Universidade de Aveiro
Departamento de Electrónica, Telecomunicações e Informática,
2009

João Alexandre da Silva Costa e Cunha

Controlo holonómico e comportamentos na equipa CAMBADA

Holonomic control and behaviours for the CAMBADA Robotic Soccer Team

Universidade de Aveiro
Departamento de Electrónica, Telecomunicações e Informática,
2009

João Alexandre da Silva Costa e Cunha

Controlo holonómico e comportamentos na equipa CAMBADA

Holonomic control 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 Tomás António Mendes Oliveira e Silva
professor associado da Universidade de Aveiro

Doutor Agostinho Gil Teixeira Lopes
investigador auxiliar da Universidade do Minho

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

À minha família, em especial aos meus pais, Manuel e Estefânia, e ao meu irmão, agradeço o apoio incondicional ao longo destes anos. Aos professores Nuno Lau e João Rodrigues agradeço a orientação e a disponibilidade quando dúvidas surgiam. Agradeço a todos os elementos da equipa CAMBADA pela ajuda e ambiente criativo fornecido. Agradeço em especial ao Nuno Figueiredo a ajuda para que o previsão da posição da bola estivesse completa a tempo do Robótica'09 e ao Gustavo Corrente pela ajuda em adaptar a solução dos Tribots. Por fim agradeço a todos os meus amigos, pela amizade e especialmente a compreensão pelo meu desaparecimento nestes últimos meses. Espero ver-vos em breve...

Abstract

CAMBADA is the RoboCup Middle Size League robotic soccer team created by researchers of the ATRI group of IEETA of University of Aveiro. This thesis presents the developed contributions in holonomic motion control and high-level behaviours.

At the motion control level, several restrictions affecting holonomic motion were addressed. This is of vital importance since that given the highly dynamic environment of a soccer game it is crucial to move and position the robots efficiently in the field.

At the behaviours level, given the importance of the ball in the soccer game, and considering previous work regarding the estimation of the ball velocity, the active interception behaviour was implemented allowing the robots to engage the ball predicting its movement rather than moving directly to it considering it static.

Given the full autonomy of the robots, their perception of the game should be as close to reality as possible. In order to supply additional information to the robot regarding the state of the game, a method to detect if it is stuck was developed.

The implemented work improved the team performance and contributed to the victory of a National Championship (Robótica'09) and a remarkable third-place in the RoboCup 2009 in Graz, Austria.

Resumo

CAMBADA é a equipa de futebol robótico da liga de tamanho médio do RoboCup criada por investigadores do grupo ATRI do IEETA da Universidade de Aveiro. Esta dissertação apresenta as contribuições desenvolvidas no controlo do movimento holonómico e comportamentos de alto-nível.

Ao nível do controlo do movimento, várias restrições que afectam o movimento holonómico foram tratadas. Isto é de vital importância visto que dado o ambiente altamente dinâmico de um jogo de futebol é crucial mover e posicionar os robots eficazmente no campo.

Ao nivel dos comportamentos, dada a importância da bola no jogo de futebol, e considerando o trabalho realizado no que diz respeito à estimativa da velocidade da bola, o comportamento de intercepção activa foi desenvolvido permitindo aos robots apanhar a bola prevendo o seu movimento em vez de se moverem directamente para a ela considerando-a estática.

Dada a autonomia completa dos robots, a sua percepção do mundo deve ser o mais próxima possivel da realidade. De modo a fornecer, ao robot, informação adicional respeitante ao estado do jogo, um método para determinar se está preso foi desenvolvido.

O trabalho implementado melhorou a performance da equipa e contribuiu para vitória de um campeonato nacional, Robótica'09, e a um notável terceiro lugar no RoboCup'09 em Graz, Áustria.

Contents

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Objectives . . . 1
- 1.3 RoboCup . . . 2
  - 1.3.1 Medium Size League (MSL) . . . 3
- 1.4 Thesis Structure . . . 4

2 Robot Locomotion **5**
- 2.1 Wheeled Robots . . . 5
  - 2.1.1 Wheels . . . 5
  - 2.1.2 Typical Wheel Configurations . . . 6
  - 2.1.3 Inverse Kinematics . . . 7
- 2.2 Biological Inspiration . . . 8
  - 2.2.1 Legged Robots . . . 8
  - 2.2.2 Other Biologically Inspired Types of Locomotion . . . 9
- 2.3 Common Wheel Configurations in MSL . . . 12
- 2.4 Summary . . . 13

3 The CAMBADA Team **15**
- 3.1 CAMBADA Robots . . . 15
- 3.2 Architecture . . . 16
  - 3.2.1 Hardware . . . 16
  - 3.2.2 Software . . . 18
- 3.3 Robot Motion Control Loop . . . 21
- 3.4 Summary . . . 22

4 Holonomic Control **23**
- 4.1 Restrictions . . . 23
  - 4.1.1 Discrete Time Control . . . 24
  - 4.1.2 Acceleration . . . 25
  - 4.1.3 Actuator Saturation . . . 26
  - 4.1.4 Control Latency . . . 27
- 4.2 Predictive Control . . . 27

4.2.1 Robot Model . . . 29
4.2.2 Ball Model . . . 29
4.3 New Acceleration Filter . . . 32
4.4 Estimating Model Parameters . . . 34
4.5 Results . . . 35
4.6 Summary . . . 36

5 Stuck Detection **39**
5.1 Known Methods for Stuck Detection . . . 39
5.1.1 Haptic Sensors . . . 39
5.1.2 Bumper Sensors . . . 40
5.1.3 Stasis Sensors . . . 40
5.1.4 Accelerometer Sensor . . . 40
5.1.5 Image Processing . . . 41
5.2 CAMBADA Implementation . . . 41
5.2.1 Results . . . 43
5.3 Summary . . . 43

6 Behaviours **45**
6.1 Active Interception . . . 46
6.1.1 Corrections to the Interception Point . . . 49
6.1.2 Role Assignment . . . 50
6.2 Intercepting the ball near the border lines . . . 51
6.3 Summary . . . 54

7 Conclusion and Future Work **55**
7.1 Conclusion . . . 55
7.2 Future Work . . . 56

List of Figures

1.1 A Robótica'2009 game. 3
1.2 A Middle Size League field with official markings. 4

2.1 Example of orientable wheels. 6
2.2 Example of swedish wheels 6
2.3 Three examples of commercially available humanoid robots 9
2.4 Example of a crawling robot. 10
2.5 Example of a realist fish robot. 11
2.6 Example of an underwater biologically inspired robot, RoboLobster [1]. . . 11
2.7 Example of a flying robot. 12

3.1 A CAMBADA robot. 15
3.2 Different robots shapes in MSL. 16
3.3 The CAMBADA top layer. 17
3.4 The middle layer. 18
3.5 The lower layer. 19
3.6 The micro-controller network. 20
3.7 The software architecture 21
3.8 Flow-chart of the robot motion control. 22

4.1 Deviation in the robot movement. 24
4.2 Discrete time control effect. 25
4.3 Acceleration limiter. 26
4.4 Effect of control latency over cycle time. 27
4.5 Predicting the robot position at the end of delay considering previous commands. 28
4.6 Changes in the information flow in the robot motion control scope. 28
4.7 The considered robot dimensions assuming the ball as a particle. 30
4.8 The reflection of the ball velocity when it collides with the robot. 31
4.9 The method for faster ball velocity convergence after a collision. 32
4.10 Robot path with different values of maximum acceleration. 33
4.11 New implementation of the acceleration limiter. 34
4.12 The robot path using new acceleration limiter and robot pose prediction after delay. 35

4.13 The resulting robot path with constant angular velocity. 36
4.14 Predicted ball positions after control delay. 37

5.1 Diagram of the discrepancy between the desired and estimated velocities. . 42
5.2 Evolution of desired and estimated robot speeds when stuck. 44

6.1 Class diagram of all CAMBADA behaviours. 45
6.2 Described path when the robot moves to the estimated ball position 46
6.3 Geometric representation of an interception. 48
6.4 The captured position during an interception. 49
6.5 Corrections to the interception point. 50
6.6 *isBallInDangerZone* verifications. 52
6.7 *getApproachPoint* implementation. 53

List of Tables

7.1 CAMBADA results in national competitions 56

Chapter 1

Introduction

1.1 Motivation

CAMBADA[1] is the University of Aveiro's Middle Size League robotic soccer team. The project started in 2003 within the IEETA's[2] ATRI[3] research group. It involves researchers from different scientific areas such as image analysis and processing, control, artificial intelligence, multi-agent coordination, sensor fusion, etc.

Since its creation, CAMBADA has participated in several competitions both national and international revealing an impressive progress which culminated in the 5th place in the World Championship RoboCup 2007, the 1st place in Robótica 2007, Robótica 2008 and Robótica 2009 becoming tri-National Champion, and the 1st place in the World Championship RoboCup 2008, in Suzhou, China.

In order to face new challenges and an ever-growing competition, the team aims for continuous improvement to maintain the level of achieved performance. With this in mind and considering previous years improvements in sensor information interpretation, the necessity for an efficient robot control arises to enable the development of predictive behaviours, specially in the robot motion control since part of the team's success relies in placing the robot in the desired position.

1.2 Objectives

It is the objective of this project to refine and revise the existent robots behaviours and implement new ones if necessary, as well as the implementation of sensor and information fusion techniques to build a better representation of the robot's worldstate. Holonomic control related issues should also be addressed to improve the robot's motion. The inno-

[1]CAMBADA is an acronym of Cooperative Autonomous Mobile roBots with Advanced Distributed Architecture

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

[3]Actividade Transversal em Robótica Inteligente - Transverse Activity in Intelligent Robotics

vations developed in this project should improve the robot's individual performance and the team's overall gameplay.

1.3 RoboCup

RoboCup is an international project focused on fostering innovation in robotics, artificial intelligence and related areas. Its long term goal is, by the year 2050, to develop a team of fully autonomous humanoid robots that can play and win against the human world champion soccer team [2]. To accomplish such goal RoboCup is divided in various competitions which can be divided in different leagues:

- RoboCup Junior
  - Dance League
  - Rescue League
  - Soccer League
- RoboCup Rescue
  - Robot League
  - Simulation League
    - Agent Competition
    - Infrastructure Competition
- RoboCup Soccer
  - Humanoid Leagues
    - Kid Size
    - Teen Size
  - Middle Size League
  - Mixed Reality
  - Simulation Leagues
    - 2D Simulation League
    - 3D Simulation League
    - 3D Development League
  - Small Size League
  - Standard Platform League
- RoboCup@Home

Each competition focuses on potential future applications of robotics such as search and rescue in catastrophic scenarios or daily activities at home. Given its singularities, the different leagues enable researchers to address problems in different areas of robotics such as hardware, machine vision or software.

1.3.1 Medium Size League (MSL)

CAMBADA robots were designed to compete in RoboCup's Middle Size League. In this league teams of 5 robots with maximum dimension of $50cm \times 50cm \times 80cm$ and a maximum weight of 40 Kg play in a $12m \times 18m$ depicted in 1.2 according to robot-adapted official FIFA rules [3].

Figure 1.1: A Robótica'2009 game.

Robots must be fully autonomous and have all sensors on-board. For cooperation purposes, robots can communicate with team-mates and the coach through a wireless connection. The coach, must be autonomous as well, and is normally in an external computer that must have no sensors, deciding accordingly with the information provided by the players. The sole human intervention is the one provided by the referee's decisions which enforces the game rules. The environment in which the robots play has well defined colours, such as the green field, the white lines, the orange ball, and the mainly black robots. However this will tend to disappear such as the already planned change in the 2010 edition, where the robots will play with a generic FIFA ball instead of the current orange coloured ball.

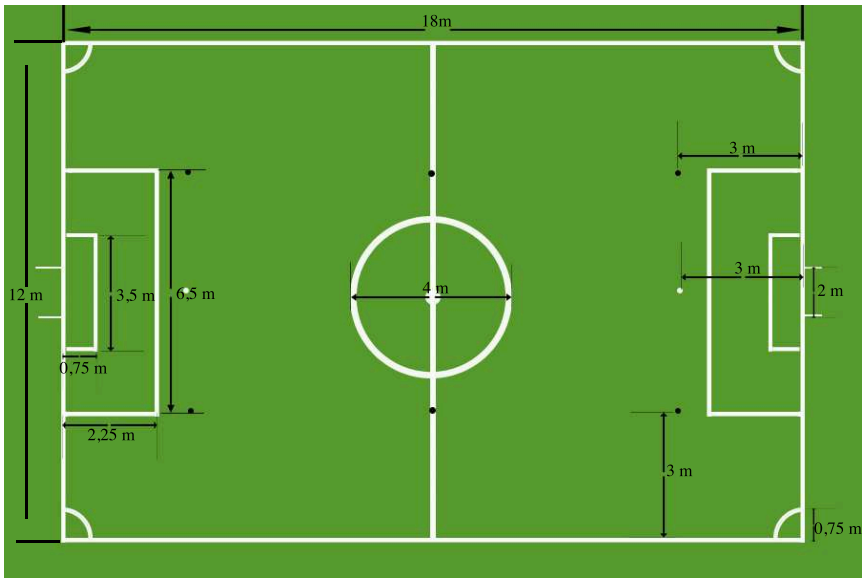


Figure 1.2: A Middle Size League field with official markings.

1.4 Thesis Structure

This thesis is structured as follows:

In chapter 2 an overview of the state of the art in mobile robots is presented with special focus for wheeled robots, in which CAMBADA robots are included, such as different wheels used and different wheel configurations.

Chapter 3 introduces the CAMBADA robots architecture and developed work at the beginning of this thesis work.

In chapter 4 the restrictions affecting holonomic motion control and the implementation of predictive control are discussed.

Chapter 5 presents methods for self-detection of stuck situations and the implementation used in the CAMBADA team.

Chapter 6 describes the contributions made at the behavioral level of the robot, such as refinement of existing behaviours and implementation of new ones.

Finally, chapter 7 discusses the results achieved and presents the conclusion and suggestions for future work.

Chapter 2

Robot Locomotion

This chapter presents an overview of the state of the art on robot locomotion. Special focus is given to wheeled robots, of which CAMBADA robots are an example. Biologically inspired locomotion methods are also discussed such as legged locomotion, underwater locomotion and flying robots.

2.1 Wheeled Robots

Since the dawn of robotics, with Walter's tortoise, robots move with wheels. Although not biologically inspired, these robots offer some advantages relative to other methods of locomotion. Wheeled robots do not suffer from static or dynamic stability problems since the robot center of gravity doesn't change when stopped neither in motion. From a control a point of view, wheeled robots are also simpler than other robots, since to achieve the desired movement power needs to be supplied to the wheels which is accomplished with feedback and PIDs. However, to move for instance legged robots, complex models must be used in order to obtain a forward motion while maintaining a vertical posture.

2.1.1 Wheels

In mobile robotics there are different type of wheels in use. The different combinations of these wheels allow for diverse types of motions. Depending on the wheeled robot purpose, a designer should choose wisely the adequate type of motion, since in most cases a simpler wheel combination should suffice.

The different types of wheels used in robotics are [4]:

Fixed wheel as the name implies is a wheel with its center fixed to the robot frame and the angle between the wheel plane and the robot frame is constant.

Centered orientable wheel differs from the fixed wheels since the wheel plane rotates around a vertical axle which passes through the center of the wheel. Therefore the angle between the robot frame and the wheel plane is time varying. A centered orientable wheel is in the center of Fig 2.1.

Off-centered orientable wheel or "castor wheel" is also orientable in respect to the robot frame. However the vertical axle of rotation does not pass through the center of the wheel. Several off-centered orientable are shown in the corners of Fig 2.1.

Swedish wheels are quite different from conventional wheels. In a conventional wheel, the velocity of the point of contact between the wheel and the ground is equal to zero. This is not true for a swedish wheel. The velocity of the point of contact between a swedish wheel and the ground has only one component that is equal to zero. Examples of swedish wheels are depicted in Fig 2.2.

Figure 2.1: Example of orientable wheels, adapted from [5].

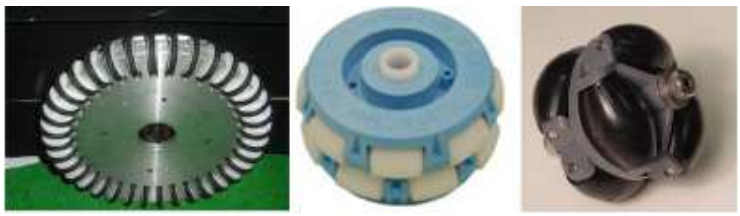


Figure 2.2: Example of swedish wheels, adapted from [6].

2.1.2 Typical Wheel Configurations

As mentioned, the combinations of the different wheels presented result in a variety of motions. The types of motion are usually characterized by the controlable degrees of freedom enabled by a specific combination of wheels. The most common types are [4]:

Omni-Directional Motion means a wheeled robot can move in any direction while facing any orientation. Therefore the robot does not to perform any reorientation of the wheels in order to move to any given direction. This is accomplished by combining three or more swedish wheels or three or more off-centered orientable wheels. Since these robots control all degrees of freedom they are considered holonomic.

Differential Motion is characterized by robots with two fixed wheels placed on a common axle but driven by separate motors. This kind of wheel configuration allows the robot to turn without having to steer any of its wheels. However, robots using differential motion are non-holonomic since they can't move sideways.

Ackerman steering [7] is widely used in automobiles. It combines one or more fixed wheels placed on a common axle and a separate axle with one or more centered orientable wheels for steering. The robot can move forward and backwards and turn sideways. When turning the robot describes an arc around an instantaneous center of rotation (ICR) which is orthogonal to all the wheel planes. If more than one steering wheel is used the orientation of the wheels must be coordinated to ensure a single ICR, otherwise the robot will suffer from slippage of the front wheels when turning. Robots using this type of motion are non-holonomic since the fixed wheels prevent the robots from moving sideways.

Synchro Drive is a configuration of at least two centered orientable wheels. This allows the robot to achieve holonomy however to perform a direction variation, the wheels must be reoriented. If there is no slippage the robot can only describe a curve with a center in the interception of the lines orthogonal to the wheel planes. Therefore the orientation of the wheels must be changed in order to achieve the other degrees of freedom.

2.1.3 Inverse Kinematics

To move a wheeled robot, velocities to all the driven wheels must be supplied. This is usually achieved by translating the desired velocities to be applied at the robot frame into the wheels setpoints and it is also known as inverse kinematics.

Inverse kinematic models can be deduced based on previously mentioned constraints [4]. For conventional wheels the assumed constraints are that the velocities along and orthogonal to the wheel planes must be zero. As aforecited in swedish wheels only one of the velocity components must be zero.

The constraints of each wheel composing the robot's wheel configuration can be written under the general matrix form [4]:

$$J_1(\beta_c, \beta_{oc}) \cdot R(\theta) \cdot \xi + J_2 \cdot \varphi = 0 \quad (2.1)$$

$$C_1(\beta_c, \beta_{oc}) \cdot R(\theta) \cdot \xi + C_2 \cdot \beta_{oc} = 0 \quad (2.2)$$

where $J_1(\beta_c, \beta_{oc})$ is a matrix composed by the constraints orthogonal to the wheel plane and J_2 is a diagonal matrix composed by the radius of each wheel. $C_1(\beta_c, \beta_{oc})$ is a matrix composed by the constraints along the wheel plane and $C_2\beta_{oc}$ is a matrix with the information of the distances between the center of the off-centered orientable wheels and the axle of rotation. β_c, β_{oc} are the orientation of the centered orientable wheels and off-centered orientable wheels, respectively. ξ are the desired velocities in world coordinates and φ are the velocities to be applied at each wheel. $R(\theta)$ is a rotation matrix to translate the desired velocities from world coordinates to velocities with respect to the robot frame, depending on the orientation of the robot, θ.

Given a set of desired velocities in world coordinates, ξ, the desired velocities to be applied at each wheel and the orientation of the centered and off-centered orientable wheels can be calculated by solving the equations for φ, β_c and β_{oc}, respectively.

2.2 Biological Inspiration

Although wheeled robots offer some advantages regarding other types of locomotion they suffer from a major drawback shared by other wheeled vehicles: they need roads or evens surfaces to move. However most scenarios where future robots are expected to be applied don't provide these conditions, for example: search and rescue situations where robots might have to crawl over rubble, domestic robots might have to climb stairs, planetary exploration where robots might have to move over different surfaces such as rocks and sand, aerial surveillance or robots required to perform tasks underwater.

For these situations robotic engineers take inspiration from animal locomotion that can be emulated in robotics. This lead to the development of walking robots with different number of legs, swimming, flying and crawling robots.

2.2.1 Legged Robots

Although a seemingly easy task, walking on legs is a fairly complex process. Animals need a learning period to do it properly. Hence, it is no surprise that no robot can walk with the same dexterity as animals, though this subject has been heavily researched in the past decades.

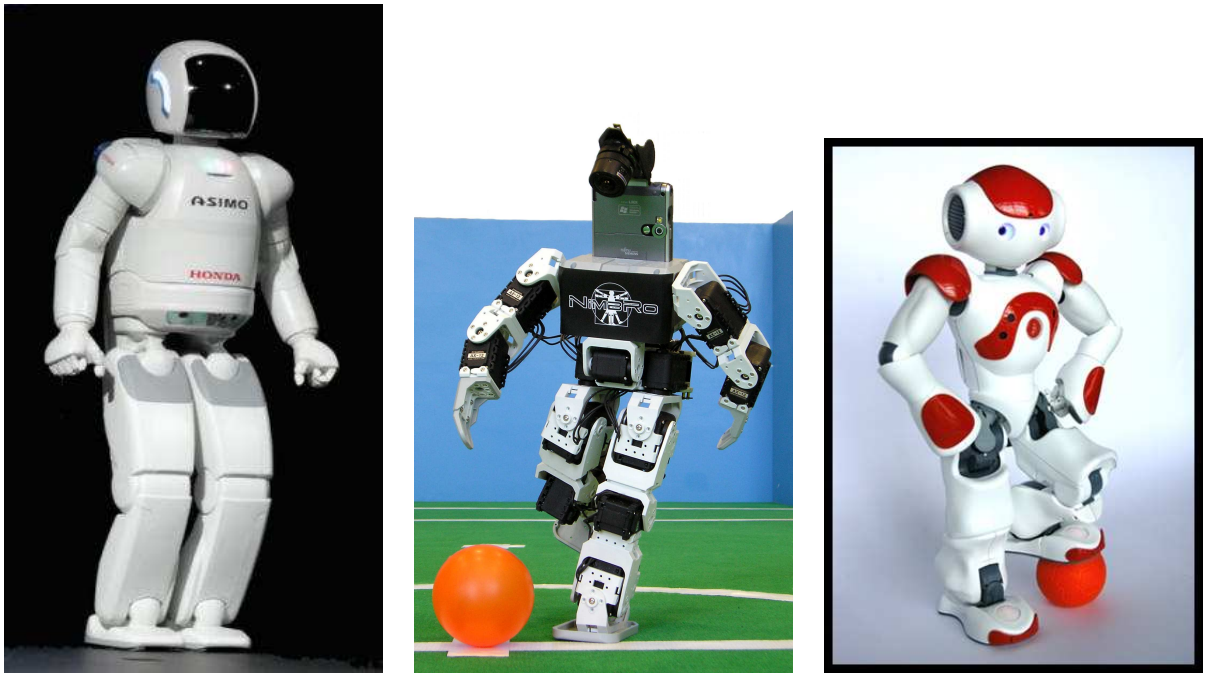
To effectively make a robot walk two issues must be considered: stability control to guarantee a vertical posture and motion control to enable a forward motion [7]. Legged robots must be able to stand vertically when stopped, also known as *static stability*, and while moving, *dynamic stability*.

Full static stability can be achieved with three legs, as long as the vertical projection of the center-of-gravity vector lies between the points of contact of the legs with the ground.

Static stability is obviously more difficult with two legs since it is only possible to guarantee lateral stability when the center-of-gravity vector lies between the feet. Backward-forward stability is ensured by active control systems or large feet, which increases the area in contact with the ground.

To obtain dynamic stability during motion, the support forces of the legs, momentum and inertial forces must be considered [7].

Three examples of two-legged humanoid robots are shown in Fig 2.3.



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

Figure 2.3: Three examples of commercially available humanoid robots. **a)** Honda's Asimo, adapted from [8]. **b)** Bioloid adaptation of Nimbro's humanoid, adapted from [9]. **c)** Aldebaran's NAO robot, adapted from [10].

2.2.2 Other Biologically Inspired Types of Locomotion

Animals developed other types of locomotion that robotic engineers have been trying to mimic with their robots.

One example are the snake-like robots that move their limbless bodies by crawling segments of the robot body as shown in Fig 2.4. In fact the segments that compose the robot body enable a large number of degrees of freedom which characterizes these robots as hyperredundant. Although the segments have limited rotation, the combined motion of the segments allow large movements of the body. Snake robots achieve forward motion by supplying internal torques to the segments, as snakes do with muscle contraction. Although the earlier snake robots were designed to move on two-dimensions, there have been developed robots the can move some body segments vertically adapting to ground variations.

Another type of locomotion inspiring robotic engineers is underwater locomotion, shown in Fig 2.5. Unlike ground robots, underwater robots must move in 3D environments. Wa-

Figure 2.4: Example of a crawling robot, adapted from [11].

ter also poses additional problems. For instance water density and currents, for instance, introduce non-linearities which hinder the robot control. Besides being waterproof, underwater robots must also be robust to sustain water pressure when diving great depths. The robot dynamics may also be affected if it is equipped with manipulators which may extend or contract and carry variable payload.

Some tasks performed by underwater robots such as ocean exploration or monitorization may require a robot to stay underwater for long periods of time, thus battery and power consumption are a crucial point when designing such robots, especially the type of locomotion which can be very consuming.

Many underwater robots are equipped with thrusters to move. However these actuators are not quite efficient in terms of power consumption and therefore robotic engineers are turning to fish for inspiration when designing underwater robots. Fish are exquisitely adapted to underwater environment and motion is produced by the water displacement resultant of fin movement, specially the tail fin movement.

Other underwater animals are the source of inspiration for robot designers such as lobsters, crabs and water snakes [7], as shown in Fig 2.6.

At last, robotic engineers have also been creating flying robots. These robots present similar control issues to underwater robots. They also have to move in a 3D environment and variations in atmospheric density at different altitudes render PIDs and linear control nearly impossible [7]. Flying robots must also overcome gravity and drag to remain airbourne.

Although most flying robots are inspired in airplanes and helicopters and use propellers to achieve motion, there have been some recent developments in bird-like robots. As other flying animals, these robots move by flapping their wings. These robots must be light

Figure 2.5: Example of a realist fish robot, adapted from [12].

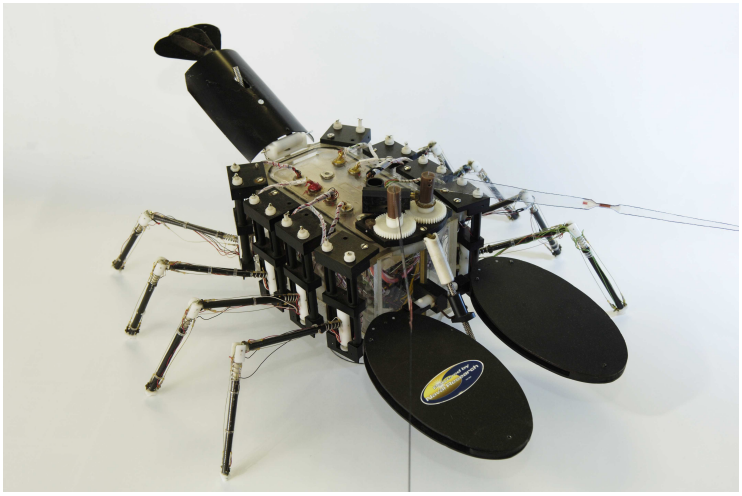


Figure 2.6: Example of an underwater biologically inspired robot, RoboLobster [1].

so that the ratio of wing area to body mass is large enough to allow flight [7]. However robots with these specifications suffer weather conditions such as wind gusts which makes full-autonomy quite difficult. An example of a biologically inspired flying robot is depicted in Fig 2.7.

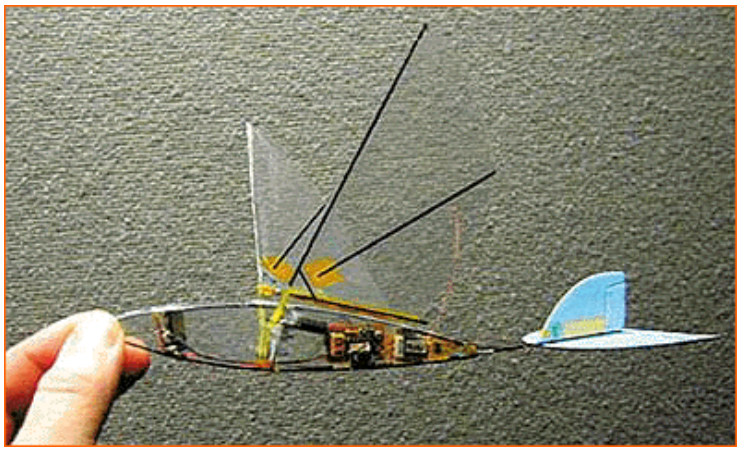


Figure 2.7: Example of a flying robot, adapted from [13].

2.3 Common Wheel Configurations in MSL

The RoboCup Middle Size League is composed entirely by wheeled robots. In this section, the locomotion types used by most teams are presented.

Given the nature of a soccer game, omni-directional motion offers several advantages specially being able to move sideways while facing the ball. This can make a pass reception easier than any other type of locomotion presented. Therefore it is only natural that almost all teams choose to use this type of locomotion.

All teams achieve omni-directional motion through the use of swedish wheels. As mentioned in 2.1.1, at least three swedish wheels must be used. In fact this is the configuration most teams use, including CAMBADA. However in recent years, a trend is arising where teams, like MRL [14], 1. RFC Stuttgart [15] and NuBot [16], use an additional swedish wheel. This configuration appears to take advantage of the fact that although omni-directional, the robots move forward most of the time during a soccer game. In this situation, using three wheels, the wheel in the back of the robot is passive and adds nothing to the forward movement. However using four swedish wheels, the number of active wheels doubles as all wheels contribute to the forward movement, considering that the configuration of the wheels doesn't have any swedish wheel orthogonal to the foward motion.

An exception to the use of omni-directional motion is the ISEP team that uses two differential driven fixed wheels and two castor wheels. The team chooses to use this type of motion since it is easier to apply to other land robots while aware of its inefficiency in the soccer field [17].

2.4 Summary

This chapter presented the state of the art on robot locomotion. Types of wheels and locomotion types used on wheeled robots were described. An overview of biologically inspired locomotions like crawling, swimming and flying were presented. Finally the wheel configurations used in the MSL and its advantages were discussed.

Chapter 3

The CAMBADA Team

The CAMBADA team is composed by six robots designed to specifically play soccer. The first version of the robots was completed in 2004 and since then there has been a steady evolution in the robots structure to face occurring challenges. This chapter describes the stage of the CAMBADA robots at the beginning of this thesis work.

3.1 CAMBADA Robots

Competing in the RoboCup's Middle Size League, the CAMBADA robots must not exceed the maximum dimensions of $50cm \times 50cm \times 80cm$. The rules however don't impose a particular shape leaving that decision to each team. CAMBADA robots have a conical shape with a base radius of $24cm$ and are $71cm$ high as can be seen in Fig 3.1.

Figure 3.1: A CAMBADA robot.

However, CAMBADA robots are the exception rather than the rule, as most of the MSL teams choose to build their robots on a quadrangular or triangular base [18][15][19][20][16][21][22][23][24], shown in Fig 3.2.

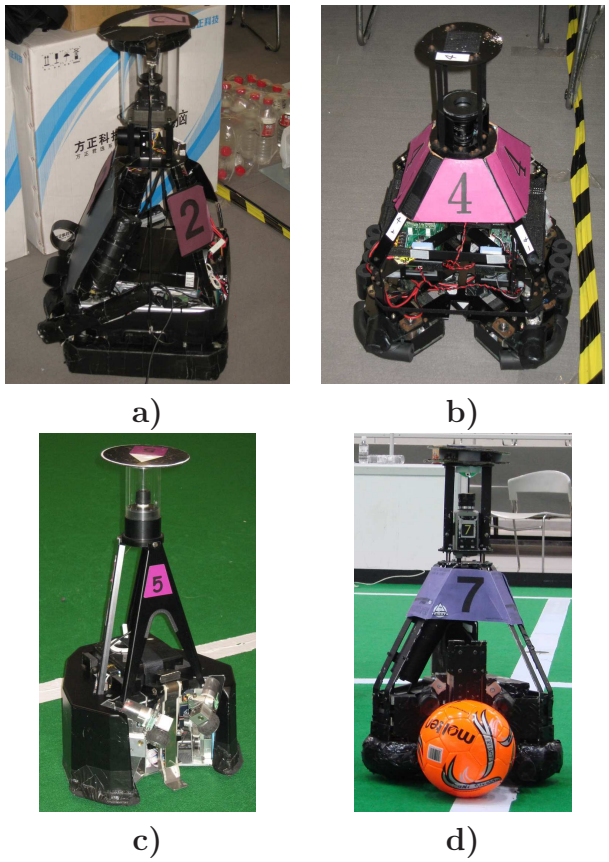


Figure 3.2: Different robots shapes in MSL. **a)** Eigen Robot. **b)** TKU Robot. **c)** TechUnited Robot. **d)** Tribots Robots.

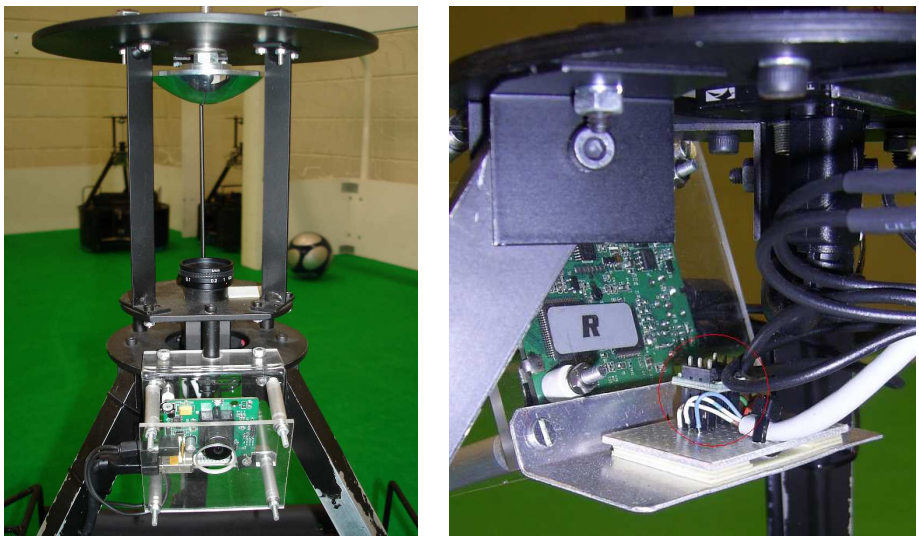
One of the CAMBADA robots, the goalkeeper, has metal appendices covering more area, while still complying with the maximum dimensions, that emulate the goalkeepers' arms. Some teams in the MSL choose to build mobile frames to improve the goalkeeper's performance. The CAMBADA goalkeeper frame is fixed to the robot.

3.2 Architecture

3.2.1 Hardware

The robot's physical structure is built on a modular approach [25] with four main modules or layers [26].

The top layer has the robot's vision system. This system is comprises a camera pointing upwards to an hyperbolic mirror allowing the robot to have a 360 degrees view and a frontal camera to allow the detection of the ball at greater distances [27]. Also in the top layer, an electronic compass is used for localization. For more detail on the vision sub-system please refer to [28][29]. The top layer is highlighted in Fig 3.3



a) **b)**

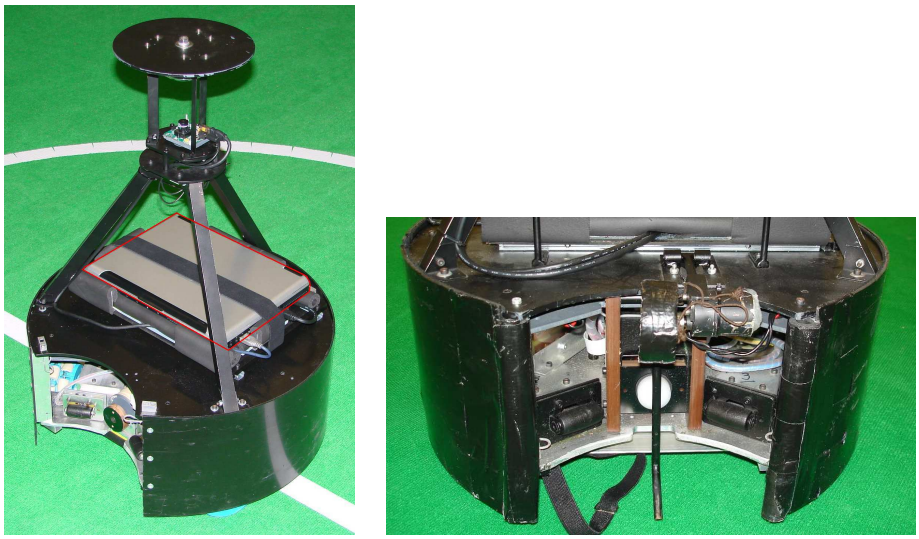
Figure 3.3: The CAMBADA top layer. **a**] The vision system. **b)** The electronic compass.

The middle layer, shown in Fig 3.4 houses the processing unit, currently a 12" laptop, which collects the data from the sensors and computes the commands provided to the actuators. The laptop executes the vision software along with all high level and decision software and can be seen as the brain of the robot. Given the positional advantage, a ball retention device is placed on this layer.

Finally, the lowest layer is composed by the robots motion system and kicking device. The robots move with the aid of a set of three omni-wheels, disposed at the periphery of the robot at angles that differ 120 degrees from each other, powered by 3 24 V / 150 W Maxon motors. On this layer there is an electromagnetic kicking device. Also, for ball handling purposes, a barrier sensor is installed underneath the robot's base, that signals the higher level that the ball is under control. The different components are shown in Fig 3.5

Beneath the middle layer, a network of micro-controllers is placed to control the low-level sensing/actuation system, or the nervous system of the robot. In order to comply with real-time constraints the FTT-CAN[1] protocol is used [30]. The sensing and actuation system is highly distributed, meaning that the nodes in the network control different functions of the robot, namely [31]:

[1]Flexible Time-Triggered communication over CAN



a) b)

Figure 3.4: The middle layer. **a)** The processing unit. **b)** The ball retention system.

Motion which controls the desired speed to be applied at each motor;

Odometry that calculates the displacement of the robot;

Kick that applies the desired power to the kicking device and the ball retention system and transmits the state of the barrier sensor;

Compass that relays the electronic compass data;

System Monitor that forwards the robot's batteries info as well as the state of all the nodes.

The different nodes in the low-level sensing/actuation system and its physical implementation can be seen in Fig 3.6

3.2.2 Software

Following the CAMBADA hardware approach, the software is also distributed. Therefore, five different processes are executed concurrently. All the processes run at the robot's processing unit in Linux.

All processes communicate by means of a RTDB[2] which is physically implemented in shared memory. The RTDB is a data structure which contains the essential state variables to control the robot. The RTDB is divided in two regions, the local and shared regions. The local section holds the data local to the processes and is not to be broadcasted to the

[2]Real-Time DataBase

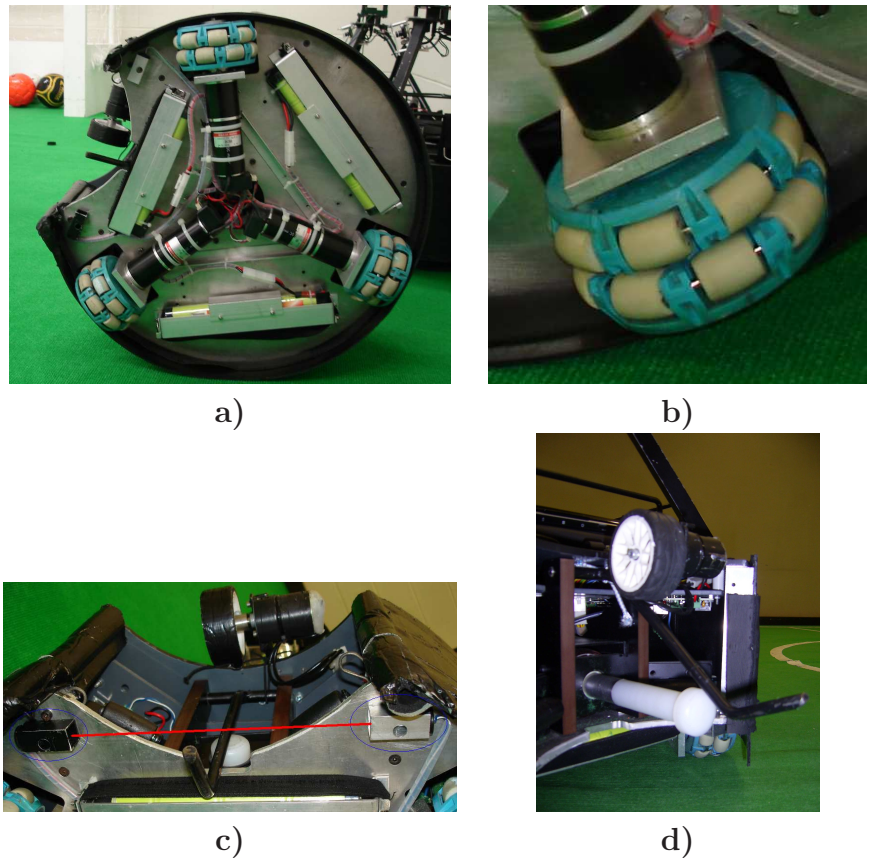


Figure 3.5: The lower layer. **a)** The motion system. **b)** Detail of an omni-wheel. **c)** The barrier sensor. **d)** The kicking system.

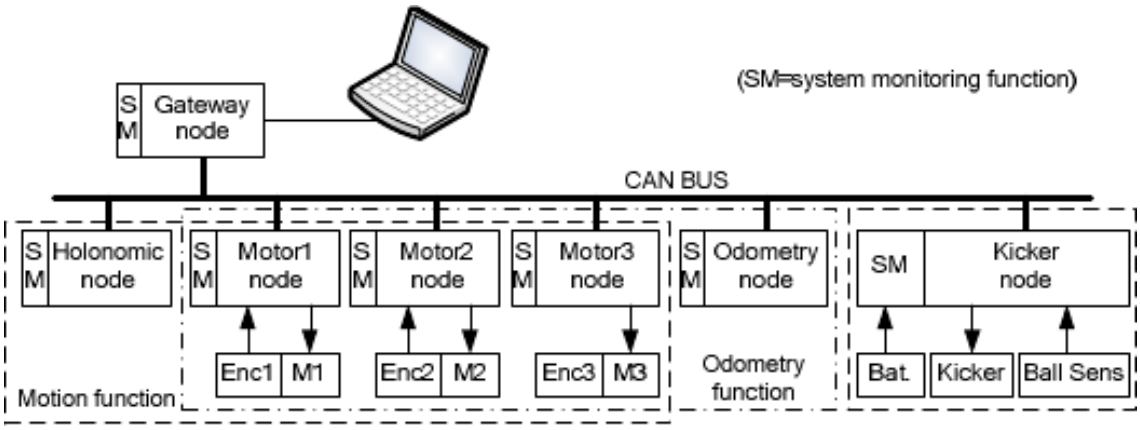
other robots. The shared section is further divided to contain the data of the world state as perceived by the team. One of the areas is written by the robot itself and broadcasted to the rest of the team while the other remaining areas store the information received from the other team-mates. The RTDB implementation guarantees the temporal validity of the data, with small tolerances [32].

The software architecture is depicted in Fig 3.7

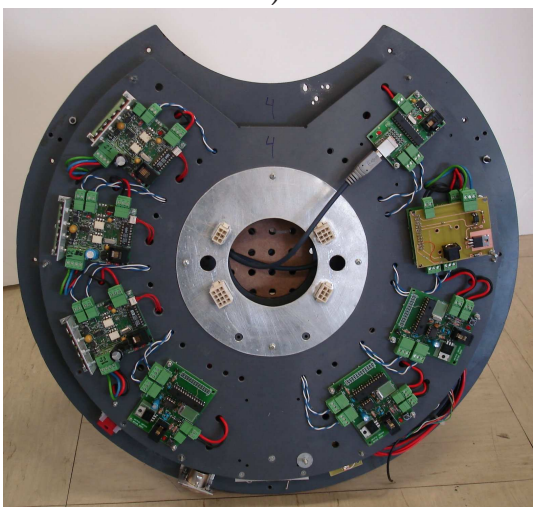
The processes composing the CAMBADA software are:

Vision which is responsible for acquiring the visual data from the cameras in the vision system, processing and transmitting the relevant info to the CAMBADA agent. The transmitted data is the position of the ball, the lines detected for localization purposes and obstacles positions. Given the well structured environment the robots play in, all this data is currently acquired by color segmentation [28][27].

Agent is the process that integrates the sensor information and constructs the robot's worldstate. The agent then decides the command to be applied, based on the perception of the worldstate, accordingly to a pre-defined strategy [33].



a)



b)

Figure 3.6: The micro-controller network. **a)** The hardware architecture, adapted from [25]. **b)** The physical structure.

Comm that handles the inter-robot communication, receiving the information shared by the team-mates and transmitting the data from the shared section of the RTDB to the team-mates [34][35].

HWcomm or hardware communication process is responsible for transmitting the data to and from the low-level sensing and actuation system.

Monitor that checks the state of the remaining processes relaunching them in case of abnormal termination.

Given the real-time constraints all process scheduling is handled by a library specifically developed for the task, pman, process manager.

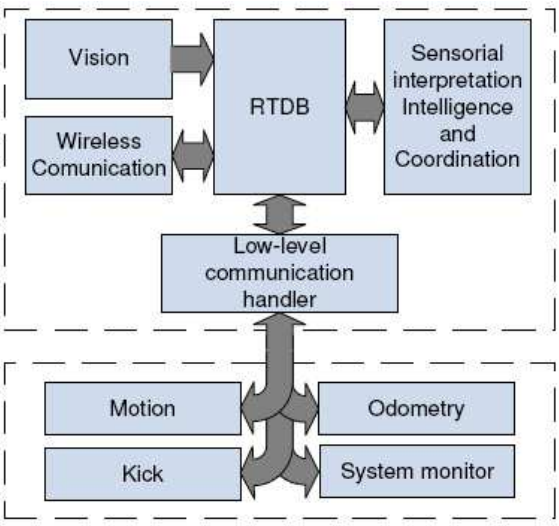


Figure 3.7: The software architecture, adapted from [25].

3.3 Robot Motion Control Loop

Part of this project's work focuses on robot motion control. Since CAMBADA robots are a discrete time closed-loop controlled system, this sections presents an overview of a control cycle.

Firstly, the robot control cycle is imposed by the camera's frame-rate, which means that when a frame is captured a new control cycle begins. The remainder of the robot control must be done before a new frame is available. The CAMBADA robots are therefore a good example of a real-time system since careful task scheduling must be achieved.

After a frame is acquired, processed and its relevant data injected in the RTDB, the vision process signals the agent process to begin executing.

The agent starts by integrating the sensorial data recently acquired, such as the vision info, the odometry displacement, the electronic compass data and the barrier state data. After all data is integrated, the worldstate is updated.

Then a decision module, assigns a given role to the robot so it will execute the team's strategy. Consequently, the role chooses an adequate behavior that performs a specific task.

In the behaviour a command is calculated for each of the robots actuators. In the robot motion control scope, setpoints to each of the robot's wheels must be provided at every control cycle.

This is firstly done by translating the desired pose from field coordinates to robot coordinates which gives an offset relative to the robot's current pose. From this offset, error measurements are determined that are then applied to PID controllers to determine the linear and angular velocities to be applied at the robot's frame. The command data is then injected at the RTDB.

Finally the low-level communication process, reads the command data from the RTDB and produces the respective messages to be sent to the low-level sensing and actuation system.

Historically, there was a micro-controller in the low-level sensing and actuation system, responsible for the specific task of mapping the wheels setpoints from the velocities to the applied at the robot's frame. However the computation costs were considered too heavy for a PIC and this task is currently executed in an Holonomic Motion module implemented in the HWcomm process.

The HWcomm process also collects the data from the low-level system with the recent low-level sensor readings and introduces the data in the RTDB for the next control cycle.

The overall motion control cycle is depicted in Fig 3.8.

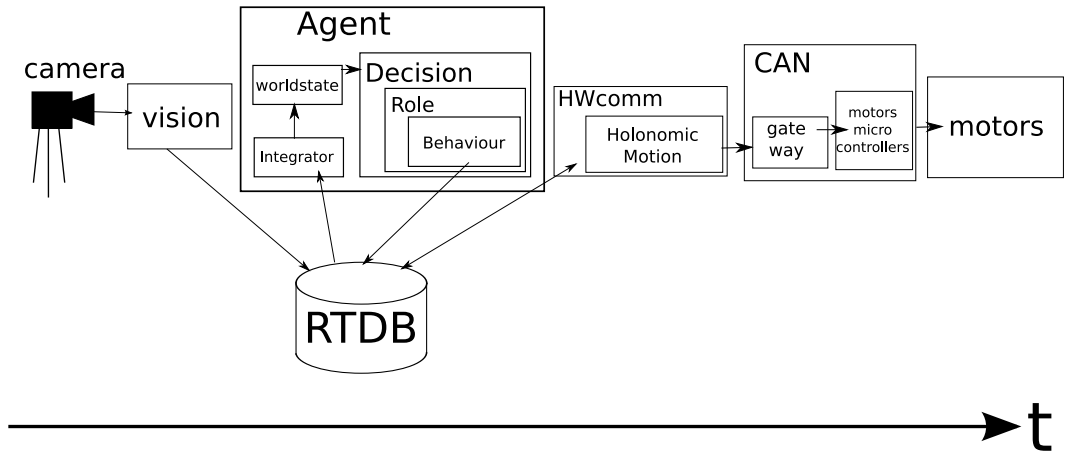


Figure 3.8: Flow-chart of the robot motion control.

Although the

3.4 Summary

At the end of this chapter, one should be familiar with the CAMBADA robots structure and architecture. The different hardware components were presents, namely, the sensors, controllers and actuators. The software architecture was also discussed along with an overview of a control cycle.

Chapter 4

Holonomic Control

In this chapter an overview of the CAMBADA motion is presented and holonomic control related issues are referred. The current solutions are discussed and compared to developed contributions.

4.1 Restrictions

As mentioned before, the CAMBADA robots are mobile wheeled robots. They move with the aid of 3 omni-wheels placed at the periphery of the robot at angles that differ 120 degrees from each other, each being powered by a 24 V / 150 W Maxon motor. Such configuration enables a robot to move in any direction while facing any orientation. Since all degrees of freedom are controlable the robot's motion is holonomic.

Ideally the robot's linear and angular velocities would be mutually independent. This means that a robot could move in a straight line while rotating over itself. In practice, such control is negatively affected by a diverse array of factors that if ignored results in an undesired robot movement.

While rotating and moving at the same time, the robot deviates considerably from the ideal straight line. This deviation is consistently to the right when rotating clockwise and to the left when rotating counter-clockwise, as shown in Fig 4.1.

An initial solution to minimize the deviation consists in rotating the robot at the initial position and then applying the linear velocity. This solution is not ideal since it doesn't take advantage of holonomic motion. Also, CAMBADA robots are among the slowest moving robots in the MSL [36][37][38][39], so this solution only emphasizes the speed difference.

The physical restrictions affecting holonomic motion control presented in the following sections are: discrete time control, maximum acceleration, actuator saturation, and control latency. Given the difficulty in quantification, wheel slippage and slacks are not accounted for the solution of the problem at this stage.

For clarification on the remainder of the chapter, a command at this level is a tuple v_x, v_y, ω which represents the desired velocities with respect to the robot frame.

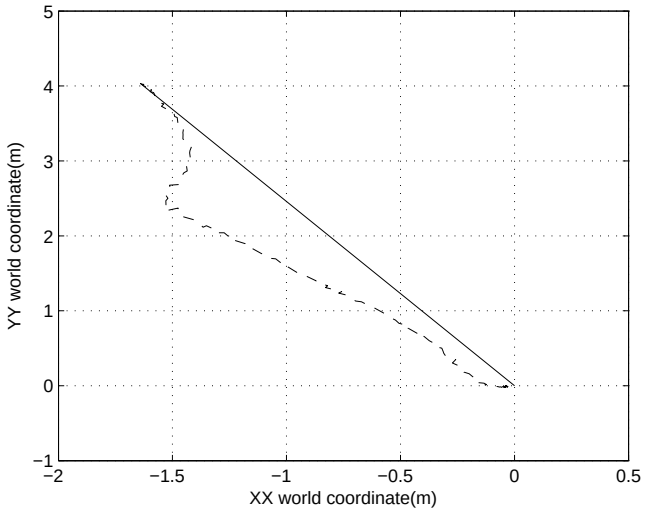


Figure 4.1: Deviation in the robot movement when it rotates and moves simultaneously. The dashed line is the real robot path and the solid line would be the ideal straight path. The robot is rotating clockwise and moving towards (0,0).

4.1.1 Discrete Time Control

Firstly, no digital computer closed loop control exists that can update its output on a continuous basis since these systems are discrete in nature. Also, no real system can read their sensors measurements, process and apply commands instantaneously. Therefore discrete time control usually happens sequentially over a course of time defined as control cycle. This means that a command applied at any given instant is active until the command of the following cycle is applied.

This limitation prevents some robot movements as is the case of moving in a straight line while rotating over its center, as this movement would require continuous variations of the wheels speed setup. Using constant setups during the control cycle, the robot will describe an arc instead of moving over a straight line. The solution is to make the initial and final positions of the arc described in each control cycle lie on the intended straight line. This can be obtained by applying an angle offset to the linear velocity vector. Besides this angular offset, the speed needs to be increased since the length of an arc is longer than the length of its corresponding chord.

This problem can be addressed mathematically using the circle geometry and the chord-arc relationship as represented in Fig 4.2. The direction deviation is given by the angle α between the chord and the tangent to the arc in the chord's initial position. This shall be half the integral of the angular velocity. The chord-arc relationship states that given a chord and its central angle, the length of the arc connecting the initial and final positions of the chord is given by multiplying the chord's length by

$$\frac{ArcLength}{ChordLength} = \frac{2\alpha r}{2\sin(\alpha) r} = \frac{\alpha}{\sin(\alpha)}$$

Since the angle used for correcting the direction,α is an inscribed angle, the arc's central angle shall be the double. So, as mentioned before, $\alpha = \frac{\omega \cdot \Delta t}{2}$.

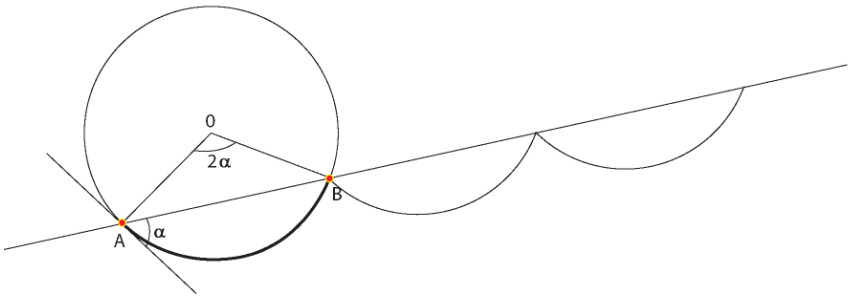


Figure 4.2: While rotating and moving simultaneously the robot will describe a series of arcs along the straight line.

In the robot, at every cycle, the length of the chord or straight line is calculated by integrating desired linear velocity to be applied over a cycle time. Then if the desired rotational velocity is not zero the length of the arc is calculated as mentioned above. The final velocity to be applied is obtained by $\vec{v}_{arc} = \frac{\alpha}{\sin(\alpha)} \times \vec{v}_{line}$, where $v_l\vec{i}ne = \sqrt{v_x^2 + v_y^2}$.

4.1.2 Acceleration

Another restriction affecting omni-directional motion is the allowed values for the acceleration. In the RoboCup Small Size League some teams limit their robots acceleration by wheel traction only [40]. Given the Middle Size League robot's dimensions, this would result in a very dangerous behavior for the robot because if a large velocity variation is allowed in a single cycle, there is a possibility of tipping off, severely damaging the robot's components. Given the electric nature of the used motors such unlimited acceleration can also cause large current peaks reducing the hardware lifetime and battery charge.

Therefore, the CAMBADA robots have an acceleration limiter at software level. This means that at each cycle the robot's velocity varies only up to a certain amount around the previous cycle's velocity. The velocity then changes over time smoothly. A geometric representation of the limiting algorithm is given in Fig 4.3.

The CAMBADA robots also have a angular acceleration filter mainly to avoid current peaks since a large variation of angular velocity wouldn't make the robot tip off.

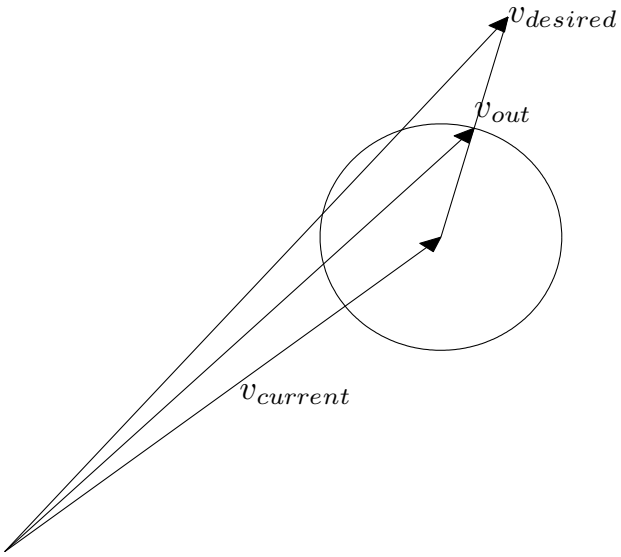


Figure 4.3: Acceleration limiter. The circle represents all the velocities that can be achieved given the maximum acceleration.

4.1.3 Actuator Saturation

Another factor adversely affecting an omni-directional robot control is the saturation of its different actuators. Of all the different actuators of CAMBADA robots, this project's work will focus only on the motors controlling the omni-wheels. The saturation of the motors is an indirect indication of the maximum velocity a robot can achieve. In the CAMBADA case, simulation results point to a maximum speed of 2.2 m/s. This physical factor must not be overlooked. It affects not only the proportion between linear and angular velocities but also the proportion between the linear velocities components (v_x, v_y) diverting the robot from its path in case of actuator saturation.

Assuming a saturation value C, considering the 3 omni wheels, the velocities v_x, v_y and v_a, as the velocities along the robot's axis and angular velocity, respectively, the setpoints to apply at each wheel are calculated. Assuming that all setpoints exceed C, if no measure is taken the robot would rotate at maximum speed instead of moving according v_x, v_y and v_a. The solution involves determining a correction factor to be multiplied by all the velocities to be applied [41].

This correction factor is $\frac{C}{|S|}$, where S is the highest setpoint calculated.

This ensures that one of the setpoints is applied the maximum input possible and the others are in the original proportion before the saturation limitation. The resulting movement will be slower than desired but the direction of movement will be maintained while moving as fast as possible.

4.1.4 Control Latency

The final physical restriction mentioned in this thesis is the system delay. In the RoboCup Middle Size League, some teams have already addressed this issue, as is the example of the Tribots team [42]. System delay results from the sum of all small delays in the control loop and greatly affects the robot control. A delay between the sensory input and the actuator output means a command calculated based on a worldstate A will be executed on a worldstate B that may or may not be the same as A. Considering the highly dynamic nature of a robotic soccer game, the worldstates will most likely not be the same. The problem is depicted in 4.4.

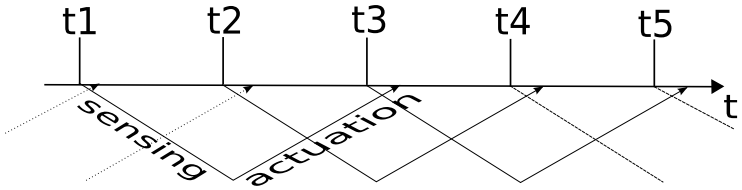


Figure 4.4: Effect of control latency over cycle time. A command issue at based on information of a given cycle will only have effect some cycles later.

The problem worsens as the delay increases not only because the error between the worldstates increases but also because as the delay overcomes the cycle time, more and more commands will affect the worldstate before the current command will start to produce effects. This fact explains the strong deviation while moving and rotating simultaneously shown in Fig 4.1. The robot is stopped, and will try to move and rotate. Since the command calculated in the initial cycle will only reach the actuator some cycles later, all the control cycles in between them will calculate the same command since the robot will not move until the first command arrives. Ideally after the robot moved from the first command, a new one should be input considering the robot's new pose.

4.2 Predictive Control

Reflecting other teams solutions [40], a prediction module was implemented to determine the pose of a robot when the current command will reach the actuators, given the sensed pose and a buffer storing the commands that have already been issued but have yet to be executed due to the system delay. Thus, all commands are calculated without delay.

The commands considered in the buffer of the prediction module do not reflect the desired velocities, but the actual input commands at the motors, after the saturation and acceleration limiters are taken in to account.

Considering the CAMBADA software structure presented in chapter 3, the desired commands are calculated on the agent process and transmitted, through the RTDB, to the low-level communication handler process, or HWcomm, where they are processed through

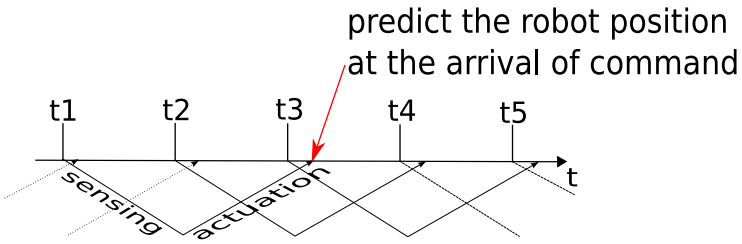


Figure 4.5: Predicting the robot position at the end of delay considering previous commands.

the saturation and acceleration limiters before being mapped to setpoints. In order to implement the prediction module in the agent process where the desired commands will be calculated accounting the predicted robot pose, the output commands of HWcomm must be sent back to the agent process, again through the RTDB, as shown in Fig 4.6.

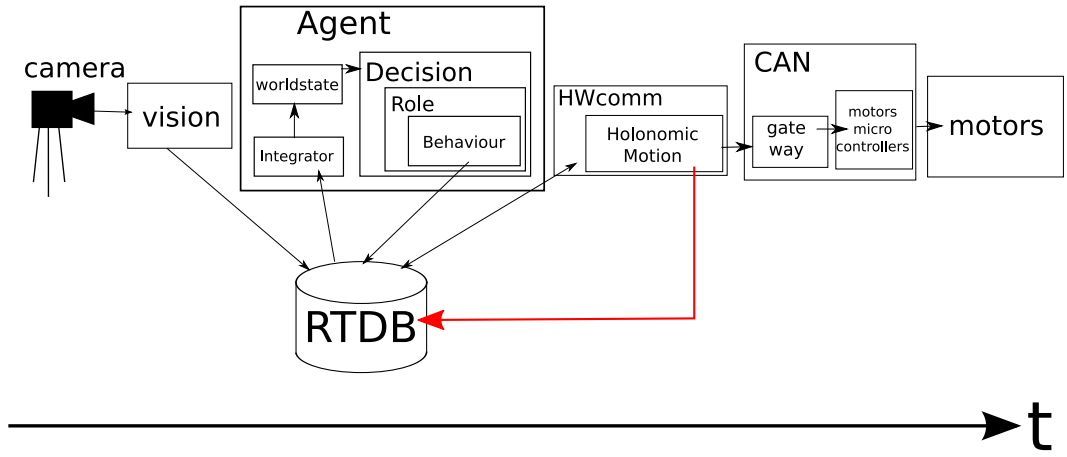


Figure 4.6: Changes in the information flow in the robot motion control scope.

With the information available in the RTDB, the module *Integrator* was altered to read the last processed command from the RTDB and store it in a sliding window with size $\lceil \frac{t_{delay}}{t_{cycle}} \rceil$.

Predicting the robot pose at the end of control latency is not sufficient. For coherency all of the world state as perceived by the robot should be predicted such as the ball, the team-mates and the opponents position. However such goal is not feasible since a CAMBADA robot doesn't visually distinguish an obstacle from friend or foe. While it is possible since the team-mates positions are shared through wireless communication, this process is not synchronized with the robot control cycle, running in the best case every 0.100 seconds. Thus making the prediction calculus inaccurate. Also, at the moment no tracking is done to determine an opponent path, thus no velocity estimation is done. Therefore predicting the opponents position at the end of the control latency is impossible,

since opponent are considered stationary, for obstacle avoidance purposes. On the other hand, the ball's position and velocity are estimated and therefore a position prediction is possible.

These predictions are executed in the method *predict* at the end of the *Integrator* module execution which edits the robot pose and ball position in the constructed worldstate that will then support all the high-level decision taken by the agent.

4.2.1 Robot Model

As each command is active for an entire cycle and given the electric motors used, a purely uniform movement is assumed for each cycle. Then the predicted pose, $\hat{p}_{t_{pred}}$ for the robot is given by,

$$\hat{p}_{t_{pred}} = p_{t_{sensed}} + rem \cdot c_{t_{n-1}} + \sum_i c_{t_i} \Delta t, i = sensed - n..sensed - 1 \tag{4.1}$$

where

- $p_{t_{sensed}}$ is the self-localization determined pose,
- c_{t_i} is the command issued in the i-th control cycle,
- $n = \lfloor \frac{t_{delay}}{t_{cycle}} \rfloor$,
- $rem = delay - n \cdot \Delta t$.

4.2.2 Ball Model

Since at the moment the robots don't detect if the ball is accelerating or slowing down, it is assumed the ball moves with uniform movement. Thus, the predicted ball position, p_{pred_b}, after the delay is,

$$p_{pred_b} = p_{sensed_b} + v_b \times \Delta t \tag{4.2}$$

where p_{sensed_b} is the position of the ball determined by the robot sensors, v_b is the ball velocity and Δt is the control latency.

However, this could cause a situation where the robot would predict the ball inside it's own body frame. Therefore the ball prediction model must account for collisions with the robot.

Instead of calculating when two circular bodies collide, the ball and the robot, for simplicity, the ball is considered as a particle and the ball radius,r_B is added to the robot's radius,r_R, since the robot has a circular base.

Then a collision point is calculated when a line, the path described by the ball center, intersects a moving circle, the robot, with a radius equal to the robot radius and the ball

radius. The path described by the moving circle is the resulting predicted path from the commands that have not yet arrived at the motors.

The collision time ,t , is calculated by,

$$\begin{aligned} (x - x_R)^2 + (y - y_R)^2 &= (r_R + r_B)^2 && (4.3) \\ x &= x_B + v_{B_x} \times t && (4.4) \\ y &= y_B + v_{B_y} \times t && (4.5) \end{aligned}$$

where x_R and y_R are the components of the world coordinates of the center of the robot frame given by 4.1.

If the collision time is lower than the control delay, the ball will collide with the robot before all commands sent are executed, and the collision point,$p_{collision}$, will be given by replacing t in 4.4 and 4.5.

However, since the robot base is not a full circle, the robot's front where the ball is engaged must also be accounted for. The robot model is depicted in Fig 4.7. The real robot dimensions are represented by the black line and the considered dimensions, for collision with the ball, are represented by the blue line. If the ball center is on the blue line, then an edge of the ball has collided with the robot's frame.

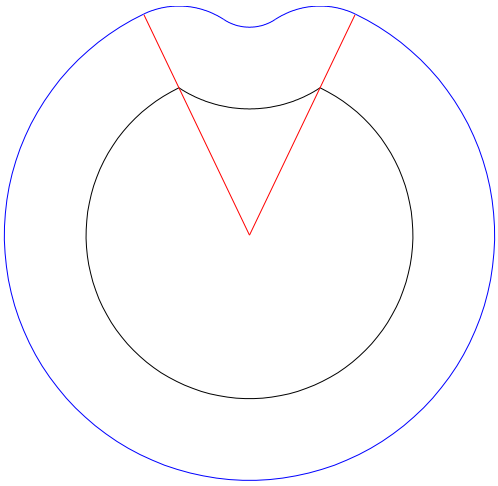


Figure 4.7: The considered robot dimensions assuming the ball as a particle.

To calculate the direction of the ball velocity after the collision, it is assumed that the ball is reflected, like light on a mirror, after hitting the robot's body. Despite not being a realistic model, the obtained results show its simplicity outweight its inaccuracy.

Thus the ball position and velocity, $p_{B_{pred}}$ and $v_{B_{pred}}$, respectively, after the collision shall be,

$$p_{B_{pred}} = p_{collision} + v_{B_{pred}} \cdot \Delta t \tag{4.6}$$

$$v_{B_{pred}} = (v_B - v_R) - 2N\frac{(v_B - v_R) * N}{\|N\|^2} + v_R \tag{4.7}$$

where Δt is the difference between the control delay and the collision time and N is the normal vector to the robot's surface in the collision point. Note that the reflected velocity is the relative velocity between the ball and the robot which is then added to the robot velocity. This happens so that when the robot is chasing the ball, the reflected velocity resulting from a collision will not point towards the robot body. A reflection scheme is depicted in Fig 4.8.

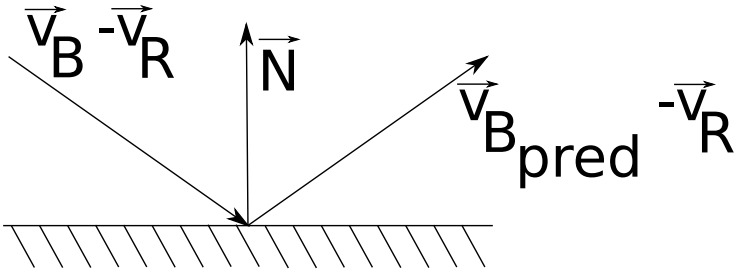


Figure 4.8: The reflection of the ball velocity when it collides with the robot.

Despite of the fact that a ball collision with the robot's front could result in the robot engaging the ball due to the ball retention system, the ball model makes no assumptions in this case, assuming there will always be a collision and therefore a reflection. On the other hand, if the ball is detected as engaged no ball prediction is done. At the end of the robot prediction position the ball is "placed" at the robot's front maintaining its relative position to the robot's center.

However a problem remains. The current estimation of the ball velocity is based on a linear regression with a dynamic number of samples. The algorithm starts with a minimum number of three samples and evolves up to a sliding window with a maximum of the ten most recent samples. This algorithm detects a collision by predicting the ball position considering the last known position and the last estimated velocity. Then it compares with the detected position of the ball. If the prediction fails by a large margin three consecutive times, the number of samples is reset to the minimum three samples [43].

This means the described method only detects a collision three cycles after the collision occurred. Ideally in the cycle after the collision occurred, a new velocity should be computed. However with the current implementation a velocity with a similar direction to the velocity before the collision is returned.

This would mean that the ball model of the predictive control would detect a collision since the estimated velocity would still point towards the robot's frame.

Thus a method is proposed that allows a faster convergence of the ball velocity when the robot detects it will collide with the ball. In the predictive control algorithm, if a collision is detected then the reflected velocity is estimated as mentioned earlier. However if the collision will happen in less than a cycle period then the number of samples in the linear regression is reset to three. These three samples are then moved to form a line with the direction of the reflected velocity. This will seem as if the ball came from inside the robot frame. Therefore the returned velocity in the following cycle will point away from the robot's frame. And since the linear regression has few samples, to represent the small degree of confidence in velocity, the estimation will rapidly converge to the real ball velocity, provided by the samples of the following cycles.

This method is explained in Fig 4.9.

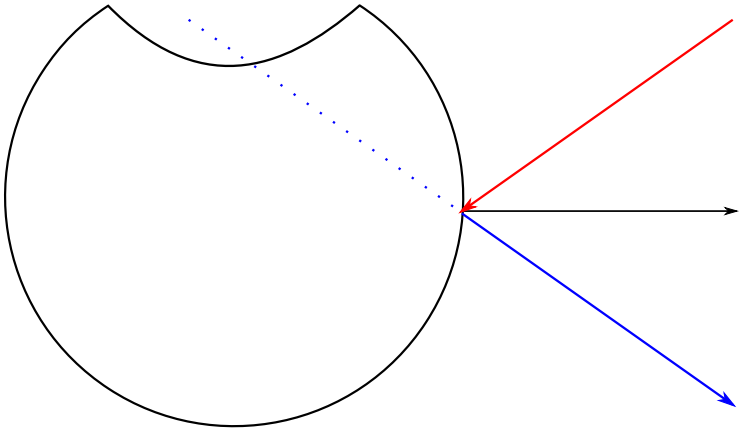


Figure 4.9: The method for faster ball velocity convergence after a collision.

4.3 New Acceleration Filter

Surprisingly the resulting path using predictive control was not the expected as the robot still doesn't move in a straight line. This is an undesired result from the current implementation of the acceleration limiter. As can be seen in Fig 4.3, if the desired velocity diverges too much from the current velocity, the output of the limiter will be a velocity that will neither be in the desired orientation nor in the desired norm. A solution to this problem is to increase the maximum acceleration of the robot, increasing the velocity variation every control cycle. However, simulation results showed that to compensate the large divergences between the velocities, the acceleration would have to be increased to values where the robot's motion stability would be compromised. These conditions when tested revealed another issue. While such high maximum acceleration allows for great velocity variations from cycle to cycle, the same high variations produce a considerable wheel slippage. Since this factor is not accounted for in the model, the resulting prediction

will be very inaccurate. Therefore, the robot will not generate the anticipated path, as shown by Fig 4.10.

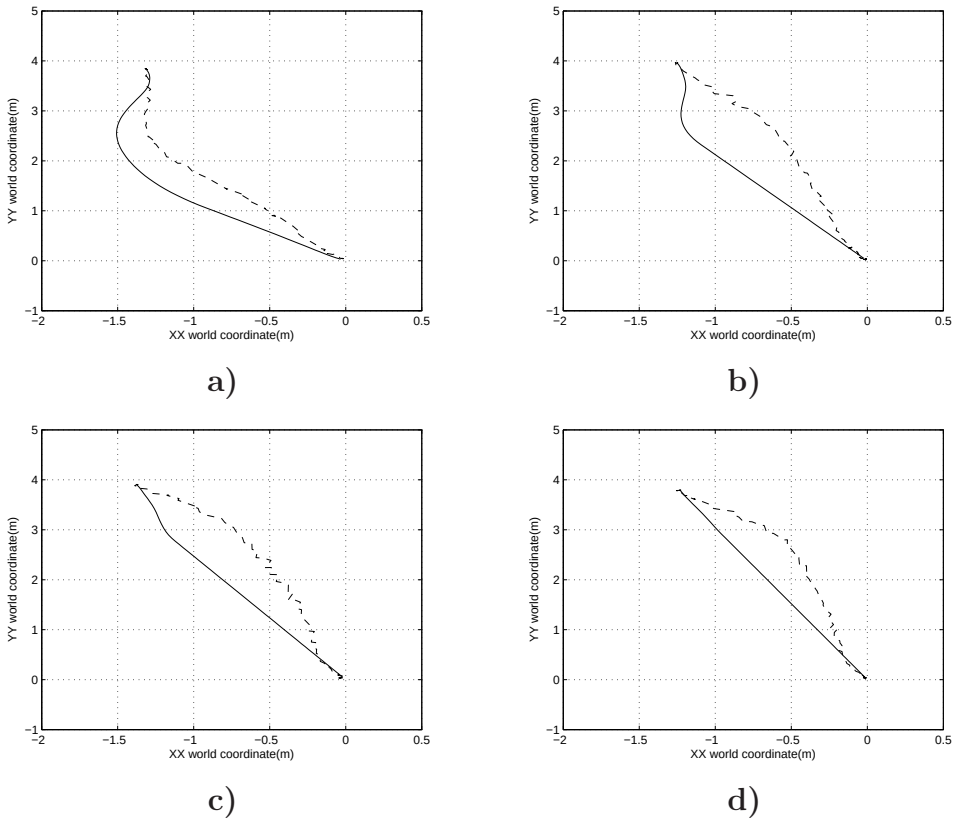


Figure 4.10: Robot path with different values of maximum acceleration. The dashed line represents the real robot path while the solid line represents the simulated robot path. **a)** Maximum acceleration $3m/s^2$. **b)** Maximum acceleration $5m/s^2$. **c)** Maximum acceleration $7m/s^2$. **d)** Maximum acceleration $10m/s^2$.

In this section, a new algorithm to update the velocity while complying with a maximum acceleration is suggested which prioritizes the velocity's direction over its norm. The solution involves reducing the robot speed, in order to successfully rotate, similarly to what we humans do when driving. In order to reduce the speed by a minimum necessary the following situations are considered:

Firstly, the closest point from the current velocity to the desired velocity vector is calculated. Uniting this point to the current velocity always creates a perpendicular line to the desired velocity. So the distance,d , between the current velocity and the closest point will be,

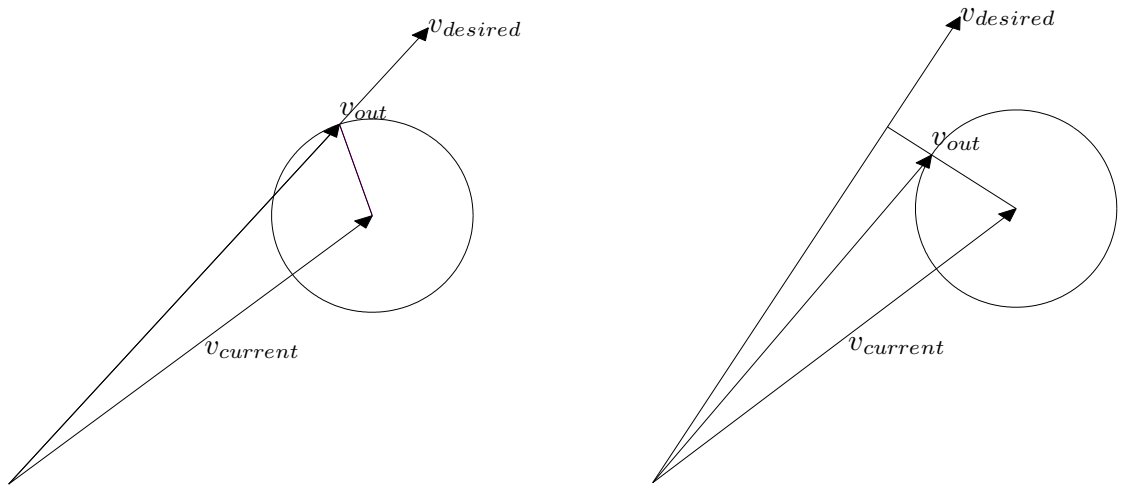
$d = sin(\Theta) \times |currentVelocity|$,

where Θ is the angle difference between the desired and current velocities.

Then if the distance is larger than the maximum velocity variation allowed, it means the robot doesn't have enough acceleration to reach the desired velocity direction. So to

minimize the direction error, the output velocity of the acceleration limiter will be in the direction of the closest point.

If the distance is smaller, that means the robot can achieve the desired direction of the velocity even if not at the same norm. So the interception point between a circumference centered at the current velocity and with a radius of maximum velocity variation and the desired velocity vector is calculated. This point represents the maximum velocity the robot can achieve given the maximum acceleration limit while moving for the desired direction.



a) **b)**

Figure 4.11: New implementation of the acceleration limiter. Left **a)**: Priority is given to the direction of the desired velocity over its norm. **b)**: When the desired velocity can't be achieved given the maximum acceleration the output will be in the perpendicular line to the desired velocity that passes in the current velocity point.

An exception must be made when the angle difference between the desired velocity and the current velocity is greater than 90 degrees. The resulting velocity using the perpendicular method would point in the opposite direction of the desired velocity. In this case the original implementation of the acceleration limiter is used.

4.4 Estimating Model Parameters

The next step is to determine the control delay. For this purpose some captures were made while the robot moved. In these captures, the robot pose determined by self-localization and the commands sent to the actuators were recorded. Since the capture produces results every cycle, the delay is determined by the difference between the cycle

corresponding to the first command sent and the cycle corresponding to the robot movement. This way, it means that when the robot pose changed the first command arrived at the motors. However, since the framerate of the camera used is 25 fps, it means the delay will be in an interval of $\frac{1}{25} = 0.040$ seconds. Using this method the delay affecting the CAMBADA robots would be in $[0.160; 0.200[$ seconds. Experiments were conducted to determine the delay with more precision, which tried to minimize the error between the simulated robot path and the robot real path. However at this stage none have proven trustworthy. So, as last resort, different delay values, between 0.160 and 0.200 seconds, were tested while the robot moved between the same two points. This method showed that when delay tends to 0.200 the robot path deviates further from the straight line. The chosen value for delay was 0.165 seconds, which provided the best resulting paths.

4.5 Results

Using the prediction module combined with the new acceleration limiter implementation the robot successfully moved in a straight line while rotating simultaneously as shown in Fig 4.12.

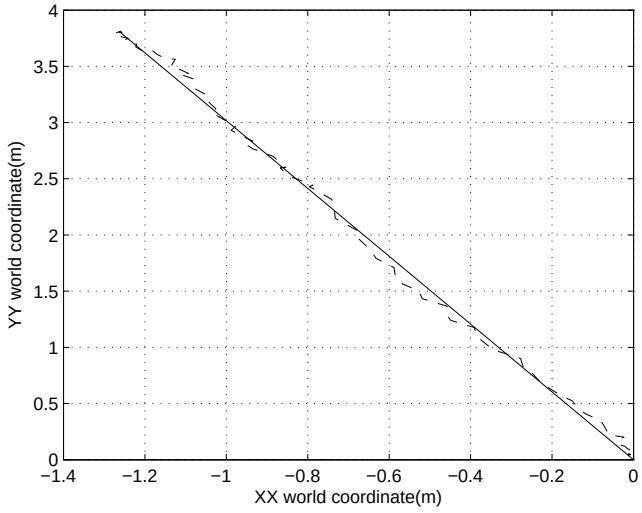


Figure 4.12: The robot path using new acceleration limiter and robot pose prediction after delay. The delay value is 0.165 seconds. The robot is rotating counter-clockwise towards (0,0)

An experience was then conducted to compare the developed solution and the original solution, where the robot rotates first and moves afterwards. The experience consisted on moving the robot between two points using the previously mentioned solutions. The process was repeated ten times and the times were recorded for every run.

With the original solution, the robot took an average 3.62 seconds to move from one point to the other with a standard deviation of 0.11 seconds. With the developed solution, the robot took an average 3.28 seconds to move between the same two points with a standard deviation of 0.16 seconds.

This results in a mean time difference of 0.34 seconds. Although the improvement is not considerably large, given the fast pace of a soccer game, the increased speed might prove crucial in some game situations.

Finally an experience was conducted where the robot moved with a constant angular velocity. Even in this adverse condition the robot performance was acceptable, even despite not moving in a straight line and producing an overshoot.

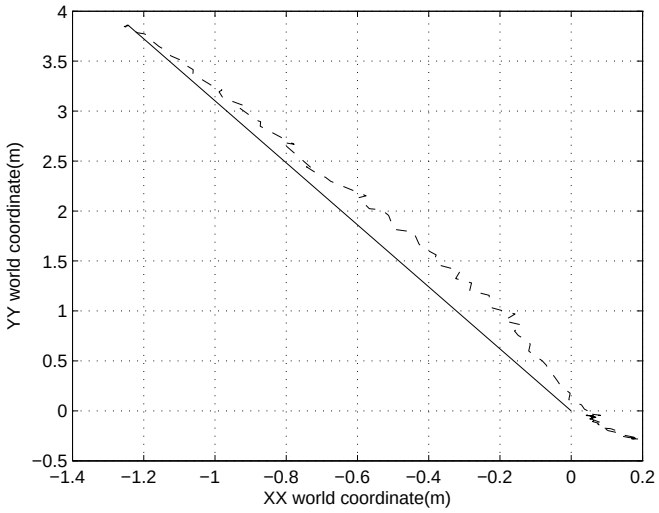


Figure 4.13: The resulting robot path with constant angular velocity. The robot is rotating counter-clockwise towards (0,0).

The reflective collision ball model also proved effective specially with the method for faster convergence of the ball velocity estimation. The robot can now predict, considerably well, the ball position after the control delay, as shown in Fig 4.14.

Following the implementation of predictive control, some ad-hoc corrections could be removed from the existing behaviours, that were implemented because of the undesired robot movement due to control delay.

4.6 Summary

Several aspects affecting holonomic motion control were presented along with developed contributions. Special focus was given to predictive control to address control latency issues. Also an innovative algorithm was implemented to update the robot velocity which

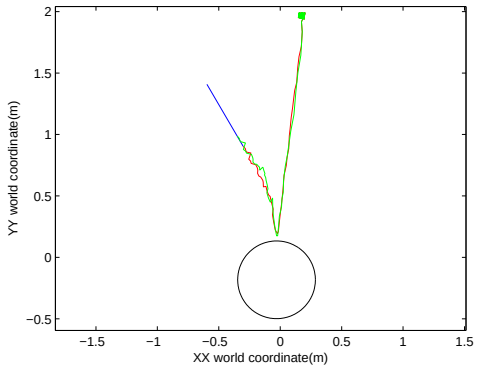


Figure 4.14: Predicted ball positions after control delay. The red line represents the ball positions sensed by the vision sensor and the green line represents the predicted ball positions. The blue line is the ball velocity.

prioritizes the direction of movement over its speed. The implemented work on this field improved the robot motion and overall speed.

Chapter 5

Stuck Detection

In robotics in general and in robotic soccer in particular, autonomous robots may benefit if they are able to detect stuck situations. A stuck situation happens when a robot is blocked from progressing in the desired trajectory. Despite one of the robot basic behaviours is obstacle avoidance to prevent these situations, robots tend to get stuck.

In the robotic soccer scope, given the highly dynamic environment of a game this situation happens oftenly, especially when two robots dispute the ball. If the situation holds for some time, a drop ball is awarded, where the robots must stay one meter away from the ball and can only move after the referee gives the signal. However, if the teams have similar robots with respect to speed, they will reach the ball at the same time and a new drop-ball may be awarded.

If not handled properly, stuck situations can have severe outcomes. The robot can damage itself, the field, the opponent's robots or worse, one of the referees. Therefore the teams must make an effort to preserve the welfare and property of the different stakeholders.

In this chapter different methods of stuck detection are presented and the implementation used in the CAMBADA team is discussed.

5.1 Known Methods for Stuck Detection

5.1.1 Haptic Sensors

Haptic, from the greek *haptikós* is the ability to grasp or perceive an object through touch [44]. Despite being considered one of the five human senses, touch is actually a somatosensory system [45] comprising pressure, texture and temperature sensors. However haptic sensors don't fully replicate touch as they are only capable of measuring pressure.

In robotics, haptic sensors found great use in solving manipulation problems applied in robotic arms. In the stuck detection scope the pressure measured by these sensors could easily determine the state of the robot.

Although these sensors reproduce the same information used by humans to get unstuck, it would be impractical to apply these sensors to the same end in robotics, since all the

robot surface would have to be coated with haptic receptors which would be very costly.

5.1.2 Bumper Sensors

Bumper sensors are simpler versions of haptic sensors. Bumper sensors have only two possible states, they either detect contact or not.

Its simplicity can be an advantage as well as a downfall. Although much cheaper than haptic sensors, it is still needed to coat all the robot surface in order to detect a stuck situation. This is accomplished using a high amount of sensors which would allow to detect from which side the robot was being blocked or with few sensors but it would result in a low angular resolution. The addition of hardware is always undesired and collisions with robots of faster teams could even damage the sensors in the process.

5.1.3 Stasis Sensors

Stasis is a state of stability, in which all forces are equal and opposing, therefore they cancel each other [46].

There is a large variety of sensors able to detect stasis situations. An example of a stasis sensor could be a simple drag wheel with an optical encoder [47]. When in a stasis situation the encoders from the driven wheels will indicate a large displacement from the previous position, when all the power supplied to the wheels was lost in slipagge. However the encoder from the drag wheel will demonstrate a small displacement since the robot did not actually move.

Although stasis detection can be simple and effective it suffers from two major drawbacks. First, when the robot is stuck, this doesn't always result in a stasis situation, especially when disputing the ball with other robots, since the robot oscillates. Hence some stuck situations could pass undetected with this method. Second, this implementation involves adding hardware to the robot which has associated costs and must be maintained. Also to achieve faster velocities the robots must be as light as possible, therefore additional hardware should not be added to reduce the weight of the robot.

5.1.4 Accelerometer Sensor

An alternative method to detect stuck situations is through accelerometers. Accelerometers measure the acceleration experienced, relative to freefall.

Besides sensing orientation, vibration and shocks, accelerometers can also detect slippage and have been used to generate kinematic models for robotic arms [48]. In [49] accelerometers are also be used to determine environmental states such as the type of surface the robots walk on, and stuck states through C4.5 classifier.

Besides involving additional hardware that is not currently integrated in the robot, this method might also require previous supervised learning in a training phase before the method can detect stuck situations since the change from game fields may change the

pattern through which the classifier detects these situations. This is inconvenient since the setup time should never be increased.

5.1.5 Image Processing

Stuck detection is also possible through image processing. In conjunction with the information from the wheel encoders, frame comparison can accurately determine the robot state. [50] proposes a subdivision of the frames, where only the portions of the frames that correspond to closer distances are accounted for since they vary less when the robot is stuck. Then the colour of each sub-frame is compared to the previous time-step sub-frame. If the colour difference of the sub-frames is above a pre-determined threshold the robot is stuck. This method optimizes stuck detection while filtering false positives.

Although image processing doesn't require additional hardware, since the CAMBADA robots are already equipped with cameras, this method has computational costs associated which can have severe impact due to the real-time constraints affecting the robot. Another problem with this approach is that the robotic soccer field is mainly green and the lack of features may render this method very erroneous.

5.2 CAMBADA Implementation

Although, there are diverse solutions, such as the previously presented, an adaptation of the solution used by the Brainstormers Tribots team was used [42].

This solution doesn't involve additional hardware and has very low computational costs which is ideal given the real-time constraints affecting robot control.

The method itself relies on the assumption that, when stuck, the estimated robot velocity is very low, since the robot will be almost stopped, despite the decision layer orders the robot to move.

Therefore, using the difference between the desired velocity, v_d, and estimated velocity, v_e, it is possible to determine if the robot was stuck. This is accomplished when the difference, $e = \|v_d - v_e\|$, is larger than a pre-determined threshold. However, as this threshold might be exceeded when the robot is accelerating or decelerating, these situations must be filtered by testing if the derivative of the desired velocity is lower than a second threshold.

Since stuck situations tend to persist for at least a few seconds, intervals of time are analyzed instead of isolated time instants.

Despite the good results obtained by the Tribots team [42], the method presented their paper was not directly applied and it was tailored to the CAMBADA robots specifications.

Due to the fact that the CAMBADA robots determine their velocity through a linear regression, the estimated velocity is a two dimension variable $\{v_x, v_y\}$ while the desired velocity has three dimensions $\{v_x, v_y, \omega\}$. This raises a problem, since it is not correct to identify stuck situations based on the comparison of the linear components of both velocities, since it might result in a false positive in situations where the robot was rotating

around an external point while facing it, as it happens when the robot is dribbling the ball while avoiding opponents This situation is depicted in Fig 5.1.

In this case the desired velocity non-null components will be v_x and ω. However, as a result of the linear regression, the estimated velocity will also have a non-zero component in v_y. Therefore, the difference between the velocities would be larger than the threshold and the robot would detect itself as stuck while it was rotating freely. To solve this problem the situation is determined not by the norm of difference of the two velocities, $e = \|v_d - v_e\|$, but, by the difference of the norms of the two velocities, $e = \|v_d\| - \|v_e\|$. Thanks to this method, the robot no longer detects itself as stuck in the previous situation.

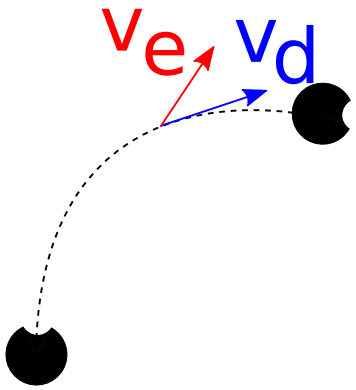


Figure 5.1: Diagram of the discrepancy between the desired and estimated velocities.

As mentioned before, the most common situation where robots get stuck is while disputing the ball, when the robots have opposite velocities and try to avoid one another keeping the ball under control.

The developed method assumes that when stuck the robot's estimated velocity is lower than the desired velocity. However, when disputing the ball, the opposite may happen. Given the different imposed velocities by the robots, the ball contracts and pushes the robots when it expands. This means the robot actually moves back and forth and sideways in a matter of a few seconds which provokes a high estimated velocity. Therefore the absolute value is considered, $e = |\|v_d\| - \|v_e\||$.

Another change was done in the determination of the threshold in the stuck detection. As M. Lauer mentions [42], this threshold must be manually configured. However this threshold shouldn't be a fixed value since if the robot gets stuck with a desired velocity lower than the threshold the detection is impossible. Thus a threshold in function of the desired velocity was used.

For an accurate comparison, the desired velocity is the oldest command sent from the HWcomm process which should be the current executed command. However, since the commands are processed through the acceleration filter and the maximum acceleration value is quite restrictive, the filter often affects the desired value. Thus testing the derivative of the desired velocity proposed in [42] makes no sense since the robot is always accelerating or decelerating as a result of the acceleration filter.

Therefore, the interval of time analised and difference threshold were tuned so that these situations don't result in a false positive. Consequently, the CAMBADA robots detect themselves as stuck if the condition $e = \|v_d\| \cdot 65\% \geqslant |\|v_d\| - \|v_e\||$ holds for at least 0.5 seconds.

Finally, a boolean variable *stuck* was added to the shared section of the RTDB which is updated every control cycle in the *Integrator* module as the robot worldstate is updated.

5.2.1 Results

The developed method was firstly tested in the CAMBADA training field in teleoperated mode to safeguard the robot hardware. Stuck situations were replicated by driving the robot against the field limits or the goal. The robot successfully identified all situations and have never reported a false positive while moving unhindered. However all tests were performed with desired velocities at least higher than 0.5 m/s. This is quite important since when tested in game situations, the goalkeeper in particular due to fine repositions report a high number of false positives per game. Therefore an additional condition was added to the method, if the desired speed is lower than the 0.5 m/s the robot considers himself unstuck.

The fact that the method doesn't work on small desired speeds might explain why the Tribots team don't mention a difference threshold depending on the desired speed. It is most likely that the threshold is above the accepted minimum speed and therefore not being able to identify stuck situations when the robot is moving with a speed lower than the threshold is better than risking false positives.

Fig 5 represents the evolution of the desired and estimated speed during a stuck test. The red line is only for visualization purposes and represents the condition tested for stuck identification, $e = \|v_d\| \cdot 65\% - |\|v_d\| - \|v_e\|| \geqslant 0$. The robot determines if it is stuck if e is below zero for more than 13 cycles which represents 0.5 ms. At first the robot is stopped and begins by accelerating trying to achieve the desired speed. Although the condition is negative in an acceleration phase the time threshold is not exceeded and therefore the robot assumes it is not being hindered on its way. Around time step 50, the robot collides with an obstacle which blocks the robot from moving therefore the estimated velocity begins to decrease while the desired velocity remains constant. Since the situation remained unchanged for more than 0.5 ms a stuck situation was identified around time step 66.

5.3 Summary

This chapter introduces the importance of detecting stuck situations during a game duration. Different methods to achieve this goal were presented and the CAMBADA implementation was discussed. The output of this method is consequently used in the construction of the robot Worldstate.

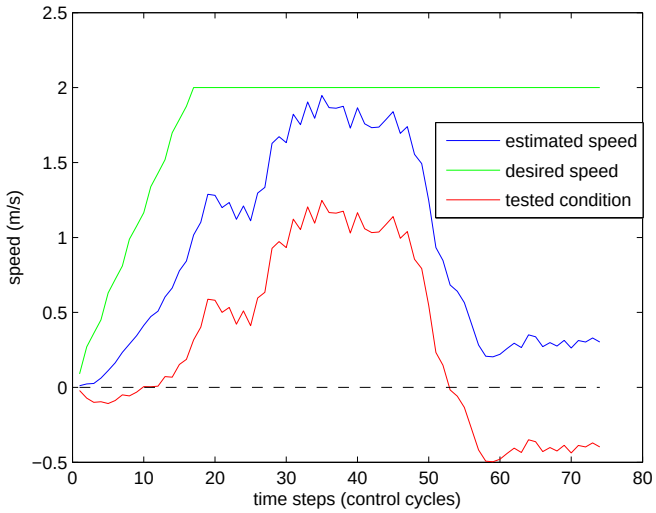


Figure 5.2: Evolution of desired and estimated robot speeds when stuck.

Chapter 6

Behaviours

The scope of this project's work also involved development at the robot's behavioural level. In this chapter the CAMBADA robotic behaviours are presented along with the developed contributions.

The different CAMBADA behaviours represent the basic tasks to be performed by the robot, such as move to a position in the field, dribble or kick the ball. A behaviour can then be seen as the basic block of a CAMBADA robot attitude. A behaviour executes a specific task by computing the desired velocities to be applied at the robot frame, activating the ball handling device and the desired strength to be applied the kicking system.

The choice of a given behaviour at each instant of the game is executed by a role which is basically a finite-state machine composed of various behaviours that allow the different robots to play distinct parts of the team overall strategy.

The various CAMBADA behaviours are depicted in Fig 6.1.

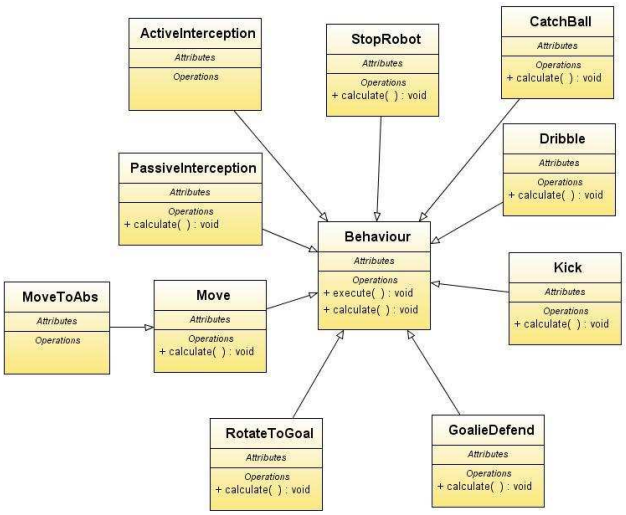


Figure 6.1: Class diagram of all CAMBADA behaviours.

All the CAMBADA behaviours derive from a generic behaviour class *Behaviour*. This class implements the method *execute* which inserts in the RTDB the different values that will later be translated to the powers to be applied at the various robot actuators.

Since the control carried out on the ball handling system is on/off, method *grabberControl* implemented on the *Behaviour* class activates the device based on the position of the ball and when the ball is engaged.

All the derived behaviours have therefore the responsability of calculating the desired velocities to be applied at the robot frame and the behaviour *Kick* in particular calculates the strength to be applied at the kicking system.

6.1 Active Interception

A behaviour previously implemented, that required refinement was the *ActiveInterception*.

The ability to intercept the ball in its path is of major importance in the robotic soccer context. The alternative, moving to the current ball position, is by no means optimal, since a robot takes more time to catch the ball and in some cases it might not even catch it. Fig 6.2 shows a possible described path when the robot moves to the estimated ball position.

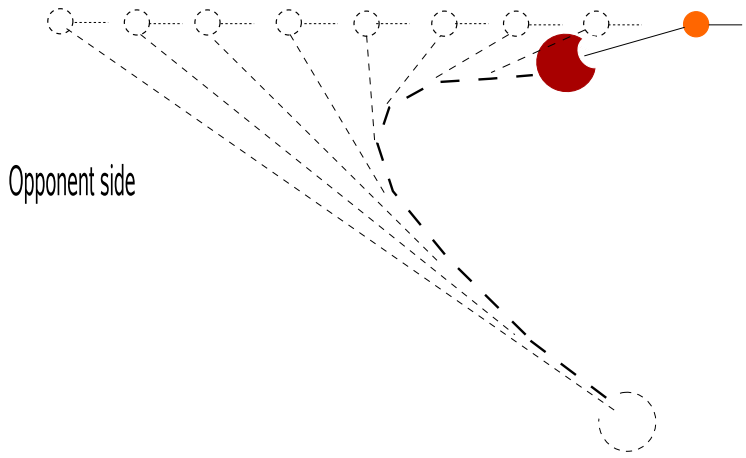


Figure 6.2: Described path when the robot moves to the estimated ball position, adapted from [43].

When considering that the ball could be being dribbled by an opponent robot in the goal's direction, not intercepting the ball could have severe negative consequences. The problem scales as the other teams strive to improve their robot's top speed.

The previous implementation [43] was based on an iterative process that considered the ball path during 20 meters in steps of 0.2 meters and calculated wether the ball or the

robot would achieve faster a given point over the ball path. When the robot would reach that point before the ball, that would be the interception point.

However in the calculation of the interception point, both the ball and the robot dynamic models were considered to have uniform movement. This means that a constant mean velocity is assumed as a parameter that represents the mean velocity with which the robot moves during the interception. This however raises a problem, since it is assumed that a robot can achieved the assumed mean velocity instantly even if it has an opposite direction of the current robot movement.

The new implementation assumes an uniformly accelerated kinematic model instead of a uniform movement kinematic model. Given the work previously described in Chapter 4 the value of the maximum acceleration imposed on the robot movement is known, $a = 3m/s^2$. Therefore the acceleration and current speed are taken into account in the interception point calculation, as well as the robot's maximum speed, max_{speed}. Since the maximum velocity depends on the direction of the movement and wheel slippage and slacks, its value was empirically obtained.

Then the position of the robot, $\vec{p}(t,\theta)$ is,

$$\vec{p}(t,\theta) = \begin{cases} \vec{p}_0 + \vec{v}_0 \cdot t + \frac{1}{2} \cdot \vec{a} \cdot t^2 & \text{if} \quad \vec{v}(t) \leq \vec{v}_{max}(\theta); \\ \vec{p}_{max}(\theta) + \vec{v}_{max}(\theta) \cdot t' & \text{if} \quad \vec{v}(t) \geq \vec{v}_{max}(\theta). \end{cases}$$

where

$$\vec{v}_{max}(\theta) = \{max_{speed} \cdot \cos(\theta), max_{speed} \cdot \sin(\theta)\} \tag{6.1}$$

$$t_{max} = \frac{\|\vec{v}_{max}(\theta) - \vec{v}_0\|}{a} \tag{6.2}$$

$$\vec{a} = \{a \cdot \cos(\phi), a \cdot \sin(\phi)\} \tag{6.3}$$

$$\phi = \arctan(v_{0_y} - v_{max_y}(\theta), v_{0_x} - v_{max_x}(\theta)) \tag{6.4}$$

$$t' = t - t_{max} \tag{6.5}$$

$$\vec{p}_{max} = \vec{p}_0 + \vec{v}_0 \cdot t_{max} + \frac{1}{2} \cdot \vec{a} \cdot {t_{max}}^2 \tag{6.6}$$

Since the model contains a non-linearity, a numerical method would be required to find an interception point $\vec{p}(t,\theta)$. Besides calculating t, the time the interception occurs, θ must also be calculated, which is the direction of the interception point in respect to the robot position or rather the direction to where the robot will move. A geometric representation of an interception is shown in Fig 6.3.

However the iterative method used previously can be reused to calculate the interception point. By testing different points in the ball path, the direction of the interception point is known. Therefore an approach similar to the one previously used is enough to determine the interception point. It is only needed to test whether the ball or the robot reaches sooner given points over the ball path.

However the iterative process was not directly reused. Instead of considering a maximum distance of 20 meters, a maximum time of ten seconds is considered, for calculating the interception point, at the end of which, if no solution is found the robot will move to

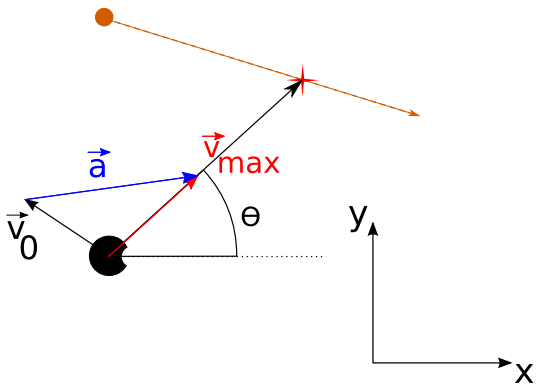


Figure 6.3: Geometric representation of an interception.

the predicted position of the ball after ten seconds. Therefore the step between iterations is now a time step of 0.1 seconds instead of a distance step.

The ball's dynamic model was however unchanged, and it's considered that over the 10 seconds duration the ball will move uniformly over its path.

Thus, in each iteration of 0.1 seconds, the predicted position of the ball, $ball_{pred}$ is calculated, therefore it is known that the ball takes the current iteration time,t_{iter} to achieve that position.

As mentioned, by knowing the possible interception point, $ball_{pred}$, the direction of movement can be deduced, $\theta = \arctan(ball_{pred_y} - p_{0_y}, ball_{pred_{x-p_{0_x}}})$.

Then the instant when the robot achieves the considered maximum velocity is calculated, t_{max} using 6.2. The time the robot takes to achieve the predicted position of the ball,$ball_{pred}$, with uniformly accelerated movement is also calculated,

$$t_{acc} = \frac{\|\vec{v}_0\| \cdot \cos(\theta - \beta) \pm \sqrt{\|\vec{v}_0\| \cdot \cos(\theta - \beta)^2 - 4 \cdot 0.5 \cdot a \cdot -\|p_0 - ball_{pred}\|}}{a} \quad (6.7)$$

where $\beta = \arctan(v_{0_y}, v_{0_x})$.

If t_{acc} is higher than the current iteration time, t_{iter}, then the robot will never reach the interception point before the ball, even considering speeds higher than the assumed maximum speed, and a new iteration is executed. If, however, t_{acc}, is lower than t_{max}, then it means the robot can intercept the ball before achieving the maximum velocity and therefore the predicted position is returned as the interception point.

If none of the previous conditions are met then the robot reaches the interception point before the ball but with speeds higher than the assumed maximum speed. Therefore it must be tested if the robot can still reach the interception point considering the robot will eventually reaches the maximum velocity and will move the rest of the path with at constant speed.

Consequently the position where the robot achieved maximum velocity is calculated, $\vec{p}_{max}$ using equations 6.6 and 6.3. Then the time the robot takes to move to the pre-

dicted position is calculated assuming the rest of the robot's path towards that position is performed at constant maximum speed,

$$t_{const} = \frac{\|\vec{ball}_{pred} - \vec{p}_{max}\|}{\|\vec{v}_{max}(\theta)} + t_{max} \tag{6.8}$$

If the time is lower the current iteration time, a solution was found, otherwise a new iteration is executed.

The described method proved to be efficient as the robot is now able to predict the ball's path and intercept it as shown in Fig 6.4. This figure shows a capture of an interception. The ball and robot positions are stored. The robot starting position is near (2.0, 6.0) and the ball starting position is near (1.5, 3.0). The robot intercepts the ball path near (-0.75, 4.5) when it collides with the ball and deflects it.

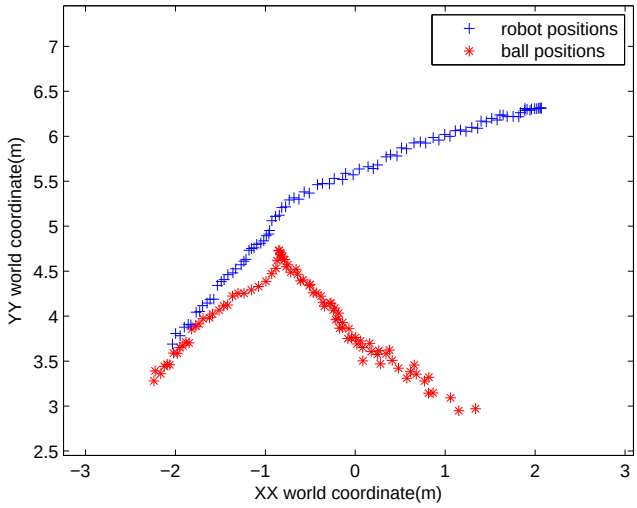


Figure 6.4: An interception. The robot positions are represented by the blue + marks and the ball positions are represented by the * red marks.

6.1.1 Corrections to the Interception Point

Although the interception point calculation seems reliable it is not desirable to always move the robot to the interception point position. Specially since the game is played in a real field which has necessarily physical limits. Therefore it is not recommended to order the robot to move to an interception point outside of the field. If ignored, besides not providing an advantage in the team performance, this situation can possibly damage the robot, one of the referees which are in the side lines, or even the audience assisting to the game.

Fig 6.5 shows the corrections performed when the interception point is outside of the field.

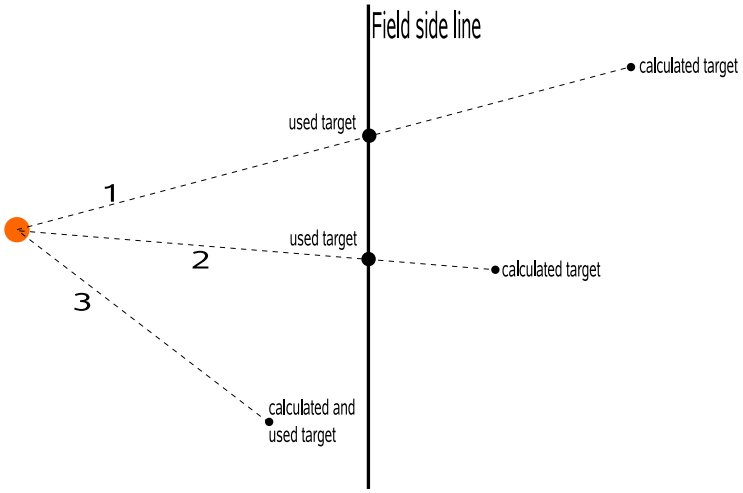


Figure 6.5: Corrections to the interception point, adapted from [43].

6.1.2 Role Assignment

In the CAMBADA robots strategy, in most situations only one robot moves towards the ball, the one with the *Striker* role, while others move in formation. The condition to choose which team-member will move to the ball was based solely on the distance between the robots and the ball. This caused some problems for example when a robot is chosen as the *Striker* while the ball is moving away from it. In this case the ball could be moving towards another robot which would capture it faster than the chosen *Striker*. It has been a team objective, for quite some time, to make this decision based on which robot catches the ball faster.

The previously proposed method for calculating the interception point could be of some help in this case. A small change in the interface of the interception point function allows to return not only the interception point as well as the time the robot takes to intercept the ball.

Therefore it is now possible to choose the robot that captures the ball faster, improving the team's defense and ball possession over the game's duration. In fact this feature was used in RoboCup'09, not to assign the *Striker Role* but to determine if a midfielder should targer the ball after the current *Striker* was dribbled by the opponent robots.

6.2 Intercepting the ball near the border lines

The predictive control evidenced an implementation issue regarding intercepting the ball near the border lines without putting it out of the field. In this case it is not desirable to make the robot move directly to the ball since it might award a set-piece to the opponent team.

First of all, the robot must be able to identify when it shouldn't move directly to the ball.

This is done in the method

```
bool isBallInDangerZone(int agentIdx=-1);
```

The implementation used before this thesis evaluated solely the robot and ball positions inside the field [43]. Specifically, the ball was considered in danger when it was closer to the side lines than a given distance and when the ball was closer to the side lines than the robot. With this implementation, a situation where the ball was near the side lines but moving towards the robot that was in the center of the field would be identified as a dangerous situation. As a result, the robot would avoid the ball which could have a disastrous outcome if the ball was being dribbled by an opponent.

Therefore, an implementation that also considers the ball velocity was developed. On the other hand the robot position is not evaluated for the detection of a dangerous situation.

Two different situations are checked. The first considers a ball with speed. Thus a dangerous situation is detected if after a few seconds the ball will be outside the field. The second situation considers a stopped ball. If the ball is very close to the field lines, the robot might drag the ball outside the field before being able to immobilize itself. Hence the robot checks if the ball is inside a danger zone, which is close to the side lines. If the condition is true then a dangerous situation is flagged to the decision layer. An exception was made to cope with the situation where the ball would move out of the field by the opponent's goal. After all this is the game's objective and therefore the robot does not consider this situation as a dangerous one.

The differences between the old version and the new implementation are detailed in Fig 6.6. The dashed line represents the danger zone considered when the ball is stopped. Notice that the danger zone does not consider the opponent goal as a dangerous one.

This figure shows four distinct situations. In situation **A** the ball is not moving outside the field however the old version identified this as a dangerous situation. In situation **B** the old version did not identified this situation as dangerous since the robot was closer to the side lines than the ball. However the robot would most likely not catch the ball before it was outside the field since the robot was moving towards the estimated ball position instead of moving towards a point in the ball path. The situation **C** is identified as dangerous in the developed implementation if after a few seconds the ball is outside the field even if the ball is not inside the danger zone. This did not happen before as the ball had to be inside the danger zone while the ball velocity was ignored. In the new implementation the situations **D** and **E** are identical, which didn't happen in the old implementation, since the

robot position is not accounted for in the identification of any of the previously described situations.

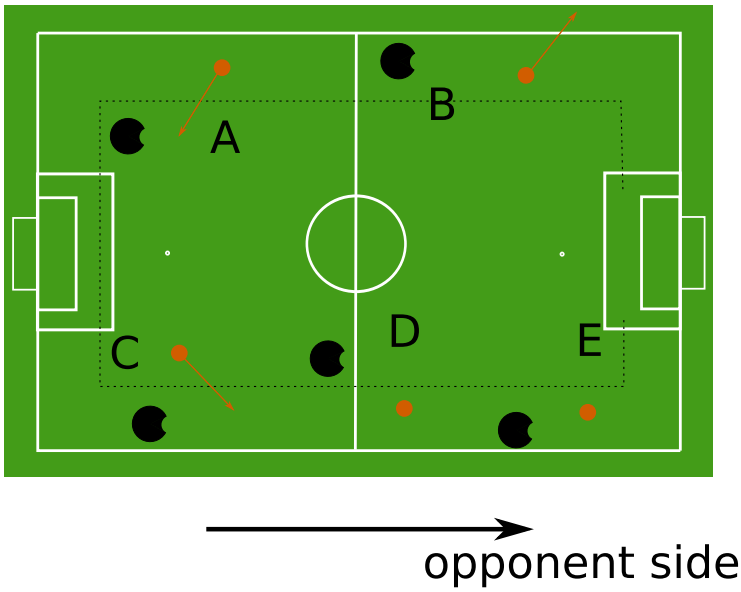


Figure 6.6: *isBallInDangerZone* verifications.

After the proposed method detects a dangerous situation the robot must act accordingly, to prevent the ball from moving outside the field.

This is accomplished using a separate method:

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

In this case there was also the need to refine the previous implementation. Concretely, the robot considered an optimal point to approach the ball without putting it outside the field and would try to rotate around the ball towards the optimal point. This optimal point was the opponent's goal, thus the robot would engage the ball minimizing its pose adjustment in order to shoot to goal. This method worked fine if the ball was stopped or moving away from the opponent's goal. However if the ball was moving to the opponents side of the field, besides not moving through the shortest path possible the robot would position itself towards a point where the ball will never pass.

The proposed method is based on the previous implementation since it works for some situations. The idea to rotate around the ball was kept and if the ball is stopped the robot approaches the ball facing the opposite penalty marker to minimize the time it would take to rotate to the goal and shoot or to minimize the rotation when in a dead angle near the opponent's corners marks. However if the ball is moving, the robot will rotate around the ball until it reaches a point in the ball's path. This means the optimal point is chosen based on the ball velocity.

The previous implementation never explicitly ordered the robot to move to the ball. This was implicitly accomplished since when the robot rotated around the ball it would eventually be nearer the side line than the ball. In this situation the ball would no longer be in danger and the robot would target the ball. However the robot will move to the ball's estimated position. Since the ball is moving towards the border line, it will be in danger before the robot will be able to intercept it. The robot will commute between rotating around the ball and moving towards it repeatedly until the ball moves outside the field.

Therefore the developed method must order the robot to move to the ball. In order not to attack the ball too soon neither too late, the algorithm will only attack the ball when inside an arc pointing to the ball path with the ball position as its center.

The changes between the two implementations are described in Fig 6.7.

The situation **A** represents the old implementation. The situation **B** represents the same situation as **A** using the new implementation. Notice the change in the optimal point which points the robot to the opponent goal when far from it and points the robot to the own side field when engaging the ball near the opponent side line since the robot doesn't shoot to goal in that zone as is considered a dead angle shot. Also notice the arc in the optimal point. If the robot enters that arc it will start moving to the ball. In situation **C** the optimal points changes as the ball is no longer stopped. If the ball is moving in danger zone the optimal approach point is in the ball path increasing the chances of the robot engaging the ball.

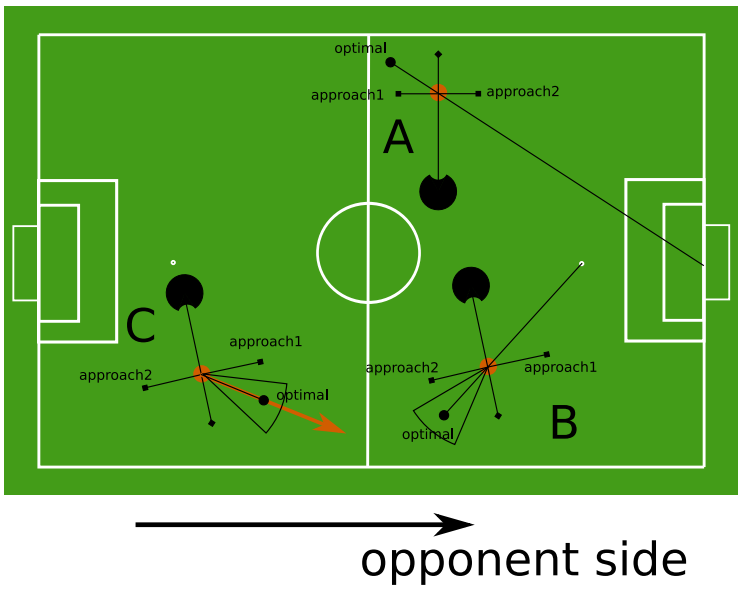


Figure 6.7: *getApproachPoint* implementation.

6.3 Summary

This chapter presents the developed work at the CAMBADA behaviors level. The *ActiveInterception* implementation was presented and proved to be very effective as the robots can now engage the ball in its path. Aside from the ball engagement improvement, the calculation method should also enhance the team defensive formation by assigning the role *Striker* to the robot that will intercept the ball sooner.

The implementation of new methods to prevent the ball from moving outside the field were also discussed which lead to a better robot performance in these situations.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

This thesis objective was to improve the CAMBADA robots behaviours. In order to achieve that goal an accurate holonomic control was required, improving both the path the robot describes as well as the time to complete the described path. A predictive motion control was implemented where the robot pose and ball position are calculated when the last issued command will cease to have effects. Therefore all commands issued are delay-free. An acceleration filter was implemented which prioritizes the desired velocity direction over its norm.

Given previous work done on the estimation of the ball velocity, the ability to intercept the ball, considering the its velocity offers tremendous advantages over moving directly to the ball.

At the behaviours level, a reliable calculation of the point the robot can intercept the ball was developed, allowing the team to regain possession of the ball faster and to raise difficulties to the opponent robots on their dribbles.

Also to regain ball possession faster and to avoid awarding throw-ins, goal kicks or corners to the opponent team, methods to identify and act accordingly to these situations were developed.

In order for the decision layer to choose the right action a precise worldstate that reliably represents the state of the soccer game is needed. Therefore, additional information was supplied in the form of the identification of stuck situations.

The ultimate objective of this thesis was to improve the robot overall performance through all the developed work. Table 7.1 shows the results of the CAMBADA team in several national competitions. Notice in particular the results obtained in the last Robótica national championship where the CAMBADA team won for the third consecutive time with an impressive average of 12.8 goals scored per game and didn't concede any goal throughout the entire tournament. In fact the, the first time the predictive control and the *ActiveInterception* behaviour were used in a real game situation was in the second half of

	Robótica'05	Robótica'06	Robótica'07	Robótica'08	Robótica'09
Goals scored	0.3	3.5	2.7	10.5	12.8
Goals conceded	3.0	1.7	0.5	0.3	0.0

Table 7.1: CAMBADA results in national competitions

the game against the 5DPO team from FEUP[1] in this tournament, where nine goals where scored against the seven goals scored in the first half. The first time these innovations were used in an entire game the CAMBADA record for the number of goals scored in a match was beaten and the game against ISocRob from IST[2], ended 22-0. The previous record was 18-0.

The methods for accurately engaging the ball near the side lines and the stuck detection were developed after the Robotica'09 and were not tested in game situations before the RoboCup'09 World Championship. Although the robot doesn't reacts when stuck situations are identified, the algorithm was still used since the output was redirected to the team basestation where the reliability of the method was tested. Although the robot was able to prevent some balls from leaving through the sidelines and this method was praised by other teams who don't perform this action yet, there is still room for improvement to make this method more reliable.

Although CAMBADA was unable to revalidate the World Champion title, it proved its worth by finishing third in RoboCup'09. This result is remarkable considering the hardware superiority of the 1. RFC Stuttgart and Tech United robots, first and second placed teams, respectively. Also, CAMBADA was the only team capable of imposing a loss to the current World Champions, 1. RFC Stuttgart, with and impressive 5-2 win in a match of the second round robin.

Part of the developed work described in this thesis lead to the publishing of the article **Predictive Control for Behavior Generation of Omni-Directional Robots** in the 14th Portuguese Conference on Artificial Intelligence, EPIA, to be held at 12-15 October 2009 in Aveiro, Portugal.

7.2 Future Work

As some problems are solved, new challenges keep arising in robotics in general and in the MSL in particular.

In order for the CAMBADA team to thrive in the coming years improvements should be continuously made.

The CAMBADA behaviours should be restructured as some behaviours were rendered useless by hardware changes, such as the behaviour *RotateToGoal*, which was used to search the coloured goals before the hyperbolic mirror was installed.

[1]Faculdade de Engenharia da Universidade do Porto

[2]Instituto Superior Técnico

The development of the *ActiveInterception* behaviour also rendered the *PassiveInteception* and *Move* behaviours useless. The creation of a behavior *MoveToBall* is suggested, that should use the methods to calculate the interception point or the approach point wether the ball is in danger of moving out of the field or not. This way it would seem transparent to the decision layer that the robot is moving to capture the ball when the ball is in danger of leaving through the side lines or when the ball is in the middle of the field as it is transparent to a soccer player in a real game. However these situations are currently discriminated at the *Role* layer.

The behaviour *Unstuck* should also be implemented and integrated in the finite-state machines of the existing roles, benefiting from a reliable identification of stuck situations.

The CAMBADA team is aiming for passes in game situations other than set-plays. To accomplish this goal, the behaviour *CatchBall* should be adapted to cover these situations since it was tuned solely for receiving the ball in set-play conditions.

However the CAMBADA defense should not be disregarded, as in some cases the team performance could benefit from having more than one robot moving to the ball, adding an explicit cooperation feature to the defenders.

At the sensor and information fusion level, the robots should be able to identify the opponent goalkeeper position, and select the correct side of the goal therefore improving the kicking accuracy.

For an accurate representation of the robot worldstate the 3D detection of the ball must be achieved which should improve the team performance, especially the goalkeeper performance in bouncing shots situations.

The robots should also be able to identify the ball possession during the game, since a free rolling ball should be treated differently than an opponent dribbled ball. The robots could also benefit from estimating the opponent velocity, which means the opponents have to be identified and tracked during the game.

The development of a simulator would greatly assist the development of future innovations since it withdraws the need for a real official size test field, opponent robots and the integrity of the robot hardware is preserved.

In the control level, and for a reliable simulator, the models of the robots should be determined including the wheel attrition, the robot top speed, different hardware components and software delays, etc.

Bibliography

[1] Lobster robot, July 2009. http://www.onr.navy.mil/media/images/gallery/hires/robots/051206-N-7676W-082a.jpg.

[2] Robocup official site. www.robocup.org.

[3] MSL Technical Committee 1997-2009. Middle size robot league rules and regulations for 2009, (2009).

[4] G. Campion, G. Bastin, and B. D'Andréa-Novel. Structural properties and classification of kinematic and dynamic models of wheeled mobile robots. In *IEEE Transactions on Robotics and Automation*, volume 12, pages 47–62, February 1996.

[5] Orientable wheels, July 2009. http://www.masterjack.co.za/images/Caster_Wheel.jpg.

[6] André Gustavo Scolari Conceição. *Controlo e Cooperação de Robôs Móveis Autónomos Omnidireccionais*. PhD thesis, Departamento de Engenharia Electrotécnica e de Computadores, Faculdade de Engenharia, Universidade do Porto, October 2007.

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

[8] Honda's asimo robot, July 2009. http://news.cnet.com/i/bto/20071116/Asimo_270x477.jpg.

[9] Nimbro's humanoid robot, July 2009. http://www.ais.uni-bonn.de/nimbro/images/robots/BioloidKicking.jpg.

[10] Aldebaran's nao humanoid robot, July 2009. http://www.aldebaran-robotics.com/eng/images/NaoRC03.jpg.

[11] Snake robot, July 2009. http://hackedgadgets.com/wp-content/snake.jpg.

[12] Robolobster, July 2009. http://www.onr.navy.mil/media/images/gallery/hires/robots/051206-N-7676W-082a.jpg.

[13] Microbat robot, July 2009. http://www.reallycooltoys.com/news/news2.html.

[14] M.Gholipour, S.Ebrahimijam, H.Rasam Fard, M.Montazeri, A.Mohseni, S.Moein, A.Zaeri, H.Hosseini, M.Yekkefallah, S.Sajjadi, and B.Eskandariun. Mrl middle size team: 2009 team description paper, 2008.

[15] O. Zweigle, U.-P. Käppeler, H. Rajaie, K. Häussermann, A. Tamke, A. Koch, B. Eckstein, F. Aichele, and P. Levi. 1. rfc stuttgart team description 2009, 2008.

[16] Hui Zhang, Xiangke Wang, Huimin Lu, Shaowu Yang, Shengcai Lu, Junhao Xiao, Fangyi Sun, Dan Hai, and Zhiqiang Zheng. Nubot team description paper 2009, 2008.

[17] J. Almeida, A. Martins, E. Silva, L. Lima, C. Almeida, N. Dias, A. Dias, and H. Silva. Iseporto robotic soccer team for robocup 2009:improving perception., 2008.

[18] Q. Sharifi, s.A. Monadjemi, s.H. Kasaee, M. Taheri, s.M. Kasaee, H. Vahiddastjerdi, M. Rahimi, A. Abdolahi, and M. Ghobadi. Team description paper of adro 2009, 2008.

[19] Yuichi Kitazumi, Shuichi Ishida, Yu Ogawa, Kota Yamada, Yusuke Sato, Mariko Oki, Hiroshi Thoriyama, Noriyuki Shinpuku, Yasunori Takemura, Amir A.F. Nassiraei, Ivan Godler, Kazuo Ishii, and Hiroyuki Miyamoto. Hibikino-musashi team description paper, 2008.

[20] A. Bonarini, A. Furlan, L. Malagò, D. Marzorati, M. Matteucci, D. Migliore, M. Restelli, and D. G. Sorrenti. Milan robocup team 2009, 2008.

[21] Bernd Kleinjohann, Philipp Adelt, Willi Richert, and Claudius Stern. The paderkicker team robocup 2009, 2008.

[22] Zhang Shunxin, Lin Xiaoyuan, Fan Haiting, Xiang Zongjie, and Chen Wanmi. Shu strive legends team description 2009, 2008.

[23] W.H.T.M. Aangenent, J.J.T.H. de Best, B.H.M. Bukkems, F.M.W. Kanters, K.J. Meessen, J.J.P.A Willems, R.J.E. Merry, and M.J.G. v.d. Molengraft. Tech united eindhoven team description 2009, 2008.

[24] Roland Hafner, Sascha Lange, Martin Riedmiller, and Stefan Welker. Brainstormers tribots team description, 2008.

[25] 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.

[26] L. Almeida, J.L. Azevedo, P. Bartolomeu, E. Brito, M.B. Cunha, J.P. Figueiredo, P. Fonseca, C. Lima, R. Marau, N. Lau, P. Pedreiras, A. Pereira, A. Pinho, F. Santos, L. Seabra Lopes, and J. Vieira. Cambada: Team description paper, 2004.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] N. Lau, L.S. Lopes, and G. Corrente. Cambada: Information sharing and team coordination. In *Proc. of the 8th Conference on Autonomous Robot Systems and Competitions, Portuguese Robotics Open - ROBOTICA'2008*, pages 27–32, Aveiro, Portugal, April 2008.

[34] 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.

[35] 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.

[36] M. Oubbati, M. Schanz, T. Buchheim, and P. Levi. Velocity control of an omnidirectional robocup player with recurrent neural networks. In *RoboCup 2005: Robot Soccer World Cup IX*, volume 4020 of *Lecture Notes in Computer Science*, pages 691–701. Springer Berlin / Heidelberg, 2006.

[37] R. Hafner, Lange S., M. Lauer, and M. Riedmiller. Brainstormers tribots team description. RoboCup International Symposium 2008, CD Proc., Suzhou, China.

[38] Y. Sato, S. Yamaguchi, and et al. Hibikino-musashi team description paper. RoboCup International Symposium 2008, CD Proc., Suzhou, China.

[39] Ethercat robots win german open. Press Release, EtherCAT Technology Group, 8 May 2008.

[40] S. Behnke, A. Egorova, A. Gloye, R. Rojas, and M. Simon. Predicting away robot control latency. In *RoboCup 2003: Robot Soccer World Cup VII*, volume 3020 of *Lecture Notes in Computer Science*, pages 712–719. Springer Berlin / Heidelberg, 2004.

[41] P.-G. Plöger, G. Indiveri, and J. Paulus. Motion control of swedish wheeled mobile robots in the presence of actuator saturation. In *RoboCup 2006: Robot Soccer World Cup X*, volume 4434 of *Lecture Notes in Computer Science*, pages 35–46. Springer Berlin / Heidelberg, 2007.

[42] M Lauer. Ego-motion estimation and collision detection for omnidirectional robots. In *RoboCup 2006: Robot Soccer World Cup X*, volume 4434 of *Lecture Notes in Computer Science*, pages 466–473. Springer Berlin / Heidelberg, 2007.

[43] João Manuel Leite da Silva. Sensor fusion and behaviours for the cambada robotic soccer team. Master's thesis, University of Aveiro, 2008.

[44] Haptic definition, June 2009. dictionary.reference.com/browse/haptic.

[45] Somatosensory system, June 2009. http://en.wikipedia.org/wiki/Somatosensory_system.

[46] Stasis, June 2009. http://en.wikipedia.org/wiki/Stasis.

[47] Stasis logic, June 2009. www.schursastrophotography.com/robotics/stasislogic.html.

[48] G. Canepa, J.M. Hollerbach, and A.J.M. Boelen. Kinematic calibration by means of a triaxial accelerometer. In *ICRA-1994, the International Conference on Robotics and Automation*, volume 4, pages 2776–2782, May 1994.

[49] D. Vail and M. Veloso. Learning from accelerometer data on a legged robot. In *In Proceedings of the 5th IFAC/EURON Symposium on Intelligent Autonomous Vehicles*, July 2004.

[50] J. Baltes and J. Anderson. A pragmatic approach to robot rescue: The keystone fire brigade. In *Proc. AAAI-02 Mobile Robot Competition Workshop*, pages 38–43, July 2002.