Universidade de Aveiro
2015

Departamento de Electrónica,
Telecomunicações e Informática
Programa de Doutoramento em Informática
das Universidades de Aveiro, Minho e Porto

Alina Liliana Trifan

Deteção de objetos coloridos em robôs inteligentes com restrições temporais

Time-constrained colored object detection for intelligent robots

Universidade de Aveiro
2015

Departamento de Electrónica,
Telecomunicações e Informática
Programa de Doutoramento em Informática
das Universidades de Aveiro, Minho e Porto

Alina Liliana Trifan

Deteção de objetos coloridos em robôs inteligentes com restrições temporais

Time-constrained colored object detection for intelligent robots

Dissertação apresentada às Universidade de Aveiro, Minho e Porto para cumprimento dos requesitos necessários à obtenção do grau de Doutor em Informática, realizada sob a orientação científica de António José Ribeiro Neves e Manuel Bernardo Cunha, Professores do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro

Trabalho financiado pelas seguintes entidades:

agradecimentos

Primeiro que tudo, gostava de agradecer ao meu orientador, Professor António Neves, pela orientação, partilha de conhecimentos e todo o apoio durante o meu doutoramento. Agradeço a sua paciência, o seu otimismo contagiante e todas as conversas inspiradoras que me ajudaram concluir esta Tese. Estou agradecida ao meu co-orientador, Professor Bernardo Cunha, pelo seu entusiasmo e por todas as revisões do meu trabalho. Queria também agradecer ao Professor José Luís Azevedo por toda a sua ajuda e pelas palavras de incentivo durante cada competição robótica. À Universidade de Aveiro, especialmente ao IEETA e ao DETI, agradeço por me providenciarem as condições necessárias para desenvolver este trabalho e todo o apoio financeiro prestado nas mais diversas ocasiões. Esta Tese não podia ter sido realizada sem a cooperação dos meus colegas do laboratório IRIS, principalmente os que fazem parte do projeto CAMBADA. O tempo que passei na Universidade de Aveiro não teria sido igual sem a amizade dos meus colegas Aneesh e Bruno. Aos meus amigos Diana, Cristina, Paul, Annamaria e Marisa, um grande obrigada por estarem sempre perto, mesmo estando à milhares de kilometros. Ao Tozé, obrigada por partilhares a minha paixão por uma vida mais saudável e por seres o meu companheiro de treino. À minha mãe, um muito obrigada por me ter permitido seguir os meus sonhos. Por último, um agradecimento muito especial ao HB pelo apoio e amor incondicionados, pela compreensão e motivação nos momentos de menos força e pela felicidade que me proporciona todos os dias.

acknowledgements

First and foremost, I would like to express my endless gratitude to Professor António Neves, my main advisor, for his guidance, sharing of knowledge and continuous support. Thank you for all your patience, contagious optimism and for all the inspiring discussions that led to the conclusion of this Thesis. I am thankful to Professor Bernardo Cunha, my co-adviser, for his enthusiasm and for always reviewing my work. I am also grateful to Professor José Luís Azevedo for his help and his kind words at the end of each robotics competition. I would like to thank the University of Aveiro, mainly IEETA and DETI, for providing me the conditions to pursue this Ph.D. and for all the financial support. This Thesis could not have been possible without the cooperation of my colleagues from IRIS Laboratory, mainly the members of the CAMBADA project. The time at the University of Aveiro would not have been the same without the friendship of Aneesh and Bruno. To my friends Diana, Cristina, Paul, Annamaria and Marisa, a huge thank you for always having my back and for having you close, even though we were many kilometres away. Tozé, thank you for sharing my passion for a healthier living and for becoming my training buddy. To my mother, a huge thank you for allowing me to follow my dreams. Last and most importantly, I would like to thank HB for his support and unconditional love, for his understanding and motivation through my weakest moments and for the daily dose of happiness.

Palavras-chave

Constrangimento de tempo de processamento, robôs moveis e autónomos, sistema de visão, deteção por cor, deteção de objetos

Resumo

A deteção de objetos em imagem digital com baixo custo computacional e tempo de processamento é uma necessidade de várias áreas de investigação, como por exemplo: robótica, vigilância por vídeo ou análise de trânsito. Com os mais recentes avanços da tecnologia e o decréscimo nos preços dos sensores e das câmaras digitais, hoje em dia as câmaras podem adquirir imagens com resoluções e taxas de acquisição bastante elevadas. Isso significa que uma grande quantidade de informação tem que ser processada num intervalo de tempo muito curto. Este trabalho de doutoramento está focado no problema da deteção de objetos coloridos, com restrições temporais, e usa como plataforma de teste robôs inteligentes que jogam futebol de uma forma autónoma. O trabalho está integrado no laboratório de Robôs e Sistemas Inteligentes do Instituto de Engenharia Electrónica e Informática da Universidade de Aveiro. A equipa de futebol robótico CAMBADA foi utilizada como plataforma de teste de todo o trabalho desenvolvido neste doutoramento. O objetivo era apresentar uma solução modular para um sistema de visão que pudesse ser usado pelos robôs da equipa CAMBADA para deteção de objetos coloridos, com restrições temporais. Foram desenvolvidos algoritmos para a calibração colormetrica da câmara, optimização da pesquisa numa imagem e formação de manchas de cor e com base neles construiram-se sistemas de visão que pudessem ser usados pelos mesmos rôbos. Estes sistemas de visão foram usados em ambientes competitivos de futebol robótico e os resultados obtidos provam a sua eficiência. Além disso, todos os algoritmos e software desenvolvidos foram agregados numa biblioteca de visão por computador, de acesso livre e sem custos, que fora desenvolvida para a deteção com restrições temporais de objetos coloridos. Foram desenvolvidas também várias ferramentas para a monitorização remota e a calibração de um sistema de visão robótico.

Keywords Time-constrained processing, autonomous mobile robots, vision system, color detection, object detection

Abstract

Object detection in digital images with small computational costs and processing time is a necessity of some of the most diverse domains, such as: robotics, video surveillance or traffic analysis, among others. With current advances in technology and the decrease of prices in image sensors and video cameras, nowadays digital cameras can acquire images at high resolutions and frame rates. This means that a great amount of visual information has to be processed in a small amount of time. This Ph.D. addresses the problem of time-constrained detection of colored objects and uses as a test bed intelligent robots that play soccer in an autonomous manner. This work is integrated in the Intelligent Robotics and Systems Laboratory from the Institute of Electronics and Informatics Engineering of Aveiro of University of Aveiro. The autonomous robotic soccer team CAMBADA was the test bed of all the work developed during this Ph.D. The objective was to present a modular solution of a vision system that could be used for time-constrained color detection in the aforementioned robotic project. Algorithms for colormetric camera calibration, optimal image scanning and colored blob formation have been designed and based on them functional robotic vision systems have been developed. These vision systems have been used in real adversarial soccer scenarios and the results obtained prove their efficiency. Moreover, all the developed algorithms and software have been aggregated within a computer vision library, which is both free and open source and designed for time-constrained colored object detection. In addition, several support tools have been implemented as well, as tools for remote monitoring or color calibration of a robotic vision system.

Contents

List of Figures

List of Tables

Acronyms

MSL	Middle Size League
SPL	Standard Platform League
HL	Humanoid League
SSL	Small Size League
SL	Simulation League
FIFA	Fédération Internationale de Football Association
RLE	Run Length Encoding
LUT	Look-Up Table
GUI	Graphical User Interface
RTDB	Real-Time Database
MSV	Mean Sample Value

"There is no innovation
and creativity without failure."
Brené Brown

1 Introduction

For more than half of century, the scientific community of researchers working in computer vision has focused on enabling machines with an artificial sense of vision, at least as good as the human one. Two of the key characteristics of human visual processing that are most difficult to reproduce in an artificial vision system are the discerning of relevant information in a given situation or environment and the rapid processing of this information, like "in the blink of an eye". The amount of information that the human processing unit, the brain, receives from the eyes must be at least two orders of magnitude greater than all the information it obtains from the other senses [7]. The human brain can process all the visual information provided by the eyes in a short amount of time since it possesses 10^{10} neurons, out of which, some have over 10000 synapses with other neurons [7]. Looking at each neuron as a microprocessor and considering that these microprocessors are able to work in parallel, the human CPU cannot even be compared to any computer that has been invented so far.

We live in what could be called nowadays the "speed era", in which things around us happen and expand at a fast pace. Either it's technological innovations, medical procedures or social interaction, everything around us seems to evolve at an elevated rhythm and our actions and evolution have to follow this fast trend. Just as we humans need to react fast, prioritize what is relevant for us and process that information in real-time, this Ph.D. Thesis intends to endorse autonomous intelligent robots with similar capabilities. The solution that we propose comes in the form of time-constrained colored object detection algorithms, as well as a modular solution for robotic vision systems to acquire and process visual information in the smallest possible amount of time.

The word “robot” comes from the Czech word “robota”, meaning *forced labor* or *servitude.* The Merriam–Webster Dictionary[1] definition of a robot is:

- A real or imaginary machine that is controlled by a computer and is often made to look like a human or animal.
- A machine that can do the work of a person and that works automatically or is controlled by a computer.
- A machine that looks like a human being and performs various complex acts (as walking or talking) of a human being; also, a similar but fictional machine whose lack of capacity for human emotions is often emphasized.

The Robot Institute of America[2] defines a robot as a re-programmable, multi-functional manipulator designed to move material, parts, tools, or specialized devices through various programmed functions for the performance of a variety of tasks.

By some of these definitions, many of the nowadays electronic household objects, such as a vacuum cleaner or a washing machine, can be seen as robots. However, such machines do not possess intelligence, probably the most important feature of today’s most advanced robots. Although there is not a standardized definition of the word “robot”, today’s robots are more than simple manipulators and they share most of the following capabilities: sensing, artificial intelligence, movement and energetic autonomy.

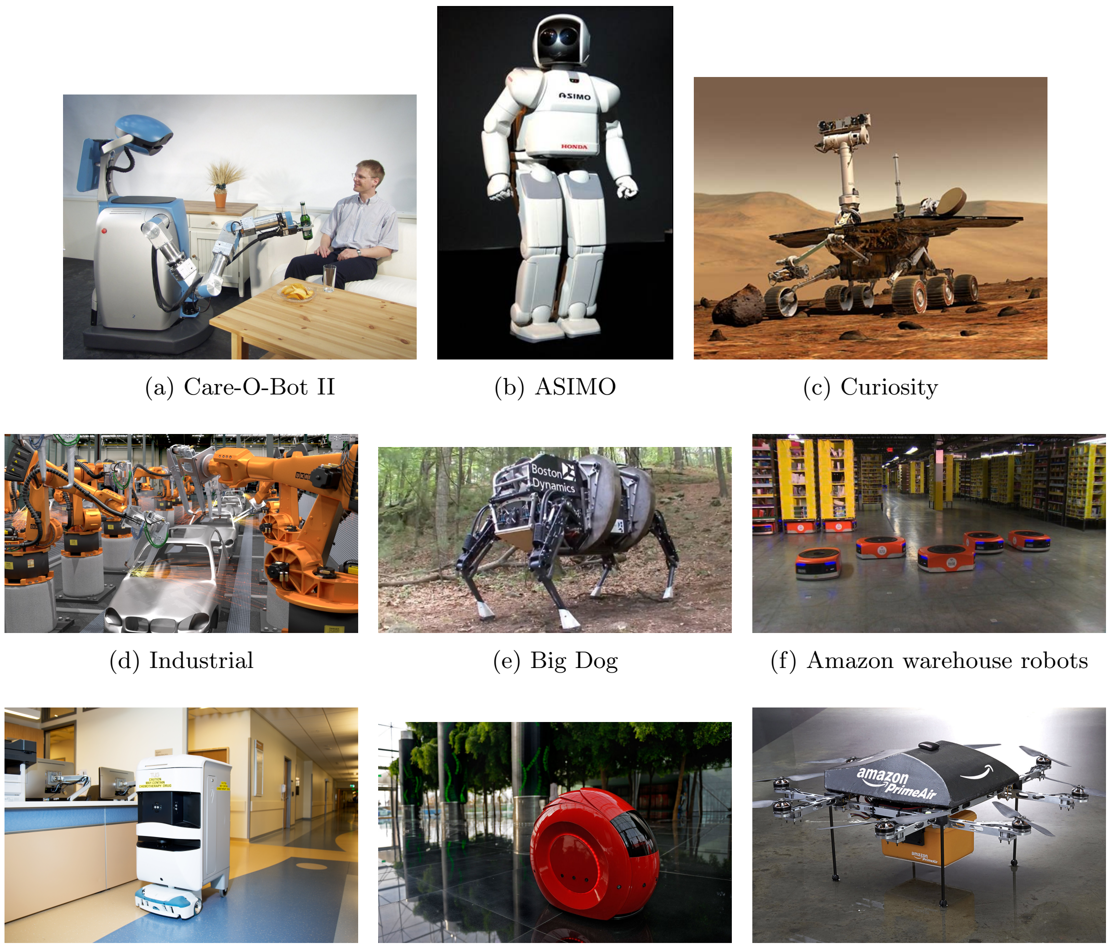
Figure 1.1 shows some of today’s most advanced robots. Nowadays autonomous robots are an ubiquitous part of industry, medicine, space exploration or rescue operations.

Among the capabilities of an advanced autonomous robot, “vision is recognized as the most powerful of robot sensory capabilities”[8]. The goal of machine or computer vision has been, for years, to endue machines with a sense of vision at least as strong as the human vision. The human vision system is capable of remarkable things but it is not solely based on our eyes, our brain is just as or even more important. There are many powerful artificial vision sensors available nowadays which allow machines to acquire images similarly or, in some cases, depending on the type of sensor, even with greater detail than the human eyes. The big issue that computer vision is still tackling with is actually providing to a robot, a machine or a computer the capabilities of image processing and understanding that the human brain has.

1.1 Motivation

Robots are an ubiquitous part of our daily life and we want them smarter and faster each day. We expect robots to be able to perform most of the actions or tasks of a person, but with

[1]`http://www.merriam-webster.com/dictionary`
[2]`www.robotics.org`



(a) Care-O-Bot II (b) ASIMO (c) Curiosity

(d) Industrial (e) Big Dog (f) Amazon warehouse robots

(g) TUG (h) Santander (i) Amazon delivery drone

Figure 1.1: Several examples of autonomous robots used for some of the most diverse tasks.

more precision, in less time and in a safe manner. In many applications the work of a robot already outperforms the work of a person, especially when tasks are repetitive, monotonous or dangerous. But in just as many applications, there is yet no comparison between the capabilities of a robot and the ones of a human.

This Ph.D. Thesis is an effort towards lowering the number of applications in which robots have limitations. It focuses on the study and development of innovative algorithms for colored object detection, that allow the development of modular vision systems to be used on intelligent robots. These vision systems meet all the challenges of time-constrained performance and they can be used on a wide range of autonomous robots, independently of their hardware architecture.

In many applications in the areas of Robotics and Automation the environment is still controlled up to a certain extent in order to allow progressive advancements in the different development directions that stand behind these applications. There are many industrial applications in which semi or fully autonomous robots perform repetitive tasks in controlled environments, where the meaningful universe for such a robot is reduced. For example, in some applications the universe of the robot is limited to a set of relevant objects and locations that are known a priori. In other applications, the illumination of a scene is controlled, meaning that it can be constant or that only artificial light is being used. In applications such as industrial inspection, traffic sign detection or robotic soccer, among others, the environment is either reduced to a set of objects of interest that are color-coded (or color-labeled) or the color segmentation of the objects of interest is the first step of the object detection procedure. This is mainly due to the fact that segmenting a region based on colors is less heavy from the point of view of the computational resources involved than the detection of objects based on generic features. To our knowledge, at the beginning of this Ph.D., there was no free, open source available library that could provide a complete set of algorithms dedicated to time-constrained object detection. This Ph.D. Thesis intends to be a contribution in that matter, providing modular and configurable time-constrained algorithms for colored object detection, an open-source computer vision library and several robotic vision systems fully usable.

1.1.1 CAMBADA Robotic Soccer Team

Several vision systems have been developed throughout this Ph.D.: a perspective (with two different aims: detection of flying objects and ground truth monitoring), an omni-directional and an embedded one. These systems have been fully integrated within the software of the robotic soccer team of University of Aveiro. The CAMBADA robotic soccer team (acronym of Cooperative Autonomous Mobile roBots with Advanced Distributed Architecture)[3] is the RoboCup[4] Middle Size League (MSL) soccer team of the University of Aveiro (IEETA - Institute of Electronics and Telematics Engineering of Aveiro and DETI - Department of Electronics Telecommunications and Informatics), Portugal (Fig. 1.2). The team was founded in 2003 and since then it has attended each year the Portuguese, European and Worldwide RoboCup competitions.

RoboCup is an international federation focused on promoting research in the area of robotics, through different types of competitions that bring together researchers from all over the world, working on the most diverse areas of robotics. The Soccer League was the first RoboCup League to be created and it uses the game of soccer to captivate both researchers and general public with interest in multi-agent systems. More detail on RoboCup and the particularities of the MSL soccer games are given in Section 2.1 and Section 2.2. An illustration of a robotic soccer game in MSL is shown in Fig. 1.3.

The CAMBADA robots are completely autonomous, able to perform holonomic motion and are equipped with an omni-directional vision system that allows them to have a 360° view of the environment in which they are found [9] (Fig. 1.4). In addition, the goalie is

[3]`http://robotica.ua.pt/CAMBADA/`

[4]`www.robocup.org`

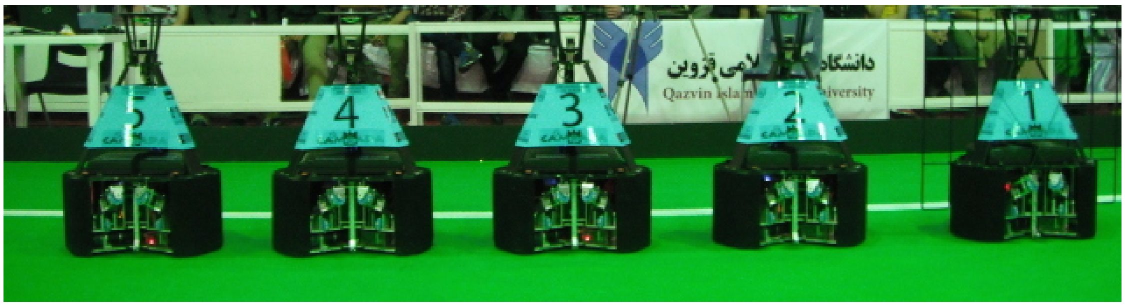


Figure 1.2: CAMBADA robotic soccer team.

Figure 1.3: A game situation in MSL.

also equipped with a perspective vision system used for the detection of balls that were sent through air. In order to play soccer, a robot has to detect, in useful time, the ball, the limits of the field and the field lines, the goal posts and the other robots that are on the field. These robots can move with a speed of 4m/s and the ball can be kicked with a velocity of more than 10m/s, which leads to the need of having fast object detection algorithms.

1.2 Research Goals and Achievements

This Thesis is intended as a contribution to the software architecture of the CAMBADA robots. The omni-directional and perspective vision systems that have been built during this Ph.D. have reached the maximum technological readiness level [10]. These systems have been tested and used in several robotic competitions in the last years, namely:

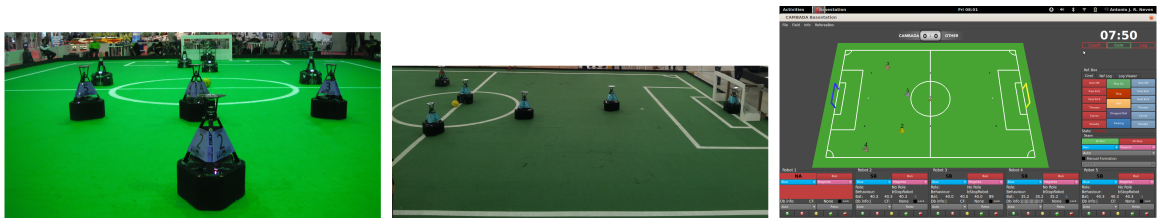


Figure 1.4: On the left and center images, an example of a robot setup during a soccer game. On the right, the world as the robot understands it, displayed on the team base-station.

- **Robotica 2013** - 13th Edition of the Portuguese Robotics Open, April 2013, Lisbon, Portugal - 2^{nd} place.
- **RoboCup 2013** - 17th RoboCup World Championship, June 2013, Eindhoven, Netherlands - 3^{rd} place.
- **RoboCup IranOpen 2014** - April 2014, Tehran, Iran - 2^{nd} place.
- **Robotica 2014** - 14th Edition of the Portuguese Robotics Open, May 2014, Espinho, Portugal - 2^{nd} place.
- **RoboCup 2014** - 18th RoboCup World Championship, July 2014, João Pessoa, Brazil - 3^{rd} place.
- **Robotica 2015** - 15th Edition of the Portuguese Robotics Open, April 2015, Vila Real, Portugal - 2^{nd} place.

In addition, the computer vision library that resulted from this work is functional, open source and freely available.

The main objective of this Thesis was the development of fast computer vision algorithms for object detection, that can be used in time-constrained robotic applications. The objectives of this work have been the following:

- Study the environment of robotic soccer from a scientific point of view and identify the challenges of this particular robotic application.
- Review published literature on computer vision algorithms and methodologies that can be used in time-constrained robotics.
- Develop time-constrained color object detection algorithms for the detection of the objects of interest in a robotic soccer game.
- Implement a fully functional and modular robotic vision system that would allow the use of the robotic soccer team in real, competitive scenarios.

- Develop support tools for the calibration of the vision system.
- Integrate all algorithms and methodologies in an open-source computer vision library for time-constrained applications, that can be shared with the scientific community and that could be a valuable contribution to this field of study.
- Deployment of the developed software and guarantee its full integration and execution in official competitions, within the existent restrictions.

1.3 Scientific Contributions

The work developed during this Ph.D. covers the field of time-constrained computer vision algorithms. The main contributions of this Thesis are:

- A modular computer vision library for time-constrained color object detection. The modular design of this library allows the independent use of the different modules that make it up in applications that use a digital camera as a sensorial element.
- A collection of time-constrained algorithms for the calibration of the colormetric parameters of a digital camera and for the detection of colored objects, either on the ground or aerial.
- Fully functional vision systems that can attend to all the challenges of autonomous robotic soccer, such as: real-time object detection, low computational resources, detection of flying object and ground truth generation.
- The design of image scanning patterns that are highly configurable to the processing time and accuracy demands of an artificial autonomous vision system.
- A novel study on the use of raw data for color object detection and the effects on terms of detection accuracy.
- A study on the delay between perception and action in an autonomous soccer robot, using both raw and RGB data.
- A graphical user interface (GUI) tool for the study of the most common color spaces and their use for manual calibration by a human user.
- A tool for remote communication with an autonomous robot and exchange of visual data. This tool supports the exchange of information and the calibration between a robot performing a given task and a remote client, without this exchange interfering with the completion of its task.

- As a result of the work developed during the 4 years of this Ph.D., several scientific manuscripts have been published in national and international peer-reviewed conferences.

1.3.1 Software Releases

An open source, modular, time-constrained computer vision library for color object detection has been developed. The library is available at `http://sweet.ua.pt/an/uavision/`, along with several test sequences and instructions on the use of the library.

Based on this library, a generic time-constrained vision system pipeline for autonomous soccer robots has been defined and several vision systems have been implemented. These vision systems have a technological readiness level of 9 and they have proven their efficiency in several official robotic competitions, as well as in laboratory tests. Moreover, an embedded vision system has been built and it is designed for the use on robotic platforms with very limited computing capabilities.

Along with these vision systems, several external calibration and monitoring tools have been built and are currently in use by the robotic soccer team.

1.3.2 Scientific Publications

The following scientific publications have resulted from the work presented in this Thesis:

▷ International peer-reviewed journals:

- **Alina Trifan**, António J. R. Neves, Bernardo Cunha, José Luís Azevedo: *Real-Time Color Coded Object Detection Using a Modular Computer Vision Library*, Advances in Computer Science : an International Journal, Volume 5, Issue 1, January 2016 (accepted).

▷ National peer-reviewed conferences/proceedings:

- **Alina Trifan**, António J. R. Neves, Bernardo Cunha, Nuno Lau: *A color-coded vision system for a NAO robot*, Proceedings of the 17th Portuguese Conference on Pattern Recognition, RecPad 2011, Porto, Portugal, October 2011.

- **Alina Trifan**, António J. R. Neves, Bernardo Cunha, Nuno Lau: *The importance of color spaces in robotic vision*, Proceedings of the 18th Portuguese Conference on Pattern Recognition, RecPad 2012, Coimbra, Portugal, October 2012.

▷ International peer-reviewed conferences/proceedings:

- **Alina Trifan**, António J. R. Neves, Bernardo Cunha, Nuno Lau: *A modular time-constrained vision system for humanoid robots*, Proceedings of IS&T Electronic Imaging 2012, San Francisco, USA, January 2012.

- **Alina Trifan**, António J. R. Neves, Bernardo Cunha: *Evaluation of color spaces for user-supervised color classification in robotic vision*, Proceedings of the 17th International Conference on Image Processing, Computer Vision, and Pattern Recognition, ICPV13, Las Vegas, Nevada, USA, July, 2013.

- **Alina Trifan**, António J. R. Neves, Bernardo Cunha : *An overview on the application of machine vision in soccer robots*, Proceedings of the 17th International Conference on Image Processing, Computer Vision, and Pattern Recognition, ICPV13, Las Vegas,Nevada, USA, July, 2013.

- António J. R. Neves, **Alina Trifan**, Bernardo Cunha : *Self-calibration of colormetric parameters in vision systems for autonomous soccer robots*, Proceedings of the 17th RoboCup International Symposium, Eindhoven, Netherlands, July, 2013.

- **Alina Trifan**, António J. R. Neves, Bernardo Cunha: *UAVision: A modular time-constrained vision library for color-coded object detection*, Proceedings of CompImage2014, Pittsburgh, USA, September, 2014.

- **Alina Trifan**, António J. R. Neves, Bernardo Cunha: UAVision: *A modular time-constrained vision library for soccer robots*, Proceedings of the 18th RoboCup International Symposium, João Pessoa, Brazil, July, 2014.

- António J. R. Neves, **Alina Trifan**, Paulo Dias, José Luís Azevedo. *Detection of aerial balls in robotic soccer using a mixture of color and depth information*, Proceedings of the 2015 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC 2015), Vila Real, Portugal, April 2015.

- António J. R. Neves, **Alina Trifan**, José Luís Azevedo. *Time-constrained detection of colored objects on raw Bayer data*, Proceedings of V ECCOMAS Thematic Conference on Computational Vision and Image Processing (VIPIMAGE 2015), Tenerife, Canary Islands, Spain, October 2015.

- **Alina Trifan**, António J. R. Neves, Bernardo Cunha. *Image scanning techniques for speeded-up color object detection*, Proceedings of V ECCOMAS Thematic Conference on Computational Vision and Image Processing (VIPIMAGE 2015), Tenerife, Canary Islands, Spain, October 2015.

- **Alina Trifan**, António J. R. Neves. *On the use of feature descriptors on raw data*, Proceedings of 5th International Conference on Pattern Recognition Applications and Methods (ICPRAM 2016), Rome, Italy, February 2016 (accepted).

- António J. R. Neves, Fred Gomes, Paulo Dias, **Alina Trifan**. *A ground truth vision system for robotic soccer*, Proceedings of 5th International Conference on Pattern Recognition Applications and Methods (ICPRAM 2016), Rome, Italy, February 2016 (accepted).

- **Alina Trifan**, António J. R. Neves. *A Survey on Lossless Compression of Bayer Color Filter Array Images*, Proceedings of ICIPACV 2016 : 18*th* International Conference on Image Processing, Analysis and Computer Vision, Los Angeles, USA, April 2016 (accepted).

- António J. R. Neves, Rui Garcia, Paulo Dias, **Alina Trifan**. *Object Detection Based on Plane Segmentation and Features Matching for a Service Robot*, Proceedings of ICIPACV 2016 : 18*th* International Conference on Image Processing, Analysis and Computer Vision, Los Angeles, USA, April 2016 (accepted).

▷ Technical reports:

- *CAMBADA2013: Team Description Paper.* R. Dias, A. J. R. Neves, J. L. Azevedo, B. Cunha, J. Cunha, P. Dias, A. Domingos, L. Ferreira, P. Fonseca, N. Lau, E. Pedrosa, A. Pereira, R. Serra, J. Silva, P. Soares and **A. Trifan**, Eindhoven, Netherlands, June 2013.

- *CAMBADA2014: Team Description Paper.* R. Dias, F. Amaral, J. L. Azevedo, R. Castro, B. Cunha, J. Cunha, P. Dias, N. Lau, C. Magalhaes, A. J. R. Neves, A. Nunes, E. Pedrosa, A. Pereira, J. Santos, J. Silva and **A. Trifan**, João Pessoa, Brazil, July 2014.

- *CAMBADA2015: Team Description Paper.* B. Cunha, A. J. R. Neves, P. Dias, J. L. Azevedo, N. Lau, R. Dias, F. Amaral, E. Pedrosa, A. Pereira, J. Silva, J. Cunha and **A. Trifan**, Hefei, China, July 2015.

- *SPL Portuguese Team: Team Description Paper for RoboCup 2011.* A. J. R. Neves, N. Lau, L. P. Reis, A. P. Moreira, **A. Trifan**, B. Pimentel, C. Sobrinho and E. Domingues, Istanbul, Turkey, July 2011.

- *SPL Portuguese Team: Team Description Paper for RoboCup 2012.* A. J. R. Neves, N. Lau, L. P. Reis, A. P. Moreira, **A. Trifan**, B. Pimentel, C. Sobrinho, E. Domingues, N. Shafii, S. Miranda and L. Cruz, Mexico City, Mexico, June 2012.

- *Portuguese Team: Team Description Paper for RoboCup 2013.* A. J. R. Neves, N. Lau, L. P. Reis, A. P. Moreira, J. Silva, **A. Trifan**, N. Shafii, V. Santos, S. Zadegan and S. Bahmankhah, Eindhoven, Netherlands, June 2013.

1.4 Thesis Structure

This Thesis is structured in 8 chapters, first of them being this Introduction.

Chapter 2 enframes the work presented in this Thesis within current similar research efforts done in the field of robotic vision. It focuses mainly on robotic vision systems used in soccer competitions.

Chapter 3 particularizes the algorithm that has been developed for the calibration of the colormetric parameters of a digital camera. We also address the intrinsic and extrinsic calibration of a digital camera and the integration of these algorithms within the library produced.

Chapter 4 focuses on the time-constrained object detection algorithms that have been developed. We present the particularities of these algorithms that allow their use in time-constrained applications and we motivate their use on other computer vision practical applications than the one at which this Thesis was aimed.

Chapter 5 presents the robotic vision systems that have been implemented using the library built.

In Chapter 6 we evaluate and discuss the performance of these vision systems in real, competitive robotic soccer scenarios.

Chapter 7 presents different tools that have been developed in order to support the robotic vision systems that have been implemented.

Chapter 8 concludes this document and contains final remarks and acknowledgements of the institutions that supported this Thesis.

"The eyes are useless when the mind is blind."

Unknown

2 Vision Systems in Autonomous Mobile Robots

Just like humans, robots can "sense" the surrounding world by means of different sensors, but usually, a digital camera is their main sensorial element. A digital camera is capable to provide a great amount of spatial, temporal and morphological information.

The vision system of a robot has to perform different tasks, depending on the purpose of that robot. The majority of the systems must perform object detection or recognition and localization. Also in a vast range of applications, an efficient robotic vision system should be able to perform its tasks under a certain time limit, which means that several constraints are imposed. One of the most important constraints is, perhaps, that algorithmic complexity is limited when it comes to time constrained applications. Thus, the limit of the processing time of such an application is usually very low, of the order of milliseconds.

In robotic soccer, which has become a popular research branch of robotics, the environment is always changing, the ball and the robots are always moving, most of the time in an unpredictable way. The vision system of a soccer robot 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 reactions. Because in a robotic soccer game the events unfold very fast, algorithms for tracking the objects of interest are not robust enough to keep up with the sudden and repetitive changes in the environment. This makes soccer an excellent test-bed for time-constrained vision systems of intelligent robots.

In this Chapter we will present a brief overview of the robotic soccer environment and we will discuss state-of-the-art scientific contributions that have been presented so far on the subject of time-constrained object detection algorithms for soccer robots.

2.1 The RoboCup Initiative

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. 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.

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, complying with the official rule of the FIFA, against the winner of the most recent World Cup."

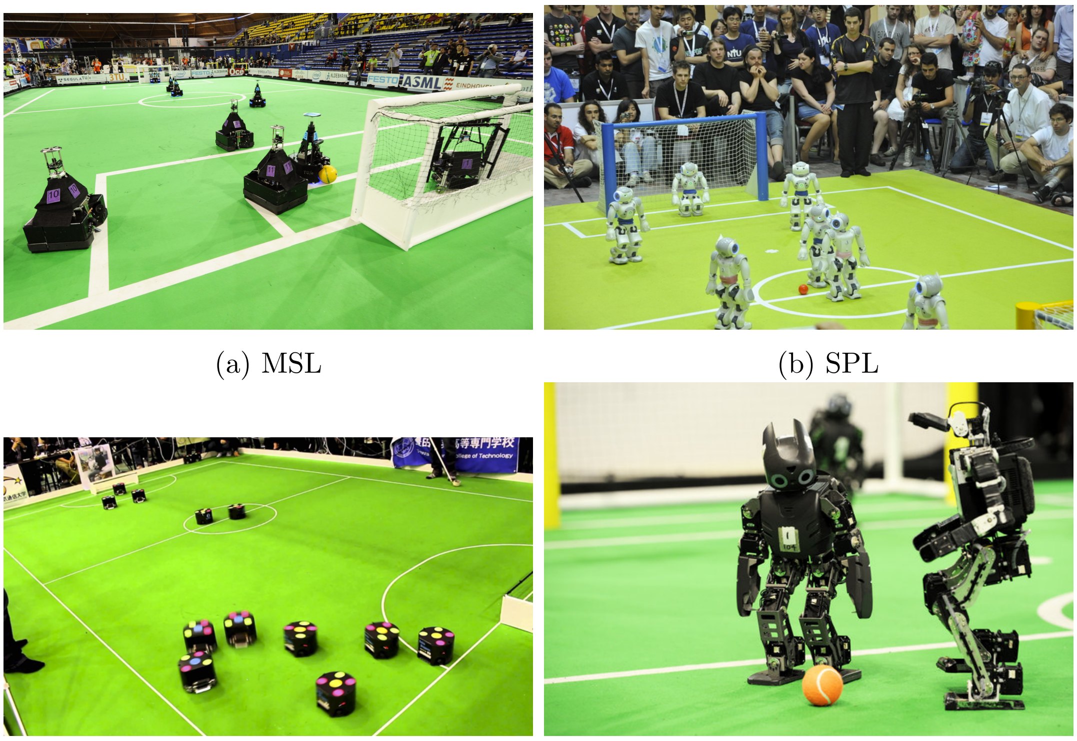
Such a goal might sound still overly ambitious given the state of the art of affordable technology and research. Building humanoid soccer players requires an equally long period of time as well as extensive efforts on a broad range of research areas. Most probably the goal will not be met in any near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

The RoboCup Robot Soccer competition is divided into the following leagues: Middle Size League, Standard Platform League (SPL), Humanoids League (HL), Small Size League (SSL) and Simulation League (SL). Each league portraits soccer games among autonomous robots, but the hardware platforms in each of the leagues differ (Fig. 2.1).

Although different in many aspects and each of them imposing different research challenges, all of these four soccer leagues share certain common features. In all leagues, robots play soccer on a green field, with white lines, according to FIFA rules that suffered small adjustments in order to be applicable to robots. Moreover, in almost all leagues, except for MSL, the robots play with an orange ball and protect white goals. Finally, all robots wear cyan or magenta body markers or t-shirts in order to distinguish between the two teams that are competing in a game.

In MSL the games are played between teams of up to five wheeled robots that each participating team is free to build, keeping in mind only the maximum dimensions and weight of the robots. In SPL all teams compete using the same robotic platform, a humanoid NAO built by Aldebaran [11] and the games take place between teams of up to five NAOs. In HL the games are played between humanoid robots, that is robots that imitate the human body, and the number of players in a game depends on the sub-league in which a team is participating: Kid-Size League, Teen League or Adult League. In SSL teams of 6 small robots identified by colored patches compete in the game of soccer. These robots must fit within an $180mm$ diameter circle and must be no higher than $15cm$. The SL competitions are the simulated version of the SPL ones.

All the physical RoboCup soccer games occur in closed, controlled environments. The robots play soccer indoor, on a green with white lines carpet and the color of the ball is


(a) MSL (b) SPL

(c) SSL (d) HL

Figure 2.1: Game situations on the four physical soccer leagues of RoboCup.

known in advance. The vision system of these robots rely on a vision sensor, responsible for the acquisition of images and a processing unit, responsible for processing the images in real-time. In most cases, the processing unit has small capabilities compared to nowadays most advanced computers, and this is due both to size and energy limitations. SSL - Vision [12] is the standardized vision system used by all competing teams in the SSL League. It that processes the data provided by two cameras that are attached to a camera bar located $4m$ above the playing surface. In this league, off-field computers are used to communicate referee commands and position information to the robots. These computers perform all of the processing required for coordination and control of the robots.

2.2 The Middle Size League

A great percentage of the work presented in this Thesis has been tested and applied to robotic soccer games taking place in the RoboCup MSL. Because of this, this Section provides an overview of this league.

In the context of RoboCup, the MSL is one of the most challenging and more evolved. In this league, each team is composed of up to 5 robots with a maximum size of $50cm \times 50cm$ width, $80cm$ height and a maximum weight of $40kg$, playing on a field of $18m \times 12m$ (Fig. 2.2).

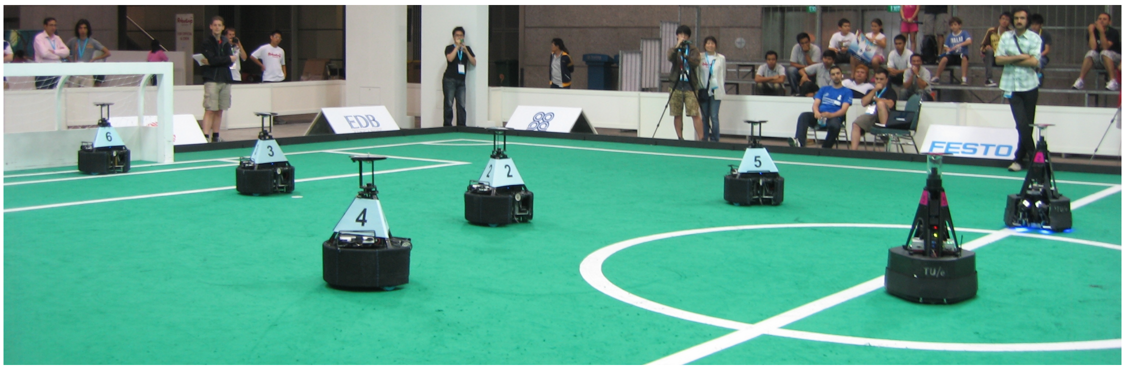


Figure 2.2: An image of a RoboCup MSL soccer game.

The RoboCup MSL competition is a standard real-world test for autonomous multi-robot systems [13]. In this league, omni-directional vision systems have been explored in the last years, allowing a robot to see in all directions at the same time without moving itself or its camera. The games of this league resemble the most the ones played among humans. This is mainly due to the fact that the environment is not as restricted as in the other leagues and the robots have reached a high level of autonomy, thus turning the games more dynamic. The color restriction of the ball has been raised. Nowadays, MSL robots play soccer with an arbitrary ball, whose color is only decided before a tournament. The only rule applied is that the surface of the ball should be 80% of a certain color.

These robots can move at speeds higher than $4m/s$ and are able to kick the ball at more than $10m/s$. The speed of the robots and the power of their kick has a great influence on the performance of a vision system that such a robot should have. The vision system has to keep up with the speed of the robot, to compensate the noise in an acquired image produced by the movements of these robots and to process the environment with high precision and at high frame rates. A high frame rate is directly related to the processing time of the vision system, given that the higher the number of frames acquired per second, the less available time for processing them. In addition, the fact that the ball is often kicked through air limits the possibilities of using object tracking algorithms.

2.3 Vision Systems in Autonomous Mobile Robots

The library that resulted from this Ph.D. work is an important contribution for the Computer Vision community since, so far, there are no free, open source machine vision libraries for this application. This library can be employed for the development of both omni-directional and perspective time-constrained vision systems. The omni-directional vision systems, which are nowadays the preferred choice in RoboCup MSL provide a 360° visualization of the surroundings. The processing of the images acquired by an omni-directional vision sensor is more complex, mainly due to the distortion of the image and higher degradation in the resolution with growing distances away from the robot [14].

CMVision [15], a machine vision library developed at the Carnegie Mellon University was one of the first approaches to build such a library. However, it remained quite incomplete and has been discontinued in 2004. Several other machine vision libraries, such as Adaptive Vision [16], CCV [17] or RoboRealm [18], provide machine vision software to be used in industrial and robotic applications but they come at a cost and they are not open source. Our library, called UAVision, aims at being a free, open-source library that can be used for robotic vision applications that have to deal with time constraints.

On the other hand, several solutions for the design of vision systems for autonomous robots systems have been proposed in the last decade, some of them in the context of the RoboCup initiative. Since the computer vision library that we propose has been mainly used for the implementation of a vision system for robots that play soccer, this Section will focus on recent work that has been done in this particular field. Even though there is no comparable computer vision library yet, the scientific papers that will be reviewed in this Section address, individually, many of the challenges that UAVision tries to solve.

A case study presented in [1] exposes the functional and non-functional requirements of a robotic vision system. The non-functional requirements: modifiability, extensibility and portability dictate the architecture of a vision system. The functional requirements usually relate to the object detection algorithm of such systems. A software architecture that fulfills the non-functional requirements criteria (Fig. 2.3) is presented. The resulting vision system has been used on a Darwin-OP humanoid robot[1] that takes part in the RoboCup HL competitions. The proposed architecture is structured on two layers: interface, control and data storage in the top layer and all the components for feature detection in the bottom layer. The top layer concentrates all external communications into two wrapper modules: a Controller module and a Blackboard module that stores locally all the information required by the low level processing stages, as well as their results. Their approach for feature detection starts with a detection of the horizon based on a combination of kinematics and search for green color, followed by vertical and horizontal scan lines in search of sequential pixels of the same color (the ball color). Results of this system working at $80fps$ are presented, however with a success rate of the ball detection of only 50% when the ball is occluded.

Illumination is an important aspect when dealing with color object detection. In robotic soccer games, most of the times the illumination of the soccer field is a mixture of artificial and natural light. This uneven illumination turns the soccer field brighter in some parts and covered by shadows in others. The vision systems of soccer robots have to overcome this challenge and accurately detect the objects of interest no matter where they are on the field. Different techniques for color object recognition that can adapt to changes in illumination are presented in [2, 3, 19].

A soft labelling of pixel class followed by a hard decision using an adaptive threshold is

[1]`http://www.trossenrobotics.com/p/darwin-OP-Deluxe-humanoid-robot.aspx`

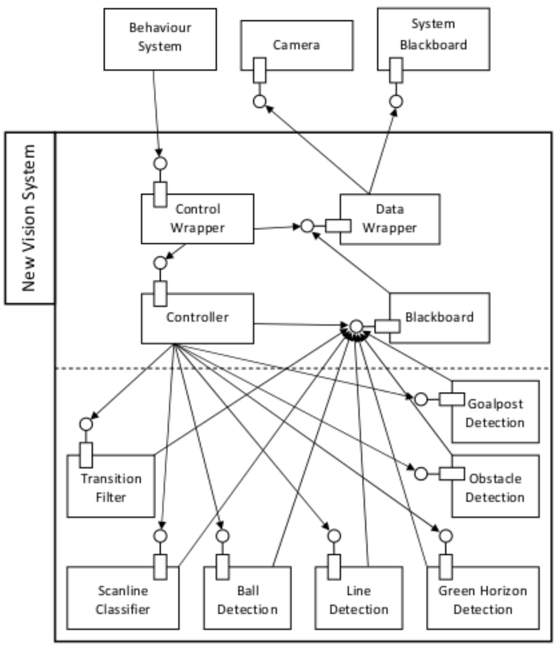


Figure 2.3: Vision system architecture proposed in [1].

detailed in [19]. Connected pixels are conglomerated using a connected component analysis. Objects of interest are detected and recognized by searching for nearby regions that match a-priori models, with soft-comparisons to account for variations in shape, size and missing features. This technique can be considered a pioneer for the computer vision algorithms applied to RoboCup, since it was one of the early ones applied to SSL robots. Results are presented on images of 320×240 pixels acquired at frame-rates of $30fps$, far from what can be achieved with nowadays digital cameras.

Similar image resolutions with processing times rounding $34ms$ are presented in [2]. Their approach is based on an adaptive color calibration method built on top of the Bayes Theorem [20] and chrominance histograms. A randomised Hough Transform [21–23] is used for ball detection and an orthogonal regression is computed for goals and flag posts detection. An initial table which contains the mapping between the chrominance and the classified object (the color map) is used to mark image regions occupied by the objects of interest. After applying the color map, the contours and neighborhood of the object are checked to see if they are indeed objects of interest. On success, histograms are updated and a new color map is calculated. A second map, that specifies the maximum chrominance set is added. A-priori probabilities for each object are calculated based on relative frequencies in the histograms and the a-posteriori probability is given by Bayes Theorem. At each pixel, the object with the highest probability is classified and the object belonging to a chrominance pair is stored in a look-up table. A result of this approach is shown in Fig. 2.4.

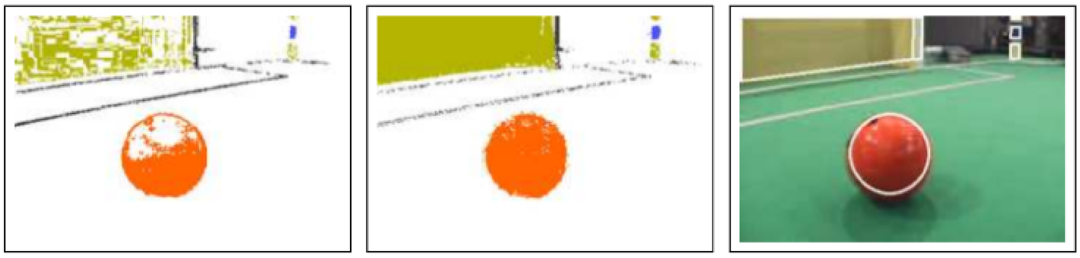


Figure 2.4: On the left, an initial color classification. In the middle, the color classification after calibration. On the right, ball, posts and flag detected. Images taken from [2].

Another solution for the vision system of a Sony Aibo [2] SSL player has been proposed in [3]. The independence of the lighting conditions of their approach can be obtained by focusing on contrast patterns in three different color channels and auto-adapting color classification. The horizon is calculated from the rotation of head and body of the robot and for each of the areas below and above the horizon an optimized grid layout is used for scanning. From the horizon, vertical scan lines extend into the lower half of the image. These lines are evenly spaced and the distance depends on the distance to the ball. Scan lines above the horizon are used for finding color coded flags placed at the corners and at the side of the field. Every pixel at the grid lines is assigned to a color class. Green is the reference color, defined by upper and lower boundaries in the YUV space. Other colors are defined in relation to the green cube. The drawback of this approach is that colors that are close to each other in color space are grouped together. When the lighting conditions change, the knowledge that the relative positions of the colors to each other is maintained, is enough to only readjust the reference cube. Object detection is done based on different sequences of edges and color transitions. This algorithm does not work in bright light or under very little light. Figure 2.5 presents results of their object detection, even with the ball partially occluded.

Other approaches for overcoming the illumination challenge during a soccer game is the use of previously trained color look-up tables. An automatic on-line color calibration of soccer-playing robots based on a geometrical model of the field lines in world coordinates and of the ball in image coordinates is presented in [24]. The localization of the robot on the field is based on edge detection. A region tracker is used, it stops growing a region when a possible field line is encountered. The stopping criterion is based on the assumption that a field line is brighter than neighbor pixels. The calibration time amounts to 1-2 seconds, which is a little bit far from the time requirements of soccer robots vision systems.

A three-step method based on adaptive camera parameters control, image segmentation and color classification is addressed in [4]. Color stability in the YUV color space is improved with a controller of the camera colormetric parameters. Segmentation is performed based

[2]`http://www.sony-aibo.com/`

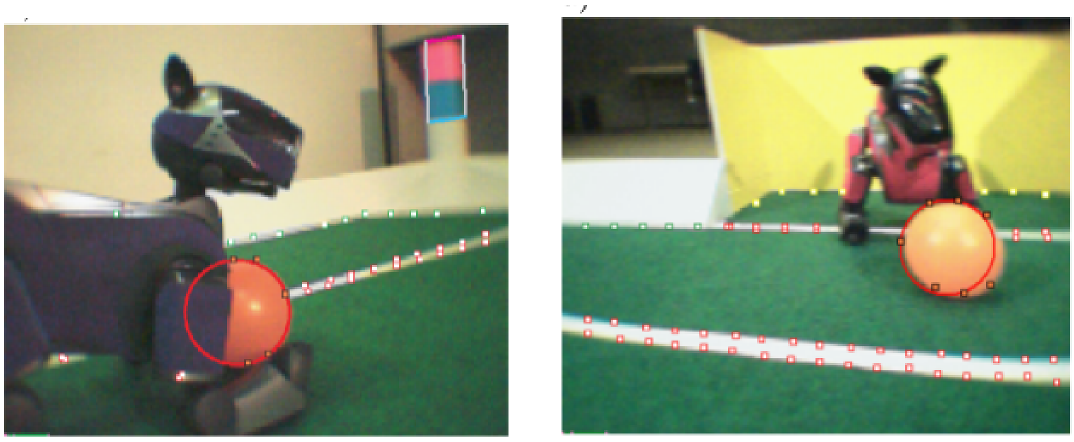


Figure 2.5: Detection of the objects of interest presented in [3].

on Markov Random Fields [25] in order to detect spatially coherent regions of uniform color belonging to objects of interest. The labeling of the colors is done based on a Gaussian color distribution (Fig. 2.6).

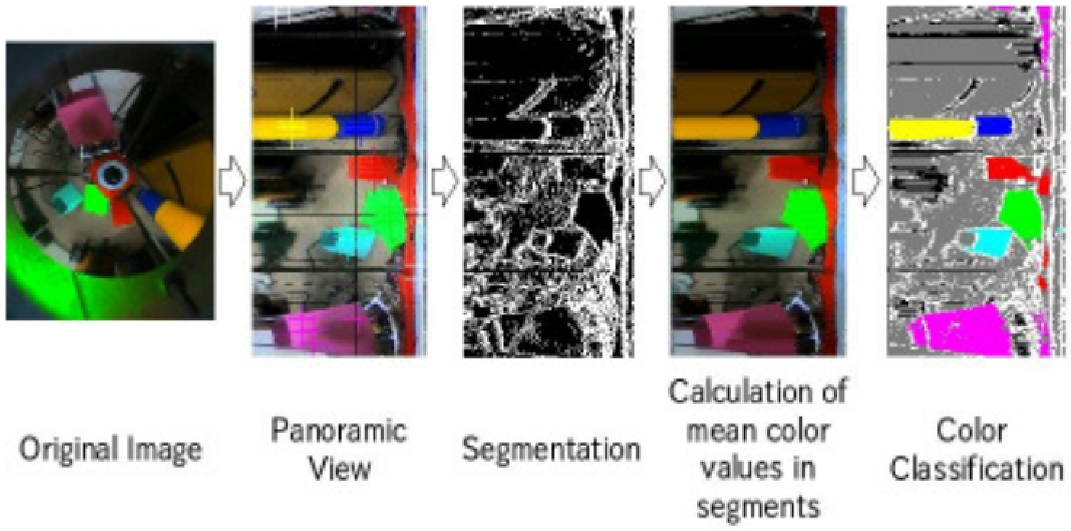


Figure 2.6: Image processing steps presented in [4].

An object recognition method for a MSL omni-directional vision system, that is adaptive to light conditions is presented in [26]. In their paper, the authors prove that the conditional probability density distributions of the YUV values mapping to each color are Gaussian. The means and variances are learnt by manual calibration. Classifying color seeds are chosen based on means and variances. The assumption that colors in an object region should be similar stands at the basis of forming color regions. Means and variances are updated in order to adapt to changing illumination. This algorithm works on images of 500×492 pixels, with an average processing time of $35ms$.

Most vision systems designed for robotic soccer still take advantage of the color labeling of the objects of interest. The color of an object is an important property that permits the leverage of image analysis and processing techniques. Color is the main clue for the detection of an object of interest in a robotic soccer game. Color segmented images for initial object hypotheses and grayscale images for final classification are used in [27]. The color segmented image is used for finding objects on the field and the grayscale image is used to decide whether those objects are Sony Aibo players in the games of RoboCup SSL. The CMVision [15] algorithm is applied to the YUV image based on four main assumptions: every pixel below the horizon line belongs to part of the field or something on the field; objects of interest are not green; objects of interest intersect the field horizon and finally, objects of interest intersect the horizon line. The field horizon is considered to be the limit of green and the highest point of four consecutive green pixels is the horizon. The algorithm for finding the objects of interest starts by scanning along each column from below the field horizon to search for non-green. When enough green is detected, the end of an object is signalized. Enough green can mean only one green pixel, depending on the height at which it has been found. Neighboring columns are grouped together and their lengths are averaged to find the length for that group. The bounding box for each group is the initial hypothesis. For the classification of the initial objects, an integral image is calculated over the grayscale image for object classification. A robot hypothesis is compared to a previous stored robot model. Occluded robots cannot be detected by this approach and the frame rate at which this system works is $30fps$, with a lower bound of $5fps$.

An omni-directional vision system proposed in [28] introduces the notion of jump points when scanning for a color of interest. 1500 jump points are used in this approach, independent of the image resolution. In the worst case scenario, for an unsuccessful search, all jump points have to be tested. However, if an object of interest is present in the image, the authors claim that it will be surely found since the jump points are distributed so that at least 7 jump points overlap with the smallest object of interest, in this case the soccer ball.

More recent efforts concentrate on providing robotic vision systems as independent of a scene as possible. The color of an object is disregarded, with consequences on the maturity and performance of such a vision system. A robust object recognition system based on a wide-baseline matching is proposed in [5]. The wide-baseline matching is performed between a reference image (object model) and a test image, in which the object of interest is searched. Local interest points are extracted independently from both the test and the reference image, then characterized using invariant descriptors. Several matches between similar descriptors from both images are used to get an affine transformation between the two images (Fig. 2.7). The algorithm has been tested in the Humanoid League and the @Home League, on images of 160×120 pixels. This approach only delivers near real-time performance, processing images at as low as $3 - 9$ frames per second.

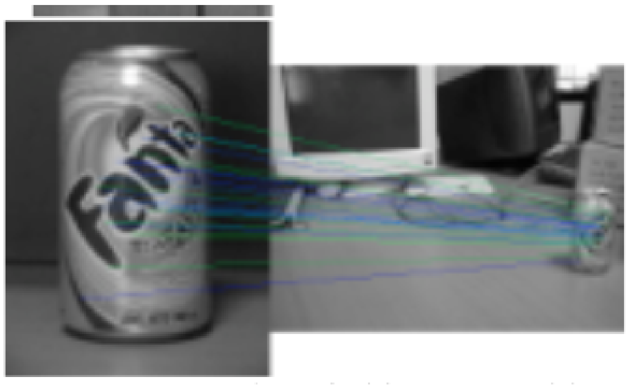


Figure 2.7: An example of an object recognition presented in [5].

A novel algorithm, Wave3D, that maintains 3D hypotheses of the presence of objects in real 3D world relative to a NAO robot from 2D images that the robot acquires is proposed in [29]. Every new frame is processed as a continuation of the processing of previous frames. The system hypothesizes 3D positions where the objects can be found and then validates them in the 2D image. The novelty of this approach resides in the fact that the objects of interest are not treated as collections of pixels, but rather real elements whose position is initially guessed.

An arbitrary ball recognition method based on omni-directional vision for soccer robots is presented in [6] and applied to MSL robots. The method is based on the approximation of the soccer ball by an ellipse and edge points calculated from the transitions between two colors, without specifying these colors. Circular and radial scan lines are used to search for color variations of every two neighboring pixels. The color variations are measured by the Euclidean distance in the YUV color space. If the color variation is higher than a predefined threshold, a possible contour point is found. The distance between each two possible contour points is calculated. If the minor and major axis value of an eclipse with its center located on the middle point of the two possible contour points, correspond to these distances, a possible ellipse center is found (Fig. 2.8). If the two points almost coincide with each other, a candidate ellipse exists and its equation is calculated. The processing time of this approach is around $100ms$, for images of 444×442 pixels.

All these vision systems address different aspects and challenges of an autonomous, time-constrained vision system for robotic soccer. However, none of them presents a complete set of algorithms that can solve all the inherent issues of such a system, from image acquisition to camera and color calibration and finally to real-time object detection. The computer vision library and the robotic vision system pipeline that are proposed in this Thesis represent a whole package that can solve all the aforementioned questions. In addition, the algorithms

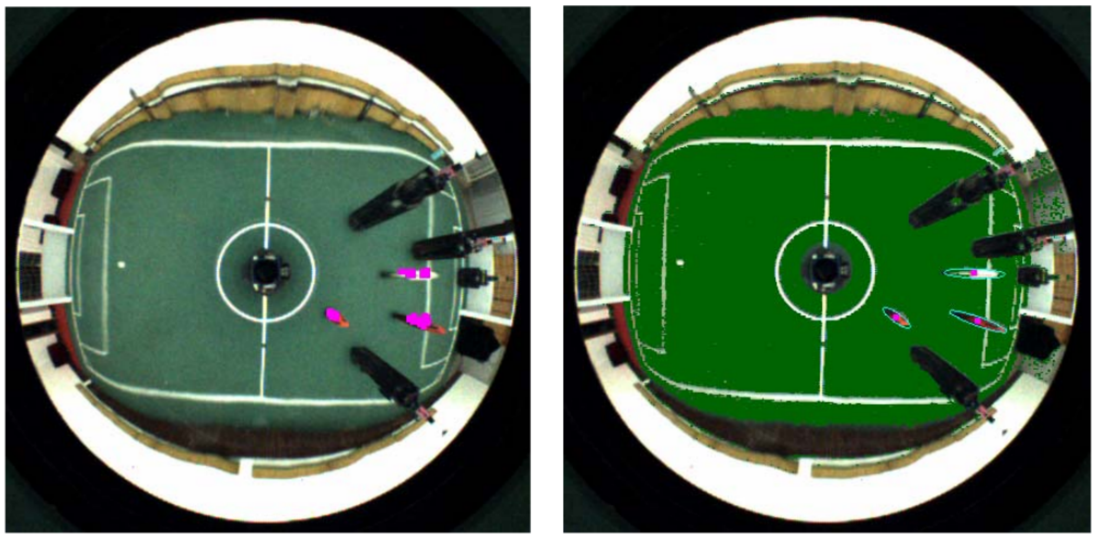


Figure 2.8: On the left image, the centres of the ellipse candidates are marked. On the right, the result of the ball detection. Images taken from [6].

that we have developed have been organized in modules that constitute the library and which are configurable and easily adapted to the available computational power. The pipeline of the vision systems that we introduce can also be applied to other vision systems than the ones we present.

Many aspects of the methods already proposed by the scientific community can be retrieved in our work, however in an optimized and time-constrained oriented manner. The algorithms that we present make use of different scan patterns in order to maximize the scan area of an image and minimize the scan time. The scan patterns that we have designed are robust and easily configurable in order to guarantee a maximum processing time and/or a level of detection accuracy required by a given application. Color clues and trained color look-up tables are also used as a hint for an object of interest and green color information is related to the area of the image that makes sense processing. On the other hand, another important assumption that can be found in literature and retrieved in our work is that the objects of interest can be of any color, except for green.

Our library and vision system integrate a real-time colormetric calibration algorithm that can be used during a soccer game, without delaying the rest of the processing pipeline. This algorithm guarantees the functioning of the system independent of the illumination system. Our approach has even been tested in outdoor environments and to our knowledge, it represents the first scientific published work on this matter. Outdoor soccer games are the next challenge to be overcame by the RoboCup community and the first steps in that direction have been given by introducing Technical Challenges related to this idea.

Another innovation that can be retrieved in our work is the use of raw data for object detection. We have conducted a study that proves that the interpolation of raw data is an unnecessary step in object detection, since the use of raw data for color object detection delivers similar results as the use of interpolated images. Another novel contribution is the study on the delay between the perception and action of an autonomous soccer robot in which we show that the use of raw data is slightly more advantageous.

Our work is also validated by a benchmarking methodology for evaluating robotic soccer vision system proposed in [30]. This methodology states that a public repository with data sets and algorithms that can be dynamically updated should me maintained, along with evaluation metrics, error functions and comparison results. The work that we present is open source, free and publicly available, along with test sequences and published comparative results of our methods.

"Look at how a single candle can both defy and define the darkness."

Anne Frank

3 Vision System Calibration

One of the main challenges of object detection algorithms is to guarantee their good performance independently of the illumination of a scene. Robotic soccer games are currently taking place indoor, on fields that are both naturally and artificially illuminated. This results in shadows on the soccer field that have to be compensated by the vision system. Moreover, one of the nearest future goals of the RoboCup Soccer League is to organize outdoor soccer games. The first steps in this directions have been made with the introduction, in the last couple of years, of technical challenges in which the robots play soccer not on a carpet but on artificial turf.

Overcoming changes in illumination is a challenge for any robot that has to perform a task in an outdoor environment, such as most search and rescue robots or traffic surveillance autonomous systems are, just to name a few. In robotic soccer games, where the color of an object is the first clue towards its detection, color consistency independent of the illumination conditions has to be guaranteed during the performance of a robot. The color consistency in digital images can be related to certain camera parameters, as it will be thoroughly described in this Chapter.

Moreover, the relation between pixel coordinates and world coordinates for an object of interest is another aspect that has to be considered in the performance of any autonomous robot. All these aspects are related to the terms "camera calibration". Camera calibration can be of different types, as we will present next, depending on the camera parameters that have to be adjusted in order to reach a high level of accuracy, either in terms of image acquisition or accuracy of the metrics. In addition, in color coded environments the process of color calibration is common, in which color ranges of a certain color of interest are determined, either statically or dynamically.

This Chapter will focus on the description of the algorithm for calibration of the colormetric parameters of a camera that has been developed and integrated as the Calibration Module of the UAVision library. This algorithm allows the use of the vision system almost independently of the illumination. Moreover, it has been tested outdoor and results are presented.

The algorithms for the intrinsic and extrinsic calibration of the camera parameters will also be described. However, these algorithms are not a novel contribution of this Thesis and represent previous work done within our research group or the international scientific community. They have been applied to the digital cameras that we use in our experimental results and have been integrated within the UAVision library.

3.1 Color Spaces

A color space is a mathematical model for defining and representing a color. We present in this section five color spaces that are the most used in computer vision. They are: RGB, HSV, YUV, HSL and HSI. The conversions between these five color spaces are based on linear equations [31, 32]. Each of the color spaces has emerged at some moment in history due to the necessity of rendering images on different devices or with different infrastructures [33].

The RGB color space [34, 35] is the foundation of much visual technology, being used mostly 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 [36]. The RGB color space is an additive color space, defined by the three chromaticities of red, green, and blue. To form a color with RGB, three colored light beams (one red, one green, and one blue) must be superimposed (Fig. 3.1). 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 space is not visually uniform and not very intuitive from a visual point of view, as humans do not perceive color as the superimposition of the three primary colors.

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 [36]. HSV stands for hue, saturation, value and it describes colors as points in a cone. The HSV color space is mathematically cylindrical (Fig. 3.2), 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 the (H, S, V) triplet depends on the particular color of red, green, and blue "primaries" used.

The central axis of the cone ranges from black at the bottom to white at the top, with

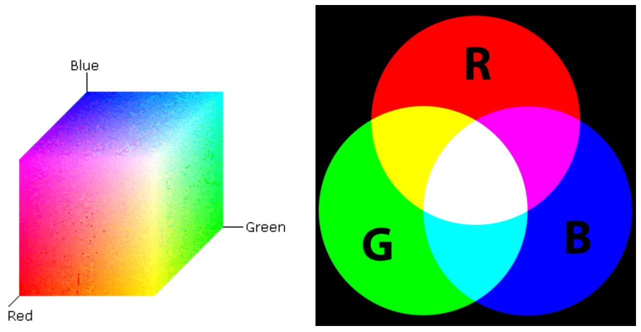


Figure 3.1: On the left, the RGB cube and on the right, an example of an additive color mixing: adding red to green yields yellow, adding all three primary colors together yields white [32].

neutral colors between them, where the angle around the axis corresponds to "hue", the distance from the axis corresponds to "saturation", and the distance along the axis corresponds to "value". The hue represents the percentage of color blend, the saturation is the strength of the color and the value is the brilliance or brightness of the color [35]. Varying H corresponds to traversing the color circle. Decreasing S (de-saturation) corresponds to increasing whiteness, and decreasing V (devaluation) corresponds to increasing blackness [35].

This color model is based on how colors are organized and conceptualized in human vision in terms of hue, lightness, and chroma, as well as on traditional color mixing methods which involve mixing brightly colored pigments with black or white to achieve lighter, darker, or less colorful colors [35]. For these reasons, the HSV color space is considered the most intuitive for humans.

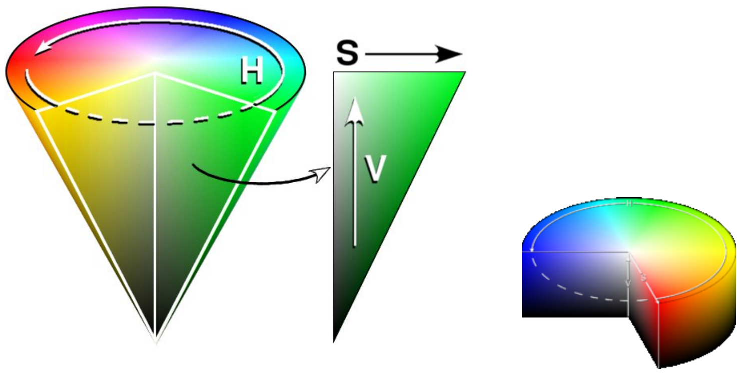


Figure 3.2: The conical and cylindrical representations of the HSV color space [32].

The HSL color space is similar to the HSV one, the definition of hue and saturation being the same as for the HSV color space [33]. The "value" component is replaced by "lightness" and the main difference is the fact that the value, or the brightness of a pure color is considered to be the brightness of white, whereas the lightness of a pure color is the lightness of medium gray. The geometrical representation of the HSL color space is a double cone or double hexcone [35] (Fig. 3.3). Although the HSL color space is used interchangeably with HSV in many textbooks, it was originally used to describe another (distinct) color space. Hue and Saturation are defined as for the HSV color space, but lightness quantifies the energy in a color rather than its non-blackness.

The HSI color space is yet another variation from the HSV color space [32]. The H and S components have the same meaning as for the HSV color space, while I stands for intensity and it is the simple average of the three components of the RGB color space. The advantage is that this representation preserves angles and distances from the RGB cube.

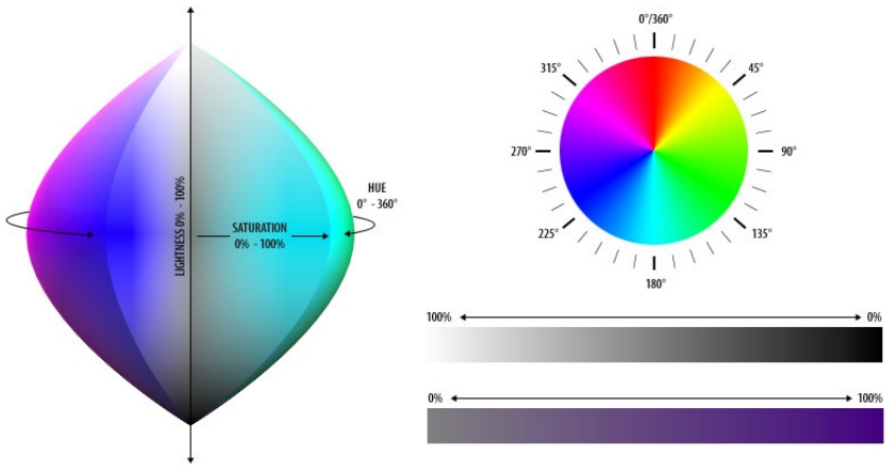


Figure 3.3: The geometrical representation of the HSL color space [33].

In the YUV color space [32, 33], the color is represented in terms of a luminance component (Y stands for luma) and two chrominance, or color, components (U and V) (Fig. 3.4). This color space appeared as a necessity of introducing color television using a black and white infrastructure and encodes a color image also taking the human perception into consideration, that is, separating the luminance information from the color information.

3.1.1 Conversions Between Color Spaces

Conversions from one color space to another are based on mathematic expressions that are presented as follows. The Image Acquisition Module of our library (described in more detail in Appendix A) supports the acquisition of RGB or raw Bayer images [37]. Conversions between different color spaces are also supported, based on the formulas found in [32]. In all following formulas, we consider the R, G and B components with values from 0 to 255.

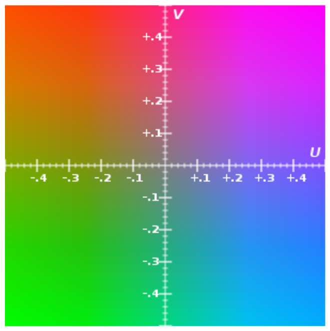


Figure 3.4: Geometrical representation of U-V color plane, when Y =0.5 [33].

- RGB to HSV:

$$V = \max(R, G, B)$$

$$S = \begin{cases} \frac{V - \min R,G,B}{V} & if\ V \neq 0 \\ \\ 0 & otherwise \end{cases}$$

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

- RGB to HSL:

$$V_{max} = \max(R, G, B)$$
$$V_{min} = \min(R, G, B)$$

$$L = \frac{V_{max} + V_{min}}{2}$$

$$S = \begin{cases} \frac{V_{max} - V_{min}}{V_{max} + V_{min}} & if\ L < 0.5 \\ \\ \frac{V_{max} - V_{min}}{2 - (V_{max} + V_{min})} & L \geq 0.5 \end{cases}$$

$$H = \begin{cases} 60(G-B)/S & if\ V_{max} = R \\ 120 + 60(B-R)/S & if\ V_{max} = G \\ 240 + 60(R-G)/S & if\ V_{max} = B \end{cases}$$

- RGB to HSI

$$\begin{aligned}
V_{max} &= \max(R, G, B) \\
V_{min} &= \min(R, G, B) \\
I &= \frac{R+G+B}{3} \\
S &= \begin{cases} \frac{V_{max}-V_{min}}{V_{max}+V_{min}} & if\ L < 0.5 \\ \\ \frac{V_{max}-V_{min}}{2-(V_{max}+V_{min})} & L \geq 0.5 \end{cases} \\
H &= \begin{cases} 60(G-B)/S & if\ V_{max} = R \\ 120 + 60(B-R)/S & if\ V_{max} = G \\ 240 + 60(R-G)/S & if\ V_{max} = B \end{cases}
\end{aligned}$$

- RGB to YUV
  $Y = K_r R + (1 - K_r - K_b)G + K_b B$
  $U = 0.5(B - Y)/(1 - K_b)$
  $V = 0.5(R - Y)/(1 - K_r)$

where $K_r = 0.299$ and $K_b = 0.114$.

3.2 Color Classification Under Different Color Spaces

In robotic soccer the color of an object of interest is the first clue towards its detection. The stability of the colormetric calibration of a digital camera results in the uniformity of a certain color in an acquired image, independently of the type of illumination. We will discuss in Section 3.3 the algorithm that we have designed for the colormetric camera calibration. In order to classify a color of interest, or to determine the digital range of a color of interest we have chosen one manual and one semi-automatic approach. With the use of a GUI tool which we present in more detail in Chapter 7, an user of any of the robotic vision systems that we propose can define either manually, or in a supervised manner, the ranges of a color of interest. This tool was intended for a study regarding color classification under the five color spaces previously presented. We present the results of this study in this Section. Two examples of images that were used in this study can be found in Fig. 3.5. For each object in the image, the user of this tool had to define color ranges for the color of the object, under the five color spaces presented. The color classification procedure could be done in two ways: either manually or user supervised.

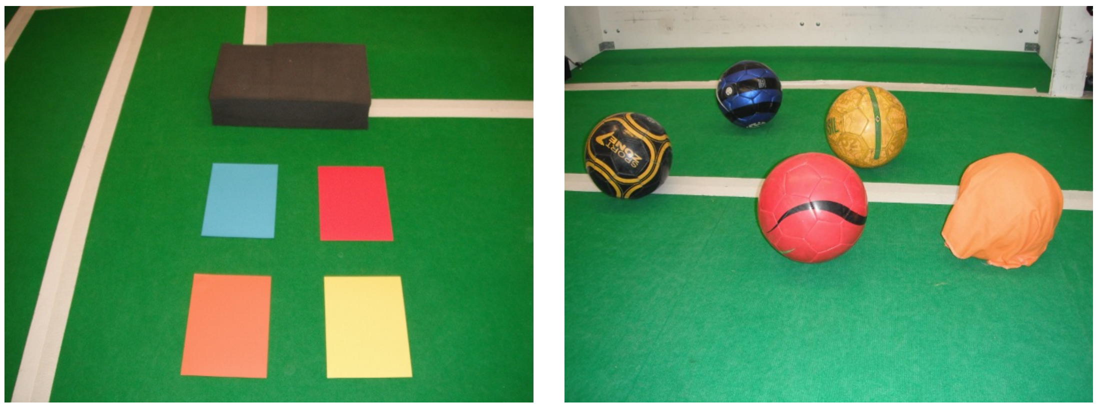


Figure 3.5: On the left, an example of a simple image and on the left a more complex image from de data set that was used in this study.

The manual classification process was based on sliders interaction. The user would define color ranges with the use of sliders and the pixels in the image belonging to that color range would be highlighted. For the supervised color classification process, two region growing algorithms have been tested.

The first approach for supervised color classification was based on the region growing algorithms provided by the OpenCV library [1] [38]. The user has to choose a starting point (seed point) for each of the colors that he wants to classify and the algorithm will return all the pixels in the image that have the same color, +/- a predefined threshold, based on the following equation:

$$src(sP.x, sP.y)_{r/g/b} - infThresh <= src(x, y)_{r/g/b} <= src(sP.x, sP.y)_{r/g/b} + supThresh \tag{3.1}$$

where sP represents the seed point, $infThresh$ is the value of the threshold to be used when calculating the minimum value of the color range and $supThresh$ is the value of the threshold to be used when calculating the maximum value of the color range for the classification of a pixel. In this example the equations are presented for (r, g, b) values of the pixels but they have been used in the same form for the triplets describing the value of a pixel in all the mentioned color spaces.

Starting from the seeding point and based on the values of the threshold introduced by the user, the algorithm will search the neighbor pixels (both 4-neighbor and 8-neighbor implementations were tested). If the color of a neighbor pixel is in the range

$$[(color_{seedPoint} - infThresh), (color_{seedPoint} + supThresh)], \tag{3.2}$$

the neighbor is classified as having the same color as the seed point and it is marked as

[1] www.opencv.org

the new seed point for the following iteration. The algorithm stops when there are no more pixels to be classified.

The second approach was the implementation of a region growing algorithm from scratch. The logic behind the algorithm is the same as previously described, with a small twist. Starting from the seed point, the 4-neighbors or the 8-neighbors pixel values are checked and if a neighbor's value is in the proximity of the seed point's value (+/- a threshold), the pixel is marked as visited and is considered to be the new seed point for the following iteration. The small twist for this algorithm is that the threshold value has to be the same for all of the three components that give the value of a pixel in each of the color spaces.

For the manual classification of the color ranges, the results are presented in Table 3.1. A number of 15 non-specialized subjects have been tested and asked to classify colors in different images. These subjects were either researchers from our research unit or undergraduate students that were taking a Computer Vision class. The results show that most of them had the best performance in the YUV color space, both in terms of correctness and performance time. This result is novel considering that in literature the idea that the HSV color space is more intuitive to humans is promoted [39].

	RBG	HSV	HSL	HSI	YUV
Average Time	5min 33s	4min 19s	4min 34s	4min 49s	4min 13s
Correctness	72%	92%	89%	85%	95%

Table 3.1: Results of the manual color classification task using he *colorSpaces* tool presented in Chapter 7.

Just like the HSV color space, in which the users actually have similar performance as in the YUV color space, the latter separates the color information into luminance and chromaticity. These characteristics make it indeed more intuitive to humans, considering it is similar to the way we perceive colors. This result is very important because most of digital cameras nowadays acquire images in the YUV color space and being able to perform the color classification in the same color space, would save important processing time that is spent in converting the YUV images to a different color space.

Table 3.2 presents the results in terms of correctness of the user-supervised classification. For this part of the study, each user had to choose a seed point for each of the colors of interest. 2 versions of the region-growing algorithm have been used in order to grow a region starting from that seed. These algorithms consider 4 or 8 neighbor pixels when growing the region.

These results show once again that the color classification algorithm performs better in the YUV color space. Moreover, in terms of implementation logic, the results prove to be more accurate when the supervising user has the possibility of choosing different threshold values for the three components of a color.

At the end of the trials, the subjects were also asked to fill in a questionnaire that would

	RBG	HSV	HSL	HSI	YUV
OpenCV-4n	87%	90%	85%	72%	92%
OpenCV-8n	90%	90%	87%	77%	93%
RG-4n	80%	75%	70%	63%	85%
RG-8n	82%	78%	71%	63%	90%

Table 3.2: Correctness of the region growing algorithms for all the color spaces.

help the authors understand if there is any preferred or easier to use color space. The questionnaire is presented in Appendix C. The results show so far that the users preferred YUV, HSV and HSL color spaces, in this order, while RGB and HSI were more difficult to handle. All of them considered the "primary colors", red, green and blue easier to be classified.

3.3 Colormetric Calibration of a Digital Camera

To extract information from an acquired image, such as shapes or colors, a good camera calibration is very important. Camera calibration refers to the calibration of intrinsic and extrinsic parameters of a camera, as well as to the calibration of its colormetric parameters. In this Section we present an algorithm for the colormetric calibration of a camera, which allows a soccer robot to perform its task independently of the lighting conditions, even in outdoor environments.

The most common colormetric parameters of a camera are: gain, exposure, gamma, white-balance, brightness, sharpness, shutter and saturation. If these colormetric parameters of a camera are wrongly calibrated, the image details are lost, noise is added to the image and it may become almost impossible to recognize anything based on shape or color (Fig. 3.6).

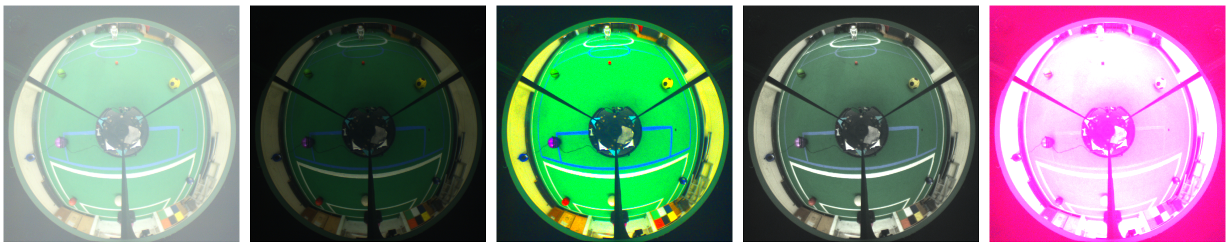


Figure 3.6: Images acquired with wrong parameters. From left to right, wrong value of gamma (high), exposure (low), saturation (high), saturation (low), white-balance (high both in Blue and Red gains).

Digital cameras working in auto-mode fail in acquiring sharp, noiseless images under certain situations or illumination conditions, even considering the most recent cameras. Algorithms developed for calibration of digital cameras assume some standard scenes under some type of light, which fails in certain environments.

Most of the teams participating in RoboCup MSL do not use any algorithm running on the robots to self-calibrate the colormetric parameters of the digital cameras. This means that their cameras are only adjusted manually before a robot starts its operation. Some of the teams however, have tried to solve the problem by developing algorithms for run-time color calibration [40, 41].

Our approach towards solving the lighting conditions independence of a robotic vision system is by adjusting the colormetric parameters of the camera in order to guarantee the correct colors in an image, independent on the source or on the amount of light.

We have designed and implemented an algorithm to configure the most important colormetric parameters of the cameras, namely gain, exposure, gamma, white-balance, brightness, sharpness and saturation, depending on the availability of these parameters in the digital camera that is being used. This approach differs from the well known problem of photometric camera calibration [42], since we are not interested in obtaining the camera response values. We are interested in configuring its parameters according to some statistical measures obtained from the acquired image.

The algorithm works in real-time, in a different thread than the main vision system of the soccer robots. Nowadays computers generally have more than one core and the processes running on them can be shifted to different cores when necessary in order to speed up the performance. This means that different threads can be executed in parallel, at the same time. When the colormetric calibration process is used by the soccer robots in run-time it runs in a different thread, which guarantees that the overall processing time is not influenced by it.

3.3.1 Colormetric Parameters of a Camera

Luminance is normally defined as a measurement of the photometric luminous intensity per unit area of light traveling in a given direction. Therefore, it is used to describe the amount of light that goes through or is emitted from a particular area and falls within a given solid angle.

Chrominance describes the way a certain amount of light is distributed among the visible spectrum. Chrominance has no luminance information but it is used together with it to describe a colored image defined, for instance, by an RGB triplet. Any RGB triplet in which the value of $R = G = B$ has no chrominance information.

Gain, exposure, gamma and contrast are related and we use the information of the luminance of the image to calibrate them. The priority is to keep gamma out and the exposure to the minimum possible value to reduce noise in the image and the effect of the moving objects in the image. Only in worst case scenarios, the algorithm calibrates the gamma and exposure time. Gain is a constant factor that is applied to all the pixels in the image when the image is acquired.

Exposure time is the time that the image sensor (CCD or CMOS) is exposed to light.

Gamma correction is the name of a non-linear operation used to code and decode luminance or RGB tristimulus values. One of the most used definitions of contrast is the difference in luminance along the 2D space that makes an object distinguishable.

The configuration of the parameters of digital cameras is crucial for object detection and has to be performed with every change in the illumination conditions of a scene. The calibration procedure should be effective and fast. The proposed calibration algorithm processes the image acquired by the camera and computes several measurements that allow the calibration of the most important colormetric parameters of a digital camera, as presented in Fig. 3.7. Besides the referred parameters, the hardware related parameters gain and exposure are also taken into consideration.

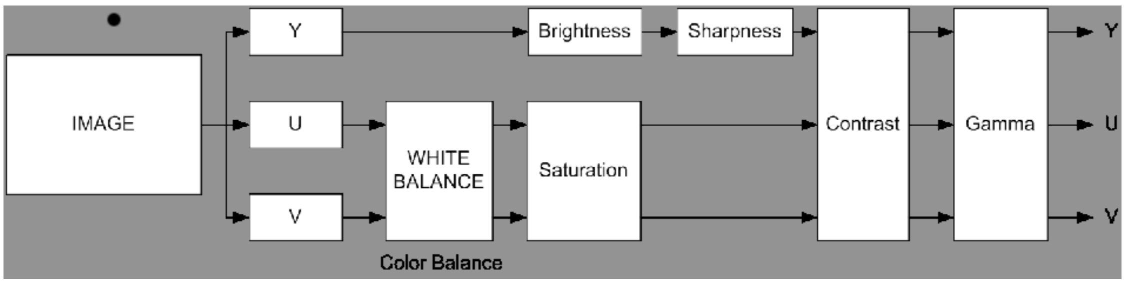


Figure 3.7: A typical image processing pipeline (inside the image device) for a tri-stimulus system. This processing can be performed on the YUV or RGB components depending on the system. This should be understood as a mere example.

To calibrate all these parameters, the histogram of luminance is calculated and a statistical measure to balance the histogram of the acquired image is used. The histogram of the luminance of an image is a representation of the number of times that each intensity value appears in the image. Histograms can indicate if the image is underexposed or overexposed. For a camera correctly calibrated, the distribution of the luminance histogram should be centered around 127 (for an 8 bits per pixel image). An underexposed image will have the histogram leaning to the left, while an overexposed image will have the histogram leaning to the right (Fig. 3.8).

Statistical measures can be extracted from digital images to quantify the image quality [43, 44]. A number of typical measures used in the literature can be computed from the image gray level histogram. Based on the experiments presented in [13], in our algorithm we use the mean sample value (MSV):

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

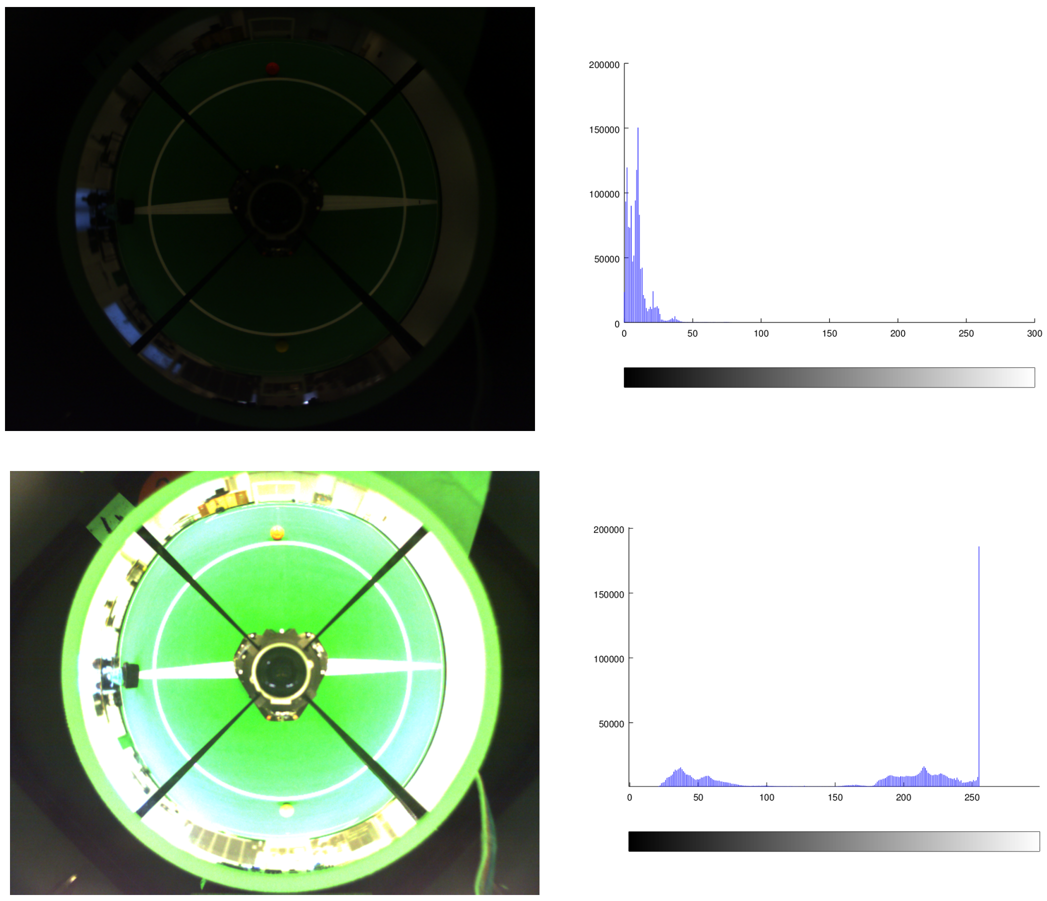


Figure 3.8: On the upper row, an image acquired with the gain parameter set at the minimum value and the histogram of the image. On the lower row, an image acquired with the gain parameter set at maximum and the corresponding image histogram. The upper image is clearly underexposed and its histogram its leaning to the left. The lower image is too bright, making object details and contours to be lost.

---

where x_j is the sum of the gray values in region j of the histogram (in our approach we divide the histogram into five regions). When the histogram values of an image are uniformly distributed within the possible values, then $MSV \approx 2.5$.

Brightness is basically a constant (or offset) that can be added (or subtracted) from the luminance component of the image. It represents a measure of the average amount of light that is integrated over the image during the exposure time. If the brightness it too high, overexposure may occur. If this happens, the saturated part or the totality of the image will become whiten. The proposed algorithm considers a black area in the image as reference to calibrate this parameter. The concept is that the black area should be black - in the RGB color space, this means that the average values of R, G and B should be close to zero in this region.

White balance is the global adjustment of the intensities of the colors (typically red, green, and blue - primary colors). An important goal of this adjustment is to render specific colors – particularly neutral colors – correctly; hence, the general method is sometimes called gray balance, neutral balance, or white balance. This balance is required because of the different color spectrum energy distribution depending on the illumination source. The proposed algorithm uses a gray area as reference to calibrate this parameter. The gray region should not have color – in the YUV color space this means that the average value of U and V should be 127 for cameras that have blue and red white balance control parameters. In cameras that have individual control of each of the three white balance gains (red, green and blue) it is expected that the average R, G and B values of a gray region to have the same value and that value should be, ideally, 127. Figure 3.9 shows the effects of the white balance parameter on the acquisition of an image.

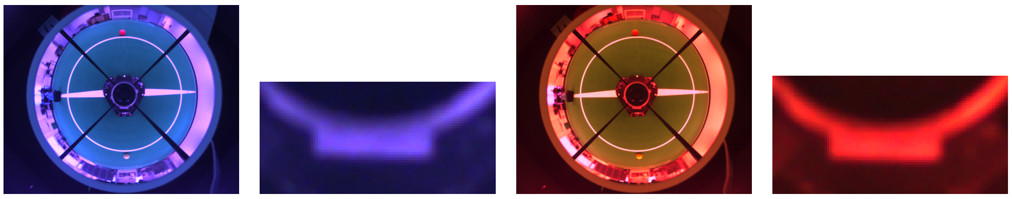


Figure 3.9: Wrongly calibrated blue and red gains and the effect that has on the acquired images. Zooming in into a supposedly gray region when each of these gains is wrongly calibrated shows that the field lines become very noise and harder to be classified as white.

In the case of the CAMBADA robots, due to the use of an omni-directional vision system, the body of the robot can be seen in the image and a white and black area were placed close to the mirror.

The saturation of a color is determined by a combination of light intensity that is acquired by a pixel and how much this light is distributed across the spectrum of different wavelengths. Saturation is sometimes also defined as the amount of white that is blended into a pure color. In the proposed algorithm, we consider the histogram of the saturation (obtained in the HSV color space) and we force the MSV value of this histogram to 2.5, calibrating thus this parameter.

Sharpness is a measure of the energy frequency spatial distribution over the image. It basically allows the control of the cut-off frequency of a low pass spatial filter. This may be very useful if the image is afterwards intended to be decimated, since it allows to prevent spatial aliases artifacts. We do not consider this parameter in the proposed calibration algorithm as in the referred applications of the robots we work with the resolution of the images acquired by the camera.

A graphical representation of the statistical measures extracted from the image acquired by the camera and their relation to the parameters to be calibrated is presented in Fig. 3.10.

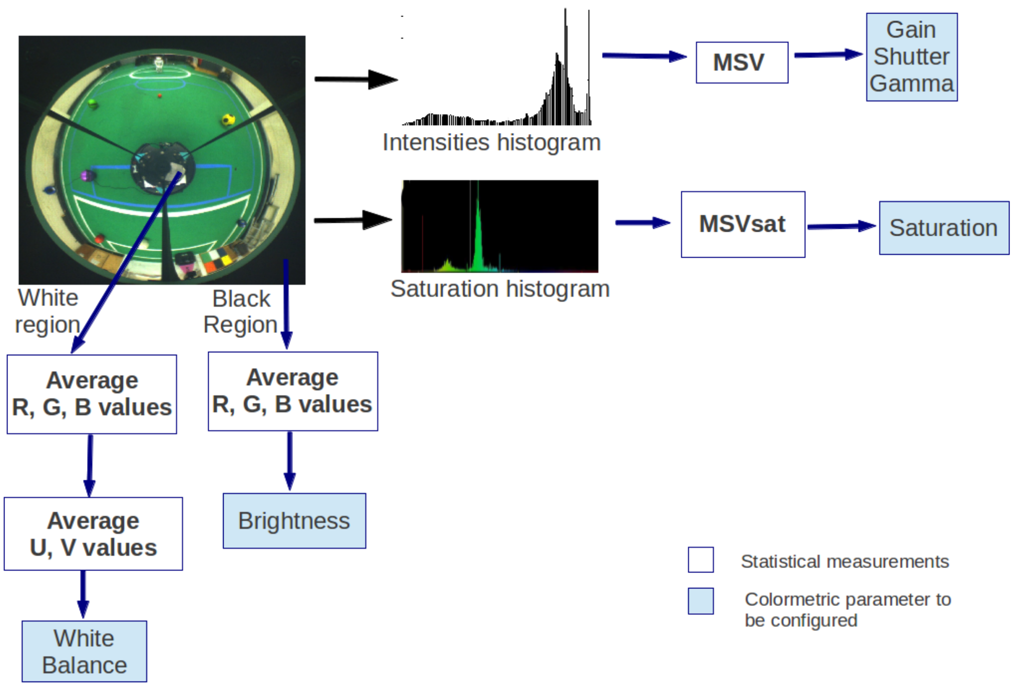


Figure 3.10: A graphical representation of the statistical measures extracted from the image acquired by the camera and their relation to the parameters to be calibrated on the camera.

3.3.2 Proposed Algorithm

The algorithm that we designed configures the most important parameters, as referred above. For each one of these parameters that are available on a given camera, a PI controller has to be implemented. PI controllers are used instead of proportional controllers as they result in better control, having no stationary error. The constants of the controller must be obtained experimentally, guaranteeing the stability of the system and an acceptable time to reach the desired reference.

The coefficients of the controller were obtained experimentally [14]: first, the proportional gain was increased until the camera parameter started to oscillate. Then, it was reduced to about 70% of that value and the integral gain was increased until an acceptable time to reach the desired reference was obtained [45]. The convergence time of the camera parameters are directly related to the efficiency of the PI controller. In this algorithm, the aim was not the deep exploration of the control efficiency, but rather the design of a simple solution that can provide the expected outcome in terms of what is the acceptable time for a given application.

The algorithm presented next starts by the configuration of the parameters related to the luminance on the image, namely gain, exposure and gamma, by this order if necessary. To improve the quality of the image, i.e. to have as less noise as possible, exposure should be as high as possible. On the other hand, gamma should be the one that gives the best dynamic range for the intensity and we only want to change it if the gain and exposure alone cannot deliver good results.

When the image acquired has enough quality in terms of luminance, considering that the MSV for the histogram of intensities is between 2 and 3, the algorithm starts calibrating the other parameters, namely white-balance and brightness, according to the ideas expressed in the previous Section. Saturation is calibrated based on the MSV value of the saturation histogram (Fig. 3.11), which should also be 2.5.

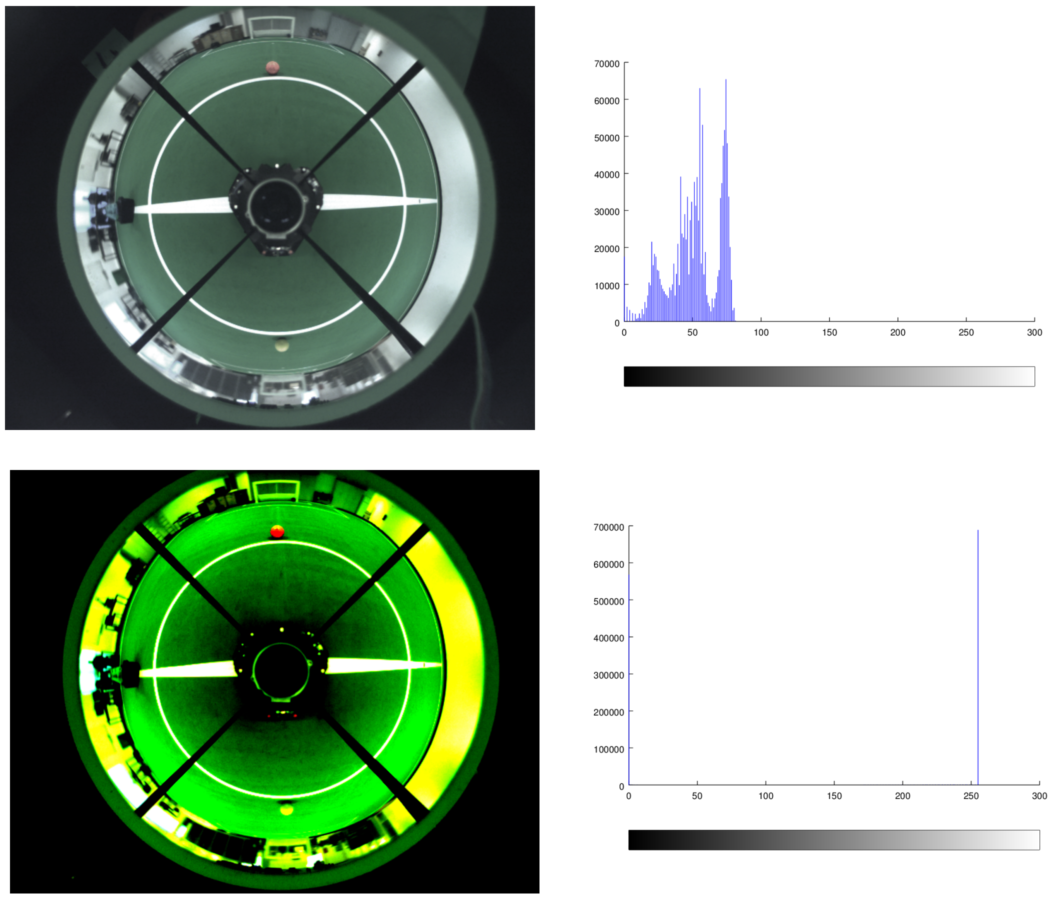


Figure 3.11: On the upper row, an image acquired when saturation is set at the minimum value and the corresponding saturation histogram. On the lower row, an image acquired when saturation is set to maximum and the corresponding saturation histogram.

The algorithm stops when all the parameters have converged. This procedure solves the problem of the correlation that exists between the parameters.

```
do
  acquire image
  calculate the histogram of Luminance
  calculate the MSV  value for Luminance
  if MSV != 2.5
    if exposure and gain are in the limits
      apply  PI controller to adjust gamma
    else if gain is in the limit
      apply  PI controller to adjust exposure
    else
      apply PI controller to adjust gain
   end
   set the camera with new gamma, exposure and gain values
  end
  if  MSV > 2 && MSV < 3
    calculate the histogram of saturation
    calculate the MSV value for saturation
    calculate average U and V values of a white area
    calculate average R, G and B values of a black area
     if MSVsat != 2.5
      apply the PI controller to adjust saturation
      set the camera with new saturation value
    end
if U != 127 || V != 127
  apply the PI controller to adjust WB_BLUE
      apply the PI controller to adjust WB_RED
      set the camera with new white-balance parameters
    end
if R != 0 || G != 0 || B != 0
      apply the PI controller to adjust brightness
      set the camera with new brightness value
end
  end
while any parameter changed
```

3.3.3 Experimental Results

The experiments that follow have been conducted using the cameras of the CAMBADA robots with different initial configurations inside a laboratory with both artificial and natural light sources. In Fig. 3.12, the experimental results are presented both when the algorithm starts with the parameters of the camera set to the lowest value as well as when set to the highest values. As it can be seen, the configuration obtained after using the proposed algorithm is approximately the same, independently of the initial configuration of the camera.

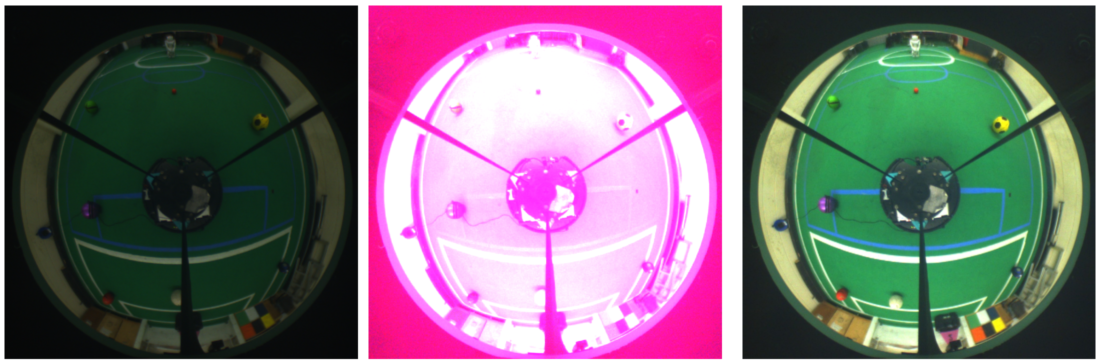


Figure 3.12: Some experiments using the proposed automated calibration procedure. On the left an image captured with some of the parameters of the camera set to lower values. In the middle, an image captured with some of the parameters of the camera set to higher values and on the right, the corresponding image obtained after applying the proposed calibration procedure.

In Fig. 3.13 we present the variation of the camera parameters related to the experiment described above. The convergence of the parameters is fast. It took less than 100 cycles for the camera to converge to the correct parameters in order to obtain the images presented in Fig. 3.12.

In this experiment, the camera was working at 30 fps and that means a calibration time below 3 seconds. These are the worst case scenarios in colormetric calibration. Most of the times, in practical use, the camera can start in auto mode and the algorithm is applied after that. In these situations, the camera converges in a reduced number of cycles.

Table 3.3 presents the average processing time taken by the calibration thread when running in runtime. The most time consuming operation is the computation of the MSV.

In Fig. 3.14 we present an image acquired with the camera in auto mode. The results obtained using the camera with the parameters in auto mode are overexposed and the white balance is not correctly configured. The auto-calibration algorithms that come integrated in the digital cameras are not sufficient in these applications because the cameras are typically expecting a natural image. In most robotic applications, as is the example of robotic soccer, the scenario has specific characteristics (omni-direction vision systems, green carpets, black robots, ...) that turn the image less natural. Moreover, as the case of the omni-directional vision system, the camera analyses the entire image and, as can be seen in Fig. 3.12, there are

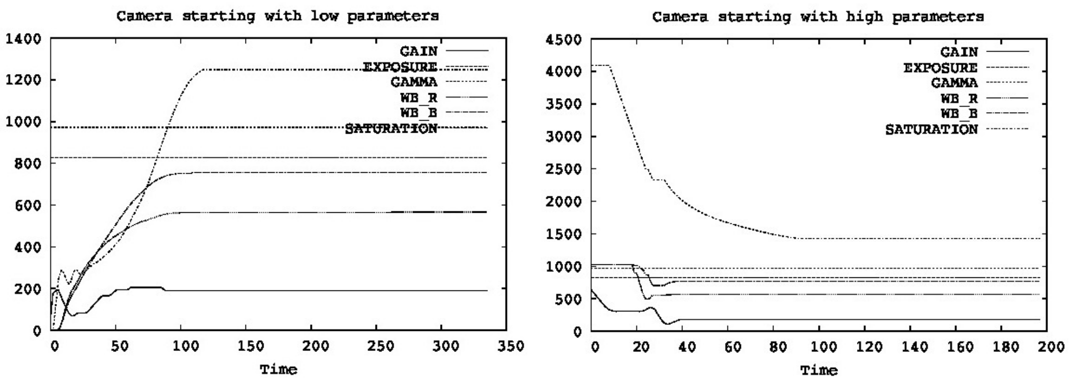


Figure 3.13: On the left, a graphic showing the variation of the parameters of the camera when the camera started with low values. On the right, a graphic showing the variation of the parameters of the camera when the camera started with high values. In both experiments, there is a fast convergence of the parameters.

Operation	Time [ms]
Calculate RGB mean value	0.0043
Calculate YUV mean value	0.0045
Calculate MSV	2.05
Set new camera parameters	0.00035
Total	2.13

Table 3.3: Processing time of the colormetric calibration algorithm.

large black regions corresponding to the robot itself. In addition, due to the changes in the environment around the robot as it moves, leaving the camera in auto mode would lead to undesirable changes in the parameters of the camera, causing problems for the correct object detection.

The good results of the automated calibration procedure can also be confirmed by the histograms presented in Fig. 3.15. The histogram of the image obtained after applying the proposed automated calibration procedure (Fig. 3.15 (b)) is centred near the intensity 127. The histogram of the image acquired using the camera in auto mode (Fig. 3.15 (a)) shows that the image is overexposed, leading to the majority of the pixels to have saturated values.

As far as we know about the work of other teams that participate in RoboCup Soccer (MSL, SPL and HL), as well as according to our personal experience, most of the robots uses their digital cameras in manual mode even with most advanced and recent cameras. This statement is easily supported and verified in practice. The algorithms developed by the industry for calibration of digital cameras assume some standard scenes under some type of light. However, besides the user manuals of the cameras tested, we did not find scientific

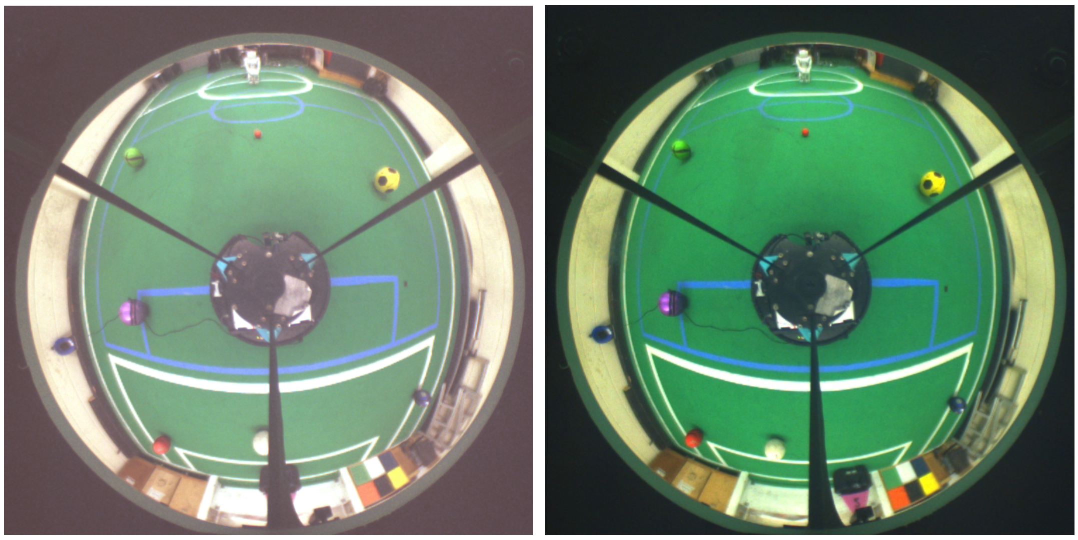


Figure 3.14: On the left, an image acquired with the camera working in auto mode and on the right, an image after the proposed algorithm.

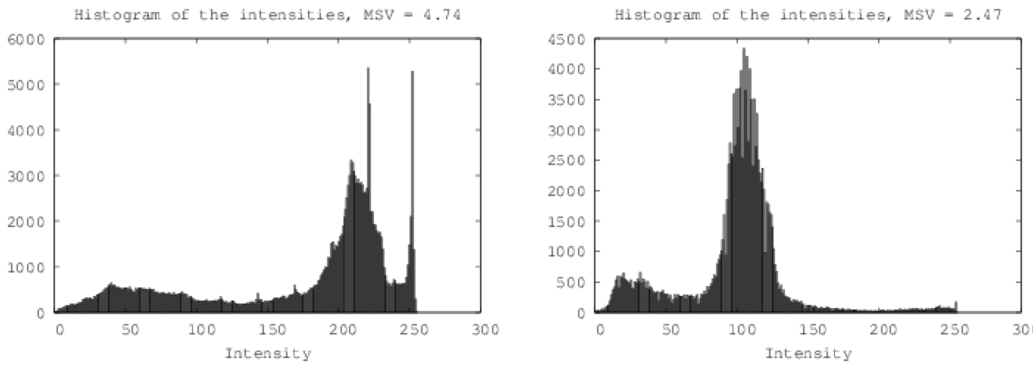


Figure 3.15: The histogram of the intensities of the two images presented in Fig. 3.12. On the left, the histogram of the image obtained with the camera parameters set to maximum. On the right, the histogram of the image obtained after applying the colormetric calibration algorithm.

references for these algorithms.

In a near future, it is expected that the MSL robots will have to play under natural lighting conditions and in outdoor fields [46]. This introduces new challenges. In outdoor fields, the illumination may change slowly during the day, due to the movement of the sun, but also may change quickly in short periods of time due to a partial and temporarily varying covering of the sun by clouds. In this case, the vision systems of the robots have to adjust, in real-time, the camera parameters, in order to adapt to new lighting conditions. Figure 3.16 shows that the algorithm works well even with different light conditions, outdoor.

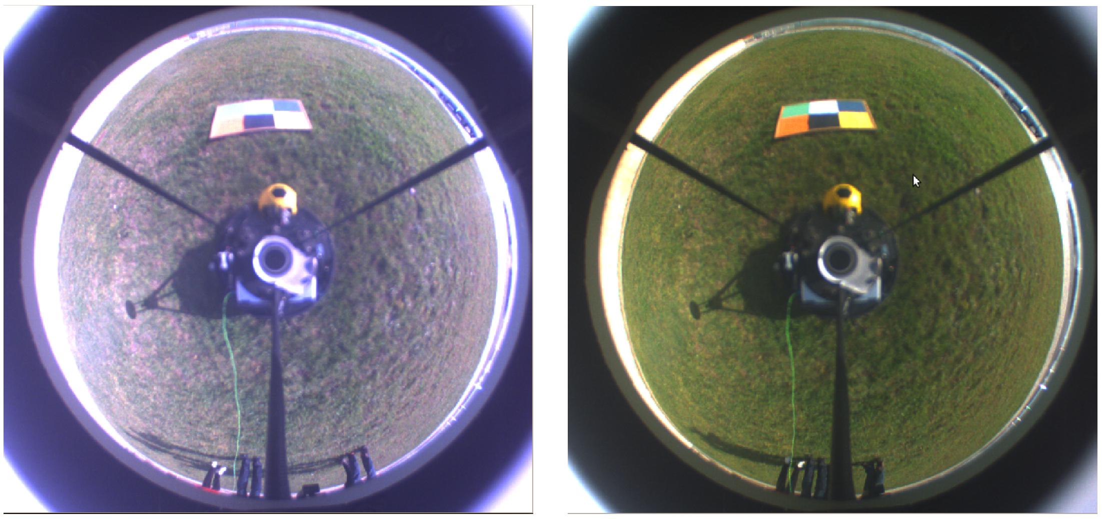


Figure 3.16: On the left, an image acquired outdoors using the camera in auto mode. The colors in this image are washed out. That happens because the camera's auto-exposure algorithm tries to compensate the black around the mirror. On the right, the same image with the camera calibrated using the proposed algorithm. In this image the colors and their contours are better defined.

3.4 Calibration of Intrinsic and Extrinsic Camera Parameters

The Camera Calibration Module of UAVision includes two algorithms for the calibration of the intrinsic and extrinsic parameters of a digital camera. A first algorithm, based on the work presented in [14, 47], can be used for the calibration of an omni-directional vision system. The second one, presented in [48], is a well-known algorithm for the intrinsic and extrinsic calibration of a perspective camera. These algorithms do not hold considerable scientific novelty within this Thesis, they have been studied and integrated within the library in order to deliver a cohesive and coherent Camera Calibration Module.

3.4.1 Pinhole Camera Model

The principle of the camera obscura, or better known as pinhole camera, is based on passing light rays through a small hole. If the hole is sufficiently small, an image of the outside view on the opposite wall inside the camera will be created [49] (Fig. 3.17). The size of the hole is of the orders of millimetres. This basic principle of projection, based on the rectilinear propagation of light has been first recorded as a natural occurring phenomenon, in ancient China. The principle is valid until today and it is used in any camera. In digital cameras the celluloid film that was initially used has been replaced by a CCD or CMOS sensor.

Camera calibration is a necessary step in any computer vision algorithm which has to ex-

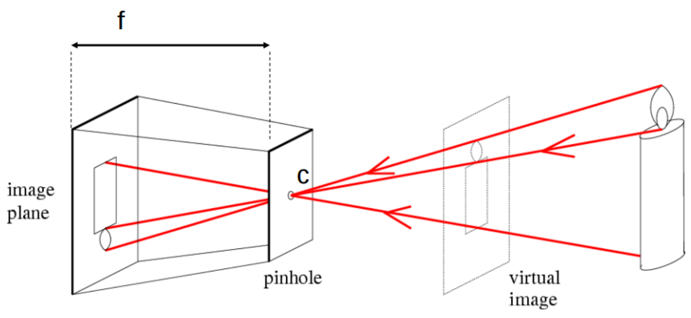


Figure 3.17: Pinhole camera principle [49]. In this representation, f represents the focal length and c is the center of the camera.

tract metric information from 2D images. For this, the intrinsic and extrinsic parameters of a camera have to be calculated. The intrinsic parameters of a camera link the image coordinates of an image point to its corresponding camera coordinates. The intrinsic parameters are: the image center (or principal point), the size of the pixel and the focal length. Moreover, since a real camera lens does not act like a perfect pinhole, there are deviations from the model illustrated above. The most significant of these is known as radial distortion [50]. The extrinsic parameters of a camera are: a 3D translation vector, t, describing the relative locations of the origins of the two coordinate systems (image coordinates and camera coordinates) and a 3×3 rotation matrix, R, an orthogonal matrix that brings the corresponding axes of the two systems onto each other.

The relationship between a 3D point P and its image projection p (Fig. 3.18) is given in [51]:

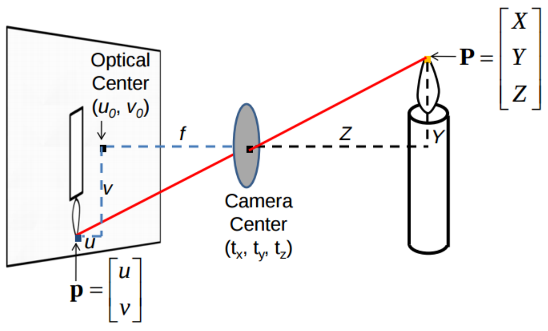

Optical Center
(u_0, v_0)
f
Z
Y
v
u
Camera Center
(t_x, t_y, t_z)
$\mathbf{P} = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$
$\mathbf{p} = \begin{bmatrix} u \\ v \end{bmatrix}$

Figure 3.18: Projection of a 3D point into a 2D point based on the pinhole model.

$$s\tilde{p} = A[Rt]\tilde{P}, \tag{3.3}$$

where :

- $\tilde{p}$ is the augmented vector, with the homogeneous coordinates of p: $\tilde{p} = [u, v, 1]^T$ and $\tilde{P} = [X, Y, Z, 1]^T$
- s is an arbitrary scale factor
- R is the rotation matrix
- t is the translation vector
- A is the camera intrinsic matrix and is given by:

$$A = \begin{pmatrix} \alpha & \gamma & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{pmatrix}$$

  with

  - (u_0, v_0) being the coordinates of the principal point
  - α and β the scale factors in image u and v axes
  - γ the parameter describing the skewness of the two image axes.

The method for camera calibration proposed in [51] is probably the most used method nowadays, mainly due do its simplicity. This approach only requires that the camera should observe a planar pattern shown at at least two different orientations. Most of the times the chosen planar pattern is a chessboard (Fig. 3.19).

The implementation of this algorithm in the OpenCV library[2] has been integrated in the UAVision library as part of the Calibration Module. A graphical application has been designed in order to calculate the relation between image coordinates and world coordinates. The user of this application has to click on a point in an image and manually introduce its coordinates in world referential. This algorithm is used for the calibration of perspective video systems.

The application works as follows: the user provides several views of the chessboard (Fig. 3.20), at different orientations. The corners of each square are calculated and the intrinsic parameters of the camera are estimated based on this relation. For the calculation of the extrinsic parameters, the user has to click on a point in an image and manually introduce its coordinates in real-world referential. The positioning of the camera relative to a global referential can be estimated by the correspondence of a pixel in a 2D image and the 3D coordinates of the same point.

[2]`http://docs.opencv.org/2.4/doc/tutorials/calib3d/camera_calibration`

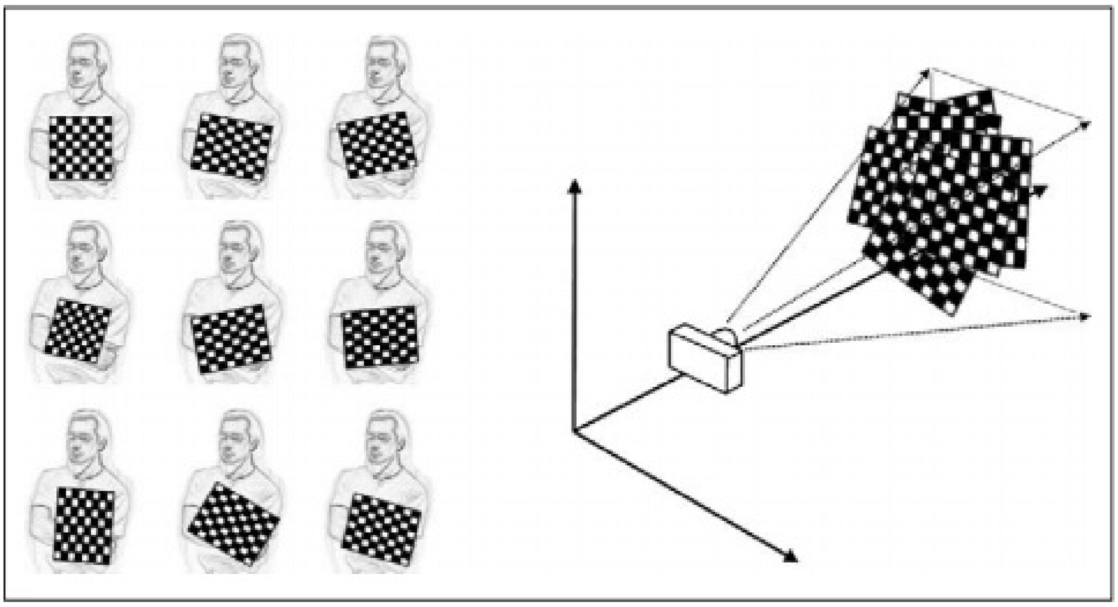


Figure 3.19: Chessboard images at different orientations [52] used for the calibration process.

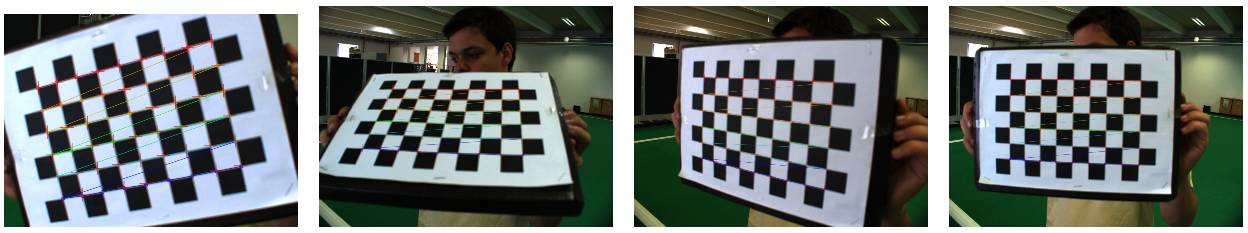


Figure 3.20: Different orientations of the chessboard pattern chosen for the calibration of the intrinsic and extrinsic camera parameters. Corners are detected and highlighted in different colors.

For an omni-directional vision system, by exploring a back-propagation ray-tracing approach and the physical properties of the mirror surface [47], the relation between the distances in the image and the distances in the real world is known.

This solution effectively compensates for the misalignment that results from the mechanical setup. Two simple image feedback tools have been developed. The first one creates a reverse mapping of the acquired image into the real world distance map. A fill-in algorithm is used to integrate image data in areas outside pixel mapping on the ground plane. This produces a plane vision from above, allowing visual check of line parallelism and circular asymmetries (Fig. 3.21).

The second generates a visual grid with 0.5 m distances between both lines and columns, which is superimposed on the original image. This provides an immediate visual clue for the need of possible further distance correction (Fig. 3.22). With this tool, it is also possible to

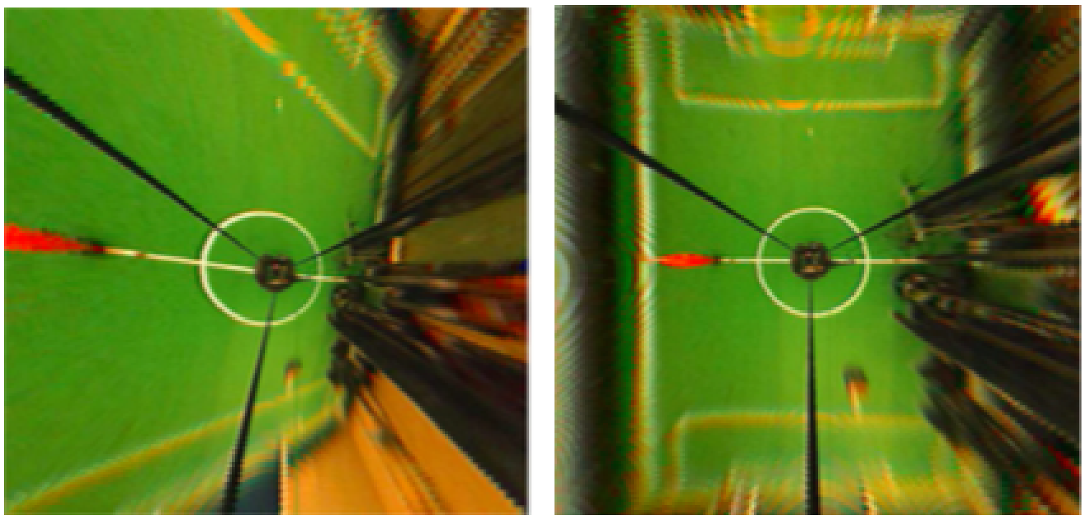


Figure 3.21: Acquired image after reverse-mapping into the distance map. On the left, the map was obtained with all misalignment parameters set to zero. On the right, after automatic correction.

determine some other important parameters, namely the mirror center and the area of the image that will be processed by the object detection algorithms.

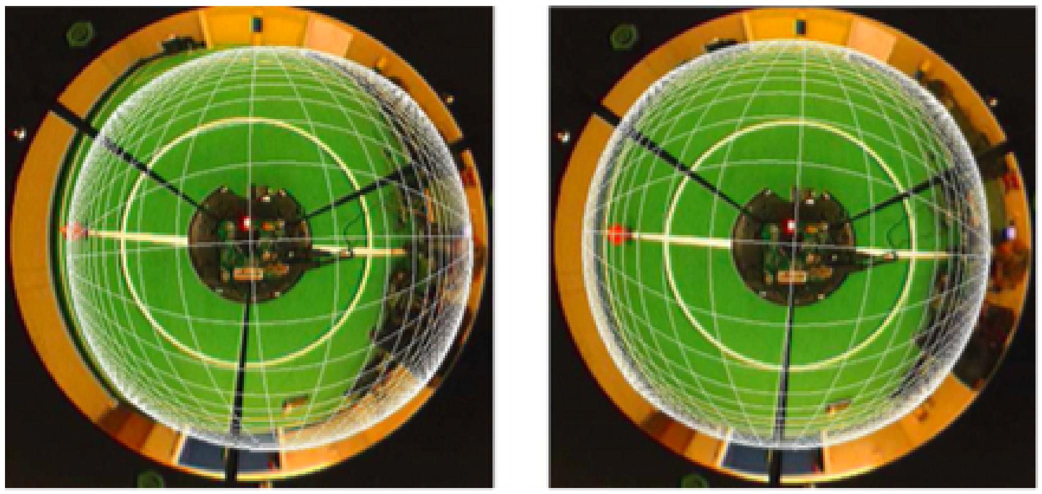


Figure 3.22: A 0.5 m grid, superimposed on the original image. On the left, with all correction parameters set to zero. On the right, the same grid after geometrical parameter extraction.

"Science never solves a problem without creating ten more."

George Bernard Shaw

4

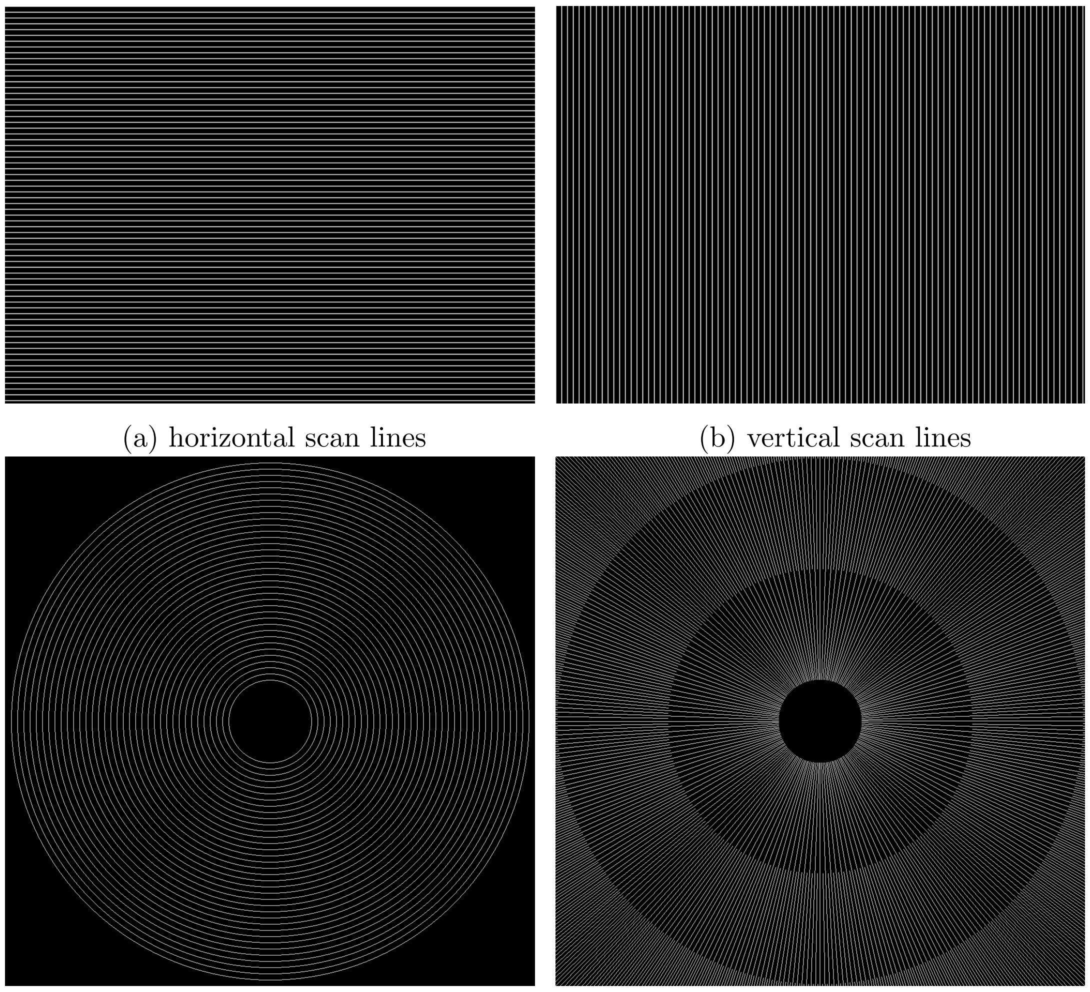
Colored Object Detection

Current advances in technology along with the decrease of prices of image sensors and digital cameras, allow nowadays that digital vision systems acquire images at resolutions higher than 1 Megapixel at high frame rates. This implies that a considerable data size has to be processed in a very small period of time, in applications where real-time operations and data processing are required. For example, for a camera working at $50fps$ the total processing time of any given real-time application should not exceed $20ms$. Industrial inspection and production lines, where autonomous robots or robotic arms repeatedly perform the same task that includes visual processing are just an example of applications in which cameras working at high frame rates are desirable. Another example is of course, intelligent autonomous robots, as the case of robotic soccer players. In MSL, the ball and the robots move at high speeds and thus the need to process the environment at the same pace arises. When the resolution of the images acquired by these cameras are of 1 Megapixel or even more, the amount of data that has to be processed is significantly high and inversely proportional to the time available for its processing.

In this Chapter we detail the time constrained algorithms that have been implemented for the detection of colored objects. These algorithms have been integrated into the UAVision library, presented in Appendix A.

4.1 Scan Lines

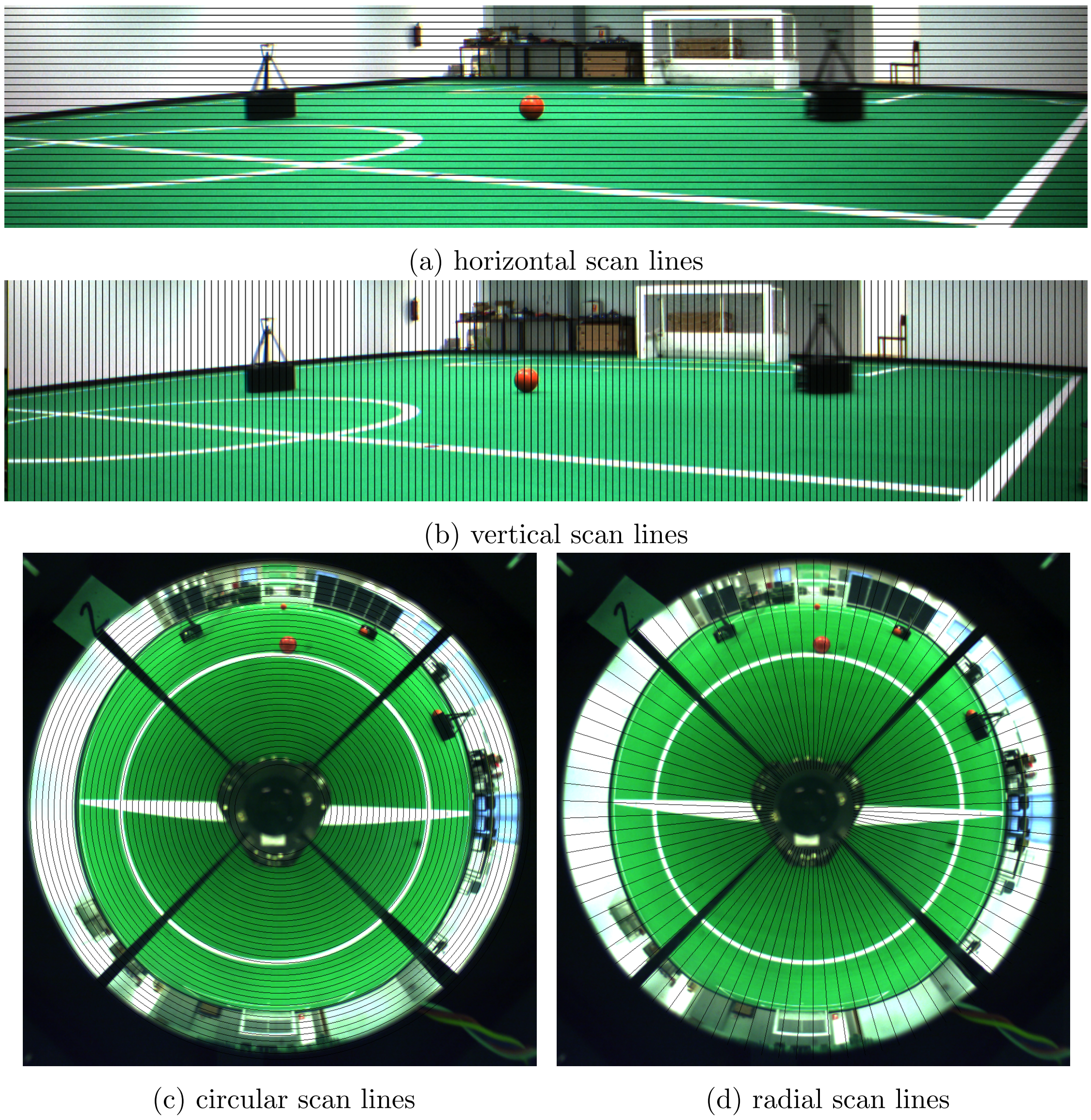
A simple solution for reducing the processing time of an object detection algorithm is to sample the data, to process only parts of an image or to acquire and process smaller resolution images. The challenge is to perform a sampling that guarantees that no important information is lost. Most algorithms for colored object detection, just as the name implies, start with a color segmentation step in which pixels of the same color are grouped together. For scanning an image in search of some type of information, we introduce the notion of scan lines. A scan line, or a search line, is a list of pixel positions in the image, whose shape depends on its type. We have designed three different types (or patterns) of scanning: linear (which can be horizontal or vertical), radial or circular. Figure 4.1 shows the three different types of scan lines.


(a) horizontal scan lines (b) vertical scan lines

(c) circular scan lines (d) radial scan lines

Figure 4.1: Examples of different types of scan lines.

Figure 4.2 shows the same types of scan lines on images acquired by a perspective and an omni-directional vision systems of the soccer robots. The horizontal and vertical scan lines are shown on an image acquired by a perspective vision system. The circular and the radial scan lines are shown on an image acquired by an omni-directional vision system. The circular and radial scan lines are not used on the central part of the image, which includes the robot itself. The radial lines presented in this image are displayed on a single level, simply for a better visualization.


(a) horizontal scan lines

(b) vertical scan lines

(c) circular scan lines

(d) radial scan lines

Figure 4.2: Examples of the 4 types of scan lines displayed on the images acquired by robotic vision systems.

In order to save processing time, the scan lines are constructed once, at the beginning of the object detection process, and saved in a structure. This approach improves the access to these pixels by other modules of the vision pipeline that will be presented next (Fig. 4.3). The scan lines are highly configurable when created. The user can choose to sweep all pixels in a given direction (horizontal, vertical, radial or anti-clockwise) or each n pixels (where n is user defined). The sparsity within a scan line is also denoted from herein as intra-scan line sparsity. Moreover, scan lines can be created adjacent or a given distance between consecutive scan lines can be specified when created. The distance between consecutive scan lines is denoted from herein as inter-scan line sparsity. A scan line is described by its start and end points and the total number of points that make up the scan line. The amount and characteristics of these scan lines depend on the parameters (start and end positions, vertical, horizontal or angular inter-scan line sparsity and intra-scan line sparsity).

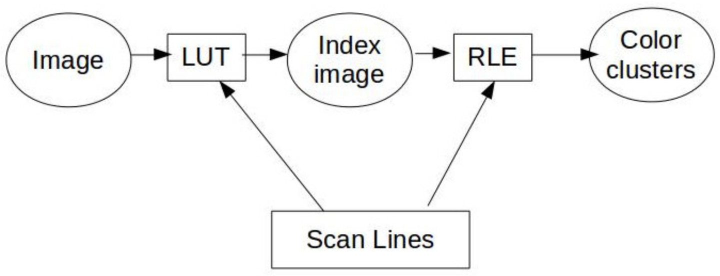


Figure 4.3: Diagram of the use of scan lines for color detection.

For the linear scan lines, the positions in the scan line are the result of a horizontal or vertical sweep of the image. Figure 4.4 presents a more graphical description of the linear scan lines.

The linear scan lines can be constructed on more than one level, if needed. In this case, the level of the scan lines correspond to different lengths. For example, in systems in which the digital camera is positioned so that it acquires images of the environment under a certain angle, vertical scan lines can be constructed on more than one level considering different distances away from the camera.

The radial and circular scan lines are constructed based on Bresenham's algorithm for approximating a straight line between two points [53], and its modified version for rendering circular arcs [54]. Bresenham's algorithm is a fast, yet simple algorithm, commonly used in computer graphics, that was initially designed for rendering straight lines on a computer screen. We have chosen this algorithm for designing the radial scan lines and a slightly modified version for the creation of the circular scan lines, mainly because the algorithm is based on cheap operations such as subtractions, additions or bit-shifting.

The algorithm assumes that the pixel coordinates on a display increase to the right and

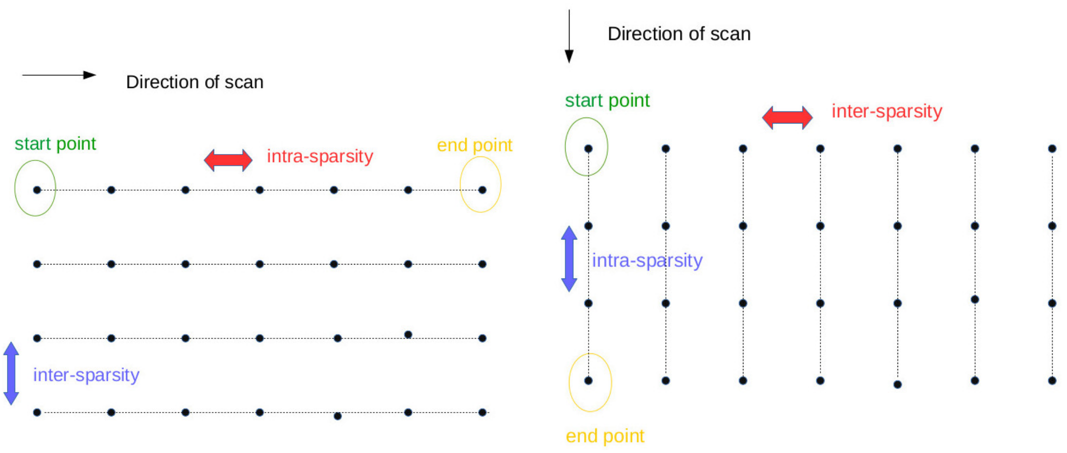


Figure 4.4: Graphical description of horizontal scan lines on the left and vertical scan lines on the right.

downwards, being the top left point $(0, 0)$. This algorithm only deals with integer positions, therefore the center of a pixel is also an integer. Given a line segment whose start point is (x_0, y_0) and its end point is (x_1, y_1), its slope can dictate the intermediate points that are found on that line segment. The slope, m, is calculated based on the following formula:

$$m = \frac{y_1 - y_0}{x_1 - x_0}. \tag{4.1}$$

If the absolute value of the slope is ≥ 1, the line is more horizontal than vertical. Otherwise, the line is more vertical. For example, for a more horizontal line segment, in which the following condition is verified:

$$|x_1 - x_0| \geq |y_1 - y_0|$$

the ideal y values for successive integer values of x can be computed starting from y_0 and repeatedly adding the slope.

The radial scan lines can be organized on multiple levels or layers (Figure 4.5).

This approach prevents the passage through the same positions more than once and significantly reduces both the time for the construction of the scan lines, as well as the total processing time of the algorithms that will follow. In this way, we guarantee that only a small, but meaningful, number of pixels will have to be processed further along the vision system pipeline.

The number of levels and total number of scan lines, as well as the intra-scan line sparsity are defined by the user when the radial scan line is created. The number of scan lines, per level, is calculated following Algorithm 1.

The pseudocode for calculating the pixel positions on a radial scan line is presented in Algorithm 2.

Algorithm 1 Computation of the number of radial scan lines on a level.

```
for $l = 0; l < nLevels; l++$ do
    if $l == 0 || l == 1$ then
        $nSensorsOnLevel \leftarrow nSensors/2^{nLevels-1}$
    else
        $nSensorsOnLevel \leftarrow 2 \times nSensorsOnLevel$
    end if
end for
```

Algorithm 2 Computation of the the integer pixel positions on a radial scan line.

```
$dx \leftarrow (x1 - x0)$
$dy \leftarrow (y1 - y0)$
$dxabs \leftarrow abs(dx)$
$dyabs \leftarrow abs(dy)$
$x \leftarrow dyabs >> 1$
$y \leftarrow dxabs >> 1$
$px \leftarrow x1$
$py \leftarrow y1$
if $dxabs \geq dyabs$ then
    for $i = 0; i < dxabs; i++$ do
        $y \leftarrow y + dyabs$
        if $y \geq dxabs$ then
            $y \leftarrow y - dxabs$
            $py \leftarrow py + signum(dy)$
        end if
        $px \leftarrow px + signum(dx)$
        $scanline \leftarrow px + py \times image.cols$
    end for
else
    for $i = 0; i < dyabs; i++$ do
        $x \leftarrow x + dxabs$
        if $x \geq dyabs$ then
            $x \leftarrow x - dyabs$
            $px \leftarrow px + signum(dx)$
        end if
        $py \leftarrow py + signum(dy)$
        $scanline \leftarrow px + py \times image.cols$
    end for
end if
```

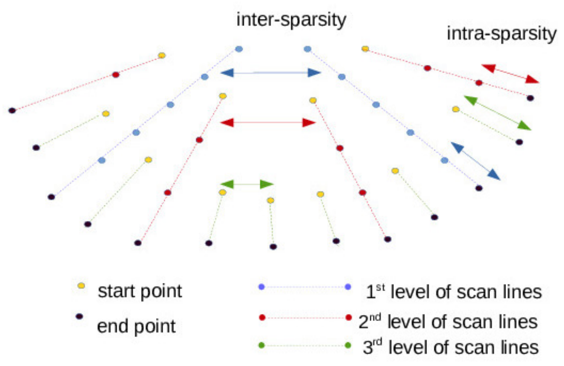


Figure 4.5: Graphical description of radial scan lines disposed on three levels.

The algorithm for calculating the pixel positions on a circular scan line considers the image plane divided in eight octants, as shown in Fig 4.6.

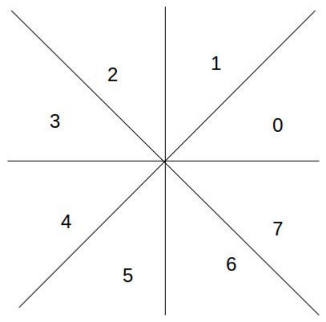


Figure 4.6: Octants considered for the construction of circular scan lines. The numbering of the octants points toward the direction in which the lines are constructed, which is anti-clockwise.

A more detailed illustration of circular scan lines can be seen in Fig. 4.7.

The equation of a circle is used in order to determine all the integer pixel positions that belong to the same scan line. Given a circle of radius r and center (x_0, y_0), its equation is given by:

$$(x - x_0)^2 + (y - y_0)^2 = r^2. \tag{4.2}$$

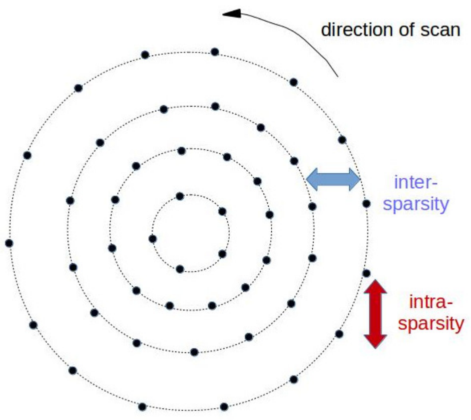


Figure 4.7: Graphical description of circular scan lines.

Given a point $P(x_p, y_p)$, the following measure gives us an idea about where P is situated relatively to the previously introduced circle:

$$\sqrt{x_p^2 + y_p^2} - \sqrt{r^2}. \tag{4.3}$$

If this measure is 0, P lies on the circle. If this measure is negative, P lies inside the circle and if this measure is positive it means that P lies outside the circle. The algorithm for constructing circular scan lines uses a lighter version of this measure in order to decide which are the pixel positions that lie on a given circular scan line:

$$error = |x_i^2 + y_i^2 - r^2|.$$

Algorithm 3 illustrates the construction of a circular arc, in the first octant.

Algorithm 3 Computation of the pixel positions on a circular arc, in the first octant.

```
1:  $error1 \leftarrow fabs(radius - sqrt((p.x)*(p.x) + (p.y)*(p.y)))$
2:  $error2 \leftarrow fabs(radius - sqrt((p.x)*(p.x) + (p.y-1)*(p.y-1)))$
3:  $error3 \leftarrow fabs(radius - sqrt((p.x)*(p.x) + (p.y+1)*(p.y+1)))$
4:  if $error2 < error1$ then
5:      $p.y \leftarrow p.y - 1$
6:      $octant \leftarrow 1$
7:  else
8:      if $error3 < error1$ then
9:          $p.y \leftarrow p.y + 1$
10:         $octant \leftarrow 7$
11:     end if
12: end if
```

4.2 Look-Up Tables

Look-Up Tables (LUT) are data structures, in this case arrays, used for replacing a runtime computation by a basic array indexing operation. We make use of look-up tables for solving different tasks of the object detection algorithms that we have designed. For a fast color classification, color classes are defined through the use of a LUT. For some vision systems, such as the case of the omni-directional vision system that we will present in more detail in Section 5.1, LUTs can be used for the extraction and storage of the pixel positions that have to be processed, taking into consideration a possible image mask. We identified another use of a LUT in the conversion of images, from one color space to another, as is the case of the conversion of Bayer images into grayscale images.

4.2.1 Color Classification LUT

For fast color classification, color classes are defined through the use of a LUT. The images can be acquired in the RGB, YUV or raw Bayer format and they are converted to an index image (image of labels) using an appropriate LUT for each one of the three possibilities (Fig. 4.8).

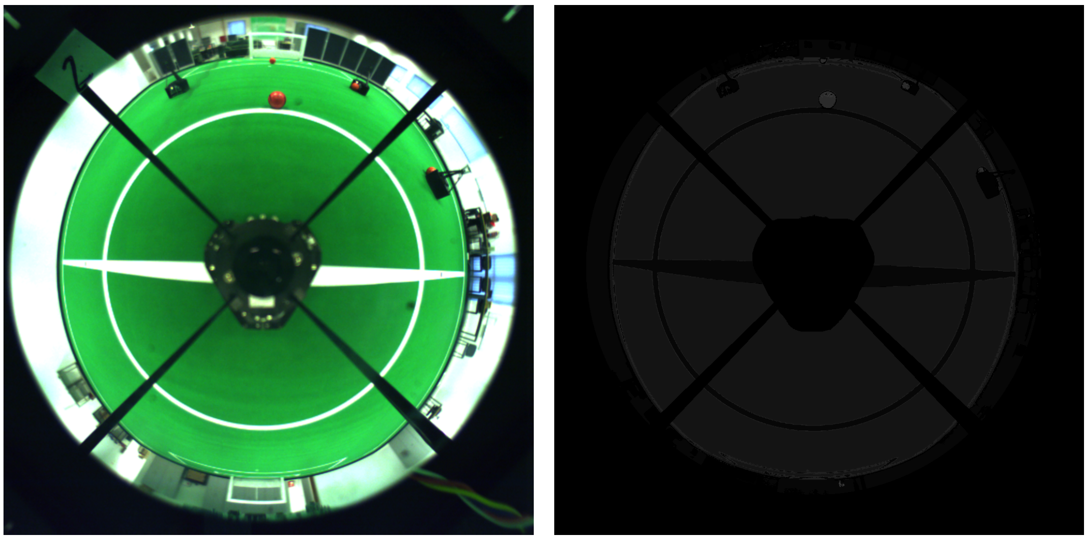


Figure 4.8: On the left, an original RGB image. On the right, the grayscale image of labels.

The table consists of 16,777,216 entries (2^{24}, 8 bits for R, 8 bits for G and 8 bits for B) with one byte each. The table size is the same for the other two possibilities (YUV or raw data), but the meaning and position of the color components in the image changes. For example, considering a RGB image, the index of the table to be chosen for a specific triplet is obtained as

$$idx = R << 16 | G << 8 | B,$$

being the $<<$ a bitwise shift to the left operation and $|$ the bitwise OR. The raw data (Bayer pattern) is also RGB information, however, the position where the R, G and B information are picked changes. These positions are also calculated only once by the LUT module in order to obtain a faster access during color classification.

The table entries can contain as many bits as the number of colors needed in a specific applications (8, 16, 32 - depending on the data type chosen for their manipulation - char, short, int, etc.). In the case of soccer robot, there are no more than 8 colors of interest. Each bit in the table entries expresses if one of the colors of interest (white, green, blue, yellow, orange, red, blue sky, black, gray - no color) is within the corresponding class or not. 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. It is possible to perform image sub-sampling in this stage in systems with limited processing capabilities in order to reduce even more the processing time. If an image mask exists, color classification is only applied to the valid pixels defined by the mask (Fig. 4.9).

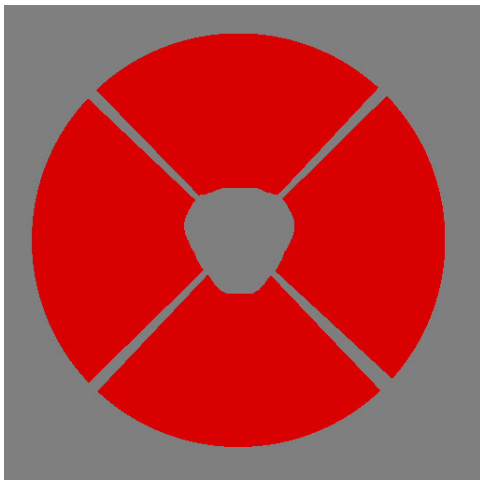


Figure 4.9: Illustration of an image mask used with the omni-directional vision system. In this particular case, only the red pixels will be processed at any time.

Algorithm 4 presents the implementation of a LUT that contains only the valid pixel position, both for RGB and Bayer images.

4.2.2 Image Conversion LUT

In digital cameras, a Color Filter Array (CFA) is a mosaic of tiny color filters placed over the pixels of an image sensor to capture color information. They are needed because the typical photosensors detect light intensity with little or no wavelength specificity, and therefore cannot separate color information [55].

Algorithm 4 Creation of a LUT with valid pixel positions, considering an image mask.

```
1: for $j = 0; j < scanlines.size(); j++$ do
2:     $scanline \leftarrow scanlines.getLine(j)$
3:     for $i = 0; i < scanline.size(); i++$ do
4:         $p \leftarrow scanline[i]$
5:         if $mask[p] > 0$ then
6:             $/ * RGB * /$
7:             $validPixels.add(p)$
8:             $validPixels.add(p * 3)$
9:             $validPixels.add(p * 3 + 1)$
10:            $validPixels.add(p * 3 + 2)$
11:            $/ * BAYER * /$
12:            $row \leftarrow p/mask.cols$
13:            $col \leftarrow p\%mask.cols$
14:            if $row\% == 0$ then
15:                if $col\% == 0$ then
16:                    $r \leftarrow p$
17:                    $g \leftarrow p + 1$
18:                    $b \leftarrow p + mask.cols + 1$
19:                else
20:                    $g \leftarrow p$
21:                    $r \leftarrow p - 1$
22:                    $b \leftarrow p + mask.cols$
23:                end if
24:            else
25:                if $col\% == 0$ then
26:                    $g \leftarrow p$
27:                    $r \leftarrow p - mask.cols$
28:                    $b \leftarrow p + 1$
29:                else
30:                    $g \leftarrow p - 1$
31:                    $r \leftarrow p - mask.cols - 1$
32:                    $b \leftarrow p$
33:                end if
34:            end if
35:            $validPixels.add(p)$
36:            $validPixels.add(r)$
37:            $validPixels.add(g)$
38:            $validPixels.add(b)$
39:        end if
40:    end for
41: end for
```

A Bayer filter mosaic is a type of CFA for arranging RGB color filters on a square grid of photosensors. Its particular arrangement is used in most single-chip digital image sensors used in digital cameras, camcorders, and scanners to create a color image. The filter pattern is 50% green, 25% red and 25% blue, usually called BGGR, RGBG, GRGB, RGGB, etc. depending on the position of the filters (Fig. 4.10).

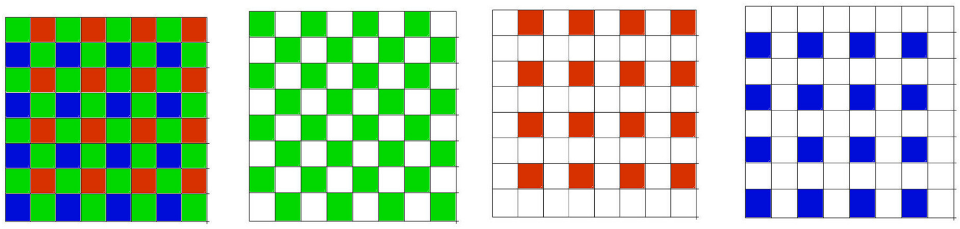


Figure 4.10: Bayer arrangement of color filters.

Most modern digital cameras acquire images using a single image sensor overlaid with a CFA, so demosaicing is part of the processing pipeline required to render these images into a viewable format. However, in most of them it is possible to retrieve images in a raw format allowing the user to demosaic them using software, rather than using the camera's built-in firmware (Fig 4.11).

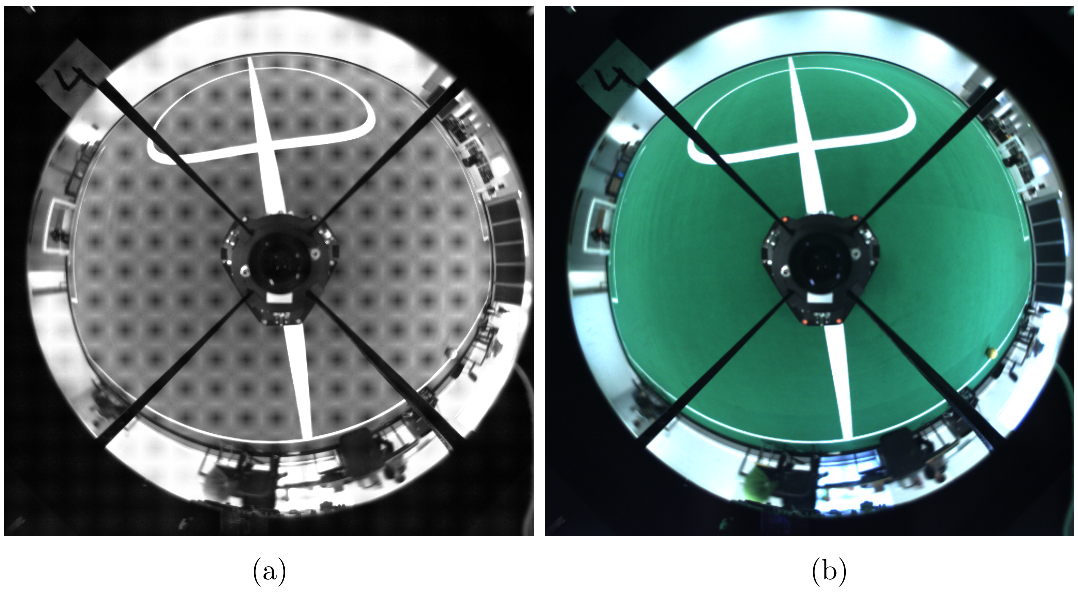


Figure 4.11: In (a) an example of a raw image acquired by the camera (grayscale image containing the RGB information directly from the sensor - Bayer pattern). In (b) the image (a) after the interpolation of the missing information for each pixel in order to obtain a complete RGB image.

In most of the image processing applications, the raw data acquired by the image sensor passes through a demosaicing algorithm in order to obtain a full color image [56]. This is a digital image process used to reconstruct a full color image from the incomplete color samples output from an image sensor overlaid with a CFA. There are several well known algorithms, namely Simple Interpolation [57], Variable Number of Gradients [58], Pixel Grouping [59] or Adaptive homogeneity-directed [60].

As it can be seen in the image on the left, the raw image data is a single channel image where the value of each pixel corresponds to the information about a specific color, depending on the position of the filters in the sensor, as described before. In this Thesis we have explored the use of raw data for object detection and we provide results in Chapter 6. This study is a novel contribution in computer vision which can have a real impact on the design of nowadays modern digital cameras. This initial study shows that the interpolation of a raw image is not a compulsory step for object detection, but rather a feature of the camera that renders a better image visualization for the user. However, this feature can be redundant when the main use of the camera is not the rendering of the images for human inspection or visualization.

Taking into consideration a CFA BGGR, the LUT for the position of the R, G and B components for each pixel is built based on Algorithm 5.

Algorithm 5 Extraction of the R, G and B pixel positions from a Bayer image.

```
1:  for $p = 0; p < cols * rows; p++$ do
2:      $row = p/cols$
3:      $col = p\%cols$
4:      if $row\%2 == 0$ then
5:          if $col\%2 == 0$ then
6:              $R \leftarrow p$
7:              $G \leftarrow p + 1$
8:              $B \leftarrow p + cols + 1$
9:          else
10:             $G \leftarrow p$
11:             $R \leftarrow p - 1$
12:             $G \leftarrow p + cols$
13:         end if
14:     else
15:         if $col\%2 == 0$ then
16:             $G \leftarrow p$
17:             $R \leftarrow p - cols$
18:             $B \leftarrow p + 1$
19:         else
20:             $G \leftarrow p - 1$
21:             $R \leftarrow p - cols - 1$
22:             $B \leftarrow p$
23:         end if
24:     end if
25: end for
```

4.3 Run Length Encoding

For each scan line, an algorithm of run length encoding (RLE) is applied in order to obtain information about the existence of a specific label of interest in that scan line. To do this, we iterate through its pixels to calculate the number of runs of a specific label and the position where they occur. Moreover, we extended this idea and the user has the option to perform the search for occurrences of other labels, both in a window before and after the occurrence of the desired label. This allows the user to determine both label transitions and labels occurrences using this approach.

For the process of color classification, the label of a pixel is its color. Nevertheless, all the previously described algorithms can be used for other applications, in which labels can have a different meaning. In Fig. 4.12 we provide an example of the RLE analysis for color classification.

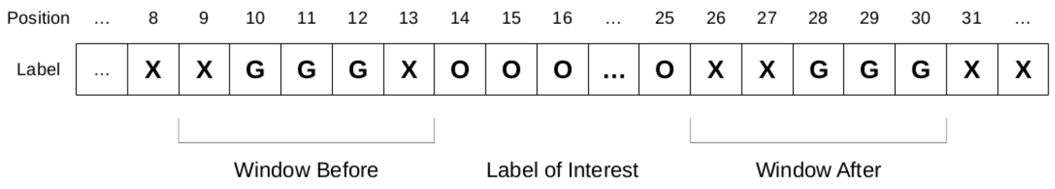

Position ... 8 9 10 11 12 13 14 15 16 ... 25 26 27 28 29 30 31 ...
Label ... X X G G G X O O O ... O X X G G G X X
Window Before
Label of Interest
Window After

Figure 4.12: RLE illustration based on a scan line. Each square represents the label corresponding to the pixel position given by each position of the scan line. In this example we are searching for the occurrence of label `O` and we analyze the neighbor pixels, both before and after in a window of 5 pixels.

Run lengths are fed with the positions defined by the scan lines. Different types of scan lines can be used for the construction of the same run length. In the same application, the user of the algorithm can define different scanning patterns and all scanned positions can be used for the construction of the same run length structure.

When searching for run lengths, the user can specify the label of interest, the label before, the label after, the search window for these last two labels and three thresholds that can be used to determine the valid information. These thresholds are application dependent and should be determined experimentally.

As a result, the user will obtain a list of positions in each scan line and, if needed, for all the scan lines, where a specific label occurs, as well as the amount of pixels in each occurrence. Taking into consideration the example of Fig. 4.12, the list of occurrences regarding the label of interest is the following:

$$list = \{(14, 12, 3, 3)\,; \ldots\}$$

In the previous example, considering a search window of 5 pixels both before and after the occurrence of the label of interest, the first quadruple in the list describes the first detection of the label of interest, the second one the second detection of the color of interest and so on. The values presented in each quadruple represent the following information, by order: position of the first found pixel with the label of interest, number of pixels with that label, number of pixels with the label chosen before the label of interest and number of pixels with the label chosen after the label of interest. In this example, the before and after label was G. The number of pixels found before or/and after the label of interest are optional and can be specified when searching for transitions between colors (Fig. 4.13).

Algorithm 6 depicts the construction of RLEs in their most complex form, considering a search of different colors in windows before and after the color of interest.

4.4 Colored Blobs

A RLE object contains information about the occurrence of a color of interest in a given scan line. The information contained by the RLE object is: the position of the first found pixel of the color of interest, the number of consecutive pixels of the same color and the position of the last found pixel of the color of interest. The middle point between the first and last found pixels of the color of interest can be calculated for each RLE.

When searching for blobs, the first blob is considered to be the first found RLE and the center of the blob is the center of the RLE. Running through all the RLEs that have been found, based on the Euclidean distance between middle points in each run lengths, the color information in adjacent RLEs is merged if the calculated Euclidean distance is between a given threshold. This threshold has to be experimentally calculated. If the calculated distance is not lower than the threshold, the given RLE is considered to be the starting point of a new blob and the process is repeated. Algorithm 7 presents the construction of the color blobs.

When joining another RLE to the Blob, the center of the blob, as well as other measurements such as area, width/height relationship and solidity are being calculated and updated. The blobs are then validated as being objects of interest if they fulfill different application-related criteria (Fig. 4.13).

Algorithm 6 Construction of RLEs.

```
1: for $j = 0; j < scanlines.size(); j++$ do
2:     for $i = 0; i < scanline.size(); i++$ do
3:         if $!image.ptr()[scanline[i]]\&colorOfInterest$ then
4:             $continue$
5:         end if
6:         $pointsColor \leftarrow 0$
7:         $pointsColorAfter \leftarrow 0$
8:         $rle.start \leftarrow scanline[i]$
9:         $rle.startBefore \leftarrow scanline[i]$
10:        while $i < scanline.size() - 1\&\&image.ptr()[scanline[i]]\&colorOfInterest$ do
11:            $pointsColor \leftarrow pointColor + 1$
12:            $i \leftarrow i + 1$
13:        end while
14:        $rle.end \leftarrow scanline[i]$
15:        $rle.endAfter \leftarrow scanline[i]$
16:        $rle.center \leftarrow scanline[i - pointsColor/2]$
17:        if $pointsColor < threshColorOfInterest$ then
18:            $continue$
19:        end if
20:        $pointsColorBefore \leftarrow 0$
21:        for $k = i - pointsColor; k > (i - pointsColor - searchWindow)\&\&k >= 0; k--$ do
22:            if $image.ptr()[scanline[k]]\&colorBefore$ then
23:                $pointsColorBefore \leftarrow pointsColorBefore + 1$
24:            end if
25:        end for
26:        $rle.startBefore \leftarrow scanline[k]$
27:        $pointsColorAfter \leftarrow 0$
28:        for $k = i; k < (i + searchWindow)\&\&k < scanline.size() - 1; k++$ do
29:            if $image.ptr()[scanline[k]]\&colorAfter$ then
30:                $pointsColorAfter \leftarrow pointsColorAfter + 1$
31:            end if
32:        end for
33:        $rle.endAfter = scanline[k]$
34:        if $pointsColorAfter >= threshColorAfter\&\&pointsColorBefore >= threshColorBefore\&\&pointsColor >= threshColorOfInterest$ then
35:            $rle.Idx \leftarrow j$
36:            $rle.lengthColor \leftarrow pointsColor * scanlines.getStep()$
37:            $rle.lengthColorBefore \leftarrow pointsColorBefore * scanlines.getStep()$
38:            $rle.lengthColorAfter = pointsColorAfter * scanlines.getStep()$
39:            $rleStruct.pushBack(rle)$
40:        end if
41:    end for
42: end for
```

Algorithm 7 Construction of color blobs.

```
1:  if $rleStructure.size() == 0$ then
2:      $return$
3:  end if
4:  for $i = 0; i < rleStruct.size(); i++$ do
5:      $idx \leftarrow findBlob(rle, threshold, i)$
6:      if $idx == -1$ then
7:          $BlobInfob(rle, i)$
8:          $blobs.pushBack(b)$
9:      else
10:         $updateBlob(blobs[idx], rle, i, robotCenter$
11:     end if
12: end for
```

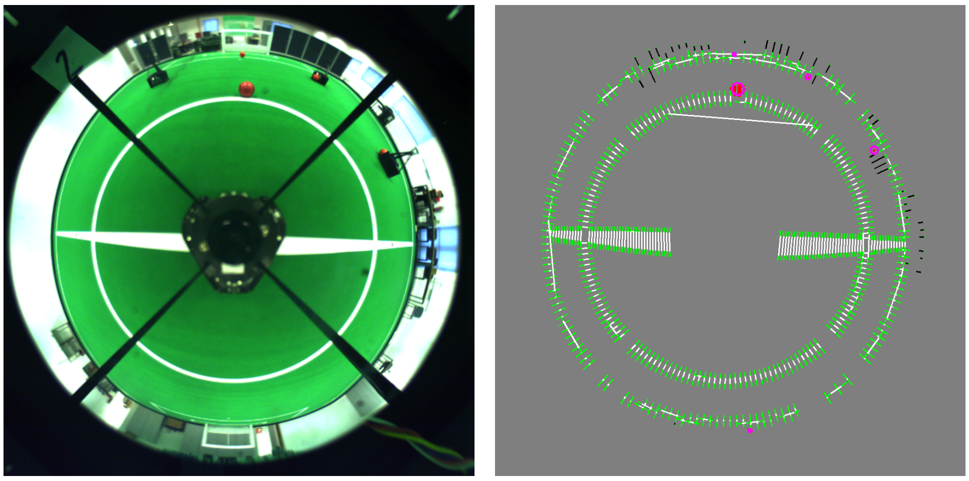


Figure 4.13: On the left, an original image. On the right, the result of RLE algorithm for the line detection, in white and green, and the blobs that have been validated as objects of interest are circled in magenta.

"There's a way to do it better - find it."

Thomas Edison

5 Applications

Based on the algorithms that we have developed, we have designed several functional vision systems that are currently in use by intelligent soccer robots. In this Chapter we detail each of these vision systems.

For a soccer robot, the surrounding world that carries meaningful information is reduced to the soccer field. The objects of interest for such a robot are: the soccer ball, the field lines and goals, and the other robots that can be either team-mates or opponents. The vision system of a soccer robot has to detect the lines of the field in order for the robot to localize itself, the ball so that the game can take place and the other robots in order to avoid them or to cooperate with them (in the case of team-mates).

We have developed a common software architecture for different types of vision systems, that serve the purpose of a soccer robot. We present in this Chapter an omni-directional vision system, a perspective one as well a Kinect-based one. The vision systems that will be detailed next follow the same pipeline, presented in Fig. 5.1.

The pipeline of the object detection procedure is the following:

- After having acquired an image and defined the scanning pattern, the original image is transformed into an image of labels using a LUT previously built. This image of color labels, also designated by index image, will be the basis of all the processing that follows.

- The positions of the scan lines in the index image are searched for colors of interest or transitions between colors of interest and this information will be run length encoded.

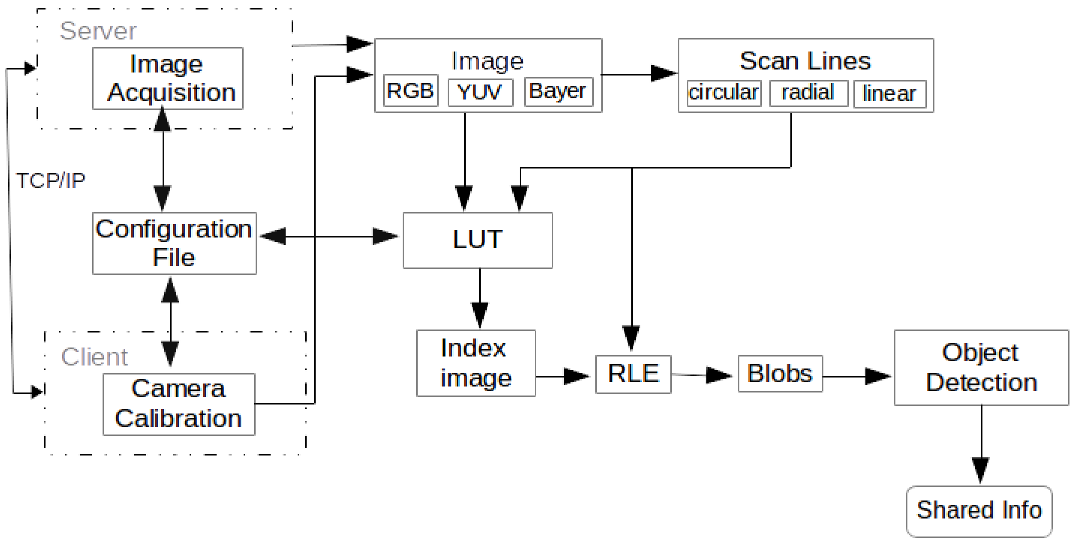


Figure 5.1: Common software pipeline of the vision systems presented in this Chapter.

- Blobs are constructed by merging adjacent RLEs of a color of interest.
- The blob is labeled as object of interest if it passes several validation criteria that are application dependent.
- When an object of interest is validated, its center coordinates are passed to higher-level processes and are shared on a Real-Time Database (RTDB) [61] in the case of the omni-directional and the Kinect-based vision system.

5.1 Omni-directional Vision System

The first vision system that has been developed is an omni-directional one and is used by all robots within the team during a soccer game. An image of the physical set-up of the digital camera and the mirror can be seen in Fig. 5.2. The camera used by the CAMBADA robots is an *IDS UI-5240CP-C-HQ-50i Color CMOS 1/1.8"* Gigabit Ehernet camera [62] and can provide images with a resolution of 1280×1024 pixels at $50fps$. However, a region of interest of 1024×1024 pixels is being acquired and used since the remaining of the image only contains parts of the robot.

Using the modules of the UAVision library, a vision system composed of two different applications has been developed with the purpose of detecting the objects of interest. The two applications are of the type client-server and have been implemented using the TCP/IP communications protocol. The core application, that runs in real time on the processing units of the robots, was implemented as a server that accepts the connection of a calibration tool

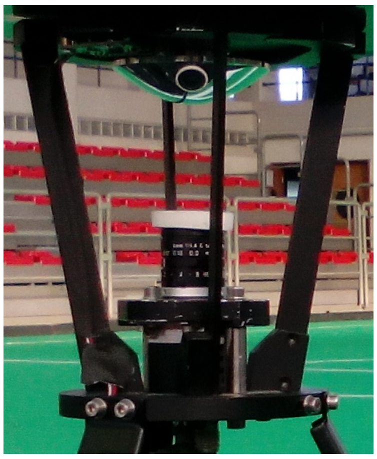


Figure 5.2: Mirror and digital camera disposal of the omni-directional vision system.

client. This client is used for configuring the vision system (color ranges, camera parameters, etc.) and for debug purposes and is presented in more detail in Chapter 7. The main task of the server application is to perform real time color-coded object detection.

The omni-directional vision system follows the generic pipeline in order to detect blobs of the color of interest or color transitions between colors of interest. For the ball detection, in order to decide if a found blob is indeed an object of interest, a mapping function of the size-distance relation has been designed (Fig. 5.3). This function maps the expected size of the soccer ball to the distance from the robot at which it is found. Further validations of the ball rely on a solidity measure and a width-height ratio measure, based on the fact that the ball is expected to be a fairly round blob.

These validations are made taking into consideration the detection of the ball even when it is partially occluded. The problem of ball occlusion is addressed taking into consideration the worst case scenario. Because we are dealing with cooperative multi-agent systems in the games of robotic soccer, even when one of the players does not have a visible ball, chances are that one of his team-mates detects the ball and the ball position is shared among them. The situation in which the ball is not visible for one or more of the robotic agents are due to the occlusion of the ball by other robots. When the ball is only partially occluded, its detection is still possible considering the restraints we can impose on the shape and features of the colored blobs. Looking at Fig. 5.3, we can observe the real ball sizes acquired at different distances. In order to guarantee the detection of occluded balls, we consider a range of $[0.5 - 1.5]$ of the calculated function.

For the detection of the field lines, we are interested in finding color transitions of the type green-white-green. A white line of the field is always surrounded by some green carpet. This logical observation narrows down to 0 the probability of considering line candidates outside of the soccer field. For each color transition, experimental thresholds of green pixels have been determined both before and after the encounter of the color of interest, in this case white.

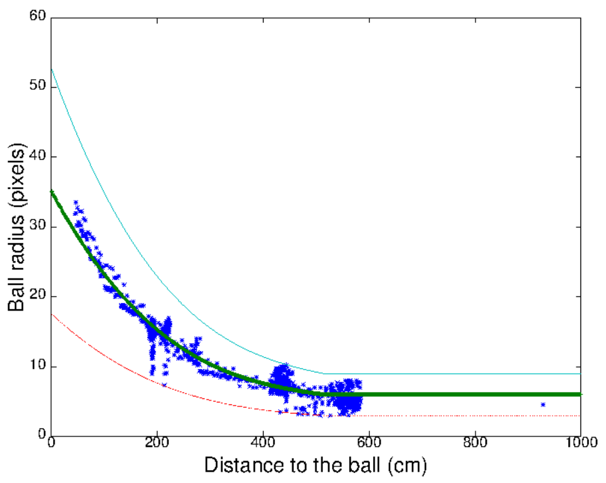


Figure 5.3: A function describing the radius (in pixels) of the object of interest (in this case, an orange soccer ball) as a function of the distance (in centimeters) between the robot and the object. The blue marks represent the measures obtained, the green line the fitted function and the cyan and red line the upper and lower bounds considered for validation.

Thresholds of 2 green pixels before and after the encounter of at least 2 white pixels on the radial scan lines and 8 white pixels on the circular ones are used in order to validate a field line.

Searching for isolated field lines is different than searching for lines that are close to another object. As seen in Fig. 5.4, many times there are robots close to a line. In these situations, since we detect lines based on color transitions, the search windows before and after the color of interest have to be adjusted in order to accommodate a possible noise introduced by the other objects. Also, the length of a line close to a robot is much bigger than the length of a line found at $6m$, for example. These challenges can be overcome by choosing proper search windows for the task. However, the processing time is directly proportional to the size of these windows. When a field line is found, the middle point of the white segment is calculated and passed to a higher level process that is responsible for the integration of this information.

Obstacles are detected based on the search for the black color. For each RLE that contains more than 4 black pixels, the lower end of the RLE (in terms of (x, y) coordinates) is calculated and passed to the same higher level integration process.

The following scan lines configurations are used for the detection of the ball, obstacles and field lines by the omni-directional vision system:

- 1440 radial scan lines disposed on 4 levels, for ball detection.
- 360 radial scan lines disposed on 1 level, for obstacles and field lines detection.
- 90 circular scan lines, for field lines detection.

In this application circular scan lines are very important for the detection of the objects of interest, mainly field lines. Circular scan lines are used for the detection of all field lines that intersect the center of the robot. Visual examples of the detected objects in an image acquired by the vision system are presented in Fig. 5.4 and Fig. 5.5. The objects of interest (balls, lines and obstacles) are correctly detected even when they are far from the robot.

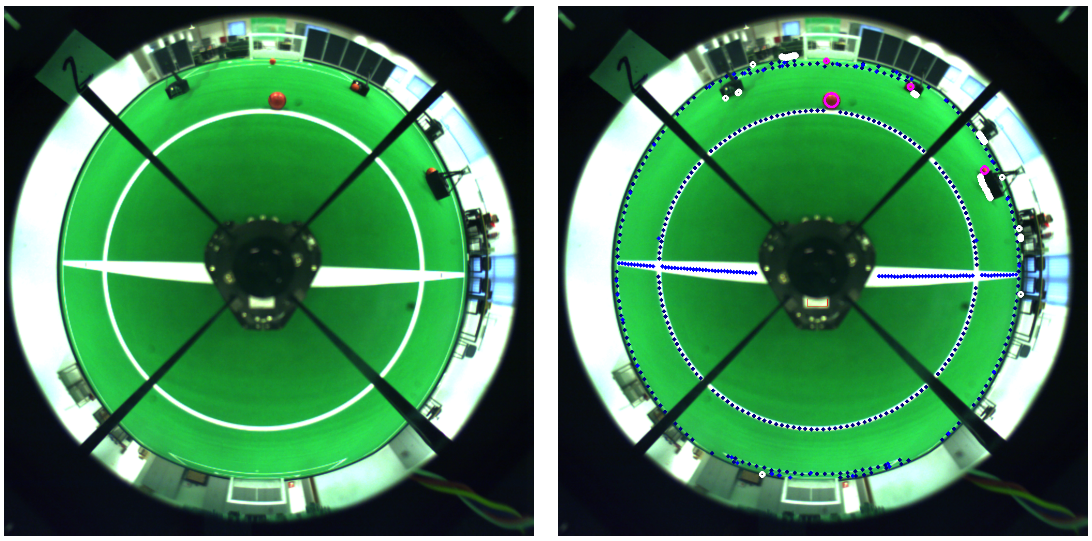


Figure 5.4: On the left, an image acquired by the omni-directional vision system. On the right, the result of the color-coded object detection. The blue circles mark the white lines, the white circles mark the black obstacles and the magenta circles mark the orange blobs that passed the validation thresholds.

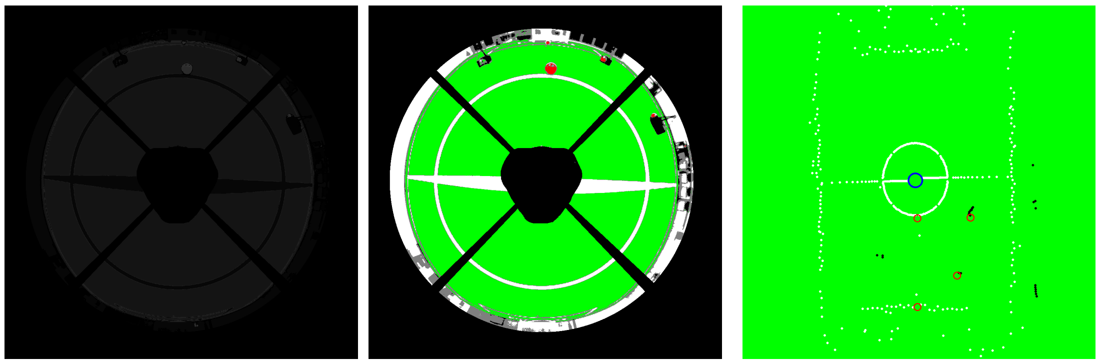


Figure 5.5: On the left, the index image in which all of the colors of interest are labeled. In the middle, the color classified image and on the right, the surrounding world from the perspective of the robot.

Being a highly configurable approach, the color object detection that we implemented can be used for the detection of different other objects of interest that have a given color. Not only the scanning pattern can be customized so that it fulfills the requirements of a given application, but the blob segmentation algorithm can be applied to any colored object. In Fig. 5.6 we present results of the use of these algorithms for the detection of body markers.

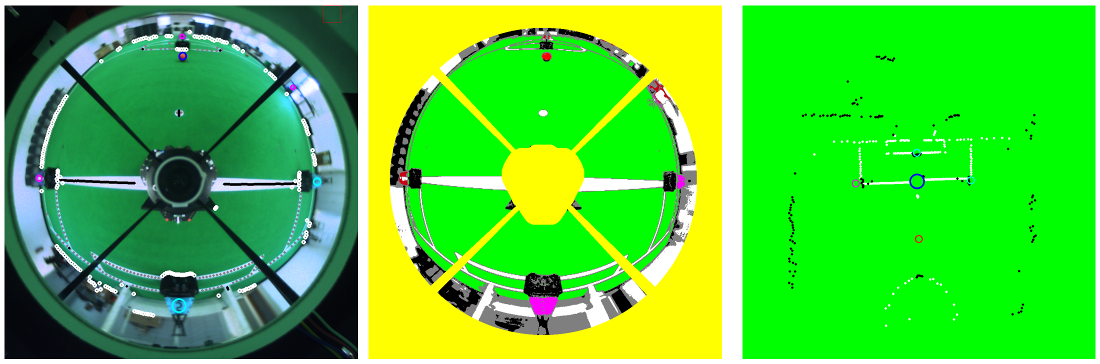


Figure 5.6: On the left, the original image in which all the detected objects are marked. The ball is marked by the blue circle, the obstacles are marked with a circle of the same color as the obstacle itself and the field lines are marked in black. In the middle, the color classified image and on the right, the surrounding world from the perspective of the robot.

Several game scenarios have been tested in our indoor laboratory. In Fig. 5.7 we present a graphic with the result of the ball detection when the ball is stopped in a given position (the central point of the field, in this case) while the robot is moving. The graphic shows a consistent ball detection while the robot is performing a tour around the field. The field lines are also properly detected, as the result of this is the correct localization of the robot in all the experiments.

The second scenario that has been tested is illustrated in Fig. 5.8. The robot is stopped on the middle line of the field and the ball is sent across the field. This graph shows that the ball detection is accurate even when the ball is found at a distance of $9m$ away from the robot. The omni-directional vision system is able to detect the ball up to distances of $11m$, but the $9m$ threshold is imposed by the integration process as a matter of caution.

Finally, in Fig. 5.9 both the robot and the ball are moving. The robot is making a tour around the soccer field, while the ball is being sent across the field.

In all these experiments, no false positives were observed and the ball has been detected in more than 90% of the frames. The times when the ball was not detected were due to the fact that it was hidden by the bars that hold the mirror of the omni-directional vision system. The measurements of the correct detections have been obtained by direct comparison to manually annotated images.

The processing time taken by the omni-directional vision system is in average, of $5.4ms$ for images of 1024×1024 pixels. We consider this to be an important contribution for this

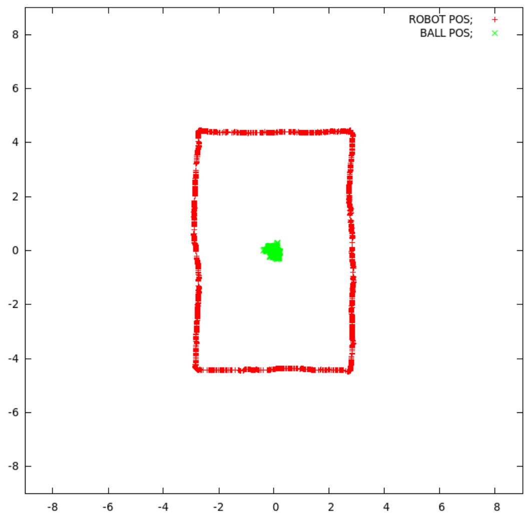


Figure 5.7: Ball detection results when the ball is stopped and the robot is in a tour around the field.

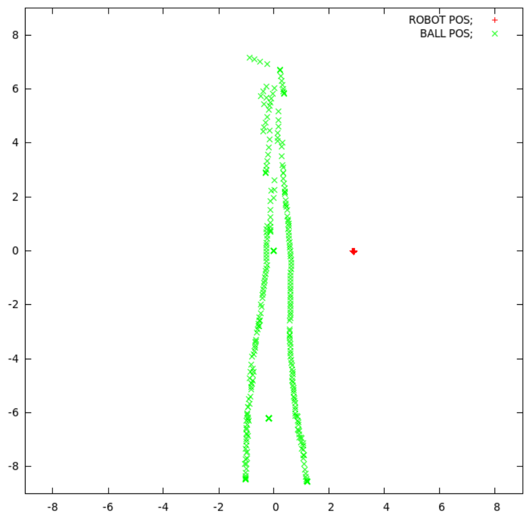


Figure 5.8: Ball detection results when the robot is stopped and the ball is sent across the field.

field of research since, until now we have not found in the literature documented real-time vision systems that can present this performance. Table 5.1 shows the processing time taken by each one of the tasks involved in the object detection pipeline, as well as the average total time taken by this system. The most time consuming tasks are the conversion of the original image into the image of labels and the run length encoding of the color information, each of them with an average processing time of $2ms$. After having constructed a solid run length

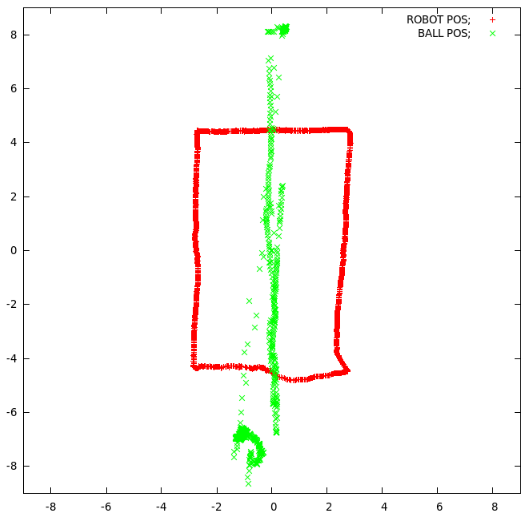


Figure 5.9: Ball detection results when both the robot and the ball are moving.

structure, the blob formation and validation tasks are straightforward.

Operation	Time (ms)
Acquisition	0.5
Conversion to idxImage	2
RLE	2
Blob creation	0.02
Blob validation	0.01
Total	**5.4**

Table 5.1: Average processing times measured for the omni-directional vision system.

5.2 Perspective Vision System

Another vision system that we have designed is a perspective one that can be used for the detection of the same objects of interest. The perspective camera used is a Point Grey Zebra2 Ethernet camera[1]. We present results obtained with the perspective vision system based on the use of the same software architecture as presented before. The images acquired from this camera have been used at a resolution of 1600×1200 pixels.

The perspective vision system is intended as a ground truth validation system that could be used during real robotic soccer games. The ground truth system is illustrated in Fig. 5.10

[1]https://www.ptgrey.com/zebra2-gige-vision-poe-hd-sdi-cameras

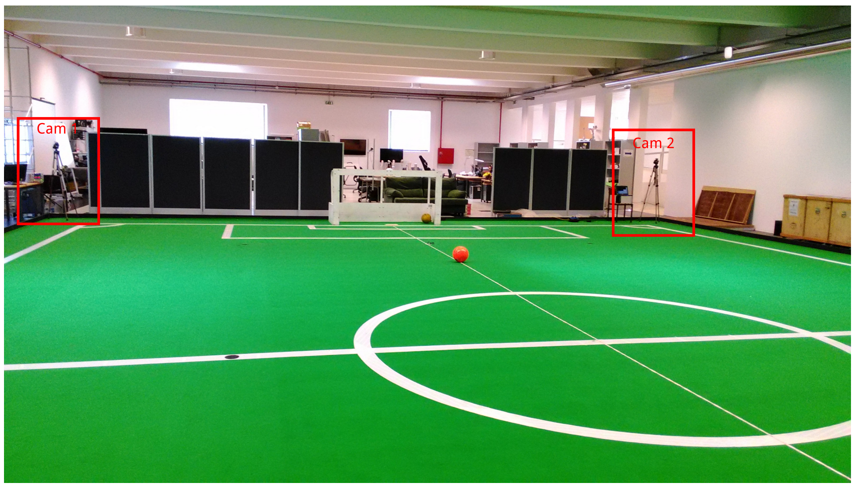


Figure 5.10: Illustration of the soccer field and the placement of the two cameras.

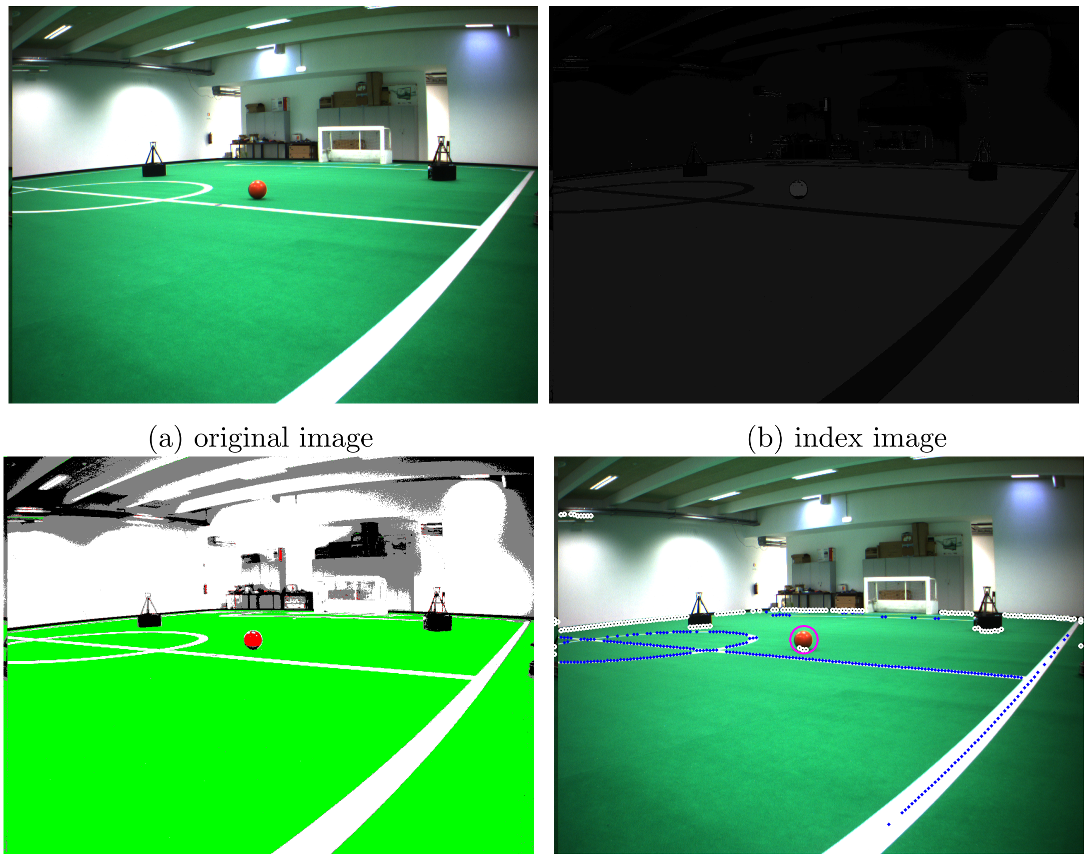
So far there is no ground truth validation system in use by the teams participating in the MSL. With this is mind, this system is intended as another contribution for this research area. A preliminary ground truth system for validating the positions of the objects of interest has been developed in a dissertation within out research group based on the perspective vision system that we present in this Chapter. This system is used for the detection of the soccer ball by one or multiple cameras strategically placed on the soccer field. The ball coordinates obtained from different cameras are interpolated in order to find the 3D ball position on the field.

Figure 5.11 shows some detection results obtained with the perspective vision system.

A graphical tool for visualizing the position of the cameras on the soccer field has been designed. Moreover, this tool supports the visualization of the projection of the ball direction vectors, for different cameras, as well as the ball position on the field (Fig. 5.12). Initial experiments using up to four cameras have been conducted.

The ball detection algorithm follows the approach presented in Fig. 5.1, in which blobs of the color of the ball are detected. The ball is validated based on a series of measurements such as: roundness, size and width/height relation. In order to find the 3D position of the ball, the ball is detected in an image acquired by the first camera and its center is calculated. This procedure is repeated for the frame acquired by the second camera. For each ball center, a vector is projected from the optical center towards the center of the ball. In a triangulation of two vectors, due to errors in the ball position, these vectors might not intersect. To overcome this, instead of calculating the intersection between two vectors, the closest point between them is calculated.

Figure 5.13 (a) shows the detection of the ball in an image and Figure 5.13 (b) shows the projection of a 3D vector towards the pixel corresponding to the center of the ball. The intersection of the vector with the plan of the field does not correspond to the real coordinates


(a) original image (b) index image

(c) color classified image (d) annotated image

Figure 5.11: Results obtained using a perspective camera.

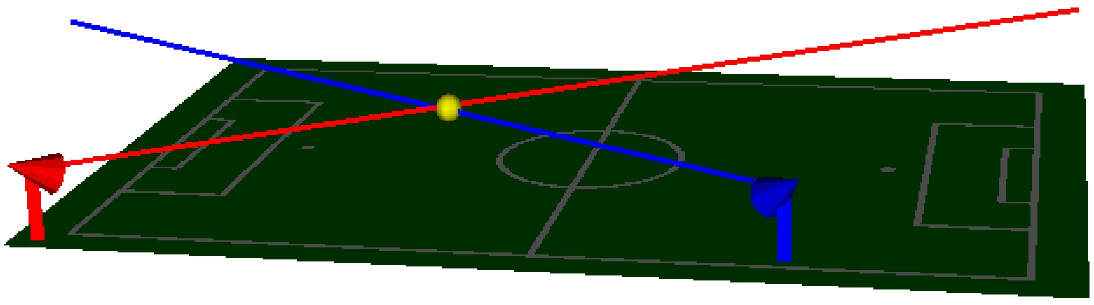


Figure 5.12: Visualization tool for the positioning of multiple cameras on the soccer field.

of the ball and this is due to the height of the ball, which is higher than the plan of the floor. To compensate this, more cameras should be used.

Figure 5.14 shows the projection of two vectors for the two different cameras used in this first test. The closest point between them defines the 3D coordinates of the ball.

The perspective vision system can detect the ball up to $18m$, taking an average time of

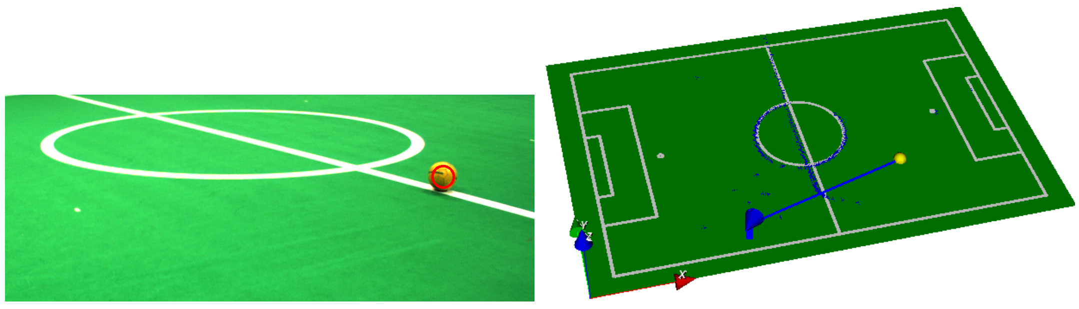

(a) (b)

Figure 5.13: (a) Ball detection using the library UAVision. (b) Projection of a vector, from the optical center towards the center of the ball.

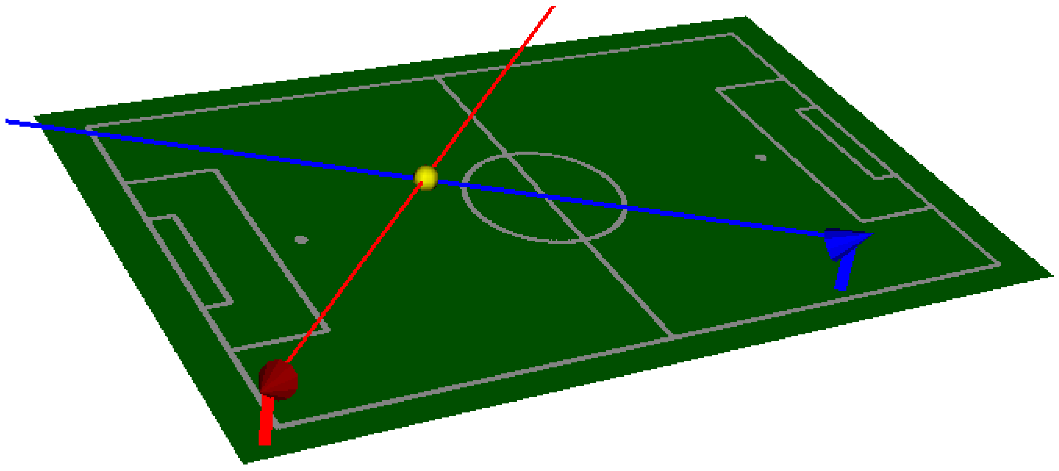


Figure 5.14: Projection of the two vectors towards the center of the ball; the intersection of these vectors defines the 3D center of the ball.

$3ms$. Table 5.2 details the processing time taken by each of the tasks. The total processing time for this system is very low, with all of its tasks working under $1ms$. For this system, horizontal and vertical scan lines have been used. The configuration of the horizontal scan lines had a 4 pixel inter-sparsity and a 4 pixel intra-sparsity. The vertical scan lines had an 8 pixel inter-sparsity and a 4 pixel intra-sparsity.

5.3 Kinect-based Vision System

Detection of aerial objects is a difficult problem to tackle given the dynamics and speed of a flying object. The problem is even more difficult when considering a non-controlled environment, when the vision system is located on a moving platform. Omni-directional vision systems only detect the ball when it is on the ground, and thus precise information

Operation	Time (ms)
Acquisition	0.5
Conversion to idxImage	1
RLE	0.8
Blob creation	0.01
Blob validation	0.01
Total	**3.1**

Table 5.2: Average processing times measured for the perspective vision system.

on the ball position is lost, when the ball is in the air. Because the MSL soccer games have evolved a lot in the last years and they have become very competitive, a solution was needed for the detection of the aerial balls by the goalie of the CAMBADA team. Nowadays most of the teams, if not all, focus on shooting the ball through the air towards the goal because this approach is more likely to be successful. We have implemented a novel approach for aerial ball detection using a mixture of color and depth information. For this, we use color information to detect blobs of the ball color and the depth information to filter the image in order to remove color information regarding objects that are found outside the soccer field limits.

A Kinect sensor has been mounted on the goalie of CAMBADA robotic soccer team (Fig. 5.15) and the Kinect-based vision system has been integrated within its architecture. This vision system has been used in the RoboCup 2014 and RoboCup Portuguese Open 2015 competitions and the logs saved by the robots prove that this addition to the regular omni-directional vision system has been successfully used in several game situations, allowing the defense of what could have been several possible goals.

Kinect is a motion sensing input device developed by Microsoft and launched on November 2010. The sensor includes an RGB camera, an infra-red laser projector, a monochrome CMOS sensor, and other components less relevant for our application. The field of view is 57 degrees horizontally, and around 43 degrees vertically. Acquisition of the 3D data from the Kinect is done using a C++ wrapper for libfreenect[2] that transforms depth images into an OpenCV[3] matrix. The images acquired have a resolution of 640×480 pixels.

In its original configuration, Kinect normal working range is from 0.8 meters to 4 meters. However, the sensor provides distance measurements for longer distances while suffering from additional errors and loss of precision, which cause a discretization effect to appear when distance increases. In our experimental results we have reached the conclusion that we can rely on the depth information retrieved from the depth sensor for distances up to $7m$.

[2]`https://github.com/OpenKinect/libfreenect`
[3]`http://docs.opencv.org`

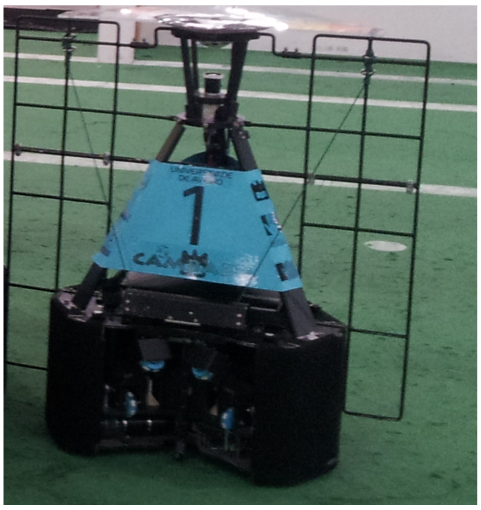


Figure 5.15: The goalie of the robotic soccer team CAMBADA with the Kinect sensor installed on its frontal part.

In Fig. 5.16 we can see an example of the images acquired by a Kinect sensor.

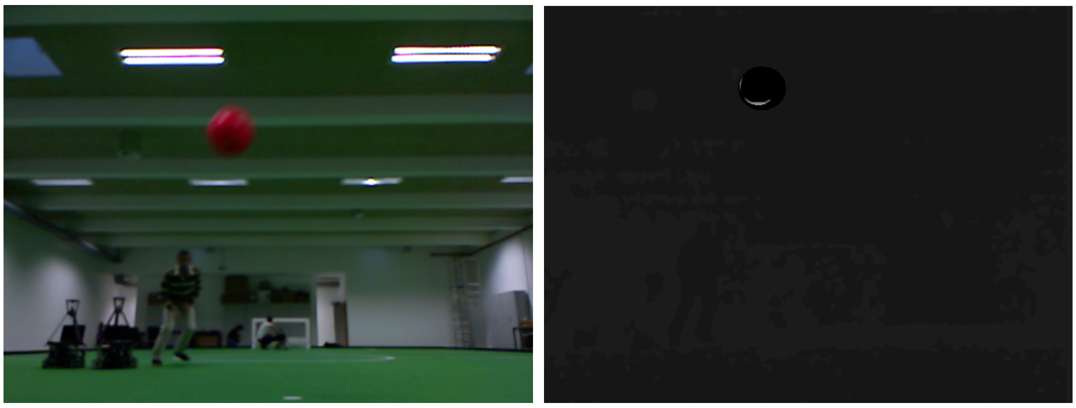


Figure 5.16: Images acquired by a Kinect sensor. On the left, the color image in RGB color space and on the right, the depth image as single channel grayscale image.

The pipeline of the Kinect-based vision system is presented in Fig. 5.17. The pipeline is similar to the general software pipeline that has been illustrated in Fig. 5.1. The additional filtering step is presented next.

5.3.1 Color Filtering Using Depth Information

In this step of the processing pipeline, the depth information from the Kinect sensor is used for discarding the color of the objects that are found farther than a certain distance (in the case of the CAMBADA goalie, $7m$ were considered). This complements the previous step

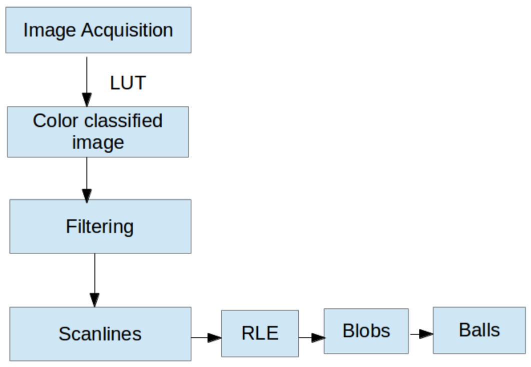


Figure 5.17: Pipeline of the Kinect-based vision system.

by filtering possible objects of the ball color found outside the field. As stated before, this step is applied after the color classification.

First, a calibration between the RGB and depth images provided by the sensor has to be performed. The default calibration parameters available in the ROS package for Kinect calibration[4] have been used for this purpose with some small experimental adjustments.

After having the correct calibration parameters, the distance in meters according to the pixel values of the depth image has been obtained. Two look-up tables relating the pixels of the depth image and the RGB image are created every cycle based on the current depth information. This allows filtering the RGB information as stated before. In Fig. 5.18 we present an example of the filtering step. As shown in the images, after filtering, the classified pixels that are farther than $7m$ are discarded. There are regions close to the limits of the image that are not filtered due to the fact that there is no correspondence between the depth and RGB pixels for those positions. These regions are not considered for processing regarding since we configured the limits of the scan lines to be inside the filtered area.

Using the distortion coefficients of the RGB image and the intrinsic parameters of the depth camera, information from the depth camera can be projected to a metric 3D space. Having a valid ball, the next step is to calculate its 3D position relative to the robot, as described next. This step is necessary in order to estimate the direction towards which the goalie should move in order to defend a goal. This is done by establishing a relation between the coordinates of the center of the blob in the RGB image and in the depth image.

To obtain the correct 3D position of the ball relatively to the center of the robot, the calibration of the position of the Kinect sensor relative to its position on the robot has to be calculated. The coordinates of the detected ball on the field coordinate frame can then be obtained by applying the transformation from Kinect to robot system coordinates and then from robot to world coordinates. In the developed vision system, for the goalkeeper, the same calibration application as presented in [63] has been employed. This application

[4]`http://wiki.ros.org/kinect_camera`

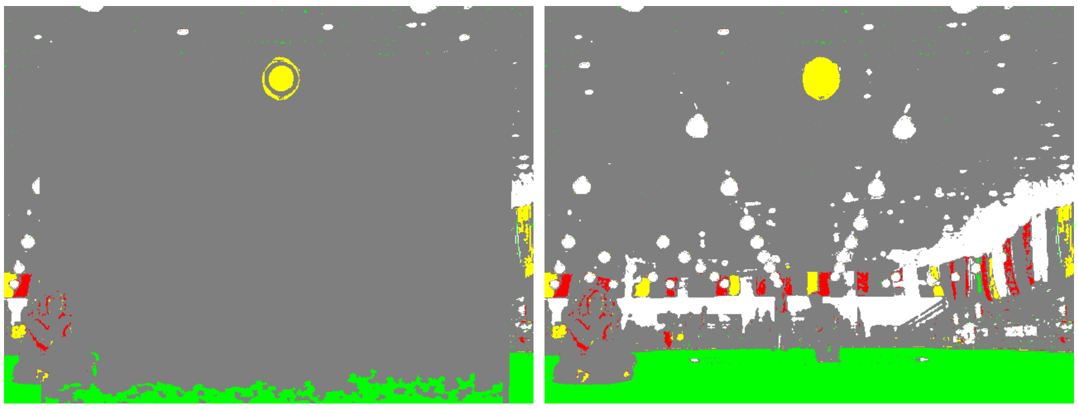
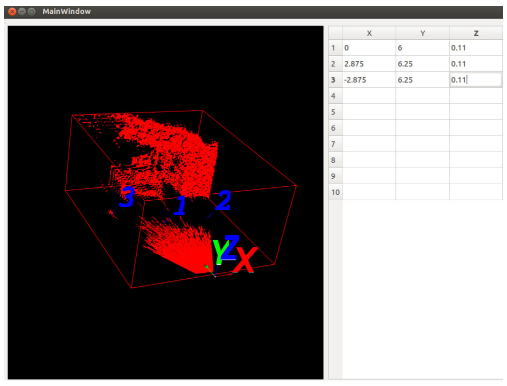


Figure 5.18: On the left, the obtained color classified image after filtering. On the right, the original classified image. These images were taken during RoboCup 2014.

(see Fig. 5.19) acquires on demand an image from Kinect and allows the user to pick some points on the 3D cloud of points. The chosen points correspond to points in the world whose relative position to the robot are known by the user. The software then evaluates the rigid body transform between the 2 coordinates systems corresponding to the position of the Kinect and its orientation relatively to the origin of the robot coordinates system.



	X	Y	Z
1	0	6	0.11
2	2.875	6.25	0.11
3	-2.875	6.25	0.11
4			
5			
6			
7			
8			
9			
10			

Figure 5.19: Application for the calibration of the position of the Kinect with 3 reference points on the Kinect cloud and the corresponding coordinates in robot coordinates system [63].

The configuration of the scan lines used in this vision system is the following: horizontal and vertical scan lines with a 2 pixel intra-sparsity and 2 pixel inter-sparsity. The total processing time taken is, in average, of $20ms$. Partial processing times are presented in Table 5.3.

Operation	Time (ms)
Acquisition	1
Color classification	2
Filtering	16.5
RLE	0.2
Blob creation	0.04
Blob validation	0.02
Total	**20**

Table 5.3: Average processing times measured for the aerial ball detection.

The most time consuming task is the one of image filtering. This is done to the fact that using a pre-defined LUT is not possible in this particular task, given that the $3D(x, y, z)$ coordinates of the current pixel depend on the depth information and this information is not static.

After having analyzed the last two games of the RoboCup 2014 competition, the semi-final where the CAMBADA team lost the game by $2 - 1$ and the 3rd/4th place game where the CAMBADA team won by $4 - 0$, the ball was detected by the proposed algorithm in 110 images, on a total of 16 game situations where the ball was kicked by the opponent team towards the goalkeeper and traveled by air (see some game situations on Fig. 5.20). As the game results shows, our team only suffered two goals, being one of them achieved by the ball moving on the ground and after being kicked by one of our own robots.

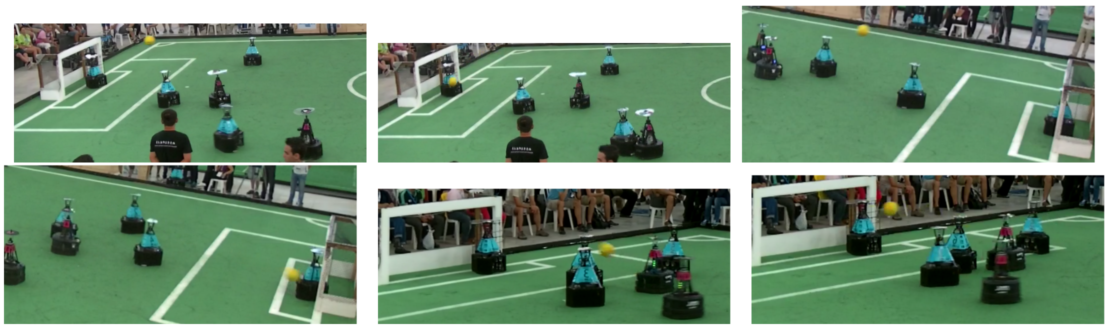


Figure 5.20: Three game situations during RoboCup where the goalie correctly defended the ball. We can see the moment after the kick and on the right the goalkeeper correctly positioned on the goal line when the ball reaches it.

"Science, at bottom, is really anti-intellectual. It always distrusts pure reason and demands the production of the objective fact."

Henry Louis Mencken

6 Performance Evaluation

In this Chapter we present further results on the use of the vision systems presented in Chapter 5. These results focus on the evaluation of the performance of the vision systems and on the comparison of our methods against other popular algorithms for color object detection.

The code is written in C++ and the main processing unit is an Intel Core i5-3340M CPU @ 2.70GHz 4 processor, running Linux (distribution Ubuntu 14.04. LTS Thrusty Tahr). This computing unit is available on the robots and has been used for the evaluation of the performance of the vision systems. Moreover, we complement these results with the use of an IGEPv2 board, a low power single-board computer[1].

We provide results with the IGEPv2 board in order to simulate a processing unit of lower capabilities, such as the ones of autonomous robots that play soccer in other RoboCup leagues. For example, an SPL robot is equipped with a single-core Intel Atom 512K Cache, 1.60 GHz, 533 MHz FSB CPU, 1GB RAM memory and an 8GB hard drive. The processing characteristics of a DARWIN robot, which is currently on of the most popular choices among teams participating in the RoboCup HL, are very similar to the ones of the NAO robots. In fact, the CPU is the same, however the HDD is a 4GB SSD. The IGEPv2 board has a 1 GHz (DM3730) ARMv7 Cortex-A8 CPU, 512MB of LPDDR SDRAM 200 MHz and an 8GB micro-SD card.

Because we do not have access to any of the robotic platforms used in RoboCup SPL or HL, we have used the IGEPv2 board to simulate their computational unit and provide results that prove the contribution of our approach for the area of embedded systems and robotics.

[1]https://www.isee.biz/products/igep-processor-boards/igepv2-dm3730

In addition, the board that we are using is an example of an embedded system that can be used in different other real-time applications.

The work that we presented in [64] and [65] were the first attempts towards building a stable and modular vision system for robots that have low processing capabilities. The approaches that we have presented in those articles have been improved and matured during the period of this PhD. and resulted in the algorithms that we presented in Chapter 4. The research project involving humanoid soccer robots was discontinued within our research group. Therefore, we present results of the object detection algorithms running on an IGEPv2 board, as a simulator of the aforementioned low processing capabilities systems.

In the implementation of the vision systems presented in Chapter 5 we did not use multi-threading. The parallelism of our software is not done for the following reason: the Linux kernel installed on the used computers is configured to use a $10ms$ scheduling interval. At this time interval the kernel determines whether the current process continues on the same core, or performs a context switch to another core. Since none of the modules of our vision system requires a processing time higher than $10ms$, except for the filtering step based on depth information, using threads would not improve its performance. In slower systems or in applications that have to deal with higher resolution images, parallelism can be an option and several tasks of the vision system can work in different threads.

The Technology Readiness Level [10] of the proposed vision systems, representing their technological maturity has reached a maximum value. The results that will be presented next support this claim.

6.1 Colored Object Detection Performance Evaluation

In this Section we present detailed experimental results of color object detection in the context of robotic soccer, based on the use of scan lines for color clustering. The experimental results were obtained both for the omni-directional and the perspective vision systems presented in Chapter 5. These experimental results focus on time measurement and on the efficiency of the object detection algorithm. We explored the parameters available on the scan lines module in order to understand its effect on the processing time and on the detection efficiency. Moreover, we present results with different image resolutions and we discuss the compromise between object detection rate and image resolution. We compare the results obtained using our approach with other well-known algorithms that will be presented next.

6.1.1 Scan Lines Performance Evaluation

In this Subsection we present comparative results on the performance of the omni-directional and perspective vision systems when using different configurations of the scan lines.

Figures 6.1 and 6.3 show examples of images captured by the two types of vision systems. Similar images have been used for generating the following results. For the omni-directional vision system, a total of 5471 frames with resolution of 1024×1024 pixels have been used, while for the perspective one the total number of used frames was 312 with a resolution of 1600×1200 pixels.

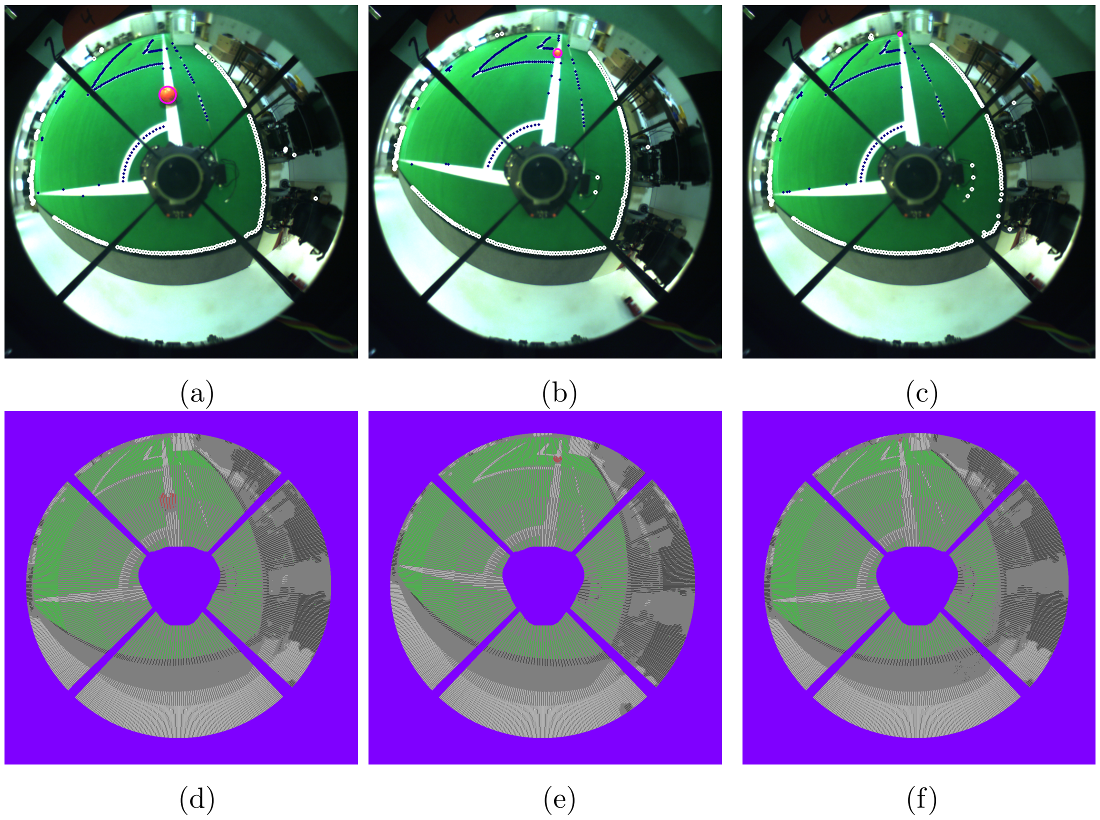


Figure 6.1: (a) Ball at 1m from the robot. (b) Ball at 4m from the robot. (c) Ball at 10m from the robot. (d), (e), (f) The results of color segmentation based on radial scan lines of the images presented in (a), (b) and (c), respectively.

Table 6.1 shows the influence of choosing a different number of levels for the radial scan lines, on an 1024×2014 resolution image. The possibility of creating more than one level of scan lines has an impact on the overall efficiency of the system, since the number of visited pixels as well as repeated visits can be adjusted according to the requirements of a given application.

Figure 6.1 shows the results of the image segmentation and clustering algorithm, followed by object detection and validation for the omni-directional vision system. The images show a correct ball detection, at distances varying from $1m$ to $10m$. Moreover, we can observe a correct detection of the white lines at different distances, needed for the robot localization, as well as a correct detection of the obstacles, used by the robots for cooperation issues (team mates and opponent detection) and for obstacles avoidance. In this experiment, a total

# scan lines	# levels	% visited pixels	% repeated pixels
2880	4	38.57	0.11
2880	3	53.07	1.48
1440	4	19.34	0
1440	3	27.28	0
720	4	9.66	0
720	3	13.64	0

Table 6.1: Effect of the scan lines levels on the number of visited pixels and the number pixels that are processed more than once (repeated pixels).

---

number of 720 radial scan lines, disposed on 4 layers, have been constructed and used, with 1 pixel sparsity within the scan line and 1 pixel inter-sparsity.

The ball has been placed at distances from $1m$ up to $12m$ from the robot. For each placement of the ball, the robot would do a complete rotation around its axis and the result of the ball detection precision and accuracy are presented in Fig. 6.2. Several numbers of radial scan lines have been tested: 360, 720 and 1440 scan lines. For 720 scan lines a sparsity of 2 and respectively 4 pixels within the scan line have been tested.

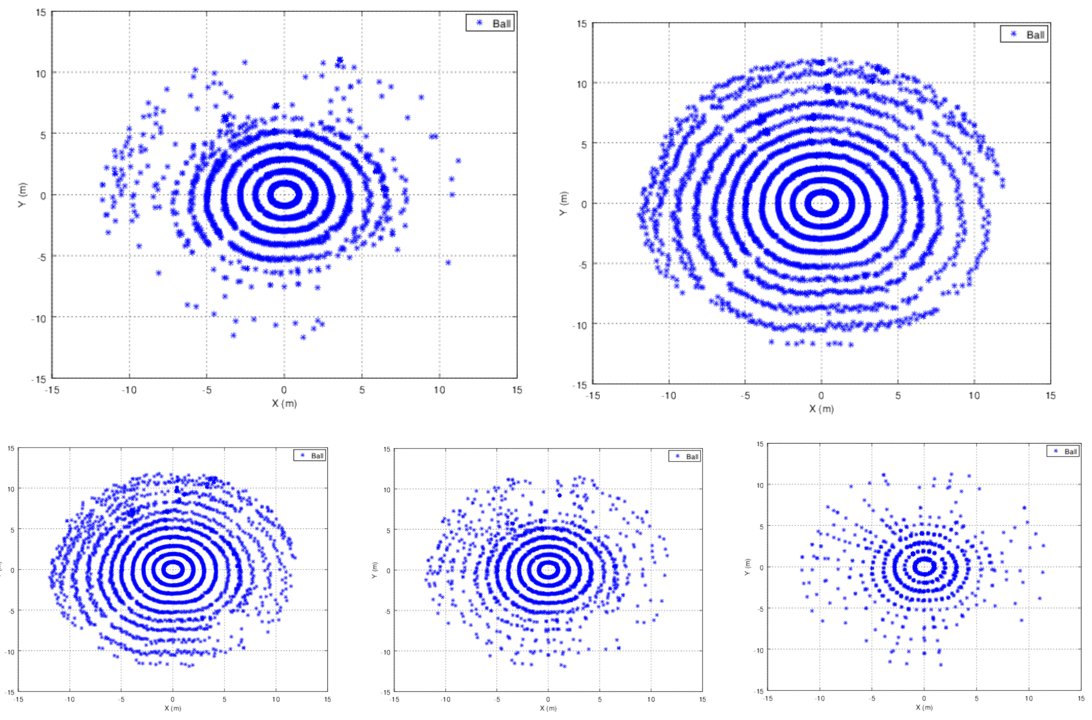


Figure 6.2: Ball detection results with the ball placed at distances ranging from $1m$ to $12m$ from the robot and with the robot rotating around its axis for different radial scan lines configurations. On the upper row, 360 scan lines on the left and 1440 scan lines on the right. On the lower row, 720 scan lines on the left, 720 scan lines with a 2 pixel intra-scan line sparsity and 720 scan lines with a 4 pixel intra-scan line sparsity.

Table 6.2 presents the detection accuracy for the five chosen configurations. $720 - 2$ refers to 720 scan lines and sparsity of 2 pixels within the scan line, while $720 - 4$ represents the configuration of 720 scan lines and sparsity of 4 pixels within the scan line.

Configuration	360	720	1440	720-2	720-4
Percentage	62.62%	95.06%	99.08%	65.37%	29.49%

Table 6.2: Ball detection percentages of the omni-directional vision system for five different radial scans configurations.

The detection percentages when using 720 scan lines are considerably higher than the ones achieved when using only 360 scan lines and slightly worse than the results obtained with 1440 scan lines. The sparsity parameter has a beneficial influence on the processing time, as it will be shown next, but the price that has to be paid is a smaller detection rate.

A percentage of 62.62% in the detection rate when using 360 scan lines can be considered a fair price to be paid in order to further reduce the processing time. This is an important issue, mainly when developing embedded systems with limited computational power. A more detailed analysis of the detection rate shows that at distances starting from $6m$, only a quarter of the balls are detected (Table 6.3).

Configuration	360	720	1440
Percentage	23.76%	90.69%	98.27%

Table 6.3: Percentages of correct ball detection for the three chosen scan lines configurations for balls found at 6m or more from the robot.

The abrupt decreasing trend in ball detection for 360 scan lines is even more evident for distances equal or greater than $9m$ (Table 6.4). For balls found at these distances, the detection rate is only 14.4%. On the other hand, while the decreasing trend can be observed for the other two configurations, the detection rates are still close to 90% for 720 scan lines and 100% for 1440 scan lines.

Configuration	360	720	1440
Percentage	14.40%	88.70%	97.11%

Table 6.4: Percentages of correct ball detection for the three chosen scan lines configurations for balls found at $9m$ or more from the robot.

Processing times for the five scan lines configurations for the omni-directional vision system are presented in Table 6.5. The maximum average processing time is measured when using 1440 radial scan lines and it is under $7ms$ for an image of 1024×1024 pixels. If the digital camera would allow it, the vision software could work at a frame rate higher than

$100fps$. By reducing the number of radial scans to half (720) with a 4 pixel intra-scan line sparsity, the processing time amounts to only $1ms$.

Configuration	360	720	1440	720-2	720 - 4
Average	2.13	3.59	6.28	1.97	1.17
Max	3.38	5.46	8.08	3.51	2.78
Min	1.89	3.30	5.89	1.82	1.01

Table 6.5: Processing time results, in ms, for the color object detection with an omni-directional vision system.

Even using a Linux Operating System without a real-time kernel, we measured a deviation of less than $2ms$ of the maximum and minimum processing times with respect to the average processing time. Table 6.6 shows that the most time consuming operation is the conversion of the original image into the image of labels.

Configuration	360	720	1440	720-2	720-4
Conversion to index image	0.98	1.01	2.27	0.68	0.54
RLE	0.87	1.53	2.91	0.81	0.49
Blobs formation	0.05	0.11	0.24	0.04	0.03
Validations	0.02	0.03	0.04	0.02	0.02

Table 6.6: Partial processing times for the omni-directional vision system.

A result of the color object detection provided by the perspective vision system are presented in Fig. 6.3. The images show a correct ball detection, at distances varying from 3 to $18m$. This is an important contribution considering the correct detection of an object with a radius of $12cm$ so far from the camera. Moreover, a correct detection of the white lines at different distances, as well as a correct detection of the obstacles can be noted.

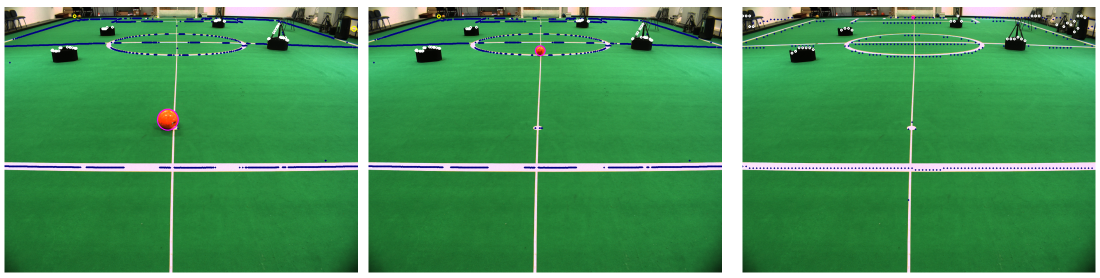
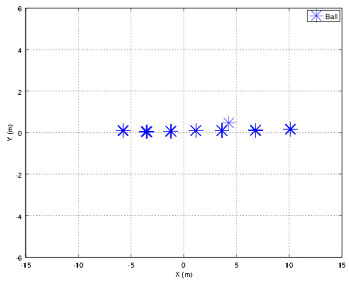
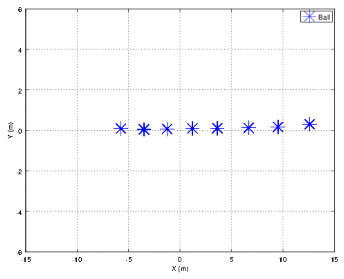


Figure 6.3: Perspective ball detection at different distances: $3m$, $6m$ and $18m$. The correct identification of lines and obstacles can also be noted.

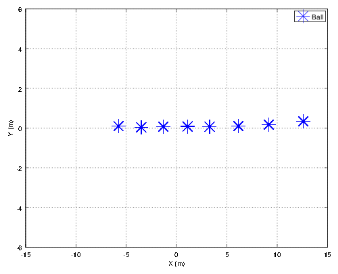
Several configurations for the vertical scan lines used in the perspective vision system have been tested. Figure 6.4 shows results of ball detection accuracy for three different configurations of the vertical scan lines: 2 pixels intra-scan line sparsity and 2 pixels inter-scan line sparsity; 4 pixels intra-scan line sparsity and 4 pixels inter-scan line sparsity and 8 pixels intra-scan line sparsity and 8 pixels inter-scan line sparsity.



(a)



(b)



(c)

Figure 6.4: Ball detection accuracy for the following vertical scan lines configurations: (a) 2 pixels intra-scan line sparsity and 2 pixels inter-scan line sparsity. (b) 4 pixels intra-scan line sparsity and 4 pixels inter-scan line sparsity. (c) 8 pixels intra-scan line sparsity and 8 pixels inter-scan line sparsity.

The most time consuming configuration of the vertical scan lines is the $2-2$ combination, whose total average processing time is $11ms$ (Table 6.7). However, for applications in which a fast performing algorithm has more priority than a high detection rate, an $8-8$ configuration of the scan lines leads to an average total time of $0.64ms$.

Configuration	2-2	2-4	2-8	4-2	4-4	4-8	8-2	8-4	8-8
Average	10.63	3.45	1.94	5.05	1.89	1.05	3.13	1.10	0.76
Max	11	3.8	2.1	5.48	2.30	1.24	3.37	1.50	0.97
Min	10.30	3.26	1.91	5.01	1.87	1.01	2.85	1.09	0.64

Table 6.7: Processing time results of the perspective vision system for different vertical scan lines configurations.

Partial processing times for the perspective vision system are presented in Table 6.8. Like in the omni-directional vision system, the most time consuming operation is the conversion of the original image into the image of labels.

Configuration	2-2	2-4	2-8	4-2	4-4	4-8	8-2	8-4	8-8
Conversion to index image	6.25	1.61	0.86	3.13	0.99	0.48	1.27	0.58	0.34
RLE	3.02	1.37	0.83	1.40	0.69	0.40	1.06	0.36	0.26
Blob formation	1.17	0.29	0.08	0.34	0.06	0.04	0.20	0.34	0.26
Validations	0.07	0.08	0.06	0.06	0.05	0.05	0.05	0.05	0.05

Table 6.8: Partial processing times for the perspective vision system for different vertical scan lines configurations.

In Table 6.9 we present the processing time of the algorithms implemented for the omni and for the perspective vision systems using an IGEPv2 board. For the omni-directional sequences, we have used 720 radial scan lines. This configuration represents the best compromise between object detection accuracy and processing time in the situation of a robotic soccer game. The perspective system is currently used for monitoring purposes with a configuration of vertical scan lines with an 8 pixels intra-scan line sparsity, as well as an 8 pixels inter-scan lines sparsity. These results show that even with a limited processing unit, it is possible to use the proposed approach at more than $15fps$.

Configuration	Persp.-8x8	Omni-720
Average	59.64	87.66
Max	67.99	137.67
Min	59.02	82.03

Table 6.9: Processing time results on a IGEPv2 processing unit.

In Table 6.10 we present the processing time of each one of the modules implemented for the omni-directional and for the perspective vision systems, running on the IGEPv2 board.

Configuration	Persp.-8x8	Omni-720
Conversion to index image	27.03	33.94
RLE	26.03	45.32
Blob formation	2.24	3.92
Validations	0.80	0.75

Table 6.10: Partial processing time for each operation on the IGEPv2 board.

For further comparisons, we tested the same algorithms, using the omni-directional sequences in the IGEPv2 board, considering lower resolutions. The average processing time obtained for a resolution of 512×512 pixels is $50ms$, while a resolution of 256×256 pixels would allow the use of the camera at almost $30fps$, with the average processing time of $35ms$. In this experiment we used the same scan lines configuration as presented before for a direct comparison.

6.1.2 Performance Evaluation on Different Image Resolutions

We have studied the performance of the omni-directional vision system with different image resolutions. We were interested in finding a relation between image resolution and the reliability of the object detection algorithm. We compare the results obtained using our approach with two other algorithms: the first one, based on finding connected contours and the second one based on region growing segmentation. We tested the following three image resolutions: 1024×1024, 512×512 and 256×256 pixels.

The OpenCV library provides two alternative algorithms for color object detection that have been used for comparison of the results. The first algorithm that has been used is a region growing algorithm [32], in which, using methods from the OpenCV library, the image is scanned horizontally and every time a pixel of the color of interest is found, it is considered to be the seed for the clustering algorithm. The second approach was based on the use of the same library for contour detection [66] and scanning the pixels inside each contour. If the pixels belonging to a given contour are of the color of interest, that contour is validated as being a blob of the color of interest. For all of the three algorithms, for fast color classification, color classes are defined through the use of the LUT already described. The use of a LUT has an important effect on reducing the processing time. Without the use of a LUT, the run-time computations involved for checking the color of a pixel would slow down the performance of the application. The image of color labels will be the basis of all the processing that has been previously described. Using an image of labels not only saves computational time but it also introduces less noise than using the grayscale version of the original image.

For this experiment, we have used a configuration of the radial scan lines of 2880 scan lines, disposed on 4 levels.

Table 6.11 shows that at any of the chosen resolutions, the blob detection algorithm performs faster and the maximum processing time of a frame that was measured was $10ms$, while for example, the contour detection algorithm can take a maximum time of $35ms$ for the processing of one frame.

	Blobs		Contours		Region Growing	
	Mean	Max	Mean	Max	Mean	Max
1024×1024	8.72	10	17.16	35	8.86	21
512×512	2.9	7	5.16	16	2.03	7
256×256	1	5	1.02	3	1	1

Table 6.11: Average and maximum time processing, in milliseconds, for each of the three algorithms studied.

Table 6.12 shows the maximum distances at which the ball is being detected by each of these algorithms. The blob detection algorithms achieves correct results for balls found at a distance of up to $12m$, even for the smallest chosen resolution, while the results of the contours detection algorithm and region growing clustering algorithm slightly degrade with distance.

	Blobs	Contours	Region Growing
1024×1024	12m	12m	11m
512×512	12m	12m	11m
256×256	12m	11m	11m

Table 6.12: Maximum distance from the robot at which the ball is correctly detected by each of the algorithms.

Table 6.13 presents the percentages of correct ball detection for each of the three algorithms. In the video sequences that have been used for tests the robot rotates around itself and the ball is placed in front of it at distances from $1m$ to $12m$. These sequences have a total of 5040 frames, out of which, the ball is present in 4580 frames. The situation in which the ball is not present in a frame is due to the hardware architecture of the vision system.

When the ball is aligned with any of the four supportive bars of the vision system, it cannot be seen in the image as it is fully hidden by one of the bars. The blob detection algorithm has a detection rate of 100% for full resolution images and the decrease of the resolution is followed by a slight decrease in the performance of the algorithm. Out of the three algorithms, the one based on contour detection is not only the heaviest from a computational point of view, but it also performs worse that the other two. Moreover, for the smallest resolution, the contours algorithm can only detect the ball in a quarter of the frames.

	Blobs	Contours	Region Growing
1024×1024	100%	68.5%	73.1%
512×512	95.7%	63.7%	69%
256×256	85.7%	24.6%	53.9%

Table 6.13: Percentages of correct ball detection for the three algorithms.

All three algorithms can detect the ball up to distances of at least $11m$ ($12m$ in some cases). However, for some of the algorithms, the detection rate at longer distances is very small. Tables 6.14 and 6.15 present the detection accuracy for balls that are found further than $6m$ and $9m$, respectively, from the robot.

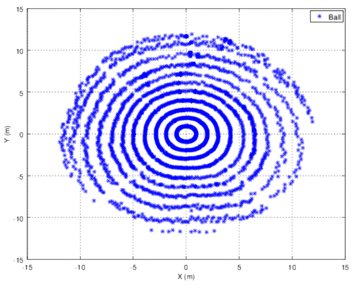
	Blobs	Contours	Region Growing
1024×1024	100%	63.8%	70%
512×512	91%	62.8%	64.7%
256×256	70%	16.3%	22.9%

Table 6.14: Percentages of correct ball detection for the three algorithms for balls found at $6m$ or more from the robot.

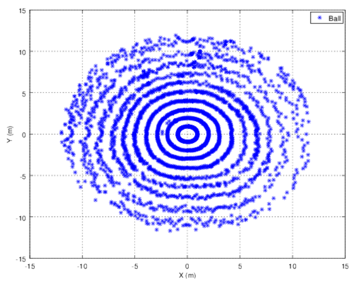
	Blobs	Contours	Region Growing
1024×1024	100%	67.6%	68.8%
512×512	78.4%	58.8%	66.2%
256×256	44.5%	11.5%	11.2%

Table 6.15: Percentages of correct ball detection for the three algorithms for balls found at $9m$ or more from the robot.

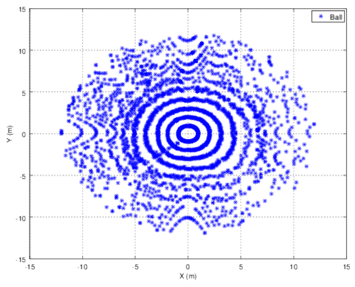
Graphics 6.5, 6.6, 6.7 show the ball detection accuracy for the three algorithms, for each of the three resolutions. The graphics illustrate the position of the detected balls, relative to the robot, in world coordinate at the ground level.



(a) 1024 × 1024



(b) 512 × 512

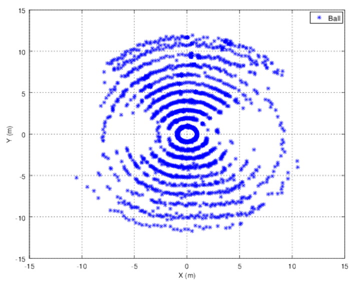


(c) 256 × 256

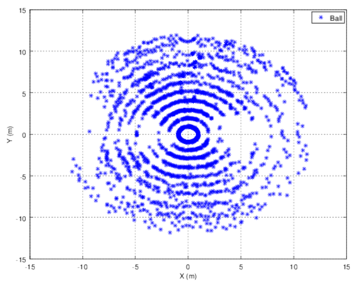
Figure 6.5: Graphics representing the ball detection for the three resolutions using the blob algorithm.

The four "valleys" that can be noted in all of the graphics represent four bars that support the vision system in the hardware architecture of the robot. When the ball is aligned with any of these bars, the ball is not visible.

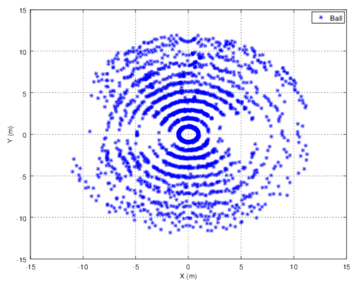
For images of 256 × 256 pixels, the contours detection algorithms provides quite poor results. When the ball is found at $6m$ or more from the robot, it is only detected in 16% of the cases. The results are even less reliable for balls found at $9m$ or more from the robot. In this case, only 10% of the balls are correctly detected.



(a) 1024 × 1024



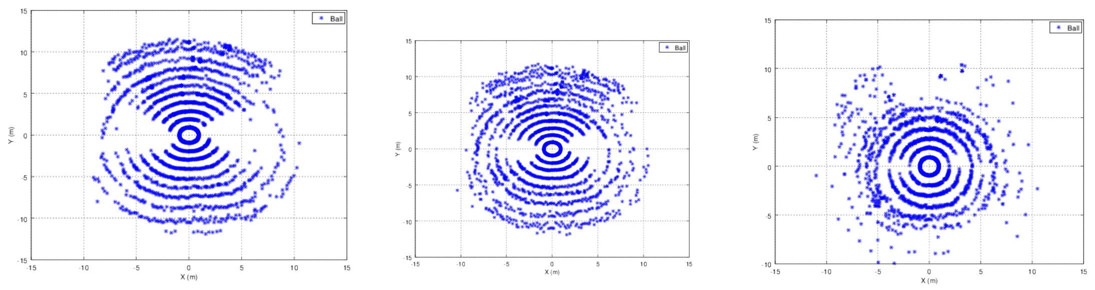
(b) 512 × 512



(c) 256 × 256

Figure 6.6: Graphics representing the ball detection for the three resolutions using the contours detection algorithm.

The region growing clustering algorithm delivers slightly better results, compared to the contours detection algorithm but it is still far from achieving the detection rate of the blob detection algorithm. These results prove that working with images at full resolution can provide optimal results in a small amount of time when using an appropriate scanning pattern of the image. For applications where detection accuracy and total lack of false positives is required, the algorithm based on radial scan lines and blob formation is a suitable solution since its detection rate is 100% and it can be used with sensors working at $100fps$, if the hardware allows it. A smaller resolution can lead to an even smaller processing time, for applications where fast performance is preferred against a having 100% correct detection.



(a) 1024×1024 (b) 512×512 (c) 256×256

Figure 6.7: Graphics representing the ball detection for the three resolutions using the region growing clustering algorithm.

In Fig. 6.8 we present an image with the ball correctly detected by the blob algorithm at a distance of $12m$ from the robot.

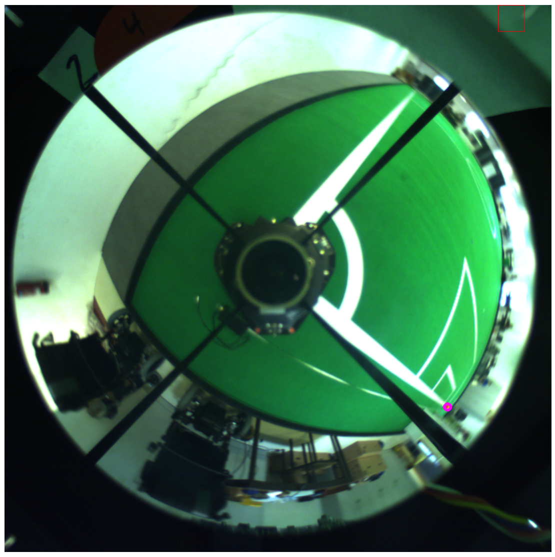


Figure 6.8: Ball detection using the blob algorithm at $12m$. The detected ball is marked with a magenta circle.

Another proof of the effectiveness of the proposed algorithm was obtained by comparing its results in terms of processing time with the results of the watershed transform [67], a well known algorithm for color segmentation. This is just an example of how the use of a LUT for color color labeling is more effective. The watershed transform is a method for defining color regions in an image, but its complexity leads to high processing times, as it will be presented next. For this comparison, we have used sequences obtained with the perspective vision system. The implementation of the watershed algorithm provided by the OpenCV library [31] has been used. For an image of 1600×1200 pixels, the same used in the perspective vision system, the watershed transform algorithm working on the laptop with the $i5$ processor takes an average of $128.4ms$. Table 6.16 presents the partial processing times taken by the Watershed transform:

Operation	Time [ms]
Canny Edge Detection	11.1
Contour Detection and Marking	7.3
Applying the Watershed Transform	110

Table 6.16: Partial processing times for the Watershed transform.

The same algorithm, running on the IGEPv2 board, takes an average time of $1509.8ms$. Table 6.17 presents the distribution of this processing time among the several steps of the processing pipeline:

Operation	Time [ms]
Canny Edge Detection	356.1
Contour Detection and Marking	120.2
Applying the Watershed Transform	1033.6

Table 6.17: Partial processing times for the Watershed transform on the IGEPv2 board.

For this experiment, the thresholds used in the Canny edge detection parameters have been experimentally obtained, through iterations until the green carpet of the soccer field became a single region. Looking carefully into these last processing times, we can observe that a simple operation of edge detection takes more time than the detection of the objects of interest in the referred applications using the proposed approach.

An example of the application of this algorithm in the referred image is presented in Fig. 6.9. We have to underline that using this approach for object detection would require another step for evaluating regions with the same color.

6.2 Delay Measurements between Perception and Action

In addition to the good performance in the detection of objects, both in terms of number of times that an object is visible and detected and in terms of error in its position, the vision system must also perform well in minimizing the delay between the perception of the environment and the reaction of the robot. We consider this study an important contribution of this Thesis since we did not come across anything similar in the literature and its results are relevant for the area of intelligent autonomous robotics. It is obvious that the delay between perception and action depends on several factors, namely the type of the sensor used, the processing unit, the communication channels and the actuators, among others.

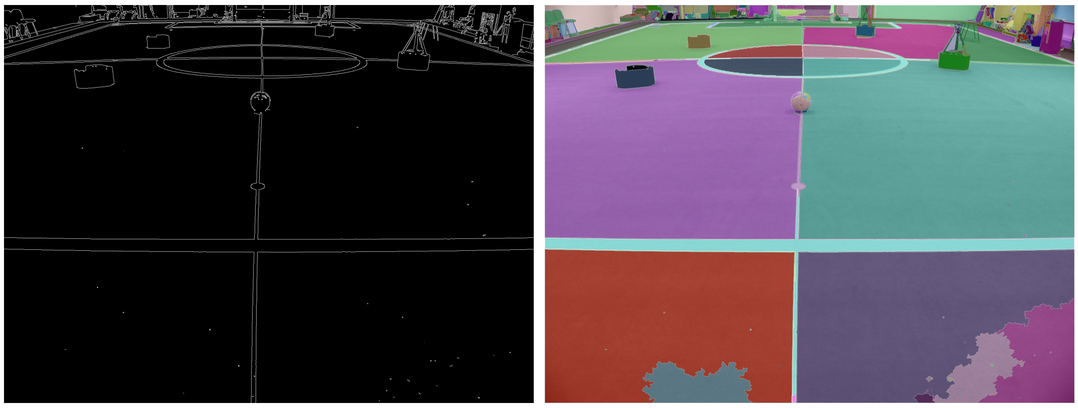


Figure 6.9: On the left, edges obtained after the Canny algorithm. On the right, regions marked after running the Watershed algorithm. A different color is assigned to every region. For object detection, a last step of analyzing the average color of each region is still needed.

6.2.1 Measuring Setup

The delay measurement system that has been used for this study is based on a micro-controller board connected to the omni-directional vision processing PC through an USB port (Fig. 6.10). The micro-controller periodically turns on a LED, during a fixed period of time, and measures the time that the whole vision system takes to detect it in the acquired image. The measured time includes camera acquisition time, image data transfer time and vision processing time. In the setup used for these experiments, the camera is configured with an exposure time of $5ms$ and Gigabit Ethernet connection that can provide for our setup (cable, network card, computer and operating system) $80MB/s$, far from the theoretical $125MB/s$. The vision software in this setup is a simplification of the one described in [68] since we do not have to detect the obstacles and white lines and takes, on average, $3ms$ to detect the led. The vision system detects the led on and when that happens, it sends a message to the micro controller. This is the normal working mode of every robot that uses vision as its main sensor.

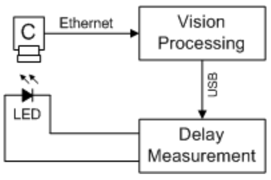


Figure 6.10: The blocks used in our measurement setup.

6.2.2 Delay Measurements

Some digital cameras do not provide frame rates above $30fps$ in RGB, but allow the access to the raw data of the sensor at higher frame rates, mostly due to processing and bandwidth restrictions when the interpolating algorithm runs on the camera. Moreover, it is expected that the conversion between the raw sensor data and a specific color space in the camera or in the device driver on the computer side will take some time. Therefore, we compared the delay when acquiring frames in RGB and when accessing the raw data. Based on the graphics presented in Fig. 6.11, we measured an improvement of 23% of the reaction time at $50fps$ when using raw data. If the camera used by the robot could deliver raw data, these results prove that it is more advantageous to use it as such.

Using a different algorithm in another application, the delay time will be proportional to the processing time of the algorithm used and the scene complexity. Regarding external conditions, namely illumination or other factors, we use in our robots algorithms that guarantee the same properties of the acquired images [69].

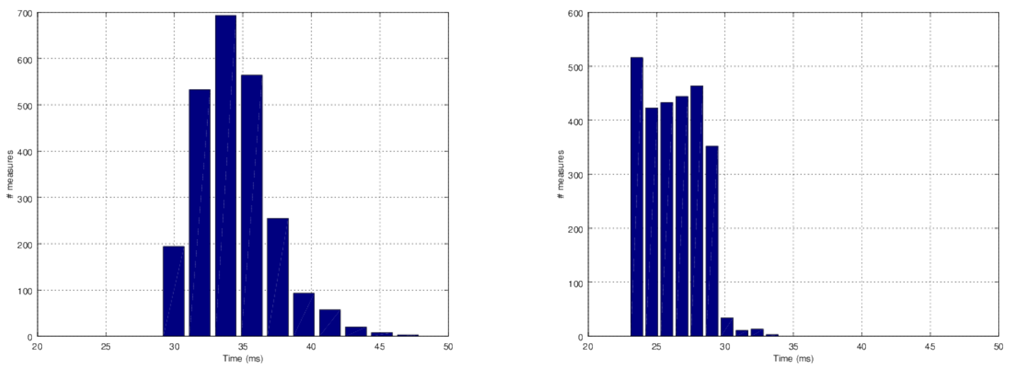


Figure 6.11: Histograms showing the delay between perception and action on the vision system of the CAMBADA robots. On the left, the camera is acquiring RGB frames and is working at $50fps$ (average = $34.2ms$, max = $48ms$, min = $29ms$). On the right, the camera acquiring single channel frames, returning the raw data of the sensor (Bayer pattern) (average = $26.4ms$, max = $34ms$, min = $23ms$).

6.2.3 Detection Performance on Raw Images

After concluding that using the raw images is more advantageous for the compensation of the delay between perception and action in autonomous mobile, we pursued a study on the use of our algorithms for object detection directly on the raw data.

In Fig. 6.12 we present the ball position detected by the omni-directional vision system after the robot performs a kick, considering two different configurations. In one case, the vision system was processing directly the raw image data acquired from the camera and in the other one the full RGB data was processed. As we can see in both graphics, the ball is correctly detected for all its trajectory, both when we process the raw image data and when the processing is performed in the full RGB image.

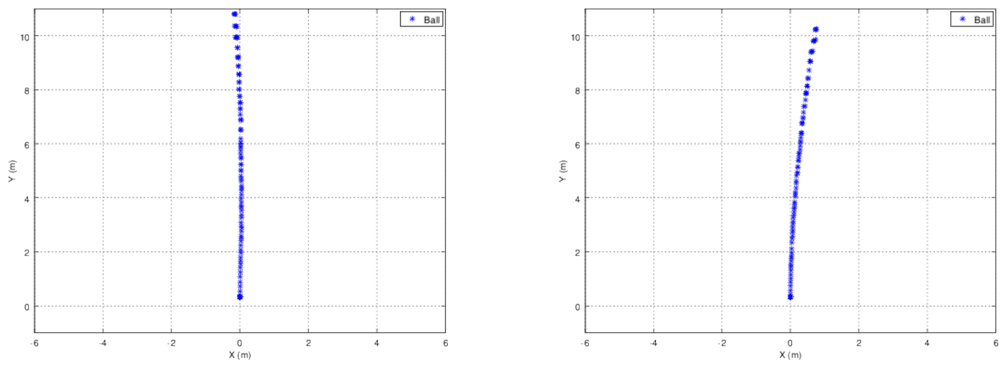


Figure 6.12: Ball detection after being kicked by the robot. On the left, a graphic showing the ball detection on full RGB images after demosicing and on the right, the ball detection on raw image data.

Figure 6.13 shows an example of the result of the color detection algorithms applied directly to a raw image.

Two game scenarios have been tested using the CAMBADA autonomous mobile robots. In each one, two experiments have been made. In one experiment, the vision system processed the full RGB image after interpolation. In the other, the raw image data was processed.

In Fig. 6.14 we present a graphic with the result of the ball detection when the ball is stopped at a given position (the central point of the field, in this case) while the robot is moving. The graphic shows a consistent ball detection with both configurations while the robot is performing a tour around the field. The field lines are also properly detected, as it is proven by the correct localization of the robot in all the experiments.

The second scenario that has been tested is illustrated in Fig. 6.15, where both the robot and the ball are moving. The robot is making a tour around the soccer field, while the ball is being sent across the field. In all these experiments, no false positives were observed and the ball has been detected in more than 90% of the images acquired by the robot. Most of the times when the ball was not detected was due to the fact that it was hidden by the bars that hold the mirror of the omni-directional vision system.

We can observe a different robot trajectory in the tours performed by the robots when the vision system processed the full RGB images, when comparing with the scenario when the vision system processed the raw image data. Since the control parameters were adjusted to the vision system processing the raw image data, the robot has a different behaviour in

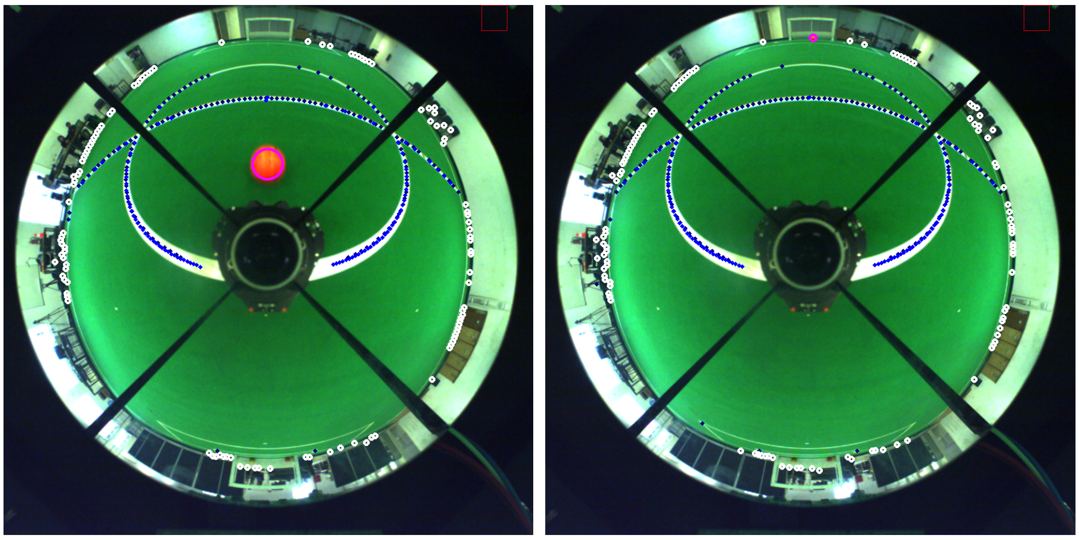


Figure 6.13: Images acquired by the omni-directional vision system with the result of the color-coded object detection on raw images. On the left, the ball was at one meter of the robot. On the right, the ball was at $10m$ from the robot. Blue circles mark the white lines, the white circles mark the black obstacles and the magenta circles mark the orange blobs that passed the validation thresholds and are considered ball.

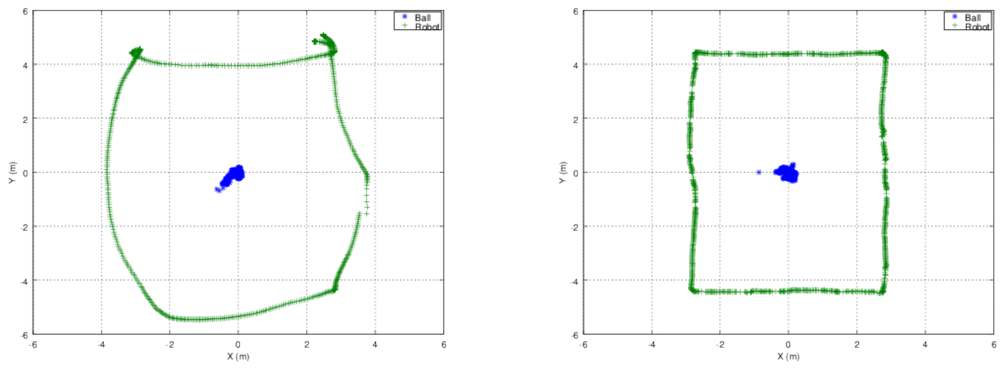


Figure 6.14: Ball detection with the robot moving around the field and the ball stopped in the middle of the field. On the left, a graphic showing the ball detection on full RGB images after interpolation and on the right the ball detection on raw image data.

this situation.

The processing time of each captured image, both in raw data and full RGB color, using the developed vision system is considerably low. The full execution of the pipeline, from image acquisition to object detection after validation (balls, lines and obstacles) only takes at most $10ms$, allowing thus a frame rate of $100fps$ if the digital camera supports it and if

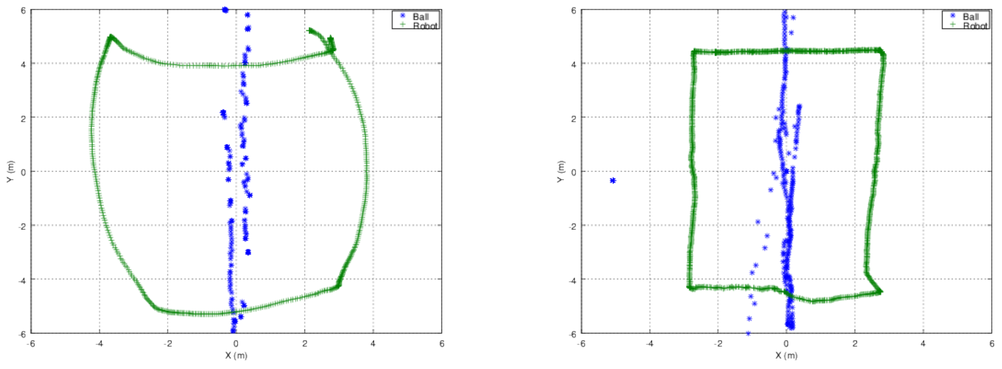


Figure 6.15: Ball detection when both the robot and the ball are moving. On the left, a graphic showing the ball detection on full RGB images after interpolation and on the right bottom the ball detection on raw image data.

---

other control or artificial intelligence processes can run in parallel.

"The production of too many useful things results in too many useless people."

Karl Max

7 Tools

In this Chapter we present three tools that have been developed to complement the robotic vision systems presented in Chapter 5. The first tool was designed with the purpose of choosing the most intuitive color space to be used for manual or user-supervised color calibration. The second tool, with its initial design, was mainly dedicated to the task of manual color classification. A more complex version of this tool has been developed and it allows not only manual but also supervised color classification and eases the robot-human interaction and information exchange.

7.1 *colorSpaces* Tool

A study on the use of color spaces in artificial vision has been conducted, in order to understand if there is a more appropriate one to be used in this kind of applications. A tool for defining color ranges, both by an user and by a semi-automatic algorithm of region growing, under different color spaces, has been developed. The results obtained by using this tool facilitate the choice of a color space when implementing a vision system for robotic soccer players and its application can be extended to any computer vision procedure that requires color segmentation or classification.

As a first functionality, the tool can be used for the manual classification of several colors, under five color spaces - RGB, HSV, HSL, HSI and YUV - (Fig. 7.1). Using sliders, users can manually determine the ranges for each one of the following colors: red, blue, green, yellow, orange, white and black.

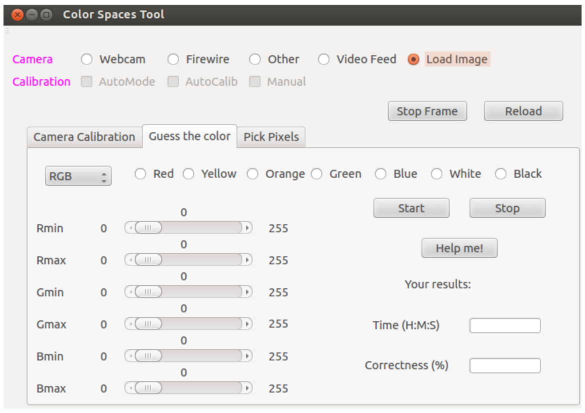


Figure 7.1: Illustration of the feature of the *colorSpaces* tool allowing manual classification of colors.

This feature works as follows: the user chooses an image that wants to classify, called *TrialImageX*, where X is a number identifying the chosen image. The set of trial images contains several images that have different degrees of difficulty of color classification. That is, some of the images contain simple colored objects, without any shadows and without being affected by much noise. The color ranges of these objects are very close to the color ranges established in the literature, which makes the classification easier. The trial images set contains also images that might be more difficult to classify since they contain objects whose colors are affected by the illumination (Fig. 7.2).

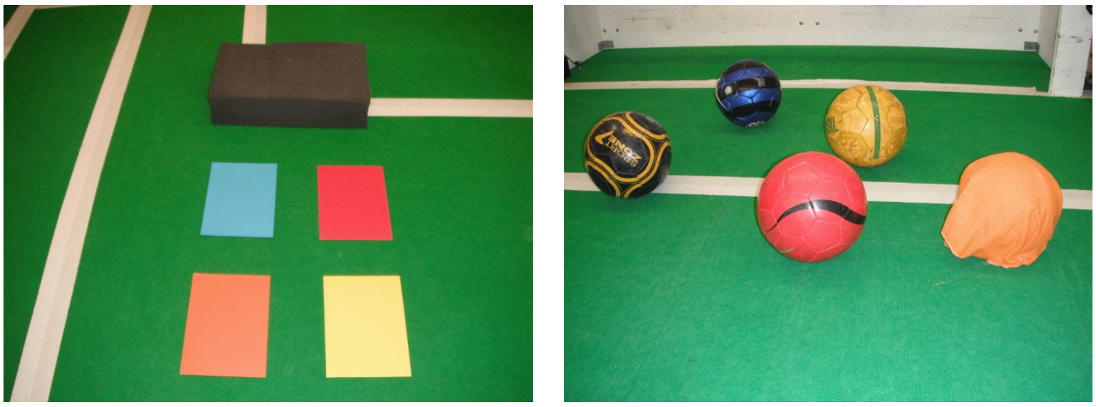


Figure 7.2: On the left, an example of a simple image used for testing manual color classification. On the right, a more complex image that contains color gradients, shadows and objects with more details.

When gathering results about this part of the study, each user has been advised to try the classification of at least one simple image and one complex image. When choosing the *TrialImageX*, the *GroundTruthX* image is also loaded and will be further used for generating the results of the classification. The *ground truth* image is an image that has been already classified by an experienced user (Fig. 7.3). This image is considered to have the correct color ranges for all the colors and will be used for computing the correctness of the classification of each user. The correctness is calculated by direct comparison, pixel by pixel, of the images *GroundTruthX* and *TrialImageX*, when the user concludes the classification. In terms of results, the users are asked to record their correctness as well as their performance time, for each of the color spaces. The performance time is calculated by the graphical tool from the moment in which the user started the classification until the moment that he decided that his classification is correct. When the user finishes the process of color classification, he has to request the comparison with the ground truth.

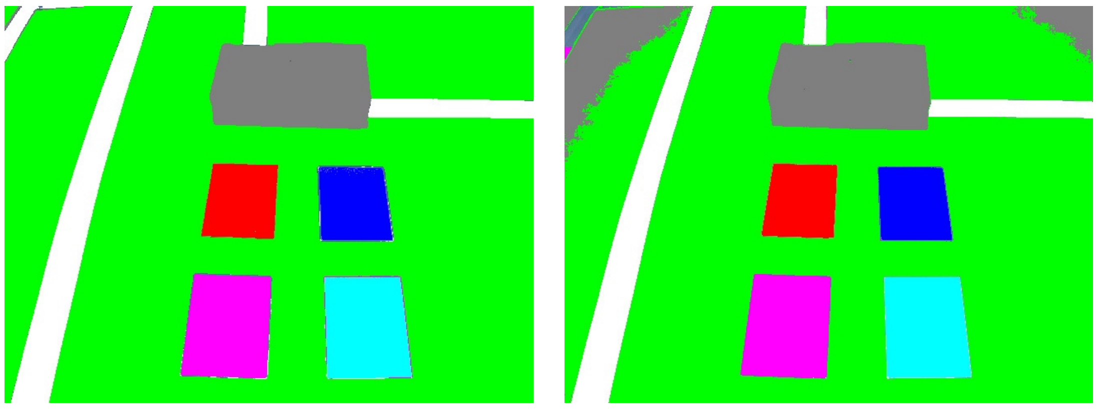


Figure 7.3: On the left, one of the images used as ground truth validation. On the right, the color classification performed by an user.

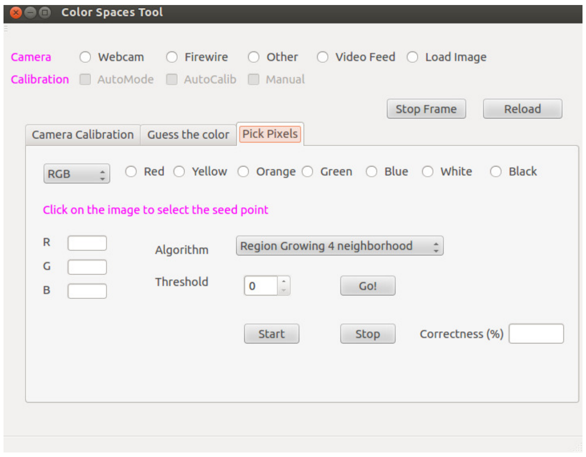
For the second part of the evaluation method, we have tested two different implementations of a region growing segmentation algorithm [32, 70], that have been detailed in Section 3.2. The user chooses a trial image from the same set that was already presented and the correctness of the performance of the algorithm is calculated, as before, by direct comparison with the corresponding ground truth image. This feature of the *colorSpaces* tool is illustrated in Fig. 7.4.

7.2 *calibVision* Tool

calibVision is a graphical tool designed based on the *highgui module* of the OpenCV library [1]for the manual classification of color classes. The user defines color ranges for each of the colors of interest using this tool (Fig. 7.5).

The performance of the vision systems that have been presented in Chapter 5 is directly influenced by the quality of the color classification. Prior to the use of any of the vision

[1]http://docs.opencv.org/2.4/modules/highgui/doc/highgui.html

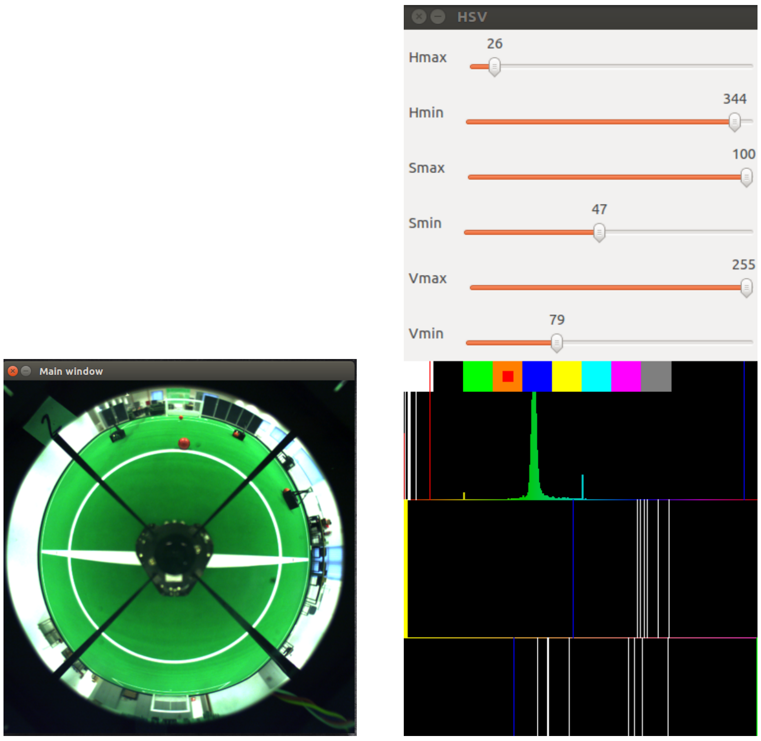


RGB

Algorithm: Region Growing 4 neighborhood

Threshold: 0

Figure 7.4: Illustration of the feature of the *colorSpaces* tool allowing supervised classification of the colors.



Hmax 26

Hmin 344

Smax 100

Smin 47

Vmax 255

Vmin 79

Figure 7.5: A screenshot of the *calibVision* tool.

systems in a new environment, a color classification is required. The result of the classification procedure is a range of maximum and minimum H, S and V values for each of the colors of interest that is saved onto the configuration file. The structure of this file is presented in Appendix B. We have chosen the approach of manual color classification based on the use of the *calibVision* tool. The use of the algorithm for colormetric calibration of the camera allows the re-use of the same configuration file and the same color ranges as long as the robots perform in the same environment. Because of this, the task of manual color classification cannot be considered time-consuming and is not compulsory prior to each performance of the robots, only when the environment changes.

This tool can communicate with the vision system running on the robots through a TCP/IP connection. The TCP/IP module of the UAVision Library has been employed in order to build this connection. This module, described with more emphasis in Section A.4 addresses the implementation of a client-server type of communication system. The vision system that runs on the robot works as a server that accepts a client connection. In this case, the *calibVision* tool is the client. The client can request visual information from the robot and evaluate the perception of the robot on the surrounding environment. The request and information exchange is done based on key interactions. These key interactions are presented in more detail in Appendix D. The information exchanged can be an image (with or without debug symbols), an image mask or a color range structure. The color range structure contains the maximum and minimum limits of the *(H, S, V)* triplet for each color of interest.

With the use of sliders and mouse interaction, the user defines the H, S and V minimum and maximum values for a color of interest (Fig. 7.5). The tool supports up to 8 colors of interest. The user can click on the image and the H, S and V value of a certain pixel is drawn on a window of the tool in order to ease the color classification procedure and reduce time. Figure 7.6 shows a result of a color classification process. Once the user finishes this process, he sends the color ranges to the server side, where they are updated accordingly, both on the configuration file and in real-time use in the color segmentation step of the object detection pipeline.

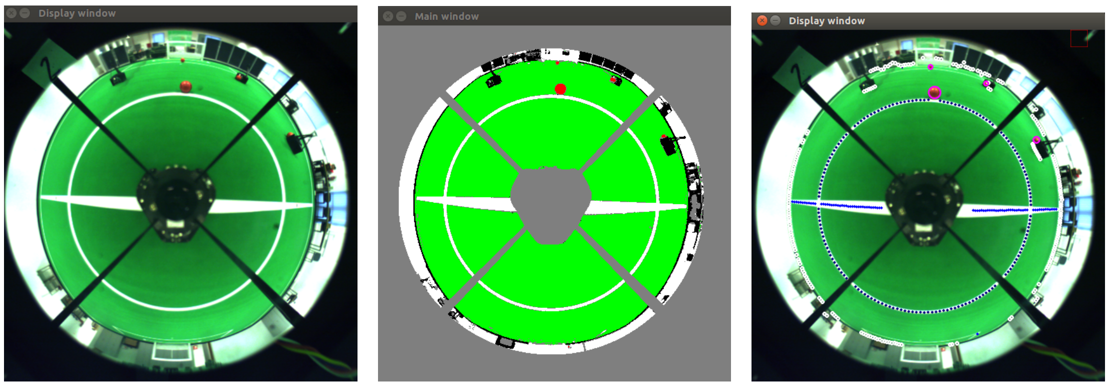


Figure 7.6: On the left, an original image acquired by the robot that was sent to the *calibVision* client. In the middle, the color classification result on the client side. On the right, the debug points displayed after the color ranges have been received on the server side and the objects of interest have been detected.

Moreover, this tool can be used to send commands to the vision system, such as activating the flag for the auto-calibration of the colormetric parameters. These commands are sent by key interactions.

7.3 *GUICalib* Tool

GUICalib tool was designed after the experience with *calibVision* as a calibration tool of the robotic vision systems, but more generic, flexible and user-friendly. This tool (Fig. 7.7) works as a client that can connect to the vision system server. Apart from supporting the color classification procedure, both manually and user-supervised, this tool integrates more features that will be presented next.

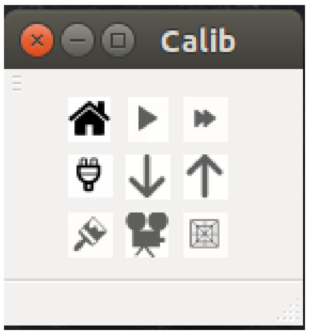


Figure 7.7: A screenshot of the *GUICalib*

The *Local Access* function (Fig. 7.8) of this tool allows the user to choose the type of camera that is available either locally or on the server side, as well as the configuration file. If there is no configuration file, the user can choose to create a new one, that will include default values for the type of camera chosen. The types of digital cameras that are currently supported are: USB, Ethernet, Firewire or Kinect.

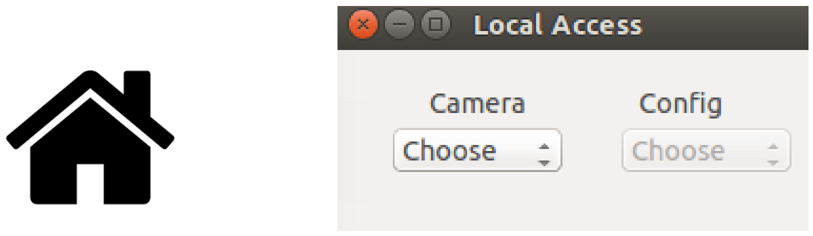


Figure 7.8: Local Access function.

The *Grab Frames* functions allow the user to choose to grab a single frame or to receive a continuous stream from the server (Fig. 7.9). In a game situation, if performed by an experienced user, a single frame can be sufficient for the color classification process. When the user wants to monitor the detection of the object detection algorithm during the performance of the robots, for example, grabbing a continuous video stream from the robot can be of more help. These frames are available at the client side either as the original images that the cameras acquire or with debug information overimposed. The debug information represents visual marks on the objects of interest that are being detected when the frame was grabbed.

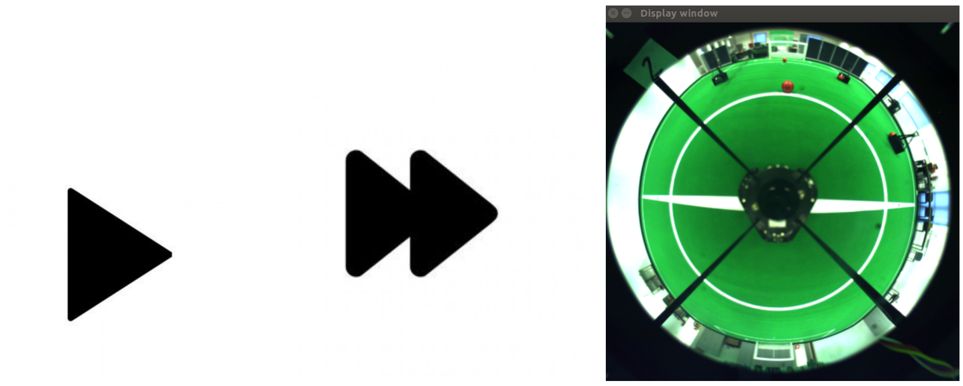
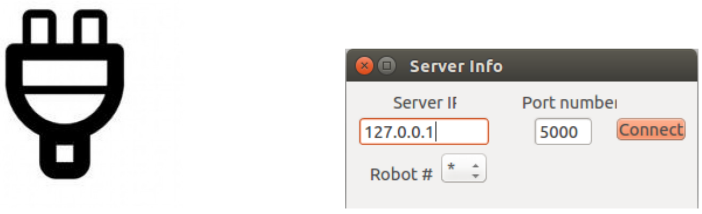


Figure 7.9: Grab Frames function.

The *Server Connection* function awaits for a server address and port in order to establish a TCP/IP connection (Fig. 7.10). By default, the client tool connects to the local host, where it expects to find a server running. When in use with the CAMBADA robots, their IPs have been predefined and the user has the option to chose the number of the robot to which he wants to connect. When the robot number is chosen, the options for introducing the IP and port are disabled.



Server IP: 127.0.0.1

Port number: 5000

Robot #: *

Figure 7.10: Server Connection function.

The *Receive Info* and *Send Info* functions provide the options for the client to request different information from the server and at his turn, to send back to the server the same type of information (Fig. 7.11). This information can be the whole configuration file or only individual blocks of information, such as: the color ranges, camera settings, image mask or map of coordinates.

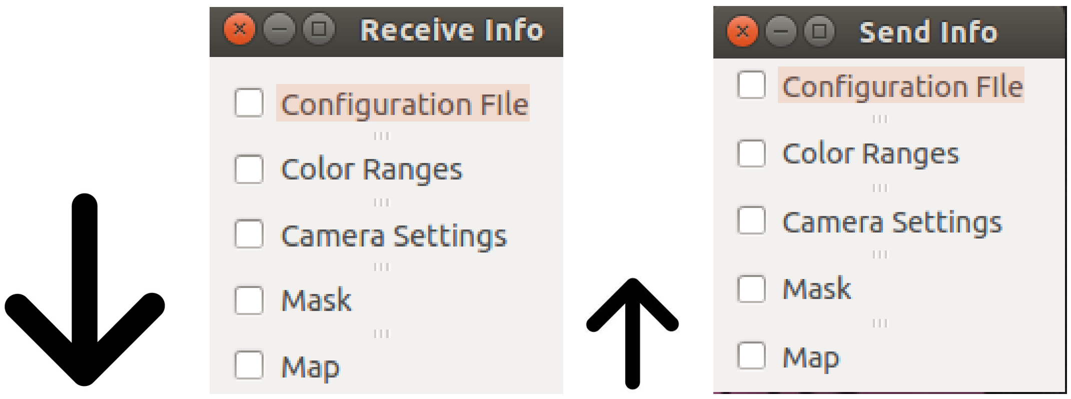
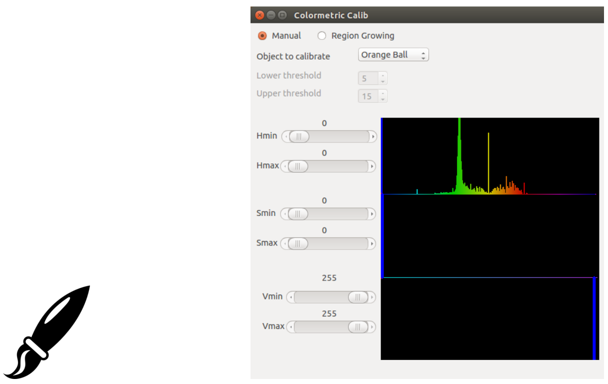


Figure 7.11: Receive Info function on the left and Send Info Function on the right.

GUICalib supports the same manual color classification procedure as *calibVision* through the *Color Classification function*. In addition, a method for semi-autonomous, supervised color classification is available (Fig. 7.12). First the user has to choose the object of interest whose color he wants to classify. When choosing manual mode, the sliders become enabled and the user can choose the (H, S, V) maximum and minimum values for each of the objects of interest. In supervised mode, the threshold input lines become enabled and the sliders are disabled. In this mode the user chooses the seed on the object that we wants to color classify and can choose a minimum and maximum threshold values. After having chosen the thresholds, the user has to click on a point in the image that belongs to the object of interest that he chose. A region growing algorithm uses the clicked point as a seed for growing a region of that color. The minimum and maximum threshold values will be used for comparing the neighboring pixel values with the seed in order to establish if they are part of the same color region.The (H, S, V) average values of that region will be used as the color ranges of that specific color.



[✓ Manual] Region Growing

Object to calibrate: Orange Ball

Lower threshold: 5

Upper threshold: 15

Hmin: 0

Hmax: 0

Smin: 0

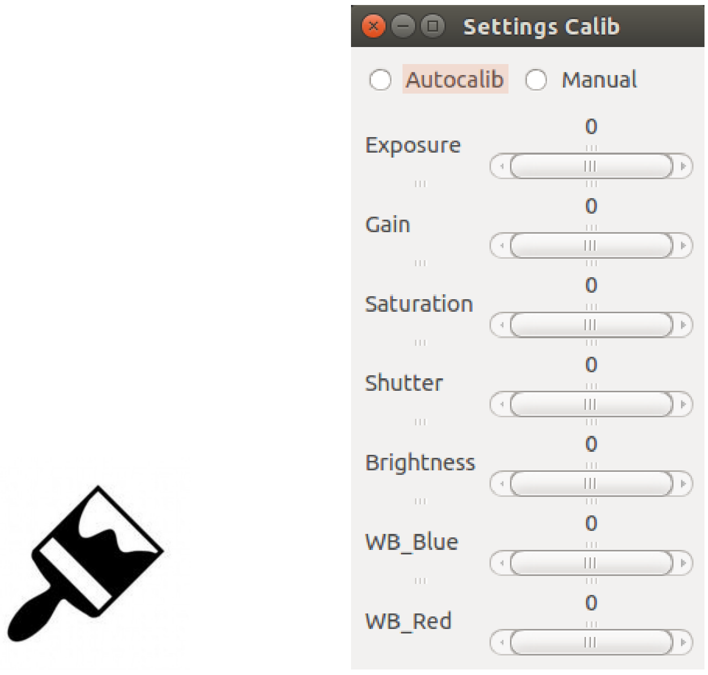
Smax: 0

Vmin: 255

Vmax: 255

Figure 7.12: Color Classification function.

Finally, the *Manual Calibration* function allows the user to manually change the colormetric parameters of the camera in use, by means of sliders. Figure 7.13 shows all the colormetric parameters supported by this tool. However, not all of them are available on all the cameras supported. When working with a given camera that does not include any one of these parameters, the corresponding sliders are disabled. Moreover, this tool supports the option of activating an auto-calibration flag, which will start the automatic colormetric calibration of the camera on the server side.


Settings Calib

○ Autocalib ○ Manual

Exposure 0

Gain 0

Saturation 0

Shutter 0

Brightness 0

WB_Blue 0

WB_Red 0

Figure 7.13: Manual Calibration function.

"You do not need to know all the answers. No one is smart enough to ask you all the questions."

Unknown

8 Conclusions and Future Work

This document presented an approach for time-constrained color object detection algorithms for robotic applications. Several robotic vision systems and algorithms have been detailed and experimental results with application in autonomous robotic soccer have been presented. In this Chapter, we draw the conclusions and discuss the effectiveness of the proposed solutions. Moreover, we identify possible future work directions.

8.1 Conclusions and Discussion

Robotic soccer has been in the last years a popular test bed for innovations in artificial intelligence, biped walking, multi-agent cooperation and computer vision. The game of soccer imposes many challenges that can lead to great innovations in all the aforementioned research areas. In the case of computer vision, since the movements of the robots impose many challenges on the task of processing the environment, a soccer game can only take place by overcoming them.

This Ph.D. was focused on providing a modular solution for the detection of colored objects, with focus on the robotic soccer games in the RoboCup MSL. We introduced a novel computer vision library, UAVision, that can be used for solving the task of time-constrained colored object detection. The library is composed of several modules that can be used separately for solving individual tasks of an object detection pipeline or that can be joined into a complete time-constrained vision system, such as the ones presented in Chapter 5. An important contribution of this Thesis was the study on the raw images, on their use for object detection as well as on the effect of the delay between the perception and action of an autonomous intelligent robot.

We have presented different approaches for solving most of the image processing tasks of a vision system for soccer robots in Chapter 2. These approaches make use of the color information in order to detect the objects of interest. However, we have not found any complete description of a functional robotic vision system for robotic soccer, each of these related works only addressing individual aspects of this challenge. Therefore, we consider this Ph.D. an important contribution for this research area. The work that we presented is a complete set of algorithms and support tools that allow the implementation of a fully functional robotic vision system for colored object detection.

In Chapter 3 we have addressed the calibration aspect of a camera, with more emphasis on a novel algorithm for colormetric calibration of the camera that we proposed. Before proceeding to any type of image processing it is necessary to guarantee, in any artificial vision system, a precise camera calibration. This calibration can reflect either the tuning of the intrinsic and extrinsic parameters of the camera in order to find a viable correspondence between the camera frame and world frame, or it can maximize the quality of the acquired image by finding the most appropriate values for the colormetric parameters of a camera. An efficient colormetric calibration of a camera should result in less noise in an image. We have introduced an algorithm for the colormetric calibration of a camera that can work in real-time, without affecting the global processing time of the vision system. This algorithm allows the robots to play soccer outdoor, which is the main next goal of the RoboCup Soccer League. The results that we have presented show that it is possible for the camera parameters to converge to stable and optimal values, even when working in very dark or very bright environments.

Chapter 4 was dedicated to the algorithms developed for time-constrained color detection. We have presented a flexible approach for scanning an image, that ensures the optimal compromise between the amount of information that is being processed and the processing time spent. We have introduced as well several LUTs based on which complex operations can be replaced by simple indexations. The use of LUTs further improves the processing time of the global system. Blobs have been introduced as data structures and they can be seen as the initial hypothesis of an object of interest. Given the geometry of an object of interest, different measurements of the blobs have been presented and they are used for further refinement of the valid candidates. All the algorithms that have been described have been merged into a computer vision library. These algorithms can be used independently of each other or they can be used as a whole, complementing each other towards solving a complex task, such is the one of detecting objects of interest during a robotic soccer game.

The solutions that we have developed, based on the complementary use of these algorithms have been presented in Chapter 5. We have presented a common structure of three kind of robotic vision systems: an omni-directional one, a perspective one and a Kinect-based one developed for the detection of aerial balls. The vision systems that we have presented have proven their efficiency in several robotic competitions throughout the last years. With average processing times under 10ms for images of 1024×1024 pixels and detection rates very close to 100%, these vision systems represent an effective contribution not just for the RoboCup community, but to the computer vision and robotics fields in general, as well.

Furthermore, we discussed the performance of our approaches in Chapter 6. We provided results of the algorithms for different image resolutions and we highlighted the effects of resolution, scanning pattern and number of scanned points on the object detection rates and accuracy. We have compared our approach with two well-known computer vision algorithms for colored object detection, which are supported by the OpenCV library. The results prove that our solution outperforms the other two, both in terms of detection accuracy and processing time for each of the tested resolutions. We have included results of the use of an IGEP v2 controller as main processing unit, in order to simulate a possible robotic system with lower computing resources. These results showed that it is possible to use the complete software pipeline at high frame rates on robotic platforms that are both affordable and reduced in size and processing capabilities. Moreover, we have presented a study on the use of raw images for object detection that proves that the detection efficiency is not compromised and that the interpolation step of raw images is not compulsory for object detection. This study was complemented with experiments on the use of raw and RGB images for the exploration of the delay between perception and action of an intelligent autonomous robot.

Chapter 7 presented different tools that we have developed as further support of the software used in soccer games. The first tool served as a study for the most appropriate color space to be used for applications which involve human supervision and human recognition of color. The second tool, in its two versions, is another significant contribution for the RoboCup Soccer League. It is used for the manual or semi-automatic training of the colors of interest and it is an approach widely used by this community. In addition, this tool can be used for remote monitoring of the vision systems of the robots, in real-time, without interfering with their normal flow of operations.

8.2 Future Work

There is always room to improve any research effort and this work is not an exception. Towards the end of its development, several development directions have been identified and are presented as follows:

- On the matter of colormetric calibration, the development of an autonomous procedure for the detection of the white area would increase the autonomy of the algorithm. In its current version, the algorithm uses a white area that is defined in advance for the tuning of the white balance parameter. Using the approach of scan lines to search for white areas that are always present in the image, without depending on their predefined position, would turn the algorithm more autonomous and independent of prior artifacts. However, this has a direct implication on the processing time of the calibration algorithm. In addition, a solution for dealing with situations in which there is no white information present in the captured images should be thoroughly designed.

- The scan patterns that have been implemented and used in this paper can be employed for other types of search, different than color search. Transitions between intensities, which lead to edge detection, can be found using scan lines. This approach can stand at the basis of ball detection independently of its color, as a further application to the domain of robotic soccer.

- In what concerns raw images, their use by the Computer Vision community has been so far limited. We have proven that in terms of object detection, there are no negative aspects when using the raw data directly. Moreover, we have shown that the delay between perception and action in a robot can be reduced when processing directly the raw images. In addition, we have conducted an initial study on the compression of raw data, which shows that the raw data can be compressed in an efficient way. A further study on the compression of such images could be of great importance for domains that require remote monitoring or exchange of big amount of visual data between an agent and a base station.

- Still on the subject of raw data, we have also conducted a set of initial experiments on the use of the SURF [71] and SIFT [72] feature transforms directly on Color Filter Array images. The results that we have obtained show that the use of these transforms directly on raw data is similar to their use on grayscale images in terms of number and quality of keypoints. We are interested in further continuing this research direction and perform generic object detection on raw data.

8.3 Acknowledgements

This work was developed in the Institute of Electronics and Informatics Engineering of University of Aveiro and was partially supported by FEDER through the Operational Program Competitiveness Factors - COMPETE and by National Funds through FCT - Foundation for Science and Technology in a context of a PhD Grant (FCT reference SFRH/BD/85855/2012) and in the context of the projects UID/CEC/00127/2013 and Incentivo/EEI/UI0127/2014.

"There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult."

Tony Hoare

A
UAVision Library

Even though digital cameras are quite inexpensive nowadays, thus making artificial vision an affordable and popular sensor in many applications, the research done in this field has still many challenges to overcome. The main challenge when developing an artificial vision system is to process all the information acquired by the sensors within the limit of the frame rate and to decide in the smallest possible amount of time which of this information is relevant for the completion of a given task.

This Appendix presents the UAVision library, a free, open source computer vision library that resulted from the work accomplished during this Ph.D. The library aims at being a complete collection of software for real-time color object detection. UAVision can be split into several independent modules that will be presented in the following sections. The architecture of the modules that make up this library is of the type "plug and play", meaning that it offers support for different digital cameras technologies and the software created using the library can be shared among different types of cameras. Hence, the library is modular as each module can be used independently as a link in an image processing chain or several modules can be used for creating a complete pipeline of an artificial vision system.

This library has been developed taking into consideration time constraints. All the algorithms behind it have been implemented focusing on maintaining the processing time as low as possible and allowing the use of digital cameras at the maximum frame rates that their hardware supports. In Autonomous Mobile Robotics, performing in "real-time" is a demand of almost all applications. The main purpose of robots is to imitate humans and we, humans, have the capacity of analysing the surrounding world in "real-time". Even though there is not a strict definition of real-time, almost always it refers to the amount of time elapsed between the acquisition of two consecutive frames. Real-time processing means processing the information captured by the digital cameras within the limits of the frame rate.

The UAVision library has been built using some of the features of the OpenCV library, mostly data structures for image manipulation. Although a powerful tool for developers working in image processing, the OpenCV library does not provide all the necessary support for implementing a complete robotic vision system from scratch. To name some of the contributions of our library, it provides support for accessing different types of cameras using the same interface, algorithms for camera calibration (colormetric, catadioptric and perspective calibration) and the possibility of using image masks to process only relevant parts of the image.

This Thesis aims at being an important contribution for the Computer Vision community and it introduces a novel computer vision library, whose design focuses on maintaining a low processing time, a common constraint of nowadays autonomous systems.

A.1 Image Acquisition Module

Three different camera technologies are currently supported and have been used in practical applications in order to prove the versatility of the proposed library and to exemplify the range of options that an user has for implementing a vision system using the UAVision library.

UAVision provides the necessary software for accessing and capturing images from three different camera interfaces, so far: USB (including a Kinect[1] sensor), Firewire and Ethernet cameras. For this purpose, the Factory Design Pattern [73] has been used in the implementation of the Image Acquisition Module of the library. A factory called "Camera" has been implemented and the user of the library can choose from these four different types of cameras in the moment of the instantiation (Fig. A.1). This software module uses some of the basic structures from the core functionality of OpenCV library: the *Mat* structure as a container of the frames that are grabbed and the *Point* structure for the manipulation of points in 2D and 3D coordinates. Images can be acquired in the YUV, RGB color format or as raw, Bayer data. The library also provides software for accessing and capturing both color and depth images from a Kinect sensor [74].

Apart from the Kinect sensor, the drivers for the following cameras have been implemented and are currently supported:

- IDS UI-5240CP-C-HQ-50i Color CMOS 1/1.8 Gigabit Ehernet camera[2].

[1]https://dev.windows.com/en-us/kinect

[2]https://en.ids-imaging.com/store/ui-5240cp.html

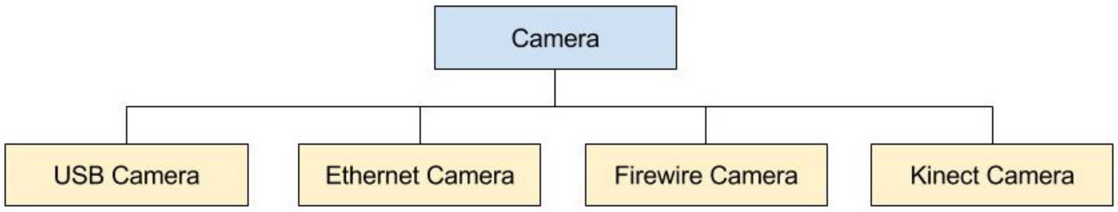


Figure A.1: Illustration of the Camera Factory pattern.

- Point Grey Flea3 FireWire camera[3].
- Point Grey Zebra2 Ethernet camera[4].

Apart from the image acquisition software, this module also supports methods for converting images between the most used color spaces: RGB to HSV, HSV to RGB, RGB to YUV, YUV to RGB, Bayer to RGB and RGB to Bayer (Fig. A.2).

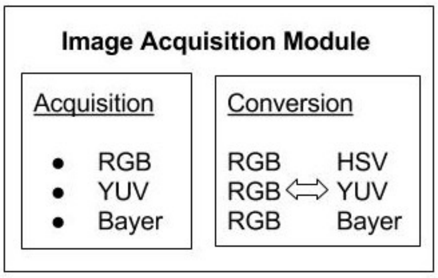


Figure A.2: Image Acquisition Module.

A.2 Camera Calibration Module

The correct calibration of all the parameters related to a vision system is crucial in any application. The Camera Calibration Module (Fig. A.3) includes algorithms for calibration of the intrinsic and extrinsic camera parameters, the computation of the inverse distance map, the calibration of the colormetric camera parameters and the detection and definition of regions in the image that do not have to be processed. The process of the vision system calibration handles four main blocks of information: camera settings, image mask, map and color ranges.

[3]https://www.ptgrey.com/flea3-ieee-1394b-firewire-cameras
[4]https://www.ptgrey.com/zebra2-gige-vision-poe-hd-sdi-cameras

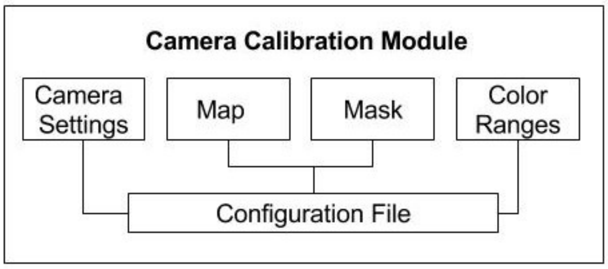


Figure A.3: Camera Calibration Module.

The *camera settings* block represents basic camera information. Among others, this includes: the resolution of the acquired image, the Region of Interest regarding the CCD or CMOS of the camera and the colormetric parameters. This block is also designed following the Factory Design Pattern (Fig. A.4).

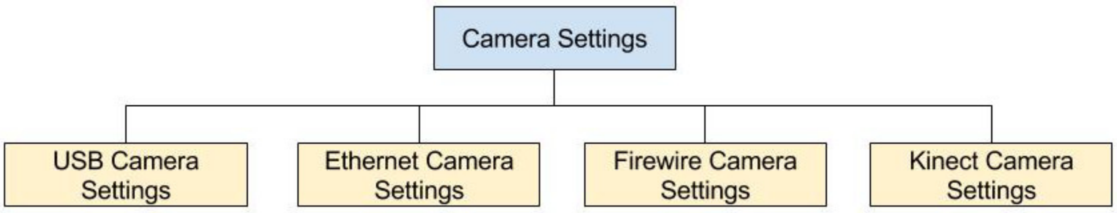


Figure A.4: Illustration of the Camera Settings Factory pattern.

The *image mask* is a binary image representing the areas of the image that do not have to be processed, since it is known a-priori that they do not contain relevant information. Using a mask for marking non-interest region is an important approach especially when dealing with a omni-directional vision system. These systems provide a 360°of the surrounding environment and that includes, most of the times, parts of the robot itself. By using a mask that allows skipping the processing of certain regions, the processing time decreases significantly.

The *map*, as the name suggests, is a matrix that represents the mapping between pixel coordinates and world coordinates for a specific plan. This mapping between the pixels of the acquired image and real coordinates allows the vision system to characterize the objects of interest, their position or size.

The *color ranges* block contains the color ranges for each color of interest (at most 8 different colors) in a specific color space (ex. RGB, YUV, HSV, etc.). In practical means, it contains the lower and upper bounds of each one of the three color components for a specific color of interest (Fig. A.6).

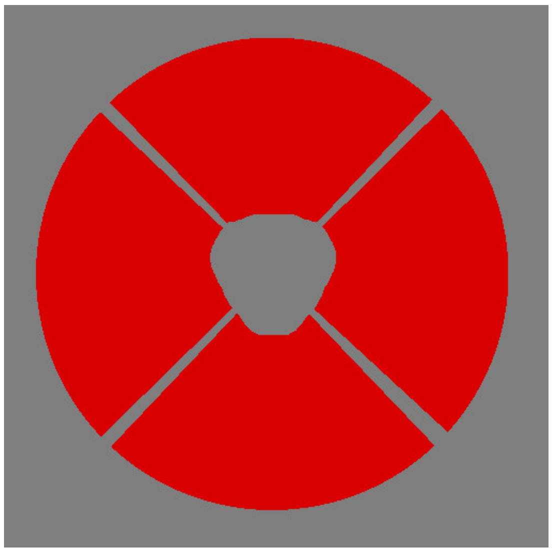


Figure A.5: Illustration of an image mask used with the omni-directional vision system. In this particular case, only the red pixels will be processed at any time.

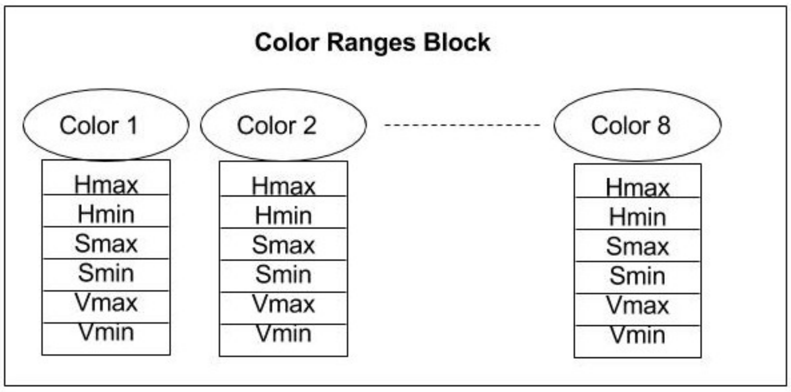


Figure A.6: Information contained by the color ranges block.

The Camera Calibration module of the library allows the storage of all of the information contained in these four blocks in a binary or text configuration file that is presented in Appendix B.

A.3 Color-coded Object Segmentation Module

The Color-Coded Object Segmentation Module is composed of three sub-modules that are illustrated in Fig. A.7).

- **Look-Up Table** - For fast color classification, color classes are defined through the use

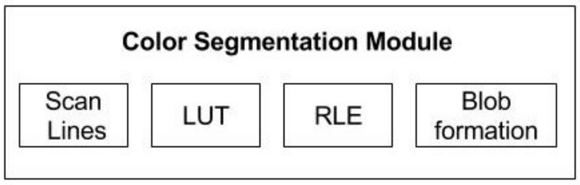


Figure A.7: Color Object Detection Module.

of a look-up table. A LUT represents a data structure, in this case an array, used for replacing a runtime computation by a basic array indexing operation. This approach has been chosen in order to save significant processing time. The images can be acquired in the RGB, YUV or Bayer format and they are converted to an index image (image of labels) using an appropriate LUT for each one of the three possibilities.

- **Scan lines** - To extract color information from the image we have implemented three types of search lines, which we also call scan lines: radial, linear (horizontal or vertical) and circular (Fig. A.8). They are constructed once, when the application starts, and saved in a structure in order to improve the access to these pixels in the color extraction module.

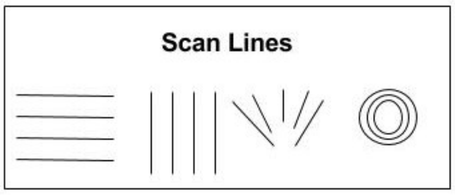


Figure A.8: Scan Lines Software Block.

- **Run Length Encoding** - For each scan line, an algorithm of run length encoding is applied in order to obtain information about the existence of a specific color of interest in that scan line. To do this, we iterate through its pixels to calculate the number of runs of a specific color and the position where they occur. Moreover, we extended this idea and it is optional to search, in a window before and after the occurrence of the desired color, for the occurrence of other colors. This allows the user to determine both color transitions and color occurrences using this approach. As a result of this module, the user will obtain a list of positions in each scan line and, if needed, for all the scan lines, where a specific color occurs, as well as the amount of pixels in each occurrence.
- **Blob formation** - To detect objects with a specific color in a scene, one has to detect

regions in the image of that color, usually designated as blobs. In order to construct the blobs, we use information about the position where a specific color occurs based on the Run Length Sub-module previously described. We iterate through all of the run lengths of a specific color and we apply an algorithm of clustering based on Euclidean distance.

A.4 TCP Communications Module

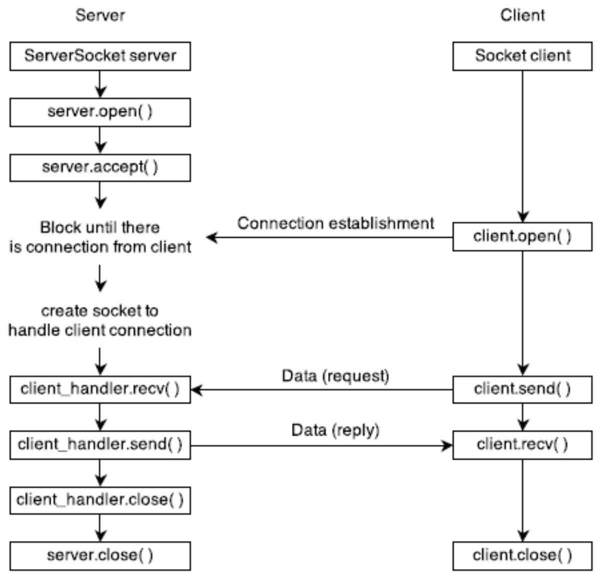


Figure A.9: TCP Communications Module.

Two of the major limitations that have to be considered when working with autonomous robots are the energetic autonomy of the robot and its processing capacities. The broad experience in autonomous robotics within our research group has pointed out the fact that certain calibration tasks, that have to be performed prior to the functioning of a robot, might require too much of the on board resources. Therefore, following the approach “divide and conquer” a TCP Communications Module has been integrated within the library.

The TCP module (Fig. A.9) allows us to remotely communicate with a robot and distribute some of the tasks that are solely related to calibration processes or real-time monitoring of the performance of the robot. This module is particularly relevant when developing applications for logging or monitoring, in which the robot performs autonomously and there is the need of having a real-time remote logger/monitor.

This module allows the user to have a client and a server that can exchange information via a TCP connection. The exchanged information is an image, with or without any annotated information. Based on this module, the vision system that will run on a robot can act as a server and an external application, the client, can request information from the server while it is performing its regular task. By using threads, the communications with the client will not affect the normal flow of operations on the server side. A tool for color calibration and monitoring of the vision system has been developed and was presented in Chapter 7.

"Information is not knowledge."

Albert Einstein

B
Structure of the Configuration File

Data	Size (bytes)
File version	4
Existent camera settings flag	4
Existing color ranges flag	4
Existing map flag	4
Existing mask flag	4
Number of columns (nCols)	4
Number of rows (nRows)	4
CCD center column	4
CCD center row	4
Image center column	4
Image center row	4
CCD interior radius	4
CCD exterior radius	4
Frame rate	4
Video mode	4
Colour coding	4
Number of channels	4
Used camera flag	4
In-use camera flag	4
Map (vector of nRows*nCols of doubles)	nCols*nRows*1
Mask (vector of nRows*nCols of unsigned chars)	nCols*nRows*8
Color range (name - 64 chars, Hmin, Hmax, Smin, Smax, Vmin, Vmax)	9*92
Colourmetric parameter A	4
Colourmetric parameter B	4
...	...
Colourmetric parameter N	4

Table B.1: Meaning and size of the information that can be registered in a typical configuration file. Colourmetric parameters from A to N depend on the type of camera and can be any of the following, depending on what a given camera supports: exposure, brightness, shutter, gain, saturation, white balance.

"The scientist is not a person who gives the right answers, he is one who asks the right questions."

Claude-Levi Strausse

C
Color Classification Questionnaire

Each user of the *colorSpaces* tool, that was presented in Chapter 7 was asked to fill in a questionnaire concerning his interaction this tool. The questions asked are presented as follows, along with the averaged responses obtained from the users.

• *Are you working in the area of computer vision/image processing?*

The first question was relevant for the authors of the applications in order to establish a connection between the performance time of each user and its background in a related field. The subjects were mainly non experienced users, only 20% of them have had some experience in this field. The users that were familiar to these issues, performed slightly faster (in average, 30 seconds faster).

• *The notion of color spaces was familiar to you at the beginning of this test?*

All users replied that they were aware of the definition of a color space. However, most of them were not familiar with the characteristics of each color space. The tool has a "Help me!" button which allows the user to choose a color space and the geometrical representation of the color spaces will be displayed in a separate window, similar to the ones presented in Section 3.2.

• *Do you think that some of the color spaces were more intuitive to use, than the others? If yes, which ones were easier to use?*

65% of the users' first choice was YUV, followed by HSL. In opposition to this, RGB was the more difficult one for all the users.

• *Do you think that some of the colors that you had to classify were more intuitive to classify than others? If yes, which ones?*

85% of the users answered that white, black and the RGB primary colors were easier to classify that the rest.

"Talk is cheap. Show me the code"

Linus Torvalds

D
Software Arguments

All the software implemented during this Ph.D. is free and open source [75]. We detail here the commands for running any of the vision systems that have been developed, as well as all of their optional arguments. The software was implemented in C++ and runs under Ubuntu 14.04 LTS Trusty Thar.

D.1 Omni-directional Vision System

The command for running the software behind the omni-directional vision system is:

```
./omniVision  -cf configFileName,
```

where $configFileName$ represents the name of the configuration file to be used.

Other optional arguments that can be used are:

```
-h (help)
-nodisp (do not show image)
-server (accept connections from a client)
-debug (show visual debug information on the image)
-v (verbose)
```

```
-port # (port number when in server mode)
-load "fileToLoad" (load a video file)
-save "fileToSave" (save a video file)
-loop (play video file in loop mode)
-fs (full size display)
-bayer (acquire images in the bayer format)
-rgb (acquire images in RGB format; used by default)
-ball # (the color of the object of interest: 1 - orange 2 - yellow 3 - magenta
4 - cyan 5 - blue; if nothing is specified, orange is considered by default)
-cam # (camera type: 0 - Ethernet, 1 - USB, 2 - Firewire)
-ncf "newConfigFileName" (create a default configuration file for the type
of camera chosen)
-ns # (number of radial scan lines)
-step # (intra-scan line sparsity)
-p "polynomialFile" (load a file that contains the coefficients
of the size-distance function of the object of interest)
-v (verbose mode)
-tv (time verbose mode)
```

When the vision software runs in display and debug mode (*./omniVision -debug*), the following keys can be used:

```
1 - display the original image
2 - display the "index" image (grayscale)
3 - display the color segmented image (RGB)
4 - display the "reality of the robot" image (RGB)
p - pause image
f - freeze image
g - every hit on the g key displays only a frame
a - start the colormetric auto-calibration of the camera
d - display debug information on the original image
s - save the current image to the disk
u - update the configuration file
h - display the help manual on the original image
q - quit
```

D.2 Perspective Vision System

The command for running the software behind the perspective vision system is:

```
./frontVision  -cf configFileName,
```

where *configFileName* represents the name of the configuration file to be used. The software expects the following file *../config/frontParams.xml* that contain the intrinsic and extrinsic coefficients of the camera, that resulted from the process of camera calibration and that are needed for the triangulation process.

Other optional arguments that can be used are:

```
-h (help)
-nodisp (do not show image)
-server (accept connections from a client)
-debug (shows visual debug information on the image)
-v (verbose)
-port # (port number when in server mode)
-load "fileToLoad" (load a video file)
-save "fileToSave" (save a video file)
-loop (play video file in loop mode)
-bayer (acquire images in the bayer format)
-rgb (acquire images in RGB format; used by default)
-fs (full size display)
-ball # (the color of the object of interest: 1 - orange 2 - yellow 3 - magenta
4 - cyan 5 - blue; if nothing is specified, orange is considered by default)
-cam # (camera type: 0 - Ethernet, 1 - USB, 2 - Firewire)
-ncf "newConfigFileName" (create a default configuration file for the type
of camera chosen)
-step1 # (inter-scan line sparsity)
-step2 # (intra-scan line sparsity)
-p "polynomialFile" (load a file that contains the coefficients
of the size-distance function of the object of interest)
-v (verbose mode)
-tv (time verbose mode)
```

When the vision software runs in display and debug mode (*./frontVision -debug*), the following keys can be used:

```
1 - display the original image
2 - display the "index" image (grayscale)
3 - display the color segmented image (RGB)
4 - display the "reality of the robot" image (RGB)
p - pause image
f - freeze image
g - every hit on the g key displays only a frame.
a - start the colormetric auto-calibration of the camera
d - display debug information on the original image
s - save the current image to the disk
u - update the configuration file
h - display the help information on the original image
q - quit.
```

D.3 Kinect-based Vision System

The command for running the software of the Kinect-based vision system is:

```
./kinectVision -cf configFileName,
```

where $configFileName$ represents the name of the configuration file to be used. The software expects the following file *../config/KinectOnBoard.xml* that contains the calibration parameters of the Kinect camera.

Other optional arguments that can be used are:

```
-h (help)
-nodisp (do not show image)
-server (accept connections a client)
-debug (shows visual debug information on the image)
-v (verbose)
-port # (port number when in server mode)
-load "fileToLoad" (load a video file)
-save "fileToSave" (save a video file)
-loop (play video file in loop mode)
-ball # (the color of the object of interest: 1 - orange 2 - yellow 3 - magenta
4 - cyan 5 - blue; if nothing is specified, orange is considered by default)
-ncf "newConfigFileName" (create a default configuration file)
-step1 # (inter-scan line sparsity)
-step2 # (intra-scan line sparsity)
-p "polynomialFile" (load a file that contains the coefficients
of the size-distance function of the object of interest)
-v (verbose mode)
-tv (time verbose mode)
```

When the vision software runs in display and debug mode (*./kinectVision -debug*), the following keys can be used:

```
1 - show the original image
2 - show the "index" image (grayscale)
3 - show the color segmented image (RGB)
4 - show the "reality of the robot" image (RGB)
p - pause image
f - freeze image
g - every hit on the g key displays only a frame
d - display debug information on the original image
s - save the current image to the disk
u - update the configuration file
h - display the help information on the original image
q - quit
```

D.4 *calibVision* Tool

When any of the previous vision systems work in server mode, a remote client called **calibVision** can connect to this server with two different purposes:

- Remote color calibration;
- Receive information in real time about what a robot is seeing.

calibVision can be run as follows:

```
./calibVision -addr # -port $
```

where # represents the IP address of the server and $ the port number.

Other command line arguments that can be used:

```
-h (help)
-mask # (filename with a mask)
```

The following key interactions are supported:

```
i - request an image from the server
c - request the color ranges from the server
m - receive the mask from the server
x - receive the camera settings from the server
y - the trackbars and HSV histogram pop up
C - send the color ranges to the server
M - send the mask to the server
X - send the camera settings to the server
a - start the auto-calibration of the camera settings on the server side
e - manual calibration of the camera settings using sliders
h - opens the help menu in a different window
q - quit
```

Bibliography

[1] Shannon Fenn, Alexandre Mendes, and David Budden. Addressing the non-functional requirements of computer vision systems: A case study. *CoRR*, abs/1410.8623, 2014.

[2] Claudia Gönner, Martin Rous, and Karl F. Kraiss. Real-Time Adaptive Colour Segmentation for the RoboCup Middle Size League. In *Proceedings of RoboCup 2004: Robot Soccer World Cup VIII*, volume 3276, pages 402–409, Lisbon, Portugal, June 2005.

[3] Matthias Jngel, Jan Hoffmann, and Martin Ltzsch. A real-time auto-adjusting vision system for robotic soccer. In Daniel Polani, Andrea Bonarini, Brett Browning, and Kazuo Yoshida, editors, *7th International Workshop on RoboCup 2003 (Robot World Cup Soccer Games and Conferences)*, volume 3020, pages 214–225. Springer, 2004.

[4] Yasutake Takahashi, Walter Nowak, Thomas Wisspeintner, Yasutake Takahashi, Walter Nowak, and Thomas Wisspeintner. Adaptive Recognition of Color-Coded Objects in Indoor and Outdoor Environments RoboCup 2007: Robot Soccer World Cup XI. In Ubbo Visser, Fernando Ribeiro, Takeshi Ohashi, Frank Dellaert, Ubbo Visser, Fernando Ribeiro, Takeshi Ohashi, and Frank Dellaert, editors, *RoboCup 2007: Robot Soccer World Cup XI*, volume 5001 of *Lecture Notes in Computer Science*, chapter 6, pages 65–76. Springer Berlin / Heidelberg, 2008.

[5] Patricio Loncomilla and Javier Ruiz del Solar. Robust object recognition using wide baseline matching for robocup applications. In Ubbo Visser, Fernando Ribeiro, Takeshi Ohashi, and Frank Dellaert, editors, *RoboCup*, volume 5001 of *Lecture Notes in Computer Science*. Springer, 2007.

[6] H. Lu, H. Zhang, and Z. Zheng. Arbitrary ball recognition based on omni-directional vision for soccer robots. In *Proc. of RoboCup 2008*, 2008.

[7] E. R. Davies. *Machine Vision - Theory, Algorithms, Practicalities.* Elsevier, 3 edition, 2005.

[8] King Sun Fu, R. C. Gonzalez, and C. S. G. Lee, editors. *Robotics: Control, Sensing, Vision, and Intelligence.* McGraw-Hill, Inc., New York, NY, USA, 1987.

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

[10] USA DEPARTMENT OF DEFENSE. Technology readiness assessment (tra) guidance, 2011.

[11] Aldebaran Robotics official website. `http://www.aldebaran-robotics.com`, 2014. Last visited - September, 2014.

[12] Z. Zickler, T. Laue, O. Birbach, M. Wongphati, and M. Veloso. SSL-Vision: The Shared Vision System for the RoboCup Small Size League. RoboCup 2011, Istanbul, Turkey, 2011.

[13] A. J. R Neves, L. P. Reis, and A. Sousa. Real-time vision in the robocup - robotic soccer international competitions. In *Proc. of the ECCOMAS Thematic Conference on Computational Vision and Medical Image Processing, VIPImage 2009*, October 2009.

[14] 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, mar 2011.

[15] CMVision. `http://www.cs.cmu.edu/~jbruce/cmvision/`, 2014. Last visited - September, 2014.

[16] Adaptive Vision. `https://www.adaptive-vision.com/en/home/`, 2014. Last visited - September, 2014.

[17] CCV. `http://libccv.org/`, 2014. Last visited - September, 2014.

[18] Roborealm. `http://www.roborealm.com/index.php`, 2014. Last visited - September, 2014.

[19] Brett Browning and Manuela M. Veloso. Real-time, adaptive color-based robot vision. In *IROS*, pages 3871–3876. IEEE, 2005.

[20] Thomas Bayes. An essay towards solving a problem in the doctrine of chances. by the late rev. mr. bayes, f. r. s. communicated by mr. price, in a letter to john canton, m. a. and f. r. s. *Philosophical Transactions of the Royal Society*, 53:370–418, 1763.

[21] Paul Hough. Methods and means for recognizing complex patterns. 1962.

[22] R.O. Duda and P.E. Hart. Use of the hough transformation to detect lines and curves in pictures. Technical Report 36, AI Center, SRI International, Apr 1971. SRI Project 8259 Comm. ACM, Vol 15, No. 1.

[23] D. H. Ballard. Generalizing the hough transform to detect arbitrary shapes. In M. A. Fischler and O. Firschein, editors, *Readings in Computer Vision: Issues, Problems, Principles, and Paradigms*, pages 714–725. Kaufmann, Los Altos, CA., 1987.

[24] Ketill Gunnarsson, Fabian Wiesel, and Ral Rojas. The color and the shape: Automatic on-line color calibration for autonomous robots. In Ansgar Bredenfeld, Adam Jacoff, Itsuki Noda, and Yasutake Takahashi, editors, *RoboCup*, volume 4020 of *Lecture Notes in Computer Science*, pages 347–358. Springer, 2005.

[25] Stan Z. Li. *Markov Random Field Modeling in Image Analysis.* Springer Publishing Company, Incorporated, 3rd edition, 2009.

[26] H. Lu, Z. Zheng, F. Liu, and X. Wang. A robust object recognition method for soccer robots. In *Proc. of the 7th World Congress on Intelligent Control and Automation*, Chongqing, China, June 2008.

[27] Juan Fasola and Manuela M. Veloso. Real-time object detection using segmented and grayscale images. In *ICRA*, pages 4088–4093. IEEE, 2006.

[28] Mansour Jamzad, Abbas Hadjkhodabakhshi, and Vahab Mirrokni. Object detection and localization using omnidirectional vision in the robocup environment. *Scientia Iranica*, 14(6):599611, 2007.

[29] Francisco Martin and Manuela Veloso. Effective real-time visual object detection. *Progress in Artificial Intelligence*, 1(4):259–265, 2012.

[30] Ricardo Dodds, Luca Iocchi, Pablo Guerrero, and Javier Ruiz-del Solar. Benchmarks for robotic soccer vision. In Thomas Rfer, N.Michael Mayer, Jesus Savage, and Uluc Saranl, editors, *RoboCup 2011: Robot Soccer World Cup XV*, volume 7416 of *Lecture Notes in Computer Science*, pages 427–439. Springer Berlin Heidelberg, 2012.

[31] OpenCV. `http://docs.opencv.org`, 2014. Last visited - September, 2014.

[32] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing.* Addison-Wesley Longman Publishing Co., 2 edition, 2001.

[33] G.D. Hastings and A. Rubin. Colour spaces - a review of historic and modern colour models. *The South African Optometrist*, 71:133–143, 2012.

[34] G.W. Meyer and D.P. Greenberg. Perceptual color spaces for computer graphics. *Computer Graphics*, 14:254–261, 2012.

[35] George H. Joblove and Donald Greenberg. Color spaces for computer graphics. In *Proceedings of the 5th annual conference on Computer Graphics and Interactive Techniques*, pages 20–25, 1978.

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

[37] B.E. Bayer. Color imaging array, July 20 1976. US Patent 3,971,065.

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

[39] M. Tkalcic and J. F. Tasic. Color spaces - perceptual, historical and application background. *The IEEE Region 8 EUROCON 2003*, 1:304–308, 2003.

[40] P. Heinemann, F. Sehnke, Felix S., and A. Zell. Towards a calibration-free robot: The act algorithm for automatic online color training. pages 363–370, 2007.

[41] Yasutake Takahashi, Walter Nowak, and Thomas Wisspeintner. Adaptive recognition of color-coded objects in indoor and outdoor environments. In *RoboCup 2007: Robot Soccer World Cup XI*, Lecture Notes in Artificial Intelligence. Springer, 2007.

[42] G. Krawczyk, M. Goesele, and H Seidel. Photometric calibration of high dynamic range cameras. Research Report MPI-I-2005-4-005, Max-Planck-Institut für Informatik, Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany, April 2005.

[43] M. V. Shirvaikar. An optimal measure for camera focus and exposure. In *Proc. of the IEEE Southeastern Symposium on System Theory*, Atlanta, USA, March 2004.

[44] Navid Nourani-Vatani and Jonathan Roberts. Automatic camera exposure control. In *Proc. of the 2007 Australasian Conference on Robotics and Automation*, Brisbane, Australia, December 2007.

[45] K. J. AAström and T. Hägglund. *PID Controllers: Theory, Design, and Tuning.* Instrument Society of America, Research Triangle Park, NC, 2 edition, 1995.

[46] Gerd Mayer, Hans Utz, and Gerhard K. Kraetzschmar. Playing robot soccer under natural light: A case study. In *In 7th International Workshop on RoboCup 2003 (Robot World Cup Soccer Games and Conferences), Lecture Notes in Artificial Intelligence*, pages 238–249. Springer, 2004.

[47] B. Cunha, J. L. Azevedo, N. Lau, and L. Almeida. Obtaining the inverse distance map from a non-svp hyperbolic catadioptric robotic vision system. In *Proc. of the RoboCup 2007*, Atlanta, USA, 2007.

[48] Zhengyou Zhang. Flexible camera calibration by viewing a plane from unknown orientations. In *in ICCV*, pages 666–673, 1999.

[49] David A. Forsyth and Jean Ponce. *Computer Vision: A Modern Approach.* Prentice Hall, 1 edition, August 2002.

[50] Paul Beardsley, David Murray, and Andrew Zisserman. Camera calibration using multiple images. In G. Sandini, editor, *Computer Vision ECCV'92*, volume 588 of *Lecture Notes in Computer Science*, pages 312–320. Springer Berlin Heidelberg, 1992.

[51] Zhengyou Zhang. A flexible new technique for camera calibration. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(11):1330–1334, November 2000.

[52] Adrian Kaehler and Bradski Gary. *Learning OpenCV.* O'Reilly Media, 2013.

[53] J. E. Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4(1):25–30, 1965.

[54] Jack Bresenham. A linear algorithm for incremental digital display of circular arcs. *Commun. ACM*, 20(2):100–106, 1977.

[55] Junichi Nakamura. *Image Sensors and Signal Processing for Digital Still Cameras.* CRC Press, Inc., Boca Raton, FL, USA, 2005.

[56] Ron Kimmel. Demosaicing: image reconstruction from color ccd samples. *IEEE Transactions on Image Processing*, 1999.

[57] Henrique S. Malvar, Li wei He, and Ross Cutler. High-quality linear interpolation for demosaicing of bayer-patterned color images. In *Proceedings of the IEEE International Conference on Speech, Acoustics, and Signal Processing*, 2004.

[58] E. Chang, S. Cheung, and D. Y. Pan. Color filter array recovery using a threshold-based variable number of gradients. In *Proceedings of SPIE*, volume 3650, page 36, 1999.

[59] Lanlan Chang and Yap-Peng Tan. Hybrid color filter array demosaicking for effective artifact suppression. *J. Electronic Imaging*, 15(1):013003, 2006.

[60] Keigo Hirakawa, Student Member, and Thomas W. Parks. Adaptive homogeneity-directed demosaicing algorithm. *IEEE Trans. Image Processing*, 14:360–369, 2005.

[61] 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.

[62] IDS. `https://en.ids-imaging.com`, 2014. Last visited - September, 2014.

[63] Paulo Dias, Joo Silva, Rafael Castro, and Antonio J. R. Neves. Detection of aerial balls using a Kinect sensor. In *RoboCup 2014 Symposium*, Joao Pessoa, Brazil, July 25 2014.

[64] A. Trifan, A. J. R. Neves, B. Cunha, and N. Lau. A modular real-time vision system for humanoid robots. In *Proceedings of SPIE IS&T Electronic Imaging 2012*, January 2012.

[65] A. Trifan, A. J. R. Neves, B. Cunha, and N. Lau. A color-coded real-time vision system for a nao humanoid robot. In *Procedings of the 17th Portuguese Conference on Pattern Recognition, RecPad 2011*, October 2011.

[66] J. F. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6), November 1986.

[67] Jos B.T.M. Roerdink and Arnold Meijster. The watershed transform: Definitions, algorithms and parallelization strategies. *Fundam. Inf.*, 41(1,2):187–228, April 2000.

[68] A. Trifan, A. J. R. Neves, B. Cunha, and J. L. Azevedo. Uavision: a modular time-constrained library for soccer robots. In *Proc. of RoboCup 2014*, 2014.

[69] Antonio J. R. Neves, Alina Trifan, and Bernardo Cunha. Self-calibration of colormetric parameters in vision systems for autonomous soccer robots. In Sven Behnke, Manuela M. Veloso, Arnaud Visser, and Rong Xiong, editors, *RoboCup 2013: Robot Soccer World Cup XVII*, volume 8371 of *Lecture Notes in Computer Science*, pages 183–194. Springer, 2014.

[70] M. Petrou and P. Bosdogianni. *Image Processing the Fundamentals.* Wiley, 2004.

[71] H. Bay, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. *Computer Vision and Image Understanding (CVIU)*, 110(4):346–359, 2008.

[72] D. G. Lowe. Object recognition from local scale-invariant features. In *Proc. of the International Conference on Computer Vision*, September 1999.

[73] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[74] Microsoft Kinect official website. `http://www.microsoft.com/en-us/kinectforwindows/`, 2014. Last visited - September, 2014.

[75] UAVision. `http://sweet.ua.pt/an/uavision/`, 2015. Last visited - November 2015.