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

Rafael Filipe dos Reis Castro

Desenvolvimento de um guarda-redes para a liga de robôs médios

Development of a goalkeeper for the Middle Size League

Rafael Filipe dos Reis Castro

Desenvolvimento de um guarda-redes para a Liga de Robôs Médios

Development of a goalkeeper for the Middle Size League

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requesitos 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 José Luís Azevedo, Professores do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro

o júri / the jury

presidente / president

Professor Doutor Artur José Carneiro Pereira
Professor Auxiliar da Universidade de Aveiro

vogais / examiners committee

Professor Doutor Paulo José Cerqueira Gomes da Costa
Professor Auxiliar da Universidade do Porto - Faculdade de Engenharia

Professor Doutor José Nuno Panelas Nunes Lau
Professor Auxiliar da Universidade de Aveiro (orientador)

agradecimentos / acknowledgements

Quero agradecer a todos os elementos da CAMBADA pela disponibilidade para discutir ideias e pelo apoio na execução desta tese, pelo bom ambiente e momentos passados no IRIS LAB que permitiram que as fases mais difíceis do trabalho fossem mais fáceis de superar. Agradeço especialmente ao João e ao Ricardo pelas dicas, ideias e orientações que me deram durante estes meses de trabalho.

Um especial agradecimento aos meus orientadores Professor Nuno Lau e Professor José Luís Azevedo por estarem sempre disponíveis para me esclarecerem as dúvidas e por terem acompanhado o meu trabalho semanalmente de forma exemplar. Foram uma ajuda fundamental principalmente no que diz respeito à execução deste documento.

Um grande obrigado aos meus amigos mais próximos e colegas de curso, sem vocês estes cinco anos teriam sido muito mais difíceis. Ao meu irmão que esteve sempre presente quando precisei. À Catarina por me ter ajudado a encontrar o caminho quando estava mais perdido mesmo que isso implicasse um sermão de duas horas, obrigado pela paciência, pelos conselhos e pelo apoio que me deste.

Por fim, quero agradecer aos meus pais pelo esforço que fizeram para que eu pudesse frequentar a universidade e também pelo apoio incondicional. Sem vocês eu não teria conseguido. Obrigado.

Resumo

CAMBADA (Cooperative Autonomous Mobile roBots with Advanced Distributed Architecture) é a equipa de futebol robótico da Universidade de Aveiro que participa na RoboCup Middle Size League. Esta equipa foi criada por investigadores das unidades de investigação IEETA e ATRI. A robótica é uma área em constante evolução mas o trabalho desenvolvido no âmbito desta tese dificilmente vai ser ultrapassado mesmo com o constante avanço desta área, pois o objetivo fundamental do Guarda-Redes vai continuar sempre o mesmo: não sofrer golos. Nesta dissertação são apresentadas contribuições no comportamento de alto nível do Guarda-Redes da equipa CAMBADA, no entanto nem tudo se aplica exclusivamente ao Guarda-Redes, sendo importante para qualquer jogador de campo e podendo ser aplicável num futuro próximo. Neste trabalho são abordados dois grandes tópicos, o primeiro é o posicionamento do guarda-redes em campo que se estende aos posicionamentos em jogo-corrido, encostado aos postes, relativo à baliza e a uma defesa mais agressiva com saídas ao jogador com bola. O outro tópico abordado é a capacidade de deteção e defesa de remates pelo ar. A configuração do Guarda-Redes de uma forma simples foi um dos objetivos neste trabalho, de forma a evitar ao máximo que a sua ação seja demasiado estática, podendo facilmente tirar proveito desta fácil configuração consoante os adversários da equipa CAMBADA. Foi desenvolvida uma interface gráfica a fim de controlar estas configurações.

O trabalho desenvolvido foi testado em laboratório diferenciando-se bastante do antigo comportamento que era muito defensivo e cujo o raio de ação era muito próximo da linha de golo, para um comportamento mais imprevisível, mais aproximado a um Guarda-Redes humano ao pressionar o atacante adversário dispondo de um maior leque de opções defensivas.

Abstract

CAMBADA (Cooperative Autonomous Mobile Robots with Advanced Distributed Architecture) is the robotic soccer team of the University of Aveiro that participates in RoboCup Middle Size League. This team was created by researchers from research units IEETA and ATRI. Robotics is an area in constant evolution but the work developed within this thesis will hardly be overcome even with the constant progress in this area, since the primary goal of the goalkeeper will always remain the same: not conceding goals.

Contributions presented in this dissertation are in terms of high-level behavior of the goalkeeper of the team CAMBADA , however not everything applies exclusively to the goalkeeper, it is important for any field player and can be applicable in the near future. This work addressed two main topics, the first one is the positioning of the goalkeeper on the field that extends to the run-game placements, near the goal posts placement and a more aggressive defense behavior with pressure to the player with the ball. The second topic addressed is the ability of detection and defend shots by air. The configuration of the goalkeeper in a simple way has been one of the objectives of this study, to avoid at all costs that his actions are too static and can easily take advantage of this easy setup to change the goalkeeper behavior depending on the opponents of CAMBADA. A graphical user interface to control these settings was developed.

The work has been tested in the laboratory differing greatly from the previous behavior that was very defensive and whose radius of action was very close to the goal line for a more unpredictable, closer to an human Goalkeeper behavior that applying pressure to the opponent attacker feature a wider range of defensive options.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

In the recent years Robotics is an area that is in constant evolution that propose new objectives every day. Everyone loves to see a robot playing soccer without human interaction and most of them wonder how it is possible, seems that the robots have a brain just like human have. One of the main reasons to choose this thesis was to enhance the knowledge about robotics in general and get together robotic and the sport competition environment, especially soccer, makes people interest and develop more because there is a final aim every year, participate and try to win the competition. The fact that CAMBADA (Cooperative Autonomous Mobile Robots with Advanced Distributed Architecture) the robotic soccer team of the University of Aveiro participates in RoboCup Middle Size League, an annual international soccer competition also had huge impact in terms of motivation about this thesis. Why develop the goalkeeper then? As said before robotics is in constant evolution and the old goalkeeper was already developed three years ago, so it was an important topic to approach inside the CAMBADA team. It was the perfect time to improve it and help this team getting stronger. The fact that the previous goalkeeper was so old already means that it would need an huge improvements and there were so many interesting issues to improve. The goalkeeper is an important piece on a real soccer team and that importance is even greater on robotic soccer. It is an key player if a team want to have success. This is almost an impossible task to do with the previous goalkeeper and there were already in the league so many teams with a goalkeeper better than the CAMBADA and if this team wants to be again on the track of success the goalkeeper is a good place to start. It is still impossible to lose without conceding goals. These were the main reasons of choosing this

thesis.

1.2 Objectives

It is the objective of this project to develop the software of the CAMBADA goalkeeper. There are some topics to approach in order to develop this goalkeeper such as: improve the global positioning of the Goalkeeper taking the best use of his good positioning to defend the most shots on goal, use the full potential of the goalkeeper bounds, enhance the goalkeeper localization on the field, particularly its localization relative to the goal, equip the goalkeeper with a way to detect aerial balls because until now all the robots of CAMBADA team only had a 2D (2 dimension) vision mechanism, this also an important goal of this thesis because most of the shots are aerial shots and was impossible to keep tracking the ball when it was above ground therefore mostly of the goals allowed were from aerial shots.

1.3 Thesis Structure

In the chapter 2 there is an overview on topics of CAMBADA that in some way had impact or are important on this thesis. It is also given an introduction to the RoboCup, the organization and its annual event, given more attention to the Middle-Size League (MSL) competition and important rules for the goalkeeper, where this thesis is inserted.

Following, in chapter 3 there is presented work developed on the goalkeeper 3D vision mechanism by other MSL league teams, Tech United, NuBot and Hibikino Musashi. Is also presented real human goalkeeper defensive positioning behaviors.

In the chapter 4, it is shown the work done in the positioning of the goalkeeper. It is also described the new `roleGoalie` and the three new behaviors developed in this thesis, `BGoaliePositioning`, `BGoalieInsideArea` and `BGoalieStandStill`. After the behaviors the graphical user interface developed to configure the goalkeeper is presented. In the end of the chapter is exhibited the results of the work developed under this chapter comparing with the previous goalkeeper.

In the chapter 5, is made an introduction to some Kinect specifications that were needed to take into account. It is also explained the algorithm to detect flying object that was created before this thesis started as well as the modifications made on the algorithm. This chapter ends with experimental results about the parametrization and precision of the

improved algorithm.

In the last chapter of this thesis, chapter 6 is made a conclusion on all the work developed on this thesis and also presented some work that can be done in the future to improve even more the goalkeeper.

Chapter 2

CAMBADA

Cooperative Autonomous Mobile roBots with Advanced Distributed Architecture, usually know as CAMBADA [5] is a Portuguese soccer team from University of Aveiro that participates in the RoboCup Middle Size League (MSL). This project started in October 2003 and it was coordinated by the IEETA [6]. In the figure 2.1 can be seen the actual CAMBADA robots. Since its creation, CAMBADA has participated in several competitions both national and international. It is a team that revealed an impressive development participating every year since 2004 in Robótica, an annual national festival of robotics, competition that was won by CAMBADA for six consecutive years 2007-2012. Regarding international competitions, this team participated every year except for 2005 on the RoboCup, an annual international robotics competition, counting with an impressive win in 2008 and five 3rd places already. These results can be check on the Table 2.1.

Year	**Location**	**Result**
2004	Lisbon, Portugal	5th Place
2006	Bremen, Germany	1st Round
2007	Atlanta, USA	5th Place
2008	Suzhou, China	1st Place
2009	Graz, Austria	3rd Place
2010	Singapore	3rd Place
2011	Istanbul, Turkey	3rd Place
2012	Mexico City, Mexico	4th Place
2013	Eindhoven, Netherlands	3rd Place
2014	João Pessoa, Brazil	3rd Place

Table 2.1: CAMBADA Participation on RoboCup.

Figure 2.1: CAMBADA Team Robots.

2.1 RoboCup

RoboCup [7] is an international project that was created to promote the research and innovations in robotics, artificial intelligence and related areas by setting a long term goal. The Dream besides RoboCup is "By the middle of the 21st century, a team of fully autonomous humanoid robot soccer players shall win a soccer game, complying with the official rules of FIFA, against the winner of the most recent World Cup."[8] RoboCup is also the name of an annual competition that is held annually since 1997 in different countries. This competition is the most important annual meeting and keeps the teams motivated to continue pursuing the year 2050 dream. In this event teams also have the opportunity to show and discuss relevant and important improvements since the previous year. To make this goal possible RoboCup is divided in different leagues:

RoboCup Junior:

"RoboCupJunior is a project-oriented educational initiative that sponsors local, regional and international robotic events for young students. It is designed to introduce RoboCup to primary and secondary school children"[9]

RoboCup @Home

"The RoboCup@Home league aims to develop service and robot technology with high relevance for future personal domestic applications. It is the largest international annual competition for autonomous service robots and is part of the RoboCup initiative"[10]

RoboCup Rescue

"The intention of the RoboCupRescue project is to promote research and development in this socially significant domain at various levels involving multi-agent team work coordination, physical robotic agents for search and rescue, information infrastructures, personal digital assistants, a standard simulator and decision support systems, evaluation benchmarks for rescue strategies and robotic systems that are all integrated into a comprehensive systems in future"[11]

RoboCup Soccer

"The main focus of the RoboCup competitions is the game of football/ soccer, where the research goals concern cooperative multi-robot and multi-agent systems in dynamic adversarial environments. All robots in this league are fully autonomous". [12] RoboCup soccer is divided in this leagues:

- Humanoid League
- Simulation League
- Standard Platform League
- Small Size League
- Middle Size League

2.1.1 Middle Size League (MSL)

CAMBADA robots were built to compete on RoboCup's Middle Size League, an official league of RoboCup where two teams made up of five autonomous robots, that must fit in a 52cm x 52cm x 80cm box and a maximum weight of 40 Kg, who play on a field similar to a football field, but smaller in size (18 m x 12 m), figure 2.3. Robots have all the sensors

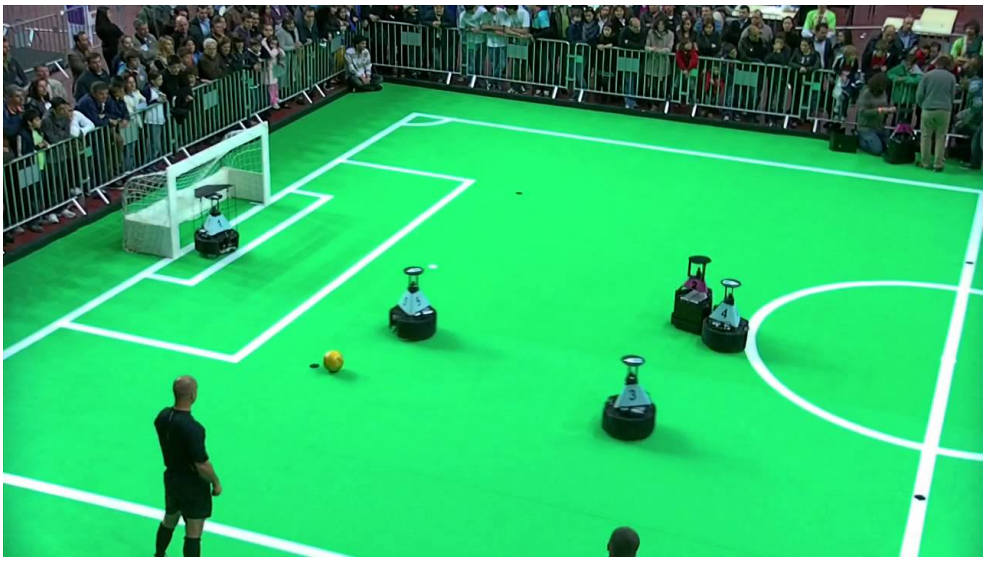


Figure 2.2: RoboCup Middle Size League (MSL) game.

and actuators on-board, which enables them to play completely autonomously, without any human intervention. The only human interaction allowed is the referee (yes, it is still human) who can remove the robots off the field. The robots have wireless connection in order to communicate among themselves and with an external computer acting as a coach. This coach computer has no sensor of any kind, it only knows what is reported by the playing robots, who should be able to evaluate the state of the world and make their own decisions, that must fulfill the team objective.

The environment in which the robots play is well defined, green field, white lines and mainly black robots. The game ball is a generic FIFA approved ball. The rules [1] of the game are similar to the official FIFA rules, with some required changes to enable the robots to play. I will explain only two important rule for my thesis, one is the fact that the keeper can temporarily expand its dimensions to 62 cm x 62 cm x 80 cm or 52 cm x 52 cm x 90 cm, but only for 1 second, if the goal is endangered by an approaching ball. After go back to the normal dimensions, he has to wait 4 seconds before he is allowed to expand again. The other important rule is whether a goal is valid or not, the goal is only valid if the robot that made the kick is inside the opponent side of the field (attacking half of the field), however this does not apply if the robot shot on his own goal. Also, the robot cannot receive the ball on his own half of the field and move to the opponent half of the field and score, in order to the goal to be valid the ball needs to be received on the attacking field assuming that the ball has rolled freely for at least one meter before it was

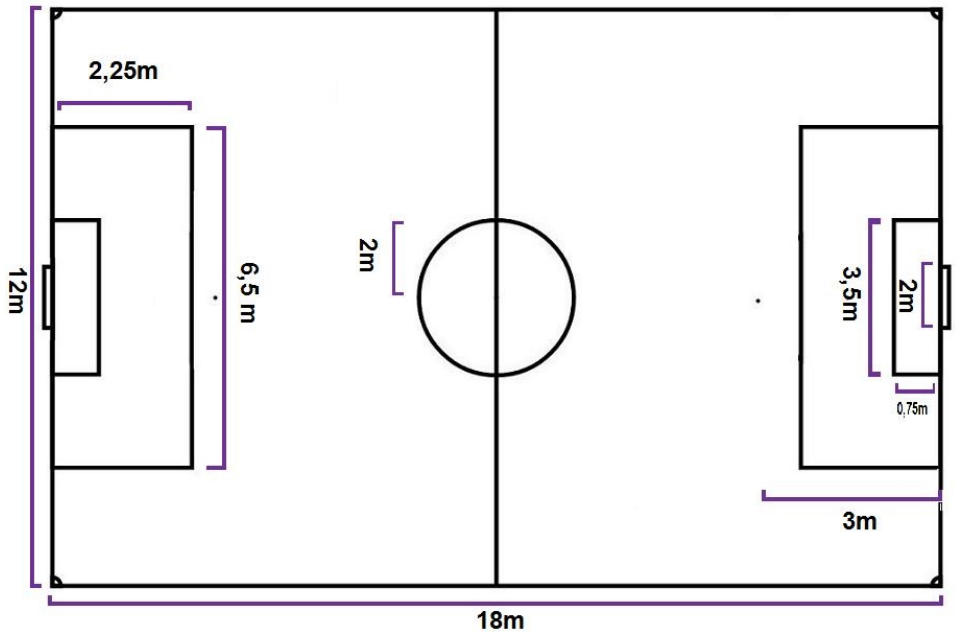


Figure 2.3: MSL field dimensions according to the 2014 rules.

received by the robot (in case of a team pass). However if the robot steals the ball from an opponent already in the attacking field he is able to shoot and score. A scheme about the valid ways to score a goal can be seen on the figure 2.4.

This particular league attracts great public attention. Developing a team of robots of this size requires great mechanical and electronic challenges. On the other hand, autonomy restriction sets its own challenges in signal processing, control and artificial intelligence. The competitive scenario is regularly reviewed to ensure the challenge is adequate to the state of technology.

2.2 General Architecture

In order to start explaining how the CAMBADA goalkeeper works is needed a brief description on how CAMBADA code is structured. The general architecture [13] is based on a main processing unit (a PC with the Linux operating system) that is responsible for the higher-level behavior coordination (the coordination layer). This unit controls the processing of visual information acquired by the vision system, the execution of high-level control functions, is responsible for receiving sensing information and sending actuating commands to control the robot (low level control layer) and the handling of external communication

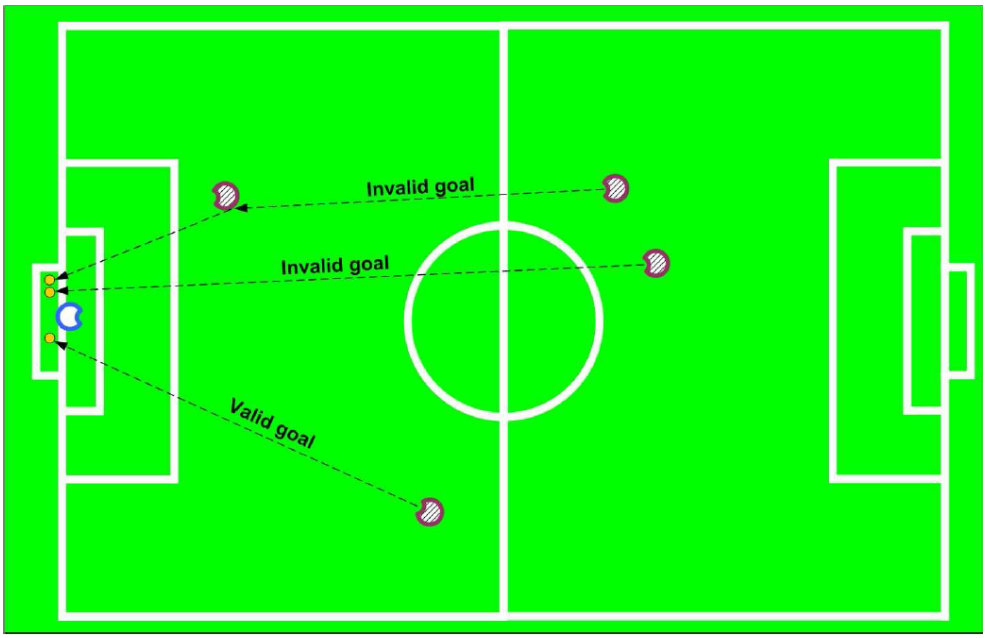


Figure 2.4: MSL valid methods of scoring, adapted from MSL Rules [1].

with the other robots via Wi-Fi IEEE 802.11a or 802.11b standards. This communication is supported by a middle-ware, the **R**eal**t**ime **D**ata **B**ase (RtDB) [14] that provides access to a distributed database to all the team members (robots). The low-level control layer is responsible for four main functions, control the velocity of the motors, control the kicking mechanism, take care of the ball handling and inertial related measures. An overview of the main blocks of this architecture [2] is illustrated in Figure 2.5.

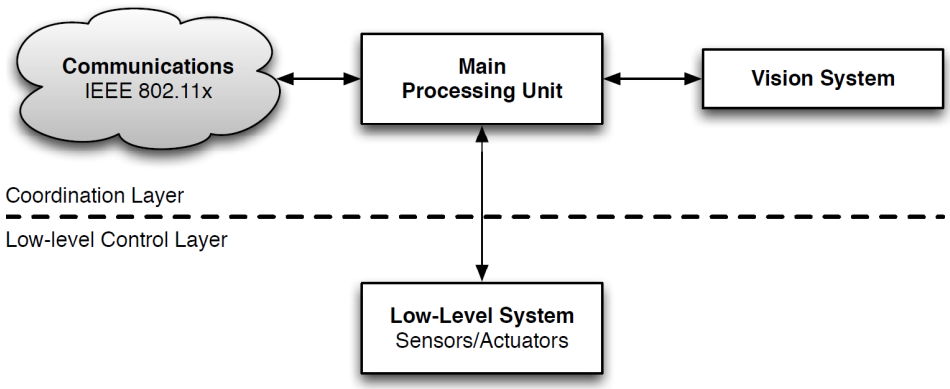


Figure 2.5: Overview of a CAMBADA robot architecture. Image from [2]

2.3 Field Axes

The field is one of the most important subjects to the robots, in order to score a goal that is needed to know is where the robot is on the field, so it can shoot wisely, to defend it also needs to know where to move precisely to defend the opponent team shoots. To make the axes general and easy to use, the axes are always dependent from our goal. So the axes on the field are like this:

- **Origin of the referential**, the center of the field.
- **YY axes**, looking at the opponent goal, this axis will be aligned with the attacking direction,positive in the opponent goal and negative in our goal.
- **XX axes**, again looking at the opponent goal, the XX axis will be negative on the left side of the field and positive on the right side of the field.

The Image 2.6 represents how the axes looks like in the field.

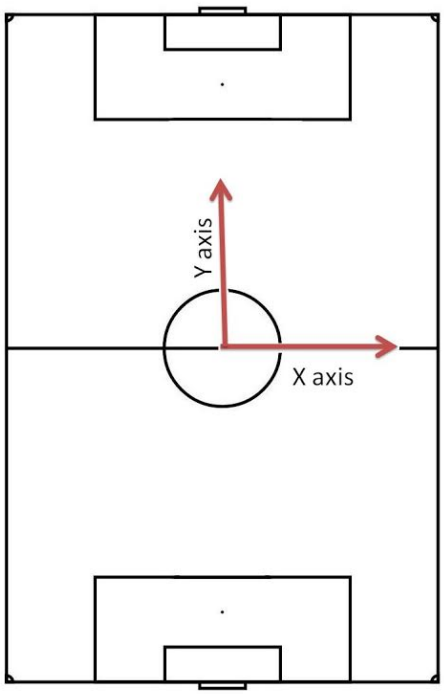


Figure 2.6: CAMBADA axes in the field.

2.4 DriveVector

In CAMBADA there is a transversal module that stores information that needs to be send to the low level actuators, the **DriveVector**. This class contains velocities about the XX and YY axes, **velX** and **velY**, and also the angular velocity, **velA**, but also keeps some data for the kicker and grabber states.

This class has some important functions that were used on this thesis such as the **allOff()** function, that reset all velocity attributes to 0, the kick power is also set to 0 and the grabber is turned off. The other function that was used is **motorsOff()** that only reset the velocity attributes to 0 but doesn't change the kicker and grabber states.

2.5 Coordination and strategy

In CAMBADA each robot is an independent agent, the software agent is the process where all the Artificial Intelligence resides and is responsible for taking high-level decisions, and coordinates its actions with its teammates through communication and information exchange. The resulting behavior of the individual robot should be integrated into the global team strategy, leading to cooperative actions by all the robots. This is achieved by the use of **roles** and **behaviors** [15] that define each robot individual actions in the field.

Each **role** is the task/position that the robot has in the team in a specific moment. We have roles such as `RoleGoalie` (goalkeeper), `RoleMidFielder` or `RoleStriker`, basic positions on the field, but there are more specific roles, as for example `RoleBarrier` to place in the barrier on a free-kick, or `RolePenalty` to score a penalty. On other hand, **Behaviors** are the basic sensorimotor skills of the robot, some examples of behaviors are kicking the ball, search for the ball or pass the ball to a team mate. During a game, except for the goalkeeper, the robots usually swap roles with each other, depending on some factors as the position on the field. Is the set between the "Role" and "Behavior" that define how the robot will play inside the field.

2.6 Behaviors

Behaviors in CAMBADA are complex, it consists on three major parts, whats the purpose of this behavior, and also two types of conditions, the `Invocation Condition` that specifies which conditions will trigger the behavior (when to enter on this behavior)

and the `Commitment Condition` that control the release of the behavior (when to leave of the behavior). Therefore a behavior can be selected if the invocation condition is true and the commitment condition will be true while the behavior is pertinent. Each role also has a hierarchy of behaviors, this way the role will enter on the first behavior that has its invocation condition true and will remain on that behavior until its commitment condition becomes false or until the invocation condition of an higher priority behavior switches to true.

2.7 Previous Goalkeeper

At the start of this work on the CAMBADA Goalkeeper, it was composed by two behaviors and the role goalie itself. The role determined the priority of the behaviors that the goalkeeper could have, the first one has the highest priority and is called **BGoalieGoToGoal**. The invocations of this behavior are based on the position of the goalkeeper on the field, if too far away from the goal it will trigger and enter on this behavior. This is a simple but required behavior since it is important to control the path to the goal to be sure we avoid collisions with other robots or with the goalposts, its only objective is to move the goalkeeper to the middle of the goal. The other behavior that existed was the **BGoalieDefend** where all the Artificial Intelligence of the goalkeeper resides. This behavior calculates the position that the goalkeeper should move to defend the shoots and/or position himself well in the goal.

The pseudo-algorithm of the previous goalkeeper can be seen in the Algoritm 1 but it will be also explained. So it starts with a verification on the ball velocity, if this velocity is too low it should be ignored because the ball will just not reach the goal or we do not have precision to detect if the ball is really stopped or is rolling slowly. If this happens we will set the center of the goal as the final trajectory point. Next it is verified if the robot does not see the ball but the last time he have seen the ball is not more than `timeToWait` milliseconds we will keep the goalkeeper on the same place until we see the ball again or until the `timeOut` expires, if the robot sees the ball the `ballNotVisibleMS` timer will be restarted. Now we have reached a stage when either the robot sees the ball or the ball is not seen more than `timeToWait` milliseconds. So the next thing to do will be calculate the intersection between the ball trajectory and the goal, when this intersection doesn't exists the robot will just align himself with the ball on the XX axes, if the intersection exists the robot will place in the intersection point. The last step done on this algorithm is to check

for how long the ball is not seen, if it is not seen for more than 400ms and less than 1600ms the robot will move himself about the YY axes to the goal line and keeping his position on the XX axes, when the ball is lost for more than 1600ms point to move will just be the middle of the goal. Last but not least, it is made some small adjustments on the position on the XX axes to avoid the goalkeeper to crash with the goal posts and then is calculated the angle from the YY axes between the goalkeeper and the ball. The max angle that is allowed is 20 degrees and the minimum is -20 degrees from Y. Finally the information of the target point and the rotation will be send to the lower level using the `DriveVector` as outPut.

Algorithm 1 Old Goalkeeper algorithm

Input:

Output: DriveVector dv

```
1: if ballVell.length() < 0.4 then
2:    finalBallTrajectPoint = ourGoal.middle()
3: end if
4: if ball.notVisible then
5:    if ballNotVisibleMS.elapsed() < timeToWait then
6:       dv.motorsOff()
7:       return
8:    end if
9: else
10:   ballNotVisibleMS.restart()
11: end if
12: if not intersect(ballTraject, goal) then {Ball is going out}
13:   targetPos.x = ballAbs.x
14: else
15:   targetPos = intersect(ballTraject, goal)
16: end if
17: if 400ms < ballNotVisibleMS.elapsed() < 1600ms then
18:   targetPos.y = ourGoalLine
19:   targetPos.x = myPos.x
20: else if ballNotVisibleMS.elapsed() > 1600ms then
21:   targetPos = ourGoal.middle()
22: end if
23: if targetPos.x > LeftGoalPost then {Crop LeftGoalPost}
24:   targetPos.x = LeftGoalPost - SafeDist
25: end if
26: if targetPos.x < RightGoalPost then {Crop RightGoalPost}
27:   targetPos.x = RightGoalPost - SafeDist
28: end if
29: angleFromY = abs2rel(ballAbs).getAngleFromY()
30: if angleFromY < -20 then
31:   angleFromY = -20
32: end if
33: if angleFromY > 20 then
34:   angleFromY = 20
35: end if
36: dv = move(targetPos,angleFromY)
37: return dv {move to targetPos} l
```

Chapter 3

Goalkeeper techniques

This chapter presents an overview of some techniques actually used by both football goalkeeper and robotic goalkeeper. Most of them are important on the decisions made on this thesis. A special emphasis will be given to the positioning on the field, mostly from human goalkeeper, and also some work that was already developed on other teams of RoboCup MSL will be presented.

3.1 Human Goalkeeper

Goalkeeper positioning is the most important skill that a goalkeeper should have, if it is in the right position, most likely there will be no need to dive or make difficult saves, the good positioning will make it a lot easier to make saves. Even top level goalkeepers position themselves between some reference points and the ball, usually this reference point is the center of the goal, figure 3.1, but not always, depending on the opponent the goalkeeper is against, the reference point can be adjusted. First, to position accurately, we must know where the goal is, although this sounds obvious during a game we can loose the track of the goal very easily, so goalkeepers usually keep looking back to track the posts from time to time [16]. The most common on the goalkeeper is to look always towards the ball so the area that the goalkeeper occupies on the goal is the biggest possible.

Other important technique is to position far enough of the goal line to cover the angle created by the ball and both posts. The goalkeeper should be able to cover either posts within a couple of quick steps. The angle and the keeper's ability determine the forward/backward positioning. The more the distance between you and the goal the more the area is covered for ground shoots [17].

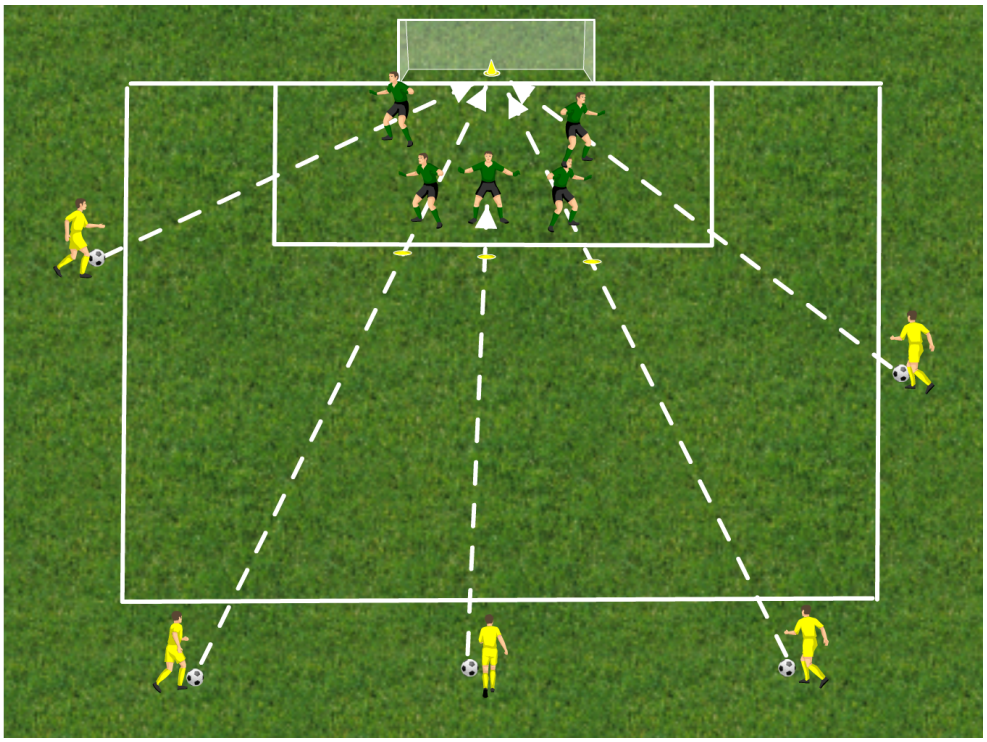


Figure 3.1: Basic Goalkeeper Positioning.

3.2 3-Dimensional vision mechanisms in MSL teams

In order to establish a mechanism that tracks aerial balls it is needed to study what the other teams of the league already have or are trying to build. This section will present the Goalie's Stereo Vision System that `NuBot Team` presented in 2013, the Kinect ball detection that `Tech United` started to use in 2013 and the Fusion of Omni and Monocular camera for the goalie used by `Hibikino Musashi` presented earlier this year (2014).

3.2.1 Kinect ball detection - Tech United

Every field-player of Tech United team is equipped with an omni-vision module, just like almost all the other teams of the league (MSL), that provides a two dimensional image of the surrounding environment. This way their capacity of detecting air balls are very limited, so they took advantage of a depth-based camera, the Kinect camera, to make a three-dimensional representation of the environment. Combining both images provided enough information to distinguish a ball from its surroundings. The Tech United algorithm [18] uses the color to separate a possible ball from the rest of the environment, this method

has one downside, it requires calibration before each game, but `CAMBADA team` already need to do this kind of calibrations for the vision process so its nothing that this team is not used to. This algorithm verifies every possible ball from the RGB camera on the depth camera of the Kinect. So this process works the following way:

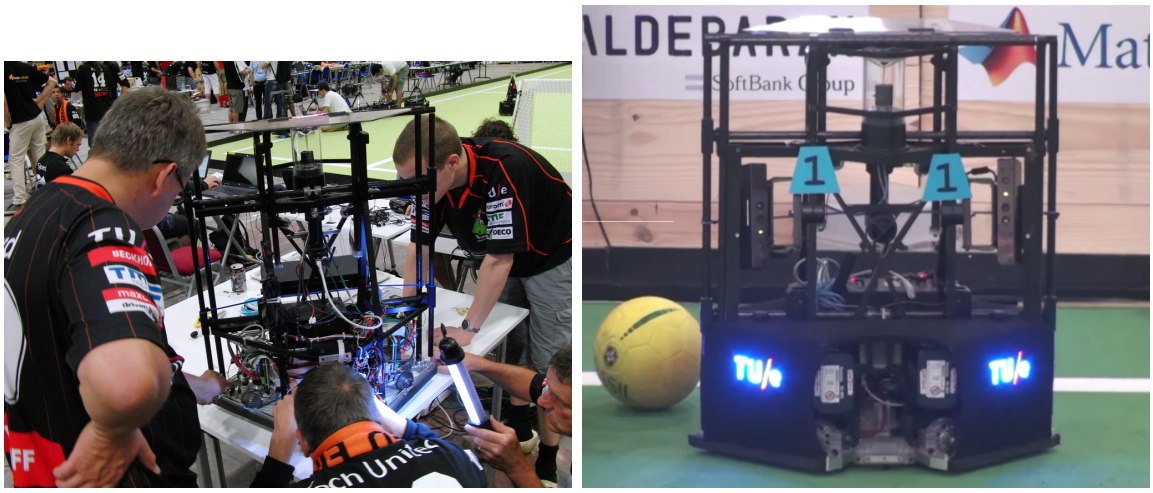
> " Our algorithm uses color segmentation to distinguish a ball from its surroundings, so game color calibration is required. After clustering using a k-means algorithm, a set of possible ball candidates is obtained that can be evaluated using information out of the depth image. We calculate the position of all ball candidates by averaging the position of the individual pixels in each cluster. Using this position, ball candidates outside the field can be discarded. Moreover, the size of the clusters can be compared to the expected size of a ball at this distance, which gives an indication for the reliability of the ball candidate." [19]

Analyzing the `Tech United` goalkeeper in action, it can be said that this method is working pretty well for their team and they are winning many games due to the great exhibitions of the goalkeeper, so this could be a possible method to consider. The fact that mechanically their goalkeeper is way better than almost any other goalkeeper in the league cannot be forgotten, because it moves faster to the sides than frontward/backward which allows him to move where he wants faster due to the fact that they move almost always to the sides. Apart from this, it also has top and side extensors that allow to expand in size for 1 second. However there is a fact that Tech United is a team that generally does not suffer allot of goals and they do not defend that well as a team, they rely a lot on their goalkeeper so this mechanism of 3-Dimensional ball detection needs to work well.

In 2013, they used only one Kinect on the center of the goalkeeper, Figure 3.2(a), but earlier this year they presented the goalkeeper with 2 Kinect, one in the right side of the robot and the other one in the left side, as we can see in figure 3.2(b) this allowed them to have a wider field of view, since the Kinect got a limited field of view.

3.2.2 The Goalie's Stereo Vision System - NuBot Team

`NuBot team` decided to use a Stereo Vision system, a system similar to the human binocular vision, to locate balls in 3-Dimensional spaces. Since most of the shoots in the competition are kicking the ball up from the ground, it is very important for the goalie to locate and estimate the ball motion. This mechanism consists on fitting off the ball's



(a) 2013 (b) 2014

Figure 3.2: Tech United Goalie

trace and predict the touchdown-point, point where the ball will reach the ground again. Based on this prediction information the goalie has more chances of intercept the shooting ball. This is accomplished by calculating and recording, after the ball leaves the ground, the 3D coordinates of the ball and make a parabola fitting using these coordinate points of the flying ball. This parabola is updated when a new point is received, so that way the parabola error is minimized over the course of the flying ball, as `NuBot` explained on the 2013 Team Description paper[20].

The stereo camera used is one produced by Point Gray Research Company that can be seen on the figure 3.3, the cost of one of this cameras should be around 1500 euros, much more expensive than a normal Kinect.

3.2.3 Fusion of Omni and Monocular camera for the goalie - Hibikino Musashi Team

Like almost all the other teams of the RoboCup MSL, Hibikino already had an omni-directional camera that gives a 360 degrees image of the surroundings. Using only the omni-directional camera is almost an impossible task to detect the height of a flying ball, so they also use information of a monocular front camera and use information from both cameras to calculate the position of the ball.

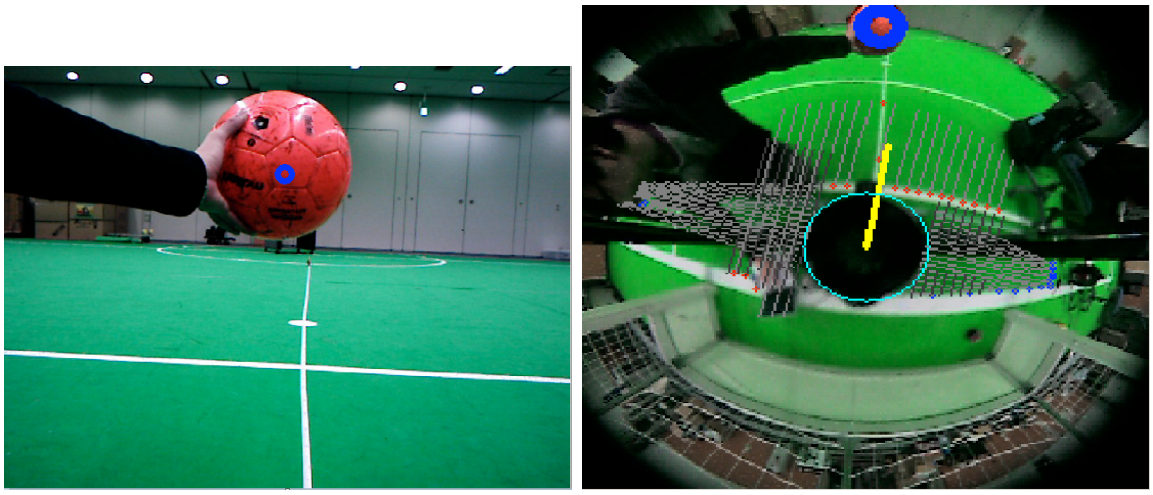
"To detect the height of a flying ball is a demanding task by using only

Figure 3.3: Bumblebee 2 - Point Gray Research stereo camera.

> an omni-directional camera. So we added a monocular camera to realize a stereoscopic view of the ball. Image obtained from camera is used in perspective projection. We developed the expression for omni-directional camera's and monocular camera's ray. The intersection of the two lines is the ball's position."[21]

This fusion of the both cameras gives some information on the height of a flying ball, but this requires that both of the cameras detect the ball, this is an enormous downside of this method, because the omni-directional camera easily loses the ball, this method should only track balls until 50 or 60 cm above the ground, because when ball rises more than that height the omni-directional camera will lose the ball. Still, knowing the exact coordinate of the ball until 50 or 60 cm is better than only detecting ground balls, nevertheless to make this fusion work out it is necessary a great investment and work and the return is not worth all the work that needs to be done.

The ball detection of this algorithm can be seen on the next figure, Monocular camera, figure 3.4(a) and Omni-directional, figure 3.4(b)



(a) Monocular camera (b) Omnidiretional camera

Figure 3.4: Hibikino Ball detection

Chapter 4

Goalkeeper - Positioning

4.1 Motivation

The approach to make the goalkeeper better was to start to understand how the old goalkeeper works and which were the main problems/limitations that should be addressed. One of the biggest problems that was found was the low amplitude of movements of the goalkeeper, his position was always near the goal line. Sometimes this is a good choice, specially for aerial shoots, but most of the times it could be a bit further from the goal line without compromising the defense of aerial shoots. Another big problem was that it was not as flexible as it should be and all the positioning was hard-coded which does not let the user choose the goalkeeper to be more close to the goal line or far from it. When playing against some teams that do not shoot so often by air it may be better to make the keeper play more aggressively and if playing against team with heavy high shoots propensity, force it to stay in the goal line most often. The goal references of the keeper were 2 (two) static points (goal posts points) and for some reason if the goal changed his place even for a few minutes or seconds, for example, one robot crashed with the goal and moved it, the goalkeeper had no knowledge of this change. This could let one side of the goal open and make it easier to score a goal in that side. Having more precision on the real position of the goal also let us be more precise with the approximation of the goalkeeper to the posts, where we can improve two different topics, first of all the distance to the goal post, to be as far from the goal post as possible and still make it impossible for the ball to enter between the keeper and the goal post, and second direction, since the old goalkeeper had the direction cropped at 20 (twenty) and -20 (minus twenty) degrees. It was realized that sometimes these 20 degrees are too low and we can rotate a little more to be more

efficient and close a bigger area of the goal. Also we can use better all the structure of the goalkeeper, the "arms" and the height. Finally, the goalkeeper never left the goal area on defense duty, for example when an opponent is going isolated to the goal it could not move forward to close the shoot angles.

4.2 Positioning

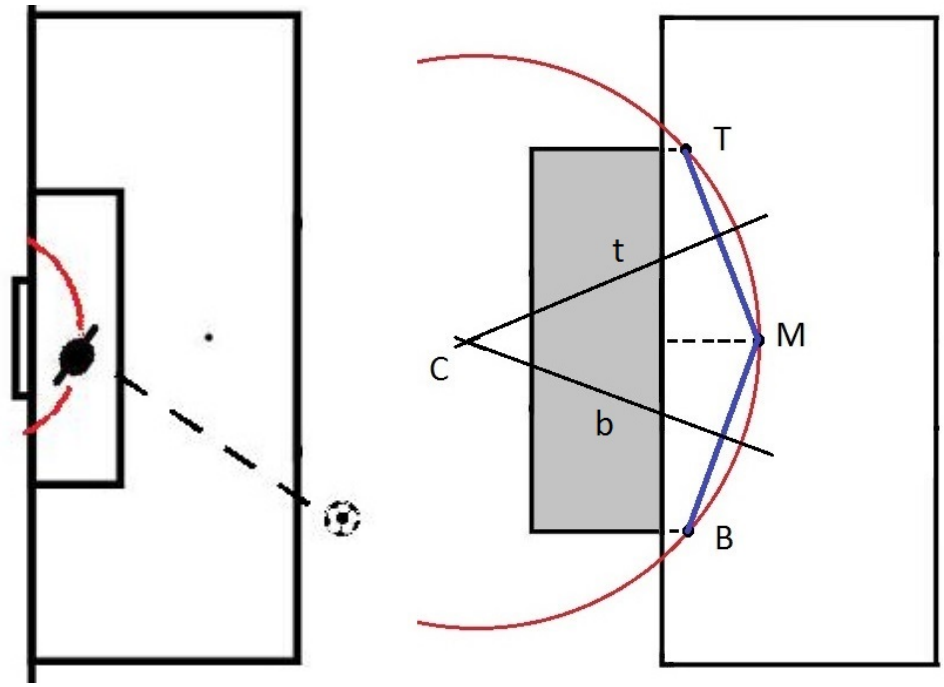
4.2.1 Global

After identifying the problems that the goalkeeper had,this section explains how to solve each problem and on what the solution was based. Resuming, the main topics to address are the **low amplitude** of movements of the goalkeeper, both in terms of its positioning and the goal detection. Work better with the goalkeeper when he is **near the goalposts** taking advantage of his body and calculating the farthest safe distance from the goalposts and **leaving the goal on defense duty** when opponents are isolated and the goalkeeper is the only player between the ball and our goal.

Starting off with the positioning of the keeper, the option was to be always between the center of the goal and the ball, with this approach the keeper zone of action will be defined on a circular arc around the goal, as it can be seen on figure 4.1(a). The red arc passes through the zones where the goalkeeper stands depending on the position of the ball. Hence, to build this circle of action we will need to give at least 3 points to draw a circle, one point is the easiest and more logical to give, the max distance that the keeper will be from the goal line when he is on the middle of the goal. Assuming that the goal is in the correct place, so the goal center will be the point (0.0,-9.0), if the distance to the middle is 0.8m then the that point will be (0.0,-8.2). For the other 2 points, since the circumference is symmetric, so these points are also symmetric from the middle point only one more point will be needed to insert because from one point it is easy to determinate the other one. For an easier user interface a GUI (Graphical User interface) application was created application where these points will be specified. This application will be presented later on section 4.6.

After the three points are specified the goal is to find the center of the circumference. The Figure 4.1(b) helps to understand how to find the center, the distance from the middle of the goal to the point **M** is **circumference middle point distance**, the distance from the left post to point **T**, **circumference top point distance** and the distance from the right post to the point **B**, **circumference bottom point distance**. A straight line that

contains point **T** and **M**,**top middle line** and another one that contains point **M** and **B**, **middle bottom line**. After these 2 lines are determined, find the point that is the center of each one of the lines then apply the line segment bisector around the respective center point, this way the 2 new lines will always cross each other on one point that is the center of the circumference, black lines on the respective figure, that contains the 3 initial points. On figure 4.1(b) considering the 3 given points, **T - top point**, **M - middle point** and **B - bottom point**. The line segment bisector of the line from top point to middle point is **t** and the same proceeding for middle and bottom points is **b**, the interception point will be the point **C**, the center of the circumference. Knowing the center of the circumference, and the radius (distance between the center and one of the given points) it is trivial to draw the circumference.



(a) Positioning around the goal. (b) Find center of a circumference.

Figure 4.1: Goalkeeper circular positioning

This circular positioning is based on Human Goalkeeper Techniques chapter 3.1, where it is explained the importance of keeping a position between reference points and the ball. In this case the reference point will always be the center of the goal and the position of the keeper will be in the interception point between the semi-circumference and the

straight line that contains the ball position and the middle of the goal. It is easy to change the circumference by changing the initial points so the keeper positioning can be totally configurable by the user.

4.2.2 Bisector of a triangle

After the implementation of the global positioning it was felt that sometimes using the goal center as a reference, the goalkeeper was protecting better one side of the goal than it was protecting the other side. So using some maths to try to understand what was happening, it was found that the problem was: taking as an example the triangle of the Figure 4.2, assuming that **C** is the point where the ball is and **B** to **A** is the goal, if the bisector of the triangle on C is made, it will divide the C angle in 2 (two) congruent angles (with the same degrees), so the keeper should protect equally both sides of the goal. This did not happened on the earlier approach, one side of the goal, the closest side from the ball were less protected than the other side.

So, the problem was that the intersection point of the bisector with the goal is not always in the center of the goal, as it can be seen on the Figure 4.2, x is shorter than y. Adjusting the reference point to this new interception point resolves this problem.

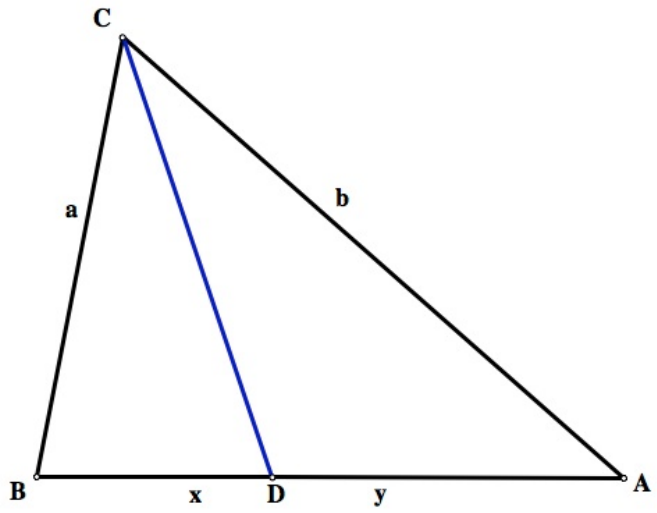


Figure 4.2: Bisector of a triangle. Image from[3]

4.2.3 Goalkeeper relative to the goal

Using the goal posts as points that can move over time gives the goalkeeper much more precision on where the goal really is, but on the other side it makes us re-calculate every point that is related to the goal on every robot cycle, for example, the circumference for the global positioning that was mentioned before, section 4.2.1. This is already implemented, the referential of the goalkeeper is now based on the middle of the real goal and the circumference and every other point related to the goal are re-calculated every frame. However this is already working it is currently not being used, because it is currently impossible to determinate the real goal posts location in the CAMBADA code, which will be discussed in the future work section 6.2.

All this work was developed knowing that in a near future this will be possible to estimate the real position of the goal posts, so the goalkeeper is already prepared to receive those points. The detection the goal posts can be done using the **CAMBADA Vision** mechanism or using a **Laser Range finder**, a device that measures the distance from the device to a target.

4.2.4 Positioning around goal posts

Positioning near the goalposts is an important topic on the goalkeeper because the positioning around the goalposts needs to be the best possible, it cannot allow the ball to pass between the goal post and the goalkeeper, but should be as far from to the goal post as possible(making sure that no ball can enter between the space left from robot to post). This way the goalkeeper keeps protecting the post side because the ball cannot enter between it and the post and the positioning is the closest possible to the middle of the goal enables it to, if needed, move faster to the other side of the goal as the distance is shorter. Hereupon, it is tricky to find the "perfect" position for the goalkeeper near the posts, so the best option is to make it user parameterized and to do that a circumference is drawn around each goalpost with the radius that was chosen by the user and then intersect these post circumferences with the circumference where the goalkeeper will stand, addressed on section 4.2.1. This way we transform the goalkeeper circumference into a circular arc, delimited by a top and bottom limits. The top limit, **maxTopPoint** (nearest point of the left post) will be the intersection between the circumference with center on the left post and the goalkeeper circumference while the bottom limit, **maxBottomPoint** (nearest point of the right post) is the intersection between the right post circumference and the

goalkeeper circumference. Note that these intersections can have up to two intersection points, if that happens, one of those points needs to be chosen. The point chosen for the left post is always the point where XX coordinate is bigger and for the right post it will be the point where the XX coordinate is lower. Figure 4.3 shows a diagram of these ideas, where the post circumferences are the orange circumferences and the black points are the **maxTopPoint** and **maxBottomPoint**.

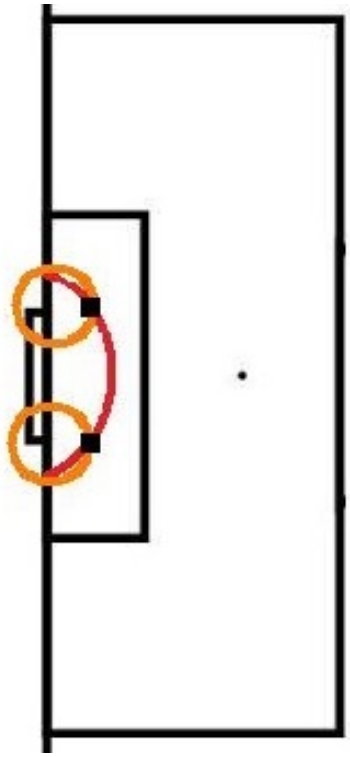
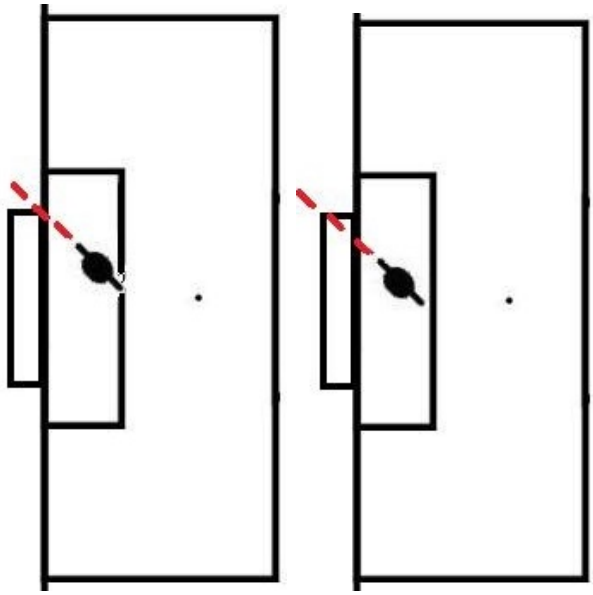


Figure 4.3: Semi-circumference of the goalkeeper

Having the circular arc allows to position the goalkeeper, but then it is also important to define the orientation of the goalkeeper, or where the goalkeeper is "facing" to. Usually it is always looking towards the ball, except sometimes when it is near one of the goalposts. It is needed to calculate the max rotation that the keeper can have, and that rotation will be when a straight line passes by the two goalkeeper "arms" (metallic structure that the goalkeeper has) and also passes by the goal post, so it is taken the maximum possible advantage of the goalkeeper structure. Figure 4.4(a) presents an example of a good orientation, while in the figure 4.4(b) a bad example is demonstrated. Assuming that the center of the goalkeeper is at the same distance to the left goal post, on the first image the distance between the edge of the goalkeeper, his "arms" and the goal post will be lower

than the same distance on the second image.

The rotation from the line (between the goal center and the ball position) to the YY axis will be calculated in degrees, **calculated orientation**, to find the correct orientation for the goalkeeper face the ball. After this the maximum orientation on each post will be determined. It was checked that when the angle between the goalkeeper orientation and a line that crosses the goalkeeper center and the goal post is exactly 90 degrees the goalkeeper is using the full potential of its bounds near the goal post. Is figured the direction that the keeper is facing the goalpost then rotate this direction 90 degrees to the inner side of the field (rotate 90 degrees if the closest post is the left post and rotate -90 degrees if the closest one is the right post) is the maximum rotation allowed near the goal post. Having the calculated orientation and the maximum orientation allowed it is easy to make a clipping on this orientation so that the full potential of the goalkeeper bounds may be used.



(a) Good rotation example (b) Bad rotation example

Figure 4.4: Goalkeeper Rotation near the goal posts

4.3 Active behavior - pressure the opponent

Active behavior of going out of the goal when the goalkeeper is the last robot between the ball and the goal line is pretty important and can solve some defensive problems that

CAMBADA team had, however the goalkeeper needs to be sure that it is the right time to go out of the goal because this is hard and dangerous to do if not properly done. First of all it is important to have the knowledge that the opponent is dribbling the ball towards our goal it should approach the opponent and try to defend. Second, the goalkeeper cannot move forward too soon, as it has some risks in terms of aerial balls, so the time that it takes to the goalkeeper to leave the goal and get next to the opponent should be the least possible. Lastly, if the ball is not on the central zone of the field, YY axis less than $|1.3|$, the goalkeeper should not leave the goal, because its position should already be near one goal post and it is already protecting most of the goal and since this active behavior should only activates when we are sure that it would help much more than staying on the goal line, which is clearly not the case.

In order to get out of the goal some conditions have to be guaranteed:

- The opponent is dribbling the ball and is already close to our goal
- The goalkeeper is the last option possible
- The opponent is on the central zone of the field (YY axis less than $|1.3|$)

In spite of the importance of the conditions to trigger it is also important what to do when the conditions are triggered, it needs to leave the goal on the direction of the opponent and stay at a certain distance.

4.4 Role Goalie

Since the goalkeeper is much more user configurable than it was before and on every cycle many points are redefined, the role Goalie needed a major update. The first time that it is called, it reads all the configurable variables for the goalkeeper from the `CAMBADAconfXML`, a XML that is used to store most of the configuration variables of the CAMBADA team or if the variables do not exist will assign the default values.

On every cycle at the start it is checked if the ball is visible or not to update the last time the ball was seen, then the goal posts position is read to update left and right posts and also create the new goal line, goal center and goal orientation. This goal orientation will be used to determine the goalkeeper circumference. After having the new goal, the goalkeeper circumference will be calculated again as well as the goal posts circumferences (Blue Goal Posts or Yellow Goal Posts). The reason to have different goal posts on each

goal is because the robots have small positioning error on the field, if the goalkeeper has an error of 5 centimeters to the left post on the yellow goal for example probably it will have the same 5 centimeters of error to the right post of the other goal. To correct this errors it is needed to work on each goal post individually. It also has the advantage of defining the distance to each post individually which is crucial to maintain the same distances to all the posts. Finally, the last step to do is determine the `maxTopPoint` and the `maxBottomPoint`, limits of the goalkeeper arc, with the intersection of the goal posts circumferences and the goalkeeper circumference creating the circumference arc where the goalkeeper will move.

Algorithm 2 Role Goalie - every cycle calculations

Input: cambadaConfigXML config
Output:

```
1: if ball is visible then
2:     restart ballNotSeen times
3: end if
4: update left post and right post position
5: update new goal line
6: calculate goal center and goal orientation
7: determine goalkeeper circumference
8: if my goal color = Blue then
9:     create Blue goal post circumferences
10: else
11:     create Yellow goal post circumferences
12: end if
13: determine maxTopPoint and maxBottomPoint of the circumference
```

4.5 Behaviors

This section presents the three behaviors that were developed to the goalkeeper on this thesis, `BGoaliePositioning`, `BGoalieStandStill` and `BGoalieInsideArea`. It makes sense to refer again that the behaviors have an hierarchy that is defined and this hierarchy, from the higher to the lower, is the following:

- BGoalieStandStill
- BGoalieInsideArea
- BGoaliePositioning

The behaviors will not be presented by hierarchy order because it makes sense to start presenting `BGoaliePositioning` where resides most of the artificial intelligence of the goalkeeper.

4.5.1 BGoaliePositioning

As said before in this behavior resides great part of the intelligence of the goalkeeper, this is the behavior that the goalkeeper uses most of the time, but that does not mean that it is the most important behavior. Since this is the behavior with the lowest priority the **Invocation conditions** is always true and the **Commitment conditions** always false so this will be invoked when no other behavior is invoked and will be released when any other behavior is invoked. This behavior is the most complex of them all because it solves many defensive issues such as moving the goalkeeper to the center of the goal after not seeing the ball for a defined time interval, the defense of shoots to the goal, the global positioning of the goalkeeper while the game is running, the defensive position near the goal post previously presented on section 4.2.4 and also the more active posture presented on section 4.3.

These are the main 5 blocks of this behavior which will be presented in detail:

- moving the goalkeeper to the center of the goal after ball not seen more than a defined time interval
- the defense of shoots to the goal
- the global positioning of the goalkeeper while the game is running
- the defensive position near the goal post
- the more active posture

In the beginning it is checked if the ball is not seen for more than `ballNotSeenMaxTimer` seconds, a variable defined by the user, if so the `targetPos` will be the center of the goal and the goalkeeper will be oriented about the YY axis, then the goalkeeper will be moved to the target position with the desired orientation and the behavior will return. The second big block is detecting if the ball is on the defensive half court and the velocity of the ball about the YY axis is less than zero, because shoots need to be done from the attacking side of the field (our defensive half) and the ball is heading towards our end field, when

these conditions are true it calculates the ball trajectory and checks if this trajectory will intersect the goal line (between goal posts). When the intersection exists, it is already known that the shoot is going in the goal direction and the goalkeeper needs to be in the trajectory of the ball. At this point another intersection is made between the goalkeeper circumference and the ball trajectory and this intersection will be the target position of the goalkeeper. As this intersection can have one or two intersection points, in case it has two points the right one will be the one that has the highest value in the YY axis. The flag `globalPositioningMode` is also set to false, so the global positioning will not overwrite this target position. If the earlier conditions are not met and the `globalPositioningMode` flag is still true, the global positioning block will determine the goalkeeper behavior. This block defines the normal positioning of the goalkeeper on the goal during the play on, the goalkeeper will be over the goalkeeper circumference, and it consists in detecting the angle constituted by the straight lines between the ball position and each one of the goalposts, then find one point on each straight line that is at the same distance of the ball and calculate the mean of that two points. This point, which will be named `bisectorPoint` because is one point that belongs to the bisector of the goal posts straight lines. Having this point is easy to get the `bisectorLine`, the line that passes on the ball position and on the point calculated before.The intersection of this line with the goalkeeper circumference gives the position to place the goalkeeper. The last step to do on this block, if the intersection made before gives more than one intersection point, is, one more time, to choose the right one, which in this case will be the point closest to the ball position. At this point the `targetPos` is known, either from the defense of a shoot or from the global positioning then this position is adjusted because of the goal posts. To find the orientation, calculate the orientation to look forward to the ball from the target position and adjust this orientation if it is above the maximum allowed near the goalposts, as explained in section 4.2.4. Now it is time to test if it is required to have a more active approach analyzing if the conditions enter/leave the active mode are met. The conditions to enter on this mode were already defined in section 4.3 and if any of these conditions are not met (with a defined margin) it will leave this mode. The margins were set to avoid the constant flipping between entering on the conditions and immediately leaving, so for example to enter on the active method the ball needs to be less than 2.5 meters from the goal center but to leave this mode it needs to be more than 2.7 meters, these 20 centimeters assure some continuity margin. When the active mode is to be applied, the goalkeeper will be forced to play active and applied pressure near the attacker, the new target position will be between the target position and

the ball, at a distance of half a meter from the ball. A pseudo-code of this explanation is also presented on the Algorithm 3.

4.5.2 BGoalieStandStill

This behavior is a minor behavior but still significant, its only function is to force the goalkeeper to stand still on some situations, because sometimes it is better to stand still and do not move than score an own goal. It is triggered in some specific situations such as when the ball is behind the goalkeeper, Y coordinate of the ball is smaller than goalkeeper Y coordinate, assuming that the referential is relative to the goalkeeper and not to the field (YY axis is aligned with the front of the goalkeeper), the ball is closer to the robot than a defined safety margin and the ball is on the line goal and any touch on the ball could throw it inside the goal. Since it is a very delicate situation and sometimes the robot does not see the ball it is a must to use the last known ball when the ball is lost to keep triggering this behavior.

The **invocation conditions** of this behavior are the set of this conditions:

- (ball is behind the goalkeeper **and** ball is closer to the robot than a defined security margin **and** ball is close enough to the goal) **or** ball is in the goal line

Note: the conditions above are using the ball detected by the goalkeeper, but when he does not detect the ball the last known position is used instead.

The **commitment conditions** are the same as the invocation conditions, so this behavior will be release when the invocation condition is not met.

4.5.3 BGoalieInsideArea

When the goalkeeper stands most of the times near the goal line there is no motive to worry about when the ball is behind the goalkeeper because when that happens the ball is already inside the goal, but since the actual goalkeeper leaves the goal on defense duty and has a more aggressive defensive posture the probability of having a ball behind the goalkeeper is higher and can not be ignored. This behavior is critical and needs to be error free, because with a simple error is possible to touch on the ball and score an own goal, so it is extremely important to avoid touching the ball when it is behind the goalkeeper and also avoid crash on any other obstacles because with the impact the robot can involuntary touch on the ball. This behavior includes two different approaches, one that is the more

Algorithm 3 Behaviour - goalie positioning

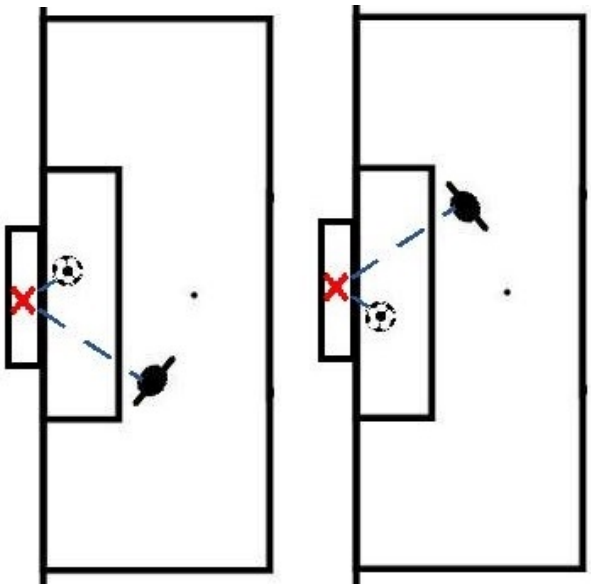
Input: RoleGoalie parent, DriveVector dv
Output: DriveVector dv

```
1: globalPositioningMode = true
2: if parent->ballNotSeen > ballNotSeenMaxTimer then {ball is not seen more than ballNotSeenMaxTimer seconds}
3:    targetPos = center of the goal
4:    rotation = orientation vector about YY axis
5:    dv = move(targetPos, rotation)
6:    return dv
7: else if ball in defensive court and ballVel.y < 0 then {ball heading in our end line direction}
8:    calculate ball trajectory
9:    goalIntersection = intersect(ball trajectory, end line)
10:   if goalIntersection and goalIntersection is between goal posts then
11:      circumferenceIntersection = intersect(ball trajectory, goalkeeper circumference)
12:      if circumferenceIntersection then
13:         globalPositioningMode = false
14:         targetPos = firstCircumferenceIntersection
15:         if circumferenceIntersection.size()>1 and secondCircunferenceIntersection.y > firstCircumferenceIntersection.y then {exists two intersections, choose the right one}
16:            targetPos = secondCircunferenceIntersection
17:         end if
18:      end if
19:   end if
20: end if
21: if globalPositioningMode then
22:   ballLeftPointDir = vector from ballAbs to leftPost
23:   ballRightPointDir = vector from ballAbs to rightPost
24:   bisectorPoint = point that is the bisector between ballLeftPointDir and ballRightPointDir
25:   bisectorLine = line(ballAbs, bisectorPoint)
26:   finalPos = intersect(bisectorLine, goalkeeper circumference)
27:   targetPos = firstFinalPos
28:   if finalPos.size()>1 then {exists two intersections, choose the closest to the ball}
29:      if distance(firstFinalPos, ballAbs) > distance(secondFinalPos, ballAbs) then
30:         targetPos = secondFinalPos
31:      end if
32:   end if
33: end if
```

```
34: adjust targetPos because of the goal posts
35: rotation = calculate the rotation to look forward to the ball from the targetPos
36: clipping rotation (maxRotation near goal posts)
37: if opponentDribbling and distance(ballAbs, goalCenter) < 2.5 meters and |ballAbs.x|
    < 1.3 meters then {verify if needed an active behavior}
38:    activeMode = true
39: end if
40: if not opponentDribbling or distance(ballAbs, goalCenter) > 2.7 meters or |ballAbs.x|
    > 1.4 meters then {conditions to exit active behavior}
41:    activeMode = false
42: end if
43: if activeMode then
44:    vecBallTargetPos = vector from targetPos to ballAbs
45:    vecBallTargetPos.setlength(vecBallTargetPos.length()-0.5 meters)
46:    targetPos = targetPos + vecBallTargetPos {force the goalkeeper to play active and
    apply pressure to the attacker}
47: end if
48: dv = move(targetPos, rotation)
49: return dv
```

intuitive, try to go to a point inside the goal so that the goalkeeper contour the ball and as soon as it is behind the ball just change to the normal defensive behavior considering that touching the ball is not critical anymore 4.5(a). This seems simple to do but it cannot be forgotten that sometimes this will be too risky or even impossible to do, if the ball is behind the goalkeeper and too close to contour the ball, it will be a extremely hard task and the time that would take this contour and the difficulty level is enormous 4.6(a), so the best option in this case is to protect the ball with the goalkeeper body, placing itself ahead of the ball 4.6(b), this way if an opponent tries to go to the ball will crash on the goalkeeper first which is a foul. The behavior earlier presented on section 4.5.2, **Behavior Stand Still** will complement this behavior, when the goalkeeper is getting too close to a behind ball it will force the goalkeeper to stop since its priority is higher.

Explaining now how this behavior works, first of all it assures if the ball is really between the goal post, if this happens, check which side of the goalkeeper the ball is, if it is by the left side, then the goalkeeper should move to a place behind the goal on the right side of the ball, to avoid crashing with the ball when moving to that place, figure 4.5(a), if the ball is right of the goalkeeper then the point to move will be a point inside the goal by the left side of the ball position, figure 4.5(b). After this point is calculated `targetPos` it needs to be clipped to avoid colliding with any goalpost. The next step to do is to verify if the



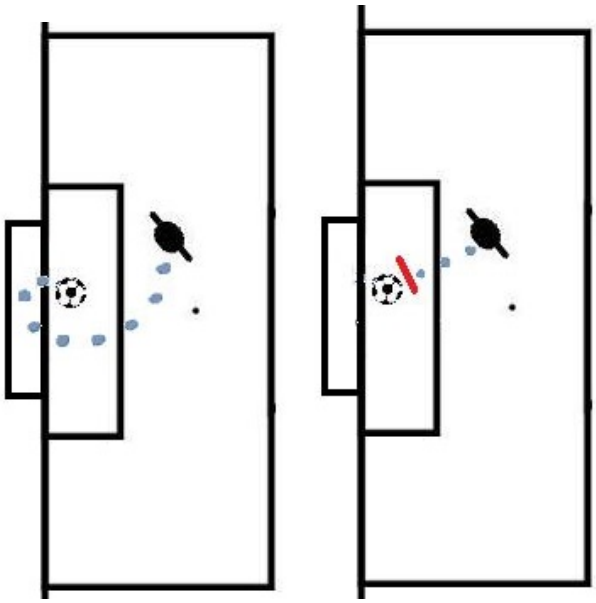
(a) ball left of goalkeeper (b) ball right of the goalkeeper

Figure 4.5: Ball behind the goalkeeper

path to go to that calculated point is safe or not, if cannot go to the `targetPos` because the space to move between the ball and the goalpost is too low and the goalkeeper would need to contour the ball by the other side of the goal, it is better to just protect the ball in front of it, so this `targetPos` will be adjusted to a point in front of the ball. At this point the position to move is already calculated, so it is only needed to calculate the orientation (where the goalkeeper is facing), if the best option is to protect the ball with the body, the goalkeeper should be facing to the opponent half field, this way it occupies the most area ahead of the ball, in case the best option is to contour the ball then the best rotation is to be facing the ball, this way the probability of colliding on the ball with his "arms" is lower. After both point to go and rotation are calculated, it is just needed to pass them to the `DriveVector`, which will be responsible of passing this arguments to the lower level actuator and motors. The pseudo-code of this behavior is also presented on Algorithm 4 for an easier understanding.

The **invocation conditions** of this behavior are the **logical and** of this conditions:

- ball is close to the goal line in the YY axis (parent->ballAbsolute.y < -8.4)
- ball is between the goal post in the XX axis (|parent->ballAbsolute.x| < 1)
- ball is less than 0.2 meters ahead (YY axis) of the goalkeeper, position relative



(a) Hard path to go (b) Body protecting

Figure 4.6: Goalkeeper protecting ball with body

to the goalkeeper (ball.posRel.y < 0.2)

The **commitment conditions** are false, behavior will be released, when any of the following conditions is met.

- ball is **not** between the goal post in the XX axis (|parent->ballAbsolute.x| $>$ 1)

- ball is **more** than 0.2 meters ahead (YY axis) of the goalkeeper, position relative to the goalkeeper (ball.posRel.y > 0.2)

4.6 Goalie configuration interface

In order to present a more intuitive way of configuring some parameters/variables it was decided to make a graphical user interface, in this interface the user will configure parameters for the goalkeeper such as the points to specify the goalkeeper circumference, the distance to maintain to each one of the goal posts (Blue goal and Yellow goal) and the security margin to stop when a ball is behind the goalkeeper. It was found that the two point to make the goalkeeper circumference needed to be configured in a graphical

Algorithm 4 Behaviour - ball inside area

Input: RoleGoalie parent, DriveVector dv
Output: DriveVector dv

```
1: if parent->leftPostFloating.x < ballAbs.x < parent->rightPostFloating.x then {ball between goal posts - careful mode}
2:    if ball is at right side of the goalkeeper then
3:       targetPos = Vec(ballAbs.x - 0.25, -field->halfLength* 1.05); {Point inside the goal on the left side of the ball}
4:    else
5:       targetPos = Vec(ballAbs.x + 0.25, -field->halfLength* 1.05); {Point inside the goal on the right side of the ball}
6:    end if
7:    clipping targetPos to avoid crash with goal posts
8:    if too risky to go behind the ball then
9:       targetPos = point ahead of the ball {Risky move to go behind the ball then use body to protect the ball}
10:      modeProtectBall = true
11:   end if
12: end if
13: if modeProtectBall then
14:   rotation = Vector(0,1) {look in the direction of the opponent field}
15: else
16:   rotation = calculate the rotation to look forward to the ball from the targetPos
17: end if
18: dv = move(targetPos, rotation)
19: return dv
```

way on the field, so the user has some more feedback about where the points really are on the field. This interface also draws the circumference based on the two actual points and when the user moves any of the points the circumference will be redesigned and its radius and center recalculated and presented on the application. The way to insert this point is "drag-and-drop" on the field about the YY axis. To make the configurations easier in the CAMBADA team and also because there was already one graphical user interface for other CAMBADA parameters `CAMBADA config v3.0`, this goalkeeper interface was created into a new tab on the global CAMBADA configurable interface, figure 4.7. All the other configuration parameters of CAMBADA were written on `CAMBADAconfXML.xml` so the goalkeeper parameters were also stored on this file on a specific sub block, `goalieParam` 4.8.

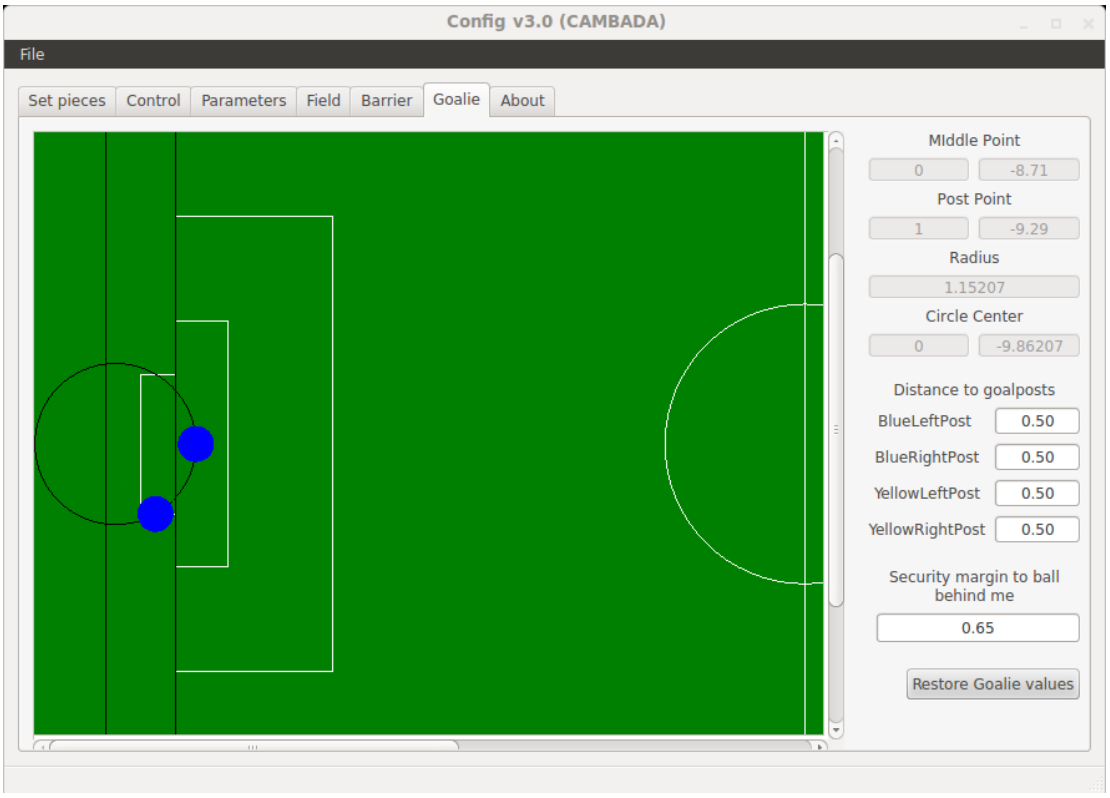


Figure 4.7: Goalie graphical user interface

```
<goalieParam name="bLeft" value="0.500000" comment=""/>
<goalieParam name="bRight" value="0.500000" comment=""/>
<goalieParam name="midLength" value="-8.710000" comment=""/>
<goalieParam name="securityMargin" value="0.650000" comment=""/>
<goalieParam name="sideLength" value="-9.290000" comment=""/>
<goalieParam name="yLeft" value="0.500000" comment=""/>
<goalieParam name="yRight" value="0.500000" comment=""/>
```

Figure 4.8: Goalie parameters written on CAMBADA xml file

4.7 Experimental results

4.7.1 Positioning

To verify the efficiency of the new goalkeeper positioning some tests were made to check if it was an improvement compared to the previous goalkeeper. This test consists on placing one CAMBADA robot acting as a striker on a specific place in the field shooting at the goal. This shoot is a ground shoot so that way the goalkeeper can track the ball on its trajectory and successfully defend. The direction of the shoot is controlled so the striker aims always to the same spot, near one goalpost, so every attempt is the closest possible

to the previous attempt. On each position on the field the striker shoot twenty times to the goal with the new Goalkeeper defending the goal. The test is repeated on the same position with the previous goalkeeper defending the goal. Only shoots that are on the goal direction are counted, the ones that hit the goalpost and did not enter on the goal are not considered. Two different positions on the field were chosen for this experiment, the first one is from 4 meters distance in front of the goal shooting at one side of the goal. The side chosen was the right side of the goal (striker view). The setup can be seen on figure 4.9.

Figure 4.9: Front shoot test setup

The other position on the field to test was from one side of the field, 2 meters to the right of the front shoot position, aiming at the left side of the goal (striker view). From this position twenty shoots were also made while the new goalkeeper was defending and twenty more with the previous goalkeeper defending. The number of times that the goalkeeper successfully prevents the ball to enter on the goal is noted for the results. The table 4.1 presents the number of defenses and the percentage of defenses.

Shoot type	**Goalkeeper**	**number** of **defenses**	**total shoots**	**defense percentage**
Front	**Previous**	9	20	45%
	New	12	20	60%
Side	**Previous**	8	20	40%
	New	15	20	75%

Table 4.1: Positioning defense results

Figure 4.10: Side shoot test setup

Analyzing the results from table 4.1, for the front shoot the previous goalkeeper defended 9 out of 20 shoots which is a total 45% of the shoots while the new goalkeeper defended 12 shoots out of 20, a percentage of 60% of the shoots. The new goalkeeper defended 33% more shoots than the previous one. This slight increase is due to the fact that the new goalkeeper is a little further from the goal than the previous which causes an increase in the covered area.

Regarding the side shoot the previous goalkeeper defended 8 out of 20 shoots which is a percentage of 40% of the total shoots, while the new goalkeeper defended 15 out of 20 shoots, an amazing 75% of successfully defenses. This increase is higher than the one of the front shoot due to the fact that the new goalkeeper position himself better around the goalpost than before, closer to the goal center, leading to a faster response when needed to defend on the other side of the goal, which is the case. The efficiency in the side shoots increase to 88% more shoots defended.

4.7.2 Goal occupation in percentage

Changing the position of the goalkeeper on the goal or adjusting the rotation of the goalkeeper causes an huge impact on the percentage that the goalkeeper occupies on the goal. A slightly 1% degree rotation change can be the difference between letting score a goal or defend a shoot. So while running the tests from the previous subsection 4.7.1 the percentage of the goal occupied was measured in field. In the table 4.2 the occupation

values are presented.

Shoot type	Goalkeeper	Occupied percentage
Front	**Previous**	43%
	New	47%
Side	**Previous**	63%
	New	52%

Table 4.2: Goal percentage occupied by goalkeeper

While the previous goalkeeper only occupied 43% of the goal, the new goalkeeper occupies 47%. This small increase is due to the fact that the new goalkeeper is farther from the goal line than the previous one. This justifies the fact that the new goalkeeper defended 33% more front shoots than the previous one.

Comparing the values on the side shoot the difference is way bigger, 52% by the previous goalkeeper versus 63% occupied by the new one. This difference is affected by the rotation of the goalkeeper and also how close the goalkeeper is from the goal post. In the image 4.11 can be seen where the new goalkeeper position himself when the ball is in the side shoot position from the previous subsection 4.7.1. With the ball on the same position in the field, the old goalkeeper position himself as it can be seen on figure 4.12. The new version is using better the rotation to occupy a more percentage of the goal.

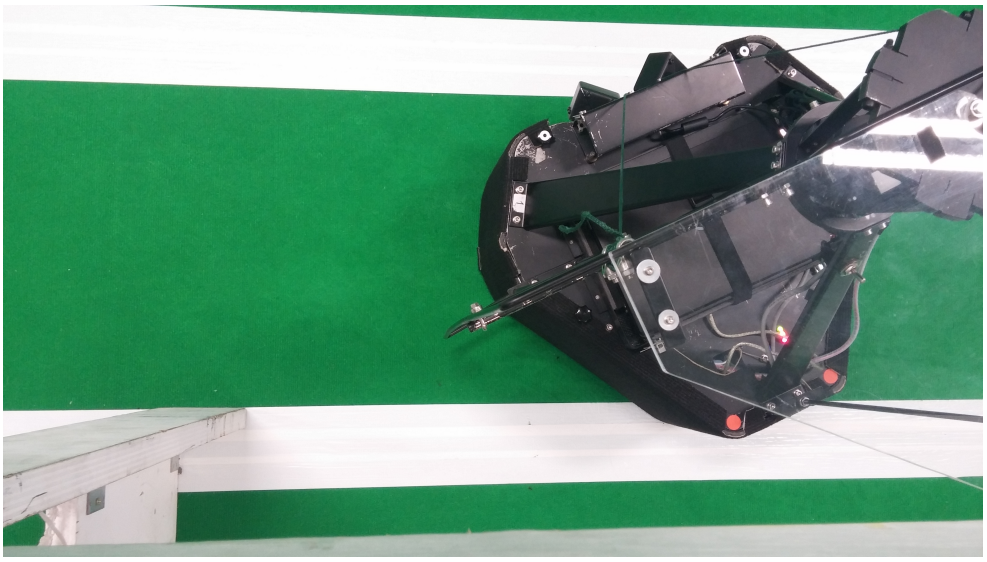


Figure 4.11: New goalkeeper near goal post.

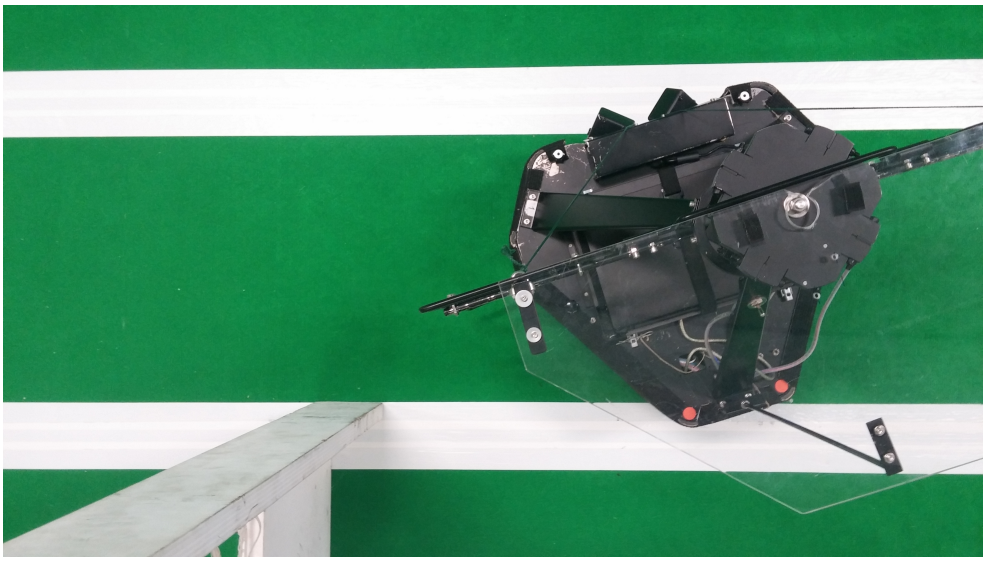


Figure 4.12: Previous goalkeeper near goal post.

4.7.3 Active behavior

The shooting positions on the subsection 4.7.1 tests were too far away to enable the active behavior from the goalkeeper, so in this experience the shooting point is 2 meters ahead of the goal. The striker starts with the ball, moves to the shooting point and try to shoot to the right side of the goal. The previous goalkeeper tries to defend the shoots with the normal positioning near the goal while the new goalkeeper tries to make an aggressive approach to the striker to avoid letting the striker scores. If the striker takes too long to shoot and let the goalkeeper move towards him the final position on the new goalkeeper can be checked on figure 4.13. The previous one stands near the goal as figure 4.14 suggests.

Table 4.3 presents the defense rate of both the goalkeepers. While the new goalkeeper defended 14 out of 20 shoots, 70% of the shoots, the previous one defended only 5 out of 20 total shoots, 25% of the shoots.

Shoot type	**Goalkeeper**	**number** of **defenses**	**total shoots**	**defense percentage**
Active Behavior	**Previous**	5	20	25%
	New	14	20	70%

Table 4.3: Active behavior defense results

With the new goalkeeper the increase of defended shoots is huge, and with such a low defense percentage without leaving the goal on defense duty, it is much better to try

Figure 4.13: New goalkeeper active behavior.

to deny the striker leaving the goal. The faster the striker shoots the lowest will be the percentage of defenses because it takes some time since the goalkeeper leave the goal till when it is applying pressure to the opponent, but if teams try to shoot close to the goal for an higher chance of scoring leaving the goal will always be the best solution.

Figure 4.14: Previous goalkeeper no active behavior.

Chapter 5

Goalkeeper - 3D balls

5.1 Motivation

This chapter presents the detection of 3D balls, an important topic on the MSL, not only for the goalkeeper but also for every field player of the teams. Given that every robot already have the capacity to shoot in height, detecting the ball when in flight is becoming as important as ever. There are already solutions using more than a single camera, either using two additional cameras to provide stereo vision, like `NuBot Team` uses, section 3.2.2, or combining the information from the omni directional camera with a monocular camera, section 3.2.3, but this requires a coordination between cameras and also an additional camera. These additional cameras can be expensive, for example the NuBot's stereo camera costs 1500 euros. A cheap alternative can be using the Kinect just like `Tech United` uses, taking advantage of his depth sensor and RGB camera.

5.2 Kinect Sensor

Kinect is a motion sensor developed by PrimeSense to Microsoft for video games consoles that enable the users to control and interact with the console via natural user interface such as gestures and spoken commands. It is composed by a RGB camera module, a Depth infrared sensor, microphone to detect voice commands and detect 48 body articulation points to precise the human movement. The Kinect was developed for video consoles purposes but its usage in past beyond the world of games. This "all-in-one" sensor has a friendly price of 100 euros per piece and the 640x480 resolution RGB camera working at a refresh rate of 30 Hz and the 320x420 resolution working also at 30 Hz depth sensor

makes this sensor great in terms of quality/price relationship. The Kinect assures good precision values between 0,6 meters and 4,5 meters of distance. The only limitation is the low amplitude of field of view, 58 degrees of horizontal field-of-view and 44 degrees of vertical field-of-view.

5.3 Aerial object detection algorithm

In order to make the goalkeeper defend aerial balls using the Kinect sensor there was already some work developed before this thesis started, namely an algorithm to detect aerial objects [22]. This algorithm uses the voxelization of the 3D space of the Kinect's depth infrared sensor given the **voxel size**. For each voxel, the number of points inside is computed and stored. This way the algorithm works with the occupancy of the voxels instead of working with all the cloud points. This step allows an increase of the processing speed reducing the global computation and also allows to define a flying object as an object that occupies one or more voxels with a minimum number of points and whose surrounding voxels are empty, a 3D mask, where the inside voxels are not empty and the outside voxels are empty. This **mask size** (in voxels) is defined by the user, but the most common masks are 5 (5 x 5 x 5 cube) and 7 (7 x 7 x 7 cube). The inner voxels of the mask define the maximum size of the object to be detected while the outer voxels define the minimum distance to any other object in order for that object to be considered a flying object. For example, with a mask size of 5 and a voxel size of 0,15 meters it will be created a red cube with 0,15 x 5 meters of side and a inner blue cube of 0,15 x 3 meters of side, colors that serve for visual purposes only, can be seen on Figure 5.1. If there are some points detected on the inner zone of the mask and no points detected on the red zone and there are more points than a defined value, it is a potential candidate for being a flying object. On this example the ball can have up to 0,45 meters to fit on the inner part of the mask and is has to be at least 0,15 meters away from every other point. The values used for the voxel and the mask depend on the size of the ball to detect but also have many consequences on the global process time. The smaller the voxel is, the more voxels will be created in the field-of-view and more iterations will be needed to sweep all of them. In terms of grid size, the highest the grid size more time will take the program to run. A 7 x 7 x 7 cube on each iteration takes longer than a 5 x 5 x 5 grid. These values need to be a compromise between precision and performance, while on one side a small voxel size would result in more false flying objects specially when far away from the Kinect, on the other side a mask with an

inside volume of almost the same size of the ball would miss to detect several balls. To detect all the possible candidate a sweep over all the voxelization area is made, moving the mask unitarily (one voxel is the unity). The voxel and mask sizes of the figure are not the same as the example given, it is just an illustration. After the sweep done the candidates are analyzed to check which one is the most reliable.

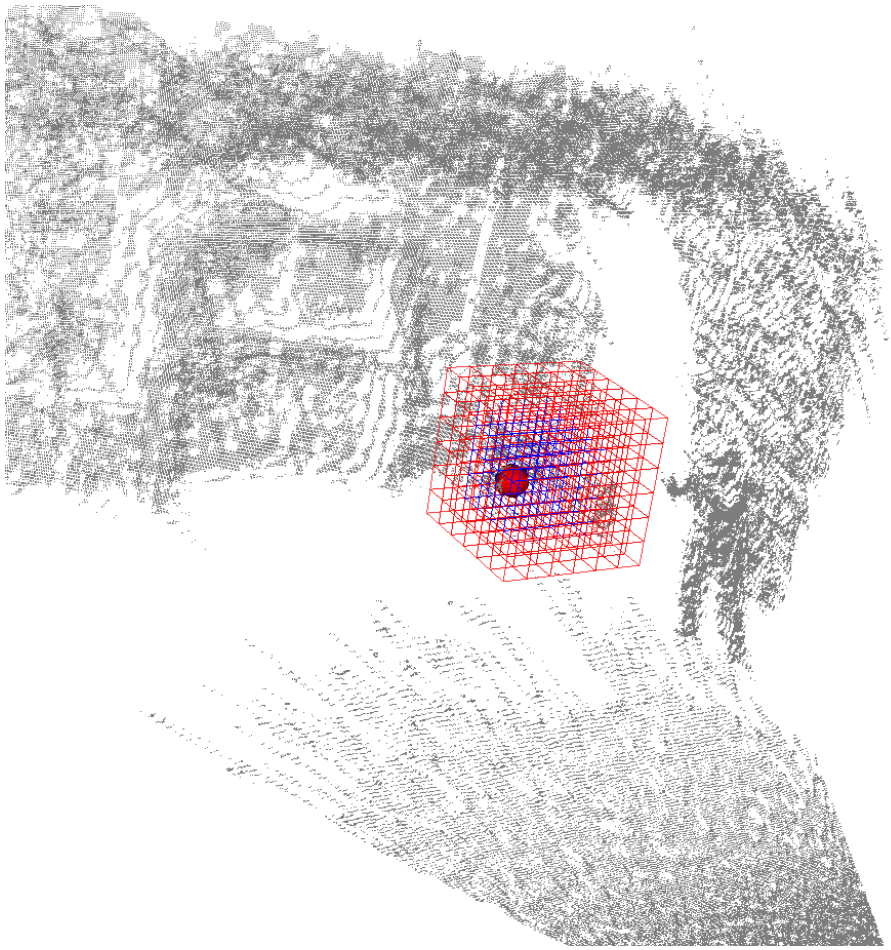


Figure 5.1: Aereal object detection algorithm detecting one ball. Image from [23].

5.4 Trajectory estimation algorithm

In robotic soccer, only the position of the 3D ball does not give the maximum confidence to the goalkeeper. The ideal is to calculate the ball trajectory based on its position over time so that the goalkeeper can move to the right position to prevent goals. At the same time that the detection algorithm 5.3 detects the ball, its position is stored in a trajectory

estimation module that hold up to the last 10 ball position. Assuming that the air resistance is negligible, which is not always true, the flying trajectory was approximated to a simple quadratic equation. Every time a new ball is detected the error between this new point and the actual trajectory is calculated and, if the error is less than a defined threshold, the point is supported by the trajectory and the trajectory updated. The pseudo-code for the trajectory estimation can be seen on Algorithm 5 and also on the paper presented on the RoboCup 2014 symposium[23].

Algorithm 5 Trajectory estimation algorithm from [23]

Input:
Output:

```
update ball history
if ball detected then
    if new position support previous trajectory then
        Compute trajectory with all points
    else
        Compute new trajectory with last 3 points
    end if
    if trajectory error below threshold then
        update new trajectory
    end if
end if
if not(last 2 positions exist and support trajectory) then
    reset trajectory
end if
```

To fit the points that support the trajectory to a parabolic equation was used Eigen[24], a C++ template library for linear algebra such as matrices, vectors and related algorithms, more precisely the Eigen Singular Value Decomposition (SVD) algorithm. It can be seen on the figure 5.2 the red spheres represent positions of the ball and small blue spheres the calculated trajectory of the ball.

5.5 Existing algorithm limitations

Although these algorithms could detect and predict the trajectory of flying objects they were never really tested before in terms of reliability to error and precision, nevertheless there are some points that draws our attention. First of all the algorithm was made to be used in a very controlled environment, only one flying object, no more than one ball,

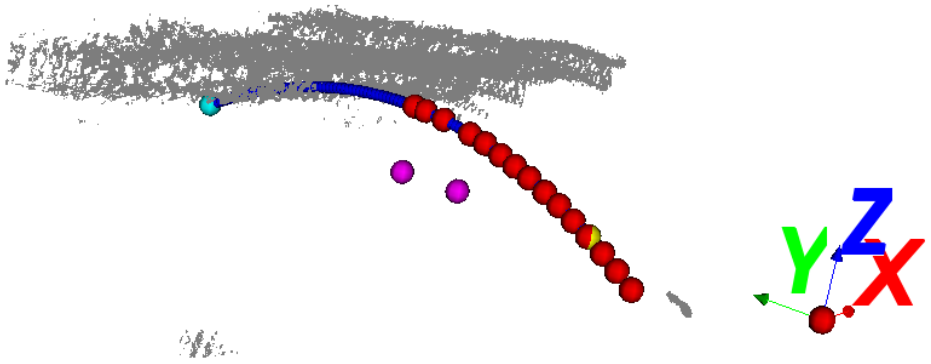


Figure 5.2: Trajectory estimation: blue spheres represent the trajectory computed from the red spheres, earlier ball position. Image from [23].

and the reaction to any other object on the Kinect field-of-view could trick the algorithm and not be able to detect the right ball. If the algorithm detects more than one possible candidate it has no way to choose one over another besides the number of points detected. Also shows deficiencies when the object detected is near the edges of the field-of-view (FOV) because there is no information if there are points outside of the FOV, there is no way of differentiate if it is really a flying object or if it is the edge of any object that is almost outside of the Kinect FOV and by default the algorithm marked this points as a new flying object.

5.6 Validating Kinect depth sensor with RGB camera

Validation of possible candidates with their color is a required issue to minimize the candidates to be a possible ball. Since the ball is heavily controlled in the MSL league, color and dimensions, the algorithm can be adjusted to detect only object with specific parameters. In order to validate a depth pixel with color there are some steps to follow. First of all it is needed to convert the depth pixel into the correspondent color (RGB) pixel. To solve this problem a Kinect calibration guide was followed [4], mapping depth pixels with color pixels. This guide is based on the IEEE paper "Joint depth and color camera calibration with distortion correction" [25]. It uses Kinect intrinsic parameters of RGB 5.1 and depth 5.2 to make the calibration of the cameras.

First of all it is needed to reverse the distortion of both RGB and depth images using the distortion coefficients of the Kinect. Using the depth camera intrinsic parameters, each pixel (x_d,y_d) of the depth camera can be projected into a 3D point (P3D) using Equation 5.1, where fx_d, fy_d, cx_d and cy_d are intrinsic parameters of the depth camera from Table 5.2.

$$\begin{aligned} P3D.x &= \frac{(x_d - cx_d) * depth(x_d, y_d)}{fx_d} \\ P3D.y &= \frac{(y_d - cy_d) * depth(x_d, y_d)}{fy_d} \\ P3D.z &= depth(x_d, y_d) \end{aligned} \tag{5.1}$$

As soon as the 3D point is calculated, it needs to be rotated and translated using a specific rotation matrix 1 and translation matrix 2 due for calibration and a new 3D point will be created, P3D'. Using this point we are able to finally find the correspondent coordinates from the RGB camera using Equation 5.2, where cx_d and cy_d are intrinsic parameters of the depth camera from Table 5.2 and R is the rotation matrix 1 and T the translation matrix 2

$$P3D' = R.P3D + T \tag{5.2}$$

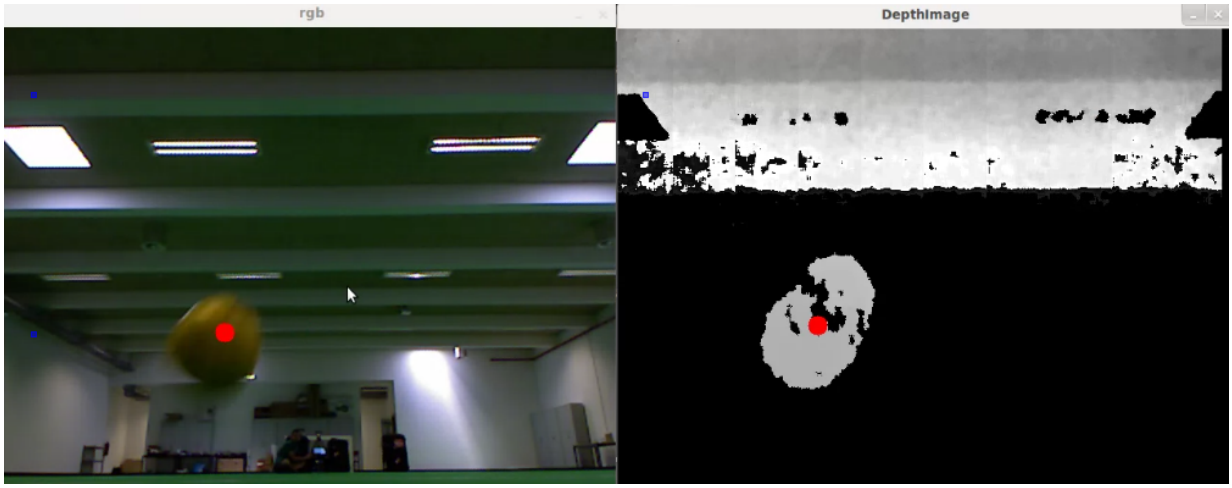
$$\begin{aligned} P2D_{rgb}.x &= \frac{P3D'.x * fx_{rgb}}{P3D'.z} + cx_rgb \\ P2D_{rgb}.y &= \frac{P3D'.y * fy_{rgb}}{P3D'.z} + cy_rgb \end{aligned} \tag{5.3}$$

To check the transformation from the depth sensor pixel to the RGB pixel, the two images were displayed. With this debugging mode setup displaying both images the time of execution of each cycle rises abruptly, around 300ms per cycle, so it was harder to get a clear image. One robot was placed on the half of the field shooting balls to the goal. The Kinect sensor was placed on the goal line simulating a real game situation. In Figure 5.3(a) and Figure 5.3(b) are presented 2 examples where the red circle from the left side of the figures is calculated from the ball position detected from the depth camera (depth red circle). The goal of this visual experiment was to see the red circle marked on the left side

of the image being placed on the center of the ball. This did not happened with 100% precision rate because the images from the two cameras were not taken at the same exact time and with the fast movement of the ball it is enough to move the ball slightly away from the point marked as a red circle.

This suggests that the transformation of the 2D depth sensor to the 2D RGB sensor is working and with a relatively low margin of error.

(a) Example 1



(b) Example 2

Figure 5.3: Mapping depth ball to RGB ball.

$$R = \begin{bmatrix} 9.99846e^{-01} & 1.26353e^{-03} & -1.74872e^{-02} \\ -1.47790e^{-03} & 9.99923e^{-01} & -1.22513e^{-02} \\ 1.74704e^{-02} & 1.22753e^{-02} & 9.99772e^{-01} \end{bmatrix}$$

Matrix 1: Kinect's Rotation matrix from [4]

$$T = \begin{bmatrix} 1.99852e^{-02} & -7.44237e^{-04} & -1.09167e^{-02} \end{bmatrix}$$

Matrix 2: Kinect's Translation matrix from [4]

Parameter Name	**Parameter Value**
fx_{rgb}	$5.29215e^{+02}$
fy_{rgb}	$5.25563e^{+02}$
cx_{rgb}	$3.28942e^{+02}$
cy_{rgb}	$2.67480e^{+02}$
$k1_{rgb}$	$2.64516e^{-01}$
$k2_{rgb}$	-8.39907e^{-01}
$p1_{rgb}$	-1.99223e^{-03}
$p2_{rgb}$	$1.43719e^{-03}$
$k3_{rgb}$	$9.11924e^{-01}$

Table 5.1: RGB intrinsic parameters from [4].

Parameter Name	**Parameter Value**
fx_d	5.94214e^{+02}
fy_d	5.91040e^{+02}
cx_d	3.39307e^{+02}
cy_d	2.42739e^{+02}
$k1_d$	-2.63864e^{-01}
$k2_d$	9.99668e^{-01}
$p1_d$	-7.62758e^{-04}
$p2_d$	5.03509e^{-03}
$k3_d$	-1.30536e^{+00}

Table 5.2: Depth intrinsic parameters from [4].

5.7 Kinect on the goalkeeper

Since the field-of-view of the Kinect sensor is limited, 58 degrees horizontally and 44 degrees vertically, the spot to place the Kinect sensor on the goalkeeper is an important matter of study. Also the fact that the goalkeeper defend strong shoots forces to place the sensor on a protected zone. To make the Kinect the most shock resistant possible a

metallic structure was used to protect the Kinect body. This way the Kinect can stand on the front of the goalkeeper, the structure is exhibited in a front view on Figure 5.4 and in a side view on Figure 5.5.

The orientation of the sensor will be straight ahead of the robot, aligned with its YY axis so both sides of the Kinect field-of-view are used equally. The problem will be the inclination of the Kinect, with the 44 degrees horizontal are split in 22 degrees to the top side of the plan and another 22 degrees for the bottom side of the plan. This bottom side is not needed because the Kinect is only necessary to detect aerial balls, so placing the Kinect with 20 degrees inclination, ignoring the 2 degrees of the margin, this device will only detect objects above its height. Assuming that the Kinect has a 40 degrees vertical field-of-view the maximum height inside the field-of-view will be calculated using Equation 5.4. In the Table 5.3 is presented the maximum height value when the ball is 1, 2 and 3 meters from the Kinect. The heightOffset is required because Kinect is on the top of the robot and the distance from the Kinect to the ground need to be added.

$$maximumHeight = distance * tan(40^{o}) + heightOffset \tag{5.4}$$

Distance (m)	**Maximum height (m)**
1	0.84 + heightOffset
2	1.68 + heightOffset
3	2.52 + heightOffset

Table 5.3: Maximum height with 40 degrees field-of-view

5.8 Kinect calibration on robot

When a ball is detected by Kinect the ball coordinates are in the Kinect referential, therefore it is important to make a calibration transforming Kinect coordinates into robot's coordinates. This could be done using a manual transformation since the inclination value of the Kinect on the robot and the distance of it to the ground are known, a translation about the Kinect ZZ axis and a rotation about the XX axis would solve the problem. However to avoid measure errors an application was developed to make the calibration. This application consists on retrieving an image from the Kinect depth camera and after the image is captured mark a set of points whose position is known. Then, for each of this marked point insert its coordinates relatively to the robot. Then the coordinates of the

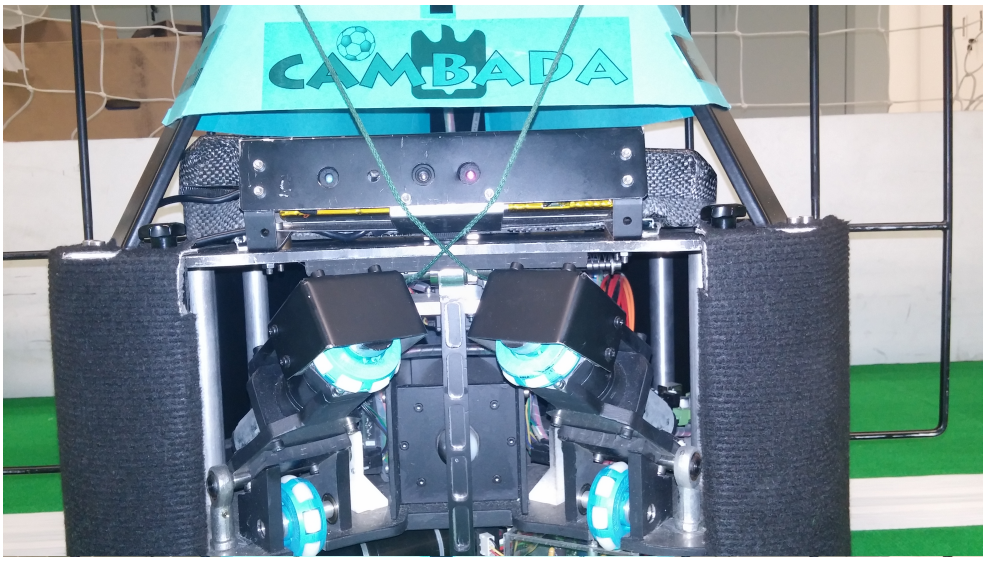


Figure 5.4: Kinect on the goalkeeper-Front view.

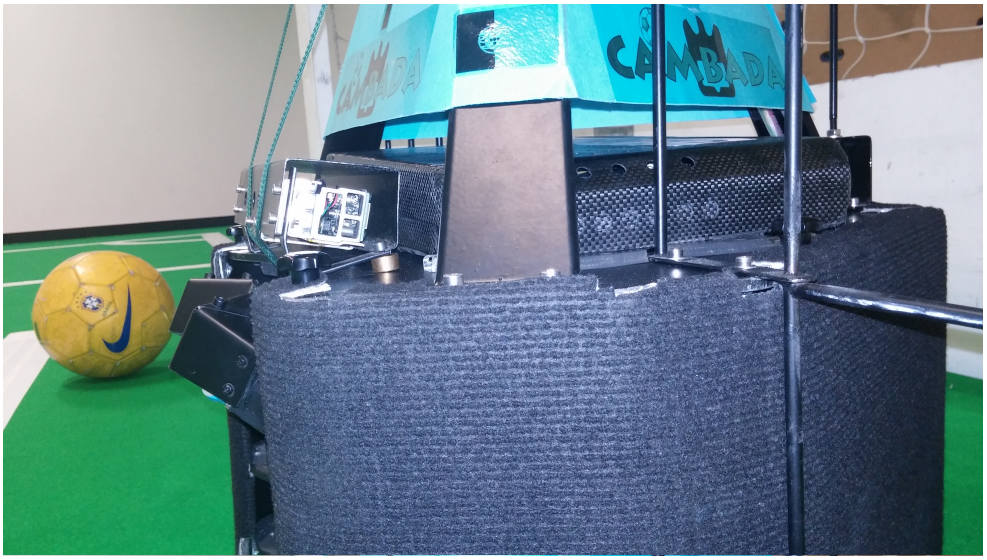


Figure 5.5: Kinect on the goalkeeper-Side view.

two different systems (Kinect and inserted values) are analyzed to find the rotation and translation required to make the rigid transformation of the Kinect plane into the robot's plane. In the Figure 5.6it is presented this calibration application.

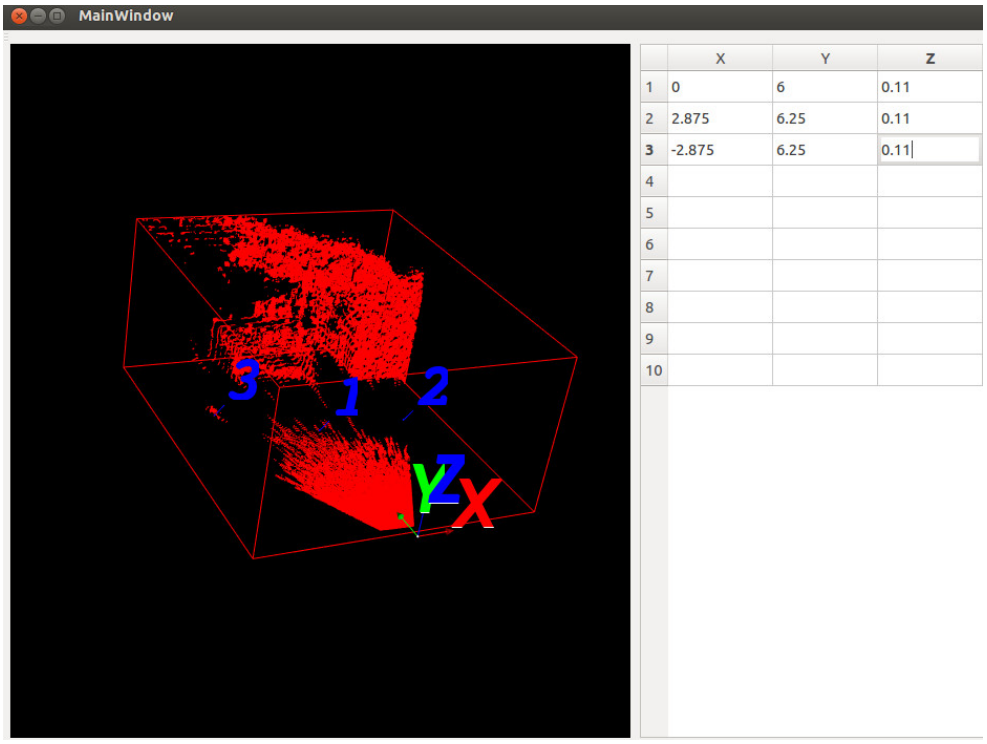


Figure 5.6: Kinect calibration application from [23]. On the left side is captured the image and the points are marked. On the right side the coordinates of each point is inserted.

5.9 Experimental results

In order to implement the 3D vision on the goalkeeper it is required to run some tests to accept the values given by the Kinect and make sure that these values are completely reliable. To check if the algorithm is working as intended and the precision of the detected balls three tests were made.

The first test was the parametrization of the algorithm to find the best combination of voxel and mask sizes for the algorithm presented on section 5.3. In this test, the range and its precision and the processing time per cycle are analyzed .

The second test is related to the precision of detecting a static ball using the values from the parametrization test. This test the precision itself, the absolute deviation of the captured values, the percentage of frames where the ball is detected correctly and the number of frames that a wrong ball is detected.

The last test is detecting the ball during an air trajectory.

5.9.1 Algorithm parametrization

The aim of this test is choosing the right voxel size and mask size combination values for the algorithm presented in section 5.3. The test consists in placing a ball suspended on the air using a string attached to a stick in a setup point at which the distance to the Kinect is known. The setup points used were all at the height (ZZ axis) of 1,2 meters, in front of the kinect camera (x = 0) changing only the distance about the YY axis to the Kinect. The distances used were 2 meters, 3 meters, 4 meters, 5 meters and 6 meters. The Kinect was placed on the goalkeeper, Figure 5.7. After the setup is done the algorithm starts running until the Kinect detects 150 samples (frames) and dumps the position values into a file. This same algorithm will run again, this time without writing anything to the file just to calculate the time of execution of a single frame. This time will vary with the CPU used, but the values were obtained using a laptop with an **Intel®Core texttrademark i5-3230M CPU @ 2.60GHz**[26]. The time is important because Kinect works at 30 frames per second, and the cycle time should not be higher than 33 milliseconds to use the full capacity of 30 frames per second.

After the raw data were all collected the mean values, the standard deviation and the hit percentage, number of frames that the ball was detected within a 1 meter cube with center on the setup point divided by the total number of frames were calculated. These values are presented in table 5.4.

Figure 5.7: Kinect detecting a static ball setup.

The voxel and mask size then were chosen depending on three major factors:

- **Precision** of the detected ball using the mean and standard deviation of all the values.
- **Hit percentage** of the algorithm mostly at longer distances.
- **Cycle time** duration required to capture each frame.

Parameters (voxel _ mask)	**Setup Point x/y/z(m)**	**Mean Values x/y/z(m)**	**Std x/y/z(m)**	**Hit Percentage**
10_7	0.0/2.0/1.2	-0.09/2.02/1.21	0.002/0.003/0.002	100%
	0.0/3.0/1.2	-0.11/2.96/1.22	0.004/0.004/0.004	100%
	0.0/4.0/1.2	-0.07/3.97/1.20	0.002/0.007/0.002	99%
	0.0/5.0/1.2	-0.07/5.04/1.19	0.004/0.015/0.006	99%
	0.0/6.0/1.2	0.01/6.11/1.21	0.002/0.023/0.003	25%
12_5	0.0/2.0/1.2	-0.08/2.01/1.21	0.003/0.002/0.002	100%
	0.0/3.0/1.2	-0.11/2.96/1.21	0.005/0.007/0.004	100%
	0.0/4.0/1.2	-0.07/3.98/1.19	0.004/0.008/0.007	100%
	0.0/5.0/1.2	-0.06/5.05/1.20	0.006/0.015/0.006	97%
	0.0/6.0/1.2	0.05/6.09/1.21	0.004/0.036/0.004	93%
18_5	0.0/2.0/1.2	-0.09/2.02/1.20	0.002/0.002/0.002	100%
	0.0/3.0/1.2	-0.10/2.97/1.21	0.003/0.002/0.003	100%
	0.0/4.0/1.2	-0.07/3.98/1.19	0.005/0.005/0.005	99%
	0.0/5.0/1.2	-0.06/5.06/1.19	0.004/0.011/0.004	90%
	0.0/6.0/1.2	0.01/6.10/1.21	0.004/0.009/0.003	9%

Table 5.4: Detecting static balls using different parameters

Evaluating the data from the table 5.4, the mean values are close to the setup point on every parameter/setup point combination. The variation on the YY axis is the one most relevant because it was subject to measures so the setup point of 5 and 6 meters from the Kinect has measure errors. The standard deviation was in the order of the millimeters in the distance range of 0 to 4 meters and in the order of the centimeters for more than 5 meters away. The highest standard deviation value is 3.6 centimeters about the YY axis when the ball was 6 meters away, which is an acceptable value taking into account the distance. The precision is good enough for every parameter tested. In terms of hit percentage for the 10_7 parameter was always near 100% until 5 meters away, but when tested 6 meters away went down to 25%. The 12_5 parameter was always near 100% until 5 meters away and dropped to 93% at 6 meters away. Lastly, the 18_5 parameter detected

arround 100% until 4 meters away, 90% at 5 meters and only 9% at 6 meters. In terms of hit percentage the 12_5 parameter mode was clearly the best, allowing to detect further.

This already eliminates the 10_7 parameter mode from the choice, because, among the three, it is the most computer processing demanding and the results were not better. It is the one with the lowest voxel size and the highest mask which requires more global voxels and in each voxel iteration it is a 7x7x7 cube. Just for information the cycle time using this mode was 35.4 ms. Analyzing the cycle time of the other two parameter modes, for the 12_5 parameter the mean time of cycle was 26.8 ms while in the 18_5 parameter mode the mean time was 22.3 ms. Both this parameter modes are taking less than the pretended 33 ms using the tested processor so the best parametrization is the 12_5, 12 centimeters is the voxel size and the mask size is 5. This means that the algorithm detect objects up to 12x3 (inner part of the mask) centimeters and more than 12 centimeters away from any other object.

5.9.2 Precision with static ball

The setup of the second test is similar to the first one, the ball suspended by a string and the Kinect in the goalkeeper detecting the balls , figure 5.7, but this test is more accurate than the first one and uses the best parameter from the previous test, section 5.9.1. On this test the distance will be increased due to the good percentage at 6 meters in order to check the range with this parameters. The setup points used will be the same, height of 1.2 meters, 0 in the XX axis and the only axis that changes is the YY. False positives will also be analyzed on this test, corresponding to points detected as a ball that are outside of an one meter side cube with center on the setup point. Lastly will be presented and analyzed a 3D visual representation of the points detected. The results obtained on this test are presented on table 5.5.

Setup Point x/y/z(m)	**Mean Values x/y/z(m)**	**Std x/y/z(m)**	**Hit Percentage**	**False Positives**
0.0/2.0/1.2	0.03/1.98/1.18	0.001/0.004/0.001	100%	0
0.0/3.0/1.2	0.06/3.00/1.19	0.004/0.005/0.002	100%	0
0.0/4.0/1.2	0.06/3.96/1.20	0.004/0.009/0.002	100%	0
0.0/5.0/1.2	0.05/4.94/1.20	0.009/0.012/0.004	97%	0
0.0/6.0/1.2	0.00/5.91/1.19	0.004/0.019/0.005	87%	0
0.0/7.0/1.2	0.03/6.82/1.21	0.007/0.039/0.006	17%	1

Table 5.5: Detecting fixed balls using the best parameters

The results obtained from this test demonstrates that the hit precision is good until 6 meters, which was already known from the previous test 5.9.1, the hit precision slightly decreased from 93% to 87% when the distance in the YY axis is 6 meters, but at 7 meters the decrease is huge, down to only 17%. Thus the maximum distance where the Kinect values can be trusted lies between 6 and 7 meters away.

There are small errors in the mean values compared with the setup point that are related with the fact that the XX and YY axes are measured with a measure tape and there is error associated. The standard deviation is in the order of the millimeters in the XX and ZZ axes and increases slowly with the distance to the Kinect. In the YY axis the standard deviation is higher than in the other axes which is visible in the 3D visualization of the detected points, figure 5.8, also in the XoY plane visualization, figure 5.9 and in the YoZ plane visualization, figure 5.10.

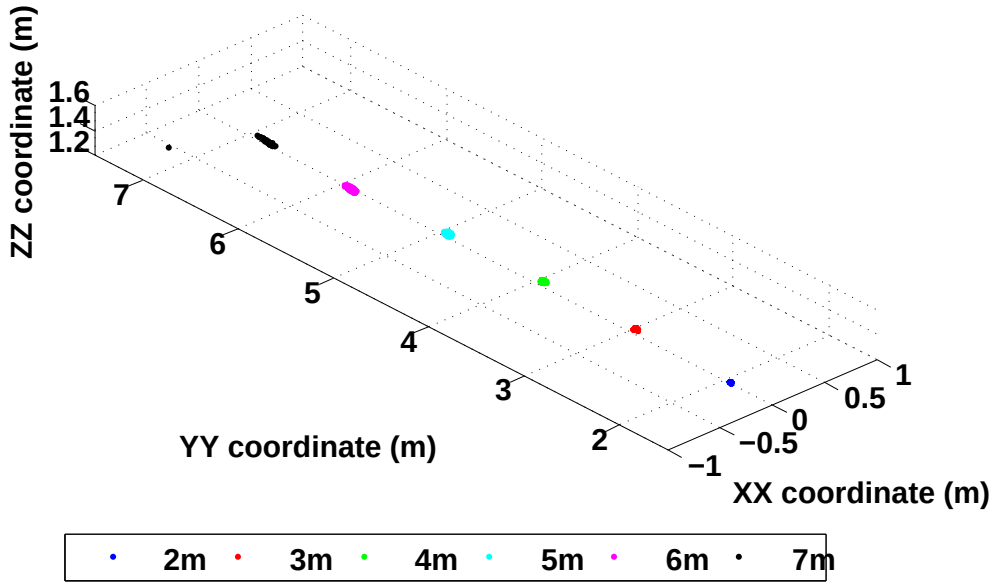


Figure 5.8: 3D visualization of fixed ball points detected.

Finally, it was detected 1 false positive value, that is when a ball is detected outside of an one meter side cube centered on the setup point because this test was done without any color validation of the ball and the structure that was supporting the ball was detected as a ball. In the figure 5.8 is presented a 3D visualization of the points detected in the six different setup points where each setup point is marked with its own color. A top view is also presented on figure 5.9 where the false positive value can be clearly seen as a black point marked in the right bottom side of the referential. That point was detected 1.05 meters to the left of the remaining points (XX coordinate is negative to the left side). The

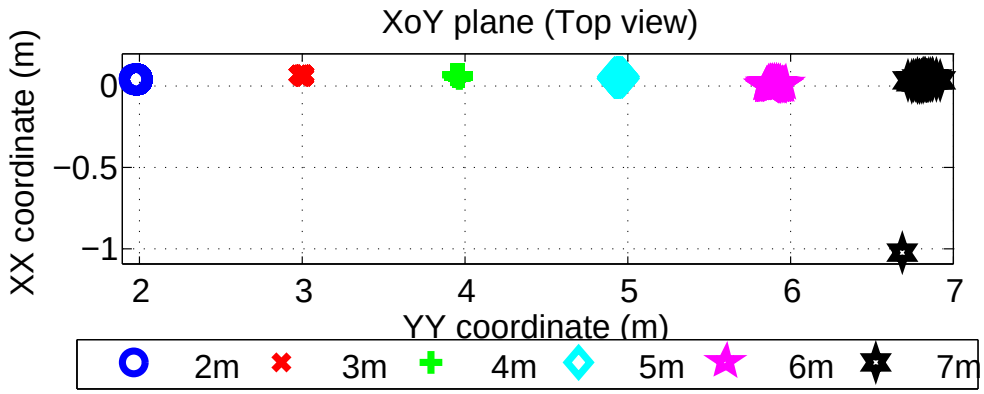


Figure 5.9: XoY plane visualization of fixed ball points detected (top view).

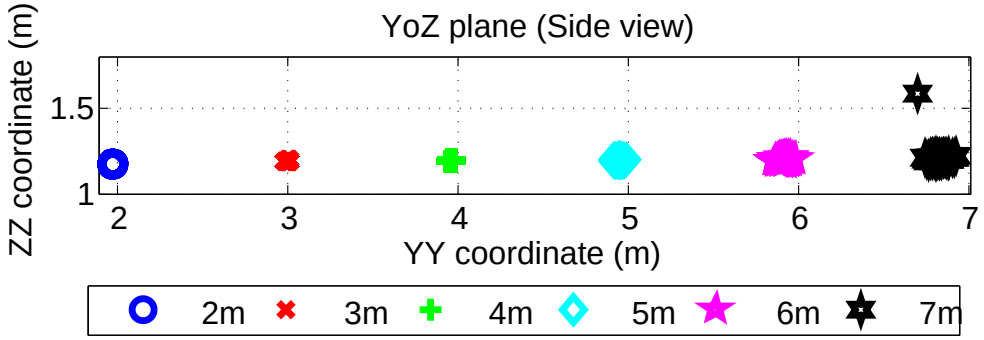


Figure 5.10: YoZ visualization of fixed ball points detected (side view).

side view is presented on figure 5.10 where the false positive point is 0.4 meters higher than any other point.

5.9.3 Detecting ball in movement

During a game the ball is, most of the times,in movement, and find the ball standing still in the air as it was in the test done in the chapter 5.9.2 is physically impossible due to the gravity. Therefore the main goal of the Kinect is to detect the ball while it is moving.

For this test the Kinect was placed on the goal line and a CAMBADA robot was shooting the ball from the other side of the field. The total distance between the shooting robot and the Kinect was eleven (11) meters. As it is impossible for the Kinect to detect an object at 11 meters away it is impossible to track the ball during the entire duration of the shoot. It was decided to track only the last part of the shooting trajectory, the most important part and the one that is needed in a real game. Four different shoots were

tracked each one with a different shooting strength. The aim of this test is to check how many ball positions are detected during the shoot to decide if enough points are detected for the trajectory prediction algorithm. Then calculate the percentage of frames where the ball is detected after the first ball detection. In the table 5.6 it is presented an overview of the data collected with this experiment. In the figure 5.11 is presented the 3D visualization of the trajectory of the shoots. The center of the referential is the shooting point and the Kinect is on the floor at the 11 meters mark.

Shooting strength	**Frames detected**	**Total frames**	**Detection Percentage**
35	9	14	64%
40	12	16	75%
45	10	17	59%
50	16	17	94%

Table 5.6: Detecting ball in movement using the best parameters

For the first shoot, CAMBADA shooting strength of 35, 9 frames were detected over the trajectory and in 5 frames no ball were detected, in 64 percent of the frames the ball was detected. The 3D trajectory is marked as green on the graphic. If all the frames detected the ball the distance between the green points would be the same in the horizontal axis, but since some are lost there is a bigger gap between some points. This shoot ended 2.2 meters away from the Kinect. The second shoot, shooting strength of 40, marked as red in the 3D trajectory graphic 12 frames were detected from a total of 16 frames, which makes 75 percent of the total frames. Since the strength used to make the kick was higher than before the trajectory ended closer to the Kinect than the older one.

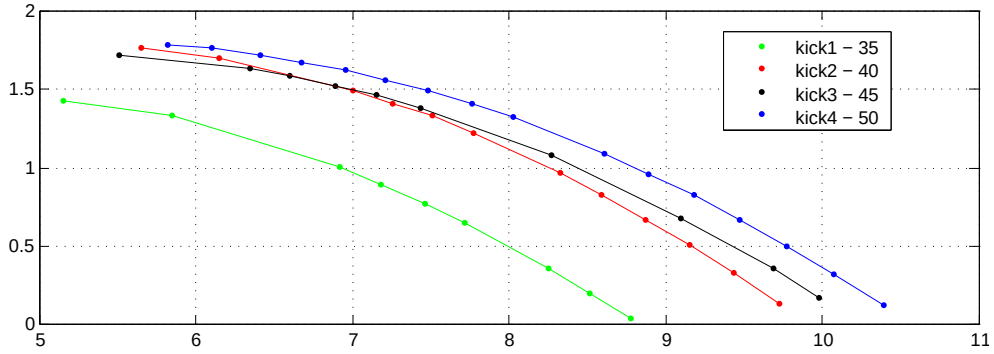


Figure 5.11: Ball in movement 3D trajectory

For the third kick, shooting strength of 45, in 10 frames out of 17 the ball was detected

to make the trajectory. It is the black trajectory on the graphic. Of all the four shoots this is the one with the lowest detection percentage with only 0.59. Lastly, the forth shoot was the one with the better detection with a rate of 94 percent where the ball was detected in 16 out of 17 frames. In this trajectory on the graphic, blue trajectory, the gap between the points detected is constant for almost the entire trajectory. These results makes believe that the points detected on movement are enough to create a trajectory for defending aerial balls. However this test was made in a controlled environment with only one ball to detect and almost no garbage on the Kinect's field-of-view.

Chapter 6

Conclusion and future work

6.1 Conclusion

The main objective of this thesis was to improve CAMBADA goalkeeper software in order to get a better overall behavior.

The problem encountered on the start of this thesis was the low amplitude of movements of the previous goalkeeper and its behavior was the same on every single game. To achieve the proposed goals a new approach was followed, make the goalkeeper more dynamic and make every parameter user configurable, this way the goalkeeper can be slightly adjusted from game to game, depending on the CAMBADA opponents. The new goalkeeper also protects equally both sides of the goal, which was not happening before, and the time to react to shoots on any side of the goal is the same.

The current goalkeeper acts not as a single point on the field but as a real player who has a body and occupies a given area on the field, hence it uses the full potential of its bounds which is extremely important when the goalkeeper is positioning near the goal posts and to avoid colliding with any other object, mainly goal posts.

Still referencing the positioning of the goalkeeper, it was also developed the ability of leaving the goal on defense duty, playing pressure to the attacking opponent. Yet this ability still needs some improvements, but it is already a good start point. Another new capability of the goalkeeper is the fact that the goal position is not static like it was on the previous goalkeeper, the whole goalkeeper receives the goal posts position and all the behavior is dependent of the goal position. The benefits of having this capability cannot be tested yet because there is still no way of calculate these post positions but once it is possible the benefits will be huge.

The tests made to the global positioning of the goalkeeper shows an improvement on the goalkeeper behavior, defending 33% more shoots from a position in front of the goal and 88% more shoots defended when the ball were shoot from the sides of the field. These results were obtained due to the better positioning near the goal posts and to the fact that both sides of the goal are now equally protected. This also makes the goalkeeper occupy more area on the goal from the point of view of the striker both from the front of the goal and from the side. The goalkeeper rotation is now better adjusted, mainly near the goal posts which increases the occupation on the goal having an huge impact while defending shoots from the sides.

Still from the topic of the positioning, the goalkeeper is now ready to use the new functionality of leaving the goal on defense duty leading to and increase from 25% of close shoots defended to 70%. This is extremely important when the goalkeeper is the last robot standing between the ball and the goal.

In terms of the 3D detection of aerial balls, it is already possible and tested in the field. However, it still needs to be used and coordinated with the 2D vision from the CAMBADA robots to make the defense of shoots by air.

6.2 Future Work

Although some great improvements were made on the goalkeeper this is not a finished job, there is still more room for improvements. A detection of the goal either using a laser RangeFinder or making a more accurate analysis of the 2D mechanism vision is still critical to the goalkeeper for a better positioning on the goal and minimize the positioning errors that in the robotics area is an issue that always happens.

The accuracy currently given on whenever the opponent is dribbling the ball or not is still not enough for a precise attack on the opponent when leaving the goal to defend. Even though the results show an increase in defending near shoots while leaving the goal, they can be improved when the opponent dribbling knowledge is more precise.

The last point of the future work is the fact that the goalkeeper still can not defend aerial balls, for future work the usage of the Kinect sensor that was already tested must be merged with the 2D vision mechanism which will improve the overall defending behavior of the CAMBADA goalkeeper.

Bibliography

[1] MSL rules 2014. http://wiki.robocup.org/images/d/d1/Msl-rules2014.pdf. Accessed August 19, 2014.

[2] Ricardo Jorge Maurício Dias. Agent architecture of the CAMBADA robotics soccer team. Master's thesis, University of Aveiro, Portugal, 2014.

[3] Bisector of an angle of a triangle. http://jwilson.coe.uga.edu/emt725/Bisect/bisect.html. Accessed October 6, 2014.

[4] Mapping depth pixels with color pixels. http://nicolas.burrus.name/index.php/Research/KinectCalibration. Accessed May 17, 2014.

[5] R. Dias, F. Amaral, J. L. Azevedo, R. Castro, B. Cunha, J. Cunha, P. Dias, N. Lau, C. Magalhães, A. J. R. Neves, A. Nunes, E. Pedrosa, A. Pereira, J. Santos, J. Silva, and A. Trifan. CAMBADA'2014: Team Description Paper 2014. In *CD Proceedings of RoboCup Symposium 2014*, João Pessoa, Brazil, 2014.

[6] IEETA page. http://wiki.ieeta.pt/wiki/index.php/Main_Page. Accessed August 2, 2014.

[7] Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, and Eiichi Osawa. RoboCup: The robot world cup initiative. In *Proc. of The First International Conference on Autonomous Agent (Agents-97)*, pages 340–347, Marina del Rey, 1997.

[8] RoboCup Objectives. http://www.robocup.org/about-robocup/objective/. Accessed August 2, 2014.

[9] RoboCup Junior League. http://www.robocup.org/robocup-junior/. Accessed August 2, 2014.

[10] RoboCup @Home League. http://www.robocup.org/robocup-home/. Accessed August 2, 2014.

[11] RoboCup Rescue League. http://www.robocup.org/robocup-rescue/. Accessed August 2, 2014.

[12] RoboCup Soccer League. http://www.robocup.org/robocup-soccer/. Accessed August 2, 2014.

[13] CAMBADA General Architecture. http://robotica.ua.pt/CAMBADA/robots.htm#descr. Accessed August 25, 2014.

[14] 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 *Computer and Information Sciences – ISCIS 2004: 19th International Symposium, Proceedings*, volume 3280 of *Lecture Notes in Computer Science*. Springer, 2004.

[15] António J. R. Neves, José Luís Azevedo, Bernardo Cunha, Nuno Lau, João Silva, Frederico Santos, Gustavo Corrente, Daniel A. Martins, Nuno Figueiredo, Artur Pereira, Luís Almeida, Luís Seabra Lopes, Armando J. Pinho, João Rodrigues, and Paulo Pedreiras. CAMBADA soccer team: from robot architecture to multiagent coordination. In *Robot Soccer*, chapter 2, pages 19–45. INTECH, 2010.

[16] Goalkeeper basic positioning and techniques. http://www.grassrootscoaching.com/members/page.phtml?id=276. Accessed August 19, 2014.

[17] Goalkeeper basic positioning. http://www.jbgoalkeeping.com/position.html. Accessed August 19, 2014.

[18] Tech United 3D ball detection using Kinect. http://www.techunited.nl/wiki/index.php?title=3D_Ball_Recognition_with_Kinect. Accessed August 19, 2014.

[19] F.B.F. Schoenmakers, G. Koudijs, C.A. Lopez Martinez, M. Briegel, H.H.C.M. van Wesel, J.P.J. Groenen, O. Hendriks, O.F.C. Klooster, R.P.T. Soetens, and M.J.G. van de Molengraft. Tech United Eindhoven Team Description - Middle Size League. In *CD Proceedings of RoboCup Symposium 2013*, 2013.

[20] Zhiwen Zeng, Dan Xiong, Qinghua Yu, Kaihong Huang, Huimin Lu Shuai Cheng, Xiangke Wang, Hui Zhang, Xun Li, and Zhiqiang Zheng. NuBot Team Description Paper - Middle Size League. In *CD Proceedings of RoboCup Symposium 2013*, 2013.

[21] Amir A.F. Nassiraei, Shuichi Ishida, Noriyuki Shinpuku, Miyuki Hayashi, Naoya Hirao, Kazunori Fujimoto, Kazutaka Fukuda, Kazutomo Takanaka, Ivan Godler, Kazuo Ishii1, and Hiro. Hibikino-Musashi Team Description Paper - Middle Size League. In *CD Proceedings of RoboCup Symposium 2014*, 2014.

[22] António José Neves, Bernardo Cunha, and Paulo Dias. Detection of flying objects using Kinect. Technical report, IEETA, 2013.

[23] António José Neves, João Silva, Paulo Dias, and Rafael Castro. Detection of aerial balls using a Kinect sensor. In *Robot Soccer World Cup XVIII*, Lecture Notes in Artificial Intelligence. Springer, 2015.

[24] Eigen - C++ linear algebra template. `http://eigen.tuxfamily.org/index.php?title=Main_Page`. Accessed October 21, 2014.

[25] Daniel Herrera C. and Janne Heikkila. Joint depth and color camera calibration with distortion correction. *IEEE Transactions on pattern analysis and machine intelligence*, 34(10):2058–2064, October 2012.

[26] Processor used on kinect algorithm tests. `http://ark.intel.com/products/64896/Intel-Core-i5-3320M-Processor-3M-Cache-up-to-3_30-GHz`. Accessed November 16, 2014.