are not usable. Also, the Kalman filter can be extended [14] for non-linear systems with the Extended and Unscented Kalman filter versions, but although most systems are not correctly modeled by linear systems, the complexity and computational costs of more accurate models are higher, while the linear approach that Kalman does (considering the inaccuracy of the model) is good enough and computationally more simple and sustainable.

In an actual implementation of the filter, the measurement noise covariance R is usually obtained by practical data, as one is able to make a set of sample measurements in order to determine the variance of the measurement noise.
The determination of the process noise covariance Q is generally more difficult as we typically do not have a way to measure the process noise directly and independently. Sometimes a relatively simple process model can produce acceptable results if one considers enough uncertainty to the process through Q.
Whether or not we have a rational basis for choosing the parameters, we can often obtain better filter performance by *tuning* the parameters Q and R. This tuning process is usually performed offline, meaning that with defined sets of samples, tries for several values are made in order to find one that satisfactorily fits the problem. If both Q and R are constant, both the estimation error covariance $\hat{P}_k$ and Kalman gain K_k will stabilise quickly and then remain constant. In this case, these values can be pre-computed by running the filter offline. However, we frequently have non constant measurement error, for example, a distance measure is usually more erroneous as the distance gets higher. Also, the process noise Q is sometimes changed dynamically (becoming Q_k) during filter operation to adjust to different dynamics. An example is the tracking of a user's head in a 3D virtual environment, where we might reduce the magnitude of Q_k if the user seems to be moving the head slowly and increase it when the dynamics start changing rapidly [14].

3.3 Particle filter

Particle filter is a Monte Carlo [16] model estimation technique much used in information fusion. It is often an alternative to the Extended Kalman filter and Unscented Kalman filter.

The functional idea of the filter is that at each time step, multiple copies (particles) of the variables of interest are used, each one with an associated weight that signifies the quality of that specific particle. The estimation of the variables of interest is obtained by a weighted sum of all particles [17]. The particle filter is a method that similarly to Kalman filter, has two phases, prediction and update. In the first prediction phase, an *a priori* estimation of the particles state is done, by advancing the state accordingly to a given action model that is applied to the previous state (a process similar to the Kalman filter prediction phase) and by adding a noise with some known distribution, not necessarily Gaussian. The update process is made by re-evaluating the particles weight based on the latest sensory information available. This re-weighting process tends to increase the higher probability particles and decrease the lower ones. At some point, the weights are likely to be concentrated on a small number of particles. Most of the times this is not desirable and a resampling process is executed, where new particles are generated in order to keep the process sensitive to variations. There are several methods to do this resampling, however all based on random process for choosing weighted samples, a Monte Carlo approach.

When compared to Kalman filter, the particle filter has the advantages of being able to deal with non-linearities and non-Gaussian noise. However, its need for a great amount of samples in the resampling process due to the Monte Carlo approach, increase its computational weight and complexity. Given the assumptions on the Kalman filter are valid, its performance cannot be beaten by a particle filter [18, 17].

Given the environment of the CAMBADA project, although it is known that the problem is not exactly linear, a linear approach can describe it quite accurately, with an acceptable Gaussian noise model for the noise of the measurements. Thus, a Kalman filter can be applied with satisfactory results and at a lower computational cost.

3.4 Mapping and localisation

One important issue of sensor fusion on mobile robots is its ability to know where in the world it is. For that purpose, two requirements need to be met: 1) a model of the environment has to be known; 2) a method to place itself on the map. In the first problem, usually addressed as mapping, the robot must be able to traverse a location and through sensor data build a model of the place. The second problem is intimatelly related to the first because to advance through an area, the robot needs to locate itself on the map it has built or is building. The combining of the two problems is usually addressed as SLAM (Simultaneous Localisation And Mapping) and is currently a major research topic, since autonomous robot need localisation for mapping [3].

3.4.1 Mapping

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:

Metric or grid-based mapping: it is a representation based on the measures of the space 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 3.1). 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 over this kind of map could be “advance for X meters, turn θ degrees and advance other Y meters”;

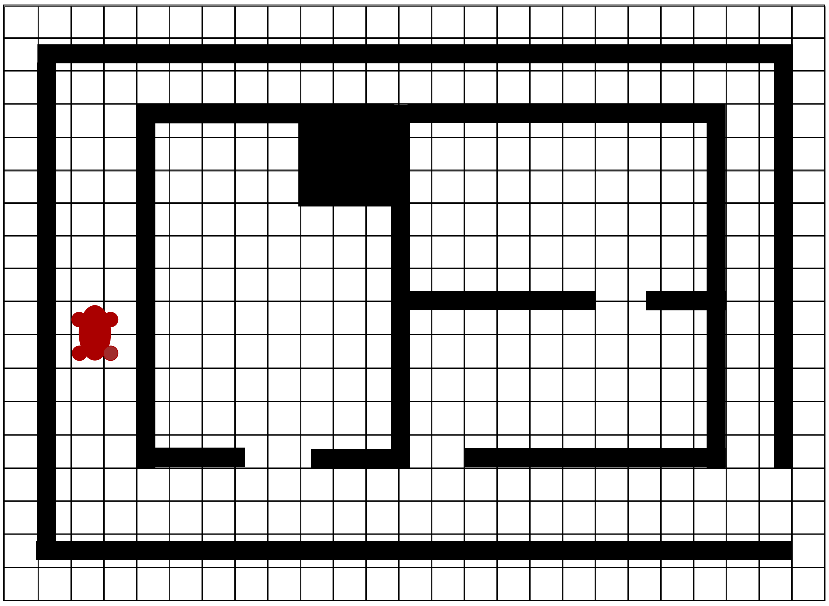


Figure 3.1: Illustration of a metric or grid-based map. All the known environment objects have known measures and positions.

Topological mapping: the representation of the space is not based on precise measurements but rather on landmarks (figure 3.2). 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 plan like “go straight till you see *this landmark*, turn left after it and follow until you reach *that landmark*”;

Examples of applications of these techniques are presented in [3].

3.4.2 Localisation

Having a map of the space, the robot usually needs to know where it is. Localisation requires then either *map matching* or *landmark detection*. Both methods can rely on sonar or vision-based systems, or, in outdoor navigation, on GPS.

An old technique of localisation using landmarks is triangulation, and although it has some hundreds of years, it is still successfully used in many situations. It is based on measurements on the angles between the own position and at least three visible landmarks, with known distances between each pair. Less than three landmarks can be used, but the distance of the own position to the landmarks themselves must be known. A GPS system, using the signals from three or four satellites can be seen as a form of triangulation [3].

Whatever the way of determining the distances used to localise a robot in its environment, they are surely imprecise and include a variety of disturbances. To obtain reliable estimates on the robot’s true position, stochastic methods are often applied, with models of both the system dynamics and the sensor dynamics, methods like Kalman filter, Bayes rule, particle filtering, among others [3].

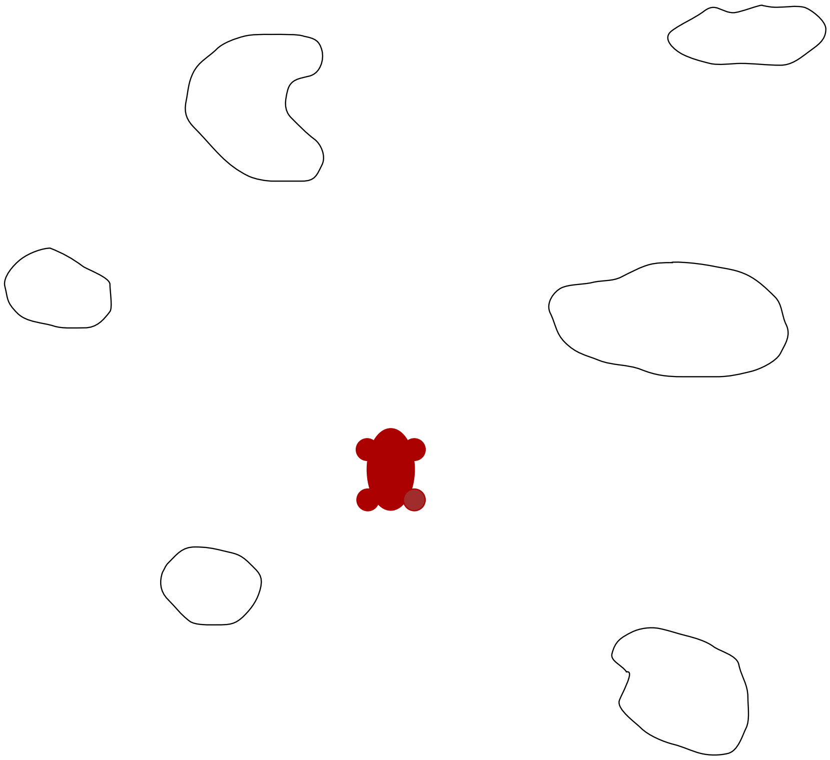


Figure 3.2: Illustration of a landmark map. The known landmarks are positioned relatively to each other, but no precise measures are known, navigation is only possible by sighting of the landmarks.

3.5 Summary

Hopefully after reading this chapter, the concept of sensor fusion and information fusion became more clear. The detailed formulation of two techniques, linear regression and Kalman filter (being the later one of the most used techniques), was presented, to provide the mathematical background for understanding their utilisation described in chapter 6. An overview on the particle filter functioning was presented, as it is another of the most commonly used techniques. Between Kalman and particle filters, a justification of the choice of a Kalman filter over the particle filter is explained.

A common robotic problem that requires sensor fusion is the mapping and localisation of robots on their environment. Two approaches of mapping were presented, metric or grid-based and landmark mappings. These two problems are usually intimately related, and thus, are usually addressed as one single SLAM (Simultaneous Localisation And Mapping) problem. The CAMBADA project also has SLAM issues, described in later sections.

Chapter 4

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] 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 teams involved. The CAMBADA robots have obviously evolved over the years, being described in this chapter their current state.

4.1 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 [19, 20] shape, circle [21] and squared [22, 23], but they can assume any shape (as long as they stay within the allowed sizes). 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 [19], electromagnetic [21] or electromechanic (with springs) [20].

The vision system has several implementations as well, being the most popular and adopted by most teams an omni-directional system [21, 19, 24], implemented in several different ways but commonly with a camera pointed up to a convex mirror on the top of the robot.

---

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

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

[3] The 2008 MSL team description papers (TDP) describing their robots can be found in the MSL pre-registration material: http://www.robocup-cn.org/en/index.php

4.2 CAMBADA Robots

4.2.1 Physical structure

The CAMBADA robots were designed and completely built in-house. Each robot is built upon a circular aluminium chassis (with roughly 485 mm diameter), which supports three independent motors (one for each holonomic wheel, allowing for omnidirectional motion), an electromagnetic kicking device (figure 4.1.b) and three NiMH batteries (figure 4.1).

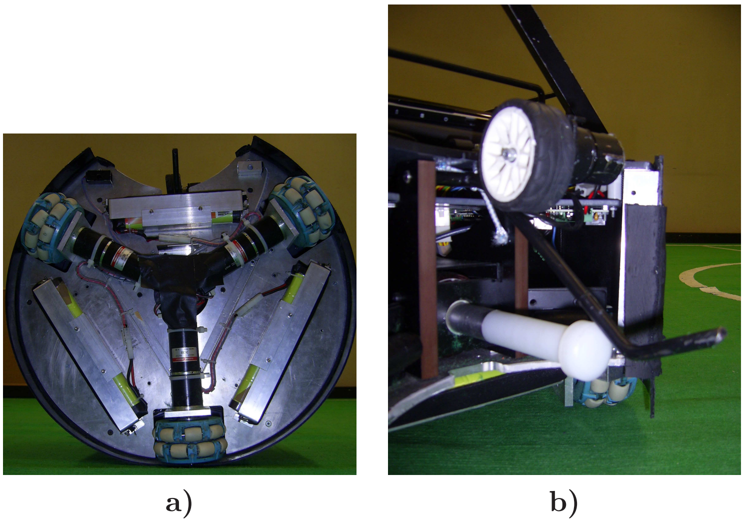
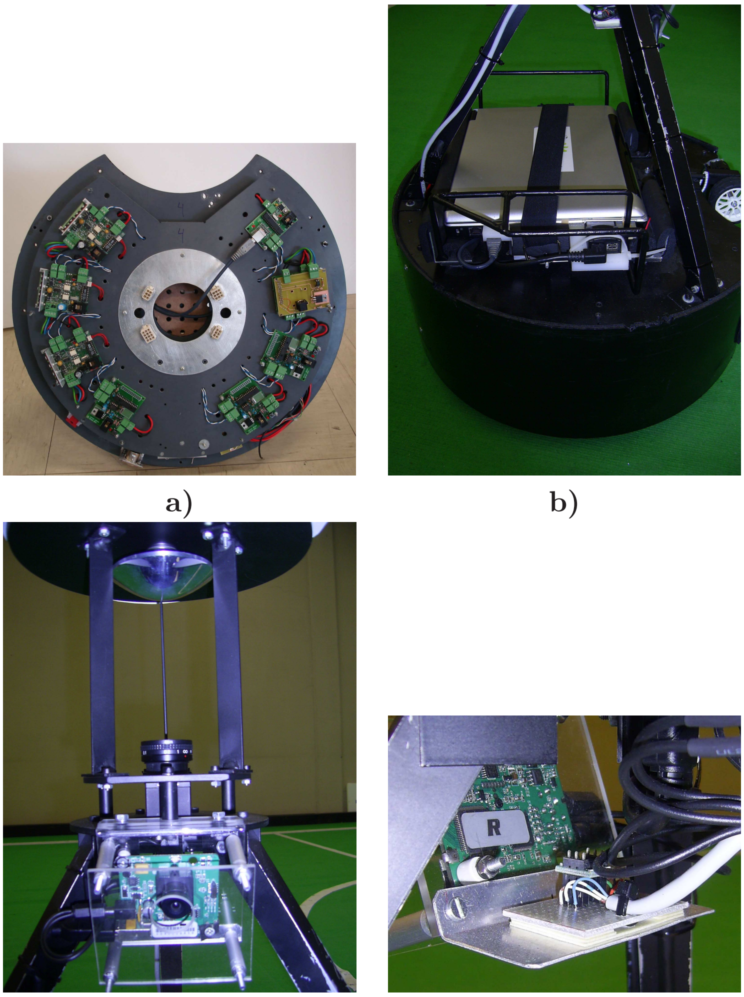

a) **b)**

Figure 4.1: Pictures of **a)**: the CAMBADA robot base, where the holonomic wheels, motors and batteries are visible; **b)**: the CAMBADA robot kicker and grabber system.

The remaining parts of the robot are placed in three higher layers, namely: the first layer upon the chassis is used to place all the electronic modules such as motor controllers (figure 4.2.a); the second layer contains a PC (currently a 12" notebook based on an Intel Core2Duo processor) (figure 4.2.b); finally on the top of the robots stands an omnidirectional vision system consisting of a camera pointing to an hyperbolic mirror, a frontal vision system, consisting of a camera pointing forward and an electronic compass, used for localisation purposes (figures 4.2.c and 4.2.d). 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 system. These two sections can be easily separated from each other, allowing an easy access both to the mechanical components and to the electronic modules [25].

4.2.2 General architecture

The general architecture of the CAMBADA robots has been described in [26, 27, 28]. 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



a) **b)**

c) **d)**

Figure 4.2: Pictures of **a)**: the CAMBADA robot micro controller network; **b)**: the processing unit, on the second layer, under the support tripod base; **c)**: the vision systems on the top of the robot. The frontal camera is seen in the lower part of the picture, the camera and the hyperbolic mirror it points to are also visible; **d)**: the electronic compass, installed on the back of the frontal camera support.

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 [29, 30] on top of IEEE 802.11b, 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 [21].

Hardware

The low-level sensing/actuation system (figure 4.3) is implemented through a set of microcontrollers interconnected by a network. Controller Area Network (CAN) [31], a real-time fieldbus typical in distributed embedded systems, has been chosen for the task. To complement the network, the FTT-CAN (Flexible Time-Triggered communication over CAN) [32] higher-level transmission control protocol is used to enhance its real-time performance, composability and fault-tolerance. The low-level sensing/actuation system executes five main functions, namely, Motion control, Odometry, Kicking, System monitoring and Compass. The Motion control function provides holonomic motion using 3 DC motors actuating over holonomic wheels. The Odometry function combines the encoder readings from the 3 motors and provides coherent robot displacement information that is then sent to the coordination layer. The Kick function includes the control of an electromagnetic kicker and a ball handler (grabber, figure 4.1.b) 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 Compass function reads the angle from magnetic north given by the electronic compass. The low-level control layer communicates with the coordination layer through a gateway, which filters the information on both layers, allowing only relevant information for the other layer to pass [21].

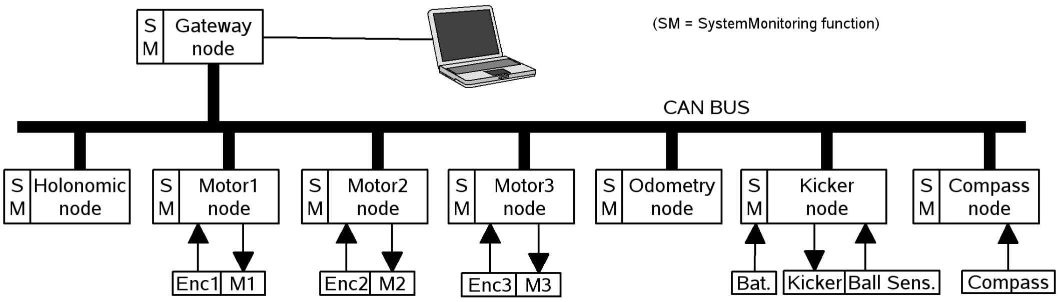


Figure 4.3: Hardware architecture diagram.

Software

The software system (figure 4.4) 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 microcontrollers. A cooperative sensing approach based on a Real-Time Database (RTDB) [26, 29, 33] 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 odometry information coming from the low-level layer. This information 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 lowlevel control layer [21].

The processes interacting with the RTDB are:

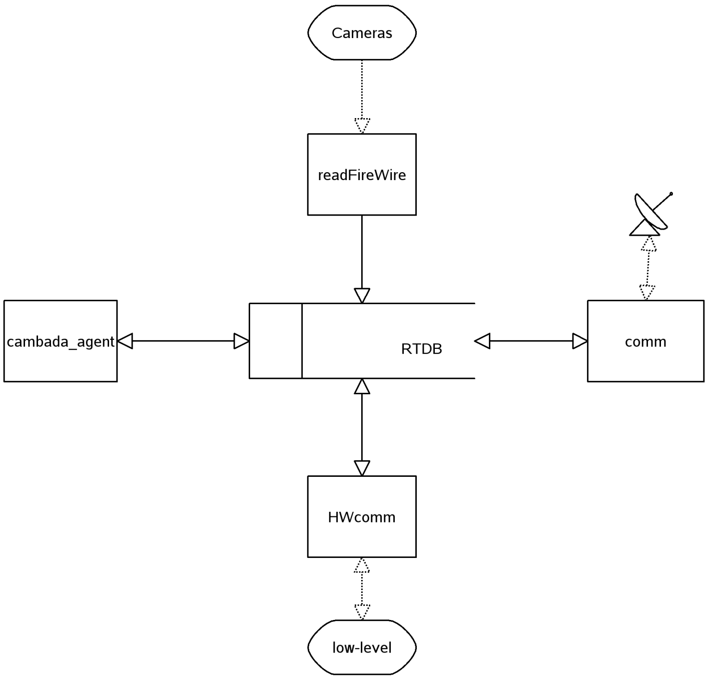


Figure 4.4: Software architecture diagram.

- **readFireWire**, vision process, responsible for retrieving the cameras frames and make all the image processing. The process has several steps, from colour segmentation to object identification and relative position estimate. The processed values are then injected in the RTDB;
- **HWcomm**, hardware comunication process, gets information from the hardware micro controllers and sends commands to those micro controllers so they execute them through the actuators in the low-level;
- **comm**, communication process, responsible for transmitting the RTBD information relevant for and from team mates;
- **cambada_agent**, control process that integrates the information and decides which actions to take and which orders to send to the actuators.

4.3 CAMBADA vision overview

The vision process *readFireWire* 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 fps. It analyses the current image and fills its RTDB structures with the *relative* (see section 5.1.1) positions of the detected objects . Since the analysis is made for 3D object on a 2D image, a projection of the object centre to the ground plane must be considered. Frames by both the omni-directional and frontal vision systems are presented in figure 4.5.

The frontal vision system is the “secondary” system, currently used only for checking for the ball when it is not visible by the omni-directional system. This is because the frontal

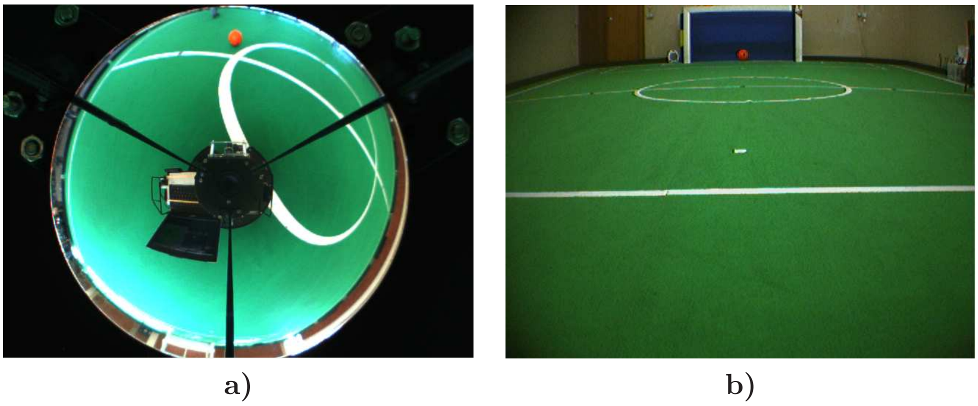


Figure 4.5: Vision frames **a)**: Omni-directional vision frame; **b)**: Frontal vision frame.

vision system is oriented in only one direction, which limits the robot vision field to its front. However, as the image size is the same as the omni-directional image, the same field area is represented by more pixels, allowing for a better pixel-metric distance relation. This fact makes it easier to detect the ball at longer distances.

Distance Map

Given an image from the omni-directional camera to analyse, a necessary step is to map the image distances to real-world ground plane distances with the robot as the centre of the system. An approach to this problem could be a mechanical setup that ensures a symmetrical solution for the problem by means of single viewpoint (SVP) approach. However, this solution requires high quality components (camera and hyperbolic mirror) and a precision mechanical setup. A solution for the team was developed to calculate the robot centred distances map with a non-SVP catadioptric setup, by exploring a back-propagation ray-tracing approach and the mathematical properties of the mirror surface [21, 34].

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. Also, the mapping of the pixels on the image to real world distance is not direct at all. A relation for the omni-directional vision between pixel and real world metric distance is represented in figure 4.6. This obviously means that farther distances are worstly evaluated (visually perceptible in figure 4.5.a), since the distance represented by each pixel increases non-linearly as the pixel is farther from the centre of the image. The frontal vision has similar distance mapping problems, as pixels get farther from the bottom of the image.

For details on the vision system, refer to [35, 36, 37].

4.4 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

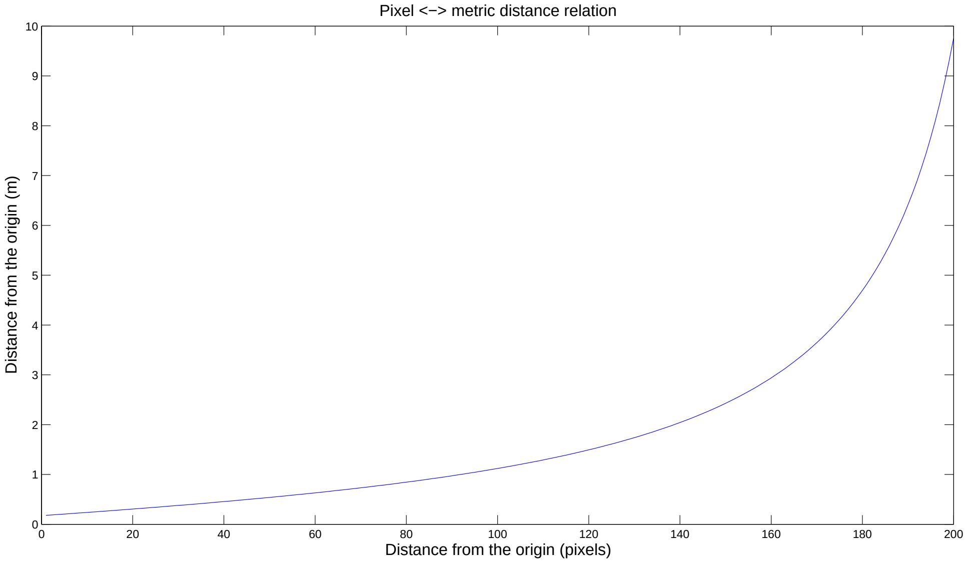


Figure 4.6: Relation between pixel and metric distances.

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 a lower level *Behaviour*, which defines the orders to be sent down to the actuators in order to fulfil a task, for example, a Move behaviour, to get to a given point on the field.

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

The behaviour controllers are an implementation of PID controller (section 2.3). Each behaviour should have its own controller, tunned for a desired response. Some behaviours that require much precision should have a smooth response, even if it takes more time to achieve it. Others may have a more abrupt response, even if they oscillate a bit around the desired value.

4.5 Localisation

Self-localisation 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 completely known, as every agent knows exactly the layout of the game field. Given the known mapping, the agent has then to locate itself on it. An effective way to do it is to use a fusion between a topological mapping approach, where the landmarks are the lines of the field and a metric mapping approach (see section 3.4), by applying the known field line measures to the detected landmarks. Many teams in MSL use this kind of approach, based on visual detection of white lines and matching with the known "map" of the environment, the field layout. The way of implementing the fusion of the landmarks with the distances vary from team to team, with application of particle filter, Kalman filter, Hough transform, among others (see MSL teams TDP[4] [20, 38, 23, 22, 39, 19, 40]).

The CAMBADA team localisation algorithm is also based on this approach on the field lines as landmarks, with fusion of a strictly metric approach on the odometry sensors. It is based on the approach of the Tribots team[5] technique [41], with some adaptations. It can be seen as an error minimisation task, with a derived measure of reliability of the calculated position so that a stochastic sensor fusion process can be applied to increase the estimate accuracy [41]. For localisation purposes, only the onmi-directional vision system is used. From the centre of the image (the centre of the robot), 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. The idea is to analyse the detected line points, estimating a position, and through an error function describe the fitness of the estimate. This is done by reducing the error of the matching between the seen lines and the known field lines. The error function must be defined considering the substantial amount of noise that affect the detected line points which would distort the representation estimate. Although the odometry measurement quality is much affected with time, within the reduced cycle times achieved in the application, consecutive readings produce acceptable results and thus, having the visual estimation, it is fused with the odometry values to refine the estimate. This fusion is done based on a Kalman filter for the robot position estimated by odometry and the robot position estimated by visual information. This approach allows the agent to estimate its position even if no visual information is available, although it is not reliable to use only odometry values to estimate the position for more than a very few cycles, as slidings and frictions on the wheels produce large errors on the readings in short time.

To find the initial position on the field, situation where there is no *a priori* estimation, an initial position has to be "created" to apply the error minimisation idea. One approach consisted on a Monte Carlo approach, generating random positions inside the field and estimate the error for that position, considering the visual information. This was done while the estimated error was above a given threshold or a maximum of 2000 times. However, in practice it was verified that finding the initial position took about 5 seconds; after many tests, it was verified that the cycle always runned 2000 times. Although probabilistically it would in some run be less, and knowing that a random position can be found very close to the real position but also very far, a new approach was idealised. By dividing the field in a squares matrix, the cycle now run for each of those squares testing for the one that minimises the estimation error. With a square definition of 1 meter (considering also the 1 meter margin around the field), the algorithm would run the cycle at most 280 times, reducing the run time to less than 2 seconds. After having the position coordinates and the visually estimated orientation, that orientation is confirmed by the method described in section 6.4.

---

[4] MSL team description papers (TDP) describing their approaches can be found in the MSL pre-registration material: http://www.robocup-cn.org/en/index.php

[5] http://www.ni.uos.de/index.php?id=2

4.6 Summary

This chapter presented a description of most aspects of the CAMBADA robots, their physical structure and hardware and software architectures. It gave a general idea on how the software agent runs, the order of the tasks executed. The controlling role of the vision over the software agent was explained, and some emphasis given to the distance map issue, as it is affects later integration features (section 6.2). The CAMBADA SLAM algorithm was presented and the changes done within this thesis scope presented. The measured execution times prove that the changes were effective.

Chapter 5

World state

The *Worldstate* is the base over which the agent keeps its perception of the world, it maintains all the information available in a structured manner. As stated in the previous chapter, all the information in the worldstate is updated every agent cycle before any decision is taken.

5.1 Organisation

One cannot build a good model of the world if it is not well organised. In order to do that, some assumptions had to be made on location systems, and data structures had to be created for different purposes and types of information. Some of the main classes built around the *Worldstate* are described below in 5.1.2.

5.1.1 Coordinate systems

To play in the field, the agents necessarily have to know how to identify at least the goals they are using, in order to play in the right direction. Better than just knowing which side to go, is to know where in the field the agent is. Localisation algorithms are able to estimate this, but they must work on an absolute base, common to all of them. That base is the fitting of a two dimensional cartesian system on the field. The CAMBADA team has adopted a system defined as figure 5.1 states, where origin is the field centre and the YY axis grows in the attack direction. This coordinate system is referred to as *absolute*, since it is a static referential.

However, the robot has its vision system on board. As the robot moves, it cannot tell where in the field it is before obtaining the visual information on the field elements positions, and those positions cannot be obtained if the robot does not know where it is. Thus, a second coordinate system is needed, one that can be used by the robot independently of its position on the field. As the position of the camera on the robot is known, the logical way to do it is to have a cartesian system relative to the robot. That system has its centre on the robot centre and the YY axis grows in the robot front direction (figure 5.2). This coordinate system is referred to as *relative*, since its positions are relative to the robot position, not to the static field referential.

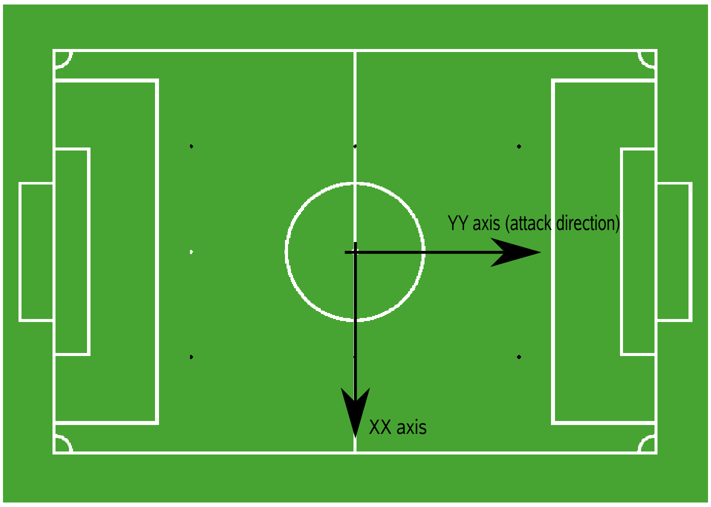


Figure 5.1: Field coordinate system (absolute coordinates).

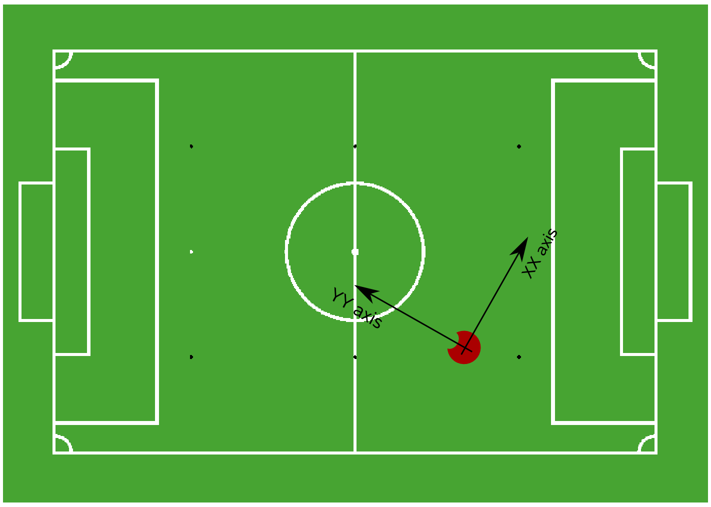


Figure 5.2: Robot coordinate system (relative coordinates).

To change the representation of a point from one coordinate system to another, one can apply a translation of the point to the desired system axes origin and then a rotation corresponding to the displacement between the axes.

In figure 5.3, (X_r, Y_r) represents the position of the robot on the field coordinates and θ is the angle between the two coordinate systems.

The transformation of a point expressed in robot coordinates to a point $\begin{bmatrix} X' \\ Y' \end{bmatrix}$ expressed in field coordinates would be accomplished applying the geometric transformation matrix

$$\begin{bmatrix} X' \\ Y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix} + \begin{bmatrix} X_r \\ Y_r \end{bmatrix}$$

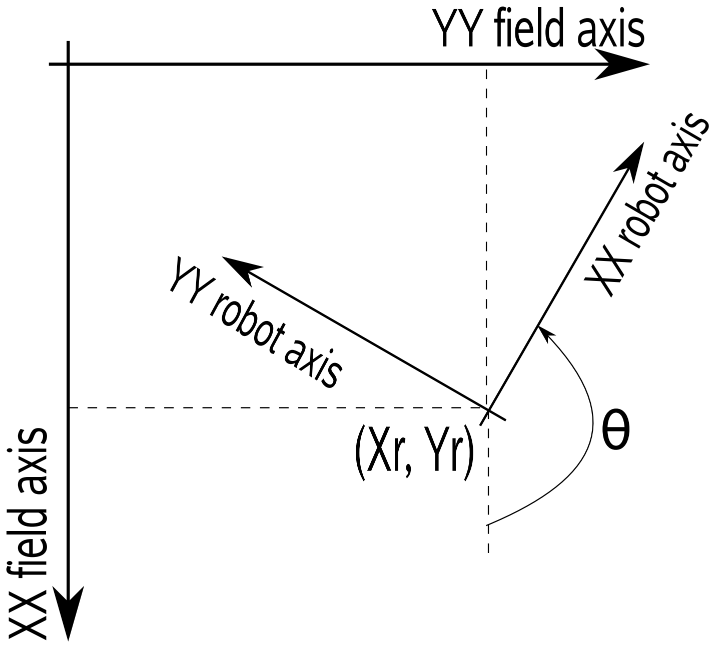


Figure 5.3: Illustration of the two coordinate systems the relations between them that are necessary to transform the coordinates of a given point.

where $\begin{bmatrix} X \\ Y \end{bmatrix}$ is the vector representation of the original point coordinates, $\begin{bmatrix} X_r \\ Y_r \end{bmatrix}$ is the vector representation of the robot position on the field coordinates and θ is the angle between the coordinate systems. Analogously, to transform a point expressed in field coordinates to robot coordinates, a similar transformation matrix would be applied, but for the symmetrics for θ and $\begin{bmatrix} X_r \\ Y_r \end{bmatrix}$.

All the *(x,y)* positions are defined using a *Vec* type which includes two attributes, x and y, representing a point coordinates in a cartesian system. Also, the velocities used are represented with a *Vec*, representing a velocity vector with start at the axis origin and end at the defined *Vec* point.

5.1.2 Data structures

The *Worldstate* class has to treat and keep information about several elements of the environment. This includes information about the robot itself, its environment obstacles, the general game's state and mapping of the field for strategic purposes. To keep this information, several data structures have to be used:

- *ConfigXML* → parser for the xml configuration file, not built to exactly keep information on anything, but rather to provide a tool to get the information on the *cambada.conf.xml* file;

- *WSGameState* → enumeration to indicate which is the game's state;

- *ZoneMatrix* → a class that divides the field in a matrix and classifies the state of each cell as free of obstacles or not. Used for path planning;
- *UtilField* → in an utility approach, this class has the methods to calculate the utility of the field positions. Used for path planning;
- *vector<Vec> obstacles* → a vector with the position of all the detected obstacles given by the vision. To be filled by the integrator;
- *RobotWS* → class that keeps the information of the agent's description of the world. This is the main structure, to be filled by the integrator;

RobotWS

This structure keeps the information of the agent's internal world state. The *Worldstate* class keeps an array of these structures, one for each playing agent

```
RobotWS cambada[N_CAMBADAS];
```

and for simplifying, a static direct pointer to the self agent

```
static RobotWS* mySelf;
```

in order to keep the information of itself and all the shared information of the team mates. The *RobotWS* class was built in a hierarchical structure, in order to keep all the needed information organised and parameterised (figure 5.4). The information structures on the class include information on some field objects and robot internal information:

***me*:** the agent itself, a *Player*;

***ball*:** the game ball;

***ourGoal, theirGoal*:** both the own and the opponent goals;

***battery[]*:** an array with the status of each of the robot's batteries;

***coordinationFlag*:** a flag for agent coordination;

All the objects on the field have two common properties, a *position* on an *(x,y)* system and a *visible* flag to define if it is visible by the agent. Generalised from *Object* we have the *Goal*, which is a static object that has two other positions, the *leftPost* and the *rightPost*; also from *Object*, we have some that are not static, thus defining a *MobileObject*, which is characterised by having a *velocity* vector.

Fitting into *MobileObject*, we have the ball and the player.

The *Ball* class keeps an attribute to indicate if the robot has it *engaged*. Also, as the vision information on the ball can only be given relatively to the robot, its *(x,y) position* as an object is kept as a relative position. As for most purposes the absolute position on the field is needed, the conversion method *rel2abs* (presented in section 5.2.1) was systematically invoked. Thus, another position *posAbs* is used to keep the absolute ball position. Another attribute of the ball is the *type*, which is an enumerate that defines how the ball is perceived, either by the agent itself or by means of teammates information fusion.

The *Player* keeps additional information on the running agent:

***number*:** the identifying number of the agent;

***role*:** the role currently assumed by the agent;

***behaviour*:** the behaviour currently assumed by the agent;

***roleAuto*:** a flag to indicate if the role attribution is in automatic mode;

***running*:** a flag to indicate that the agent is in running status;

***coaching*:** a flag to indicate if the agent is being coached externally;

***teamColor*:** the colour being used by the agent's own team;

***goalColor*:** the colour of the agent's own goal (despite the disappearance of the goal colour, they still have to be differentiated in a univocal way);

***orientation*:** the orientation is the angle between the absolute field axes and the robot relative axes, given both the axes assumes the normal counter-clockwise angles;

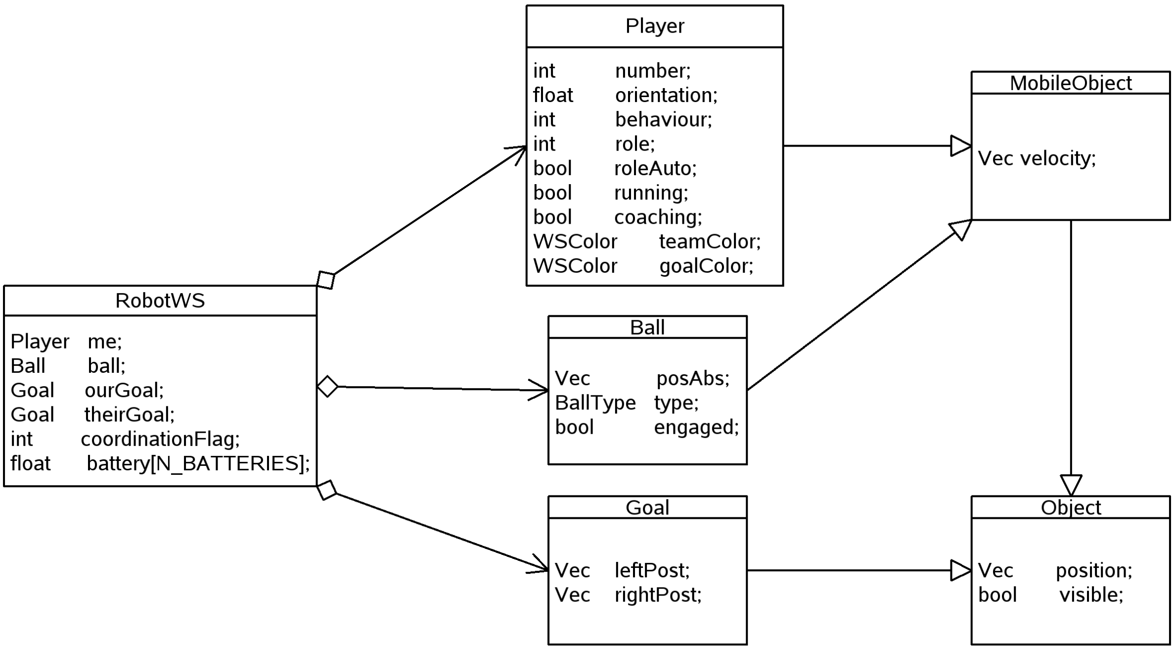

Player
int number;
float orientation;
int behaviour;
int role;
bool roleAuto;
bool running;
bool coaching;
WSColor teamColor;
WSColor goalColor;
MobileObject
Vec velocity;
RobotWS
Player me;
Ball ball;
Goal ourGoal;
Goal theirGoal;
int coordinationFlag;
float battery[N_BATTERIES];
Ball
Vec posAbs;
BallType type;
bool engaged;
Goal
Vec leftPost;
Vec rightPost;
Object
Vec position;
bool visible;

Figure 5.4: Class diagram for the field objects.

5.2 Methods and algorithms

Having just the raw information kept on this class does not give the higher levels enough advantage. Thus, a set of tools should be created to manipulate these data in order to also obtain usefull information derived from the base one.

5.2.1 Transformation methods

Since there are two distinct coordinate system, methods to transform coordinates between the two systems are needed:

- `Vec rel2abs(const Vec& position , const double& orientation, const Vec& rel);`

  → transforms a position in relative coordinates into the absolute coordinate axis. The relative axis is defined by the given orientation and centre position;

- `Vec rel2abs(const Vec& rel);`

  → also transforms a relative position into an absolute position, considering the own agent position and orientation as the relative axis;

- `Vec abs2rel(const Vec& abs);`

  → transforms an absolute position in a relative one, using the own agent as the centre of the relative axis;

- `Vec abs2relDelta(const Vec&);`

  → rotates a relative position to the own agent relative position, but without translating it;

- `Vec myRelVelocity();`

  → transforms the robot velocity, represented natively in absolute coordinates, to relative velocity vector;

These transformations are done by applying the transformation matrix described in section 5.1.1 or in the case of the last two, only the rotation matrix

$$\begin{bmatrix} \Delta X' \\ \Delta Y' \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} \Delta X_{abs} \\ \Delta Y_{abs} \end{bmatrix}.$$

5.2.2 Basic queries (getters)

In the *cambada.conf.xml* file, there is a group of information on field measures and static positions, expressed in millimetres, that are constantly needed. To apply these measures though, they must be given in meters. Despite the simple conversion from millimetres, it became clear that tools for making that conversion would drastically reduce and simplify the code on the decision level. Thus, a set of these methods were implemented to return the desired values, each with a self-explanatory name:

- `double getFieldLength();`
- `double getFieldHalfLength();`
- `double getFieldWidth();`
- `double getFieldHalfWidth();`
- `double getGoalAreaLength();`
- `double getGoalAreaWidth();`
- `Vec getTheirGoal();`
- `Vec getOurGoal();`

- `double getGoalWidth();`
- `double getGoalHalfWidth();`
- `double getGoalHeight();`
- `double getPenaltyAreaLength();`
- `double getPenaltyAreaWidth();`
- `double getPenaltyMarkerDistance();`

5.2.3 Advanced queries and utilities

In the implementation of the high-level decision, there are a set of conditions on the worldstate that are recurrently verified. While for some of these methods the objective is easily perceived by the name, for others the respective algorithm explanation is presented.

- `KickEvaluation kickable();`

  → evaluates the ball for a set of conditions to get its kicking status. The possible states are

  > **Try to score** when the position of the robot is favourable to try a shoot to goal;
  >
  > **Too far** when the robot is farther than a defined distance from the opponent goal;
  >
  > **Dead angle** when the robot is near the goal line and the angle to the middle of the goal is very narrow;
  >
  > **Not aligned** when the robot is not aligned to shoot in goal (checked by the method described below;
  >
  > **Obstacle** if an obstacle is near the robot, which would prevent it from kicking over it;

- `bool shootInGoal(double offset = 0.35);`

  → checks if the robot position is a good one to shoot in goal by extending a line that should intercept the goal line within a given offset from the goal centre (usually both the goal posts) (figure 5.5);

- `bool insideField(Vec position, float margin = 0.0);`

  → verifies if a given position is inside the game field. A margin around the field can also be considered;

- `bool isNear(Vec point, double dist = 0.1);`

  → checks if the running agent is near a given absolute position, that is, if it is within *dist* metres from the *point*;

- `bool isNearTheirGoalArea(double distance = 0.0);`

  → checks if the agent is near the opponent goal area. Used for foul avoidance, since the robot should not enter the opponent goal area [1]. A *distance* in metres can be considered around the goal area, creating an additional margin;

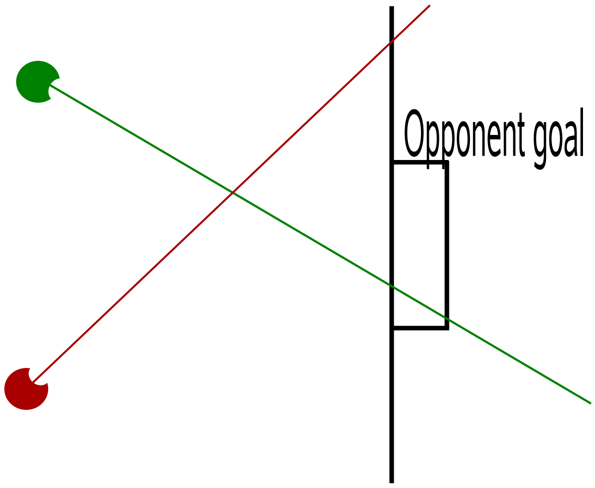


Figure 5.5: Illustration of a "shootInGoal" evaluation. The green player (top of image) shoot line intersects the opponent goal line within the goal posts, *true* would be returned. The red player (bottom of image) shoot line doesn't intersect the opponent goal line between the posts, and therefore, *false* would be returned.

- `bool isNearOurGoalArea(double distance = 0.0);`

  → analogously to the previous method, tests for the agent proximity to its own goal area. The same *distance* margin can be defined;

- `bool isBallNearOurGoalArea(double distance = 0.0);`
  `bool isBallNearTheirGoalArea(double distance = 0.0);`

  → analogous to the previous two functions, but these ones concerning the ball;

- `DangerType isBallInDangerZone(bool goingToApproachPoint, int agentIdx = -1);`

  → verifies if the ball is in a field area where it is in danger of going out. If it is too close to the limits of the field, the agent should approach it by the limit side (figure 5.6). This method indicates in which danger the ball is, either going out by a side line, by its own goal line, by the opponent goal line or in no danger of going out;

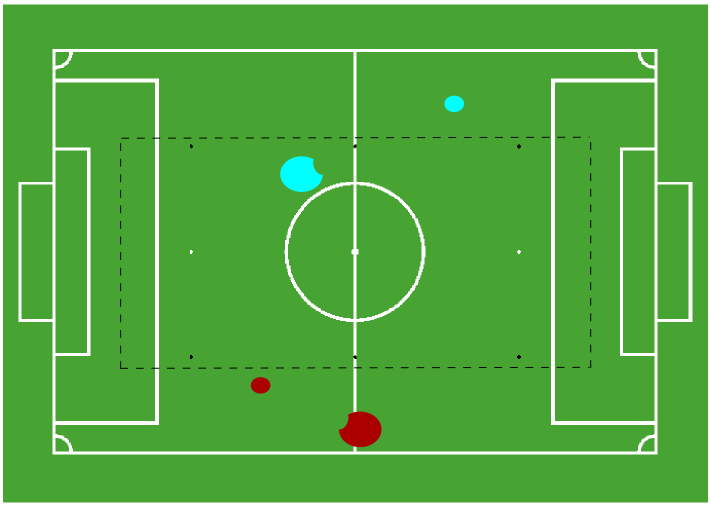


Figure 5.6: Illustration of an “isBallInDangerZone” evaluation. The blue player (top) would get a *true* for the respectively blue ball in danger zone because it is in the limit corridor and the player is more inside the field than the ball. The red player (bottom) would get a *false* because, despite the respectively red ball being in the limit corridor, the player itself is even closer to the line, so it would already make an approach from outside to inside.

- `int getClosestToAbs(Vec posAbs);`

  → verifies all the agents position (including itself) and returns the one closer to the given absolute position;

- `Vec getApproachPoint( Vec facingPoint, int agentIdx = -1 );`

  → calculates a point to approach the ball rather than go directly to it. The approach point calculation takes into account an absolute position to which the agent wishes to be facing when catching the ball, the facing point. A circumference around the ball is considered, to define how far from it the approach should be made. The optimal approach point is considered the one given by the vector *facing point-ball*, plus the defined circumference radius. Considering the robot position, two possible approach points are calculated, over the defined circumference and perpendicular to the *robot-ball* line. These approach points are tested and the closer to the optimal approach point is chosen (if the distance to it is less than the optimal). This allows for an approximation “around” the ball rather than directly to it, and as it tends to the optimal approach, the robot tends to get to a position where it can go straight to the ball and catch it without need for any rotation to the target (figure 5.7);

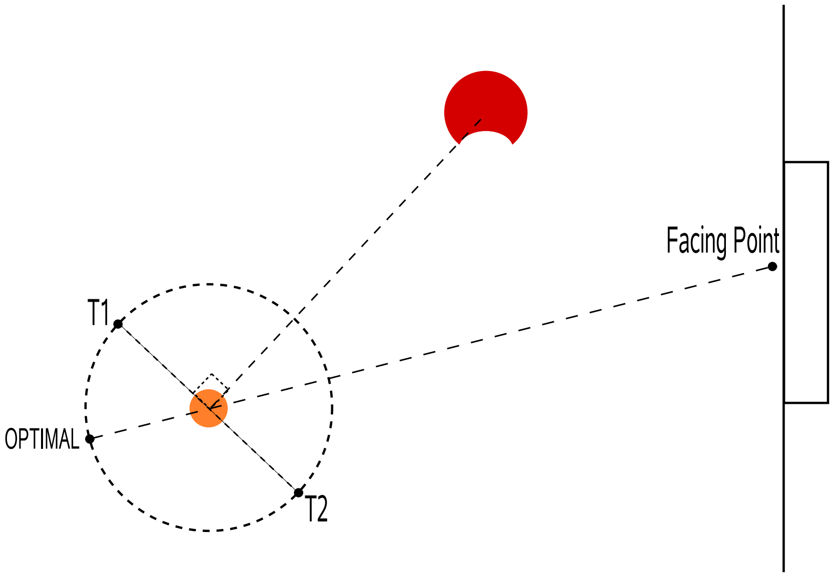


Figure 5.7: Illustration of the "approachPoint" algorithm. *Facing Point* is the point that the player should be facing when catches the ball, thus, *optimal* would be the optimal approach point, as if the player could catch the ball from that direction, would be automatically facing the destination. An approach on that point is given by the closest of the targets *T1* and *T2*, perpendicular to the *player-ball* line.

- `bool isLineClear(Vec absPosition);`

  → verifies if the line between the running agent and a given absolute position is free of obstacles. An offset around the line is obviously considered, both because of noisy positions and obstacles volume (figure 5.8);

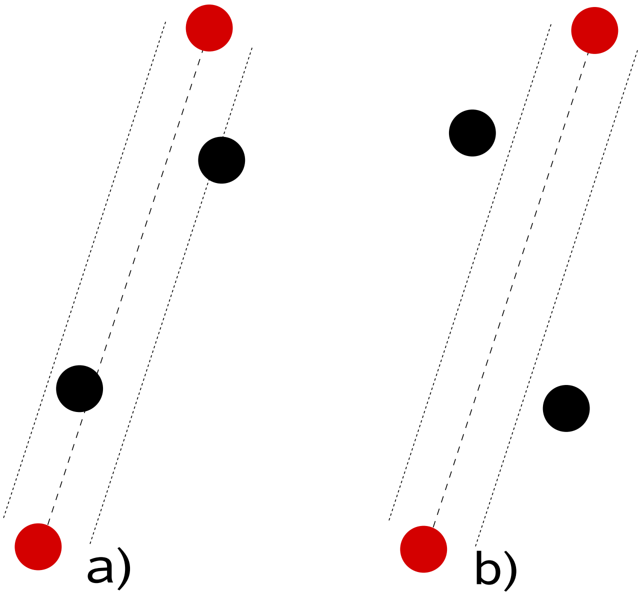


Figure 5.8: Illustration of the “isLineClear” algorithm. Testing a line between the two red players, in situation **a)** it would return false, as 2 obstacles are within the considered area. Situation **b)** would return true, as the tested area is clear of any obstacle.

- `bool isSliceClear( Vec center, Vec outLine );`

  → verifies if a “slice” is free of obstacles. The slice is defined between a given circle centre and a circle outline. In a pass situation, this approach tests for obstacles in an increasingly larger area, which can be used to take into account that the ball takes some time to cover the distance to its target, and that opponents near the target have more time to intercept it than opponents near the origin (figure 5.9);

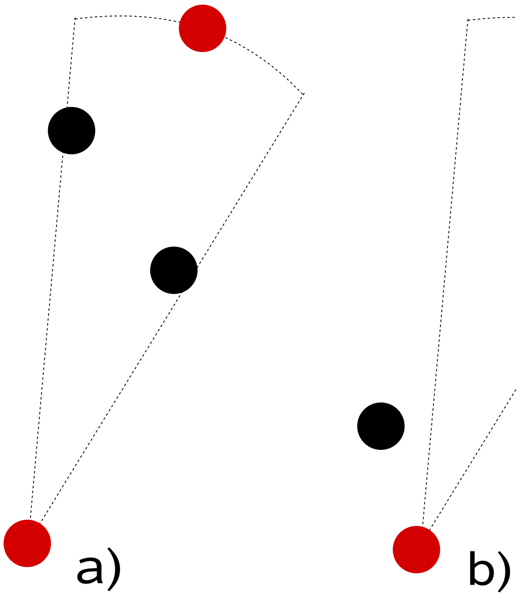


Figure 5.9: Illustration of the "isSliceClear" algorithm. A slice between the two red players is tested, being the bottom player the origin. In situation **a)** a false would be returned, as there are obstacles within the slice, while situation **b)** would return true, because the slice is free of obstacles.

5.2.4 Interception point algorithms

In order to improve the team performance concerning the approach to the ball, the need for some kind of interception method was required. By taking the ball velocity into account, the robot could eventually predict where and when could it intercept the ball. Although these algorithms are implemented in the *Worldstate*, the detailed descriptions are presented on the chapter 7, subsections 7.2.1 and 7.2.2, as they are intimately related with the *ActiveInterception* behaviour.

5.3 Summary

After reading this chapter, one should be familiar with both the used representation coordinates system and the main data kept in the representation of the state of the world, mainly concerning the robot information. Most of the tools developed to manipulate and get more complex information from the *worldstate* were presented. Most of these tools were developed based on the needs of the high level and proved to provide simplification of the code and also allowed the development of new movements and strategies that boosted the team performance.

Chapter 6

Integration

The integration of the information available to the robot is a crucial step. In the CAMBADA software architecture this is the function of the *Integrator* module. As values, mainly from the camera, are available to the agent, they have to be properly handled so that a relevant and trustworthy database is built to represent the current state of the world as it is seen by the robot. This is the first part of the agent cycle. The integrator is responsible for the treatment of data and filling up the information on the worldstate presented in the previous chapter.

Almost all the results presented on this chapter are provided by captures and thus all the positions are the ones estimated by the robot. Also some results are obtained by theoretical values, and are identified as such in the respective section.

6.1 Organisation

The integrator starts by getting the RTDB vision information and fill the respective structures, presented bellow. It then updates the information provided by the low-level micro controllers. This update includes getting the odometry values and calculating the displacements since last cycle for x, y and angular components and getting the current battery and ball barrier status. A verification of the control station is made, by reading the coach information on the RTDB. The control station defines the team colour, goal colour and the robot role (which is automatic in game situations). Also, the start and stop commands are given by the control station. The team mates information is read from the RTDB and the obstacle vector in the worldstate is updated by the current vision obstacle information.

The localisation algorithm then uses the vision information of the field lines and the odometry displacements to estimate the agent position on the field. After getting the position, a linear regression can be updated and used to get the agent's velocity. Since the line based localisation can be ambiguous due to symmetric field positions, a verification is needed. This verification is made by getting the robot orientation from the electronic compass and estimate the orientation error. Afterwards, the integration of the ball is done, the gamestate is updated and a free path for the opponent goal is estimated.

All the information described above must be kept by the *Integrator*. For that, there are several classes and data structures to handle the information treatment and fusion:

- `LinearRegression;`

→ the *LinearRegression* class allows to estimate a linear regression. It is used for velocity estimation of both the ball and the robot. Described in detail in section 6.3;

- `BallPositionKalman;`

  → this *BallPositionKalman* is a class built to implement a Kalman filter to filter the ball positions. It is described in section 6.2;

- `Compass;`

  → a *Compass* class, which is responsible for receiving and treating the degree information given by the electronic compass. Its functioning is described in section 6.4;

- `VisionInfo;`

  → the *VisionInfo* data structure holds the information written on the RTDB by the vision process concerning the omni-directional camera, which includes information about field lines, obstacles and ball;

- `FrontVisionInfo;`

  → the *FrontVisionInfo* data structure holds the information on the RTDB concerning the frontal camera, currently including only information of the ball;

- `LowLevelInfo;`

  → the *LowLevelInfo* class holds the information of the low-level layer, namely odometry, battery status and ball barrier sensor. It has the tools for calculating the displacements on odometry between consecutive cycles;

- `CambadaLoc;`

  → The *CambadaLoc* class has the tools for estimating the robot position on the field. It receives both the low-level odometry information and the vision field lines information;

- `CoachInfo;`

  → a *CoachInfo* class to hold information of the control station "orders";

- `ConfigXML*;`

  → a pointer to the robot *ConfigXML* object, parsed from the *cambada.conf.xml* file;

- `Strategy*;`

  → a pointer to the robot *Strategy* object, holding the strategic information for the team;

- `WorldState*;`

  → a pointer to the robot *Worldstate* object. The previous classes and data structures can be seen as intermediate tools to temporarily hold the information and treat it, with the necessary estimations and refinement. The final refined information is then filled in the *Worldstate* object, being the one used by the robot.

The representation of the current *WorldState* is based on the analysis of the previous one and the integration of the information of the current cycle, kept and treated by the presented "temporary" classes and integrated coherently in the *WorldState* (figure 6.1).

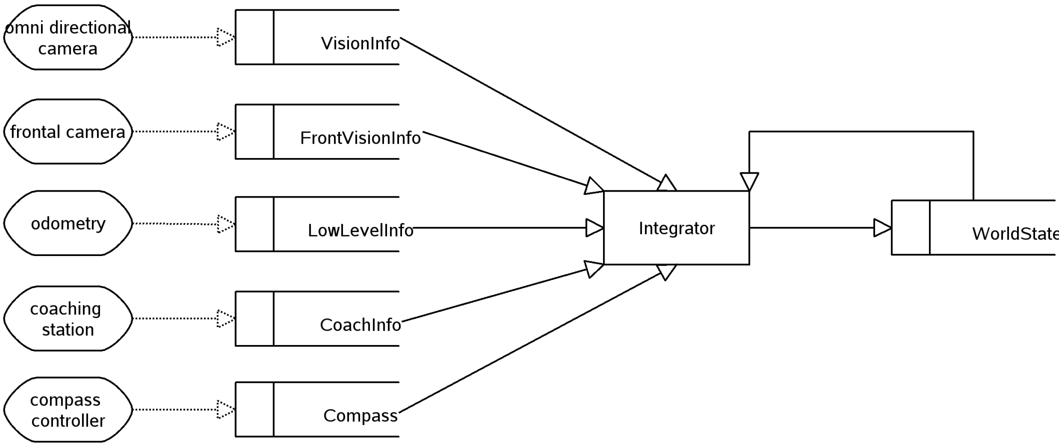


Figure 6.1: Integrator functionality diagram.

6.2 Ball position filter

In order to keep a good tracking of the ball, some refinement of the noisy measures of the ball position is required. In order to achieve that, an implementation of a Kalman filter was proposed and implemented. It should filter the ball position to keep its trajectory steady but keeping the response time to ball deviations low enough for reactivity to be hardly affected.

It is assumed that the ball speed 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 rather acceptable model for the ball movement. Also, no friction is considered to affect the ball, and the model doesn't include any kind of control over the ball.

6.2.1 Model

Given the assumptions made for the ball movement model, the Kalman filter parameters (section 3.2) are mostly known. For this case, we want to maintain a model of where the ball is in the field, knowing that we have measures of the position every ΔT seconds, but these measures are imprecise.

The ball state estimate (it is one of the filter state variable defined in 3.2), which in this case includes its position and velocity, is described by the vector

$$\hat{X}_k = \begin{bmatrix} Pos \\ Vel \end{bmatrix} \tag{6.1}$$

where Pos is the position and Vel is the velocity of the ball at time step k. Since there are no controls on the ball B_k and U (from equation 3.5) are not used, and according to the laws of motion, the prediction of the position calculated by the filter through equation 3.5 is given by

$$\hat{X}_{kpredicted} = F_k X_{k-1}$$

where the state transition model F_k is

$$F_k = \begin{bmatrix} 1 & \Delta T \\ 0 & 1 \end{bmatrix}.$$

The F_k matrixes are ideally equal in each time step k, since the cycle time should be the same. However, it is known to have small variations of ΔT, because the cycle time is not ideal and may vary a few milliseconds.

On the other hand, being the position the only measurement available, the observation model vector H_k is constant and thus simplified to H

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

At each time step a noisy measurement of the ball true position is available. Assuming a normal distribution for the noise, with mean 0 and standard deviation σ_z (as required by the Kalman filter; section 3.2), according to equation 3.4, the observation noise covariance matrix is actually the variance scalar value

$$R_k = E[v_k v_k^T] = [\sigma_z^2].$$

6.2.2 Initialisation

In order for the filter to work, an initial state (i.e. an initial ball position) and an initial covariance matrix must be provided. In an attempt to reduce the filter convergence time by providing a probable position, the initial position is considered to be the first measurement of the ball position. If the initial position was perfectly known, the initial covariance should be a zero matrix, which would indicate no error in that position. However, taking into account that this initial position is not exactly known because it is noisy, the initial covariance matrix must be initialised with suitable large values on its diagonal

$$P_0 = \begin{bmatrix} (1m)^2 & 0 \\ 0 & (1m/s)^2 \end{bmatrix}.$$

The measurement noise σ_z is rather easy to induce, since a good approximation of the measurement deviation can be obtained by reading the positions calculated by the robot for a standing ball and taking the statistical standard deviation of the values.

In an initial approach, this value was made accessible and alterable in the *cambada.conf.xml* file, to allow for adjustments depending on the robots general configuration.

However, later developments were made in order to adapt the filter more dynamically, allowing for the definition of the measurement noise at each time step, dependent on the ball distance to the robot. This change was made because the farther the ball is from the robot, the noisier the measurement is.

As for the process noise, it is not trivial to know a completely justified covariance matrix. Empirically though, one can verify that a null process noise practically ignores the read measures, making the filter define the state based only on its prediction, making it too smooth and therefore unusable (figure 6.2.a). On the other hand, if it is too high, the read measures are taken into too much account and the filter returns the measures themselves as illustrated in figure 6.2.b. The results presented in the figure were theoretically generated with known deviation, to easily visualise the described phenomena.

Face to this situation, one has to find a compromise between stability and reaction. Given the nature of the two components of the filter state, position and speed, one may consider that they do not correlate.

Because we assume a 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. From this,

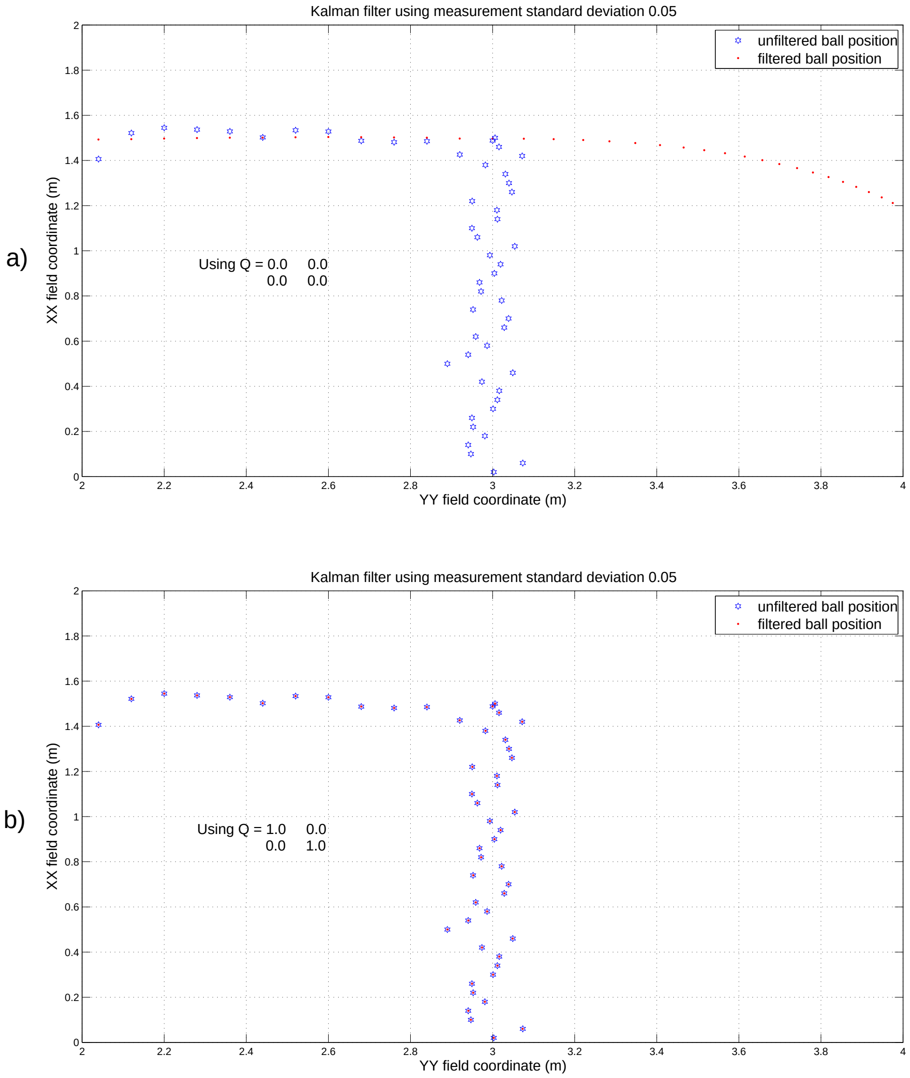


Figure 6.2: Plot of ball positions with a hard path deviation. Measure standard deviation is 0.05m. Read positions are blue, filtered positions are red, being the read positions created theoretically with a 0.05m standard deviation. Top (**a)**): a null process noise covariance matrix is used. Bottom (**b)**): an identity process noise covariance matrix is used.

one can conclude that the filter will have a process noise bigger in respect to the speed than to the position, and thus we'll have a Q matrix in the form

$$Q = \begin{bmatrix} PN & 0 \\ 0 & SN \end{bmatrix} \tag{6.2}$$

where PN is the *position noise*, SN is the *speed noise*.

6.2.3 Implementation

The implementation of the Kalman filter provides noise reduction by estimating the ball position. Through the estimation and given the real measures, it is also possible to implement a system to detect variations on the ball path.

The code of the Kalman filter implementation was built in a class *BallPositionKalman* with attributes:

- `Vec lastMeasure;`

  → estimated position of the ball, first element of the vector $\hat{X}_k$ (expression 6.1), defining one of the filter state variables at a given time step;

- `Vec lastVel;`

  → velocity of the ball, as second element of the vector $\hat{X}_k$ (expression 6.1), defining one of the filter state variables at a given time step;

- `bool lastCycleVisible;`

  → internal boolean variable to indicate the ball was visible in the cycle before the current one;

- `unsigned long lastTime;`

  → value in milliseconds of the time of the last cycle, in order to calculate ΔT with the current time;

- `static int hardDeviationCount;`

  → counter of the number of hard deviations from the prediction that occurred;

- `double readingDeviation;`

  → value of the measures standard deviation;

- `double R;`

  → observation noise covariance matrix, assumes a scalar value as exposed above;

- `double Q[2][2];`

  → process noise covariance matrix;

- `double S;`

  → residual of the covariance matrixes; due to $H = \begin{bmatrix} 1 & 0 \end{bmatrix}$, equation 3.9 assumes the value $P_k(1,1) + R_k$;

- `double covarMatrix[2][2];`

  → current model covariance matrix P;

- `double modelMatrix[2][2];`

  → model matrix F;

- `double kalmanGain[2];`

  → Kalman gain vector K;

and methods:

- `BallPositionKalman( double readingDeviation );`

  → main constructor, calls the method to set noise matrices and initialises the other internal variables to default startup values;

- `void setNoise( double readingDeviation );`

  → sets the noise matrixes according to the wished measurement standard deviation. In the latter implementation with dynamic noise (described latter on this section), this method has to be invoked by the integrator every cycle, to define that cycle calculated measurement standard deviation;

- `Vec filterPosition( Vec readPosition, unsigned long instant );`

  → receives the current time step position and time and returns the estimation of the position. Updates the filter state variables $\hat{X}_k$ and $\hat{P}_k$ and checks for hard deviations on the predicted path, incrementing the internal variable if a deviation is detected. Resets the filter if the ball was not visible, so it may restart the convergence process;

- `void resetFilter( Vec setAsLast, unsigned long instant );`

  → resets the filter to the initial state, with identity covariance matrix, null velocity and no deviations;

- `void setNotVisible();`

  → sets the ball as not visible, for reset purposes;

- `bool hardDeviation();`

  → checks the hard deviation counter for the defined threshold and returns true if the threshold is reached;

The deviation detection is based on the difference between the positions given by the filter state and the real measures. If it is bigger than a given value, the counter *hardDeviationCount* is incremented. The integrator itself is responsible for checking at each cycle if the counter has reached the threshold and act accordingly. Figure 6.3 illustrates a situation where a path deviation would be detected. A ball moving from top left to bottom right encounters an obstacle and is deviated. The difference between the read measures R4, R5 and R6 and their respective predictions P4, P5 and P6 would be a distance longer than the given threshold.

6.2.4 Results

As stated earlier, the variance of the measurements is relatively easy to estimate. Since we can take noisy readings on the ball position, with a static ball one can calculate the statistical standard deviation of the measures.

As for the process noise covariance matrix, several tests were executed and a matrix that proved to achieve acceptable results, keeping a good smoothness/reactivity relationship both with theoretical and practical data, was as defined in expression 6.2 with values PN for the position noise and SN for the speed noise of:

$$PN = \left(\frac{\sigma_z}{3}\right)^2 \qquad SN = \left(0.08\frac{\sigma_z}{\Delta T}\right)^2 \qquad \text{with } \Delta T = 40ms.$$

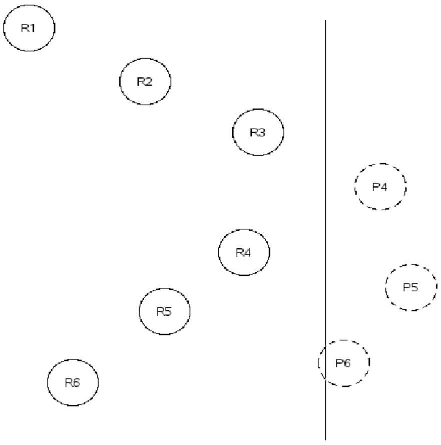


Figure 6.3: This is an hypothetical situation where a deviation would be detected. The differences between R4-P4, R5-P5 and R6-P6 are larger than the threshold.

The general filter performance was tested with a set of theoretical positions of the ball, created with known standard deviation of $\sigma_z = 0.05$. Using the above definition for Q, the filter was executed over those values. The result is a not too smooth, but quickly responsive filtering, illustrated in figure 6.4.

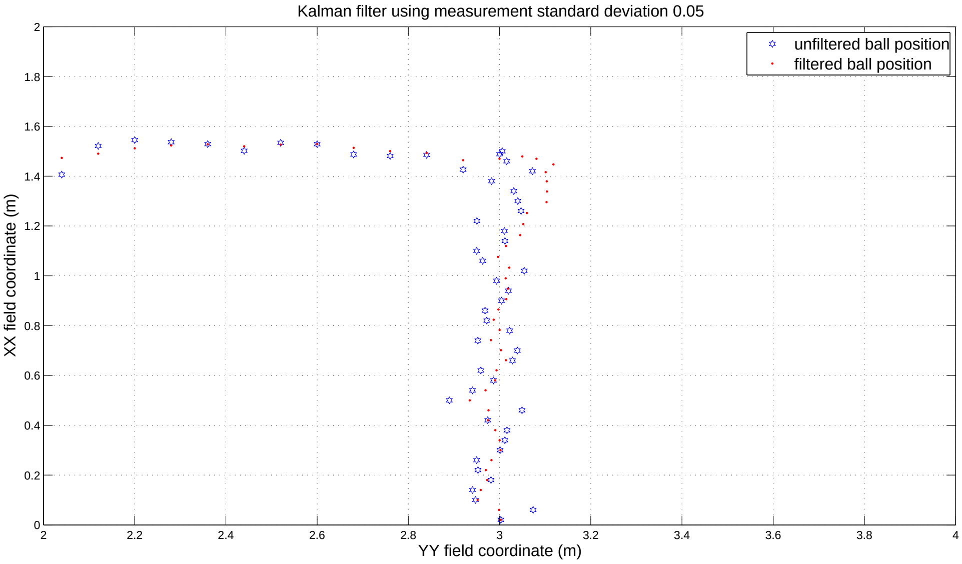


Figure 6.4: Kalman filter over theoretical noisy positions of the ball. The unfiltered ball positions are the blue hexagrams, filtered positions are the red dots.

Two approaches were done concerning the definition of the noise covariance matrices. In an earlier approach, the noise covariance matrices are created when the filter is instantiated and they remain constant. In the latter approach, these matrices are dynamic, they can be changed while the filter is running.

First approach: static robot measurements, constant noise

Given a group of situations where **both** the robot and the ball were in static positions on the field, a few cycles were run and the ball position saved into a file. The plot of one of the noisy measures is represented in figure 6.5. The data was treated in Matlab and standard deviations were calculated, resulting in a mean of approximately $\sigma_z = 0.05$ metres, being this the value of measurement deviation "injected" to the filter.

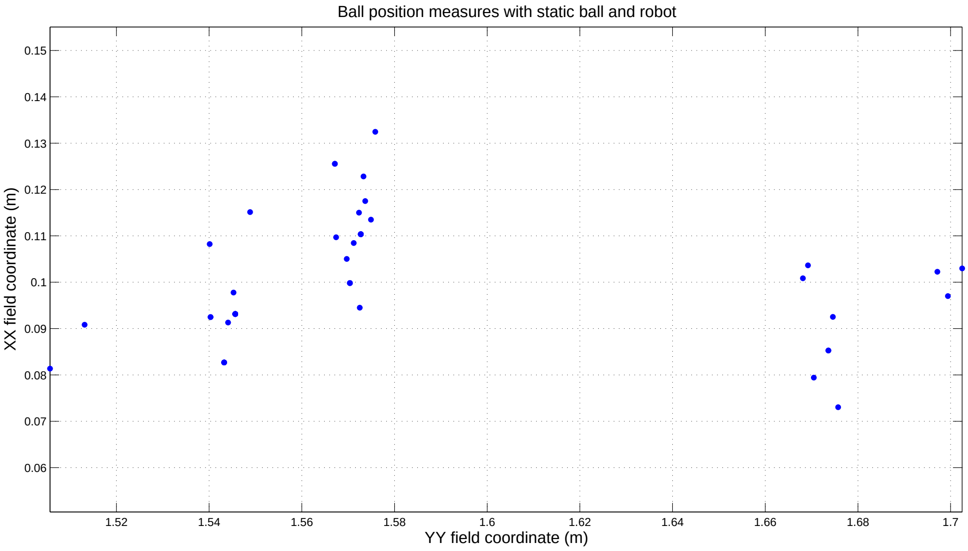


Figure 6.5: Noisy position of a static ball taken from a static robot.

Second approach: rotating robot measurements, noise depend on the distance

Typically in a game, the robot is constantly moving around the field, being its measurements affected by trepidation and its own location noise. To try to reproduce that noisy environment, measurements were taken with the robot rotating over itself keeping the ball static. This experience was performed for several distances. The self rotation movement is one of the movements that causes more errors. A rotation movement causes wider radial displacements with the increase of the radius. Thus, as the ball is farther from the robot, the noise tends to increase.

The experiences were repeated with 4 CAMBADA robots, several rotating speeds were tried and as expected, the faster the robot rotates, the noisier are the measurements. Also, it was verified that the robot could not even detect the ball if the speed and distance were too high. Generally to all robots, at five meters it could not detect the ball at all. Experiences were made for some set speeds of 2π, π and $\frac{\pi}{2}$ rad/s. While the robot was rotating at the defined speed, the ball was gradually moved away from the robot and the approximated distance where it became not visible was registered. After becoming "invisible", the rotation speed was diminished for a lower value and the test resumed (table 6.1).

Rotating speed (rad/s)	Distance (m)
6.28	1
3.14	3
1.57	4
0.00	5

Table 6.1: The right column has the value in meters for which the robot, with the angular speed of the left column, failed to detect the ball. At 5 meters the ball was not detected at all.

Captures were made with the ball at approximately 1, 2 and 3 meters with the robots rotating at 1.57 rad/s and at 4 meters rotating at 0.78 rad/s, since at 1.57 rad/s the ball was not visible. The captures were made for approximately 10 seconds, capturing several tens of position readings so that the standard deviation calculation is as accurate as possible.

Given the two dimensions of the ball position, one should find the length of the vector between the robot and the ball to get a one dimensional standard deviation dependant on the ball distance as wished. By doing so, we get the results presented in table 6.2 for the *distance-standard deviation* relationship.

Distance (m)	Robot 2 StdDev (m)	Robot 4 StdDev (m)	Robot 5 StdDev (m)	Robot 6 StdDev (m)	Mean StdDev (m)
1.0	0.0446	0.1112	0.0704	0.0260	0.0631
2.0	0.1291	0.1495	0.2495	0.0643	0.1481
3.0	0.2575	0.2564	0.5079	0.0813	0.2758
4.0	0.5087	0.4738	1.0310	0.3963	0.6024

Table 6.2: The table shows the calculated standard deviations for the captured measurement distances.

The measurements at the 4 different distances are represented in figure 6.6. The readings of the ball position are greatly affected by the robot distance map, since the vision system resolution decreases with distance, as discussed in section 4.3.

The mean on the measurements presented in table 6.2 was considered to be the standard deviation data set. Given the known set of points, a polynomial to fit them was necessary. Using the Matlab *polyfit* function, the best fit in a least-squares sense can be calculated. Tries were made with 1st and 2nd degree polynomials. The second degree polynomial was chosen:

$$\hat{\sigma} = 0.04d^2 - 0.02d + 0.016$$

where $\hat{\sigma}$ is the estimation of the standard deviation for the distance d between the robot and the ball. The 2nd degree polynomial is used because the 1st degree polynomial does not fit the data properly, and assumes negative values for positive distance, which is not acceptable (figure 6.7). Given the few known points, a 3rd degree polynomial would perfectly fit all the 4 of them. However, these known points are also estimated and thus cannot be taken as exact. For that reason, a curve that would exactly fit them is not desirable. Also, a quadratic expression was in someway expected, as the distance map line (figure 4.6) has a similar curve.

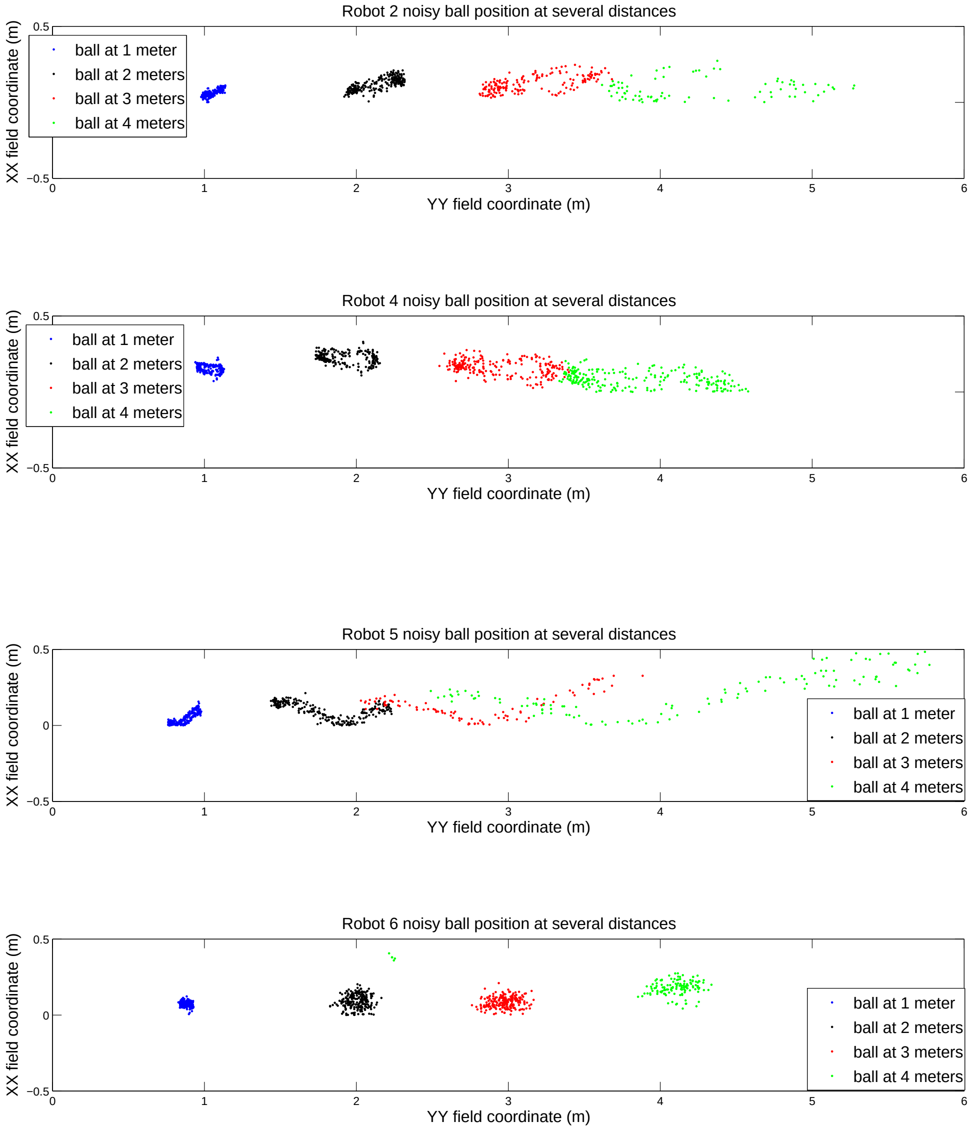


Figure 6.6: Noisy position of a static ball taken from rotating robots.

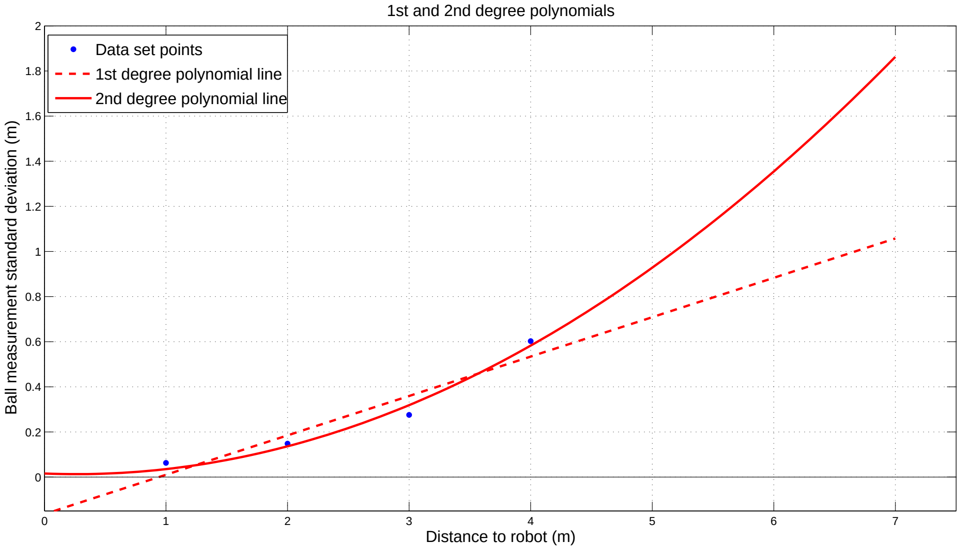


Figure 6.7: Representation of the standard deviation value for variable distance to the robot. Data set points as blue dots. 1st degree polynomial as dashed line, 2nd degree polynomial as solid line.

Having these two approaches, it was verified that the results of the second one were slightly better than the first. In a capture of the values estimated by the robot, illustrated in two parts by the two figures 6.8 and 6.9, the read measures given by the vision are represented with black dots. The robot is slightly deviated from the centre of the field (-0.08, -0.02) and the ball is kicked against it and then goes free, with a slightly curved path after the collision (caused by crests on the field carpet). For plot interpretation, one must keep in mind that both the ball and the robot are represented by their centre and have a radius, and thus, when the ball hits the robot, it is about 25cm from the robot centre. The Kalman filter has been applied to the read positions, both the constant noise approach and the dynamic noise approach. Red stars represent the positions given by the first approach and green hexagrams the ones given by the second approach. We can verify that when the ball moves with some speed, the constant noise approach tends to get more delayed (figure 6.8; in the initial approach on the robot, the ball was moving with considerable speed). Besides, as the ball was going for the robot, the noise was progressively smaller and thus the second approach tends to be more correct.

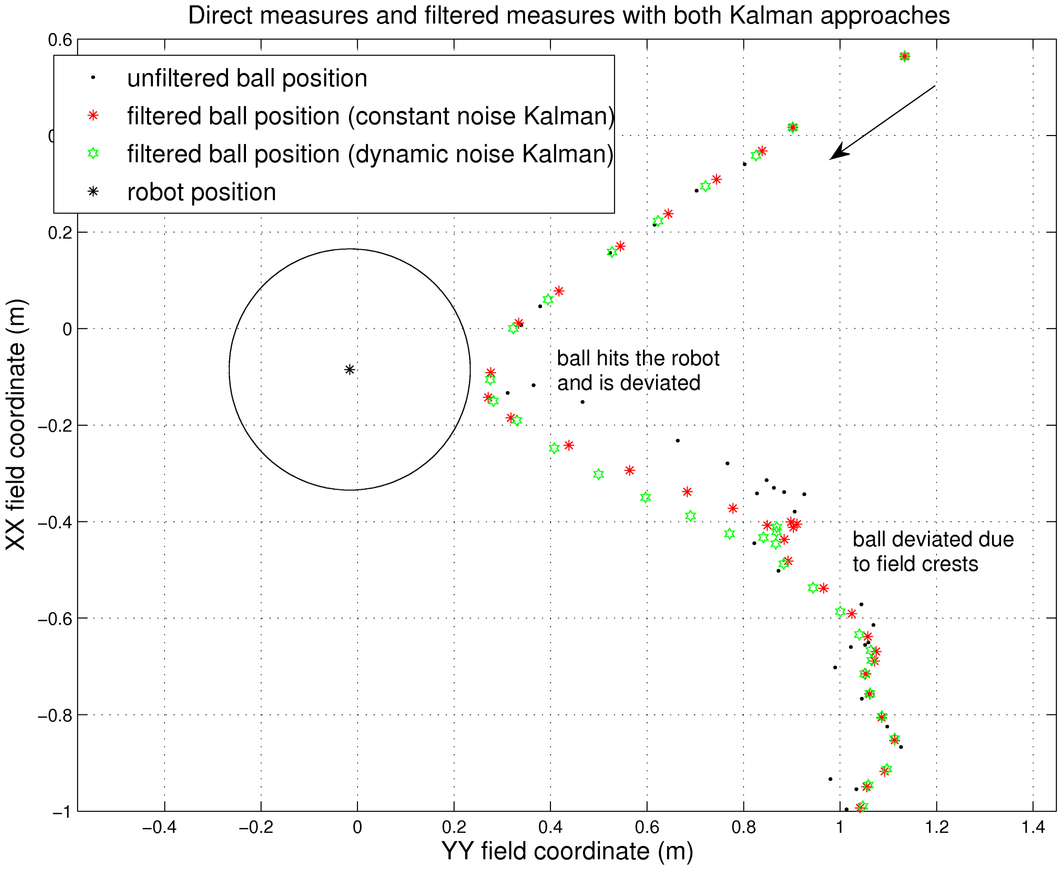


Figure 6.8: Plot of the first part of a ball movement situation, which pairs with figure 6.9. The read measures of the position are represented with black dots. Positions filtered by the Kalman filter with constant noise represented with red stars. Positions filtered by the Kalman filter with dynamic noise represented with green hexagrams. The robot position is represented by the black star.

Also, near the graphic coordinates (1.3,-1.9) (figure 6.9), a measure with a higher than average displacement is clearly seen. That measure deviates the constant noise Kalman filter more than the dynamic one, as at the given distance (around 2m), the noise calculated by the polynomial is 0.11m, which is more than double of the constant approach noise. This phenomenon tends to increase with the distance.

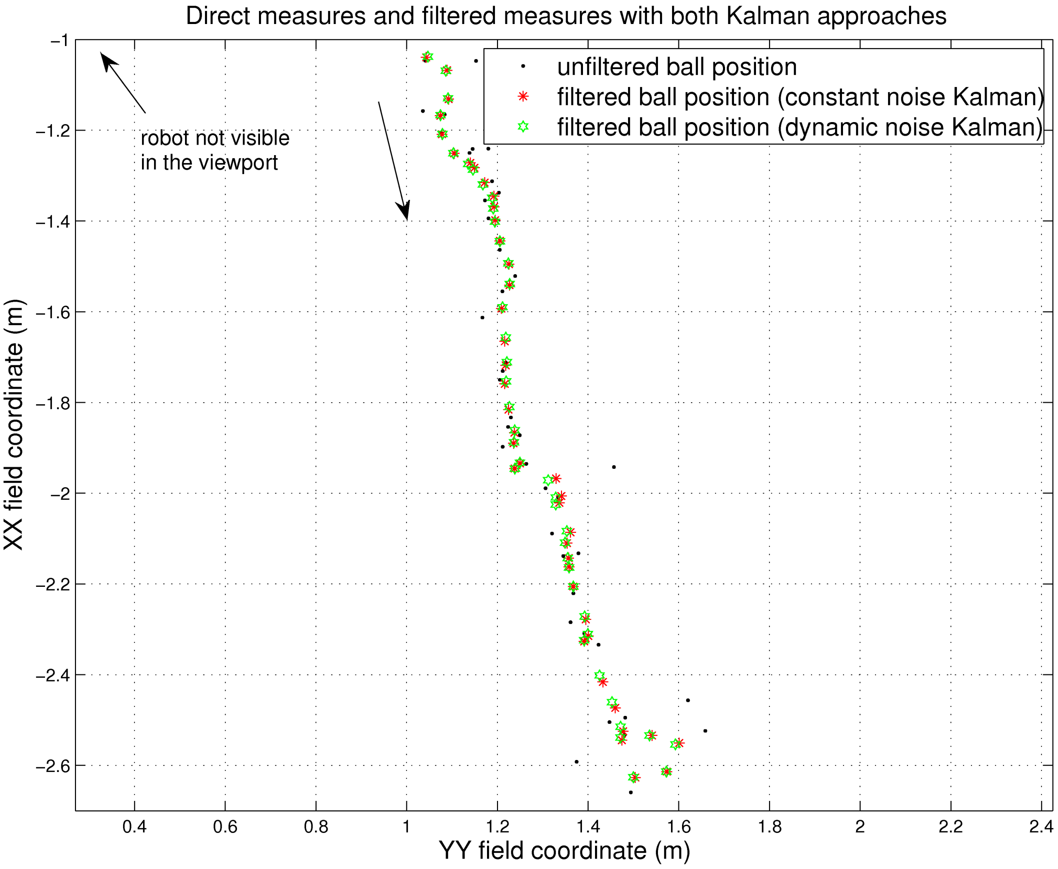


Figure 6.9: Plot of the second part of a ball movement situation, which pairs with figure 6.8. The read measures of the position are represented with black dots. Positions filtered by the Kalman filter with constant noise represented with red stars. Positions filtered by the Kalman filter with dynamic noise represented with green hexagrams. The robot position (-0.08, -0.02) is not possible to present within the defined view window.

Another capture of ball movement with deviations on its path is illustrated in figure 6.10. The robot is standing at position (0.2501,- 2.4207) and the ball moves from middle right to down right, then moves left and is again deviated, going up. It is visible that after filtering, the ball path positions are more steady, and hopefully will be able to provide a better estimation of the real velocity (discussed in next section). Once more, the green hexagrams representing the dynamic noise Kalman approach proves to be less delayed and more accurate than the constant noise approach represented by red stars. The dynamic noise approach is faster to converge when a deviation occur, which is a requirement, as the ball movement is very variable in a game environment. Thus, the implementation of the second approach was chosen to be used and the presented results from now on are based on that approach.

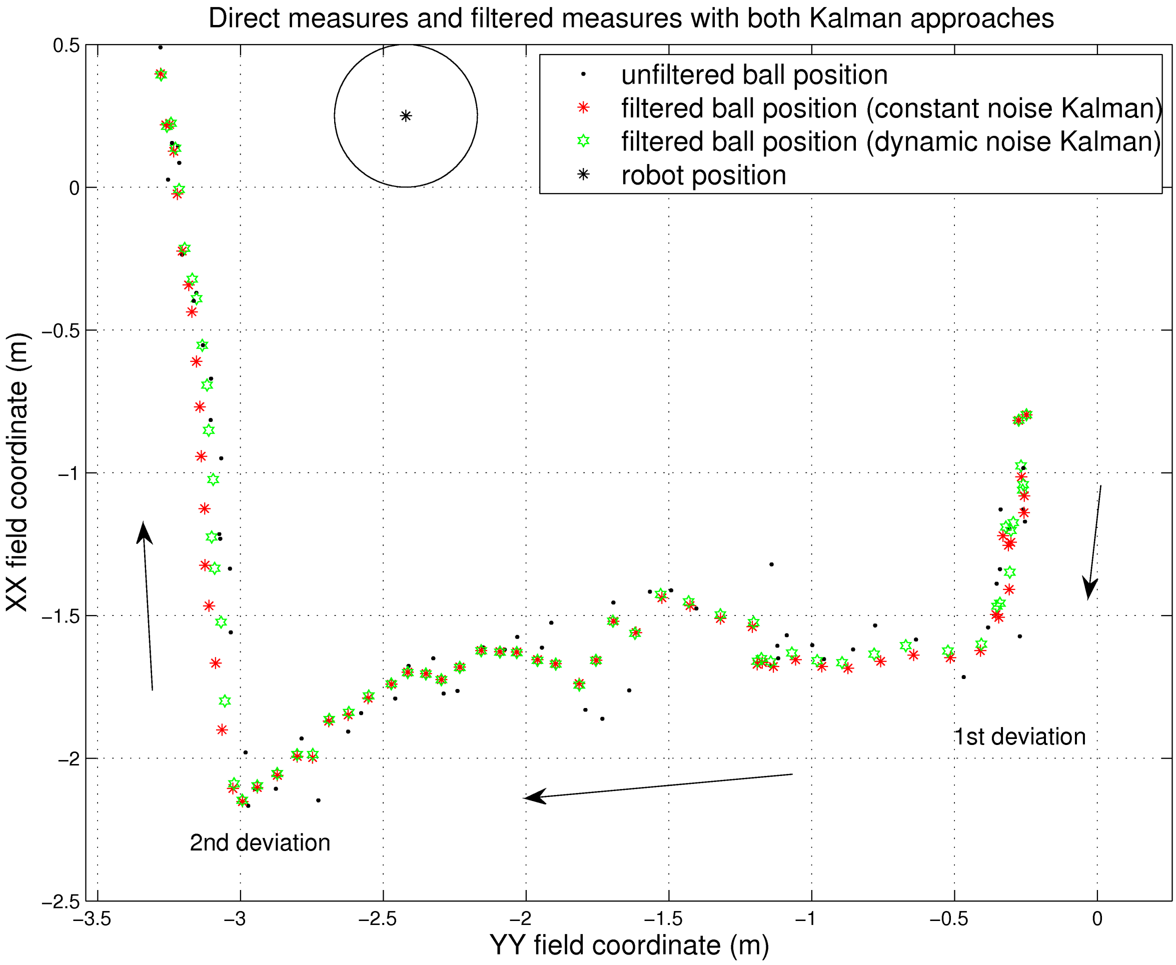


Figure 6.10: Plot of a ball movement situation. The read measures of the position are represented with black dots. Positions filtered by the Kalman filter with constant noise represented with red stars. Positions filtered by the Kalman filter with dynamic noise represented with green hexagrams. The robot position is represented by the black star.

6.3 Velocity estimation

The calculation of the ball velocity is a feature becoming more and more important over the time. It allows that better decisions can be implemented based on the ball speed value and direction. Assuming the same ball movement model described in the previous section, 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. The same principle is assumed for the robot velocity, as its positions are also affected by the noise of the localisation algorithm.

To keep a calculation of the object velocity consistent with its displacement, an implementation of a linear regression algorithm was chosen. This approach based on linear regression is similar to the Tribots team [42] velocity estimation. By keeping a buffer of the last m measures of the object position and sampling instant 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, although it is made in a transparent way, so the description in this chapter is presented generally, as if only one dimension was considered.

6.3.1 Model

It is known that the measurements of the object position are imprecise, and the time instant of each sample is available. For keeping the record of a linear velocity on the object over these conditions one can only assume the time instant as the independent variable and the measurement as the dependent one. These results in a regression of the position on the time instant. Being so, we have $x = \Delta T$ and $y = pos$ in the regression model (defined in section 3.1). The model keeps the last m measurements and respective instants as its data set. The ΔT associated with each measurement is the difference between the most recent sample instant and the sample instant of the respective buffered value. For m samples, each ΔT_i, associated with sample i is defined as $\Delta T_i = t_m - t_i$, where t_m and t_i are the time instant of samples m and i respectively.

With these variables, time and position, the fitting line gains a physical meaning. The object velocity is given by the slope of the line, the first derivative of position over time. In this case, there is little interest in predicting the position of the object in a given instant, so the α parameter is not used. Thus we define β by equation 3.1 as

$$\beta = \frac{m \sum \Delta T_i pos_i - \sum \Delta T_i \sum pos_i}{\Phi}$$

being Φ by equation 3.3

$$\Phi = m \sum \Delta T_i^2 - \left(\sum \Delta T_i \right)^2.$$

6.3.2 Implementation

By keeping a data set and calculating the slope of the line that best fits that data, we can obtain a speed and direction based on a history of the positions, which define the true object path much better than only two consecutive measures. The bigger the buffer, the smoother the variations on the speed direction would be. On the other hand, smoother the variations, longer the regression would take to converge facing a deviation of the object path, as older positions would be very far from the new path. In order to try to make the regression converge more quickly on such situations, a reset feature was implemented, which allows deletion of the older values, keeping only the n most recent ones. This reset is currently only used by the ball regression and can be activated when a deviation is detected by the ball position filter. Also a full reset that cleans all the buffered data is avaliable, to completely reset the regression when the ball is not visible, as it can become visible again in virtually any place on the field.

The linear regression code itself was built into a class *LinearRegression* with attributes:

- `unsigned int MAXSIZE;`

  → maximum allowed size for the buffers;

- `struct timeval instant;`

  → current cycle time instant;

- `deque<Vec> posBuffer;`

  → buffer of object position;

- `deque<struct timeval> timeBuffer;`

  → buffer of time instants;

and methods:

- `LinearRegression( int maxSize );`

  → main constructor, defines the internal *MAXSIZE* as *maxSize*;

- `void putNewValues( Vec pos, struct timeval instantIn);`

  → method to insert new values in the internal buffers and define the internal instant as the current one;

- `void clearBuffs();`

  → method to clear the internal buffers;

- `Vec getDeclivity();`

  → method to calculate the β parameter of the regression equation, using the current $x = times$ and $y = positions$ values in the buffers. This β parameter is the slope of the fitting line, is physically interpreted as the object speed;

- `Vec getPointInOrig();`

  → method to calculate the α parameter of the regression equation, using the current $x = times$ and $y = positions$ values in the buffers. The α parameter is the y value when x is zero;

- `void resetToNSamples( unsigned int nSamples );`

  → method to clean the older values in the internal buffers, leaving only the *nSamples* most recent ones;

6.3.3 Results

We will consider a capture of values of a ball movement with three deviations. This movement is illustrated in figures 6.11, 6.12 and 6.13. The ball had movement from left to right when the capture started, but almost immediately a deviation has occurred, and the ball moved in a downward diagonal. A new deviation made it move left, almost parallel to the horizontal axis, and finally, the first few samples of another deviation making the ball move up are also present. In all the plots, ball positions are represented as blue dots and the ball velocity vector of the corresponding sample is represented by a red line.

In figure 6.11, the velocities of each sample were estimated by instant velocity calculation, that is $\frac{\Delta D}{\Delta T}$ of consecutive samples. It is clearly not a good estimation, as the velocity vectors are too erratic and give practically no idea of the ball direction.

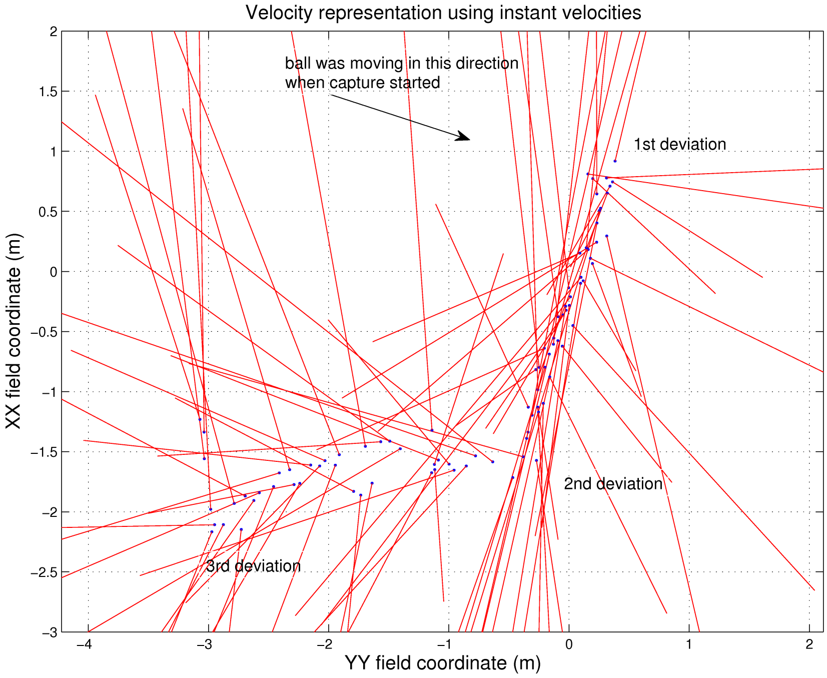


Figure 6.11: Velocity representation using consecutive measures displacement.

However, applying the linear regression algorithm, the velocity vectors are much more steady, and give a good insight of the ball direction (figure 6.12). Tests show that when a hard deviation on the path occurs, the velocity vector tends to rotate to the new path direction, taking m cycles to become stable, being m the size of the buffers in the model. This effect can be identified in figure 6.12, when path deviations occur.

To further refine the velocity estimation, the linear regression algorithm is applied over the Kalman filtered positions. With more stable positions, the velocity estimation also obtains better results. The velocity vectors in each rectilinear movement are usually more narrow and in the presence of deviations, the velocity vectors converge more quickly to the new path (figure 6.13).

The ball velocity kept on the *Worldstate* is then the one estimated by the linear regression over the Kalman filtered positions. Given the noise that is always present, to avoid residual velocities to be kept and used, a threshold (applied after the velocity estimation) was considered to define what is the minimum speed to consider the ball is in movement. A threshold of 0.2 m/s is currently used. In practice, a ball moving at less than that may be considered as being static, since the position noise itself can induce residual speed of that order to the ball.

Although the Kalman filter described in the previous section internally estimates a ball

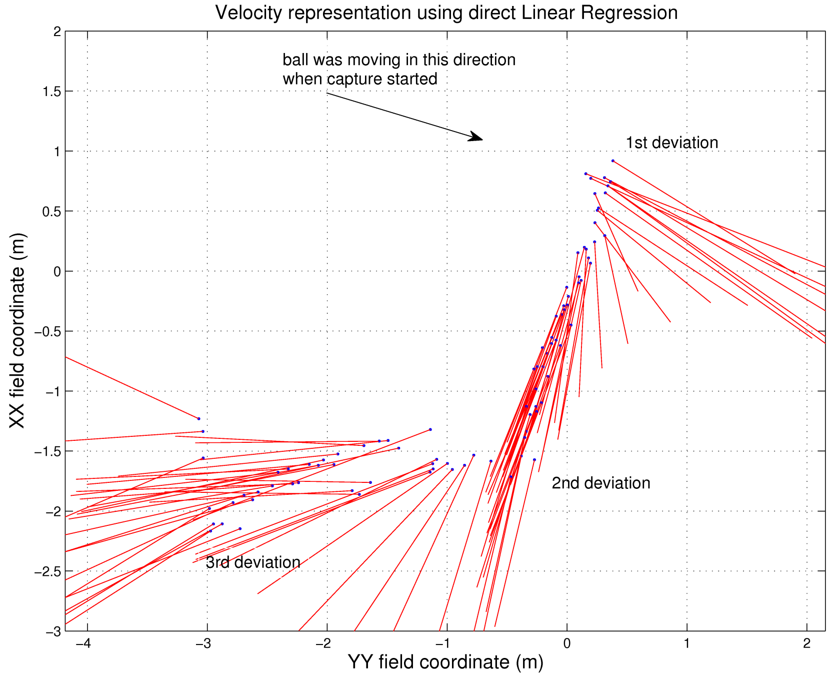


Figure 6.12: Velocity representation using linear regression over direct position measures.

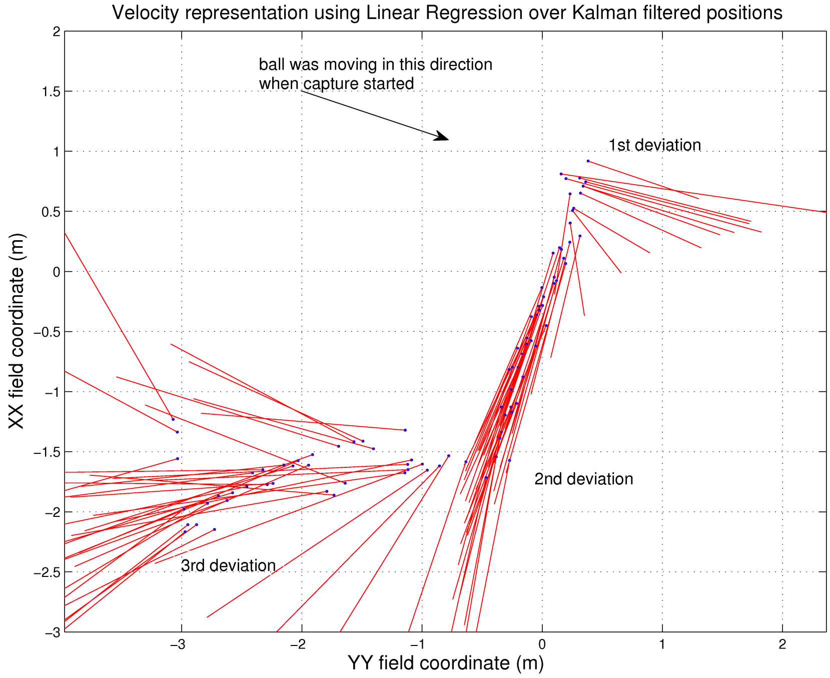


Figure 6.13: Velocity representation using linear regression over Kalman filtered positions.

velocity that could be used, tests show that the Kalman internal velocity is not desirable for several reasons. See appendix A for details on the choice of linear regression for velocity estimation.

6.4 Compass

One of the aspects of localisation is the orientation of the robot. In the MSL environment, by considering the field lines, one can obtain the robot's orientation, by matching the seen lines relatively to the robot (section 4.5). However, each position on the soccer field has a symmetric position, relatively to the field centre, that "sees" exactly the same field lines and thus the problem of knowing in which side of the field the robot is remains. In previous editions of RoboCup, there were coloured goals, which allowed to solve this problem by identifying the goal colours on each side of the field. Without the colours, a disambiguation method was required. The way to solve it was to install an electronic compass on the robot.

Knowing the field of play, the angle of one of the goals is measured and registered in the *cambada.conf.xml* file. This angle is the "north" that really matters for the robot, the magnetic north is not important. The *Compass* module keeps the angle difference between the read magnetic north and the "goal north", as the angle from the reference goal is the one necessary.

Knowing the angle from the given goal reference, the orientation estimated by the localisation algorithm can be compared to the orientation given by the compass. Since the compass readings are noisy, and only a general orientation is required, an absolute error up to 45 ° is allowed. If the absolute error is too high, larger than 135 °, the robot position is changed to the symmetric field position, as the evaluated position is not the real one, but the symmetric position on the other side of the field. This is usually addressed as mirrorring. If the absolute error is between the 45 ° and 135 °, a counter is incremented (figure 6.14). If a threshold of consecutive erroneous measures is achieved, the robot considers it is lost, and the algorithm to find the initial position is re-run.

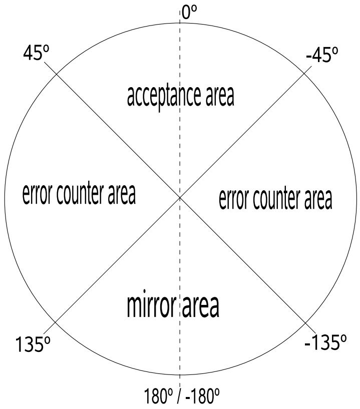


Figure 6.14: Illustration of the compass error angle intervals.

6.4.1 Results

Prior to the compass instalation, the disambiguation of the localisation was made based on the seen goal colours. In some situations (robot in corner, for example) none of the goals were visible and thus, when lost, there was no reference goal colours for the localisation algorithm to use. By not depending on the goal colours, as long as the robot sees enough field lines, with the compass it can know in which side of the field it is. The vision system performance was also increased, as it no longer needs to identify the goal colours or estimate their positions for localisation purposes.

Figures 6.15 and 6.16 illustrate parts of a capture of the robot orientation values. In both, the blue line represents the orientation values in each cycle estimated by the localisation algorithm, the black line represents the orientation read by the compass. In the beginning the robot was still and then started to rotate (figure 6.15). It is visible that there are differences between both values, as expected, but none reaches the defined maximum of 45 °

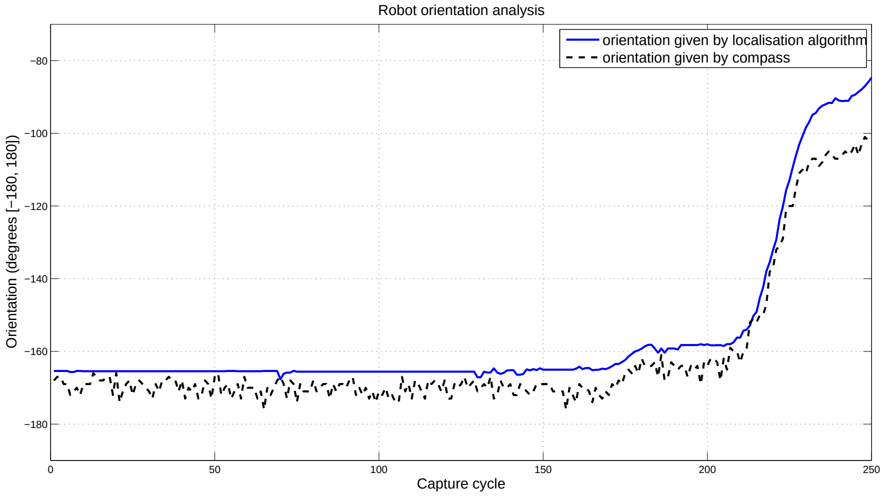


Figure 6.15: Illustration of the orientations estimated by the localisation and read from the compass.

Further in the same capture, the robot was forced to get lost for two times. In the first situation, the robot was forced to get lost by covering the camera for some cycles while moving the robot around. Without visual information, the estimated localisation gets more and more erroneous, eventually with values above the maximum accepted 45 ° (figure 6.16.a). In the second situation, the robot was tilted. That causes the visual information to be wrong since the vision system ceased to be perpendicular to the ground and thus the estimated orientation got wrong (figure 6.16.b). In both situations, the cycles where the 45 ° threshold was passed are represented by a red line connecting the orientation lines.

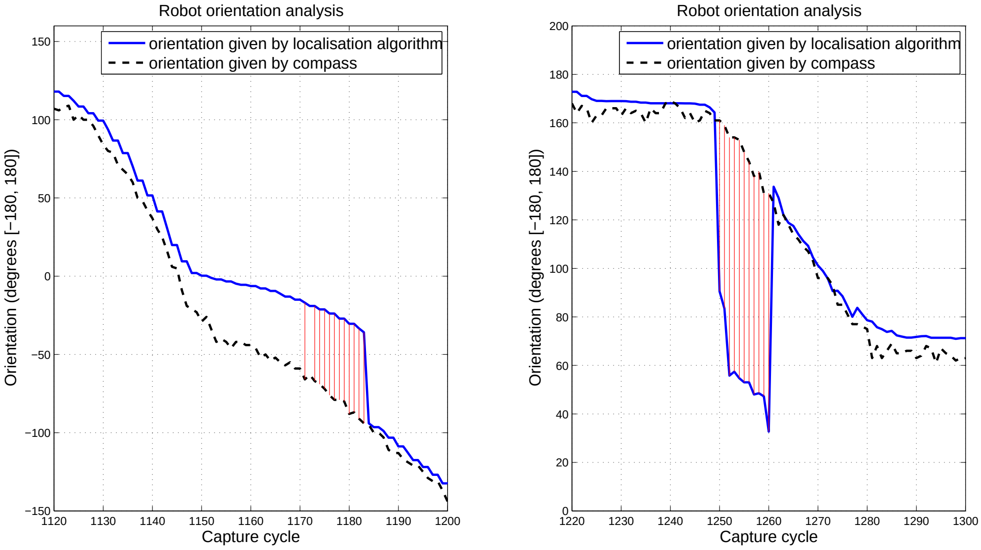


Figure 6.16: Illustration of two situations where relocalisation was forced. Right **a)**: the camera was covered while the robot moved. The estimated orientation degrades progressively and after getting higher than 45 ° the counter starts and forces relocation; Left **b)**: the robot tilted. The estimated orientation is immediately affected by more than 45 ° and the counter starts and forces relocation.

6.5 Ball position integration

The integration of the ball is an important issue, as it is the main object on the game. Most of the decisions on the higher-level depend on the information available on the ball.

The vision information on the ball is composed by two attributes, the ball relative position and the number of cycles it has not been visible. This second value is important, as the vision position is maintained if the ball is not detected in the cycle, while the counter is incremented. The cycles not visible is the way to distinguish if the position attribute is the current position or the “last seen” position. Figure 6.17 illustrates the general ball integration activity diagram, with the characteristics explained below.

6.5.1 Integration using self visual information

The omni-directional vision system is considered the main one, and thus is the primary source of the ball visual information. The ball is not considered invisible while the cycles not visible count does not reach a given threshold. This is done to avoid situations where by some reason the vision could not detect the ball for a small fraction of second (passing behind another robot for instance) would make the ball immediately not visible, and the agent would eventually give up on it. If the ball is not visible by the omni-directional vision system, the frontal vision system can be used the same way as the omni-directional is.

The absolute ball position is calculated and validated. If the cycles not visible count is zero, meaning the ball has been visible in consecutive cycles, the position and time instant are passed to the Kalman filter to get the filter estimate that is then injected to the linear regression buffer, along with the time. This is only done when the ball is visible in consecutive cycles because filtering or injecting the same position with different times would create an undesirable error, and since the time is considered on both models, the estimates don't need to be exactly cyclic. Either way, the velocity is estimated by the linear regression based on the current buffer values. The ball structure is filled with the current data, on position, velocity, visibility and engaged status. The engaged status is given by a fusion of ball position and the ball barrier sensor. As for visibility, the type is set as *Own*, as the ball information is given by the robot own sensors.

6.5.2 Integration using multi robot information

If none of the cameras can see the ball, a multi robot ball integration algorithm is executed. For all the running team mates that have the ball visible and validated by its own sensors, the absolute position is verified. Two approaches were implemented.
Through a function to get the statistics on a set of positions, mean and standard deviation, one can get the mean value of the position of the ball seen by the team mates and assume it as its own. Another approach is to simply use the ball of the team mate that has it closer to itself. The latter has been the one used recently, because the agents still get many false positives on the ball position, but usually not at short distances. With the mean ball, a single agent could compromise the mean with a false positive. However, with the recent and certainly future improvements on the vision, a mean approach would probably be better and safer. Whatever the case, the ball is assumed visible with type *Known*, to indicate that the ball is not seen by the self agent.

If no agent can see the ball, it is inevitably assumed as not visible.

Currently, the ball validation is made only by one criterion, its position needs to be inside a given margin around the field. Additionally, the position given by the omni-directional camera must be within a given distance, considered by the vision team to be a reliability limit. Another criterion that would make sense would be to validate the own ball position with the mean of the team mates, once more, only applicable if there is a high degree of certainty on the readings.

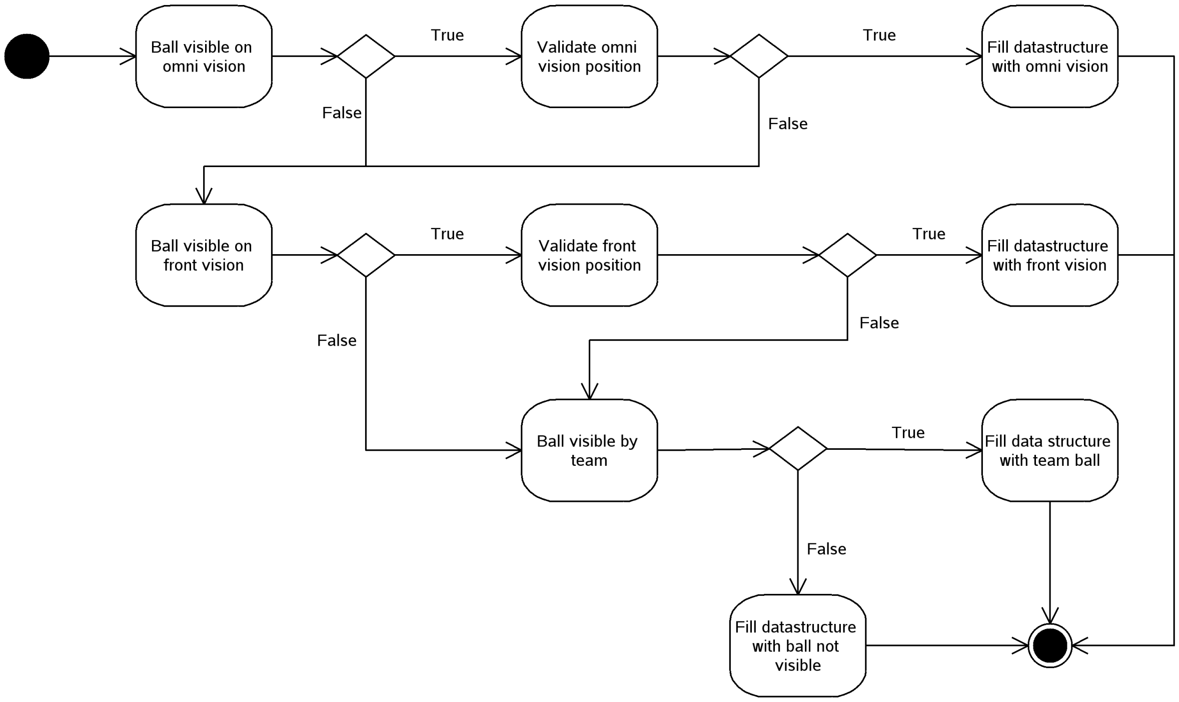

Ball visible on omni vision
True
False
Validate omni vision position
True
False
Fill datastructure with omni vision
Ball visible on front vision
True
False
Validate front vision position
True
False
Fill datastructure with front vision
Ball visible by team
True
False
Fill data structure with team ball
Fill datastructure with ball not visible

Figure 6.17: Ball integration activity diagram.

6.6 Summary

In this chapter, the integration structure was presented, giving an insight of which data is available and how it is treated. An implementation of a Kalman filter to filter the ball position was presented and explained. The obtained results used for estimation of the filter parameters are presented and discussed, as well as the general filter performance results. The choice of the second approach, using dynamic noise, should be clear after analysing these results.

Another implementation of a sensor fusion technique is presented, the linear regression. This was chosen to be used to estimate the ball and robot velocity. The parameters and implementation description was presented. It was chosen to present results over the ball velocity estimation, as the robot estimation is much similar, but with less noise on the measurements. By comparing the presented results, it should be clear that a linear regression estimation provides a better representation of the ball velocity, especially concerning direction. Also, the improvements achieved by applying the linear regression over the Kalman filtered ball positions rather than the direct measurements important for the desired performance boost.

The integration of the new electronic compass information was explained. The compass allowed the adaptation of the team to the new uncoloured goals. Although it was a necessity due to rules changes, it proved to improve the performance of the robot localisation.

Knowing the ball information is one of the most important requirements in the game. Thus, the integration of the ball can be done in several ways, to try to minimise the time it is not known at all. The prioritised structure of the ball integration was presented and illustrated.

Chapter 7

Behaviours

A behaviour, as simply put by Arkin [43] adopting the concept advocated by the behaviorist school of psychology, is a reaction to a stimulus. Pure reactive robotic systems would react directly to perception. However, CAMBADA agent behaviours use abstract representations and time history to decide how to react, defining actions like move to a given point or object, kick the ball, dribble, among others. Although the behaviours themselves would not possess any form of "intelligence", by providing an architecture that couples the various behaviours, the complete system inherits the property of flexible reorganisation under the influence of its own sensor information [44]. Thus the CAMBADA architecture couples these behaviours by using and combining them in a higher level role in order to produce effective results on the field. This chapter describes work done at the behaviours level, both new behaviours and refinement of existing ones. The current CAMBADA behaviour structure is illustrated in figure 7.1

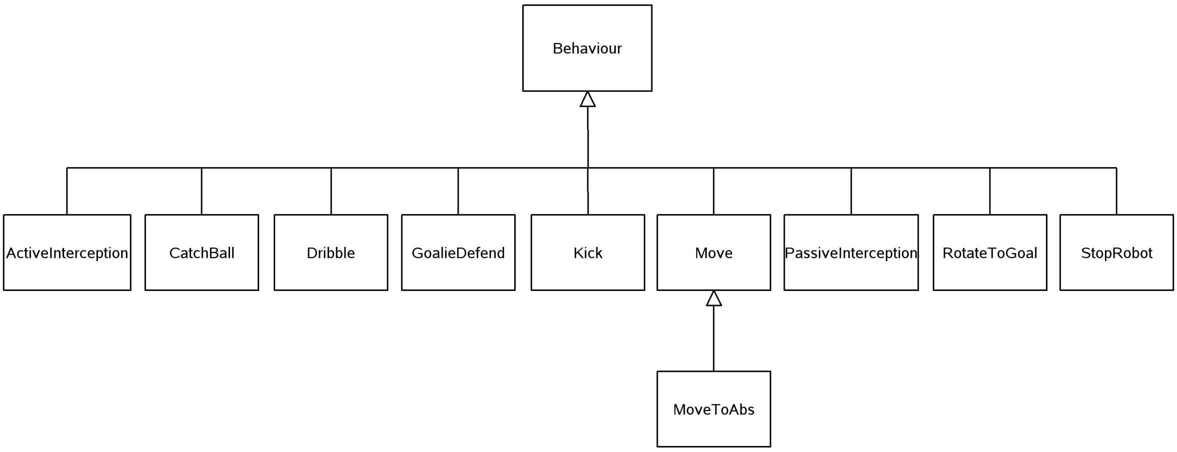


Figure 7.1: Structure of the behaviour classes.

In the scope of the accomplished work, three new behaviours were defined, *PassiveInterception*, *ActiveInterception* and *CatchBall*. A new algorithm for ball avoidance was created in the *Move* behaviour and some changes and minor refinements were made to several others, although they are not documented here.

The *Behaviour* class is the one that keeps the velocity to apply on the wheels, through 3 attributes *velX*, *velY* and *velA*, for velocity's x, y and *angular* components, respectively. Given the specificity of each different behaviour, they inherit the responsability of calculating the values of the velocity by its own means. Also, the control of both the kicker and the grabber is kept by the *Behaviour* class.

In each particular behaviour, the calculation of the robot's velocity is handled by a PID controller. Given a target point, in relative coordinates, the distance between the robot and that target point (the robot is the axes origin) is the PID input and treated as an "error" that should be null. The PID controller returns the speed value necessary to reduce the distance to zero, according to its defined parameters. For linear velocity, the components *velX* and *velY* have to be calculated from the returned value, angular speed *velA* can be defined directly. The implementation of the PID also includes a maximum value for the output which allows to limit the values to avoid over reaction, mainly concerning rotational movements that can be very fast.

All the results presented in this chapter were practically obtained from captures of the positions estimated by the agent.

7.1 Passive interception

The need for a way to approach the ball other than moving directly towards it becomes evident as it is verifiable that the robot would describe a completely unnecessary arc to get to the ball (situation illustrated in figure 7.2) and in most cases would have to chase it back to its own half field, loosing much time and strategic advantage.

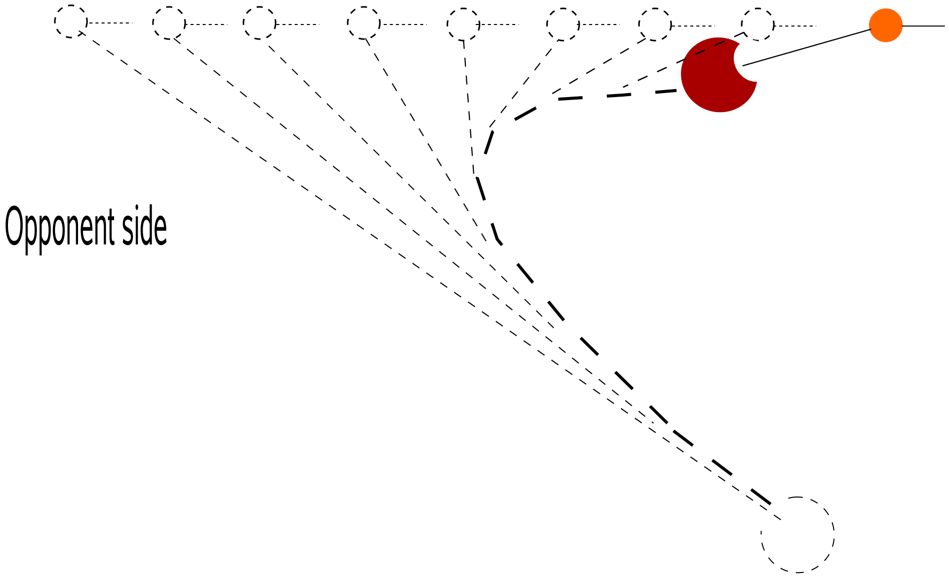


Figure 7.2: Illustration of an approach to ball. The ball is moving in a straight line, coming from the opponent field. At each cycle, the ball and the robot trajectory is represented by the dashed line. The bold dashed line is an approach on the true path the robot would follow.

One approach to intercept the ball is to passively "wait" for it. This approach could have advantages in situations where the robot should not go for the ball, but rather intercept its path and then, if the ball reaches it, move forward with it. The used approach is based on the ball velocity direction. The ball speed is extended as a line segment with origin in the ball. That line represents the ball path and the closest point of the path line to the robot position is calculated; geometrically, the perpendicular point is always the nearest. The calculated point is the one where the robot should go to in order to intercept the ball path (figure 7.3).

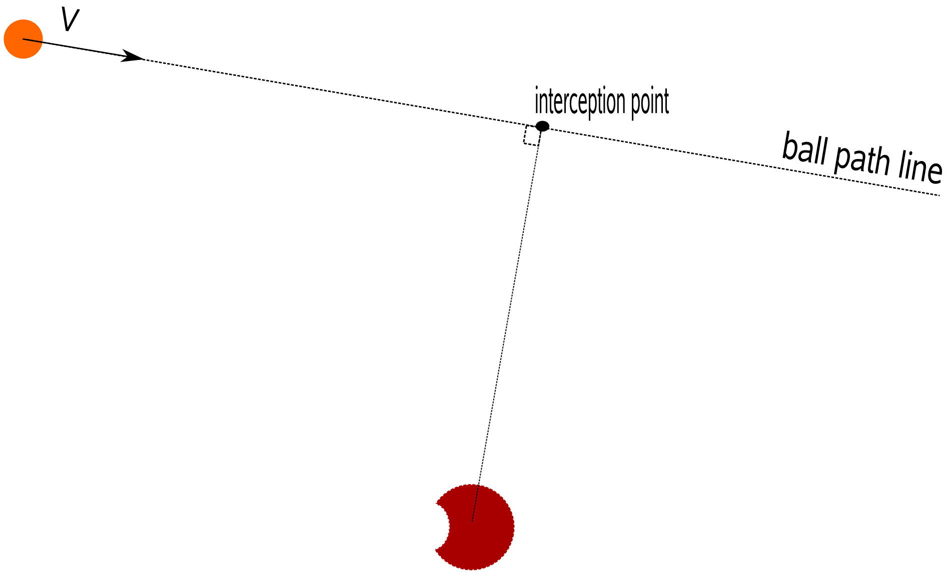


Figure 7.3: Illustration of the passive interception behaviour. The ball velocity vector V is extended as a line defining its path. The robot calculates the closest point over the line, relatively to itself, and defines it as the interception point.

If the ball is static, it makes no sense to try to calculate the interception of its path, and the ball itself is considered the target. Also, as the ball may become not visible for some time (it can for instance pass behind some other robots), a certain tolerance has been implemented, to prevent situations where the robot would automatically give up on the interception as it didn't see the ball for a few cycles. When that is the case, a counter is incremented and the previous interception point is used. Only when the counter reaches a given threshold the behaviour stops its actions.

7.1.1 Results

In order for this behaviour to be effective, it must be tunned to be very reactive as we want the robot to get to the ball path as soon as possible. Being so, it is not a precision behaviour, so for final approach on the ball, another smoother behaviour should be activated. Being the case, a PID controller for translation was defined with a high P component, 2.0, so the distance is rapidly corrected, being the limit 2.5m/s, roughly the robot maximum linear speed. Also, for a little "prediction" of the movement slope, a D component of 0.5 was defined. The rotation was also defined with high values, P=2.2 and D=0.8, for also quick angle correction, with maximum defined limit of 3.0rad/s.

This behaviour does not guarantee the robot gets to the interception point before the ball, as it only evaluates its direction and goes to the path.

Several captures of passive interception situations were made, to test and validate it. Figure 7.4 is a plot of the ball and robot positions of one of these captures. The ball positions are represented by blue dots and the robot positions are represented by red dots. The robot and the ball were stopped when the agent process was started. After a few cycles the ball was thrown and the robot activated to make a passive interception. The captured data allows to visualise the paths of both the robot and the ball. The capture was stopped after the robot chased the ball for some cycles.

In the plot, the ball moves from down right to top left in a practically straight line. The jump in the positions around plot coordinates (3, -1.5) was originated by a very fast rotation of the robot, as soon as it was activated. However, afterwards the positions are quite stable. The plot was obtained by a Matlab script built to show the evolution of interception captures. At each sample, a line representing the unitary velocity vector of the object is drawn, as well as small green circle around the current sample. The calculated interception point is represented by a small black circle, in this case, near (-0.5, -0.1). It is visually verifiable that the interception point is, as expected, the perpendicular interception of the ball velocity direction line and the robot velocity direction line. By the paths followed by both the ball and the robot, one can also verify that the robot did not intercept the ball, as at certain point, its path cease to be perpendicular to the ball path and begins chasing it.

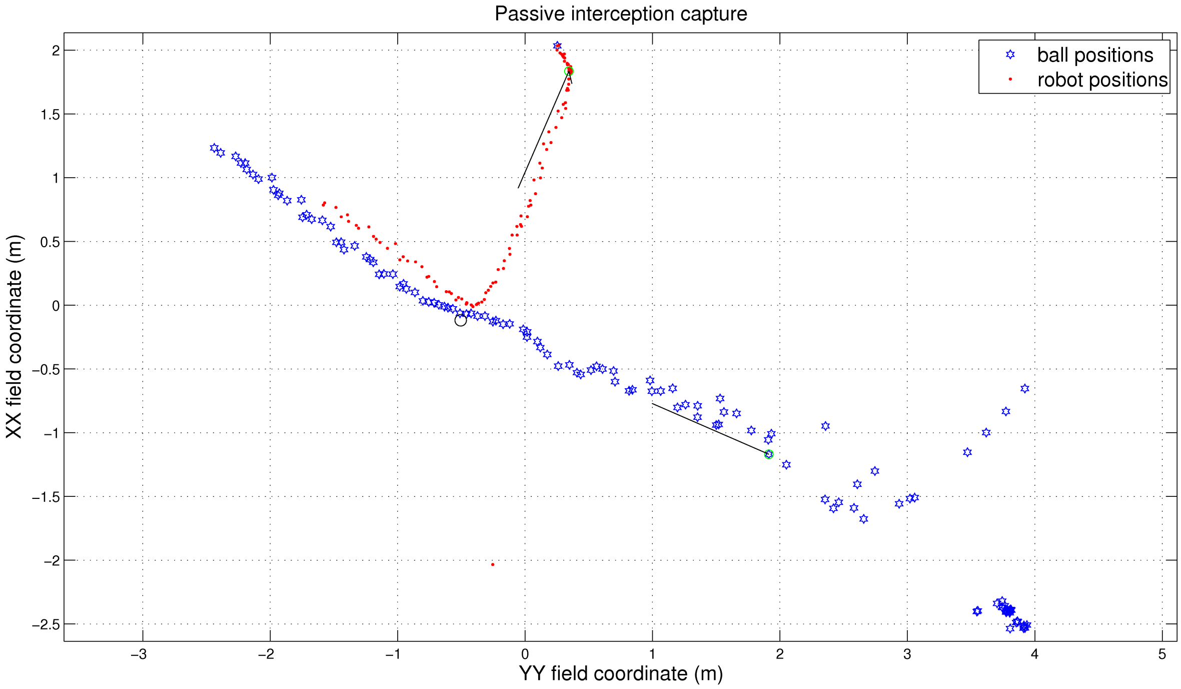


Figure 7.4: Plot of a capture of a passive interception (the presented positions are estimated by the robot). See text for details.

7.2 Active interception

The active interception appears naturally from the passive one, as in certain situations one is interested in getting in front of the ball in minimum time, and necessarily has to evaluate, besides the velocity direction, the speed value to calculate an interception point where the robot can get before the ball. The calculation of that interception point can be made by the two implemented algorithms presented in subsections 7.2.1 and 7.2.2. The algorithms of the interception point assume a constant robot speed. Considering that unknown coefficients of friction, slidings on the wheels, and other external factors affect the robot speed, the constant usually assumed is a bit conservative.

7.2.1 Uniform movement equations approach

A first approach on the problem was idealised and implemented in the *Worldstate* method *ballInterceptPoint*. Based on the assumption of uniform movement of both the ball and the robot, an equation system was developed to calculate the interception point

$$\begin{cases} \vec{B} + \vec{V}_B\ t & = & \vec{R} + \vec{V}_R\ t \\ |\vec{V}_R|^2 & = & C^2 \end{cases}$$

where:

- $\vec{B}$ is the ball position vector on the current cycle
- $\vec{V}_B$ is the ball velocity vector on the current cycle
- $\vec{R}$ is the robot position vector on the current cycle
- $\vec{V}_R$ is an assumed robot velocity vector
- t is the time instant when the positions of ball and the robot will be the same
- C is a constant we assume to be the mean speed the robot can achieve

We expand the system as:

$$\begin{cases} \vec{V}_R & = & \frac{\vec{B}-\vec{R}}{t} + \vec{V}_B \\ |\vec{V}_R|^2 & = & C^2 \end{cases} . \tag{7.1}$$

As we have both position and velocity vectors represented in their two components *(x,y)*, we substitute the first equation on the second and decompose into the equation:

$$\begin{aligned} &\Big(\frac{B_x - R_x}{t} + V_{B_x}\Big)^2 + \Big(\frac{B_y - R_y}{t} + V_{B_y}\Big)^2 = C^2 \\ \Leftrightarrow &(B_x - R_x)^2 + 2\ V_{B_x}\ t(B_x - R_x) + V_{B_x}^2\ t^2 + \\ &(B_y - R_y)^2 + 2\ V_{B_y}\ t(B_y - R_y) + V_{B_y}^2\ t^2 = C^2 t^2 \end{aligned}$$

and we obtain the quadratic equation:

$$\Rightarrow (|\vec{V}_B|^2 - C^2)t^2 + (2\ V_{B_x}(B_x - R_x) + 2\ V_{B_y}(B_y - R_y))t + |\vec{B} - \vec{R}|^2 = 0 \tag{7.2}$$

from which, through the quadratic formula

$$t = \frac{-b\pm\sqrt{b^2-4\ a\ c}}{2\ a} \quad \text{where from 7.2} \quad \begin{array}{l} a = |\vec{V_B}|^2 - C^2 \\ b = 2(V_{B_x}(B_x - R_x) + V_{B_y}(B_y - R_y)) \\ c = |\vec{B} - \vec{R}|^2 \end{array}$$

we can obtain a time of occurrence of an interception. As there are two solutions, it is necessary to interpret and choose which one to use. Therefore, the minimum positive time is the one we want, as a negative time does not make any sense, and in the case two positives occur, we want the one that correspond to the first moment of interception. With this time, and applying the law of uniform motion on the first equation of the system 7.1, we can obtain a minimum robot velocity $\vec{V_{min}}$for the interception to occur, with both *(x,y)* components. Having both the velocity and the time to apply it, one can get the interception point by

$$\overrightarrow{target} = \vec{R} + t\ \vec{V}_{min}.$$

Approach discussion

Although the approach is theoretically correct, it was experimentally verified that its results were far from the expectations. Too often, there was no real solution to the quadratic equation and thus no interception point could be calculated, *nan* values were returned. These *nan* values are originated by the fact that with the noisy measurements and ball velocity, the assumptions made on the model are not always verified and thus this strictly mathematical approach fails. Or, in some cases, the estimated velocity of the ball is too high for the system to calculate a solution for the assumed underestimated robot speed.

7.2.2 Iterative approach

Given the poor results obtained by the first approach, a new one was thought and implemented in the *Worldstate* method *ballInterceptPointIterative.* The idea is to iteratively go through the ball path line checking some positions for the time taken by both the ball and the robot until the robot takes less time than the ball, being that the point of interception.

Being an iterative process, it was built to try to reduce the running time to a minimum. A step $\overrightarrow{stepBallVelocity}$ (based on the ball velocity, hence the name) for advancing through the ball path is defined (figure 7.5), as well as a maximum length (currently 0.2 and 20 metres respectively). In each cycle, we have a static vector for the ball velocity, in *m/s*, so we can easily know the time taken by the ball to advance $|\overrightarrow{stepBallVelocity}|$ metres by dividing

$$stepBallTime = \frac{|\vec{V}_{step}|}{|\vec{V}_{ball}|}.$$

Thus each $|\overrightarrow{stepBallVelocity}|$ metres takes the ball *stepBallTime* seconds to cover. Considering the initial $\overrightarrow{target}$ as the ball position and the time for the ball to achieve it $ballTime = 0$, in each iteration they are incremented

$$\begin{aligned} \overrightarrow{target} &= \overrightarrow{target} + \overrightarrow{stepBallVelocity} \\ ballTime &= ballTime + stepBallTime. \end{aligned}$$

Given the iteration target, we can calculate the time taken by the robot to get there, by

$$robotTime = \frac{|\overrightarrow{target} - \overrightarrow{robotPosition}|}{C}$$

where C is a constant we assume to be the mean speed the robot can achieve. When

$$robotTime < ballTime$$

the interception point has been found. If this condition is not verified in any iteration, the final target is considered to be the interception point.

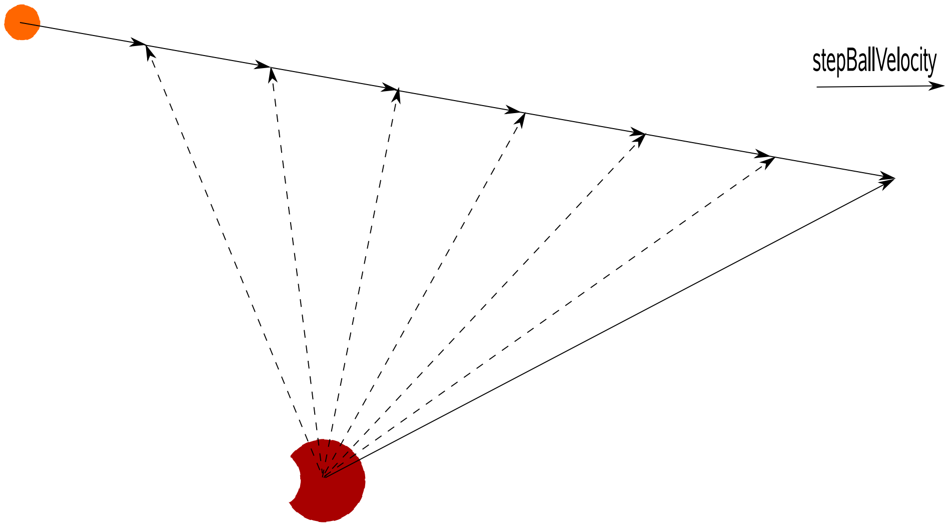


Figure 7.5: Illustration of an interception point calculation through iterative approach. At each iteration, *stepBallVelocity* is added to the ball path and the robot movement to the obtained point is tested concerning time to get there. In the illustration, the 7th iteration position is the chosen target, meaning that the time for the robot to get there would be less than the time taken by the ball.

Approach discussion

This approach proved to be much more reliable than the first one, as an actual interception point could be calculated, even if it was the maximum length point. Also, the efforts to minimise the operations proved to be effective as the algorithm executes very fast.

7.2.3 Interception point correction

As stated before, the algorithms of the interception point assume a constant robot speed that is usually a bit conservative. For that reason, if an interception point can be calculated, it is tested if it is inside the field. If it is, then the target calculated is the one the robot must head for. If not, the robot changes the target to the point over the field line by which the ball would go out. The target over the line is calculated by creating a line from the ball to the interception point and intersecting that line with the field line (figure 7.6). This is done because it makes no sense to make an interception outside the field and, as the robot can probably go faster than the assumed speed, maybe it can get to the line before the ball and intercept it. Also, even if the robot can't reach the line before the ball, the movement for the line becomes almost parallel to the ball path, which creates a good covering path in

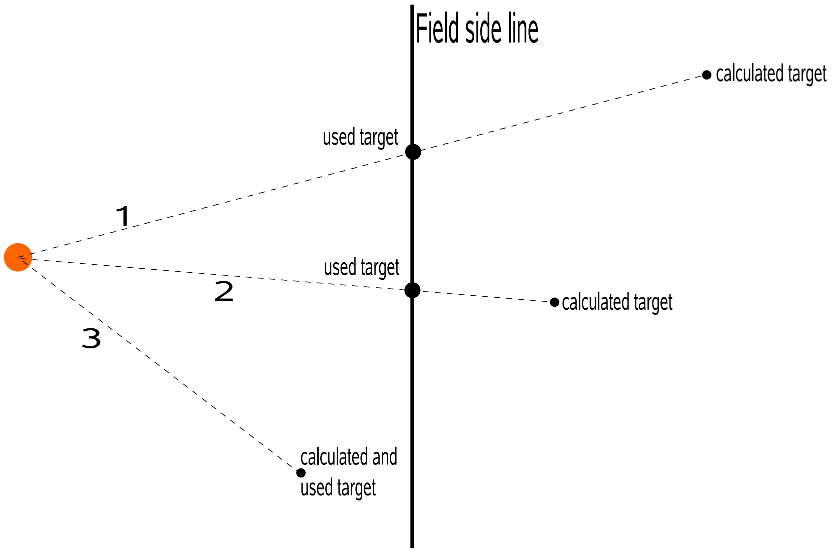


Figure 7.6: Illustration of interception point redefinition when the originally calculated is outside the field. In situations 1 and 2, the target is redefined for the point over the field line. In situation 3, the calculated target is the one used.

case an adversary would catch the ball. If an interception point cannot be calculated, the ball becomes the target.

7.2.4 Results

As the passive interception, this is a reactive behaviour, not a precision one. Given the need to get on the ball path at some point, the P component of the behaviour's translation PID controller is defined with value 2.0, to quickly correct the error as the passive interception. However, to try to avoid possible instabilities on the response, the D term is set to null (section 2.3). This is done because given the robot radius, it have about 20cm of margin around the position (since the correction is to get the robot centre to the position). With this margin, it is not critical if the state overshoots and oscillates a bit, as long as it is within the margin. However, it would be more critical if the response gets unstable. The rotation controller is defined with a P component of 2.0 and a D of 2.0, and the output limit is highly set to 6.28rad/s. With such high rotation, the robot ultimately rotates to the desired orientation and then translates to the position. This allows for the robot to quickly correct big rotation differences so that the translation movement is then less altered by rotation as described in section 7.3.

Several captures of active interception situations were made, to test and validate it. Figure 7.7 is a plot of the ball and robot positions of one of these captures. The ball positions are represented by blue dots and the robot positions are represented by red dots. The robot and the ball were stopped when the agent process was started. After a few cycles the ball was thrown and the robot activated to make an active interception. The captured data allows to visualise the paths of both the robot and the ball. The capture was stopped when the robot catched the ball.

In the plot, the ball moves from top centre to bottom centre with a sightly curved path. The plot was obtained by a Matlab script built to show the evolution of interception captures. At each sample, a line representing the unitary velocity vector of the object is drawn, as well as a small green circle around the current sample. The calculated interception point is represented by the small black circle near (-0.1, 0.2). Given the variations in speed evaluation and the conservative assumed robot speed, the robot path shows that the robot goes ahead of the ball and then goes back to catch it. For plot interpretation, one must keep in mind that the dots represent the centre of both the ball and the robot; each of these objects have a radius, and thus, when the robot has the ball engaged, the dots are necessarily separated by around 25cm. This interception takes advantage of a well refined ball velocity estimation, and proved to be quite effective. The interception point was calculated by the iterative approach described in section 7.2.2, as it is the one with most usable results.

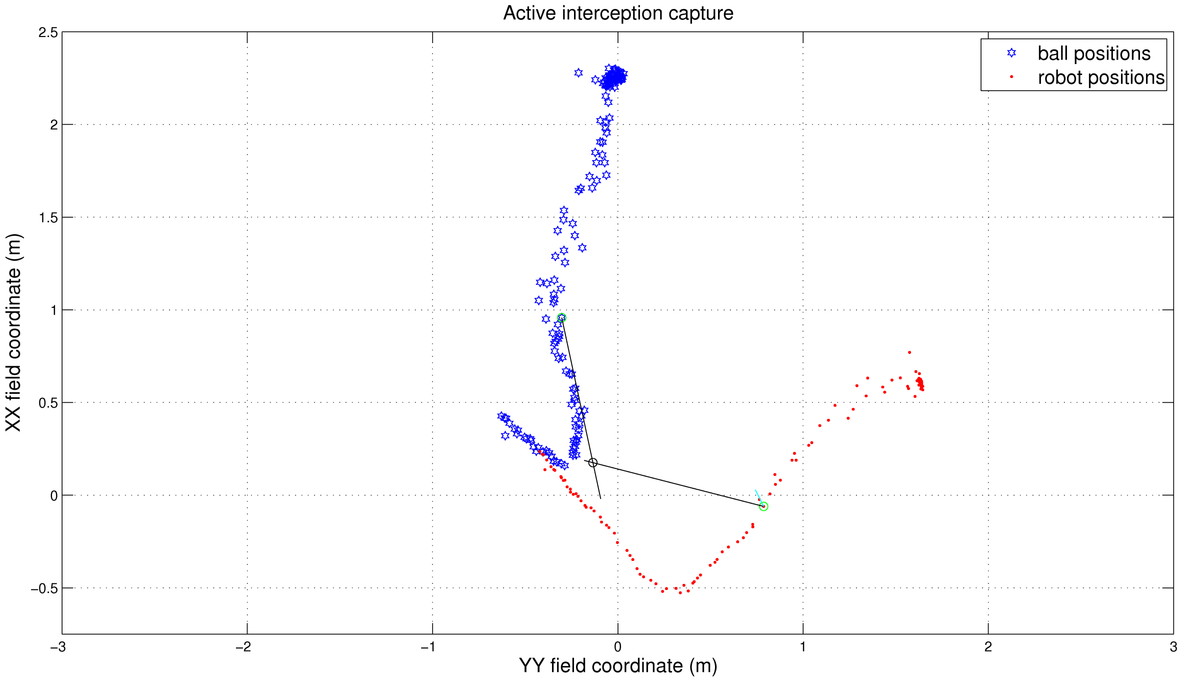


Figure 7.7: Plot of a capture of an active interception (the presented positions are estimated by the robot). See text for details.

7.3 Velocity direction correction on interceptions

Although the holonomic controller receives the velocity components on x, y and angular speed and calculates the values to apply to the motors for the robot to follow a straight line while rotating, it was verified that the angular speed affects the trajectory. Instead of the supposed straight line trajectory, the robot actually executes a curved trajectory as shown in figure 7.9 in subsection 7.3.1. This fact is probably the effect of sensing-actuating delays. Also, the velocity direction displacement increases with the increase of angular speed. For that reason, the interception behaviours try to compensate this angular displacement in order for the robot to move in the supposed straight line.

At each cycle, an angular speed is applied. When combining with a linear speed, the angular speed causes the object to move around a given point in a circle. Depending on the value of both the angular and linear speeds, the radius of the described circle varies. Having both linear and angular speeds we can calculate the displacement angle on this circular movement that the angular speed creates. According to angular velocity definition ([16, 45])

$$\omega = \frac{\Delta\theta}{\Delta t} = \frac{V}{r}$$
$$\Leftrightarrow r = \frac{V}{\omega}$$

where ω is the angular velocity, θ the angle displacement on the circle centre, V is the tangential velocity and r the radius of the circle. The tangential velocity is our linear velocity, since our desired straight line is tangent to the circle that the angular speed causes (figure 7.8).

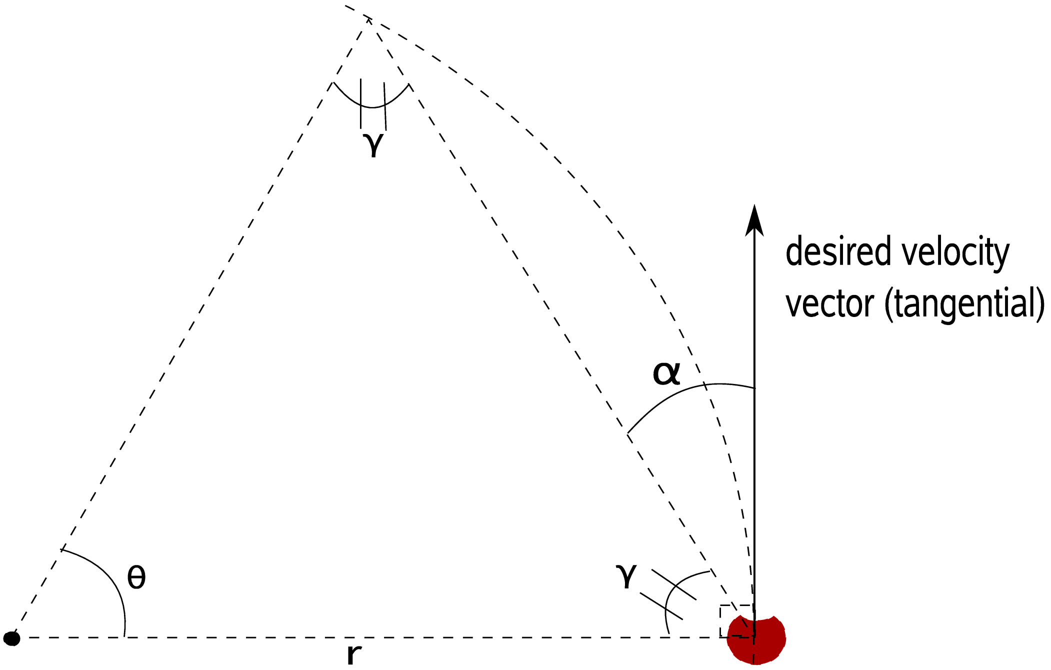


Figure 7.8: Illustration of the velocity correction approach. See text for details.

Having r, in order to measure the angle θ we approach the arc covered by the movement as $V\Delta t$, where Δt is the agent average cycle time, meaning it is considered that the calculated velocity is applied for Δt seconds. Thus we obtain

$$\theta = \frac{V\Delta t}{r}.$$

A triangle defining a slice of a circle has both outter angles with the same amplitude, thus we know that the angle γ is given by

$$\gamma = \frac{\pi - \theta}{2}$$

Knowing the tangential (desired) velocity direction, we approach the angle displacement caused by the angular speed as

$$\alpha = \frac{\pi}{2} - \gamma.$$

Finally, knowing the angle that affects the linear velocity, we can correct the desired (x,y) components by rotating them in a complementary way to the angular displacement α.

7.3.1 Results

The robot's resulting trajectory obtained when moving to a static ball is shown in figure 7.9.

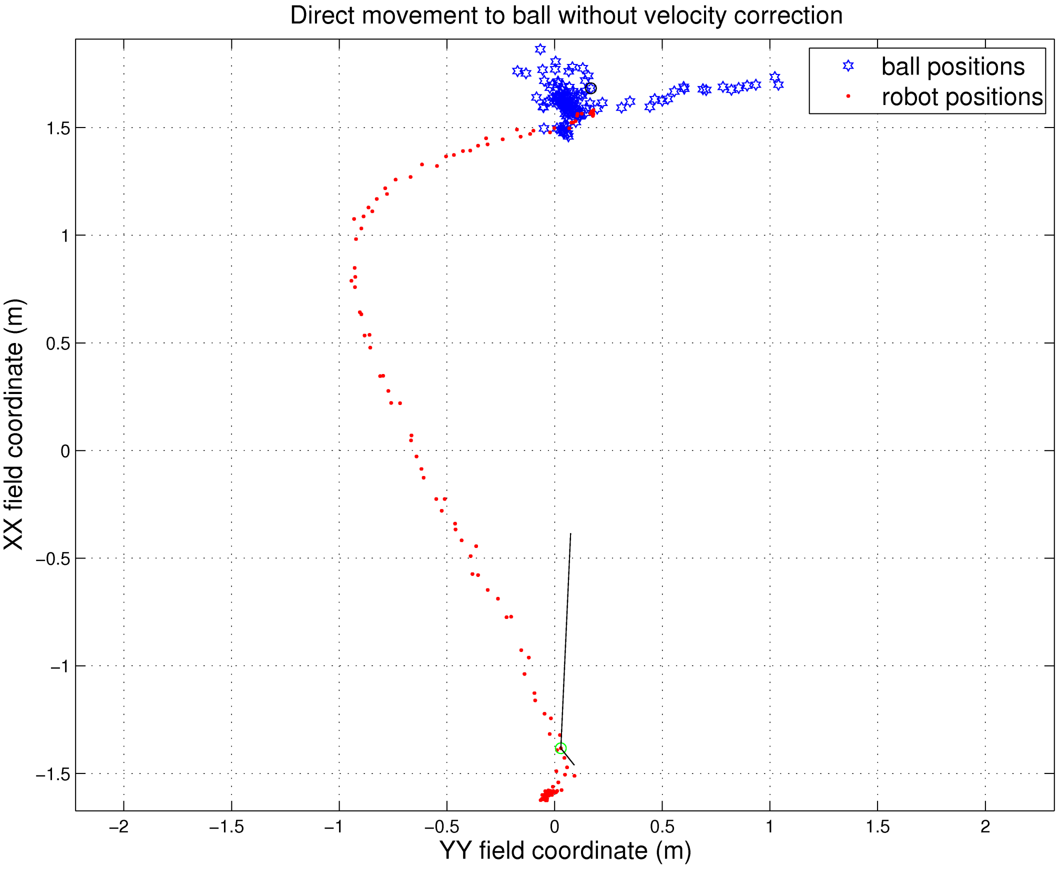


Figure 7.9: The plot shows a situation where the robot target was the static ball (the presented positions are estimated by the robot). Ball positions are the blue hexagrams, robot positions are the red dots. In the represented cycle, the robot desired velocity is the unitary black line. The small line on the robot position represents its orientation.

In the plot of figure 7.9, ball positions are the blue hexagrams and red dots are the robot positions. The plot was obtained by the same Matlab script built to show the evolution of interception captures, but the ball was defined as the target. At each sample, a black line representing the unitary velocity vector of the robot is drawn, as well as small green circle around the current sample. A small black line is also displayed in the robot's position, representing its orientation in the current sample. The shown plot corresponds to a cycle at the beginning of the movement, and the robot's orientation was about -135° from the ball position. The performed movement was supposed to be straight to the ball, while rotating to face it. It is clearly seen in the plot that although the calculated velocity is straight to the ball position, the trajectory described by the robot is heavily deviated while rotating.

To try to avoid this situation, the algorithm described above was applied to the originally calculated velocity vector. Several tries were made to find the complementary proportion of the α angle. First correcting only $-\alpha^\circ$ was tried, but poor results were obtained. The proportion factor was incremented to try to make the correction more effective. The chosen correction angle was defined as -10α, which was a correction that obtained more acceptable results. Figure 7.10 represents a situation similar to the one presented in figure 7.9, but with the robot initial orientation opposite to the first one, with a similar large angle in module. An additional green line is drawned in the robot position, representing the corrected velocity direction, used by the robot.

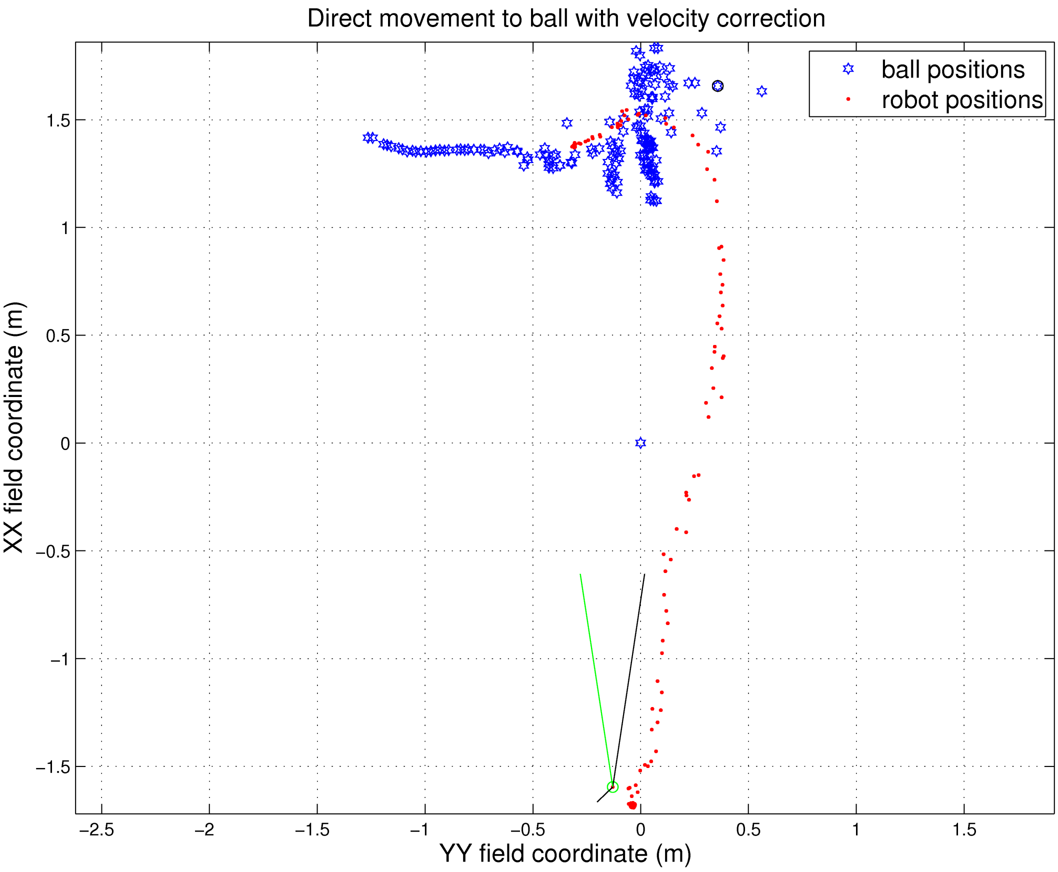


Figure 7.10: The plot shows a situation where the robot target was the static ball (the presented positions are estimated by the robot). Ball positions are the blue hexagrams, robot positions are the red dots. In the represented cycle, the robot desired velocity is the unitary black line and the corrected velocity is the unitary green line. The small line on the robot position represents its orientation.

One can easily verify in the plot that the described trajectory, although not exactly linear, is much less deviated than without the velocity correction. The obtained results improved the effectiveness of the movement, as the time to reach the target diminished with straighter paths.

7.4 Catchball (pass reception)

The utility of a pass in a soccer game is defined by the advantage position of the receiver of the pass, as it can be in a field zone without close opponents or with some other tactical advantage. However, if the robot cannot catch the ball effectively after a pass, it would loose much time to control it. That time would probably be enough for opponent robots to go near the receiver and thus, all the tactical advantage of having no opponents around would be lost. A pass set play was developed based on the definition of two agents, the one with the ball that passes it and the *Receiver*, responsible for receiving the pass. After the role attribution, the *Receiver* rotates to face the passer, the pass origin point. Since it is already facing the pass origin, the ball will be moving towards it. However, slight variations may occur, for several reasons, from noise to actual small deviations of the ball. For that reason the *Receiver* moves only in its XX axis, which is practically perpendicular to the ball path, to keep itself on the ball path. When the ball is within a given distance from it, a backward movement is made to soften the ball impact on the grabber zone and improve the catching. This backward movement speed is calculated as a given fraction of the ball speed, so that the robot would not move back too quickly, keeping the ball away or too slow, not softening the colision. The distance that activates the movement is also dependent on the ball speed.

7.4.1 Results

The PID controller for this behaviour was chosen to have a P component of 2.0 and D component of 0.5, as the passive interception, since the movement has to correct the XX position quickly and doesn't need to make an approach to the ball.

Figure 7.11 shows a capture of a *CatchBall* situation. When the capture started, the ball was taken to a position near (-0.1, 3.1) and then moved in the direction of field positive XX (YY axis on the plot). It was then thrown to the *Receiver*, which constantly made the necessary small corrections on XX axis. When the ball was closing in, it started the backward movement (always correcting XX) until the moment of the catch.

7.5 Ball avoidance

In some situations, the robot needs to avoid contact with the ball, for example, when positioning on the field for a free kick it should not touch the ball while going to the desired target position. An early implementation of this feature was based on the analysis of two distances, a minimum and a critical ball distance. If the robot was within the minimum distance, a correction of 45° to the path was made to the side closer to the target position. If the robot reached the critical distance, it would correct 90°. This approach however had two problems: 1) Since it is based on plain distances to the ball, when the robot speed is somehow increased, it is necessary to test for new distances that give enough time to react; 2) the correction angles were too "artificial", and in a worst case scenario the path to the

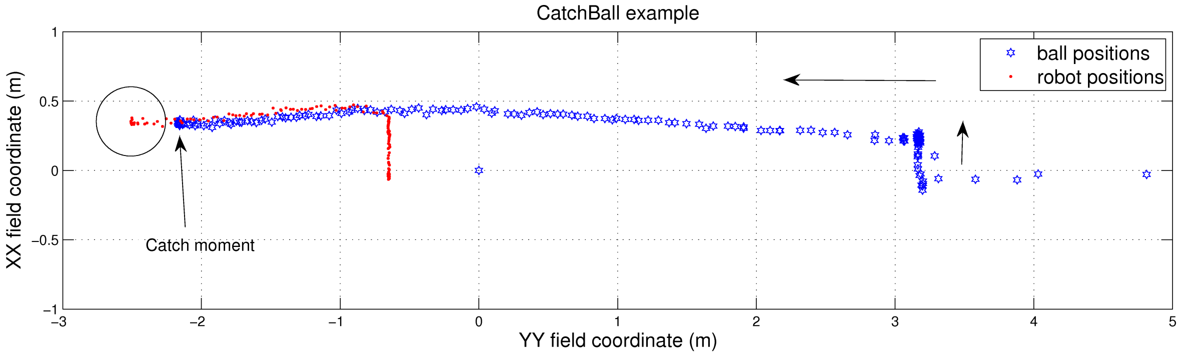


Figure 7.11: Illustration of ball catch. The robot radius is represented in the catch moment.

target could be altered by 90° while the robot was within the critical distance, loosing too much time, and it would happen even if the distance between the robot path and the ball was enough for it to pass by (figure 7.12).

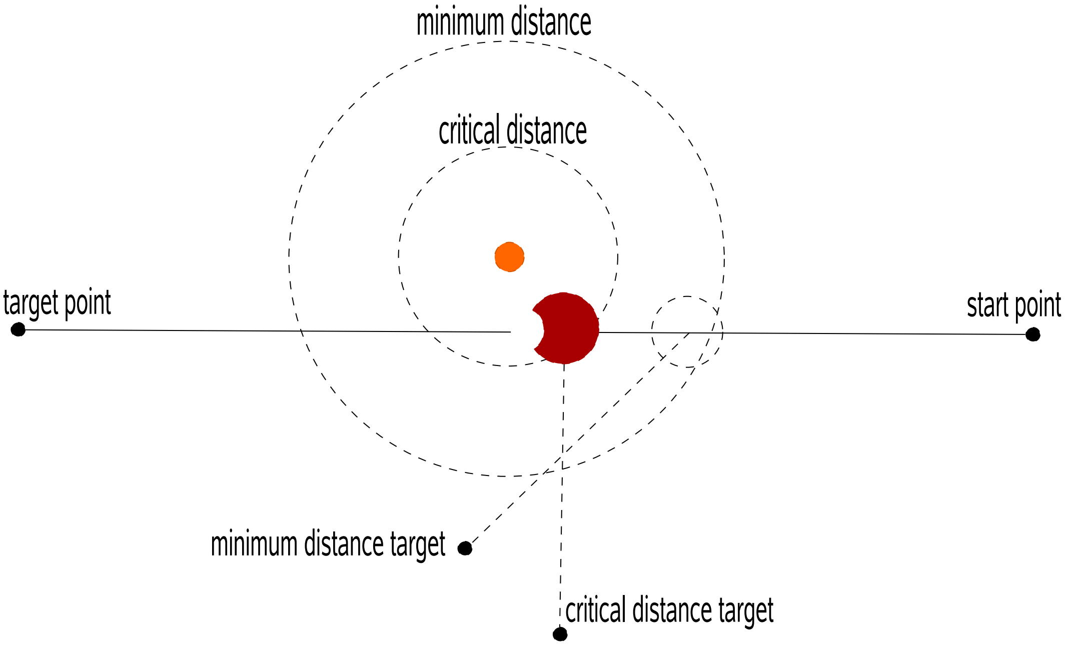


Figure 7.12: Illustration of the old approach to ball avoidance. In a situation similar to the illustrated, the robot could follow its original path without hitting the ball, and thus, no deviation would be necessary. However, the path would be altered by either 45° or 90° anyway, depending only on the distance to ball.

To try to make this movement more efficient a new approach was developed. If the ball avoidance is active, the movement always takes it into account, and the target position is corrected by some angle dependent on the need instead of fixed increments. Figure 7.13 represents the idea of this approach. Defining the vector $\vec{TB}$ between the target and the ball and the vector $\vec{TR}$ between the target and the robot, the angle θ between them is given through the scalar product definition

$$\begin{aligned}
\vec{TB}.\vec{TR} &= |\vec{TB}||\vec{TR}|\cos(\theta)\\
\theta &= \arccos\left(\frac{\vec{TB}.\vec{TR}}{|\vec{TB}||\vec{TR}|}\right)\\
&= \arccos\left(\frac{TB_xTR_x + TB_yTR_y}{|\vec{TB}||\vec{TR}|}\right).
\end{aligned}$$

By using the angle θ, we can define a right triangle where the opposite side is the distance between the ball and the robot path d and the hypotenuse is the vector $\vec{TB}$. Thus

$$\begin{aligned}
\sin(\theta) = \frac{opposite}{hypotenuse} &\Rightarrow \frac{d}{|\vec{TB}|}\\
\Leftrightarrow d = \sin(\theta)\ |\vec{TB}|
\end{aligned}$$

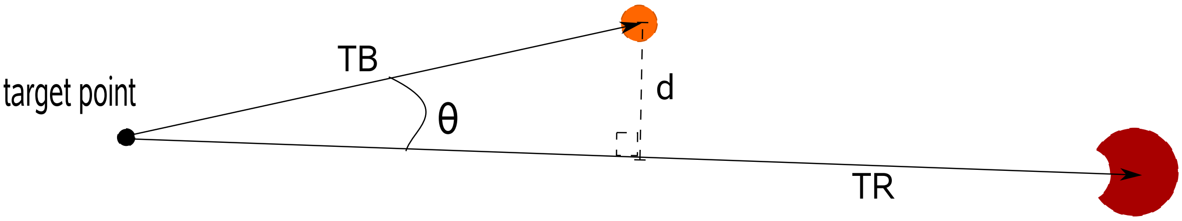


Figure 7.13: Illustration of the new approach to ball avoidance. See details on text.

Given a minimum threshold distance between the ball and the robot path, one can test if distance d is smaller than the threshold and correct the path by twice θ. This is only done when $\vec{TR}$ is larger than $\vec{TB}$, that is, the robot has not passed the ball yet.

7.6 Summary

This chapter presented the work accomplished at the behaviours level. The definition of the new interception behaviours, their ideas, implementations and results were presented. These behaviours were the first ones to make "predictive" movements, rather than direct movements to the ball. Difficulties found in the holonomic motion lead to the idealisation and implementation of an algorithm to correct the velocity direction applied to the robot when moving to a target, described and discussed in the chapter.
An early implementation of a behaviour to receive a pass was described, although it is much based on the high level pass set play assumptions. Also the ball avoidance algorithm was presented and the implementation of a new one described, that allowed for better ball avoidance, as it does not deviates the robot path much more than it is needs as the old one did.

Chapter 8

Conclusion and future work

8.1 Conclusion

While working in the CAMBADA team, its multidisciplinary characteristics become obvious. The need for a solid mechanical and hardware support is crucial for the robustness of the robot. At the software level, the vision plays a very important role, but also important is the way that the vision information is used. The creation of a world state that accurately represents the environment is a requirement, and it has been focus of part of the work presented in this thesis.

Reliable estimations of the ball position and velocity are an important aspect to develop, since high level decisions can and should be based on the way the ball is moving. It allows for “predictive” movements rather than reactive ones.

The chosen techniques for information and sensor fusion proved to be effective in accomplishing their objectives. The Kalman filter allows to filter the noise on the ball position and provides an important prediction feature which allows fast detection of deviations of the ball path. The linear regression used to estimate the velocity is also effective, and combined with the Kalman filter ability detect deviations, provides a faster way to recalculate the velocity in the new trajectory by keeping the ball movement history updated.

The integration of the new compass information due to the elimination of the goal colours was effectively done. With the existing localisation algorithm, only disambiguation between symmetric positions was necessary. Although it was previously done by visually identifying the goal colours, with the compass integration, the process was simplified in an effective way, even improving the performance (as the goals were not always visible) and reducing processed data by the vision image analysis.

Tools to manipulate and query situations in the worldstate improve the modularity of the code, as they provide the high level decision layer with a short and easy way to execute tests and manipulations that are recurrently needed.

At the behaviours level, the implementation of the interceptions and the approach point calculation are the first step for “predictive” features on the robots, as they allow for movement ahead of the ball, instead of going directly to it. However, it was only possible to make them effective due to the improvement of the ball velocity accuracy.

The work accomplished with passing and its in-game utilisation is an important achievement on the Robocup environment. The implementation of worldstate tools to evaluate possible pass corridors are important to the advance of this area. The implemented methods proved to have effective results, mainly the slice test (section 5.2.3).

However, the initial objective of organising the behaviour classes in a library was not accomplished. Although not being part of a library, the behaviours code is already quite modular, and therefore, other more functional tasks were prioritised over that objective.

Almost all the features implemented in this thesis scope were in use in the Portuguese Robotics Open Robótica 2008. The high level did not yet felt the necessity to integrate the active interception behaviour in its current structure, being this the only feature not used. After the Robótica 2008, and with the RoboCup2008 as objective, refinements and improvements have been and are still being made at the several areas.

On a general evaluation of the accomplished work, the overall objective of team performance improvement was achieved. The developed tools provide a base of work for new higher objectives that can now start to become the main ones.

8.2 Future work

Concerning sensor and information fusion, improvements on the detection of deviations are the next step. It could be done by creating a classifier able to identify patterns corresponding to deviations in the ball path, given a certain number of samples. This could be done by a simplified implementation of a neural network, for the sake of easy implementation and computation, in C language, since it can be basically seen as a weighted summation of the inputs. The cost of it would be the difficulty of the estimation of the network parameters.

Another sensor fusion issue that would certainly benefit the team would be the identification of the detected obstacles as team mates or opponents. This way only the opponents would be identified as “true” obstacles. This would allow, for instance, that two team mates could advance very close to each other, without trying to avoid each other, which would benefit cover and support coordination among team mates.

The development of worldstate tools is also a growing requirement, as new developments and approaches on the high level decisions will certainly need worldstate support.

Appendix A

Velocity estimation choice

Although in this case the Kalman filter internal functioning estimates a velocity, the obtained values were tested to confirm if the linear regression of the ball positions was still needed.

A test set was created, with a path containing a hard deviation, and the values including a know noise. The "ball" was theoretically moving in the field YY axis at a speed of 2.0 m/s and then was deviated, moving in the XX field axis at 1.0 m/s. This was done to be certain that the real speed of the ball was known, in order to verify the estimations.

In figures A.1 and A.2 are represented the ball positions and respective velocity vectors.

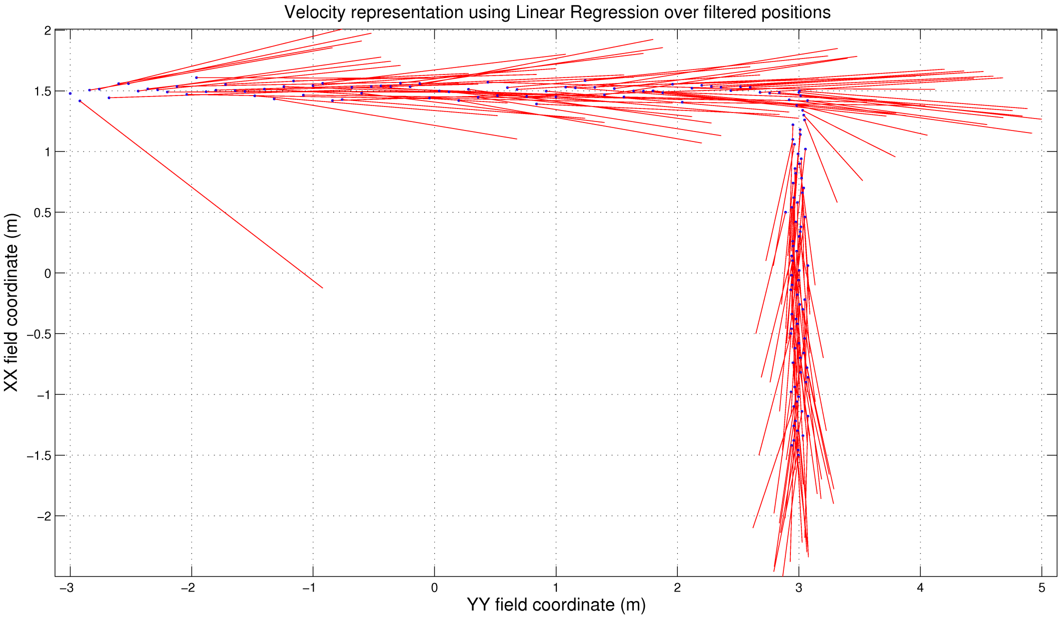


Figure A.1: Representation of the velocity of the ball in each cycle, estimated by the linear regression algorithm.

It is observable in the figures that, although the internal velocities estimated by the Kalman filter tend to have a more narrow angle around the true path, and thus a more stable path direction, its reaction in the deviation (and in the beginning of the capture) is slower than the linear regression estimated values.

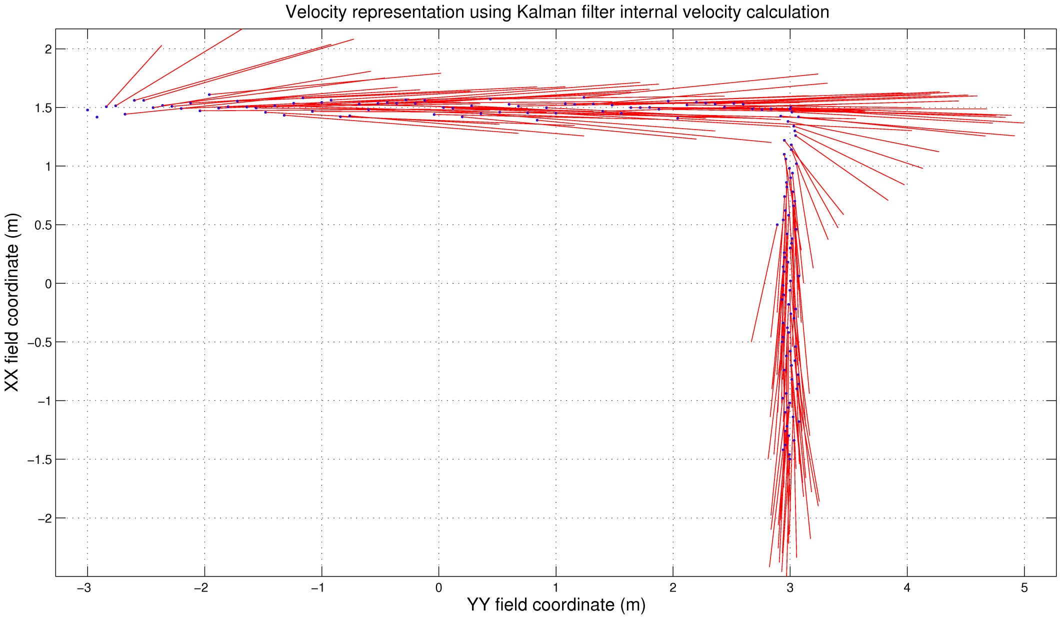


Figure A.2: Representation of the velocity of the ball in each cycle, estimated by the internal functioning of the Kalman filter.

In a similar way, it is verifiable in figure A.3 that the module of the velocity estimated internally by the Kalman filter takes much more cycles to become stable than the one estimated by the regression.

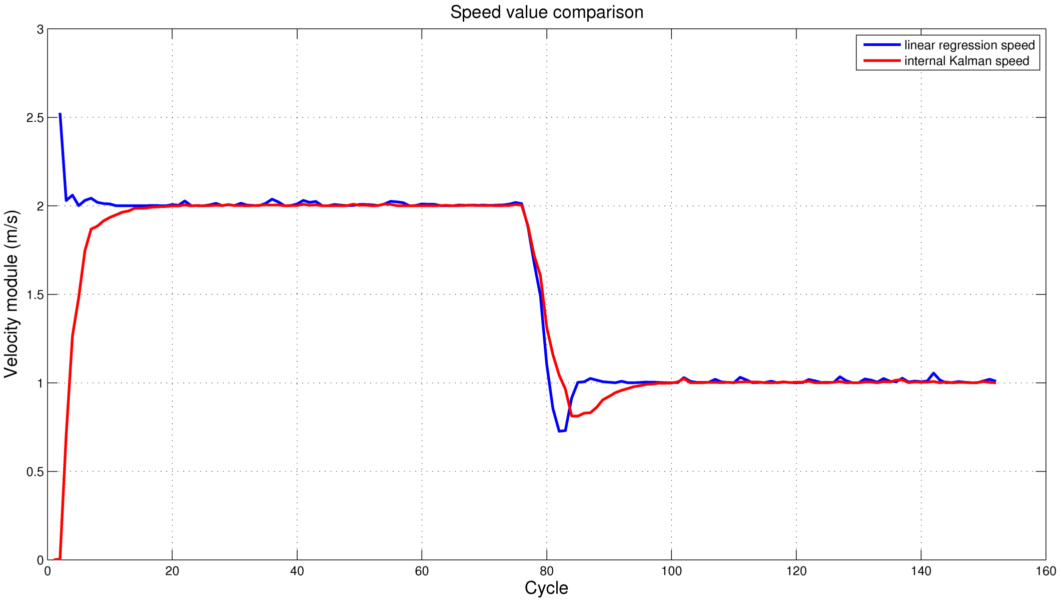


Figure A.3: Representation of the speed (velocity module) of the ball estimated by the regression algorithm (blue line) and internally by the Kalman filter (red line).

Although the overshoot of the regression estimate is slightly bigger, the choice still lies on the regression algorithm. The reason for that choice is that 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, is constantly changes its direction and thus a convergence in less than half the cycles as verifiable in the figure is much preferred.

Bibliography

[1] MSL Technical Committee 1997-2008. Middle Size Robot League Rules and Regulations for 2008, 2007.

[2] J. Wyatt. Jeremy Wyatt's Home Page, 2003. Lectures of the Intelligent Robotics course of University of Birmingham.

[3] G.A. Bekey. *Autonomous Robots: From Biological Inspiration to Implementation and Control*. The MIT Press, 2005.

[4] A. Melo. Sistemas de Controlo. Book used as reference for the Control System course of University of Aveiro, 2007.

[5] K.H. Ang, G. Chong, and Y. Li. PID control system analysis, design, and technology. *IEEE Transactions on Control Systems Technology*, 13(4):559–576, 2005.

[6] E. Cox. Fuzzy fundamentals. *Spectrum, IEEE*, 29(10):58–61, 1992.

[7] M.M. Gupta and N.K. Sinha. *Intelligent control systems: theory and applications*. IEEE Press, 1996.

[8] W.L. Brogan. *Modern control theory*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1991.

[9] B.V. Dasarathy. Information Fusion–what, where, why, when, and how? In *Information Fusion*, volume 2, pages 75–76. Elsevier, 2001.

[10] W. Elmenreich. *Sensor Fusion in Time-Triggered Systems*. PhD thesis, Technische Universitat Wien, Institut fur Technische Informatik, Vienna, Austria, 2002.

[11] R.E. Kalman. A New Approadh to Linear Filtering and Prediction Problems. *Journal of Basic Engineering*, 82(1):35–45, 1960.

[12] G.E. Dallal. Introduction to Simple Linear Regression, 2007. http://www.tufts.edu/~gdallal/slr.htm.

[13] H.J. Motulsky and A. Christopoulos. *Fitting models to biological data using linear and nonlinear regression*. GraphPad Software Inc., 2003.

[14] G. Bishop and G. Welch. An Introduction to the Kalman Filter. 2001.

[15] P.S. Maybeck. *Stochastic Models, Estimation, and Control*, volume 1, chapter 1. Academic Press, 1979.

[16] MathWorld: The Web's Most Extensive Mathematics Resource, 2008. http://mathworld.wolfram.com.

[17] I.M. Rekleitis. A Particle Filter Tutorial for Mobile Robot Localization. Technical Report TR-CIM-04-02, Centre for Intelligent Machines, McGill University, 2004.

[18] M. Mühlich. Scientific Download Page of Matthias Mühlich, 2003. http://user.uni-frankfurt.de/˜muehlich/sci/sciindex.html.

[19] R. Hafner, S. Lange, M. Lauer, and M. Riedmiller. Brainstormers Tribots Team Description, 2008.

[20] Y. Sato, S. Yamaguchi, Y. Kitazumi, Y. Ogawa, Y. Yonemura, T. Ueoka, Y. Wada, Y. Takemura, A.A.F. Nassiraei, I. Godler, K. Ishii, and H. Miyamoto. Hibikino-Musashi Team Description Paper, 2008.

[21] 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.

[22] H. Zhang, H. Lu, X. Wang, F. Sun, X. Ji, D. Hai, F. Liu, L. Cui, and Z. Zheng. NuBot Team Description Paper 2008, 2008.

[23] B. Kimiyaghalam, M.Y.A. Khanian, H.R. Fard, M. Montazeri, S. Moein, S. Morshedi, S. Ebrahimijam, H. Hosseini, and M.KH Hosseini. MRL Middle Size Team: 2008 Team Description Paper, 2008.

[24] J.J.M. Lunenburg and G. v.d. Ven. Tech United Team Description, 2008.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] R. Bosch. CAN Specification Version 2.0. Technical report, Stuttgart, Germany: Robert Bosch GmbH, 1991.

[32] L. Almeida, P. Pedreiras, and J.A.G. Fonseca. The FTT-CAN protocol: why and how. *IEEE Transactions on Industrial Electronics*, 49(6):1189–1201, 2002.

[33] 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 Symposium: Papers and Team Description Papers*. Springer, 2005.

[34] 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 *Proc. of the RoboCup 2007*, Atlanta, USA, 2007.

[35] A.J.R. Neves, G. Corrente, and A.J. Pinho. An omnidirectional vision system for soccer robots. In *Proc. of the EPIA 2007*, volume 4874 of *Lecture Notes in Artificial Inteligence*, pages 499–507. Springer, 2007.

[36] D. Martins. Image processing system for robotic applications. Master's thesis, University of Aveiro, 2008.

[37] A.J.R. Neves, D.A. Martins, and A.J. Pinho. A hybrid vision system for soccer robots using radial search lines. In *Proc. of the 8th Conference on Autonomous Robot Systems and Competitions, Portuguese Robotics Open - ROBOTICA'2008*, pages 51–55, Aveiro, Portugal, April 2008.

[38] W. Chen, Q. Cao, J. Wang, and C. Leng. JiaoLong2008 Team Description, 2008.

[39] B. Bouchard, D. Lapensée, M. Lauzon, S. Pelletier-Thibault, J-C. Roy, and G. Scott. Robofoot ÉPM Team Description Paper 2008, 2008.

[40] Y. Momozawa, K. Miyoshi, Y. Komoriya, R. Hashimoto, Y. Nakagawa, T. Adachi, M. Yamamoto, Y. Okuda, Y. Asano, and K. Demura. WinKIT 2008 Team Description, 2008.

[41] M. Lauer, S. Lange, and M. Riedmiller. Calculating the perfect match: an efficient and accurate approach for robot self-localization. In *RoboCup Symposium*. Springer, 2005.

[42] M. Lauer, S. Lange, and M. Riedmiller. *Modeling Moving Objects in a Dynamically Changing Robot Application*, pages 291–303. Springer, 2005.

[43] R.C. Arkin. *Behavior-Based Robotics*. MIT Press, 1998.

[44] E. Bicho. *Dynamic approach to behavior-based robotics: design, specification, analysis, simulation and implementation*. PhD thesis, University of Minho, 2000.

[45] Connexions - Sharing Knowledge and Building Communities, 2008. http://cnx.org.