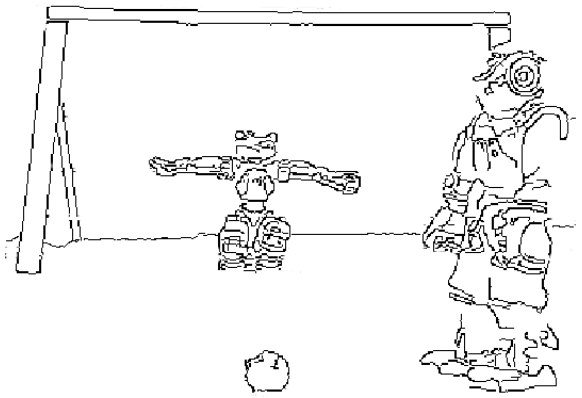
Universidade de Aveiro
Departamento de
Electrónica, Telecomunicações e Informática.
2011

Alina Liliana
Trifan

Desenvolvimento de um sistema de visão para robôs humanoides

Development of a vision system for humanoid robots



Alina Liliana
Trifan

Desenvolvimento de um sistema de visão para robôs humanoides

Development of a vision system for humanoid robots

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requesitos necessários à obtenção do grau de Mestre em Engenharia Electrónica e Telecomunicações, realizada sob a orientação científica do Doutor António José Ribeiro Neves, Professor Auxiliar Convidado do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro e do Doutor Manuel Bernardo Salvador Cunha, Professor Auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro.

o júri / the jury

presidente / president

Armando José Formoso de Pinho
professor associado da Universidade de Aveiro (por delegação da Reitora da Universidade de Aveiro)

vogais / examiners committee

Alexandre José Malheiro Bernardino
professor auxiliar do Instituto Superior Técnico da Universidade Técnica de Lisboa

Manuel Salvador Bernardo Cunha
professor auxiliar da Universidade de Aveiro (co-orientador)

António José Ribeiro Neves
professor auxiliar convidado da Universidade de Aveiro (orientador)

agradecimentos / acknowledgements

É com muito gosto que aproveito esta oportunidade para agradecer aos meu orientadores António Neves e Bernardo Cunha por todo o apoio, paciência e disponibilidade que me mostraram durante o último ano. Agradeço também aos meus pais que nunca deixaram de acreditar em mim e que sempre me apoiaram. Por fim um obrigado muito especial ao H.B. por me ter aturado nos meus piores momentos e por ter celebrado comigo todos os meus pequenos sucessos.

With all my gratitude and appreciation I would like to thank my supervisors António Neves and Bernardo Cunha for all the suport, patience and availability that they showed me during the last year. I would also like to thank my parents who always believed in me and gave me the strength to continue. Last but not least, a special thank you to H.B. who had to endure all my panic attacks in the moments of crysis and who also knew how to celebrate with me even the smallest achievements.

Resumo

Tanto para os humanos como para os robôs, a visão é um sentido muito importante e que permite a interpretação do espaço a mais do que uma dimensão. No caso de robôs humanoides, cuja forma se assemelha a um humano, a utilização de visão é um enorme desafio devido às restrições que derivam da diferença entre o corpo humano e o robô. Um sistema de visão robusto deve ser capaz de disponibilizar informação precisa sobre o ambiente e uma correcta representação dos objectos de interesse. Esta dissertação apresenta uma solução para um sistema de visão de um robô humanoide. Os algoritmos desenvolvidos para todos os módulos do sistema, desde a aquisição da imagem até ao seu processamento e detecção dos objectos de interesse foram testados num robô humanoide desenvolvido para jogar futebol. Neste tipo de aplicação, o mundo envolvente ao robô é simplificado, havendo um conjunto de cores com significado especial neste contexto. Nesta aplicação em particular, a correcta detecção em tempo real de linhas brancas num campo, balizas amarela e azul bem como bolas laranja, mostram a eficiência do sistema proposto. No entanto, o trabalho desenvolvido nesta dissertação pode facilmente ser exportado para outras plataformas humanoides com alterações mínimas, como apresentado nos resultados experimentais desta dissertação. Para além dos algoritmos referidos, esta dissertação apresenta um conjunto de aplicações que podem ser executadas num computador externo ao robô, desenvolvidas para uma melhor manipulação das imagens adquiridas pela plataforma robótica e para efeitos de depuração dos algoritmos.

Abstract

For both humans and robots vision is a very important sense that has the task of interpreting spatial data, indexed by more than one dimension. In the case of a humanoid robot, that is a robot whose structure imitates the human body, vision is a very challenging area because of all the restrictions that come from the differences between the human body and the robotic one. A robust vision system should be able to provide accurate information about the environment and a precise description of the objects of interest. This thesis presents a solution for an accurate implementation of a vision system for a humanoid robot. From acquiring images, processing them and detecting the objects of interest all the algorithms have been tested on a soccer playing humanoid robot. For a soccer playing robot the world is simplified to a number of colors that are meaningful in this context. In this particular case, the algorithms for real-time detecting the white field lines, the blue and yellow goals and the orange ball have proven their reliability. The work implemented can be easily exported to other humanoid platforms, as presented in the experimental results of this thesis. Moreover, this thesis presents several applications that can run on an external computer and that have been created for a better manipulation of the images acquired by the robotic platform and for debugging purposes.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

For both humans and robots vision is a very important sense, the latter representing the point of interest of this project. In the robotic world many different sensors can be used, but usually, the vision system is the main sensorial element as it is capable to provide a great amount of information such as spacial, temporal and morphological. Based on the robot purpose, the vision system has to perform different tasks. The majority of the systems must perform object recognition, object learning and localization perception. An efficient robotic vision system should also be able to perform its tasks in real time, which means that several constraints are imposed. First of all, real time applications imply that algorithmic complexity is limited, allowing thus a small processing time.

One of the most interesting and popular branches of robotics nowadays is robotic soccer. For a soccer playing robot vision plays probably the most important part. In robotic soccer, the environment is always changing, the ball and the robots are always moving, most of the time in an unpredictable way. The vision system is responsible for capturing all these fast changing scenes, processing them and taking valid decisions in the smallest possible amount of time, thus allowing real-time operations.

This thesis focuses on the development of a modular vision system for a humanoid robot. The work that will be presented has been applied to the humanoid robots of the Portuguese Team [7] competing in the RoboCup Standard Platform League [8]. The goal of the project was the development of a generic vision system for humanoid robots, that is a vision system that could be used with a wide range of humanoid robots. A platform for tests was needed and the first and most used available platform was the robotic soccer player NAO robot [1]. However, with a small number of adjustments, the same vision system has been successfully applied to a Bioloid humanoid robot [9]. This thesis focuses on the implementation of the vision system for the NAO robot, with emphasis on its modularity. One section will also be dedicated to the usage of the proposed video system with the Bioloid robot in the chapter of the experimental results.

1.1 Objectives

The aim of this project is finding a solution for overcoming all the challenges and restrictions that robotic vision has to face and presenting an accurate implementation of several algorithms for detecting objects of interest for a humanoid soccer-playing robot. The specific goals of this thesis are:

- Study of already existing algorithms for implementing a robotic vision system.
- Development of algorithms that can be applied to the humanoid platforms that the University can provide.
- Integration of the proposed algorithms with the given platforms.
- Testing of the developed algorithms.

1.2 Humanoid Robots

Human-friendly robotics is nowadays more and more concerned about providing a variety of mechanically constructed solutions for some of the most common daily tasks of a human being. One of the many branches of robotics that best accomplishes the task of imitating human behaviours is humanoid robotics.

A humanoid robot is a robot whose overall appearance is very similar to the human body, thus allowing it to interact with objects and environments that were initially designed exclusively for human usage. The body of the most humanoid robots consists of a head, a torso, two legs and two arms. There are also several models of robots whose bodies include a face with eyes and even a mouth. The idea behind the construction of a humanoid robot is to imitate different physical and mental tasks that humans undergo daily.

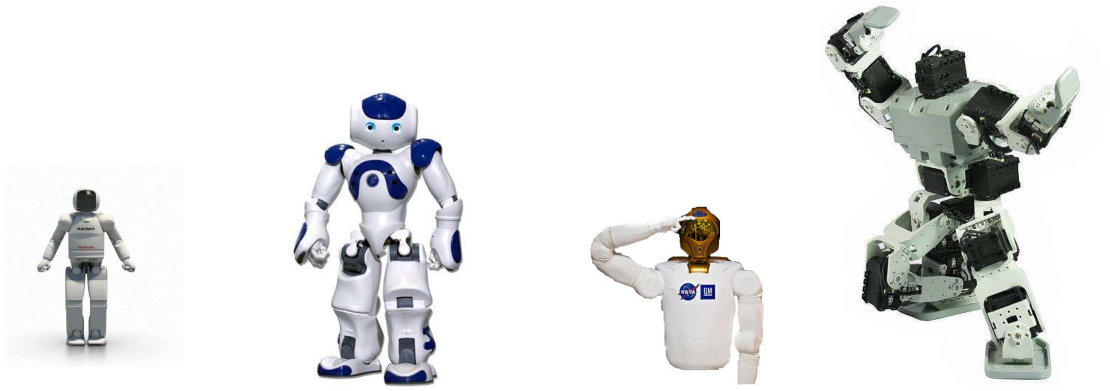
A humanoid robot that is fully autonomous should be able to:

- Gather information about the environment.
- Work for an extended period without human intervention.
- Move either all or part of itself throughout its operating environment without human assistance.
- Avoid situations that are harmful to people or itself unless those are part of its design specifications.

Probably the most important feature of a humanoid robot is that it should be capable of learning new strategies for adapting to previously unknown situations in order to accomplish its tasks, as well as to cope with changing surroundings. Being a great source of information,

vision usually is the main sensorial element of the robot [10]. Through their vision system, robots are able to sense the environment, to classify objects of interest and to continuously learn new techniques for adapting to all the changes that might occur in the surrounding world. Despite their autonomy while accomplishing given tasks, a humanoid robot, just like any other machine, requires regular maintenance.

Humanoid robots are being used nowadays in a large number of scientific research branches. Among the most interesting and innovative examples is probably robot NAO [1] (Fig. 1.1 (b)) a soccer playing robot, fully autonomous developed by Aldebaran Robotics. ASIMO (Fig. 1.1(a)), the world's most advanced humanoid robot [11] is designed to operate in the real world, where people need to reach for things, pick things up, navigate along floors and even climb stairs. Last but not least, Robonaut2 [12] (Fig. 1.1(c)) is the first robot to enter Earth's orbit on a NASA space shuttle. These three examples only represent an infinitesimal percent of what humanoid robotics stands for. It is a very wide research area through which technology can bring imagination to life.



(a) ASIMO (b) NAO (c) Robonaut2 (d) Bioloid

Figure 1.1: Several humanoid robots.

1.3 Robotic vision in soccer applications

For any soccer robot the objects of interest are: the ball, the field lines, the goals, the team members and of course the obstacles they might encounter on the field as well as the opponent team's members.

In humanoid robotics, the implementation of a vision system is yet restricted by several other elements, out of which one of the most important is the lack of stability of the digital camera due to the type of its locomotion. Another restriction comes from the limitation of the energetic autonomy and processing capabilities. These issues, along with the rest of the challenges that image processing implies, make humanoid vision a complex field of study.

Even though in the case of a robotic soccer player, the representation of a vision system might seem simplified, on a first approach, because of the color codification of the objects of interest, the things are not that simple. Figure 1.2 represents an example of what a robot "sees" during a game. However, this would be an "ideal" situation, in which its vision is not affected by its locomotion. Such frames are usually captured while the robot is not moving any parts of its body.

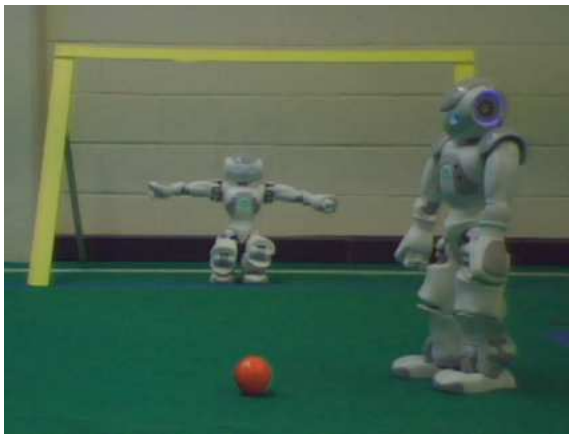


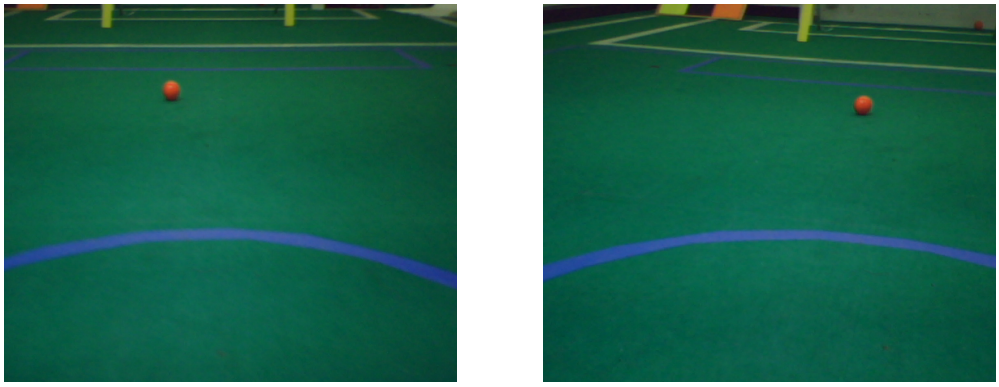
Figure 1.2: An example of an image acquired by the NAO camera during a game, representing what the robot "sees".

Figure 1.3 presents a more common acquisition of images during a soccer game. That is, apart from the objects that can be found outside the field but that are still included in the robot's representation of the surrounding world, the robot has to deal with the instability of the camera while walking. The surrounding objects can be a source of false positives in what concerns the detections of the objects of interest while the movements of the camera influence the image acquisition process.

The research work done in the field of robotic vision is an ongoing process since providing an accurate digital representation of the surrounding world is a very complex task when there is not any human sense to rely on. Moreover, with every year that passes, restrictions in the area of robotic soccer are lifted, this meaning that the environment is becoming more and more generic, without having the reliability of color information and the one of the spacial constraint.

1.4 Portuguese Team

Portuguese Team represents the joint effort of two Portuguese universities, University of Porto and University of Aveiro, to build a new research oriented and competitive SPL team. The project started in 2010 and it is composed of members of the three Portuguese teams that achieved best results in RoboCup [13] European and world championships: FC Portugal [14],



(a) (b)

Figure 1.3: Two consecutive frames acquired while the robot is moving. Due to its locomotion, consecutives frames acquired by the camera of the robot might represent different views of the surrounding world.

CAMBADA [15] and 5DPO [16].

FC Portugal is a joint project of the Universities of Porto and Aveiro in Portugal. The team participates in several RoboCup competitions since 2000, including: Simulation 2D, Simulation 3D (humanoid simulation), Coach Competition, Rescue Simulation, Rescue Infrastructure and Physical Visualization/Mixed Reality. The team research is mainly focused on creating new coordination methodologies such as tactics, formations, dynamic role exchange, strategic positioning and coaching. The research-oriented development of FC Portugal has been pushing it to be one of the most competitive over the years (Simulation 2D World champion in 2000 and European champion in 2000 and 2001, Coach Champion in 2002, 2nd place in 2003 and 2004, Rescue Simulation European champion 2006, Simulation 3D European champions in 2006 and 2007 and World Champions in RoboCup 2006).

CAMBADA is the Middle-Size League RoboCup team and project from IEETA/University of Aveiro that started in 2003. The team main research interests cover most of the Middle-Size League challenges such as robot vision, sensor fusion, multi-agent monitoring, and multi-robot team coordination. The team won RoboCup 2008, achieved 3rd place in RoboCup 2009 and RoboCup 2010 World MSL competitions and was 2nd in the RoboCup German Open 2010. Moreover, for the last 5 years it has been the national champion of the MSL.

5DPO is a project of INESC-P/FEUP aiming at researching in topics such as vision based self localization, data fusion, real-time control, decision and cooperation. The team is active in RoboCup since 1998 in the Small-Size and Middle-Size leagues. The team results have been quite good in the Small-Size League. 5DPO won three European/GermanOpen Small-Size League championships and achieved a 3rd place at RoboCup 1998 and a 2nd place at RoboCup 2006 world championships in this league.

Portuguese Team attended the first robotic competition in March 2011 in Rome, Italy

(RoboCup Mediterranean Open [17]) and will also attend RoboCup 2011 in Istanbul, Turkey. As a first approach and for testing purposes, the work presented in this thesis has been applied to the humanoid robots of the Portuguese Team.

Figure 1.4: Logo of the Portuguese Team.

1.5 Contribution and structure of the thesis

As stated in Section 1.4 the work described in this thesis has been applied in the context of the robotic soccer and all the algorithms that will be presented in Chapter 4 have been tested on the NAO humanoid robots of the Portuguese Team. Even though the goal of the project was to develop a generalized vision system for a wide range of humanoid robots, it was imperative to have an environment on which the work could be tested on. The algorithms that was implemented had good results on the NAO robots and it can be easily exported to other humanoid platforms, with a minimum number of changes. In the case of the soccer playing robot NAO, the vision system is able to acquire good quality images by calibrating the intrinsec parameters of the camera according to the environment. In the context of soccer games, the robot can detect the ball and the goals as well as the lines of the field. Moreover, the software was also used with a few adjustements with a Bioloid robot, which is a proof of the modularity of the proposed vision system.

The thesis is structured in six chapters and two appendixes, each of them aiming to explain different issues that have been approached in the development of the project. The first chapter was an introductory one, stating the objectives of the thesis and presenting the fundamental elements of the project. In this regard, there was presented an overview on the state of the art humanoid robots and also on the issues that robotic vision has to overcome nowadays. Moreover, a presentation of the SPL team Portuguese Team has been included since it provided the environment for developing and testing of the work described.

The second chapter gives a more detailed description of what humanoid robotic soccer means and also focuses on the work of the most important teams participating in RoboCup. Mainly their vision systems were emphasized. Chapter 3 presents an overview of the vision

system architecture and focuses on two support applications that were developed: NaoViewer (Section 3.3) and NaoCalib (Section 3.2). The first one, NaoViewer is a tool used for monitorizing and debugging purposes. It allows the user to follow in real-time the results of the algorithms that run on the robot. NaoCalib is a very helpful tool in the process of calibrating the colors of interest. The fourth chapter is dedicated to the description of the implementation of the main vision process that runs on the robot. The vision process can be divided into three modules which are: access of the device, calibration of the camera parameters and image acquisition and last but not least image processing which envolves color segmentation and object detection. The details of the implementation of all three modules are presented.

Chapter 5 present the results of the implemented algorithms and describes the usage of the developed vision system in a Bioloid [9] platform and finally, Chapter 6 concludes the thesis, stating also several issues that might be approached in future work developments.

Appendix A presents the prototypes of all the methods that have been developed for the several modules of the vision system, while Appendix B represents the user's manual of the applications, both the vision process running on the robot as well as the support applications.

Chapter 2

Humanoid Soccer

RoboCup is an international initiative that fosters research in robotics and artificial intelligence, on multi-robot systems in particular, through competitions like RoboCup Robot Soccer, RoboCup Rescue, RoboCup@Home and RoboCupJunior. RoboCup is designed to meet the need of handling real world complexities, though in a limited world, while maintaining an affordable problem size and research cost. RoboCup offers an integrated research task covering the broad areas of artificial intelligence and robotics. Such areas include: real-time sensor fusion, reactive behavior, strategy acquisition, learning, real-time planning, multi-agent systems, context recognition, vision, strategic decision-making, motor control, intelligent robot control, and many more.

The ultimate goal of the RoboCup initiative is stated as follows:
"By mid-21^{st} *century, a team of fully autonomous humanoid robot soccer players shall win the soccer game, comply with the official rule of the FIFA, against the winner of the most recent World Cup."*

Such a goal might sound overly ambitious given the state of the art technology. Building humanoid soccer players requires an equally long period of time as well as extensive efforts of a broad range of research areas. Most probably the goal will not be met in any near term, however it is important that such a long range goal be claimed and pursued.

The main focus of the RoboCup competitions is the game of soccer, where the research goals concern cooperative multi-robot and multi-agent systems in dynamic adversarial environments. One of the most popular soccer league in RoboCup is the Standard Platform League (SPL). In this league all teams use identical, standard robots which are fully autonomous. Therefore, the teams concentrate on software development only, while still using state-of-the-art robots. Omnidirectional vision is not allowed, forcing decision-making to trade vision resources for self-localization and ball localization. The league replaced the highly successful Four-Legged League, based on Sony's AIBO dog robots [18], and is now based on Aldebaran's Nao humanoids.

In SPL, robots play on a field with a length of $6m$ and a width of $4m$, covered with a

green carpet. All robot-visible lines on the soccer field (side lines, end lines, halfway line, centre circle, corner arcs, and the lines surrounding the penalty areas) are $50mm$ in width. The centre circle has an outside diameter of $1250mm$. In addition to this, the rest of the objects of interest are also color coded. The official ball is a Mylec orange street hockey ball. It is $65mm$ in diameter and weights 55 grams. The field lines are white and the two teams playing can have either red or blue markers. The red team will defend a yellow goal and the blue team a sky-blue goal. Figure 2.1 shows an image of a typical SPL game.

Figure 2.1: Image taken during a SPL game - the RoboCup 2010 final between the B-Human and rUNSWift teams.

The game is divided into two halfs, each being 10 minutes long, with an interval of 5 minutes between the two halfs. The only accepted human intervention during the game is the one of the referee. The decisions taken by the referee are transmitted to the robots by wireless, by means of an application called *Game Controller*. This application implements the six states in which the robot can be: *initial, ready, set, play, penalty* and *stop*. For the teams that cannot properly communicate with the *Game Controller*, the positioning and communication with the robots is done manually by the referee, who communicates his decisions to the robots with the help of the chest button and the feet bumps.

After booting, the robots are in their *initial* state. In this state, the button interface for manually setting the team color and whether the team has kick-off is active. The robots are not allowed moving in any fashion besides initially standing up. Pressing the left foot bump sensor will switch the team color. Shortly pressing the chest button will switch the robot to the *penalized* state.

In the *ready* state, the robots walk to their legal kick-off positions. They remain in this state, until the head referee decides that there is no significant progress anymore (after a

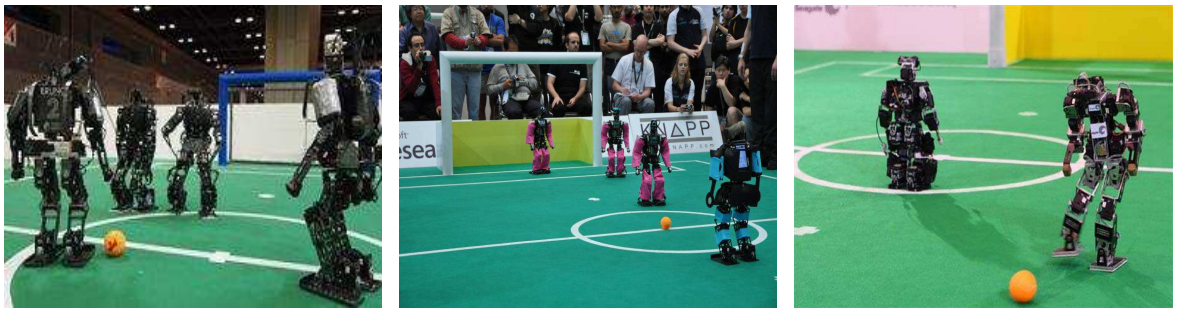
maximum of 45 seconds). This state is not available if only the button interface is implemented.

In the *set* state, the robots stop and wait for kick-off. If they are not at legal positions, they will be placed manually by the assistant referees. They are allowed to move their heads before the game (re)starts but are not allowed moving their legs or locomote in any fashion. This state is not available if only the button interface is implemented. Robots that do not listen to the *Game Controller* will be placed manually. Until the game is (re)started, they are in the *penalized* state.

In the *playing* state, the robots are playing soccer. Shortly pressing the chest button will switch the robot to the *penalized* state. A robot is in the *penalized* state when it has been penalized. It is not allowed moving in any fashion, i. e. also the head has to stop turning. Shortly pressing the chest button will switch the robot back to the *playing* state.

This *finished* state is reached when a half is finished. This state is not available if only the button interface is implemented.

Another league based on humanoid robots in the RoboCup competition is the Humanoid League. The main difference between the SPL and the Humanoid League is that in the latter teams can participate with their own robots, there is not a standard robot that has to be used. Just like in the SPL, in the Humanoid League autonomous robots with a human-like body plan and human-like senses play soccer against each other. Dynamic walking, running, and kicking the ball while maintaining balance, visual perception of the ball, other players, and the field, self-localization, and team play are among the many research issues investigated in the league. This league is divided in three subleagues, according to robot sizes:
Kid Size (30-60cm tall), Teen Size (100-120cm tall) and Adult Size (over 130cm). Examples of the robots used in these competitions can be seen in Fig. 2.2.



(a) Adult Size (b) Teen Size (c) Kid Size

Figure 2.2: Several humanoid robots competing in the RoboCup Humanoid League.

2.1 NAO Robots

The project NAO, launched in early 2005, is a humanoid robot developed by Aldebaran Robotics. Aldebaran Robotics has chosen to make NAO's technology available to any higher

education program. Throughout the program "NAO for Education" institutions of higher education all over the world can purchase NAO robots to special prices for education and research purposes. As a result, NAO is at the moment the most used humanoid robot for academic purposes worldwide. From simple visual programming to elaborate embedded modules, the versatility of the NAO enables users to explore a wide variety of subjects at whatever level of complexity.

In July 2007 NAO (version H1/H25 - Fig. 2.3(c)) was nominated as the official platform for the standard league by RoboCup Organizing Commitee, and successor to Sony's Aibo robot dog [18]. NAO's first participation at the RoboCup occured in July 2008 at Suzhou, China where 15 universities each used 2 robots per team competing in soccer matches. By 2009 almos 100 NAO competed at the RoboCup held in Graz, Austria. 24 teams (4 NAO per team) made full use of the physical and cognitive capabilities of the robot.

(a) NAO T2 (b) NAO T14 (c) NAO H1/H25

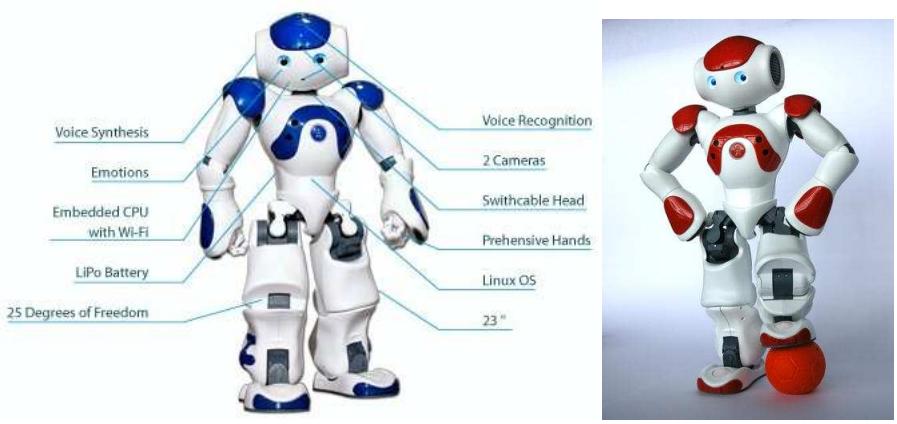
Figure 2.3: Several NAO platforms developed by Aldebaran Robotics.

More recently, the fully autonomous NAO robots (Fig. 2.5(c)) also started being used in projects that develop behaviors for humanoid robots to be used in the environment of a home [19]. In this environment, the robot has to perform specific tasks like bringing a can of soda from the fridge to a human user. Moreover, in March 2011 Aldebaran robotics launched a larger version of the NAO, the so called robot Romeo (Fig. 2.4) designed exclusively for the use at home.

The NAO robot used in the SPL (Fig. 2.5(a),(b)) has 57 centimeters of height and 4.5 kilograms. It comes equipped with a x86 AMD GEODE 500MHz CPU processor and 256 MB SDRAM / 2 GB flash memory. The robot was initially designed for entertainment and has mechanical, electronic, and cognitive features.

Reserved for the research and teaching fields, NAO Academics Edition is delivered with a complete SDK that naturally includes a description of the programming methods, examples

Figure 2.4: A computer-generated rendering shows Romeo doing chores at a home [1].



(a) (b)

Figure 2.5: Aldebaran NAO humanoid robots used in RoboCup SPL.

and the appropriate compilation and debugging tools. Apart from this, NAO Academics Edition has another standard feature called Choreographe, a software that allows NAO users to interact with the robot in a very simple manner.

NAOqi is a framework property of Aldebaran Robotics that runs on the robots and acts as a server. Different modules can plug into NAOqi either as a library or as a broker, with the latter communicating over IP with NAOqi. It supplies binding for C, C++, Python, Ruby and Urbi. NAOqi in NAO is automatically launched at startup if NAO is connected to the network. NAOqi itself comes with several notable modules designed to ease the interaction between the user and NAO. Such modules are: ALMemory, ALMotion, ALLeds, ALSonar, ALRobotPose, ALVideoDevice, ALVisionRecognition, ALVisionToolbox, etc. The names of the modules are generally relevant to their application, therefore only the ones related to vision will be presented here as follows.

ALVideoDevice is a module that allows a direct access to raw images from video source or an access to images transformed in the requested format. ALVisionRecognition is a module which detects and recognizes learned pictures, like pages of a comic book, faces of objects are even locations. The ALVisionToolbox contains several different vision functionalities, like picture taking, video recording, white balance setting. However, even though these modules were used in the first steps of this project, in its final form they are no longer used due to real time constraints, as it is described in Section 4.1.

From the point of view of the hardware most related to the developed vision system, NAO has 2 identical video cameras that are located in the forehead, respectively in the chin area. They provide a 640×480 resolution at 30 frames per second and the field of view is 58 degrees (diagonal). They can be used to identify objects in the visual field such as goals and balls, and bottom camera can ease NAO's dribbles. The native output of the camera is YUV422 packed and only one camera can be used at one time, due to the fact that the two cameras share the same bus and both drivers will be loaded into the "videodev0" kernel module.

Figure 2.6: A RGB conversion of the raw image sent by the camera.

2.2 SPL Teams

This section focuses on the most important projects in the RoboCup SPL, with emphasis on their vision systems. The information presented in this section is according to the Team Description papers of the teams that attended RoboCup in 2010.

2.2.1 B-Human

B-Human [20] is a collegiate project at the Department of Computer Science of the University of Bremen and the DFKI Research Area Safe and Secure Cognitive Systems. The goal of the project is to develop suitable software in order to participate in several RoboCup events. They participated during several years in the humanoid league until they joined the Standard Platform League in 2008. In 2010 they won both German Open and RoboCup worldcup.

Their software is based on 3 concurrent processes that run at frequency determined by the limitations of the robot's architecture. Each process represents a module of their software. The modules are: Cognition, Motion and Debug. The module related to vision is the Cognition module, which runs at a frequency of 30Hz since the camera provides 30 frames per second. This module receives camera images from Video for Linux [21] and also sensor data from the Motion module which will be used for the localization of the robot. Their vision system is constructed based on the OpenCV image processing library.

The process Cognition is structured into three funcional units: perception, modeling and behavior control [22]. The perception system is based on vertical scan lines which means that the actual amount of scanned pixels is much smaller than the image size. The modeling unit is responsible for providing an estimation of the world state, which includes the robot's position, the ball's position and velocity and the presence of obstacles. The first step of the perception unit is to provide a contour of the body of the robot. The robot being white might be easily confused with the white lines of the field. By using forward kinematics the robot knows where its body is visible in the camera image and exclude these areas from image processing.

After this, the image is processed in three steps: segmentation and region building, region classification and feature extraction. First the field borders are detected by running scan lines that start from the horizon downwards until a green segment of a minimum length is found. Next to the usual scan lines there are also used some scan lines that only recognize orange regions. For the goal detection two special scans are done to detect vertical and horizontal yellow or blue segments above the horizon. Region classification is used for establishing if the segmented regions are part of a line, goal or ball. For a white region to be considered a white line it has to consist of certain number of segments, to have a certain size, and to have a certain amount of green above and below if it is horizontally oriented or a certain amount of green on its left and right side if it is horizontally oriented.

For detecting goals, since the foot of a post must be below the horizon and the head of the post must be above, it is sufficent to scan the projection of the horizon in the image for blue or yellow segments to detect points of interest.

An orange region, in order to be considered a ball cannot be above the horizon or at a distance greater than the length of the field diagonal. The ball is validated based on the distance of the Cb and Cr components of the surrounding pixels. The center and radius of the ball are calculated and the relative position to the robot.

2.2.2 TT-UT Aston Villa

TT-UT [23] Austin Villa is a joint team of Texas Tech University and the University of Texas at Austin. TT-UT Austin Villa won the 2009 US Open and the 2010 US Open in the SPL. The team also finished third in the 2010 RoboCup competition and fourth in the 2009 RoboCup competition. Their software is divided into four main modules: vision, localization,

locomotion, and coordination.

The vision module is based on Reinforcement Learning Algorithms - the robot should be able to improve its own behaviour without the need for detailed step-by-step programming. Their algorithm, Reinforcement Learning with Decision Trees uses decision trees to learn the model by generalizing the relative effect of actions across states. The agent explores the environment until it believes it has a reasonable policy. The robot is enabled to learn the colors on the robot soccer fields by modeling colors as 3D Gaussians, using a pre-defined motion sequence.

2.2.3 MRL

Mechatronics Research Laboratory [24] was established in 2003 as an independent research centre under the supervision of Qazvin Islamic Azad University. One of its activities is participating at the RoboCup international competition.

Their approach for the vision system is mostly based on pattern recognition. The ball recognition algorithm is based on detecting a circle and filtering undesired noise. First the image is segmented and scanned to find orange spots. After performing a search of 8×8 pixels around the detected pixel, the image is scanned in four different directions starting at the detected pixel. The recognized object is most likely to be a square or a circle when the two measured dimensions are equal. The final ball settlement position is estimated by accurate ball kinetics formulation. Distance between the ball and robot is calculated by a comparison between perceived ball radius and original radius. Then the absolute distance could be achievable by filtering the elevation between camera and robot's feet. The easiest way to find the direction of ball is comparing current and latest ball positions.

Regarding goal perception, scan lines segmentation combined with fuzzy logic detection have been used.The obstacle detection can be done by vision procedure or by using the Ultra Sonic retrieved data. Ultra Sonic retrieved data is executed with a sensor fusion process to detect an obstacle in front of the robots, then vision is admitted to detect those obstacles. In the final step the robots are modeled in the world state and their location and orientation in each period are updated successively.

2.2.4 Austrian Kangaroos

The team [25] started in 2010 and is supported by the Automation and Control Institute (ACIN) and the Institute of Computer Languages Compilers and Languages Group (COMPLANG) from the Vienna University of Technology, as well as by the Institute of Computer Science from the University of Applied Sciences Technikum Wien.

They developed their vision system based on the CMVision [26] color segmentation software which provides a base to segment images in regions of interest, to whom were assigned probabilities. Moreover, more useful information is retrieved by calculating a more precise

camera pose relative to the ground plane, taking knowledge of the robot's kinematic and its internal sensor reading into account. Thus, the robots are able to estimate the distance to regions of interest and to verify the accuracy of observations that are based on object sizes in the image plane. To increase the robots world knowledge, a global model was implemented and integrated into the robot's world model via the multi-hypothesis approach. Every robot shares his knowledge with the full team.

2.2.5 CMurfs

This team [27] was created in 2010 by researchers at the Carnegie Mellon University with the purpose of joining the RoboCup SPL.

Their vision system is divided into two stages. The first one, low level vision, uses the CMVision library [26] to perform color segmentation on the image. CMVision uses a lookup table to map from YUV pixel intensities to symbolic colors, such as red, blue or orange. The library then builds up lists of the colored regions in the resulting image. These lists of regions, which specify the bounding box and centroid of each region, are then used in the second stage of vision, high level vision, for object detection. The Vision component processes the camera images and reports Vision Features, such as balls and goal posts, with a heuristic confidence value between 0 and 1, representing how confidence they are that the object is truly present in the image. The Vision Features detected by the Vision component are used by the localization component to generate a pose of the robot.

2.2.6 Cerberus

Cerberus [28] was the first international team in the Standard Legged Robot League. It started as a joint effort of Bogazici University, Turkey and Technical University of Sofia, Plovdiv Branch, Bulgaria. Currently Bogazici University is maintaining the team.

For their vision system they use a Generalized Regression Neural Network for maping the real color space to the pseudo-color space composed of a smaller set of pseudocolors, namely white, green, yellow, blue, robot-blue, orange, red and "ignore". A look up table is constructed for all possible inputs. Scan lines are used to process the image in a sparse manner, thus speeding up the entire process.

The process starts with the calculation of the horizon based on the pose of the robot's camera with respect to the contact point of the robot with the ground, that is the base foot of the robot. After that, scan lines that are 5 pixels apart from each other and perpendicular to the horizon line are constructed, such that they originate on the horizon line and terminate at the bottom of the image. Then, a scan through these scan lines is started to find where the green field starts, which is done by checking for a certain number of consecutive green pixels along the line. In order not to lose information about those important objects, a convex-hull is formed from the starting points of the green segments.

They define the real green field borders where all objects of interest fall inside. After the field borders are constructed, the shorter scan lines are extended back to these borders, thus being possible to use them to detect the goal posts and balls that are close

to the borders. After this, each scan line is traced to find colored segments on them. If two consecutive segments touch each other, they are merged into a single region. For white segments, there are some other conditions such as change in direction and change in length ration, which guarantee that all field lines are merged into smaller and more distinctive regions.

2.2.7 Borregos

Borregos [29] is a Mexican team that has been participating in the RoboCup competitions since 2004, starting in the 2D Simulation League and moving forward to the 3D Simulation League with humanoids in 2007 and finally joining also the Standard Platform League in 2010.

The vision process of this team starts by obtaining a raw image from the camera device, then a RGB matrix is extracted from the input image and the color classification process if performed. The output of the color classification is an object map, which is a matrix of the same size as the RGB matrix but instead of the RGB values it contains object labels. The pattern recognition process takes the object map and tries to find flags and then construct a vector for every recognized flag. Each vector contains distance, vertical angle and horizontal angle to flags. Color classification is the process that maps pixel values of an image to a color label that corresponds to a region of color space pre-defined in a look up table (LUT). The raw values of ball distance, direction and elevation are calculated based on the supposition that the ball is always on the floor, which is generally true. This is done by calculating the centroid of all pixels corresponding to the ball in the image and dividing them over the image width, then multiplying the resulting calculation by the fill of the view of the camera.

2.2.8 Robo Eireann

The team [30] consists of undergraduates, graduates and academics from The Hamilton Institute, Computer Science and Electronic Engineering at NUI Maynooth.

Their vision uses OpenCV but with problems in obtaining the speed needed for high levels of performance. Their approach has proven helpful for generating colour-segmentation algorithms, based on optimization working in the HSI space for captured images. The system seeks the optimal HSI space bounds for each color to minimize a compromise of false alarms and misses in color segmentation. This is then used to generate a LUT in the robot color space for color segmentation.

2.2.9 TJArk

The team [31] was established in 2004 as a part of the Lab of Robot and Intelligent Control of Tongji University in China and participated in RoboCup for the first time in 2006.

Their vision system is based on scanning the image using a color look up table in order to form segments of the same color, then using run length encoding algorithms to merge the segment into blobs which can be then used in the vision recognition. For goal detection, they scan horizontally the image then the horizontal run will be connected to blobs based on their positions. Some extra scanning is made to decide the goal post blob. When there is only one eligible post left, they use the crossbar and the corner in the forbidden area to decide whether the post is on the right or left. In order to recognize the lines and kick-off circle they adopted the approach of grouping points of the same color into blobs and then calculate the characteristics of these blobs. Inspired by the resampling method, they use the run-length encoding algorithm to detect the target region, then fit the sample to the Gaussian model or the Histogram model. If the light condition changes, they chose the MAP method to adapt the model to accommodate the current light condition.

2.3 Critical comments

The first step in developing the vision system that is presented in this document was the studying of the work that has already been developed in the area of robotic vision, with emphasis on the work of the teams currently participating in RoboCup Standard Platform League. The information about the current level of progress of each of the teams was presented in the previous subsections (Subsection 2.2.1 to 2.2.9). An overview of the work developed so far in robotic vision was needed in order to better understand the context, the challenges and the constraints that robotic vision implies. Moreover, having access to different solution proposals that are being implemented by a wide range of researchers and consequently to the advantages and possible flaws that each proposal brings, the choice of a personal approach becomes easier. All of the vision systems presented so far have certain common features and some of them can also be found in the architecture of the vision system described in this paper. They will be detailed as follows. Nevertheless, none of the teams presented have approached in their Team Description Papers the issues regarding image acquisition and calibration of the camera intrinsic parameters.

The use of an image processing library is common to all projects envolving a vision system and humanoid vision systems are not an exception. For the teams in the SPL, OpenCV and CMVision libraries are two of the most common choices. They provide a wide range of functions for manipulating images. For the project that is being described in this thesis, the library chosen is OpenCV [32].

The environment in the SPL is still color coded which means that a good color classification is a very important step in attaining good results. For this reason many teams use a LUT for pre-defining labels for the regions that have one of the colors of interest. Then the values of the pixels in the image are mapped to a corresponding color label, as it will be described in Section 4.5.

Since a vision system is a real-time application that has to perform its task in a limited amount of time due to the restraints of the video device, time management when processing an image has to be performed. For example, using a device that works at a frame rate of 30 frames per second implies that the total time that can be spent for acquiring and processing one frame is 33 ms. The main approach for obtaining a small processing time is based on the subsampling of the information acquired from the camera of the robot. In this way, either the information about the color or the information about the luminance can be subsampled while the acquisition process can still deliver good representations of the surrounding world. In addition to this, in order to reduce the time spent for scanning an image in search of one of the colors of interest, several teams choose the approach of using either vertical or horizontal line scans, which means that the number of scanned pixels is much smaller than the size of the image. For the project presented, vertical or horizontal scan lines are used for detecting transitions from green to one of the colors of interest. Also only half of the columns and half of the rows of the image are scanned in order to obtain a better processing time. The details of this process are described in Section 4.6.

After having the color of interest segmented the common approach for the majority of teams is to merge all neighbour pixels of the same color into blobs [33]. The idea of "neighbourhood" might be different from team to team but the concept is the same. For this purpose, the run-length encoding is one of the most common choices. Run-length encoding is a very simple form of data compression in which sequences of the same value, that occur a large number of time, are stored as a single data value and count, rather than as the original run. In the case of the proposed system architecture, also based on run-length encoding, pixels of the same color are grouped into blobs if they belong to parallel vertical or horizontal scan lines that are close to each other. Then the blobs pass a first validation test towards being labeled as objects of interest if they are preceded or/and followed by a certain number of green pixels. Also common for this project as well as for the other projects mentioned before, after having a valid blob is computing several measurements such as the center of mass and area of the bounding box, which prove to be very helpful in the process of validating the objects of interest. A more detailed description of these steps of the implementation of the vision system is presented in Section 4.8.

The concepts discussed might be common for a large number of robotic vision systems but every implementation has its particularities which differentiate it from the others. The particularities of the project described in this document will be described in the following chapter.

Chapter 3

Vision system architecture

This thesis presents a modular vision system developed for humanoid robots, but which can be used, with small changes, in other robotic platforms that have a digital camera as the main sensor.

In this chapter the architecture of the vision system will be described, giving more details about some support modules and applications that are not running on the robot. The main vision process running on the robot will be presented in Chapter 4. The modular vision system is presented in Fig. 3.1.

Apart from the vision system process running in real-time on the NAO robot, several other modules and applications were developed for support and debugging purposes. These modules make possible the communication of the robot with a local host as well as the sharing of the information provided by the vision process with the rest of the processes running on the robot. The two applications developed can be used either for color classification, which is the first step of the object detection process, or for image visualization an debugging, considering that the NAO physical architecture does not support any graphical interface. Moreover, the Linux operating system that comes with the robot does not have graphical server, such as XServer [1]. These two applications (NaoCalib - Section 3.2 and NaoViewer - Section 3.3) run on an computer different from the one of the robot and receive data through sockets.

3.1 Communications

Limitations of NAO's hardware architecture (see Section 2.1) make impossible the visualization of the results of the vision algorithms on the robot. Moreover, when robots are operating the human user cannot look directly to the results, in terms of what the robot "sees". This is valid for most of the mobile robots. Interfaces for the display and analysis of the images cannot be supported since they would need to access and use a great amount of the resources available on the robot. Also the operating system running on the NAO robot does

[1] www.x.org/releases/current/doc/man/man1/Xserver.1.xhtml

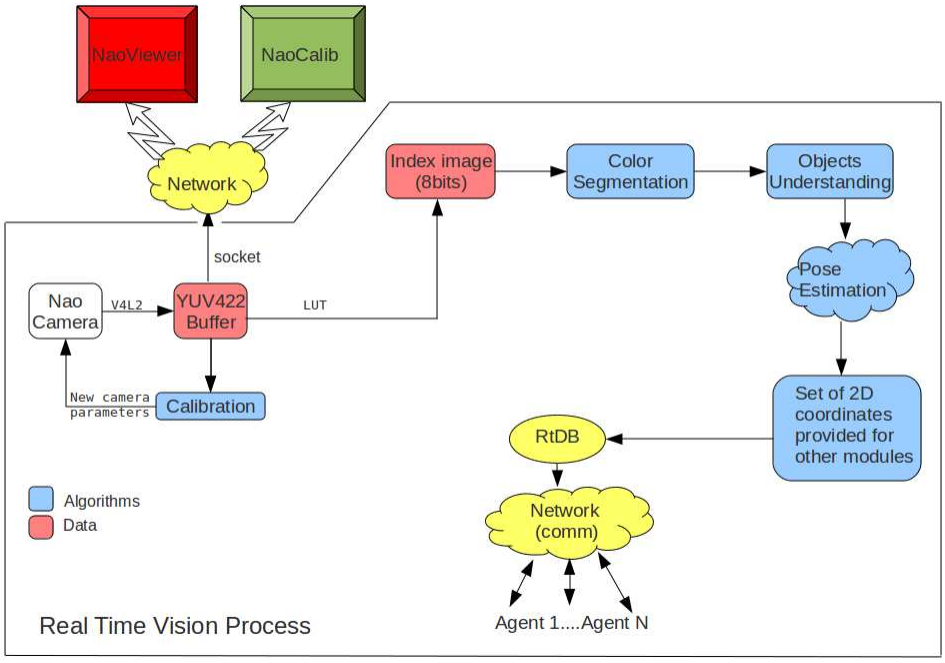


Figure 3.1: Vision system architecture including the applications that do not run on the robot.

not have graphical support. As in image processing the use of tools that allow at least image displaying it is imperative, a solution for implementing them was needed. The chosen solution is based on the development of tools that run on an external computer and communicate with the robot by means of a network socket, using a server-client model.

A socket represents an endpoint of a bidirectional inter-process communication flow across a computer network. The socket is responsible for delivering incoming data packets to the appropriate application process, based on a combination of local IP addresses and port number. Each socket is mapped by the operating system to a communicating application process. The combination of an IP address and the port into a single identity is called the socket address. The network socket can also be seen as an application programming interface (API) for the TCP/IP or UDP protocol stacks.

The client and server terms refer to two processes which will be communicating with each other. One of the two processes, the client, connects to the other process, the server, typically to make a request for information. The client needs to know of the existence and the address of the server, but the server does not need to know the address of (or even the existence of) the client prior to the connection being established. When a connection is established, both sides can send and receive information. The system calls for establishing a connection are somewhat different for the client and the server, but both involve the basic construct of a socket.

Figure 3.2 shows the steps followed when implementing a server-client communication

architecture.

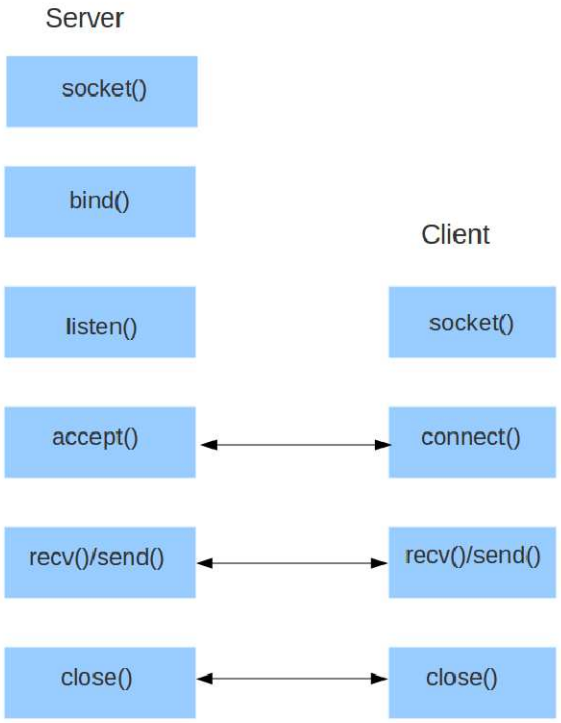


Figure 3.2: Typical client server approach.

The steps involved in establishing a socket on the server side are as follows:

- Create a socket with the *socket()* system call.
- Bind the socket to an address using the *bind()* system call. For a server socket on the Internet, an address consists of a port number on the host machine.
- Listen for connections with the *listen()* system call.
- Accept a connection with the *accept()* system call. This call typically blocks until a client connects with the server.
- Send and receive data.

The steps involved in establishing a socket on the client side are as follows:

- Create a socket with the socket() system call
- Connect the socket to the address of the server using the connect() system call
- Send and receive data

The main vision process running on the robot has the option of being a server thus sending the required information for the client applications that have been developed and that will be presented in detail in the following sections. When the main vision process is run with the option "-server" the first step is to create a socket whose address will be the combination of the IP of the host and the predefined port 5000. When the client connects to the server it will start sending the information required by the client. The client can request and receive either the YUV422 buffer with the raw data acquired by the camera or the buffer of the processed index image. The first is converted in the client side and displayed as an RGB image with a resolution of 640×480 pixels, the maximum resolution of the NAO camera is 640×480. The latter is displayed with the resolution used int he processing algorithms (for the NAO robot a sub-sampled resolution of 320×240 pixels is used). The index image represents an 8-bit image of labels for all the colors of interest and it will be further explained in Section 4.5. The client receives frames at a slower rate than the frame rate because of network delays and the limitations of the bandwidth. Typically, it is necessary 1s to receive a full frame from the robot.

3.2 NaoCalib and the configuration file

In the RoboCup SPL, as well as in other robotic applications, image analysis is simplified due to the color coding of the objects. In SPL the robots play soccer with an orange ball on a green field with white lines and yellow and blue goals. The color information of a pixel is a strong hint for object validation. Because of this, a good classification of the colors is necessary.

Along with the calibration of the parameters of the camera (presented in Section 4.4), a calibration of the color range associated to each color class has to be performed whenever the environment or the illumination conditions change. These two processes are co-dependent and crucial for image segmentation and object detection.

NaoCalib is an application created after a model used by CAMBADA [34, 35], the RoboCup Middle-Size League team of the University of Aveiro. It is used for the classification of the colors of interest and it allows the creation of a configuration file that contains the Hue (H), Saturation (S) and Value (V) minimum and maximum values of the colors of interest. The configuration file (Fig. 3.3) is a binary file that apart from the H, S and V maximum and minimum value also contains the values of the intrinsic parameters of the camera.

In other applications, the configuration file can contain more information that could be relevant for the vision process. For example, in the situation when the camera of the robot has a fixed position relative to the ground, the configuration file could contain information about the mapping between pixels and real distances. Moreover, information about the regions of the images that do not have to be processed can be added.

In the case of the NAO robot, NaoCalib is only responsible for the saving on the file of the

H, S and V range of the colors of interest. The information about the intrinsic parameters of the camera is filled when the calibration module (Section 4.4) is run. The configuration file is created on the client side and then exported to the robot and loaded when the vision process starts.

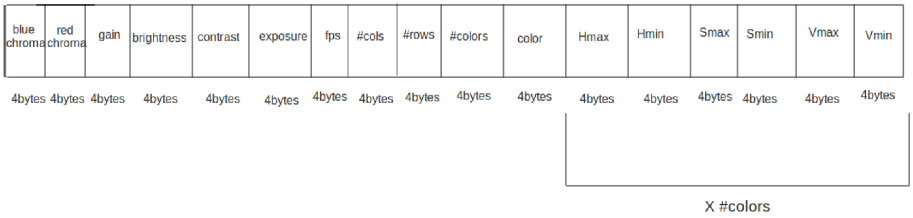
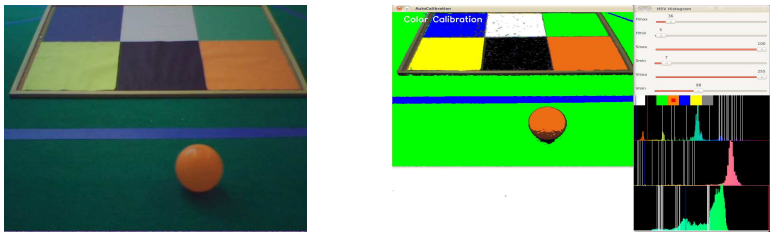


Figure 3.3: Structure of the binary configuration file used in the proposed vision system.

The calibration of the colors is performed in the HSV color space because the results are more accurate than in the case of the RGB color space [36].

Having the vision process running as a server, according to how it has been described in the previous section, NaoCalib can act as a client that receives from the NAO robot the image buffer and displays it in the RGB format with a default resolution of 640×480 pixels. These frames are used for calibrating the color range associated to each color class.

The interface is based on the histograms of the three color components (Hue, Saturation and Value) and it allows the selection of the color range for each color class with the help of sliders. For each of the colors of interest, a set of pixels corresponding to the color class that is being calibrated can be selected with the help of the mouse. The color classes correspond to the colors of interest, which are: white, green, orange, yellow, blue, blue-sky, magenta and black. Based on this and with the help of the H, S and V histograms the user can manipulate the sliders for selecting the maximum and minimum values of the three components for each color class. These values are then updated in the configuration file, copied to the robot and loaded when the vision module is next started.



(a) (b)

Figure 3.4: On the left, an original image acquired by the NAO camera. On the right, the same image with the colors of interest classified by means of the NaoCalib application.

The interaction between the user and this application is smoothen by the use of different keys of the keyboard, each of them being responsible for one of the commands that is shown

in Fig. 3.5.

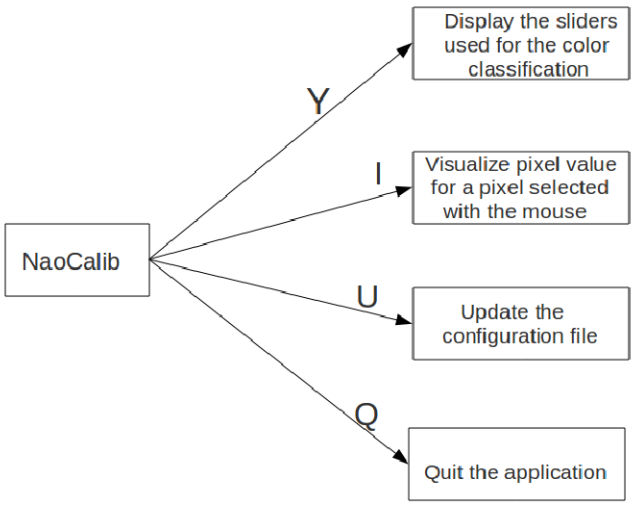


Figure 3.5: An illustration of the features of the NaoCalib application that can be accessed by pressing several keys of the PC keyboard.

3.3 NaoViewer

Another client application that has been developed and that is being used for debugging purposes is NaoViewer. NaoViewer is a client tool that has several options useful in the monitoring and testing of the implemented algorithms.

As a first option, NaoViewer can be run with the basic purpose of receiving and displaying frames acquired by the camera of the robot. The client receives the YUV422 raw buffer acquired by the NAO camera which is converted into an YUV422 image as an intermediary step and then converted to the RGB color space with the help of the OpenCV functions. The RGB image is displayed with a default resolution of 640×480 pixels.

The second option of this application is to display the 8-bit indexed image as well as its "painted" RGB version, the former also having the option of containing markers for the detected objects of interest. In this situation, the resolution used in the processing algorithms (in the case of the NAO robot, the resolution was sub-sampled to 320×240 pixels, see Section 4.5). The index image, as previously mentioned, is an 8-bit image in which all the colors of interest are labeled accordingly to the look-up table. The painted image is a 3-channels RGB image of the same resolution as the index image. The index image is scanned and for each pixel labeled as having one of the colors of interest, the color of the corresponding pixel in the RGB image is forced as having the respective pure color of interest. If there are pixels that do not have any of the colors of interest they will be painted as gray. Moreover, all the detection and validation algorithms are performed on the index image but their results can

also be visualized on the RGB painted image. Figures 3.6 and 3.7 presents an example of the features of the application.

(a) (b)

Figure 3.6: On the left, an original image acquired by the NAO camera and displayed by the NaoViewer application. On the right, the same image with the colors of interest classified by means of the NaoCalib application.

(a) (b)

Figure 3.7: On the left, the index image with all the classified colors labeled. On the right, the RGB image obtained by "painting" the index image according to the labels.

Furthermore, the NaoViewer application allows the recording of a video by saving all the frames received. This is also very helpful in the task of understanding what the robots "see" and for offline debugging. Complementary to this, a basic application for reading and displaying the video acquired has been implemented. Moreover, the main vision process was adapted to send the raw data previously saved.

3.4 Real-time database

Cooperation is one of the key words that should describe a team and this also applies in the case of a robotic soccer playing team. A common approach for achieving cooperative sensing is by means of a database where each agent publishes the information that is generated internally and that might be requested by others. In the proposed vision system a Real-time

Data Base (RtDB) based on [37] is used and it holds the state data of each agent (running, penalised or stopped), together with local images of the relevant state data of the other team members. This approach allows a robot to use the information provided from the rest of the robots to complement its own. For example, if a robot cannot get track of the ball it can easily use the position of the ball as detected by another robot.

The RtDB is implemented over a block of shared memory and it is divided into two areas. A private one, for local information only and a shared area. The private area contains information that is not to be broadcasted to the rest of the robots while the shared one contains global information provided to all players. The information is shared using a process described in 3.5. The last one is then divided into a number of areas equal to the number of agents in the team (four in the case of an SPL team), each of the area corresponding to an agent. Each of the areas is written by the corresponding agent while the remaining ones are used to store the information received from the other agents (Fig. 3.8).

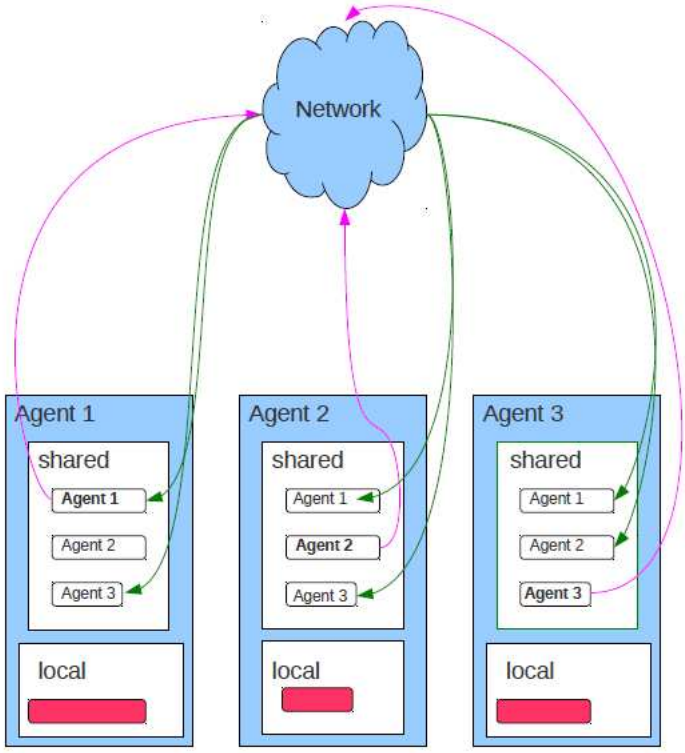


Figure 3.8: Internal representation of the RtDB [2].

The allocation of shared memory is done with a specific function call, $DB_{init}()$ which is called once by every Linux process that needs to access it. The actual allocation is executed only at the first call, the following calls return the shared memory block handler and increment a process count. The memory space can be freed by using the function call $DB_{free}()$ that decreases the process count. When the process count reaches 0 the shared memory block is

released.

The RtDB is accessed concurrently by Linux processes that process images and implement behaviors. The access is made with non-blocking function calls, $DB_{put}()$ and $DB_{get}()$ that allow writing and reading records, respectively.

3.5 Communication among agents

Taking the example of a SPL game, the agents communicate among them using an IEEE 802.11 network, sharing a single channel with the opposite team and each team can use a bandwidth of up to 500Kbps of the wireless lan. Because of the reduced bandwidth and the impossibility of controlling the access to the channel, a robust solution for agent to agent communication was needed. The approach chosen is based on the communication architecture developed by CAMBADA [3] and it uses a dynamic adaptive TDMA transmission control that will be described as follows.

TDMA stands for Time Division Multiple access and it is a channel access method for shared medium networks that allows several users to share the same frequency channel by dividing the signal into different time slots. The users will transmit in rapid succession, one after the other, each using his own time slot. The approach that is being used for the communication among team members defines a round period called team update period (T_{tup}) that sets the responsiveness for the global communication. T_{tup} is divided equally by the number of currently active team members generating the TDMA slot structure [38]. This structure is reconfigured dynamically everytime a node leaves (e.g., crashes) or joins the team. Each running agent has one single slot allocated so that all slots in the round are separated (Fig. 3.9). The target inter-slot period T_{xwin} is calculated as T_{up}/N, where N is the number of running agents.

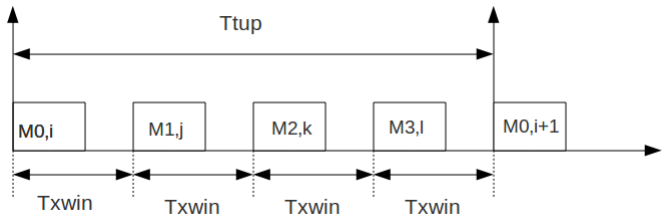


Figure 3.9: Illustration of a TDMA round [3].

A rate-monotonic scheduler is used for scheduling the transmissions generated by each agent according to the production periods specified in the RtDB records. When the respective TDMA slot comes, all currently scheduled transmissions are piggybacked on one single 802.11 frame and sent to the channel. The required synchronization is based on the reception of the frames sent by the other agents during T_{tup}. If delays affect all TDMA frames in a round, then the whole round is delayed from then on, thus its adaptive nature. Figure 3.10 shows

a TDMA round indicating the slots allocated to each agent and the adaptation of the round duration.

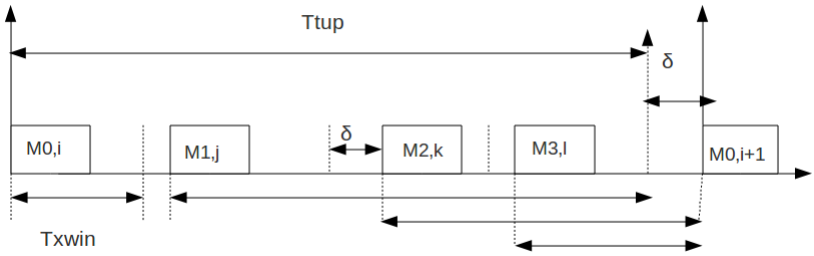


Figure 3.10: Illustration of a TDMA adaptive round [3].

Chapter 4

Vision process: from image acquisition to object detection

The architecture of the vision process is quite complex and represents the best compromise that has been reached between processing requirements and the hardware provided in order to accomplish the goals of this project.

In this chapter the main vision process that runs on the robot will be presented. First, a brief description on each of the modules developed will be presented and then, in the following sections more details about each module will be provided. The vision process can be divided into three main parts, as follows: access of the device and image acquisition, calibration of the camera parameters and object detection and understanding. The block diagram of the proposed vision process is presented in Fig. 4.1.

As mentioned before, the NAO robot has two video cameras and the presented module can be used with any of them. The current version of the software allows to switch between cameras in a small amount of time (on average, 29ms). However, since only one camera can be used at a time, only the lower camera of the robots is being used since it can provide more meaningful information about the surroundings. The switch between cameras can be very useful when more evolved game strategies will be developed and the upper camera can ease NAO's dribbles. The camera is accessed using V4L2 API [21] and its raw output is in the YUV422 packed format.

The calibration process and the one of object detections do not run simultaneously on the robot. The calibration module is not continuously running on the robot due to the processing time limitations. It is run just once whenever the environment or the lighting conditions change, having the purpose of setting the parameters of the camera so that the images acquired give the best possible representation of the real world.

For the detection module, with the use of a look-up table (LUT) the raw buffer can be converted into an 8-bits grayscale image in which only the colors of interest are mapped (orange, green, white, yellow, blue, pink, blue sky and gray - stands for no color).

The next step is the search for the colors of interest in the grayscale image and the formation of blobs of the same color. The blobs are then marked as objects if they pass the validation criterias which are constructed based on different features extracted from the blobs.

Finally, after merging the information about the objects found in the image with the estimation of the pose of the robot provided by another module running on the robot, the vision module is capable of delivering a set of 2D coordinates for the other modules that necessitate them. The details of the implementation of all these steps will be presented in the sections that follow.

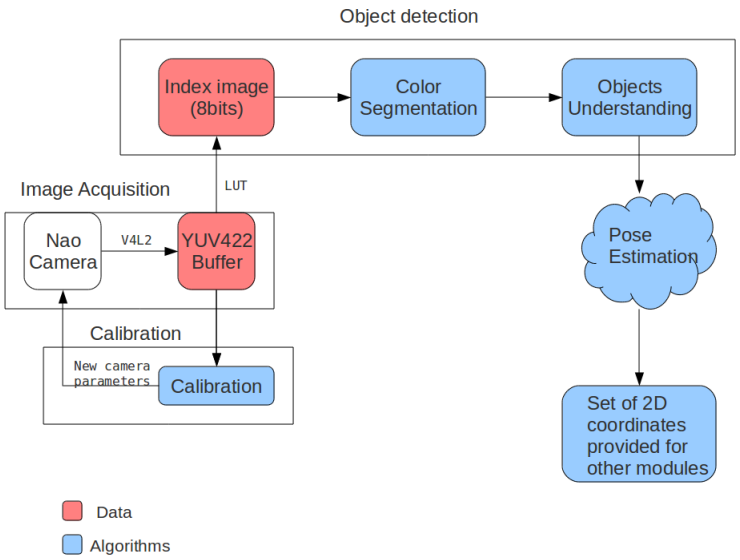


Figure 4.1: Block diagram of the vision process that runs on the robot.

4.1 Accessing the device and acquiring images

NAO has 2 identical video cameras that are located in the forehead and chin area, respectively. They provide a 640×480 resolution image at 30 frames per second. They can be used to identify objects in the visual field such as goals and balls, and bottom camera can be useful when the NAO robot is dribbling the ball. The native output of the camera is YUV422 packed.

A first approach for accessing the NAO cameras was based on the NAOqi framework that works on the robots. NAOqi is a framework property of Aldebaran Robotics that runs on the robots and acts as a server. Different modules can plug into NAOqi either as a library or as a broker, with the latter communicating over IP with NAOqi. It supplies binding for C, C++, Python, Ruby and Urbi. NAOqi itself comes with several notable modules, two of them being the ALVideoDevice and the ALVisionToolbox modules, which provide methods not only for accessing both cameras of the robot but also methods for acquiring images of

different resolutions and for converting them between different color spaces. However, after conducting a series of tests of the available methods, it was proved that this approach could not be a useful one in the process of developing real-time applications since only the acquiring time of one frame was elevated (approximately 120 ms for the minimum resolution of 160×120 pixels).

The solution chosen for accessing the cameras is based on the Video For Linux (v.2) API, a kernel interface for analog radio and video capture and output drivers. Programming a V4L2 device consists of the following steps:

- Opening the device.
- Changing device properties, selecting video and/or audio input, video standard, picture brightness, etc.
- Negotiating a data format.
- Negotiating an input/output method.
- The actual input/output loop.
- Closing the device.

The V4L2 driver is implemented as a kernel module, loaded automatically when the device is first opened. The driver module plugs into the "videodev" kernel module.

The raw format of the NAO cameras is YUV422 packed. In the YUV colorspace the Y component stands for luminance or luma and determines the brightness of the color, while the U (also called Cb) and V (also called Cr) components determine the color itself (the chroma). The luminance can be seen as a grayscale range that goes from white to black. The chromatic components are represented in Fig. 4.2. In digital formats Y, U and V range from 0 to 255.

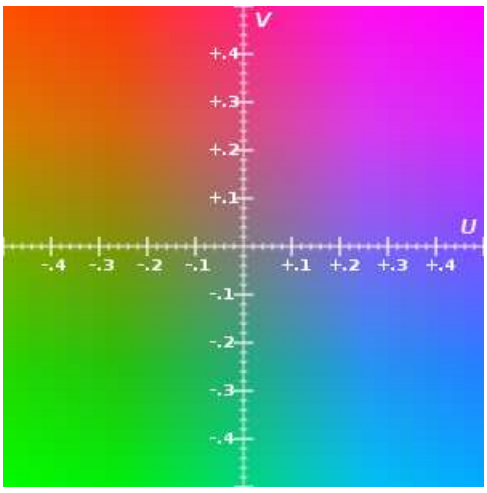
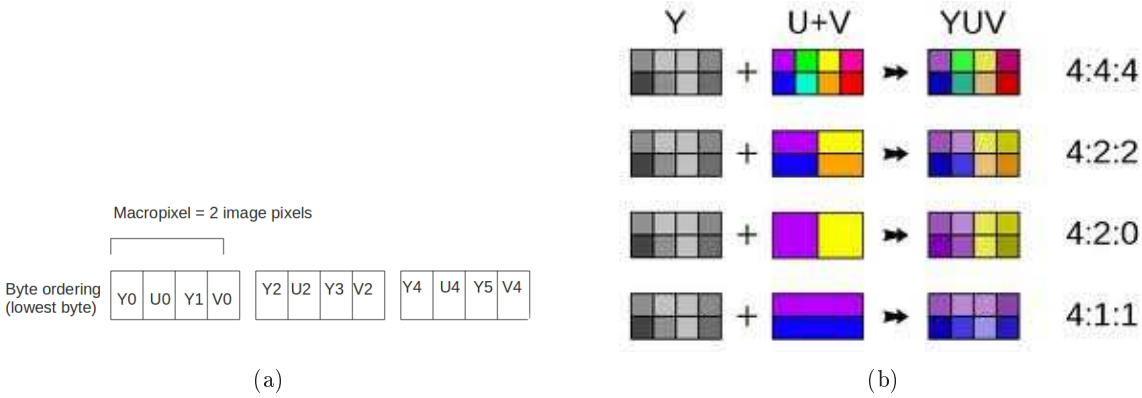


Figure 4.2: Representation of the U and V components on a scale from -0.5 to 0.5. [4].

Being a packed format it means that Y, U and V samples are packed together into macropixels which are stored in a single array (Fig. 4.3(a)). The numerical suffix attached (422) indicates the sampling position across the image line. For the Y sample, both horizontal and vertical periods are 1 while for the U and V samples the horizontal period is 2 and the vertical one is 1. This means that the two chroma components are sampled at half the sample rate of the luma: the chroma resolution is halved (Fig. 4.3(b)). By this, the bandwith of an uncompressed video signal is reduced by one third with little visual difference. The human eye is more sensitive to the luminance, thus chroma reduction does not have such a great visual impact.



(a) (b)

Figure 4.3: In (a) a YUV422 Macropixel [5] and in (b) chroma subsampling in the YUV color space illustration [4].

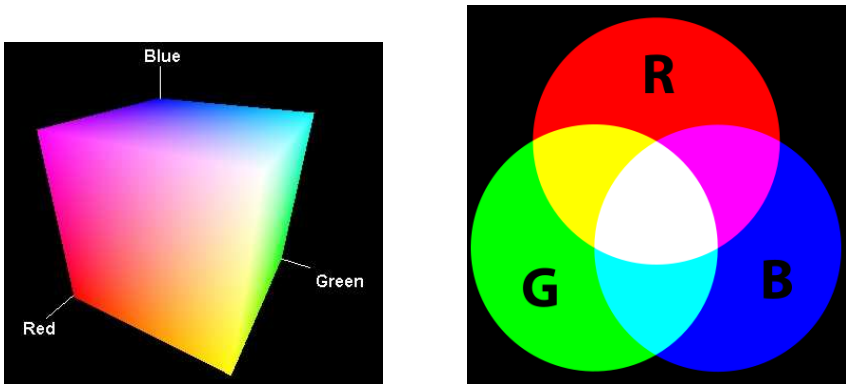
4.2 Conversions between color spaces

For a better visualization and understanding of the images provided by the NAO camera in certain situations it was necessary for them to be converted to a colorspace that is more approachable or that can provide an easier manipulation. Color space conversion stands for the representation of the colors in an image from one space to another, with the purpose of making the image translated to the new color space as close as possible to the original one.

The RGB color space is the most convenient one to work with in computer graphics since it is the closest to the way the human eye works. A RGB color space is an additive color space, defined by the three chromaticities of the red, green, and blue. The main purpose of the RGB color model is for the sensing, representation, and display of images in electronic systems, such as televisions and computers, though it has also been used in conventional photography [10].

Before the electronic age, the RGB color model already had a solid theory behind it, based in human perception of colors. To form a color with RGB, three colored light beams (one red, one green, and one blue) must be superimposed (for example by emission from a black screen, or by reflection from a white screen). Each of the three beams is called a component of

that color, and each of them can have an arbitrary intensity, from fully off to fully on, in the mixture. The RGB color model is additive in the sense that the three light beams are added together, and their light spectra add, wavelength for wavelength, to make the final color's spectrum.



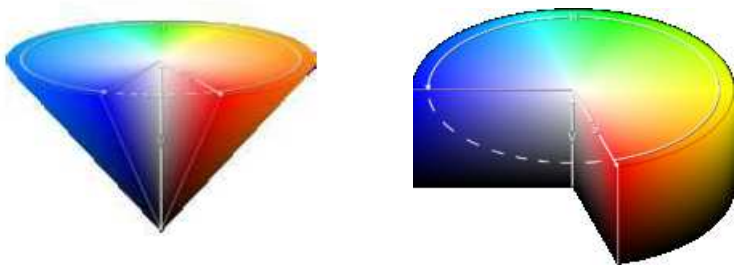
(a) (b)

Figure 4.4: On the left, the RGB cube and on the right, an example of an additive color mixing: adding red to green yields yellow, adding all three primary colors together yields white [4].

The HSV color space is a related representation of points in an RGB color space, which attempts to describe perceptual color relationships more accurately than RGB, while remaining computationally simple [10, 36]. HSV stands for hue, saturation, value and it describes colors as points in a cone whose central axis ranges from black at the bottom to white at the top (Fig. 4.5a) with neutral colors between them, where angle around the axis corresponds to "hue", distance from the axis corresponds to "saturation", and distance along the axis corresponds to "value", "lightness", or "brightness". The hue represents the percentage of color blend, the saturation the strength of the color and the value is the brilliance or brightness of the color.

The HSV color space is mathematically cylindrical, but it can be thought of conceptually as an inverted cone of colors (with a black point at the bottom, and fully–saturated colors around a circle at the top). Because HSV is a simple transformation of device–dependent RGB, the color defined by (h, s, v) triplet depends on the particular color of red, green, and blue "primaries" used. Each unique RGB device therefore has an unique HSV space to accompany it.

A first step in the conversions process is the conversion of the YUV422 packed buffer to a YUV444 image, to be compliant with the *IplImage* structure provided by OpenCV. In a YUV444 pixel, all three components are sampled at the same rate (Fig. 4.3b). Each pixel in the image contains all the information about color. In this case, the mapping of the three components is:



(a) (b)

Figure 4.5: The conical and cylindrical representations of the HSV color space [4].

Y0U0V0 Y1U1V1 ... YnUnVn

The formulas used for the conversions between color spaces are the following [39]:
YUV to RGB conversion:

$$R = Y + 1.403 \times (V - 128)$$
$$G = Y - 0.344 \times (V - 128) - 0.714 \times (U - 127)$$
$$B = Y + 1.773 \times (U - 128)$$

RGB to HSV conversion:

$$V = max(R, G, B)$$
$$S = \frac{V - min(R,G,B)}{V} \; if \; V! = 0, \; 0 \; otherwise$$

$$H = \begin{cases} 60\frac{G-B}{S} & \text{if } V = R \\ 120 + 60\frac{B-R}{S} & \text{if } V = G \\ 240 + 60\frac{R-G}{S} & \text{if } V = B \end{cases}$$

4.3 Camera parameters

One of the most important premises that have to be considered when developing a robotic vision system is that the core of the vision system, which in this case represents the NAO camera, should work within the best possible parameters. The implemented vision system has to guarantee an accurate image acquisition that can best replicate the surrounding world. The results of the image processing algorithms highly depend on the quality of the image acquired. A good calibration of the parameters of the camera is imperative for a satisfying image acquisition. In this section will be presented the most significant intrinsic parameters of the camera. These are present in almost every camera and have the grater influence on the process of image acquisition.

4.3.1 Exposure

Exposure represents the total amount of light allowed to fall on the photographic medium (in this case the image sensor). Correct exposure may be defined as an exposure that achieves the effect that was intended when taking the picture. In photography, shutter speed is a common term used to discuss exposure time, the effective length of time a camera's shutter is open. The total exposure is proportional to this exposure time, or duration of light reaching the film or image sensor. In the case of NAO cameras, the exposure has a range from 0 to 255. Overexposed images are characterized by a loss of highlight details, the bright parts of the image are all white. In an underexposed image the dark areas cannot be distinguish from the black ones. Over- or under- exposing are also referred as "shooting to the right" or "shooting to the left", respectively, as these shift the intensity histogram to the right or to the left. Figure 4.6 shows an example of an image correctly exposed (a), an overexposed image (b) and an underexposed one (c)

(a) (b) (c)

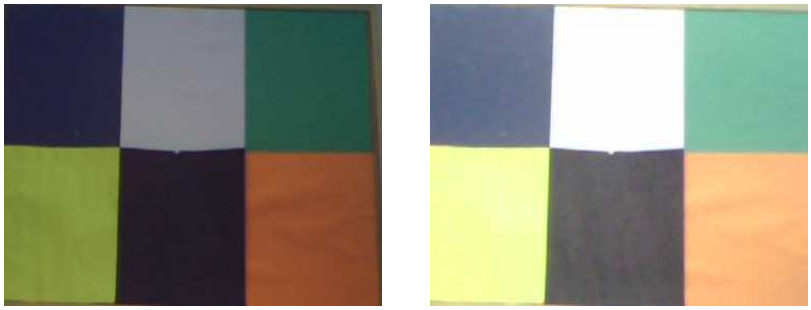
Figure 4.6: On the left, an accurate image acquired with the camera parameters correctly calibrated. In the middle, an overexposed image and on the right, a underexposed image.

4.3.2 Gain

This parameter is related to image brightness and contrast but also to the noise of the image. Increasing this factor makes the image brighter and increases the contrast but at the same time adds noise to the image since the original noise of the image is also amplified. An increase value of the gain can spread out the histogram of intensities, as well as a low value of the gain can compress it. Figure 4.7 (b) shows an example of an image acquired after setting a high value of the gain parameter.

4.3.3 White Balance

White balance is a parameter that when set correctly determines the objects that are white in real world to appear as white in the image acquired by the camera. The image colors appear different depending on the illumination under which the image was taken. Proper camera white balance has to take into account the color temperature of a light source, which



(a) (b)

Figure 4.7: On the left, an accurate image acquired with the camera parameters correctly calibrated. On the right, the same image after significantly increasing the gain value.

refers to the relative warmth or coolness of white light. When this parameter is not properly adjusted the image has a red or blue tonality. The correction can be done by adjusting the red and blue channels gain. The white balance red and blue chromas range from 0 to 255. Figure 4.8 shows an example of an image acquired with correct values of the white balance parameters (a), and images acquired when the red chroma (b) and the blue chroma (c) are not set properly.

(a) (b) (c)

Figure 4.8: On the left, an accurate image acquired with the camera parameters correctly calibrated. In the middle, an image in which the red chroma has a too elevated value, thus the redish tonality of the image. On the right, an image in which the blue chroma is too high.

4.3.4 Brightness

The brightness parameter is responsible for adjusting the black level of an image. If not set properly, the images could appear darker or brighter than they really are. The black level of an image is adjusted by adding or substracting an offset for each pixel. The value of brightness can range from 0 to 255. Figure 4.9 presents an example of a too bright (b) or a too dark (c) picture acquired with different values of the brightness parameter.

(a) (b) (c)

Figure 4.9: On the left, an accurate image acquired with the camera parameters correctly calibrated. In the middle, an image that is too bright due to a high value of the brightness parameter of the camera. On the right, a too dark image captured when the brightness parameter is too low.

4.3.5 Contrast

This parameter is useful for turning the bright colors more bright and the dark colors more dark in order to accentuate details in an image. In an image with a low contrast details cannot be distinguish since the brightness of different elements is almost the same. The range of the contrast is from 0 to 127. Figure 4.10 illustrates an example of a too bright (b) or too dark (c) image.

(a) (b) (c)

Figure 4.10: On the left, an accurate image acquired with the camera parameters correctly calibrated. In the middle, an image acquired with a high value of the contrast parameter andon the right, an image acquired with a low value of the contrast parameter of the camera.

4.4 Self-calibration of the camera intrinsic parameters

The use of camera in auto-mode has raised several issues which made the segmentation and validation of objects hard to be performed (Fig. 4.11). By using the camera in auto-mode the images acquired were far from being accurate and the colors of interest were not represented in a way that the human eye perceives them. Thus, the classification of colors was difficult to be performed and the robots' perception of colors was distorted when compared to the human one. Moreover, the camera changes its parameters along the time.

Camera calibration plays probably the most important role in the process of detecting objects in a color coded environment. It is strongly related to the degree of accuracy of the images acquired, which should give a representation of the surrounding world as close as possible to the real one. Calibration has to be performed whenever the environment changes or when the lighting conditions change over time. This means that, considering the available resources and the small amount of processing time, the calibration module has to be run only when the playing field changes or when the illumination of the field changes over time.

The use of camera in auto-mode has raised several issues which made the segmentation and validation of objects hard to be performed.

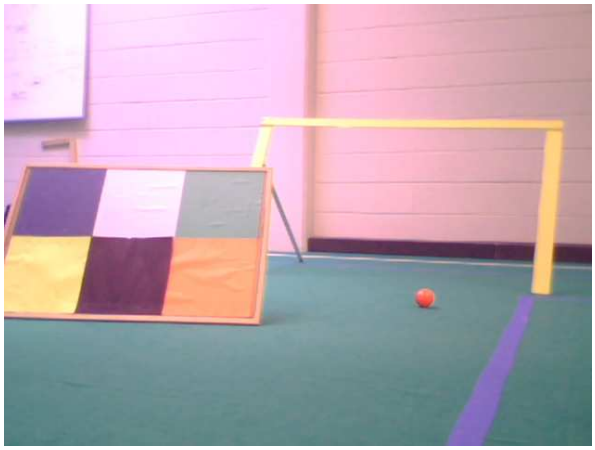


Figure 4.11: An example of an image acquired with the camera working in auto-mode. The color of the carpet is different than the green we would expect to see and the white areas and the yellow goals are too bright.

The algorithm proposed for the autocalibration requires a minimal human intervention and it is based on a PI controller (Fig. 4.12) for adjusting the exposure, the gain and the white balance of the camera based on some statistical measures performed on the acquired image [40]. The self-calibration algorithm is used only for the mentioned camera parameters since a good calibration of these parameters is sufficient for acquiring reliable images. The problems of the camera working in autmode are mostly related to the brightness of the image and to the representation of the white objects, therefore chosing an appropriate range for the gain, exposure and white balance settings can solve these problems.The human intervention is only needed for positioning a white object in a specific, known in advance area of the image for calibrating the white balance parameter of the camera. The algorithm uses the configuration file for saving and loading the values for the camera parameters.

The intensity histogram of an image, that is the histogram of the pixel intensity values, is a bar graph showing the number of pixels in an image at each different intensity values found in the image. For an 8-bit grayscale image there are 256 different possible intensities, from 0 to 255. Image histograms can also indicate the nature of the lighting conditions, the

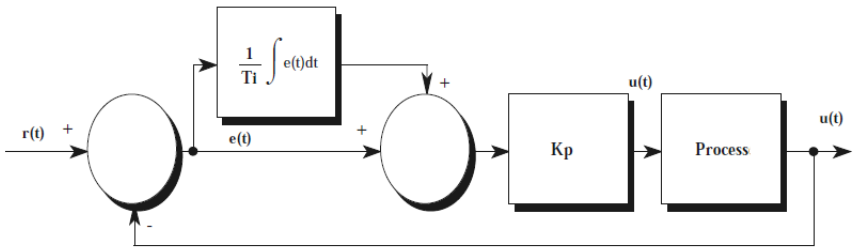


Figure 4.12: Block diagram of a PI controller. $r(t)$, $e(t)$ and $u(t)$ represent the response, the error and the control signals, respectively. The constants K_p and T_i are experimentally determined coefficients associated to the gain, respectively to the integrative action of the controller.

exposure of the image and whether it is underexposed or overexposed. The histogram can be divided into 5 regions as shown in Fig. 4.13 (b) . The left regions represent dark colors while the right regions represent light colors. An underexposed image will lean to the left while an overexposed one will be leaning to the right. Ideally most of the image should appear in the middle region of the histogram.

From the gray level histogram, the MSV can be computed based on the following formula and it represents a useful measure of the balance of the tonal distribution in the image:

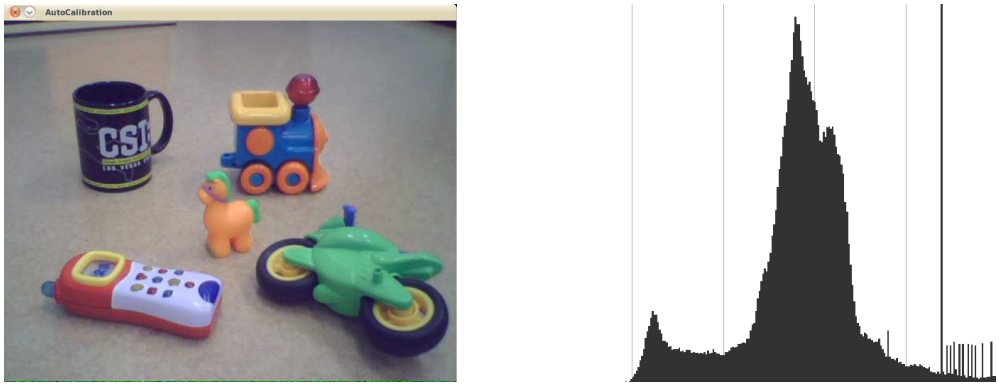
$$MSV = \frac{\Sigma_{j=0}^{4}(j+1)x_j}{\Sigma_{j=0}^{4}x_j}$$

where x_j is the sum of the gray values in region j of the histogram. The image is considered to have the best quality when the MSV ≈ 2.5. MSV is a mean measure which does not take into account regional overexposures and underexposures in the image.

The block diagram of the self-calibration algorithm is presented in Fig. 4.14.

The proposed algorithm works as follows:

- The camera starts with the parameters set accordingly to the values loaded from the configuration file and a frame is acquired.
- The histogram of the image is computed and based on this the Mean Sample Value (MSV) is calculated.
- If the MSV is not in a small proximity(± 0.2) of 2.5 the gain is compensated with the help of a PI controller.
- A new frame is acquired and the process is restarted. If the gain reaches the minimum or maxim possible value the exposure is adjusted until the MSV gets in the correct range.



(a) (b)

Figure 4.13: On the left an image acquired by the NAO camera after the intrinsic parameters of the camera have converged. On the right, the histogram of the image. As expected, most of the image appears in the middle region of the histogram.

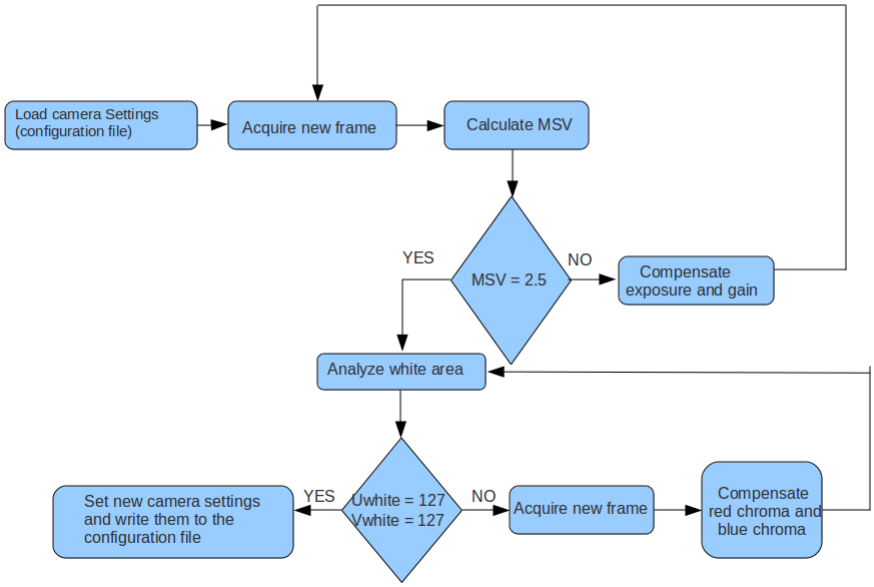


Figure 4.14: Block diagram of the autocalibration process.

- Having the gain and the exposure set the white balance is adjusted by analyzing a white object in front of the robot whose localization in the image has been previously defined. In the YUV color space the average value of U and V for a white area should be 127.
- The red chroma and blue chroma are compensated by means of the PI controller until the U and V values of the white pixels get into the range of $[125, 129]$.
- The new parameters of the camera are written to the configuration file.

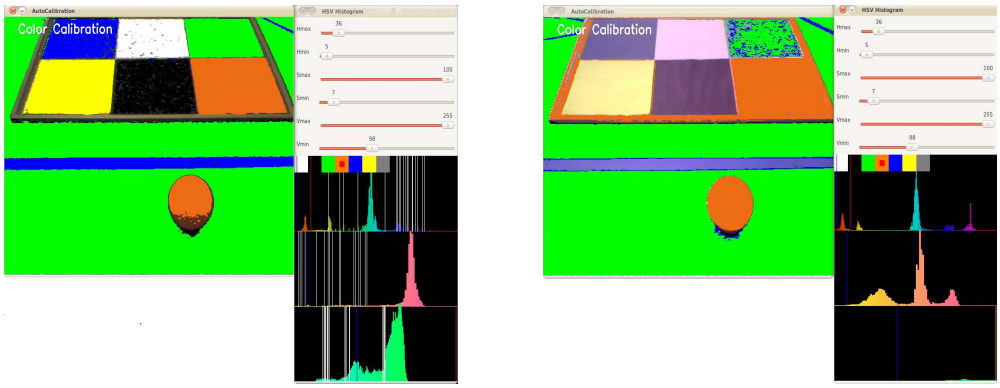
Figure 4.15 shows the results of both steps of the algorithm applied. Step one implies the

calibration of the gain and the exposure parameters and step two is the calibration of the white balance parameter.

(a) (b) (c)

Figure 4.15: On the left, an image acquired with the camera used in auto-mode. The white rectangle, in the top middle of the image, represents the white area used for calibrating the white balance parameters. In the middle, an image acquired after calibrating the gain and exposure parameters. On the right, the result of the self-calibration process, after having also the white balance parameters calibrated.

The difference between working with the NAO camera in automode and after calibrating its intrinsic parameters can be seen in Fig. 4.16.



(a) (b)

Figure 4.16: On the left a color calibration after the intrinsic parameters of the camera have converged. On the right, the result of color classification considering the same range for the colors of interest but with the camera working in auto-mode. Most of the colors of interest are lost (the blue, the yellow, the white and the black) and the shadow of the ball on the ground is now blue, which might be confusing for the robot when processing the information about the blue color.

4.5 LUT and the index image

Image analysis in the context of the SPL branch of RoboCup is simplified by the fact that the objects are color coded. The color of a pixel is a helpful clue for segmenting objects. Thus color classes are defined with the use of a look-up table(LUT) for fast color classification. A LUT represents a data structure, in this case an array used for replacing a runtime computation with the following basic array indexing operation:

$$index_{LUT} = Y << 16 + U << 8 + V$$

This approach has been chosen in order to save significant processing time. The image acquired in the YUV format is converted to an index image (image of labels) using an appropriate LUT.

The table consists of 16,777,216 entries (2^{24}, 8 bits for Y, 8 bits for U and 8 bits for V). Each bit expresses if one of the colors of interest (white, green, blue, yellow, orange, red, blue sky, gray - no color) is within the corresponding class or not (Fig. 4.17). The process of color classification is done with the help of the NaoCalib application. The details about this process were described in Section 3.2. A given color can be assigned to multiple classes at the same time. For classifying a pixel, first the value of the color of the pixel is read and then used as an index into the table. The 8-bit value then read from the table is called the "color mask" of the pixel. Table 4.1 shows three examples of the calculation equivalent to the array indexing operation necessary for determining $index_{LUT}$ and also the equivalent binary value for three of the colors of interest: red, green and blue, respectively.

RGB triplet	YUV triplet	$index_{LUT}$	*binary value*
...	...	...	...
(255, 0, 0)	(76, 84, 255)	$76{\times}2^{16}$ + $84{\times}2^{8}$ + 255	00000001
...	...	...	...
(0, 255, 0)	(149, 43, 21)	$149{\times}2^{16}$ + $43{\times}2^{8}$ + 21	00010000
...	...	...	...
(0, 0, 255)	(29, 255, 107)	$29{\times}2^{16}$ + $255{\times}2^{8}$ + 107	10000000
...	...	...	...

Table 4.1: Table presenting the calculation of the array indexing operation necessary for determining $index_{LUT}$ and, as example, the equivalent binary value for three RGB / YUV values corresponding a three colors of interest: red, green and blue, respectively.

The resulting index image is a grayscale image with the resolution of 320×240 pixels. A smaller resolution was obtained with the purpose of further decreasing the time spent on

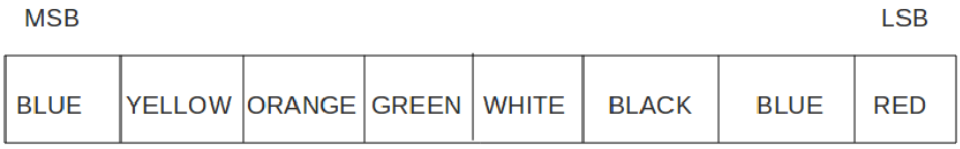


Figure 4.17: The mapping of the colors of interest for a NAO robot based on a one color to one bit relationship.

processing the image. The reduced resolution was obtained by using a subsampling approach. By using the YUV422 packed format of the image, a subsampling of the image across the image line is obtained. For the Y sample, both horizontal and vertical periods are one, while for the U and V samples the horizontal period is two and the vertical one is one. This means that the two chroma components are sampled at half the sample rate of the luma: the chroma resolution is halved.

Then, by type casting the YUV422 buffer [41], which is an unsigned char buffer to an integer one, thus making the reading of four bytes at the same time possible, every one column in four of the image is ignored, by reading only half of the luminance information (Fig. 4.18). Even though for the human eye the luminance is the component of a color that has more significance, this is not valid in the case of robotic vision. Moreover, using this approach we access four times less the memory. This image of labels will be the basis of all the processing techniques that will be described in the following sections. An example of an index image was presented in Fig. 3.7 (a).

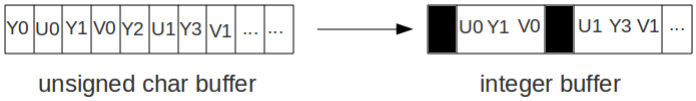


Figure 4.18: An illustration of the type casting of the unsigned char buffer to an integer one, allowing thus the reading of four bytes at the same time. Using this approach a reduced resolution of the images can be obtained. Thus the processing time is significantly decreased and reliable results can still be attained.

The algorithm for the type casting and construction of the index image is depicted next:

Algorithm 1 Algorithm of the type casting of the unsigned char buffer to an integer one and the construction of the index image.

$unsigned\ int\ *b = (unsigned\ int*)YUVbuf$
for $i = 0; i < nColsIndex \times nRowsIndex; i++$ **do**
  $lutPos = (b[i]\ \&\ 0x00FFFFFF)\ >>\ 8$
  $IndexImageData[(r \times nColsIndex + c)] = lut[lutPos]$
  $c++$
  if $c \geq nColsIndex$ **then**
    $i+ = nColsIndex$
    $c = 0$
    $r++$
  end if
end for

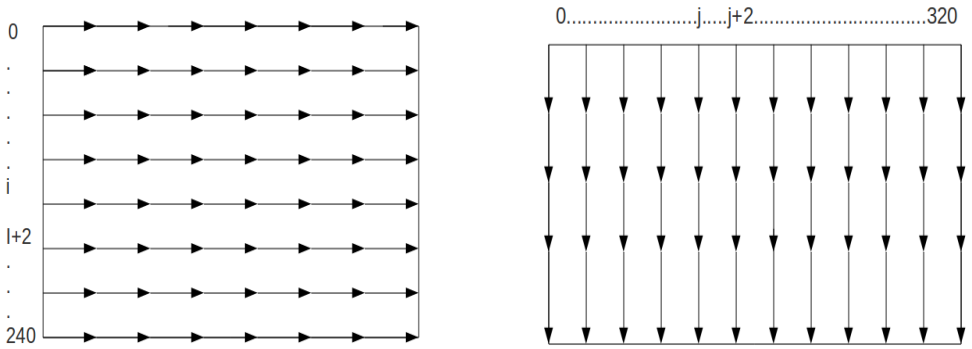
4.6 Color analysis using scan lines

Having the colors of interest labeled, scan lines are used for detecting transitions between colors. The digital representation of an image is nothing more than a matrix, each pixel of the image is an element of the matrix, whose position is specified by a pair (i, j), where i represents the number of the line and j represents the number of the column. For the segmentation of the colors of interest, horizontal and/or vertical scan lines are used. Thus, in the case of horizontal scan lines, the lines of the matrix are scanned while looking for certain values of the pixels. When using vertical scan lines, the columns of the matrix are scanned. Both types of scan lines start at the upper left corner of the image and go along the width and the height, respectively, of the image (Fig. 4.19). Scan lines of a certain color of interest are formed by adjacent pixels that have the given color.

Scan lines are used for finding transitions between colors of interest. In the robotic soccer environment, since all the objects of interest are on the green field during a game, transitions between green and another color of interest are searched. The information about the green color is used as an assertion that only the region of interest of the image, that is the soccer field, is being processed.

While scanning the image in search of a color of interest, with every new row/column a new scan line of length 0 is being initialized. The algorithm processes all the pixels of a row/column and for every green pixel found a counter called GPB is incremented (GPB stands for "green pixels before"). If a pixel of the color in search is being found, a counter called CP ("color pixel") is incremented. Since the algorithm is based on transitions of the type green-color of interest-green, after finding a given number of color pixels, a counter for the following green pixels (GPA) is also incremented every time a new green pixel is found.

The scan line is then considered a valid scan line of the color of interest in search if GPB



(a) (b)

Figure 4.19: On the left, an illustration of the horizontal scan lines. On the right, the vertical scan lines.

and GPA are larger than a predefined threshold. The length of the scan line will be the number of the color pixels that have been found. This approach is an example of run-length encoding, in which repetitive data is compressed. That is, sequences of pixels of the same colors are counted and stored as a single data value.

All the information about the valid scan lines having the same color of interest is saved into an array. The information relevant for a scan line is the initial point of the scan line, its end point and its length. Figure 4.20 shows an illustration of the transitions between green and a color of interest.

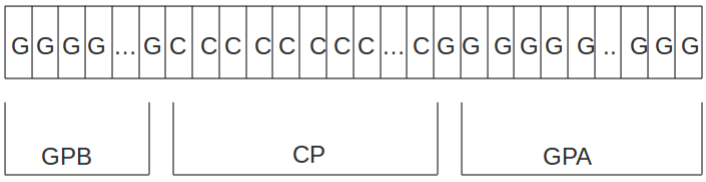


Figure 4.20: Transitions between green pixels (G) and pixels of one of the colors of interest (C).

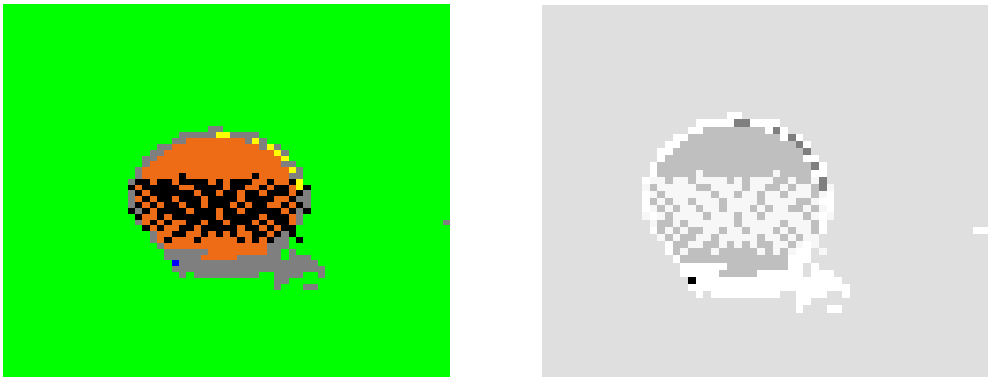
The algorithm for the horizontal search of scan lines is depicted next. The only difference in the vertical search is that the image is scanned on columns.

Algorithm 2 Algorithm of the search for horizontal transitions.

```
for i = 0; i < nRows; i+ = 2 do
  for j = 0; j < nCols; j++ do
    while imageData[i × step + j] == GREEN do
      GPB++
      j++
      continue
    end while
    while j < nCols && imageData[i × step + j] != GREEN do
      if (imageData[i × step + j] & COLOR) != 0) then
        CP++
      end if
    end while
    for k = j; k < j + 20 && k < nRows; k++ do
      if imageData[i × step + k] == GREEN then
        GPA++
      end if
    end for
    if GPB ≥ threshB && CP ≥ threshC && GPA ≥ threshA then
      vector.push(scanline)
    end if
  end for
end for
```

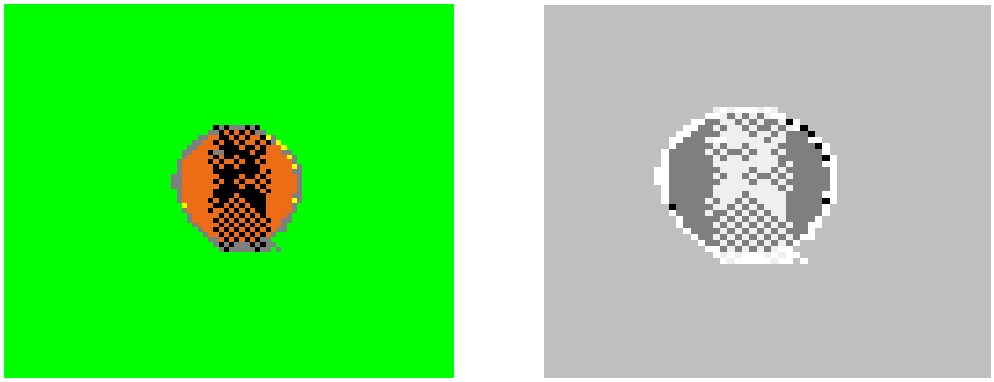
4.6.1 Orange segmentation

For the transitions between green-orange-green, both horizontal or vertical scan lines can be used. However, practical results prove that in this case the horizontal scans offer more valid information. A scan line is considered to be orange if before finding the orange pixel, a minimum number of 5 green pixels have been found and the same number of green pixels has been found after the last orange pixel detected. Moreover, in order to be considered a valid orange scan line, its lenght has to be larger that 20 pixels. The low values of the green thresholds have been chosen experimentally, in order to ensure the detection of the ball (or at least, parts of the ball) even when it is close to the white lines of the field, in the proximity of the goal posts, or when it is caught between robots. Figures 4.21 and 4.22 provide an example of horizontal and vertical orange scanlines.



(a) (b)

Figure 4.21: On the left, the RGB painted image in which the center of all valid orange horizontal scan lines are marked with a black cross. On the right, the index image.

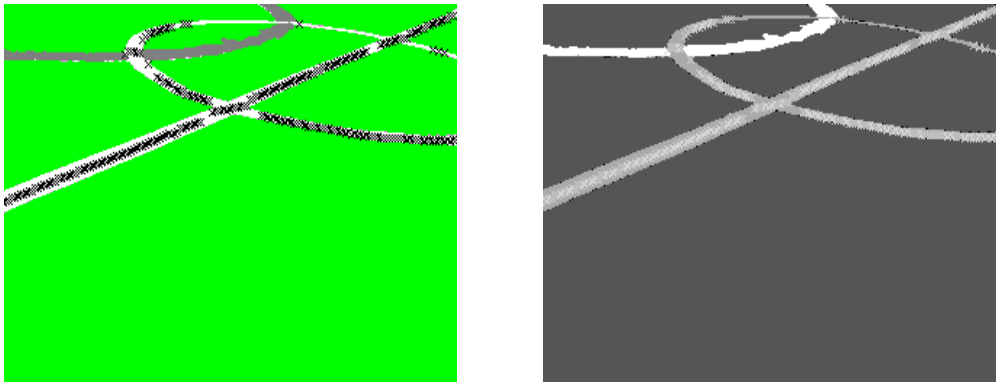


(a) (b)

Figure 4.22: On the left, the RGB painted image in which the center of all valid orange vertical scan lines are marked with a black cross. On the right, the index image.

4.6.2 White segmentation

Transitions of the type green-white-green are found using both horizontal and vertical scan lines. The horizontal scan lines are used for the detection of the side lines of the field, while the vertical ones are used for detecting the white line and circle in the middle of the field. The values of the thresholds of green pixels in this case is 10 and the length of the scan line has to be larger than 20 pixels. Figure 4.23 illustrates an example of the white lines segmentation based on vertical search for transitions of the type green-white-green.

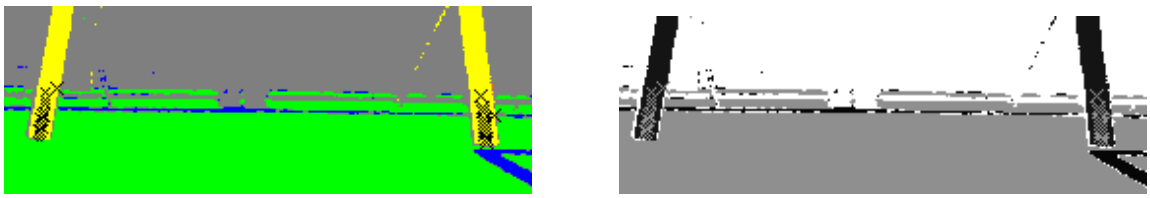


(a) (b)

Figure 4.23: On the left, the RGB painted image in which the centers of the scan lines are marked with black crosses. On the right, the corresponding index image.

4.6.3 Yellow/blue segmentation

For the detection of the blue/yellow goals horizontal scan lines are used for detecting only the lower half of the goals. Even though in this way only half of the color information related to the position of the goals is being used, it is enough and sufficiently relevant for an accurate detection of the goals. Horizontal scan lines are used in the search of green - yellow/blue - green transitions. The counter of the green pixels before and after de yellow/blue ones have to be larger than 10 and the number of yellow/blue pixels found has to be at least 25. An example of valid yellow horizontal scan lines is presented in Fig. 4.24.



(a) (b)

Figure 4.24: On the left, the RGB painted image in which the center of all valid yellow horizontal scan lines are marked with a black cross. On the right, the index image.

The values of the thresholds have been determined experimentally and they have the role of minimizing the number of false positives since the beginning of the detection algorithms.

Having in mind the fact that the robotic vision system is a real-time application which implies as low as possible processing times, the scanning of the images is done for every second row or/and column, respectively. This subsampling approach guarantees smaller processing times while still allowing the acquisition of reliable results.

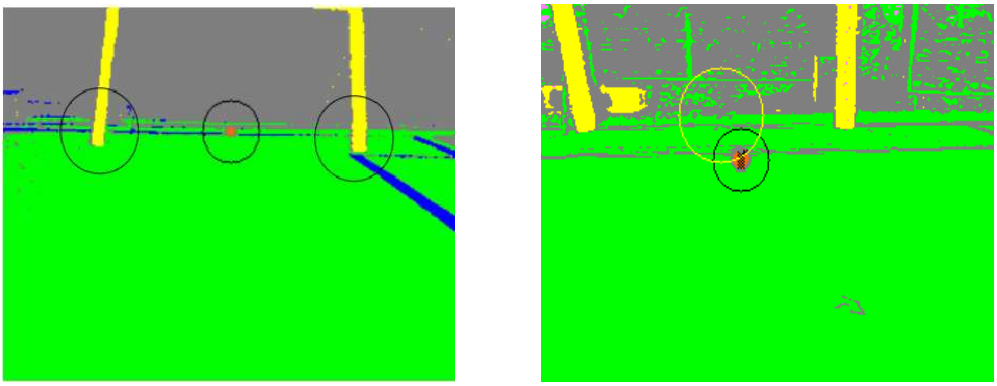
4.7 Cluster formation

Scan lines of a certain color of interest are formed by adjacent pixels that have the given color and that pass the threshold comparisons that were described in the previous section. After scanning the image, all the scan lines of a certain color are stored in a vector of scan lines. The next step is the formation of clusters from parallel scan lines that are close one to another. For this reason, the mass center of every scan line has to be known. The first scan line from the vector of scan lines is considered as part of a first cluster. From here on, all the following scan lines are analyzed and they are merged into clusters as follows:

- The distance between the mass center of the first cluster (containing only the first scan line) and the mass center of the second scan line in the vector is computed.
- If the distance between them is less than 15 pixels and the two scan lines are parallels, the second scan line is added to the cluster and the mass center of the cluster is updated accordingly.
- If the second scan line is not parallel or close enough to the first scan line, it will be the origin of a new cluster.
- The algorithm is repeated for all the scan lines of the same color that are contained by the vector.
- When there is more than one cluster, the distance between the mass center of a new scan line and the mass center of every cluster is calculated.
- The scan line will be added to the first cluster for which the distance between the mass centers respects the stated condition.
- If the scan line is not in the proximity of any already formed cluster, a new cluster will be started with that scan line.

The formation of clusters only makes sense for the segmentation of the goal posts and the ball. In the case of the white lines, the information about them that is sent to the other modules is an array of scanlines of the white color. An example of cluster detection is presented in Fig. 4.25.

The algorithm for the cluster formation considering a specific color is depicted in Algorithm 3. In the algorithm, color is considered the run length information of the color in a specific scan line.



(a) (b)

Figure 4.25: An example of cluster formation. The lower part of the yellow posts, as well as the orange ball are validated as yellow, respectively orange blobs.

Algorithm 3 Algorithm of the formation of clusters.

```
for i = 0; i < colorList.size; i++ do
  blob = NULL
  for j = 0; j < blobListSize; j++ do
    if distance(color, blobMassCenter) < threshold then
      blob = blobList[j]
    end if
  end for
  if blob! = NULL then
    blob.add(color)
    blob.update()
  else
    blob.createNewBlob()
    blob.add(color)
    blob.update()
  end if
end for
```

4.8 Object Detection

Having the blobs of colors formed is not enough for validating the blob as being one of the objects of interest. Not every orange blob in the green field is the ball as well as not every yellow or blue blob is a goal. Several measurements of the color blobs are used for validating the objects of interest.

For the yellow/blue goals the size of the blob is used for determining whether a yellow

blob is a goal or not. It has been proven experimentally that when the robot is centered on the furthest point on the field from the goals, the goals have the minimum size of 1500 pixels. Thus, in order to validate a yellow/blue blob as a goal, its size has to be larger than 1500 pixels. In the situation in which just one of the goals is visible to the robot, the mass center of the respective goal is returned. If more than two yellow clusters are validated as being the goals, only the two of them that are parallels are considered as being the goals. In this situation the middle point of the distance between the two points is returned by the detection method.

The detection of the ball is more complicated because at large distances from the robot the size of the ball can be very small, which could increase the number of false positives. The size of the orange cluster has to be larger than 45 pixels and when more than one cluster is found, the algorithm validates as being the ball the cluster whose size verifies the size condition and that is the closest to the robot. In the absence of the information about the pose of the robot, the mass center of the robot is considered to be the center column and the lower line of the image. Figure 4.26 is a graphical representation of the relation between the size of the ball in pixels and the distance from the robot at which it is found, in meters.

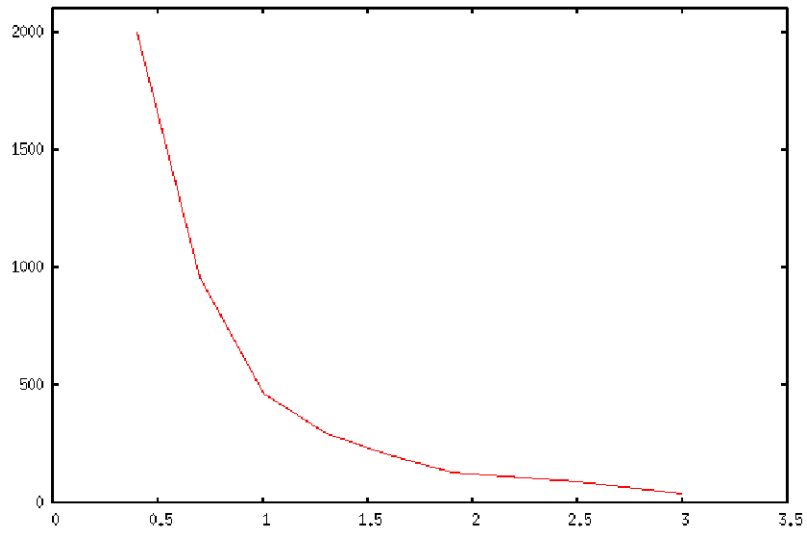
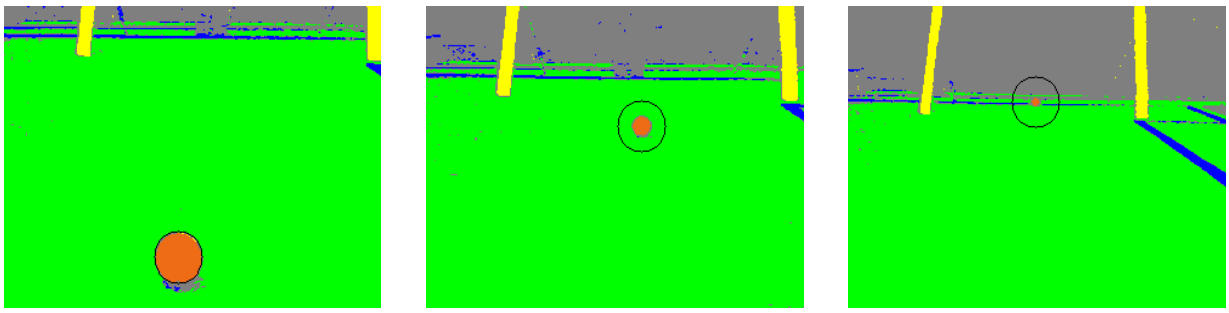


Figure 4.26: A graphic of the relation between the size of the ball and the distance from the robot at which it is found.

Moreover, Fig. 4.27 shows three examples of the ball at different sizes from the robot. The size of the ball in for each of the cases is presented.

Examples of the goals detection can be seen in Fig. 4.28. In (a) only one goal posts is seen by the robot, its mass center is marked. After having the information about the localization of the robot, merging that information with the information provided by the vision would let the robot know if he has to send the ball to his left or right. When both goal posts are visible for the robot (b), the middle of the goal is marked.

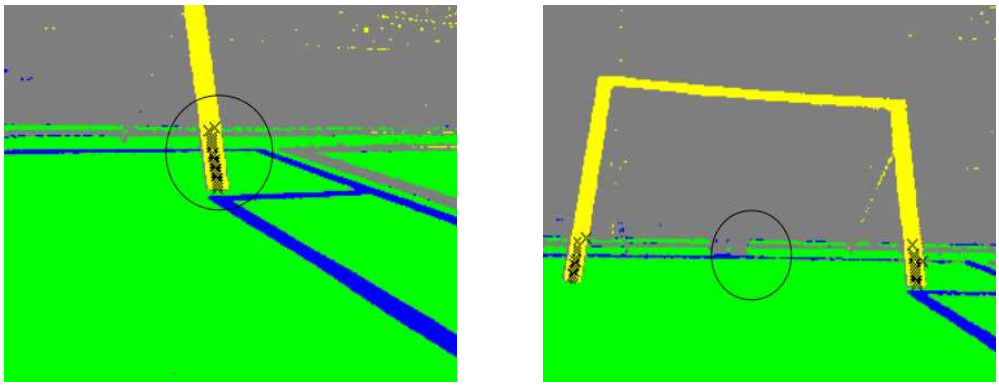


(a) distance = 50cm, size = 1360 pixels

(b) distance = 1.55m, size = 204 pixels

(c) distance = 3m, size = 48 pixels

Figure 4.27: Ball size at different distances from the robot.



(a)

(b)

Figure 4.28: On the left, the detection of just one post of the goal. On the right, the detection of the goal when both posts are visible.

Chapter 5

Experimental results

This chapter presents some results obtained with the proposed vision system. Even though the NAO robot was the main robotic platform on which the algorithms developed have been tested, the modularity of the proposed vision system has been proved by adapting it for the usage with another robotic platform, which was the Bioloid robot from Robotis. In Section 5.1 presents a series of results obtained with the NAO robot. Section 5.2 provides an introduction about the Bioloid platform, as well as an overview about the competition environment in which it has been used and finally, Section 5.3 presents the results that have been obtained with the Bioloid robot. All these results prove not only the efficiency of the proposed algorithms, but also the modularity of the vision system, which allows it to be used with several robotic platforms.

At the time of writing this document, the robots are still being prepared for participating in the RoboCup 2011 competition. Besides the vision process, several other processes run on the robot (agent, comm, rtdb) but not all of them are finalized in such way that allows the presentation of more detailed results that could transmit the idea of a robotic team. The results that will be presented are strictly related to the vision process.

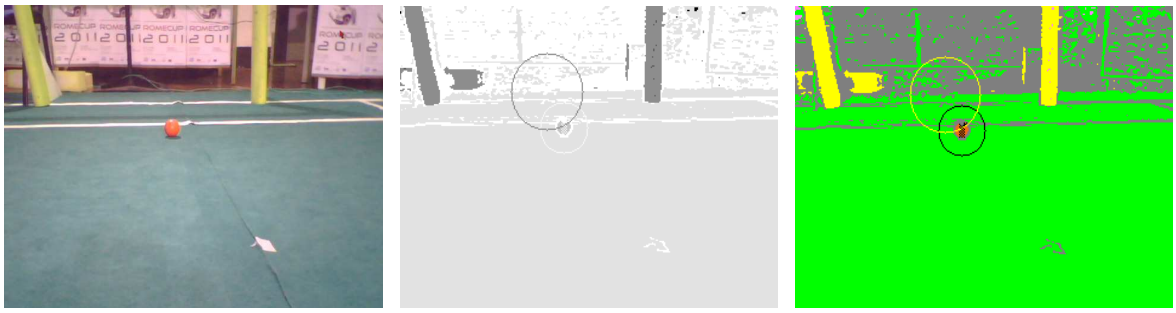
5.1 Results obtained with the NAO robot

This section presents a variety of results that were obtained by the proposed vision system as well as information about the processing times of the most important tasks that have been implemented. The images have been acquired on robotic fields that comply with the SPL standards. Some of the results were obtained on the SPL field during the RoboCup Mediterranean Open and some of them on an improvised field at the University of Aveiro. The field from the University of Aveiro is actually built inside the MSL field that the University already had and that has bigger measurements than the SPL field. For this reason, the lines of the SPL field were drawn in light blue in order to be different than the white lines of the MSL field, so that both robotic soccer teams of the university (MSL team CAMBADA and SPL

team Portuguese Team) could perform tests on the fields independently one from another.

The results that will be presented are related to the detection of the ball, of the goals and of the white lines. The information about the white lines is important for the localization process, while the information about goals can help in the correction of the calculated orientation of the robot, as well as to help in the kicking behavior. The ball is probably the most important object of interest, just like in the case of human soccer.

Figure 5.1 shows an original image acquired by the camera of the NAO robot, the index image corresponding to the original one and the painted image containing markers for the object of interest that have been detected. The black crosses represent orange points that are part of valid orange scan lines and the black circle stands for the detection of the ball. The black circle is constructed having the center in the center of mass of the orange blob formed by the validated orange scan lines. The yellow circle is a marker for the validation of the yellow goals. The center of the yellow circle is the middle of the distance between the two yellow blobs validated as being part of the yellow goals.



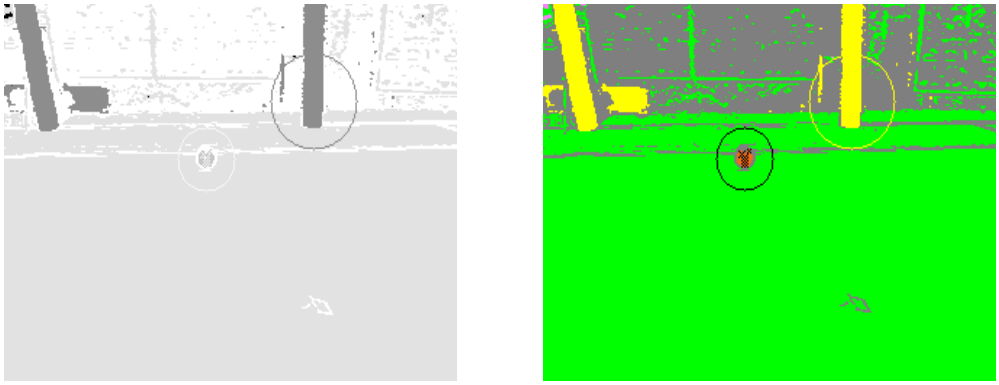
(a) (b) (c)

Figure 5.1: On the left, the original image. In the middle, the equivalent index image and on the right, the image "painted" according to the labels in the 8-bit image.

Not always both posts of the goals are being "seen" by the robot. Figure 5.2 is an example of a situation in which only a yellow post is validated as being part of the goal. The green information before the left yellow post is lost, so the yellow blob cannot be considered part of the goal. In this situation, the yellow circle marks only one validation of the goal posts and its center corresponds to the mass center of the yellow blob. After the robot is localized, it is possible to know if the one post that the robot sees is on its left or right side.

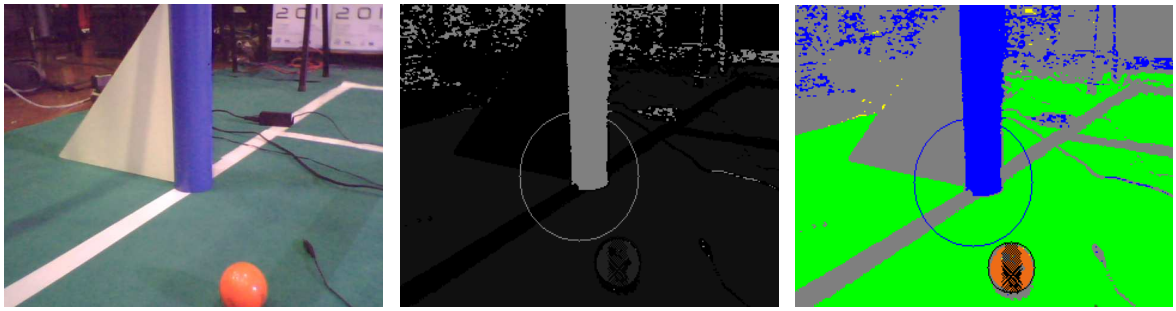
Another example of the detection of a single goal is shown in Fig. 5.3. The robot only "sees"one blue posts which is validated as being part of the blue goals.

Figure 5.4 includes also a detection of the white lines. In Fig. 5.4 (c) the green small crosses represent the two yellow valid blobs. The larger green cross marks the middle point between the two blobs. The red circle marks the center of the ball and the white crosses are valid white points that are part of the white lines of the field.



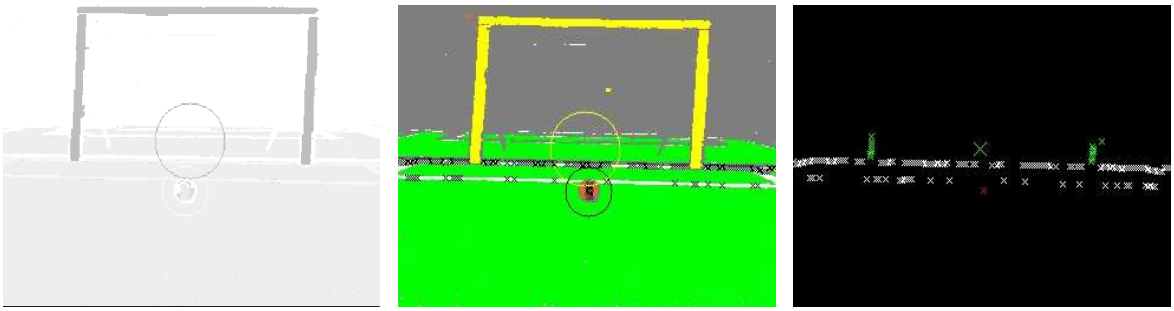
(a) (b)

Figure 5.2: On the left, the index image corresponding to the previous original image. On the right, the equivalent image "painted" according to the labels in the grayscale image. Because of changes in the illumination, some information about the green is lost and only one yellow post is validated.



(a) (b) (c)

Figure 5.3: On the left, the original image. In the middle, the index image and on the right, the painted image containing the markers for the ball and the blue goal.



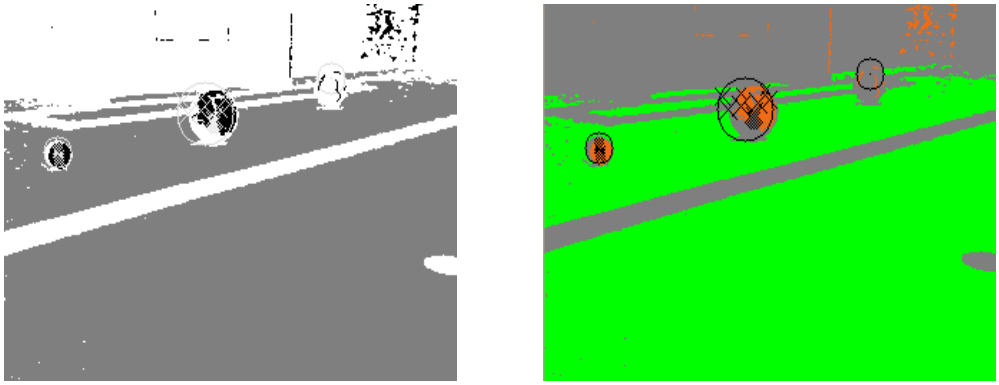
(a) (b) (c)

Figure 5.4: On the left, the index image. In the middle, the equivalent index image and on the right, an image containing only the markers of the objects of interest.

5.1.1 Ball detection

The ball is the most important object in the SPL games since the goal of every team is to mark as many goals as possible. The ball is also the most difficult object to detect not just because it is a moving object but also because at far distances from the robot, its size is very small.

When more than one orange blob is present in the image, the validation of the ball is made according to the distance from the mass center of the robot. The mass center of the robot is considered to be the center in terms of columns and last row of the image. Figure 5.5 is an example in which, having 3 orange blobs in the image, the closest one to the robot is validated as being the ball. The larger black circle stands for the validation of the ball, while the smaller circles are markers for the orange blobs that passed the size validation criteria.



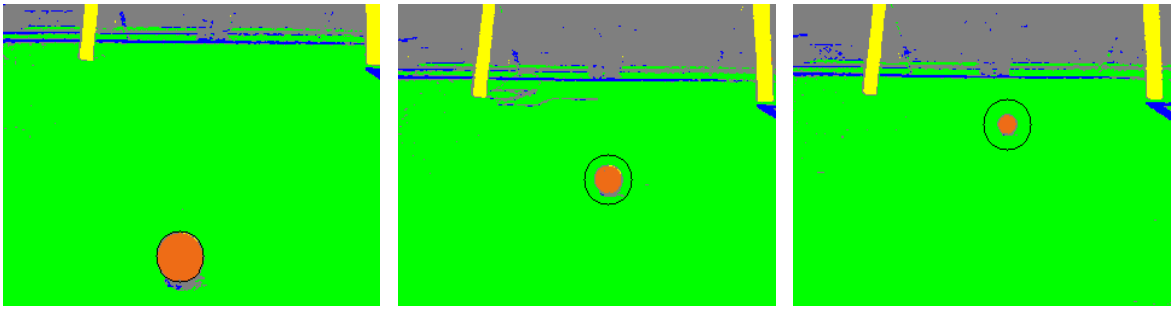
(a) (b)

Figure 5.5: On the left, the index image. On the right, the equivalent image "painted" according to the labels in the grayscale image and with the markers for all the valid orange blobs and for the blob that has been validated as being the ball.

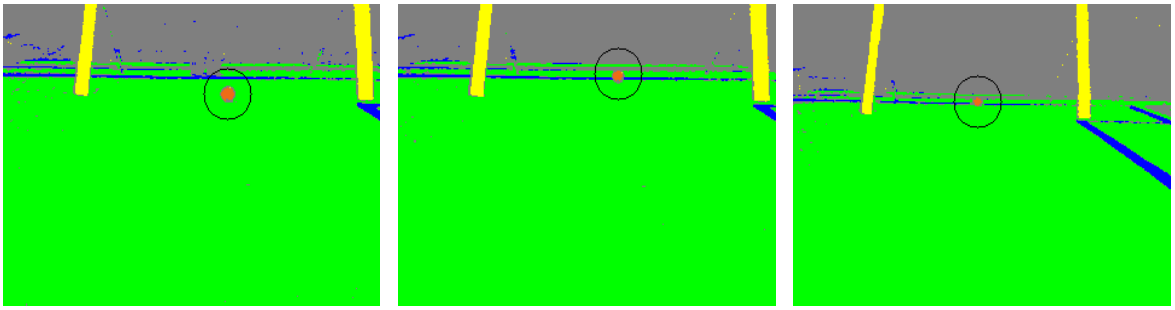
Figures 5.6 show the detections of the ball at different distances in front of the robot. The distance is increased gradually and the ball is still being detected.

In addition, Fig. 5.7 presents a graphic of the area of the ball according to the distance from the robot at which it is found. The measurements that are presented have been acquired under the following scenario: the robot was not moving, with the lower video camera perpendicular to the ground and the ball was manually placed at different distances in front of the robot. The distances vary from 0.4 to 2.7m, with a step of 0.3m. Several frames have been acquired for each of the positions of the ball and the size of the ball has been recorded for each of them. In the resulting graphic it can be seen the influence of the light in the segmentation of the ball. Due to the flickering of the illumination system of the field, the area of the ball slightly varies with every frame acquired.

Figure 5.8 shows the coordinates of the mass center of the ball for the same frames acquired in the previously described scenario. The coordinates of the mass center of the ball also vary



(a) distance = 50cm (b) distance = 1m (c) distance = 1.5m



(d) distance = 2m (e) distance = 2.5m (f) distance = 3m

Figure 5.6: Images showing the ball being detected at distances from 50cm to 3m.

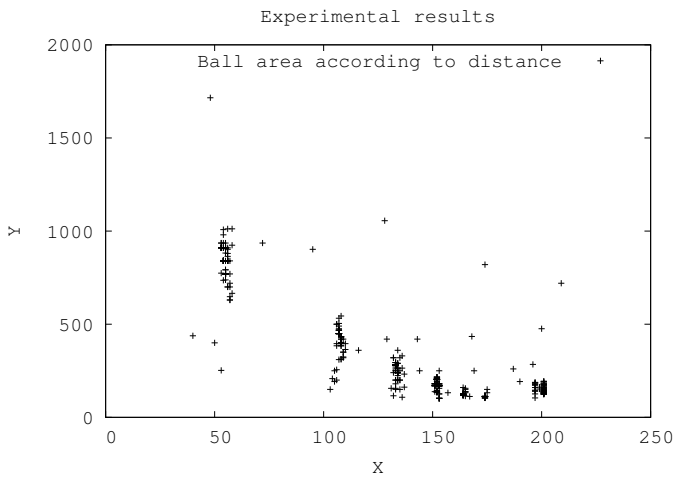


Figure 5.7: Graphic of the ball area according to the distance from the robot in the situation when the robot is not moving and the ball is placed in front of it at different distances.

for the ball not moving due to the same illumination issue.

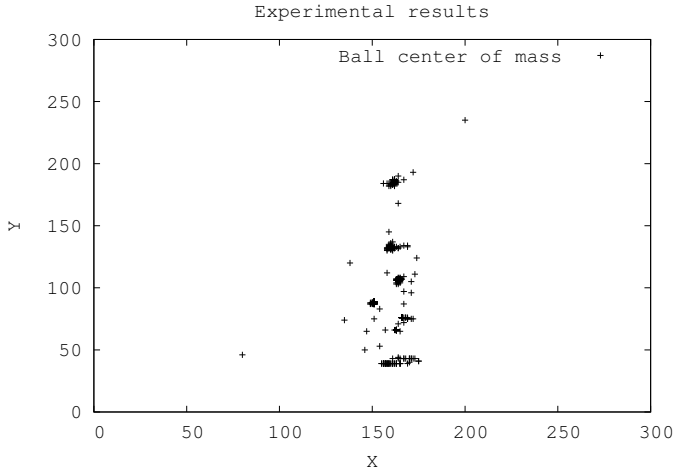


Figure 5.8: Coordinates in pixels of the mass center of the ball according to the distance from the robot in the situation when the robot is not moving and the ball is placed in front of it at different distances.

The situation in which the ball is fixed and the robot is walking towards it is described in Fig. 5.9. The graphic presents the coordinates of the mass center of the balls detected while the robot was moving towards the ball. Because of the movements of the robot, the coordinates of the mass center change in every frame. This result best show how difficult is to process images in these types of applications, when the robot is performing a type of locomotion.

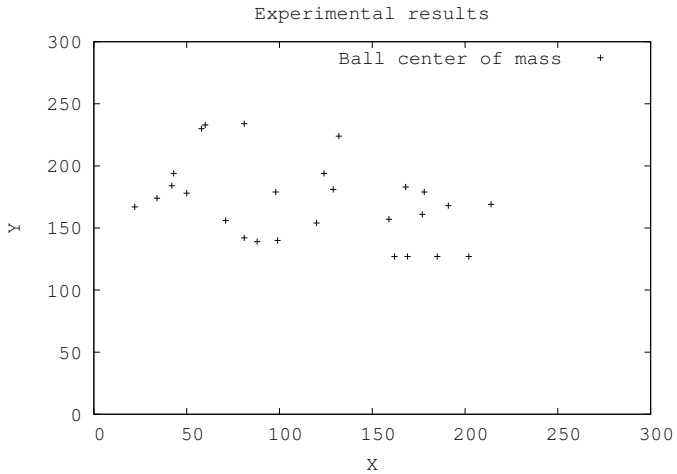


Figure 5.9: Coordinates in pixels of the mass center of the ball according to the distance from the robot acquired while the robot is moving towards a fixed ball.

As it has been described in Chapter 4, the size of the ball varies with the distance from the robot. Figure 5.10 is a graphic of the coordinates of the mass center of the ball in the following scenario: the robot is standing still while the ball is being sent towards it from a distance of approximately 2m.

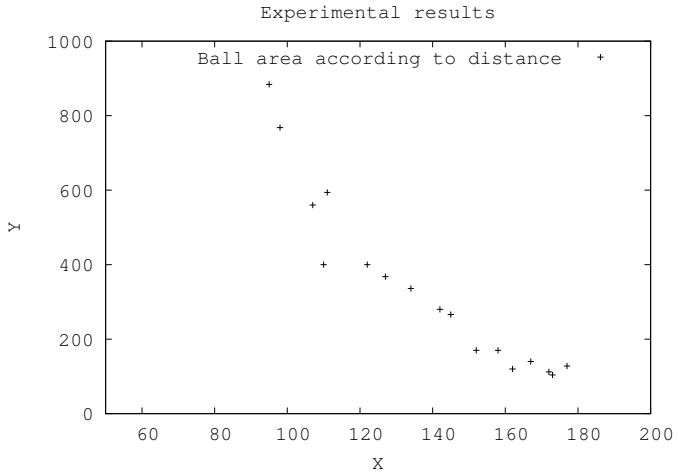


Figure 5.10: Coordinates in pixels of mass center of the ball in the situation when the robot is not moving and the ball is being sent towards it.

Another common scenario in the SPL games that has been tested is the detection of the ball when the robot is dribbling. In this case the only relevant information that can be extracted is the percentage of frames in which the ball is detected from all the acquired frames.

The performance of the vision system in terms of percentage of valid balls comparing to the number of frames acquired by the robot is recorded in Table. 5.1. The percentages represent frames in which the ball is seen and appropriately detected. The percentage of valid balls in almost all of the scenarios is high, the more critical situation being the one in which the robot is moving, as expected. Due to its type of locomotion, still images that give a good representation of the surrounding world are hard to acquire.

Scenario	Percentage
Robot dribbling the ball	93%
Robot stopped observing the ball	99%
Ball sent towards an imobile robot	99%
Robot moving towards the ball	30%

Table 5.1: Percentage of ball detections compared to the total number of frames acquired under various scenarios.

5.1.2 Processing time

Another important measure in real time systems is the processing time. As referred in Chapter 4, huge efforts have been made to optimize the software and the algorithms. The average processing time for the most time consuming tasks are presented in Table 5.2. Starting from the acquisition of the images, based on V4L2 API, passing through an efficient algorithm for subsampling, a LUT, scan lines and validation algorithms for the objects of interest, the reduced processing time obtained, in average 28ms, allows the use of the camera at a frame rate of 30 fps.

Task performed	Time
Acquiring an image	1ms
Conversion from YUV to index	15ms
Orange detection	4ms
Yellow detection	2ms
Blue detection	2ms
White lines detection	4ms

Table 5.2: Processing times spent by the vision process.

Regarding the auto-calibration procedure, Table 5.3 presents the time spent in the performance of the most important tasks of the algorithm. The main steps of the process are: reading the value of an intrinsic parameter (Read), setting the value of an intrinsic parameter of the camera (Write), calculating the MSV (MSV), calculating the error between the MSV value of the acquired frame and the desired value of 2.5 (MSV), compensating gain (Gain), exposure (Exposure), white balance values (Blue and Red) and calculating the average U and V for the white area (U and V).

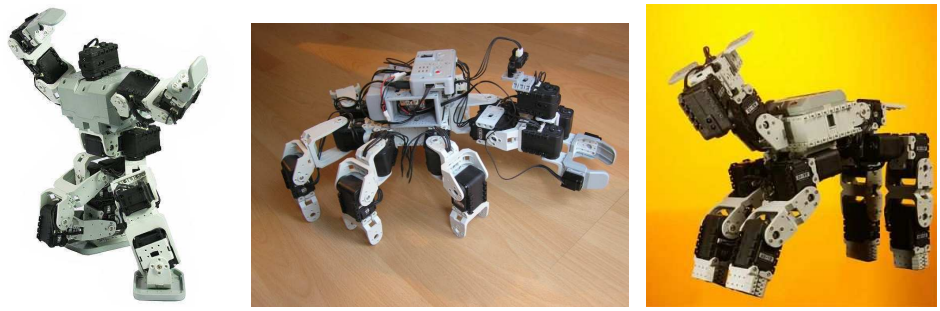
In terms of number of frames, the algorithm can be considered a fast one, since it only requires an average number of 20 frames for the parameters of the camera to converge. The times obtained do not allow the usage of this algorithm in real time, this being one of the most important future developments of the vision system that will be considered. The average total time spent by the self-calibration module is 10s when the calibration process starts from the intrinsic values of the camera in automode.

Mode	Read	MSV	Error	Gain	Exposure	U	V	Blue	Red	Write	Total
Auto	0ms	33ms	0ms	15ms	15ms	1ms	1ms	6ms	5ms	0ms	10s
0	0ms	34ms	0ms	17ms	18ms	1ms	1ms	6ms	6ms	0ms	46s
Max	0ms	34ms	0ms	15ms	16ms	1ms	1ms	6ms	6ms	0ms	1min

Table 5.3: Processing times spent by the main tasks of the self-calibration module, when the camera is started at different values of the intrinsic parameters. In the first situation, the camera starts in auto-mode, in the second situation the camera starts with all parameters set to 0. Finally, in the third situation the camera starts with all the intrinsic parameters set to their maximum values.

5.2 Bioloid

The Bioloid platform represents a robotic kit produced by the Korean robot manufacturer Robotis, which consists of several components, namely small servomechanisms Dynamixel, plastic joints, sensors and controllers which can be used to construct robots of various configurations, such as wheeled, legged, or humanoid robots.



(a) (b) (c)

Figure 5.11: Several examples of robots constructed by means of the Bioloid robotic kit.

The humanoid Bioloid (Fig. 5.11 (a)) represents the second humanoid platform on which the proposed vision system has been tested. The following subsections will present the changes that were necessary for having a functional Bioloid robot for the Micro Rato robotic competition [42], held every year at the University of Aveiro. The results that have been obtained with this platform are presented in Section 5.3.

5.2.1 The Micro-Rato competition

The Micro-Rato competition, held at the University of Aveiro is a competition between small autonomous robots whose dimensions do not exceed $300 \times 300 \times 400mm$ (Fig. 5.12). The competition is divided into two rounds: in the first one, all robots move from a starting

area with the purpose of reaching a beacon, in the middle of a maze. In the second round, the robots have to return to the starting area or at least to get as close as possible to it, using the information that they acquired during the first round.

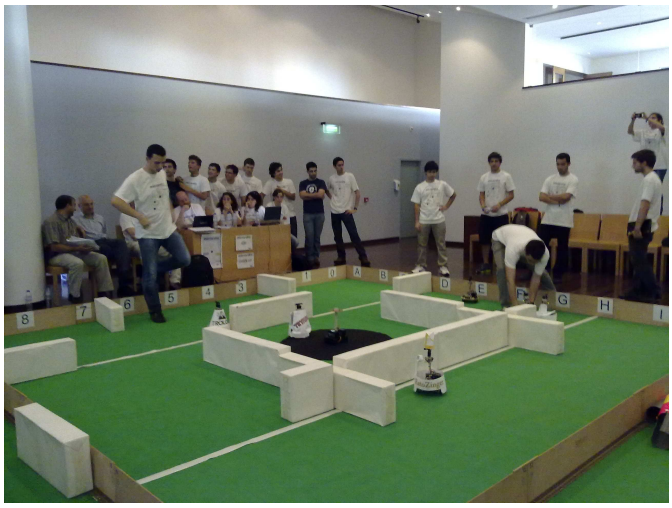


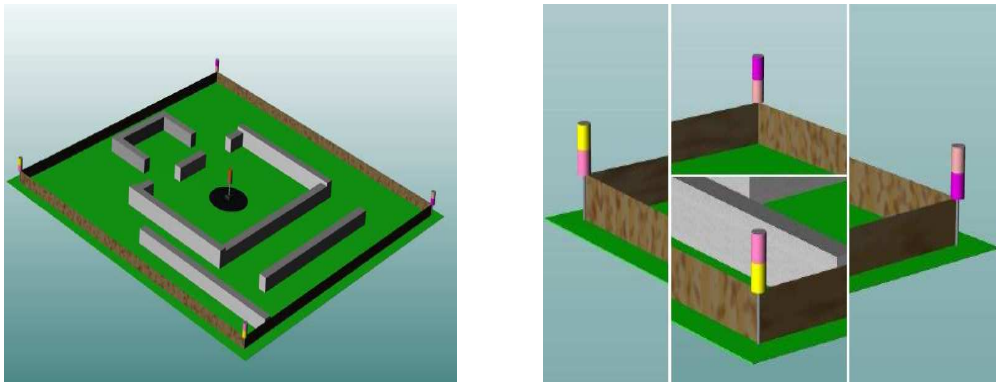
Figure 5.12: An image from the Micro Rato 2011 competition.

Most of the robots used in this competition do not rely on vision for accomplishing their tasks. It is more common the use of sensors for detecting the walls of the maze and the area of the beacon, which is an infrared emittor of 28cm high. However, the use of a vision system is possible since there are several elements that allow the detection of the obstacles, of the beacon and that can provide information about the localization of the robot.

The robots have to move on a green carpet and the walls of the maze are white (Fig. 5.13 (a)). Moreover, in each of the four corners of the maze there is a two-colored post and the beacon has also two predefined colors. Thus, the corner posts can have either one of the following color combinations: pink-blue, blue-pink, pink-yellow, yellow-pink, while the beacon is half orange, half pink (Fig. 5.13(b)). The information about the color combination of the posts is helpful for the localization of the robot, in the challenge of reaching the beacon.Therefore, by relying on visual information, it is possible to have a competitive humanoid robot in the context of Micro-Rato.

5.2.2 Bioloid vision system

The vision system that has been presented and initially tested on the NAO robots was also used with the Bioloid robot, after some small changes have been performed. The main changes that have been made were in the acquisition part, due to the fact that the Bioloid robot uses a standard USB camera, as described next, and also the colors of interest, as well as the objects of interest changed in the case of the Bioloid. The main idea behind the functioning of the vision system of the Bioloid is the same as for the NAO robot. A system of color classification



(a) (b)

Figure 5.13: On the left, an image of the Micro Rato field. On the right, a graphical representation of the four corner posts and the beacon [6].

is used for the following colors of interest: white, green, yellow, blue, pink and orange. After acquirinig an image, with the use of a look-up table, an 8-bit image of labels corresponding to the original one is constructed.

The next step of the process is the search for the colors of interest in the image of labels, using vertical scan lines and run-length coding. The information of the colors of interest are merged into blobs and based on several measurements calculated from the blobs they are validated as being objects of interest or not.

The video camera that was used with the Bioloid robot was a standard Logitech USB webcam and the process of acquiring images was different than in the case of NAO. The access of the device for the Bioloid camera was done by means of OpenCV, which provides several instinctive methods for accessing and displaying the images. The methods used by OpenCV also rely on Video For Linux v.2. This method was chosen instead of the acquisition module developed for the NAO robot since the NAO camera configuration is accessed through the I2C bus due to its special connection on the processing unit of the robot. No calibration of the camera intrinsic parameters was performed in this case since the device did not allow the access of its settings. The camera was used in auto mode in this application.

Another change that has to be made was the introduction of a new color of interest for the color classification process. Pink is not an meaningful color for the NAO soccer player, thus in the NaoCalib application had to be introduced the option of classifying the pink color. The last change concerns the search for transitions between colors of interest. For the NAO vision system, transitions between green and another colors of interest were used, so the green information was always used in the process of the color segmentation. In the case of the Bioloid robot, the method responsible for the search of transitions between colors of interest became more generic, allowing the search of transitions between any two colors. This change was needed because the detection of the posts is done by finding transitions between two

different colors of interest that are part of the posts.

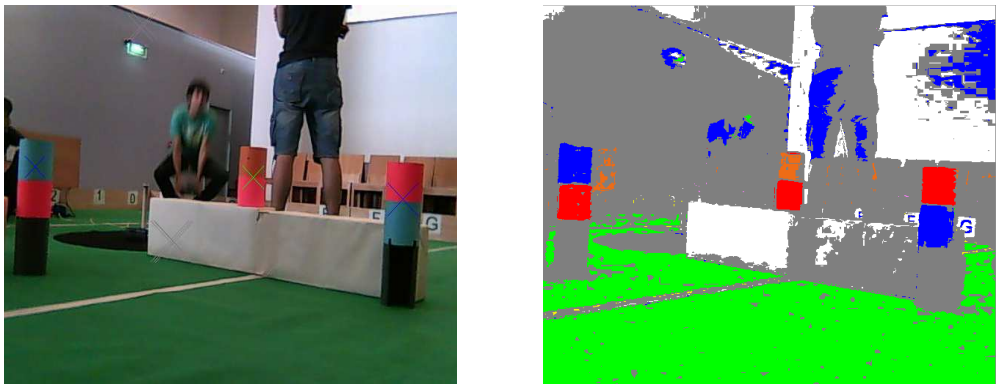
The main steps of the vision process are:

- Acquiring an image with a resolution of 640×480 pixels.
- Converting the color image into an index one with the use of a look-up table. The index image continues to be an 8-bit image in which all the colors of interest are mapped using a one bit per color relation. The resolution of the index image is 320×240 pixels and it was obtained by ignoring one in two columns and one in two rows of the original image with the purpose of reducing processing time.
- Vertical search lines are used for finding transitions between colors of interest. Transitions between yellow and pink, pink and yellow, pink and blue, blue and pink, orange and pink are searched for the detection of the posts and of the beacon. The four posts are placed in the four corners of the maze and are helpful in the challenge of reaching to the beacon. Also transitions between white and green are used for the detections of the walls of the maze which are to be avoided during the movements of the robot. The vertical search lines start with the first column of the image and continue progressively within the width of the image. For every search line, pixels are ignored as long as they are not of the first color of interest. Once a pixel of the colors of interest is found, a counter of the pixels of the same color is incremented. When no more pixels of the first color are found, pixels of the second color of interest will be searched. If there are no pixels of the second color of interest, the scan line is ignored and a new scan line will be started in the next column. Otherwise, a counter of the pixels having the second color of interest will be incremented. Before validating the scan lines the values of the two counters are compared to a threshold. Repeated experiments showed that an acceptable value for the threshold is 20 pixels.
- Clusters are formed from valid scan lines containing the same two colors of interest. The scan lines are grouped into clusters if they have the two colors of interest, in the same order and they are found at a distance of at most 50 pixels one from another. In this case, the clusters do not have the common meaning of a uniform region having a certain color, they stand for a region in the image having the sequence of two colors of interest.
- For each cluster, the area is calculated and in order to be validated as one of the posts, its area has to be in the range of [500,2000] pixels. For each valid cluster its mass center is computed. The size of the cluster is a good hint for the distance of the robot from the object.
- For the white-green transitions, clusters are not necessary and the information saved for further use is an array of scan lines containing transitions from white to green.

- The array of white-green transitions as well as the coordinates of the mass center for each post and for the beacon are loaded on the RtDB so that they can be accessed by other processes running on the robot.

5.3 Bioloid results

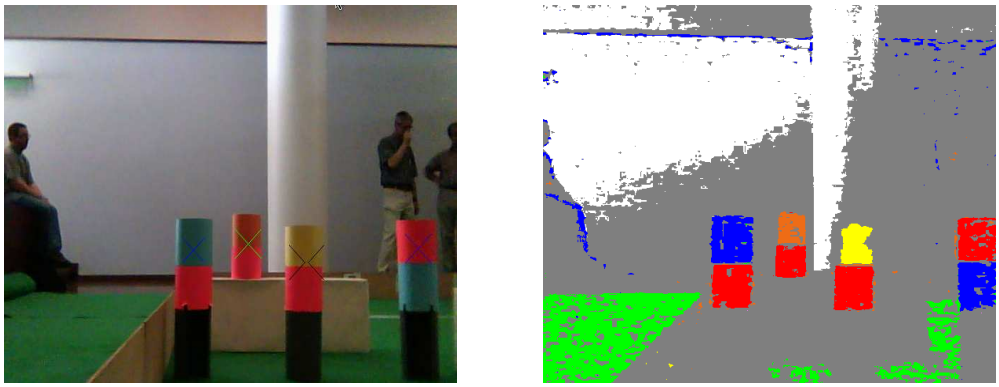
This section presents some results that were obtained by using the proposed vision system with the Bioloid humanoid robot during the Micro-Rato competition. Figure 5.14 shows an original image taken by the video camera connected to the Bioloid, and the corresponding color segmented image. The original image contains a marker (a cross) for every detected object of interest. Thus, all the posts are detected and also a part of the transitions from white to green. Not all transitions from white to green are detected since because of the illumination some of the white information is lost. Figure 5.15 shows another example of detection of the four types of colored posts.



(a) (b)

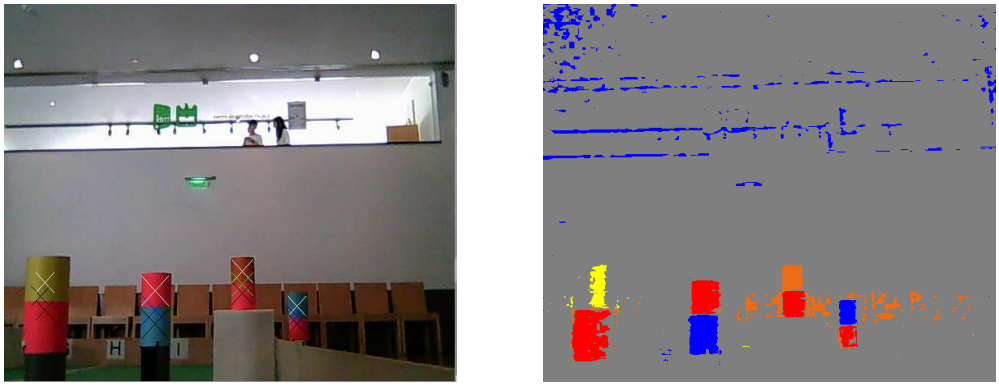
Figure 5.14: On the left, the original image acquired by the camera of the Bioloid robot, also containing the markers for the objects of interest. On the right, the corresponding color segmented image.

Figure 5.16 presents some more detailed results for the detection of the posts. For each posts, the center of mass of each of the two color blobs is marked and also the center of mass of the post is marked. The mass center of the post is found at the middle of the distance between the mass centers of the two color blobs that form the post. The four posts are found at different distances from the robot.



(a) (b)

Figure 5.15: On the left, the original image with the markers for all the posts. On the right, the color segmented image.



(a) (b)

Figure 5.16: On the left, the original image having a marker for each color blob detected and also a mark for the mass center of each post. The blue/pink post is situated the furthest possible from the robot, the length of the maze, and it is still being detected. On the right, the color segmented image.

Chapter 6

Conclusions and future work

The work presented in this thesis addressed the field of robotic vision and presented a proposal for a modular vision system that can be used for different classes of humanoid robots. The vision system has been tested and used for the NAO and Bioloid humanoid robots and for each of them results have been shown.

The vision system that has been presented has the main advantage of being a modular one, this meaning that with a small number of changes it can be applied to different classes of robots. Moreover, it runs in real-time, this being one of the most important features that a solid vision system should have. The system has been implemented from scratch and it can be divided into three main modules such as: acquisition of the images, self-calibration of the camera intrinsic parameters and color segmentation and object detection.

The self-calibration algorithm for the camera parameters is an innovative and fast converging one. The results of the processing are highly influenced by the quality of the images acquired and a good calibration of the camera settings allows achieving much better results than when having the camera working in auto-mode.

The module of color segmentation and object detections presented an approach for detecting objects of interest for a robot based on color information. The process of color segmentation is based on horizontal and vertical scan lines used for the search for transitions between two colors of interest. From adjacent scan lines blobs of the same color are formed and further on they are validated as objects of interest after the analysis of different features extracted from the blobs.

Apart from the vision process running in real-time on the robot, two other applications have been developed for the purpose of debugging and color calibration of the colors of interest, based on a client-server architecture. *NaoViewer* is a client that can receive and display both original or segmented images acquired by the camera of the robot and *NaoCalib* is an application used for a manual calibration of the colors of interest.

All the software implemented has been tested on the NAO soccer player robots of the SPL team of University of Aveiro and University of Porto, Portuguese Team. Moreover,

the algorithms have suffered minor alterations that allowed them to be applied to a Bioloid humanoid robot of the team MusErectus for a demonstration during the Micro Rato robotic contest, held at the University of Aveiro.

The effectiveness of the proposed vision system has been proved along this document by pictures showing the results of the algorithms and by the reduced processing times that have been acquired. Moreover, the participation of the Portuguese Team at RoboCup RomeCup 2011 and the demonstration of MusErectus team at Micro Rato strengthen the reliability of the vision system described in this thesis.

6.1 Future work

Robotic vision is a research area in which developments and improvements are an ongoing process. Providing an accurate representation of the surrounding world for a robot is a very difficult task that can only be achieved in small steps that can make a transition from particular descriptions to more generalized one. For this reason, several future developments can be brought to the presented project.

- First, merging the information of the pose of the robot with the information provided by the camera of the robot can significantly improve both the processing time spent and the results of the algorithm. By knowing the pose of the robot certain areas in the image containing the body of the robot could be excluded from processing, thus decreasing the global amount of processing time. Moreover, the pose of the robot could be a helpful information when computing distances from the objects of the interest or vision angles.

- Visual information should also be used for the detection of the team markers in a soccer game. The algorithms used for the segmentation of the colors of interest could be used for the segmentation of the team markers.

- Future developments of this work also include more validation criteria for the ball detection based on feature extraction and classifiers training which are more generic and are not color dependent. Algorithms like Speeded-Up Robust Features (SURF [43]) or Scale-Invariant Feature Transform (SIFT [44]) can be a good choice for the task of improving the object detections. The rules in robotic soccer are evolving from year to year, thus pushing the research in this area to become focused on more and more generic implementations that are closer to the rules for the human soccer. Thus, soon the objects of interest will no longer be color coded and solutions for their detections have to be provided.

- The choice of the color space used in the algorithms of image processing turns out to be a very important part of the performance of the vision system. Different implementations of robotic vision systems are based on different color spaces, all claiming that the specific

color space provides best results. A deeper study on the performances of the most used color space could be a helpful tool in the future development of this work.

- Another important improvement that could be brought to the work presented would be the adjusting of the self-calibration algorithm of the intrinsic parameters of the camera, so that it could be used in real time. Since the algorithm uses a white area whose position in the image is already known, this part could be improved once the pose of the robot is estimated. The fixed white are that is now use could be replaced by a white area on the body of the robot, to which he could "look" when adjusting the white balance parameter. Moreover, solutions for reducing the time spent by this process should be studied.

Appendix A

Modules of the vision system

The main classes and methods implemented for the modular vision systems described in this thesis are presented here. Detailed documentation of all software can be obtained from the source code using the *doxygen* documentation program.

• Class NaoCamera

`NaoCamera()` - Constructor

`NaoCamera()` - Destructor

`void initOpenI2CAdapter()` - Opens the I2C adapter

`void initSelectCamera(Camera camera)` - Selects camera

`void initOpenVideoDevice()` - Opens the video device

`void initSetCameraDefaults()` - Initializes default parameters of the camera

`void initSetImageFormat()` - Sets the format of the image

`void initSetFrameRate()` - Sets the frame ret

`void initRequestAndMapBuffers()` - Maps buffers

`void initQueueAllBuffers()` - Queues buffers

`void initDefaultControlSettings()` - Set default values for the control settings of the camera

`void startCapturing()` - Starts the capture

`void setSettings(const CameraSettings& settings)` - Sets the camera control settings

`const CameraSettings& getSettings()` const - Reads the camera control settings

`bool captureNew()` - Captures new frame

`const unsigned char* getImage()` const - Returns the last captured image

`unsigned getTimeStamp()` const - Time stamp of the last captured image

`Camera switchToUpper()` - Switches to upper camera

`Camera switchToLower()` - Switches to lower camera

`Camera switchCamera(Camera camera)` - Switches to the camera that is not currently in use

`Camera getCurrentCamera()` - Returns the camera currently in used

`void assertCameraSettings()` - Asserts that the actual camera settings are correct

`void writeCameraSettings()` - Writes the camera control settings

- Class Config

`Config(const char* fileName)` - Constructor
`void CreateDefault()` - Creates a default configuration file with all fields set to 0
`virtual  Config()` - Destructor
`CameraSettings &getCamSettings()` - Reads the camera settings from the configuration file
`ColorRange *colors()` const - Reads the color range of the colors of interest from the configuration file
`void setCamSettings()` - Writes the camera settings to the configuration file
`int LoadAscii(char *fName)` - Reads a text configuration file
`int SaveAscii(char *fName)` - Saves a text configuration file
`int LoadBinary()` - Reads a binary configuration file
`int SaveBinary()` - Saves a binary configuration file
`void Print()` - Prints all information retrieved from the configuration file
`const char* getFilename() const` - Returns the name of the configuration file

- Class Lut

`Lut()` - Default constructor
`Lut(Config& config)` - Constructor
`Lut()` - Destructor
`void init(ColorRange* cr)` - Initializes LUT

- Class Calibration

`Calibration()` - Constructor
`Calibration()` - Destructor
`CvHistogram* calcHistogram()` - Calculates the histogram of an image
`void update(const unsigned char *b)` - Updates the information about the statistical measurements
`void drawHistogram(IplImage* image, CvHistogram* hist)` - Draws the histogram of an image
`float calcMean(CvHistogram* hist)` - Calculates the mean value
`float calcMSV(CvHistogram* hist)` - Calculates MSV
`float calcACM(CvHistogram* hist)` - Calculates ACM
`void calibrateGain(CameraSettings &s, NaoCamera &Ncam, float err)` - Compensation of the gain parameter
`float UpdateMSV()` - Updates the MSV value
`void calibrateWB(CameraSettings &s, NaoCamera &cam, int errwb)` - Compensation of the blue chroma

`void calibrateWR(CameraSettings &s,NaoCamera &cam, int errwr)` - Compensation of the red chroma
`unsigned int getU(CvRect rect)` - Calculates the average U value for the white area in the image
`unsigned int getV(CvRect rect)` - Calculates the average V value for the white area in the image

- Class PI

`PI(float P, float I, float m, float M)` - Constructor
`pi(const pi& cpi)` - Copy constructor
`int compensate(float value, float err)` - Implements the PI compensation

- Other methods

`IplImage* Yuv422_to_Index(const unsigned char* buf, unsigned nCols, unsigned nRows, Lut &lut)` - Converts a YUV422 image to an index one
`IplImage* Yuv422_to_Index_Fast(const unsigned char* buf, unsigned nC, unsigned nR, Lut &lut)` - Converts a YUV422 image to an index one based on the type casting of the unsigned char buff
`IplImage* Yuv422_to_Yuv444(const unsigned char* buf, unsigned nCols, unsigned nRows)` - Converts a YUV422 image to a YUV444 image
`IplImage* Yuv422_to_GRAY(const unsigned char* buf, unsigned nCols, unsigned nRows)` - Converts a YUV422 image to a grayscale one
`void yuv_to_rgb(int y, int u, int v, int* r, int* g, int* b)` - Transforms YUV pixel values to RGB pixel values
`void rgb_to_hsv(int r, int g, int b, int* h, int* s, int* v)` - Transforms RGB pixel values to HSV pixel values
`void yuv_to_hsv(int y, int u, int v, int* h, int* s, int* vv)` - Transforms YUV pixel values to HSV pixel values
`IplImage* paintRGB(IplImage* img)` - Converts an index image into an RGB painted one
`int Distance(CvPoint p1, CvPoint p2)` - Calculates the distance between two points
`void DrawX(IplImage *img,CvPoint center,int d, CvScalar color)` - Draws a cross on the image
`void DrawCircle(IplImage *img, CvPoint center, int r, CvScalar color)` - Draws a circle on the image
`void FindTransVert(IplImage *img_idx, unsigned color, int index, vector< vector<ScanLine> > &obj)` - Searches for vertical transitions between colors
`void FindTransHoriz(IplImage *img_ids, unsigned color, int index,vector< vector<ScanLine> > &obj)` - Searches for horizontal transitions between colors `void FormCluster(vector<Blob> &blobs, vector<ScanLine> lines)` - Aggregates scan lines to clusters
`int findBlobs(vector<Blob> &blobs, ScanLine &sc)` - Forms blobs of the same color

`void UpdateBlob(Blob &b, ScanLine sc)` - Update the information about the blobs
`int ValidateGoals(vector<Blob> &b, CvPoint &cm)` - Validates the yellow/blue blobs as goals
`int ValidateBall(vector<Blob> &b)` - Validates the orange blob as ball
`int createSocket( int *socketId, unsigned short localPort )` - Creates a new socket on the local machine
`int connectTo( int socketID, char* destHost, unsigned short destPort )` - Connects the selected socket to a destination
`int waitForCall( int socketID, int* acceptedSocket, char* fromHost,unsigned short* fromPort )` - Waits for a connection on the socketId socket
`int sendData( int socketId, void* data, int dataLength )` - Sends the data contents to the socket's destination
`int receiveData( int socketId, void* buffer, int* bufferSize )` - Returns anything that arrives to the socket

Appendix B

User's manual

This appendix explains how to use the applications developed for the robots.
⋆ Vision

The "vision" application represents the main program of the vision software. It is responsible for the entire vision process that runs on the robot, from video acquisition, to camera calibration and object detection. Vision can be run with the following parameters:

- **-h** - displays a help menu of the program
- **-cf #config.file** - loads a configuration file with the name specified by #config.file. If no configuration file is specified, a default configuration file with the name "nao.conf" will be created and used. In the default configuration file all the field of the file are set to 0.
- **-auto** - uses the camera in automode
- **-calib** - starts the self-calibration of the camera intrinsic parameters. When the calibration module is run, the configuration file should also be specified since the camera settings will be written to it. If no configuration file is specified, a default one with the name "nao.conf" will be used.
- **-server** - works as a server that expects a client to connect in order to start sending frames. When run in server mode, several other command line arguments are needed. These are presented as follows:
- **-server period** # - a numerical value for the period at which a frame is sent
- **-f** # - a flag that can be 0 if the YUV422 buffer will be sent to the client or 1 if the buffer of the index image will be sent

⋆ NaoViewer

NaoViewer is a client that connects to the vision server and can display either the frames acquired by the robot in RGB format, or the index and the corresponding "painted" images. NaoViewer has to be run with the following command line arguments:

- **-ip** - IP of the robot on which the vision server is running
- **-f** # - the same flag that is needed in the server part. If the flag is 0 the client will display RGB images, else, the application will display the index and the "painted" images containing the markers for the objects of interest detected.
- **-cf #config.file** - the configuration file

⋆ NaoCalib

NaoCalib is the application used for the calibration of the colors of interest. When run, the arguments that is requires are:

- **-ip** - IP of the robot on which the vision server is running
- **-cf #config.file** - the configuration file

When NaoCalib is started, the user can calibrate the colors of interest by means of sliders which are displayed when pressing the **"y"** key. The sliders are manipulated with the help of the mouse and are used for setting the H, S and V range of all the colors of interest. The H, S and V histograms are also displayed next to the sliders with the purpose of better understanding the classification process. The user can place the mouse over a pixel that has one of the colors of interest and by pressing the **"i"** key, the H, S and V values of the selected pixel will be displayed on the screen. When the classification process is done, pressing **"u"** will update the information about the color ranges in the configuration file. Finally, pressing **"q"** will exit the application.

Bibliography

[1] Aldebaran Robotics official website. `http://www.aldebaran-robotics.com`. Last visited June, 2011.

[2] A. Neves, J. Azevedo, N. Lau B. Cunha, J. Silva, F. Santos, G. Corrente, D. A. Martins, N. Figueiredo, A. Pereira, L. Almeida, L. S. Lopes, and P. Pedreiras. *CAMBADA soccer team: from robot architecture to multiagent coordination*, chapter 2. I-Tech Education and Publishing, Vienna, Austria, In Vladan Papic (Ed.), Robot Soccer, 2010.

[3] Frederico Santos, Luis Almeida, Luis Seabra Lopes, José Luís Azevedo, and Manuel Bernardo Cunha. Communicating among robots in the robocup middle-size league. In *RoboCup 2009: Robot Soccer World Cup XIII*, Lecture Notes in Artificial Intelligence. Springer, 2009.

[4] Wikipedia-The Free Encyclopedia. `http://www.wikipedia.org`. Last visited May, 2011.

[5] `http://www.fourcc.org/`. Last visited May, 2011.

[6] www.microrato.ua.pt. Rules of the Micro Rato competition, 2011.

[7] A. J. R Neves, N. Lau, L.P. Reis, and A. Moreira. Portuguese Team Team Description. RoboCup 2011, Istanbul, 2011.

[8] RoboCup Standard Platform League official website. `http://www.tzi.de/spl`. Last visited June, 2011.

[9] Robotis Official Website. `http://www.robotis.com/xe/`. Last visited June, 2011.

[10] T. Acharya and A. Ray. *Image Processing: Principles and Applications*. Wiley New York, 2005.

[11] ASIMO the world's most advanced humanoid robot Official Website. `http://asimo.honda.com/`. Last visited June, 2011.

[12] NASA Official Website. `http://www.nasa.gov/`. Last visited May, 2011.

[13] RoboCup official website. `http://www.robocup.org`. Last visited June, 2011.

[14] FCPortugal official website. http://www.ieeta.pt/robocup/. Last visited May, 2011.

[15] A. J. R. Neves, L. Azevedo, M. B. Cunha, N. Lau, A. Pereira, G. Corrente, F. Santos, D. Martins, N. Figueiredo, J. Silva, J Cunha, B. Ribeiro, R. Sequeira, L. Almeida, L. S. Lopes, J. M. Rodrigues, and A. J. Pinho. CAMBADA Team Description. RoboCup 2010, Singapore, 2010.

[16] 5DPO official website. http://paginas.fe.up.pt/~robosoc/en/doku.php. Last visited May, 2011.

[17] RoboCup Mediterranean Open Official Website. http://www.robocup-mediterranean-open.org/. Last visited March, 2011.

[18] Sony Aibo Europe Official Website. http://support.sony-europe.com/aibo/index.asp. Last visited May, 2011.

[19] A. Louloudi, A. Mosallam, N. Marturi, P. Janse, and V. Hernandez. Integration of the humanoid robot nao inside a smart home - a case study. Technical report, School of Science and Technology, Orebro University, Sweden, April 2010.

[20] T. Rofer, T. Laue, C. Graf, T. Kastner, A. Fabisch, and C. Thedieck. B-Human Team Description. RoboCup 2010, Singapore, 2010.

[21] Video For Linux 2 API Specifications. http://v4l2spec.bytesex.org/. Last visited June, 2011.

[22] T. Rofer, T. Laue, J. Muller, A. Burchardt, E. Damrose, A. Fabisch, F. Feldpausch, K. Gillmann, C. Graf, T. J. Haas, A. Hartl, D. Honsel, P. Kastner, T. Kastner, B. Markowsky, M. Mester, J. Peter, O. Riemann, M. Ring, W. Sauerland, A. Schreck, I. Sieverdingbeck, F. Wenk, and J. Worch. B-Human Team Report and Code Release 2010, 2010.

[23] S. Barrett, K. Genter, M. Hausknecht, T. Hester, P. Khandelwal, J. Lee, A. Tian, M. Quinlan, M. Sridharan, and P. Stone. TT-UT Austin Villa Team Description. RoboCup2 2010, Singapore, 2010.

[24] E. Hashemi, O. A. Ghiasvand, M. G. Jadidi, A. Karimi, R. Hashemifard, M. Lashgarian, M. Shafiei, S. Mashhadt, K. Zarei, F. Faraji, M. A. Z. Harandi, and E. Mousavi. MRL Team Description. RoboCup 2010, Singapore, 2010.

[25] leader@austrian kangaroos.com. Austrian Kangaroos Team Description. RoboCup 2010, Singapore, 2010.

[26] CMVision Official Website. http://www.cs.cmu.edu/~jbruce/cmvision/. Last visited May, 2011.

[27] S. Liemhetcharat, B. Coltin, C. Mericli, and M. Veloso. CMurfs Team Description. RoboCup 2010, Singapore, 2010.

[28] H. L. Akin, T. Mericli, E. Ozukur, C. Kavaklioglu, and B. Gokce. Cerberus Team Description. RoboCup 2010, Singapore, 2010.

[29] D. Garcia, E. Carbajal, C Quintana, E. Torres, I. Gonzalez, C. Bustamante, and L. Garrido. Borregos Team Description. RoboCup 2010, Singapore, 2010.

[30] R. Middleton. NUIM Team Description. RoboCup 2010, Singapore, 2010.

[31] TJArk.office@gmail.com. TJArk Team Description. RoboCup 2010, Singapore, 2010.

[32] Gary Bradski and Adrian Kaehler. *Learning OpenCV*. O'Reilly, first edition, September 2008.

[33] Antonio J. R. Neves, Armando J. Pinho, Daniel A. Martins, and Bernardo Cunha. An efficient omnidirectional vision system for soccer robots: from calibration to object detection. *Mechatronics*, 21(2):399–410, March 2011.

[34] Ivo dos Santos Pinheiro. Automatic calibration of the cambada team vision system. Master's thesis, Universidade de Aveiro, 2008.

[35] A. J. R Neves, A. J. Pinho, D. A. Martins, and B.Cunha. An efficient omnidirectional vision system for soccer robots: From calibration to object detection. *Mechatronics Journal*, 2010.

[36] P. M. R. Caleiro, A. J. R. Neves, and A. J. Pinho. Color-spaces and color segmentation for real-time object recognition in robotic applications. *Revista do DETUA*, 4(8):940–945, June 2007.

[37] L. Almeida, P. Pedreiras, and J. A. Fonseca. The ftt-can protocol: Why and how. *IEEE Transactions on Industrial Electronics*, 49:1189–1201, 2002.

[38] A. Neves, J. Azevedo, N. Lau B. Cunha, J. Silva, F. Santos, G. Corrente, D. A. Martins, N. Figueiredo, A. Pereira, L. Almeida, L. S. Lopes, and P. Pedreiras. *CAMBADA soccer team: from robot architecture to multiagent coordination*, chapter 2. I-Tech Education and Publishing, Vienna, Austria, In Vladan Papic (Ed.), Robot Soccer, 2010.

[39] Open Source Computer Vision Library (OpenCV). `http://sourceforge.net/projects/opencvlibrary/`. Last visited June, 2011.

[40] A. J. R. Neves, A. J. Pinho B. Cunha, and I. Pinheiro. Autonomous configuration of parameters in robotic digital cameras. volume 5524 of *Lecture Notes in Computer Science*, pages 80–87. Springer, June 2009.

[41] Piyush Khandelwal, Matthew Hausknecht, Juhyun Lee, Aibo Tian, and Peter Stone. Vision calibration and processing on a humanoid soccer robot. In *The Fifth Workshop on Humanoid Soccer Robots at Humanoids 2010*, December 2010.

[42] Micro-Rato Official Website. `http://microrato.ua.pt`. Last visited June, 2011.

[43] L. Juan and O. Gwun. A comparison of sift, pca-sift and surf. *International Journal of Image Processing (IJIP)*, 3(4), 2009.

[44] H. Bay, T. Tuytelaars, and L. Van Gool. Object recognition from local scale invariant features. *European Conference on Computer Vision*, 2006.