Technical University of Crete

School of Electronic and Computer Engineering

Visual Color and Field Line Recognition and Exploitation for the RoboCup Standard Platform League

Ioannis Liverios-Marinos

Chania, October 2014

ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΝΙΚΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Οπτική Αναγνώριση και Αξιοποίηση Χρωμάτων και Διαγράμμισης Γηπέδου για το Πρωτάθλημα **Standard Platform** του **RoboCup**

Ιωάννης Λιβέριος-Μαρίνος

Χανιά, Οκτώβριος 2014

”The only thing worse than being blind is having sight but no vision.”

(Helen Keller)

Abstract

In order to complete complex tasks, both humans and robots are bound by one of the most important senses, the visual sense, which helps them perceive the state of the environment that surrounds them. Robotic soccer, known as RoboCup, represents a complex, stochastic, real-time, multi-agent, competitive domain for autonomous robots. In such domains, the ability to understand the environment is critical for the accomplishment of the assigned task and is required for a range of activities, such as locomotion, coordination, and decision making. This thesis focuses on two specific aspects of robot visual perception, color and field line recognition, in order to complement prior work on this problem by our RoboCup team "Kouretes". In RoboCup, the objects of interest are characterized by unique colors (orange ball, green field, yellow goalposts, white lines) and their recognition relies on the correct identification of image areas corresponding to the same color. However, the problem of color recognition is highly affected by the environment illumination conditions, as well as the robot's camera settings. In this thesis, we propose a new approach to color recognition, which relies on modeling off-line the signatures of the target colors in the color space under different illuminations using density estimation with Gaussian distributions and dynamically identifying and using the correct models by exploiting a metric on the most common color in the RoboCup environment (green). On the problem of field line recognition, we focus on identifying a variety of field line landmarks (straight lines, center circle, corners, T-lines), which are useful for localization. This is accomplished by searching for white pixels in the camera images, selectively keeping those that can be part of a line, and then identifying each type of line using curve fitting techniques. For each recognized line landmark, we use geometry and projection techniques to estimate its distance and bearing with respect to the robot. Our work contributes an off-line tool for color recognition and a real-time module for field line recognition appropriate for on-board execution on the Aldebaran Nao humanoid robots. The proposed methods perform reliably in most cases, failing only in extreme cases, which are typically infrequent during RoboCup games.

Acknowledgements

First of all, I would like to thank my advisor Michail G. Lagoudakis for the trust that he showed in me from the begining, his inspiration and his guidance through all of this thesis and my years as a undergraduate student.

I would also like to thank our Team Kouretes (V. Michelioudakis, N. Pavlakis, N. Kargas, S. Piperakis) and N. Kofinas and Manolis Orfanoudakis for their great help, guidance and suggestions during the implementation of my work.

Next, I would like to thank my friends for all these great years and moments we had. I am lucky to have met you and i dont know how i would have made it through all these years without you: S. Angelaki, K. Karalas, A. Koukounya, A. Mavrommatis, S. Maropaki, V. Michelioudakis K. Pechlivanis, S. Sourlantzis, G. Tabakakis, A. Chorianopoulou.

A special thank you goes to my wonderful parents and family for their love and support over the years even though they were a bit afraid of me giving it up...

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

The RoboCup Competition is an international annual aggregation of robotic competitions which intends to promote Robotics and Artificial Intelligence (AI) research. RoboCup Soccer constitutes the main RoboCup division and focuses on the game of soccer. The research goals in RoboCup Soccer concerns multi-robot and multi-agent systems in dynamic adversarial environments and all the participating teams have to find real-time solutions to some of the most difficult problems in robotics, such as perception, cognition, action, and coordination. In the Standard Platform League (SPL) all teams use identical robots (standard platform), therefore they are not concerned with hardware improvements, and they only concentrate on software development. Currently, the chosen SPL platform is the Aldebaran Nao humanoid robot.

Software development for robots competing in the RoboCup SPL essentially aims at developing autonomous agents. An autonomous robotic agent is a system that continuously perceives its environment through the robotic sensors, analyzes the percept sequence using various AI techniques, and takes actions through the robotic actuators with the goal of maximizing a utility function. The central problems of an autonomous robotic agent include environment perception, robot localization, robotic mapping, path planning, decision making under uncertainty, and learning. In order to achieve any of those or other goals a robot must first have a good belief of his environment which is inextricably linked with his Vision. This thesis studies the problem of robotic vision to recognize all colors in need of by giving a more efficient way to do this as well as recognizing all lines and there types on the field.

To illustrate the importance of the robots sight, consider a scenario where a human is blindfolded or can only recognize half colors and tries to do a simple task. Same level of difficulty applies when a robot is blind or it can't fully recognize all colors on its environment. The human eye, unlike the robots camera, can automatically adjust its pupil depending on the light conditions, so when we enter a dark room while previously being in a brighter one, our pupil has the ability to expand in order to gather more light and recognize more edges and shapes of all objects in its vision. On the contrary the robots camera need somehow to auto adjust its camera's parameters to the present light conditions which is not as simple as it seems and we will explain later, more detailed, why. Another problem of robotic vision is that most robots, like our Nao v.3, have only one active camera at any given moment and thus it can't have a depth perspective, as for that we need a second camera. Although there are some solutions for this problem, they are not so effective and unfortunately we will have to deal with it during this whole thesis.

1.1 Thesis Contribution

This thesis contributes on the development of a mechanism that addresses the problem of color and line recognition, during a robotic soccer game, using Gaussian Distributions for the first problem and pixel classification and categorization for the latter. More specifically, regarding color recognition, we created a dynamic off-line tool that classifies all colors, using Gaussian Distributions representation on the HSL cube, for each color in a major number of light conditions. In order to represent all colors through Gaussian Distributions, it is strictly necessary that we work on the HSL or HSI cube model and not on those like RGB or YUV formats. HSL and HSI are useful, basically because in the HSL/HSI formats we have Hue, Saturation and Luminance-Lightness/Intensity as parameters. This provides us the ability of ignore one of those three, we choose for our own purpose of light conditions changes, to ignore the L parameter, so that we will be able to represent all colors on Gaussian Distributions taking into consideration only two, the H and S parameters. After having one distribution for each color in many light conditions we produce a color table for the robot which now can recognize all colors in its environment. If the light conditions change we produce a new color table off-line and supply the robot with it.

Regarding the line recognition and classification we suggest a simple yet efficient way which takes advantage of the least square fitting theorem and produces a solution for the line type (straight, circle, corner). More specifically all lines, curved or not, can be represented by a second degree equation in which the "a" factor represents the line type. If "a" is near, or exactly zero, then the line is considered a straight one. On the other hand, if it's a positive or a negative number, it's considered a curved line which is facing up or down respectively, called parabola. Thus we can recognize all line types with a simple yet efficient way. So where lies the difficulty of this thesis? The problem, to which we suggest an effective solution in this thesis, is that considering the bad picture quality (640x480), the noise that the camera produces, the poor processor power as well as the obstacles that we can find in our environment, it requires a good, yet fast, algorithm that discards all useless white pixels (lines color) and keeps only those that we are in need of.

The benefits of the proposed methods are numerous, yet the most important one is that the robots can recognize all line types and use them as landmarks to provide a better belief of their current position. Regarding color recognition, we now have a more effective way of recognizing all colors and we have eliminated the human element from this process, which makes it less heuristic and faster on its application.

1.2 Thesis Outline

Chapter 2 describes the RoboCup competition, the Standard Platform League (SPL), the Aldebaran Nao humanoid robot, our SPL team Kouretes, our software architecture Monas, and our communication framework Narukom. Furthermore, it provides basic background information about Gaussian Distributions utility, as well as their use on the HSL color model and for the least square fit algorithm as well as for the center approach of a circle which will be vital on the line recognition algorithm. In Chapter 3 we define the problem of Color recognition using heuristic methods and we discuss the significance of developing an efficient and effective mechanism for addressing the problem with Gaussian Distributions. In Chapter 4 we describe our approach in detail for both color and line recognition separated. In Chapter 5 we briefly present our implementation, including our choices for off-line execution. In Chapter 6 we present a couple of scenarios demonstrating the effectiveness and the efficiency of our team color and line recognition. Finally, in Chapter 7 we propose directions for future work and conclude.

Chapter 2

Background

2.1 RoboCup

RoboCup, an abbreviation of ”Robot Soccer World Cup”, is an international annual competition which intends to promote robotics and artificial intelligence research. The founding father of RoboCup, Professor Alan Mackworth, inspired the idea of building a robot to play a soccer game autonomously in 1992. One year later, Hiroaki Kitano [1] and his research group decided to launch a novel robotic competition. Finally, in 1997 the actual establishment of the International RoboCup Federation occurred. The ambitious goal of the RoboCup Initiative is stated as follows:

> “By mid-21st 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.”

All the teams participating in RoboCup have to find real-time solutions to some of the most difficult problems in robotics (perception, cognition, action, coordination) and apply their approaches on the various leagues of the four RoboCup divisions (RoboCup Soccer, RoboCup Rescue, RoboCup@Home, Robocup Junior). Until today, noteworthy progress has been made in advancing the state-of-the-art technology, while the number of the participating researchers who aim to fulfill the initial challenge is constantly growing.

2.1.1 Standard Platform League

RoboCup Soccer constitutes one of the four RoboCup divisions and focuses mainly on the game of soccer, where the research goals concern cooperative multi-robot and multi-agent systems in dynamic adversarial environments. All robots in this division are fully autonomous. RoboCup Soccer consists of five different leagues (Humanoid, Middle Size, Simulation, Small Size, and Standard Platform). In the Standard Platform League (SPL) all teams use identical robots (standard platform). Currently, the chosen SPL platform is the Aldebaran Nao humanoid robot, therefore the teams concentrate only on software development. The participating teams are prohibited to make any changes to the hardware of the robot, meaning that off-board sensing or processing systems are not allowed. The use of directional, as opposed to omnidirectional, vision forces a trade-off of vision resources between self-localization, ball localization, player identification, and obstacle detection. The robots are completely autonomous and no human intervention from team members is allowed during the games. The only interaction of the robots with the "outer human world" is the reception of data from the Game Controller, a computer that broadcasts information about the state of the game (score, time, penalties, etc.).

The SPL games as of 2013 are conducted on a $9m \times 6m$ soccer field which consists of a green carpet marked with white lines and two yellow goals (Figure 2.1). The ball is an orange street hockey ball. Each team consists of five robots, one goal keeper, and four field players. The robot players are distinguished by colored jersey shirts, blue for one team and red for the other. The total game time is 20 minutes divided in two halves; each half lasts 10 minutes. During the 10-minutes half-time break, teams have to switch field sides and jerseys and only during this time is it permitted to change robots, change programs, etc. The complete rules of the SPL games are stated in detail in the RoboCup Standard Platform League (Nao) Rule Book [2], which is annually updated with enhancements and additional challenging requirements that propel the general progress of the league.

2.1.2 Aldebaran Nao Humanoid Robot

The current hardware platform which all SPL teams are obliged to work with is Nao, an integrated, programmable, medium-sized humanoid robot developed by Aldebaran Robotics in Paris, France. Project Nao [3] started in 2004. In August 2007 Nao officially

Figure 2.1: Standard Platform League at RoboCup 2013 in Eindhoven, Netherlands

replaced Sony's Aibo quadruped robot in the RoboCup SPL. In the past few years Nao has evolved over several designs and several versions.

Nao (version V3.3) [4] is a 58cm, 5kg humanoid robot (Figure 2.2). The Nao robot carries a fully capable computer on-board with an x86 AMD Geode processor at 500 MHz, 256 MB SDRAM, and 2 GB flash memory running an Embedded Linux distribution. It is powered by a 6-cell Lithium-Ion battery which provides about 30 minutes of continuous operation and communicates with remote computers via an IEEE 802.11g wireless or a wired ethernet link.

Nao RoboCup edition has 21 degrees of freedom; 2 in the head, 4 in each arm, 5 in each leg, and 1 in the pelvis (there are two pelvis joints which are coupled together on one servo and cannot move independently). Nao, also, features a variety of sensors and transmitters. Two cameras are mounted on the head in vertical alignment providing non-overlapping views of the lower and distant frontal areas, but only one is active each time and the view can be switched from one to the other almost instantaneously. Each camera is a 640 x 480 VGA device operating at 30fps. The native colorspace provided by the cameras is the YUV422. Four sonars (two emitters and two receivers) on the chest allow Nao to sense obstacles in front of it. In addition, the Nao has a rich inertial unit, with

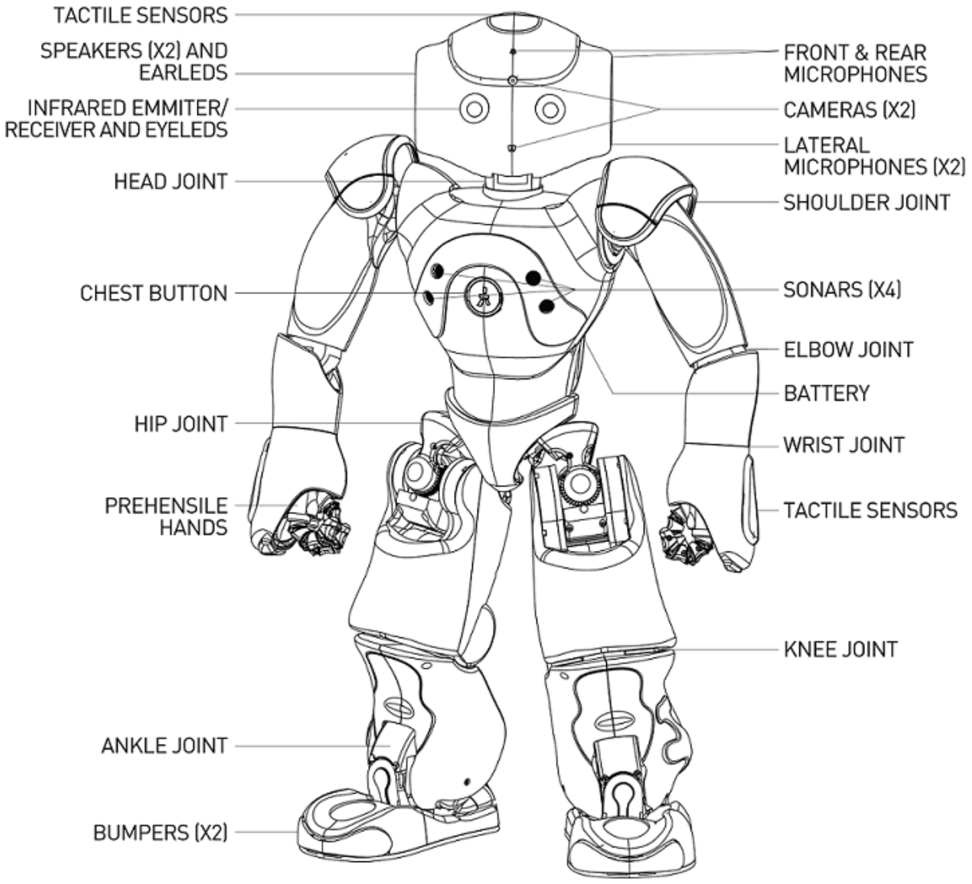


Figure 2.2: Aldebaran Nao robot (v3.3, academic edition) and its components

one 2-axis gyroscope and one 3-axis accelerometer, in the torso that provides real-time information about its instantaneous body movements. Two bumpers located at the tip of each foot are simple ON/OFF switches and can provide information on collisions of the feet with obstacles. Finally, an array of force sensitive resistors on each foot delivers feedback of the forces applied to the feet, while encoders on all servos record the actual values of all joints at each time.

Aldebaran Robotics has equipped Nao with both embedded and desktop software to be used as a base for further development (Figure 2.3). The embedded software, running on the motherboard located in the head of the robot, that the company provides includes an embedded GNU/Linux distribution and NAOqi, the main proprietary software that runs on the robot and controls it. Nao's desktop software includes Choregraphe, a visual programming application which allows the creation and the simulation of animations and

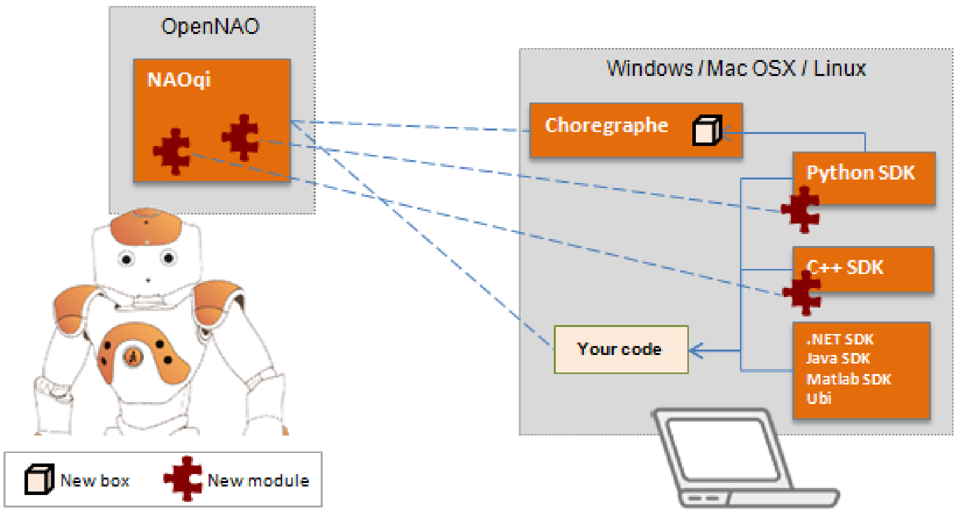


Figure 2.3: Embedded and desktop software for the Nao robot

behaviors for the robot before the final upload to the real Nao, and Telepathe which provides elementary feedback about the robot's hardware and a simple interface to accessing its camera settings. As far as the NAOqi framework is concerned, it is cross-platform, cross-language, and provides introspection which means that the framework knows which functions are available in the different modules and where. It provides parallelism, resources, synchronization, and events. NAOqi, also, allows homogeneous communication between different modules (motion, audio, video), homogeneous programming, and homogeneous information sharing. Software can be developed in C++, Python, and Urbi. The programmer can state which libraries have to be loaded when NAOqi starts via a preference file called `autoload.ini`. The available libraries contain one or more modules, which are typically classes within the library and each module consists of multiple methods (Figure 2.4).

2.2 RoboCup SPL Team Kouretes

Team Kouretes is the first and currently the only RoboCup SPL team founded in Greece, hosted in the Intelligent Systems Laboratory of the School of Electronic and Computer Engineering at the Technical University of Crete. Kouretes started developing their own robotic software framework in 2008 and the code is constantly developed and maintained

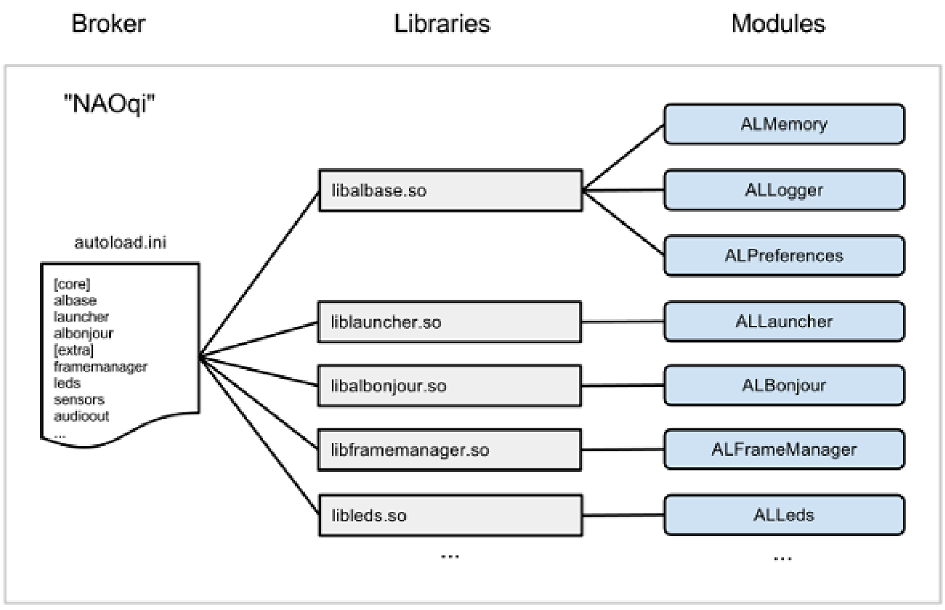


Figure 2.4: The NAOqi process

ever since. The team's publicly-available code repository includes a custom software architecture, a custom communication framework, a graphical application for behavior specification, and modules for object recognition, state estimation, localization, obstacle avoidance, behavior execution, and team coordination, which are briefly described below.

The team participates in the main RoboCup competition since 2006 in various soccer leagues (Four-Legged, Standard Platform, MSRS, Webots), as well as in various local RoboCup events (German Open, Mediterranean Open, Iran Open, RC4EW, RomeCup) and RoboCup exhibitions (Athens Digital Week, Micropolis, Schoolfest). Distinctions of the team include: 2nd place in MSRS at RoboCup 2007; 3rd place in SPL-Nao, 1st place in SPL-MSRS, among the top 8 teams in SPL-Webots at RoboCup 2008; 1st place in RomeCup 2009; 6th place in SPL-Webots at RoboCup 2009; 2nd place in SPL at RC4EW 2010; and 2nd place in SPL Open Challenge Competition at RoboCup 2011 (joint team Noxious-Kouretes). Recently, the team participated in RoboCup German Open 2012 in Magdeburg, in RoboCup Iran Open 2012 in Tehran, in RoboCup 2012 in Mexico City, in AutCup 2012 in Tehran, in RoboCup Iran Open 2013 in Tehran and in RoboCup 2013 in Eindhoven, Netherlands (Figure 2.5).

Figure 2.5: Team Kouretes 2014

2.2.1 Monas Software Architecture

Monas [5] is a flexible software architecture which provides an abstraction layer from the hardware platform and allows the synthesis of complex robot software as XML-specified Monas modules, Provider modules, and/or Statechart modules. Monas modules, the so-called agents, focus on specific functionalities and each one of them is executed independently at any desired frequency completing a series of activities at each execution. The base activities, that an agent may consist of, are described briefly below:

- `Vision` [6] is a light-weight image processing method for humanoid robots, via which Kouretes team accomplishes visual object recognition. The vision module determines the exact camera position in the 3-dimensional space and subsequently the view horizon and the sampling grid, so that scanning is approximately uniformly projected over the ground (field). The identification of regions of interest on the pixels of the sampling grid follows next utilizing an auto-calibrated color recognition scheme. Finally, detailed analysis of the identified regions of interest seeks potential matches for corresponding target objects. These matches are evaluated and filtered by several heuristics, so that the best match (if any) in terms of color, shape, and size for a target object is finally extracted. Then, the corresponding objects are returned as perceived, along with an estimate of their current distance and bearing.

- `LocalWorldState` [7] is the activity which used to realize Monte-Carlo localization (Particle Filters - PFs) and recently switched to using an Extended Kalman Filter (EKF) [8]. The belief of the robot is a probability distribution over the 3-dimensional space of coordinates and orientation (x, y, θ) represented approximately using a population of particles in the case of PFs or using a 3-dimensional Gaussian distribution in the case of EKF. Belief update is performed using an odometry motion model for omnidirectional locomotion and a landmark sensor model for the goalposts (landmarks). The robot's pose is estimated as the pose of the particle with the highest weight (PFs) or the mean (highest probability) of the Gaussian distribution (EKF).

- `SharedWorldModel` [9] is the activity that combines the local beliefs of all robots to create a common and shared estimation of the current state of the world consistent with these local beliefs. In order to generate this information, an Extended Kalman Filter (EKF) is employed with appropriate state transition and observation models, applying linearization where needed.

- `PathPlanning` [10] is the activity which accomplishes path planning with obstacle avoidance by first building a local, polar, obstacle occupancy map, which is updated constantly with real-time sonar information, taking into consideration the robot's locomotion. Afterwards, an A* search algorithm is used for path planning, the outcome of which suggests an obstacle-free path for guiding the robot to a desired destination. The way-points of the planned path are finally translated into walk commands to guide the robot along the path.

- `Behavior` is the activity which implements the desired robotic behavior. It operates on the outputs of the `Vision`, `LocalWorldState`, and `SharedWorldModel` activities and decides which one is the most appropriate action to be executed next (walk, kick, etc.). Locomotion actions are passed to the `PathPlanning` activity for obstacle-free navigation, while motion actions are sent to the `MotionController` activity for execution. Our mechanism for team coordination and planning proposed in this thesis was integrated into this activity. Details will be provided in Chapters 4 and 5.

- `HeadController` manages the movements of the robot head (camera).
- `MotionController` [11] is used for managing and executing robot locomotion commands and special actions.
- `RobotController` handles external signals on the game state.
- `LedHandler` controls the robot LEDs (eyes, ears, chest button, feet).

Provider modules accomplish the complete decoupling of the robotic hardware by collecting and filtering measurements from the robot sensors and cameras and forming them as messages in order to be utilized as input data by any interested Monas agents. Each provider module can be executed independently and at any desired frequency.

Custom Forward and Inverse Kinematics [12, 13], designed specifically for the NAO humanoid robot, have been implemented as an independent software library optimized for speed and efficiency. The library is currently being used in other team projects, such as omni-directional walk engine and dynamic kick engine.

Statechart modules, which offer an alternative intuitive graphical specification of robot behavior, have also been integrated into Monas [14]. Kouretes Statechart Engine (KSE) [15, 16] is our own graphical tool for designing and editing statecharts for robot behavior. Statecharts are automatically transformed into code and are executed on-board using a generic multi-threaded statechart engine, which provides the required concurrency and meets the real-time requirements of the activities on each robot.

KMonitor [17] is our own debugging tool created specifically for the Monas architecture that takes advantage of the modularity of Kouretes code and helps in finding errors or verifying that newly implemented features work correctly. It also allows for the easy creation of colortables, the transmission of remote commands over the network, etc.

2.3 The Eye: Modeling a Familiar Sensor

Computer vision is a diverse and relatively new field of study. In a nutshell, it is concerned with the extraction of information from images. Therefore, it is closely related to the physics of digital image acquisition and digital image representation and is mainly concerned with the signal processing, image processing, and image analysis to guide the extraction of useful information from digital images.

2.3.1 Biology of Vision

The eye The visual system is the part of the central nervous system which gives organisms the ability to process visual detail, as well as enabling the formation of several non-image photo response functions. It detects and interprets information from visible light to build a representation of the surrounding environment. The visual system carries out a number of complex tasks, including the reception of light and the formation of monocular representations; the buildup of a nuclear binocular perception from a pair of two dimensional projections; the identification and categorization of visual objects; assessing distances to and between objects; and guiding body movements in relation to visual objects. The psychological process of visual information is known as visual perception, a lack of which is called blindness. Non-image forming visual functions, independent of visual perception, include the pupillary light reflex (PLR) and circadian photoentrainment [18]. However, from the compound eyes of the ordinary house fly, to the acute vision of predatory birds all higher organisms that have organs capable of spatially perceiving light share some basic fundamental elements:

- `diaphragm:` a means of selectively collecting light into the eye with a specific directional sensitivity (i.e. light coming from a particular direction).
- `lens:` a fixed or adjustable assembly that focuses light from the diaphragm onto the photosensitive tissue using the principle of refraction
- `photo-receptors:` a collection of biological tissue that is capable of detecting light and transforming it into electrical signals send to the brain through a neural pathway.

Human vision, appearing familiar and simple to an ordinary human, consists of elements that map comfortably to the above list. The human pupil is an excellent light diaphragm, that allows the brain to control the amount of light entering the eye by expanding and shrinking according to the environment. The lens, located behind the pupil, provide for consistent viewing acuity of objects, regardless of their distance from the viewer. The retina with its photosensitive cells provides for the perception of the light that passes into the eye, and is effectively the mechanism for forming the perception of intensity, color, and acuity of the world around us into an image. The result is a highly sophisticated organ, which combined with an extraordinary complex processing machine, the brain, produces a wealth of information for the world around us. From the simple detection of light to the perception of shapes and movement, either under direct sunlight or in the darkness of night, the brain cannot be matched by any machine the human has been able to conceive and manufacture.

Binocular vision Binocular vision is a delicate procedure that takes place ”behind the scenes” in all two-eyed organisms, and provides for what appears to be an intrinsic ability for us, namely the perception of depth, using the combined processing of the images from both eyes by the human brain. This processing, uses binocular disparity, a phenomenon related to the subtle differences in the images captured by the eyes due to their different positions in relation to the observed scene. By recognizing an object observed by both eyes, the brain also estimates the position from that object in space [19]

Visual Acuity The human retina is a very complex sensor, allowing the detection of color using three basic photo-receptor cell types (cone cells), each one being sensitive to a different (but not mutually exclusive) range of wavelengths, with varying levels of sensitivity [19]. It also provides a color-less imaging pathway using rod cells, which function better in low-light conditions and provide a color oblivious ”night vision” image. Therefore, all colors are reduced to three sensory quantities, called the tristimulus values [19]. This is not the only configuration in the animal kingdom: most birds’ eyes contain a fourth colored photo-receptor cell type, sensitive to the near-ultraviolet spectrum, allowing them to see the world in a way unfamiliar to us. Many nocturnal creatures have extremely better performing vision in low light conditions. An interesting detail about our vision, is that although our brain perceives the entire world as a continuously coherent image with the same acuity and color vividness, our retinas are only able to provide this fine image only in the very center of the visual field, i.e. when we look directly at an object. The central area of the retina, named fovea [19], contains only cones in maximal density and is highly effective in regular/intense lighting. The fovea corresponds to only a small amount of our field of view, with the vast majority of the images we see at a given moment being provided by the peripheral retina, which is a coarsely populated area. This is the reason humans unconsciously look directly at the object they intent to observe, so as to provide images with maximal clarity for the brain to interpret. It is merely the brain, filling in the gaps of the environment from previous observations, the real reason we have such detailed views of the surroundings in our perception.

2.4 Color Representations

2.4.1 HSL/I Background

HSL and HSV are the two most common cylindrical-coordinate representations of points in an RGB color model. Developed in the 1970s for computer graphics applications, HSL and HSV are used today in color pickers, in image editing software, and less commonly in image analysis and computer vision. The two representations rearrange the geometry of RGB in an attempt to be more intuitive and perceptually relevant than the cartesian (cube) representation, by mapping the values into a cylinder loosely inspired by a traditional color wheel. The angle around the central vertical axis corresponds to "hue" and the distance from the axis corresponds to "saturation". These first two values give the two schemes the 'H' and 'S' in their names. The height corresponds to a third value, the system's representation of the perceived luminance in relation to the saturation. Perceived luminance is a notoriously difficult aspect of color to represent in a digital format (see disadvantages section), and this has given rise to two systems attempting to solve this issue: HSL (L for lightness) and HSV or HSB (V for value or B for brightness) [20].

2.4.2 HSL: the way it works

HSL and HSV are both cylindrical geometries, with hue, their angular dimension, starting at the red primary at 0°, passing through the green primary at 120° and the blue primary at 240°, and then wrapping back to red at 360°. In each geometry, the central vertical

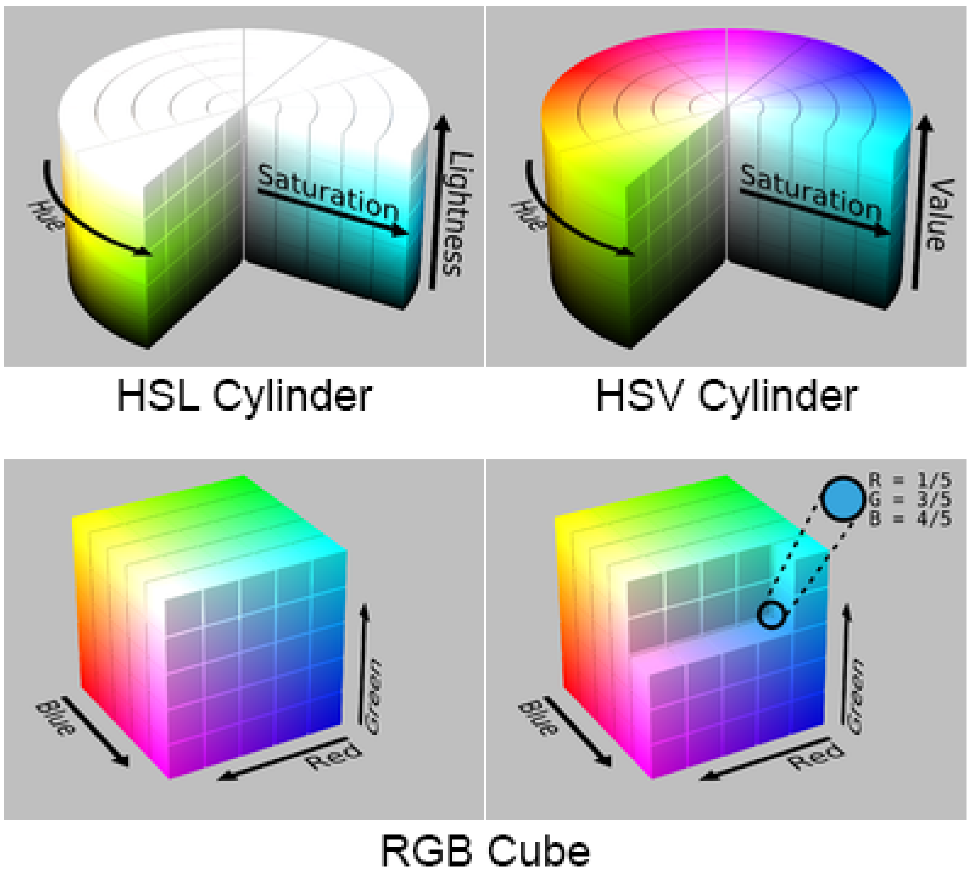


Figure 2.6: HSL and HSV Cylinders with RGB Cube

axis comprises the neutral, achromatic, or gray colors, ranging from black at lightness 0 or value 0, the bottom, to white at lightness 1 or value 1, the top. In both geometries, the additive primary and secondary colors - red, yellow, green, cyan, blue, and magenta - and linear mixtures between adjacent pairs of them, sometimes called pure colors, are arranged around the outside edge of the cylinder with saturation 1; in HSV these have value 1 while in HSL they have lightness $\frac{1}{2}$. In HSV, mixing these pure colors with white - producing so-called tints - reduces saturation, while mixing them with black - producing shades - leaves saturation unchanged. In HSL, both tints and shades have full saturation, and only mixtures with both black and white - called tones - have saturation less than 1.(Figure 2.6) [20].

On the other hand RGB color space is any additive color space based on the RGB color model. A particular RGB color space is defined by the three chromaticities of the red, green, and blue additive primaries, and can produce any chromaticity that is the triangle defined by those primary colors. The complete specification of an RGB color space also requires a white point chromaticity and a gamma correction curve. The RGB color model is represented by a cube instead of a cylinder like HSL and HSV which makes it vital to have all 3 dimensions in order to describe a color. In this thesis, as it will be explained later, having 3 dimensions is not acceptable, thus we use the HSL model [21].

2.4.3 Gaussian Distributions

In probability theory, the normal (or Gaussian) distribution is a very commonly occurring continuous probability distribution-a function that tells the probability that any real observation will fall between any two real limits or real numbers, as the curve approaches zero on either side. Normal distributions are extremely important in statistics and are often used in the natural and social sciences for real-valued random variables whose distributions are not known. The normal distribution is immensely useful because of the central limit theorem, which states that, under mild conditions, the mean of many random variables independently drawn from the same distribution is distributed approximately normally, irrespective of the form of the original distribution: physical quantities that are expected to be the sum of many independent processes (such as measurement errors) often have a distribution very close to normal. Moreover, many results and methods (such as propagation of uncertainty and least squares parameter fitting) can be derived analytically in explicit form when the relevant variables are normally distributed.

The Gaussian distribution is sometimes informally called the bell curve. However, many other distributions are bell-shaped (such as Cauchy's, Student's, and logistic). The terms Gaussian function and Gaussian bell curve are also ambiguous because they sometimes refer to multiples of the normal distribution that cannot be directly interpreted in terms of probabilities.

A normal distribution is:

$$f(x, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

The parameter μ in this definition is the mean or expectation of the distribution (and also its median and mode). The parameter μ is its standard deviation; its variance is therefore σ^2. A random variable with a Gaussian distribution is said to be normally distributed and is called a normal deviate.

If μ=0 and σ= 1, the distribution is called the standard normal distribution or the unit normal distribution, and a random variable with that distribution is a standard normal deviate.

The normal distribution is the only absolutely continuous distribution all of whose cumulants beyond the first two (i.e., other than the mean and variance) are zero. It is also the continuous distribution with the maximum entropy for a given mean and variance.

The normal distribution is a subclass of the elliptical distributions. The normal distribution is symmetric about its mean, and is non-zero over the entire real line. As such it may not be a suitable model for variables that are inherently positive or strongly skewed, such as the weight of a person or the price of a share. Such variables may be better described by other distributions, such as the log-normal distribution or the Pareto distribution [22].

2.5 Lines and Curves

2.5.1 Mathimatical Equations for Lines and Curves

For our second subject in this thesis we suggest a way, for the first time in our team "Kouretes", to recognize and categorize all lines in a socer field in the present RoboCup conditions [23] based on George Georgakis Thesis called "Field Landmark Recognition and Localization for the Robotstadium Online Soccer Competition" [24]. In order to conceive this idea we must first understand the geometry models and equations that it's based on. First of all it's known that we can estimate if a line is straight or curved and thus its type by its function type and the "Least-Squares Fit" theorem.

2.5.2 Least-Squares Fit

The method of least squares [25] is a mathematical procedure for finding the best-fitting curve to an unknown function, when only a finite set of points (values) of the function is known. It determines the form of an unknown equation that best describes our data and therefore its' most important application is in data fitting. It does so by solving

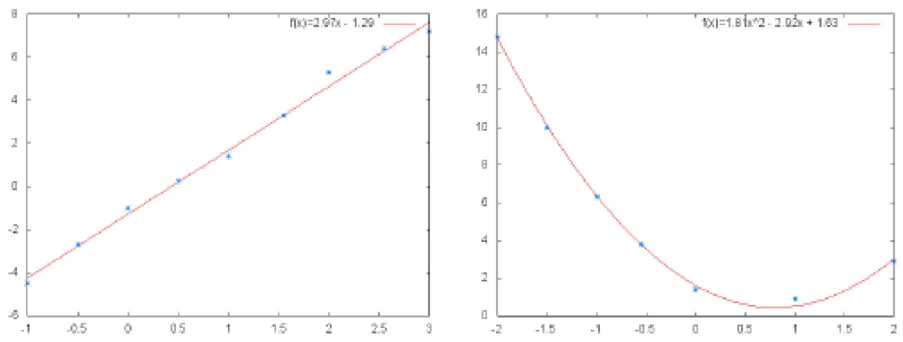


Figure 2.7: Linear (left) and polynomial (right) least-squares fitting

an undetermined system, i.e. sets of equations where there are more equations than unknowns. The solution minimizes the sum of the squares of the residuals (differences) between the fitting curve and the data points. We will examine two parametric models of curve equations used in least-squares problems:

$$y = C_0x^2 + C_1x + C_2$$

where x is an independent variable, y is a dependent variable and C_0, C_1, and C_2 are the parameters of the model. The goal of least squares is the determination of these parameters. Each parameter describes a different aspect of our curve's behavior. In the polynomial model, parameter C_0 determines the curvature, C_1 the slope, and C_2 the shifting of the curve. Therefore, if C_0 is 0, or nearly 0, then our model describes a straight line. Suppose that the data points are (x_1, y_1), (x_2, y_2), ..., (x_n, y_n). Then the linear system we have to solve in order to find the best parameters of our curve equation is:

$$\begin{pmatrix} \sum_{i=1}^n x_i^4 & \sum_{i=1}^n x_i^3 & \sum_{i=1}^n x_i^2 \\ \sum_{i=1}^n x_i^3 & \sum_{i=1}^n x_i^2 & \sum_{i=1}^n x_i \\ \sum_{i=1}^n x_i^2 & \sum_{i=1}^n x_i & \text{n} \end{pmatrix} \begin{pmatrix} c_2 \\ c_1 \\ c_0 \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^n x_i^2 y_i \\ \sum_{i=1}^n x_i y_i \\ \sum_{i=1}^n y_i \end{pmatrix}$$

Figure 2.7 shows examples of least-squares data fitting

2.5.3 Find the center of a circle

Given three points A,B,C on the plane, this method [26] is used to determine the center point and the radius of a circle that passes through all three points (Figure 2.8). Let us assume that the three points' coordinates are A(x_A, y_A),B(x_B, y_B),C(x_C, y_C). We can

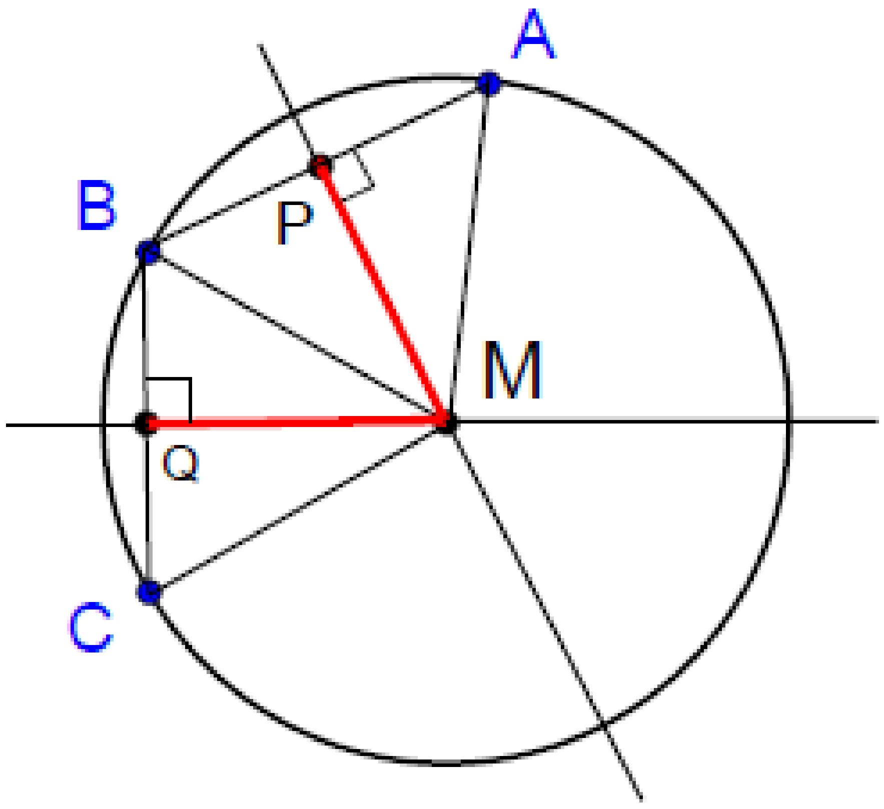


Figure 2.8: Points A,B,C,M and lines QM and PM

form two lines which pass from our given points; line l_1 passes from A and B, and line l_2 passes from B and C. These two lines have the following equations:

$$y_1 = m_1(x - x_A) + y_A$$
$$y_2 = m2(x - x_B) + y_B$$

where m_1, m_2 are the slopes of the two lines and they are given by:

$$m_1 = \frac{y_B - y_A}{x_B - x_A}$$
$$m_2 = \frac{y_C - y_B}{x_C - x_B}$$

Then, we can form two more lines that pass through the midpoints of line segments AB and BC and are perpendicular to them. The intersection point of these two lines will be the center point M of the circle. Their equations are:

$$
\begin{aligned}
y_{PM} &= -\frac{1}{m_1}(x - \frac{x_A + x_B}{2}) + \frac{y_A + y_B}{2} \\
y_{QM} &= -\frac{1}{m_2}(x - \frac{x_B + x_C}{2}) + \frac{y_B + y_C}{2}
\end{aligned}
$$

Now, we can find the coordinates of the center point M:

$$
\begin{aligned}
x_M &= \frac{m_1 m_2 (y_A - y_C) + m_2(x_A + x_B) - m_1(x_B + x_C}{2(m_2 - m_1)} \\
y_M &= -\frac{1}{m_2}(x_M - \frac{x_B + x_C}{2}) + \frac{y_B + y_C}{2}
\end{aligned}
$$

In order to find y_M we simply substitute x_M into one of the equations of the perpendicular lines and solve for y. The radius of the circle is equal to the euclidean distance of point M to any of the given points A;B;C. For better understanding of the solution, Figure 2.8 illustrates the aforementioned points and lines.

2.5.4 Find the center of an ellipse

When two circles on a plane overlap, at most two intersection points are formed. The following method [27] describes how to find those points using Figure 2.9 as reference.

The intersection points are the points $P_3(x_3, y_3)$. The other points' coordinates will be referred to as $P_0(x_0, y_0)$, $P_1(x_1, y_1)$, $P_2(x_2, y_2)$. In order to be sure that the two circles intersect, we calculate the euclidean distance d between the centers of the circles:

$$d = ||P_1 - P_0||$$

- If $d > r_0 + r_1$, then the circles do not overlap and there are no intersecting points.
- If $d < |r_0 - r_1|$, then the circles are contained within each other and again there are no intersecting points.

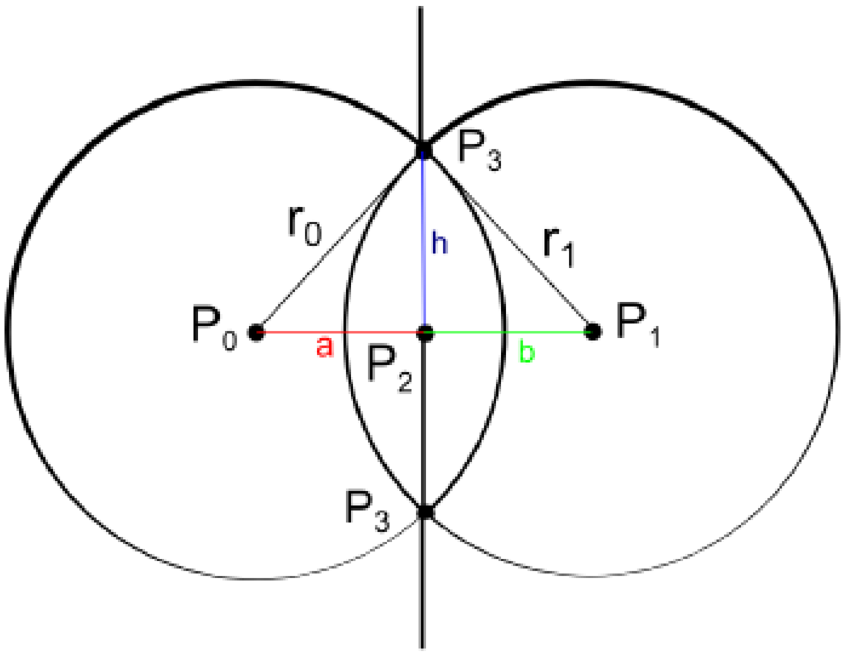


Figure 2.9: Intersection of two circles

- If d = 0 and $r_1 = r_0$, then the circles are coincident and there are infinite solutions.
- If d = r_1 + r_2 or d = |r_1 - r_2|, then the circles are tangent and there is only one intersecting point which lies along the line connecting the two centers.

If none of the above apply, then we can calculate the two intersecting points. Our first goal is to find h and P_2. We notice that two triangles are formed through points $P_0P_2P_3$ and $P_1P_2P_3$. Thus, from the Pythagorean theorem, we know that:

$$r_0^2 = h^2 + a^2$$
$$r_1^2 = h^2 + b^2$$
$$d = a + b$$

Considering the above equations, we can solve for a,

$$a = \frac{r_0^2 - r_1^2 + d^2}{2d}$$

Now we can find h by substituting a into equation $h_2 = r_0^2 - a^2$ and P_2 with the equations:

$$x_2 = x_0 + \frac{a(x_1 - x_0)}{d}$$
$$y_2 = y_0 + \frac{a(y_1 - y_0)}{d}$$

It remains to calculate the two sets of coordinates for the two P_3 points:

$$x_4 = x_2 \pm \frac{h(y_1 - y_0)}{d}$$
$$y_3 = y_2 \mp \frac{h(x_1 - x_0)}{d}$$

Chapter 3

Problem Statement

In this chapter we state the problem that this thesis studies, namely color recognition using Gaussian Distributions and Line recognition and categorization in the RoboCup made for field. To fully understand the challenges behind those problems, we first provide a detailed description of the camera and the images it delivers.

3.1 Color Recognition: The Problem

3.1.1 Naos' Camera

The Aldebaran Robotics Nao Humanoid robot as we said holds a 640x480 (WxH) pixels camera. In addition to its low resolution camera it also adds a significant amount of noise and shadows on its corners. Figure 3.1 shows a real image taken from the Nao robot in a fully still position in which we can discern both noise and shadows. In real time the nao takes and processes images that may be shacked or in an angle, as when it moves it stands from one foot to another which makes him move his hole center of mass in on leg, thus its' head has an angle which is being transferred to the image. In this case the image is being processed, a little bit differently than what we see in simulation or in fully still images, which has been implemented by Manolis Orfanoudakis (2011) in his thesis [6] and our teams' work until now. However the horizontal position index of the image has a range of [0, 640] and the vertical position index has a range of [0, 480]. The (0, 0) position is located at the upper left corner of the image. The vision system, must be able to process the video input with sufficient speed, to allow the agent to act and react promptly. Of course, this is not the only processing that takes place in real time,

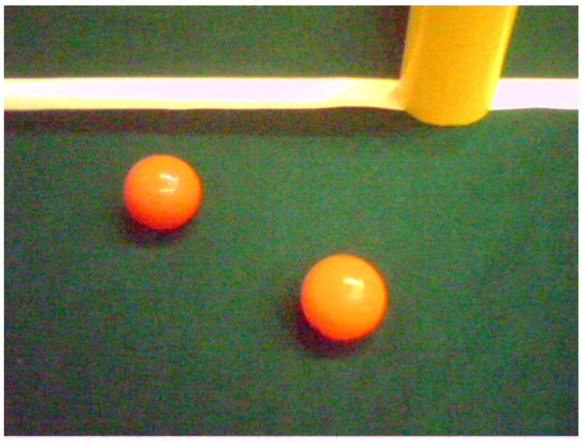


Figure 3.1: Image taken form the Nao robot without any editing

limiting even more the available processing time. To quantify this, given an operating rate of 30 fps the available processing time is 33.33 ms. Taking into account that the system must leave ample headroom for other systems, we further limit this time to 20 ms. This is an extraordinarily little time for video processing.

3.1.2 Producing a Color Table

As it has be stated before the key to a good sight-vision is a tool called "Color Table". "Color Table" is a simple file that contains all information needed by robot in order to understand its environment. Moreover, it is a mapping file which allows the robot to map each color it sees from his environment, to an existed one in the table and decide if it's a key color to some elements on the field, such as the ball (color orange), the goal (color yellow), the lines (color white) and the field (color green). As it is vital to recognize the field green color, in fact we do not use it further than obtaining the edges of an other object, where green begins the objects pixels end.
More specifically, color classification information can be stored in a 3-Dimensional lookup table. This table maps the 256 x 256 x 256 total number of possible pixel values, into a color value. A common optimization employed by many teams is to reduce the table size , for efficiency reasons: even if all possible color classes fit in 1-byte values, this makes the color table an impractical 16 MiB structure. Typically, color classes span across only a small percentage of the whole colorspace, and they are distributed in fairly compact

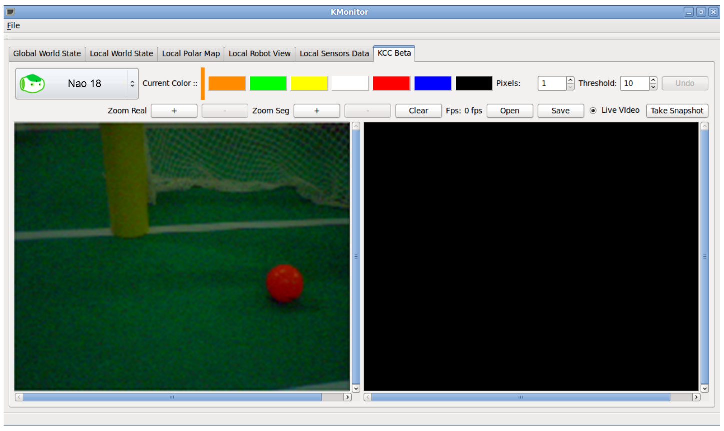


Figure 3.2: KMonitor tool

clusters. With this in mind, it is possible to subsample the color space classification into a smaller table. Using the bit mask color class modeling one can combine multiple values into a single one using bit wise operations. By grouping values in each of the three dimensions into groups of 2^N values for some $1 \leq N < 8$, we can decrease the size of each dimension by a factor of 2^N. One can visualize the discrete color space divided into cuboids, regularly placed along each dimension. Each cuboid is merged into a single value. This can drastically reduce the required storage storage, and also improve the performance of the process by limiting accesses to the main memory. A practical size of a color table can be as low as $2^8/2^4$ x $2^8/2^4$ x $2^8/2^4$ which produces a table of only 32 KiB in size. [6]
Regarding "Color table" our team possesses a tool called "KMonitor" (Figure 3.2) which allows a person to manually point, using the robots camera and sight, which pixels to recognize as which color. This tool then produces a "Color Table" which is "fed" again manually to the robot in order to actually see its environment. "KMonitor" is a tool made by M. Karamitrou as a Diploma thesis in our University under the auspices of "Kouretes" [17] . It's obvious to understand that this whole process while it produces a very good and usable result-color table, it is extremely heuristic, it also demands a lot of effort and patients each time the robot moves to a deferent environment, thus different light conditions. The outcome of this procedure is that our team has to spend 10' to 15' in every presentation or tournament that it's participating in. This thesis propose a more dynamic and less heuristic way of producing a "Color Table" without the human element in it.

3.2 Field Line Recognition: The Problem

3.2.1 Line and Types

Unlike goal and ball recognition, field line recognition is a more complex task. Lines are formed with white pixels which can also be found in other objects on the field as well, such as other robot players, banners outside the field or noise to the camera. Therefore, areas of white pixels that indicate lines must be isolated from the rest. Finding the best settings in order to separate the desired white areas as efficiently as possible is pretty challenging due to the camera's drawbacks mentioned above and the unlimited number of game situations that may occur and projected on the camera. This is true, because the environment in which our agent has to act is dynamic due to the presence of multiple agents. For example, another agent may block the field of view of our agent or blend with field lines in the image, but in any case it alters entirely the way we acquire the information we need. At the time that we have correctly recognize a line segment we have to characterize its type in order to "feed" our localization agent with noteworthy informations. In this thesis we consider 5 types of lines (Figure 3.3) :

- `Straight Lines` which can be found in the center of the field, the penalty area and the sides of the field, called touch lines
- `Curved Lines` which can be found in the center of the field, called center circle
- `Corners` which can be found in the penalty area ends' and at the corners of the field
- `T-Lines` which can be found in the center of the field where the touchline crosses the center line and where the goal line crosses the penalty area lines.
- `Cross Lines` which can be found in the center of the field and in between the center and the penalty area. This type of line is not considered useful as regarding to Naos' camera resolution it can be seen very few times.

3.2.2 Line and Localization

As mentioned before, localization or to be more specific self-localization, is a crucial problem in order to achieve autonomy for an agent. If a robot has a good belief about his

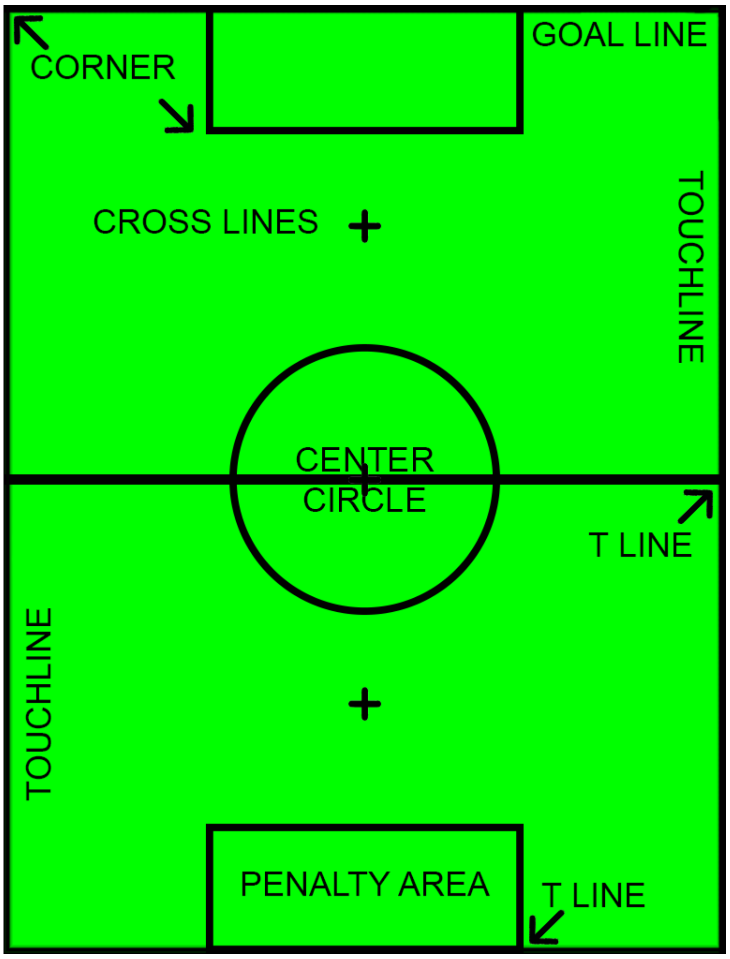


Figure 3.3: Soccer Field Line Types

current position, then it's easier for it to take better decisions regarding future actions. To do so, the agent needs crucial informations, such as landmarks or obstacles, which can be obtained by many of his sensors (e.g. camera, laser, sonar etc.). There is always an uncertainty in measurements, so consequently faults in the estimated position are expected. The goal of localization is to determine the agent's own position as accurately as possible. Several techniques, such as Kalman filters, particle filters, and constraint-based methods, are used to achieve localization. In robotic football, localization gives a significant advantage to a player. A player's field position is characterized by three variables; x, y, and ϑ, where (x, y) are the coordinates on the field plane and ϑ is the angle value that describes the player's orientation. The primary problem that has to be confronted in localization is how to deal with the errors present in the estimations.

Landmark recognition is not carried out perfectly and small deviations of the landmarks' estimated positions in regard to the player are expected. However, even small, those deviations can become significant during the computation of the candidate positions of a player in the field, so a filtering procedure that chooses the best candidate should be implemented. Besides this, a landmark with serious error measurements may not be taken into consideration for the self-localization process. This thesis doesn't involve any localization solutions neither attempts to use the new landmarks that it provides to the agent who runs the localization problem. Localization is a subject very complicated and complex that is been studied in "Kouretes" by other members such us; N.Kargas, E. Michelioudakis, N.Pavlakis in their theses [8] [28] [9].In this thesis we just "feed" the localization agent with landmarks and their positions in order for it to obtain a better belief of his current position.

3.3 Related Work

Regarding color recognition, two of the leading teams in RoboCup, "B-Human" and "Nao Team HTWK", are using two completely different approaches for this problem. "B-Human", on their latest report, use an off-line tool based on three most important parameters; white balance, exposure and gain. In this off-line tool, they manually experiment with those three parameters, in order to find the best fitting values for the current environment. This resembles to our own off-line method of producing a color table. "Nao Team HTWK", on the other hand, use a real-time capable segmentation with no need for calibration. Moreover, by applying the knowledge of the objects' shapes, they developed an object recognition algorithm which can handle changing light conditions and colors robustly without the need for prior calibration.
Regarding line recognition, "B-Human" detect field lines, line crossings, and the center circle, with a method similar to our approach. Moreover, they detect, delete, split or merge all lines in a single frame and search for meeting points. Additionally for the circle, they detect it using a two-step algorithm. The first step clusters the lines into groups, where each group represents a possible circle. In the second step, linear regression is used to fit a center circle into each cluster. Finally, if the fitting error for one of the clusters is small enough, it is accepted as being the actual center circle. "Nao Team HTWK", likewise, use a similar method of detecting, deleting, splitting or merging all lines found in the field.

Chapter 4

Our Approach

Our solution to the problem of color recognition is based on the creation of a mechanism that is responsible for color table production based on the classification of each color with Gaussian Distributions models. The output of the color table mechanism allows the robots to fully understand their environment colors. Our approach is one, that has never been used before, by any other team or university at this point, and it is considered ground breaking as it produces a better result than we ever seen in "Kouretes" and without any difficulty post base programing. As for the line recognition our solution to the problem is based on the ability of all lines, we are in need of, to be represented by a single second degree equation. The output of this mechanism are landmarks in the field that helps the localization agent to provide a better belief of his current position. This idea of Field Lines recognition is similar to this suggested in G. Georgakis thesis [24]

4.1 Color Recognition: The Solution

4.1.1 The Idea

Our idea is based on the ability of each Gaussian Distribution [22] to encircle a space of its' environment and have all its points inside it. As we consider all of our points without weight this binds us to a solution that, as we said, has all of its interior points inside the lobe of our Gaussian distribution. This conclusion gives us the ability to fully classify a new point (pixel) of an unknown picture in one of the existed distributions by estimating the probability of it being in all of the existed ones. The one with the highest probability is the one that will be classified in. The problem in this suggestion is that it would be very complicated and time consuming to check each pixel with all of the existed distributions. Our suggestion to this problem is the idea of having all distributions categorized by light conditions and checking the pixels of a full green picture with all the green distributions in order to find the light conditions. Then we take all existed distributions in this light condition and produce a color table for the robot.

4.1.2 Modeling all colors in Gaussian Distributions models

In this section we describe the first phase of our color recognition model, the modeling of all colors in all light conditions into Gaussian Distributions. Firstly, as described before, we need a variation of samples in many light conditions to obtain a good quality result. Thus we took the exact same picture in thirteen different gains from our Nao robot, starting from 0 gain to 300 gain with a difference of 25 gain in each picture (0, 25, 50, 75, 100, 125, 150, 175, 200, 225, 250, 275, 300). Through our experiments we found that gain from 0 to 100 are too dark to have a good result with and gains from 250-300 are quite the same, despite that we worked with them as with all others. In those thirteen pictures we marked the limits of each color in the picture from pixel to pixel in order to have samples-points for each color distribution(Figure 4.1).

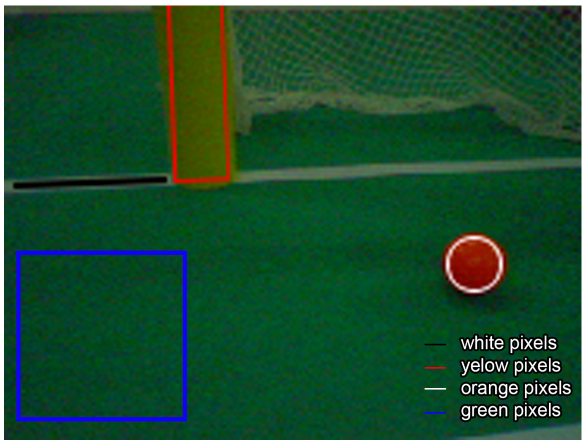


Figure 4.1: Color limits in the original raw picture from the robot

This was the reason we took the exact same pictures in all different gains, to avoid changing the limits of each color. In order to use those pixels to produce Gaussian Distributions and being able to visualize them, for our own convenience, we need to use a different color model then RGB to have the ability to remove one of the three variances R,G,B. As we said before for this cause we use the HSL color model which gives us the opportunity to use just the H and S dimensions and remove the L. Thus we can visualize all Distributions and produce a better result, since there is a significant connection between the light differences and the L parameter. Thereafter converting our pixels from YUV422 (original pixel color model from the robot) to HSL we gathered all pixels inside the thresholds and computed one Gaussian Distribution for each color, in each light conditions giving us four Distributions in thirteen different gains, as shown in Figure 4.2 for 250 gain condition.

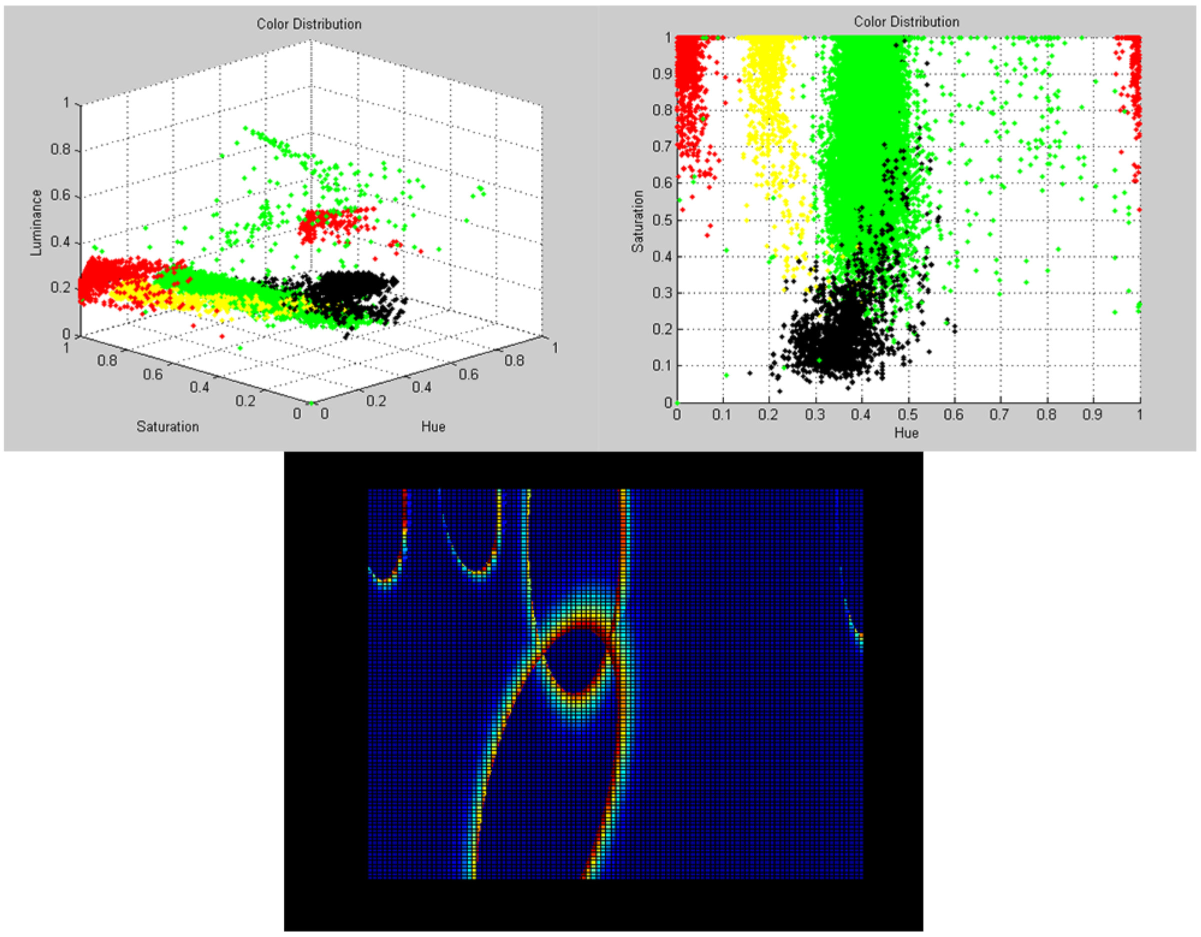


Figure 4.2: Gaussian Distributions for 250 gain picture

Now that, all distributions in thirteen different gain conditions (similar to light conditions) are in our disposal, our goal is being able to adjust our colortable selection to the environmental light conditions changes. In order to convert from YUV422 to HSL we used the following equations [29] :

YUV422 to YUV:

$$\begin{aligned} u &= yuv[0] \\ y1 &= yuv[1] \\ v &= yuv[2] \\ y2 &= yuv[3] \end{aligned}$$

YUV to RGB:

$$\begin{aligned} R &= 1.164(Y-16) + 1.596(V-128) \\ B &= 1.164(Y-16) + 2.018(U-128) \\ G &= 1.164(Y-16) - 0.813(V-128) - 0.391(U-128) \end{aligned}$$

RGB to HSL:

Hue calculation:

$$H = \begin{cases} 60°\text{x}(\frac{G'-B'}{\Delta} mod 6) & \text{if } Cmax = R' \\ 60°\text{x}(\frac{B'-R'}{\Delta} + 2) & \text{if } Cmax = G' \\ 60°\text{x}(\frac{R'-G'}{\Delta} + 4) & \text{if } Cmax = B' \end{cases}$$

Saturation calculation:

$$S = \begin{cases} 0 & \text{if } \Delta = 0 \\ \frac{\Delta}{1-|2L-1|} & \text{if } \Delta <> 0 \end{cases}$$

Lightness calculation:

$$L = (Cmax + Cmin)/2$$

As we said, all that we need is a distribution of the current green color. The means we have, to complete this task, is the "Mean" and the "Sigma" which fully describe the Gaussian Distributions. The equations we used in order to find those two variables are the followings [30] :

X:

$$X = \begin{pmatrix} X_1 \\ X_2 \\ \vdots \\ X_n \end{pmatrix}$$

Mean:

$$Mean_{Green} = \begin{pmatrix} M_R \\ M_G \\ M_B \end{pmatrix} = \begin{pmatrix} Sum_{Rvalues}/N \\ Sum_{Gvalues}/N \\ Sum_{Bvalues}/N \end{pmatrix} *$$

$$*NoWeights$$

Sigma:

$$Sigma_{Green} = \begin{pmatrix} E[(X_1-\mu_1)(X_1-\mu_1)] & E[(X_1-\mu_1)(X_2-\mu_2)] & \cdots & E[(X_1-\mu_1)(X_n-\mu_n)] \\ E[(X_2-\mu_2)(X_1-\mu_1)] & E[(X_2-\mu_2)(X_2-\mu_2)] & \cdots & E[(X_2-\mu_2)(X_n-\mu_n)] \\ \vdots & \vdots & \ddots & \vdots \\ E[(X_n-\mu_n)(X_1-\mu_1)] & E[(X_n-\mu_n)(X_2-\mu_2)] & \cdots & E[(X_{2n}-\mu_n)(X_n-\mu_n)] \end{pmatrix}$$

Afterwards, we have in our disposal all "Means" and "Sigmas" for all colors in all thirteen gain conditions. Now we need to compute the "Mean" and "Sigma" for our current green. Therefore, we put our robot in a fixed position and took a full green picture, to provide it to our off-line tool, in order to compute the current green gain conditions. As we mentioned before our Nao's camera has a lot of noise in its corners, therefore we choose not to take the whole picture as sample, but only its center pixels (Figure 4.3).

After some experiments our results showed us that we have the same output with both of the following processes:

- *Find the closest Mean*:
  We compute the "Mean" (μ) of all current pixels and compare it with all existed ones on green Gaussian Distributions. The one which is closest (Algebraic distance) to our "Mean" (μ) is the one we chose.

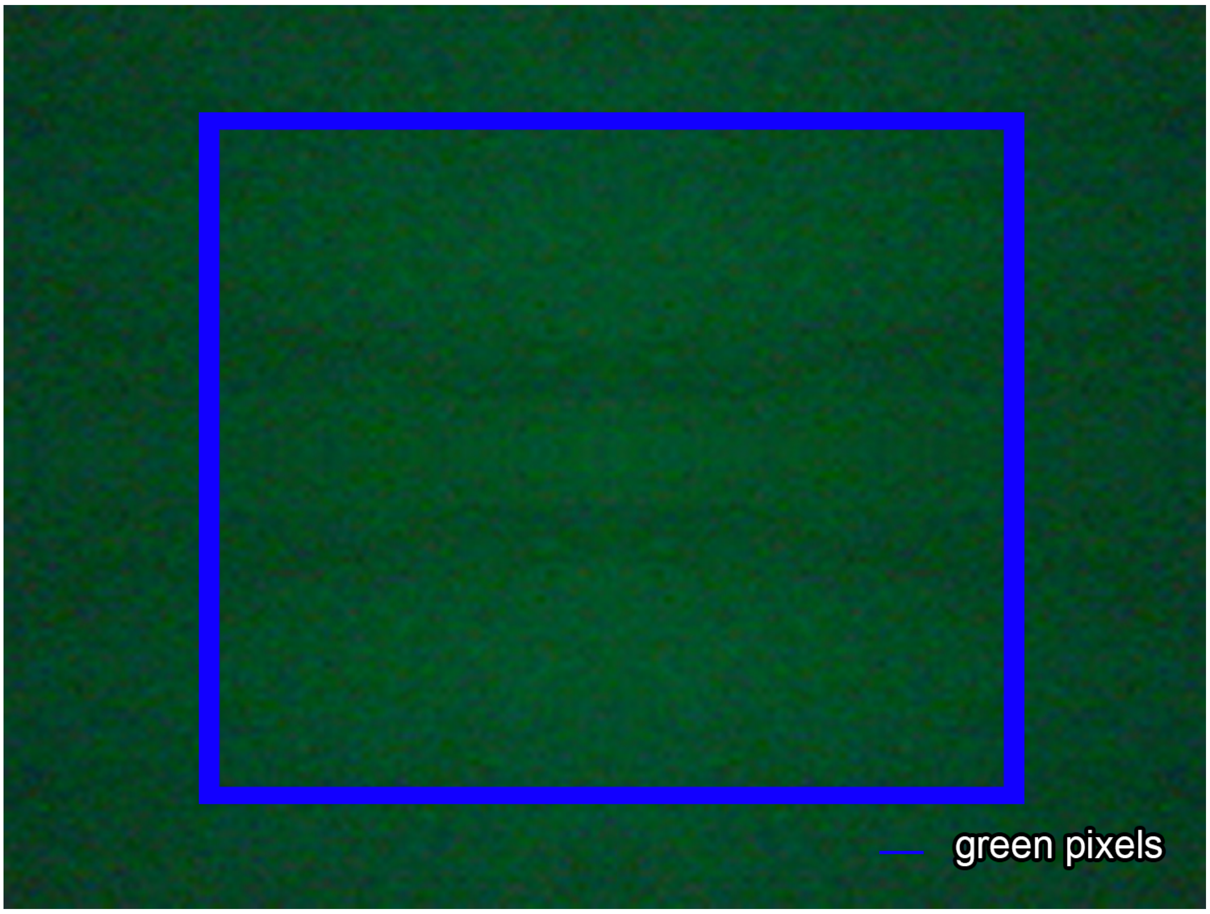


Figure 4.3: Green picture for 250 gain picture

- *Find the Highests Probability*:
  We compute the probability of each pixel to all green Gaussian Distributions and then we chose the one distribution that has been chosen the most.

Thus, in order to minimize extra work and maximize effectiveness we have chosen we have chosen the first one (finding the closest Mean), and our results were better than we expected. At this point we are able to fully perceive the light conditions that we are at and even if some insignificant errors occurred, even then, the results where very good.

4.1.3 Extracting a Color table

Our last step in this approach was to extract a fully working colortable for the robot, to make it able to fully understand his environment and his surroundings. For this purpose we took a random photo that included all colors, that we are in need of (green, orange, yellow, white), and after predicting the light conditions with the process that we just described we assigned to each pixel of this picture, the color that had the highest probability referring to the Gaussian Distributions for this light condition. At this point, we have established a connection between the real color of each pixel and the one that we assigned to it. These connections are the fundings of every colortable that we use in our team "Kouretes" for the RoboCup. More specifically, considering the YUV cube and the random photo, each pixel's real colors are coordinates in the YUV cube and the color that we assigned to this pixel is simply a value for the colortable. Thus, the colortable is simply a reference from the YUV cube to our colors. Moreover when the robot, in a real time execution, attempts to define the color of the picture it looks up to the colortable to find the value that we have assigned to those coordinates. As it's expected there are some areas that have no value assigned to. Those areas are the ones that we are not in need of. The biggest difference between this suggestion of the colortable from Gaussian Distributions and the one that we talked about, is that the one that we developed in this thesis simply needs some pictures as samples in order to produce a colortable and is not based on the human element. This makes it less heuristic and doesn't allow shadows or wrong pixels pointed in wrong colors. A simple, yet very important example to understand the difference between those two procedures, is shown in the Figure 4.4

.

4.2 Field Line Recognition: The Solution

4.2.1 The Idea

Our idea for the lines recognition and their usage as landmarks is based on the discrimination of all lines in four different types of landmarks: straight lines, curved lines, corners and T-lines. Having the ability to recognize all these types, adds fourteen different landmarks to our localization system and lets us obtain a better belief about our current position. If we analyze those four types, we can easily see that all lines, except curved ones, are formed by one or many straight lines. This conclusion adds, to our approach, the problem of finding means that can fully classify all straight lines and their meeting points into the three categories above. Considering the small resolution of Nao's camera (640x480), the big amount of noise in it and it's weak processor, our approach needs a

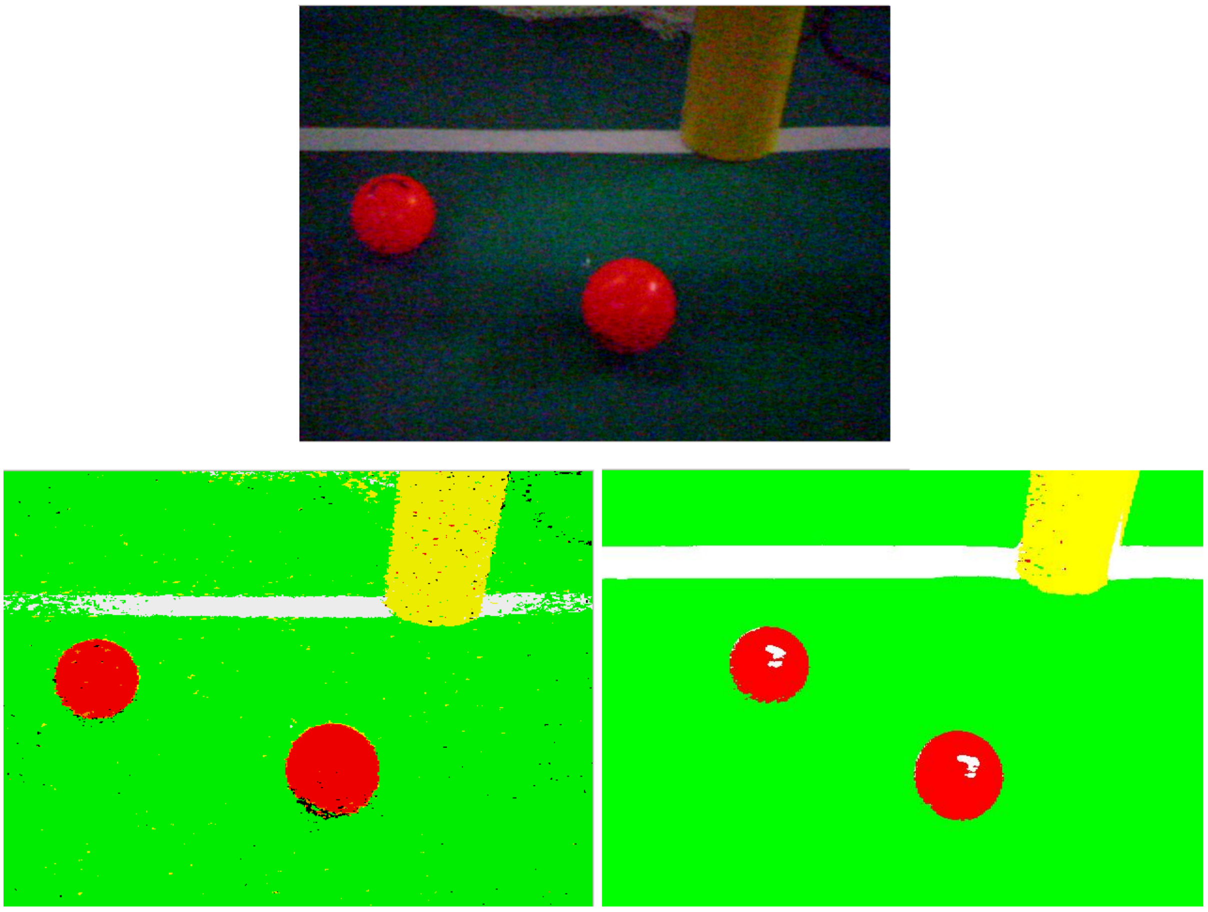


Figure 4.4: Frame as it is produced from two different colortable methods

very good yet fast algorithm. The whole idea/algorithm, which this thesis is based on, is the designation of all lines, straight and curved, by a second degree polynomial function called "Least Square fit".

4.2.2 Recognizing White pixels

First of all we need to clarify that the robots agent that completes the task of recognizing objects and all that depends on sight, called "Vision", is working on clock cycles and takes 30 frames per second. In order to simplify this problem statement we will work in one single frame meaning for one single circle for this agent. We are not working with means that connect two cycles information to obtain one greater, though our code has some ways to reject false information in some situations. Our system first needs to recognize all white pixels in the frame in order to search for lines. Thus this is a very easy and simple way, it is time consuming, which our processor can't handle in one cycle. In order to gain valuable time we need to find more complex ways of recognizing pixels and objects. Manolis Orfanoudakis in his thesis [6] suggests a two way grid (horizontal/vertical) that process only the pixels which match the horizontal and vertical grids. Those pixels will be edited in the following processes by the "Vision" agent (Figure 4.5).

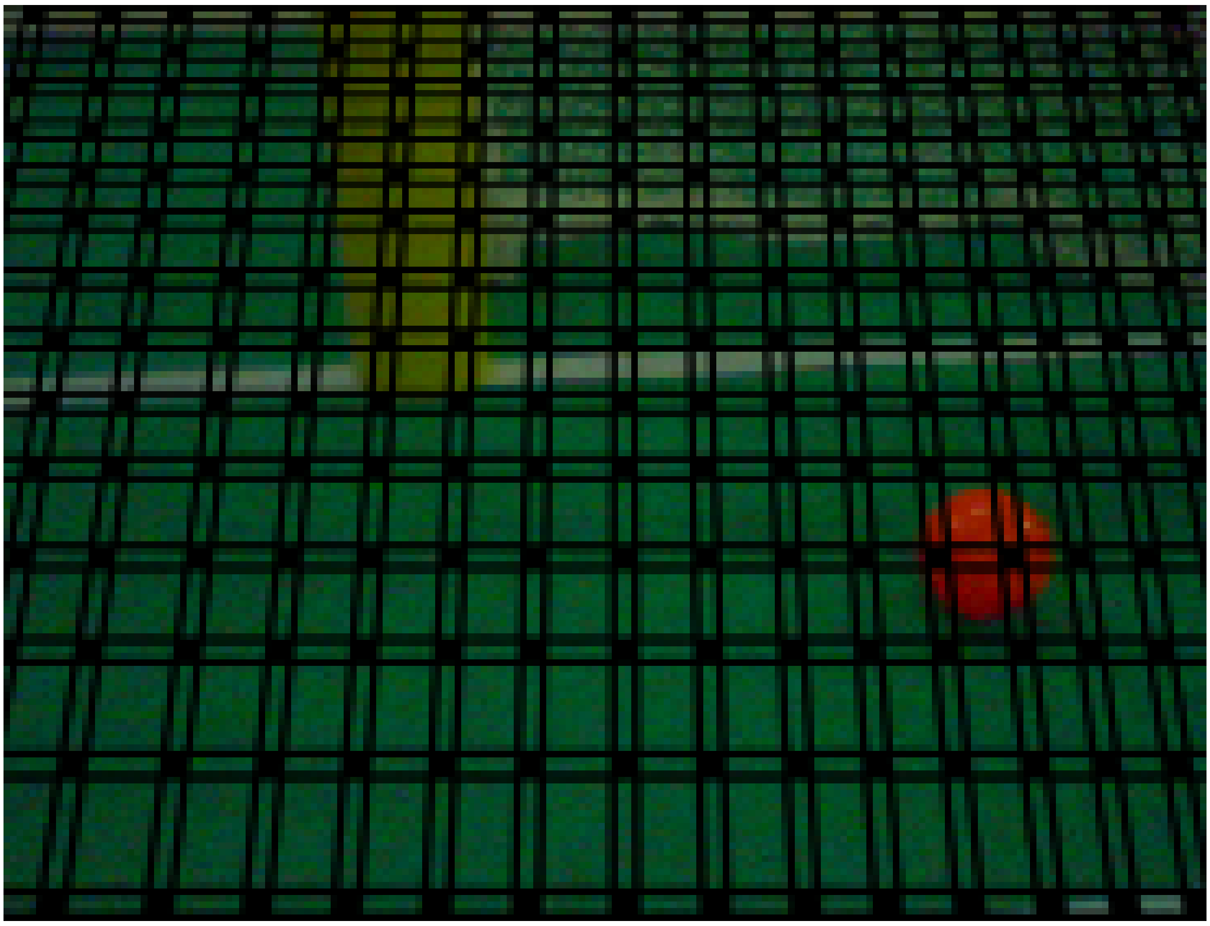


Figure 4.5: Unrealistic scaled example of grid scan

This grid scan works vertically and horizontally from the robots belief of the world, meaning that it's not vertically in the image but it is, considering his belief about the real world rotation (Figure 4.6). As we can see, the higher the grid goes, the denser it becomes. This is happening due to the theory of prospective, the closest the object is, the bigger it seams to be. Thus, in long distances we need to check for more pixels. It's also important to notice, that in this grid scan suggestion, the robot doesn't search for pixels above his belief of the horizon, where horizon is defined by the robot's position with a lot of kinematics and it's considered above his center of mass. It's also important to notice, that the scan starts from the bottom right corner (x=480,y=0) and goes from left to right, from bottom to top (x=0,y=640).

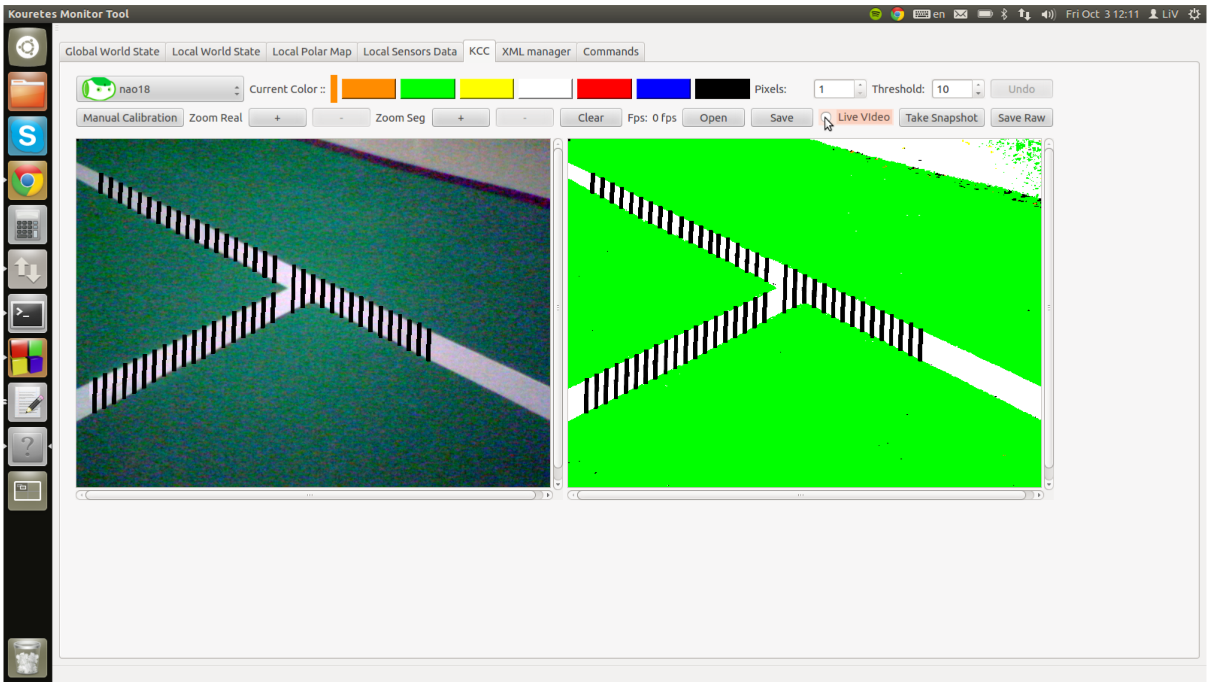


Figure 4.6: Grid scan example from the robot

In this grid scan, if we recognize a pixel of the color we need, we then add it to a row with elements of the same color. In our approach we search and store all white pixels. As we mentioned before there is a big amount of noise and since in this thesis we didn't worked with the colortable method that we suggested, as it wasn't ready till we finished both subjects, there was a big amount of wrong pointed colors too. This made our work harder and gave us many wrong results in the begging, by finding white pixels where there weren't. Thus we needed some procedures that would "clean" all wrong pixels. In order to do so, first we scanned vertically up and down from this pixel to see where the white pixels end, in this way we had a "group of white pixels" forming a white vertical line. Then we had some simple criteria that had to be matched for those pixels, to consider them acceptable:

- `Top and Bottom pixels:` Top and bottom pixels of this group had to be green. Thus, if the "group" started or ended with different color, it meant that it is not part of a line in the field (other obstacles) or there is something blocking our view.(Figure 4.7)

- `Length of groups:` The length of each "group" must be higher than 4 pixels, less than 100 pixels (too long) and less than two times the mean length of all "groups" found. Also every group cant be 1.4 times bigger than its left one or 0.625 times smaller.

- `Close-enough lines:` This criterion aims on discarding all "groups" that have been accepted till now but they have no other "groups" close to them. In this thesis we found this threshold of "close-enough" distance to be the length of the current "group" in order to have a fair criterion for both long and close to the robot objects.

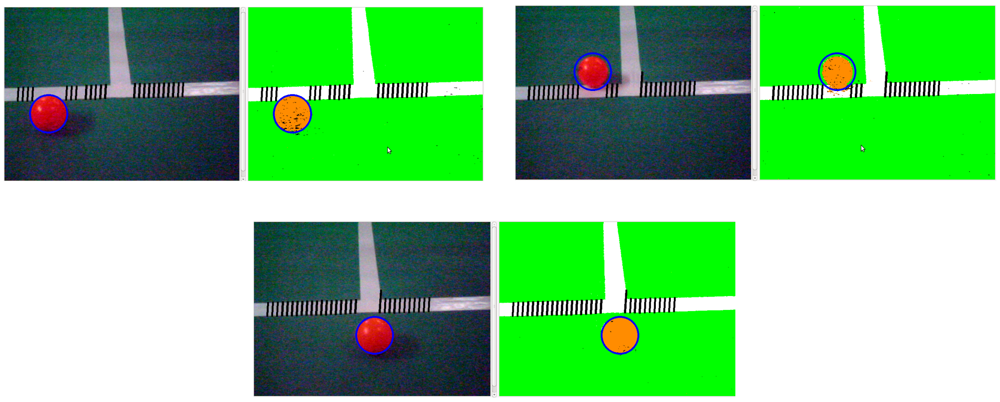


Figure 4.7: Top and Bottom pixels

With those three simple criteria we are able to ignore "groups" that are part of an obstacle such as other robots or noises but we pay the price of not recognizing full vertically lines as their length is too big (Figure 4.8). This is not a problem since its very important to throw out "groups" that belongs to obstacles (Figure 4.9) and the lines that we disposed will be eventually detected in an other time cycle with an other angle point of view (Figure 4.10).

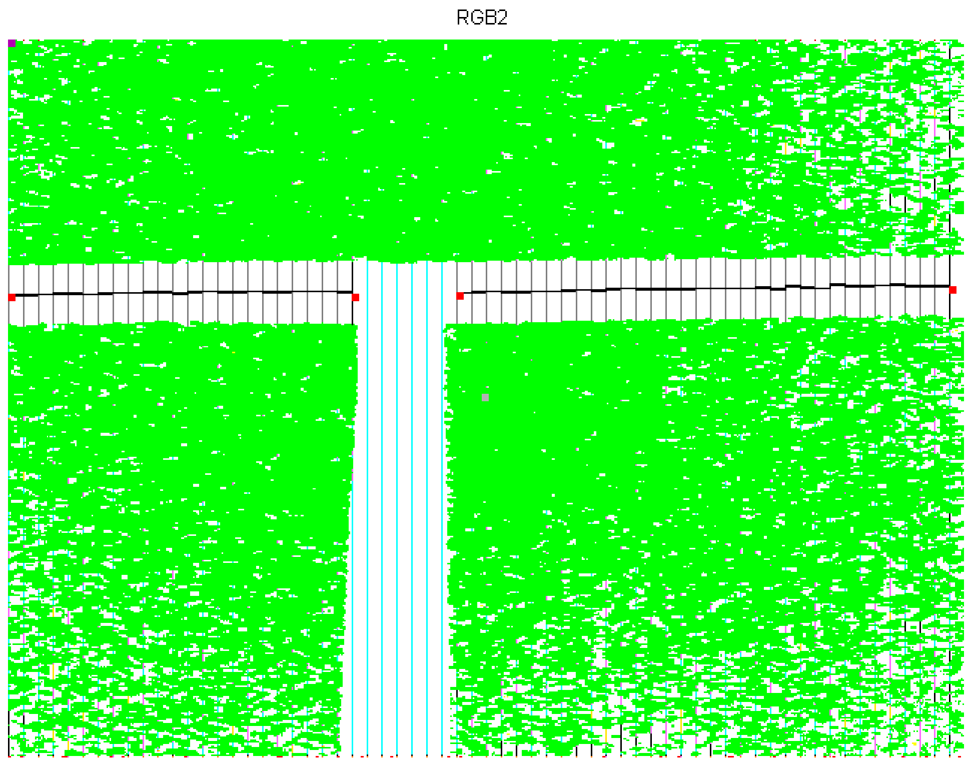


Figure 4.8: Example of a vertical ”group” being disposed

4.2.3 Forming a Line

Now that we have in our disposal all accepted ”groups” we need to use them to form a line, if possible. To do so, firstly we have to organize all ”groups” in lines that are real in the field. For example a straight line is acceptable, so is a curved and a corner, but we can’t have lines that go back and forward from groups. We need a single direction for each line. There are three categories of directions with two subcategories each:

- `Straight`
  - `from left to right`
  - `from right to left`
- `Top to Bottom`
  - `from left to right`

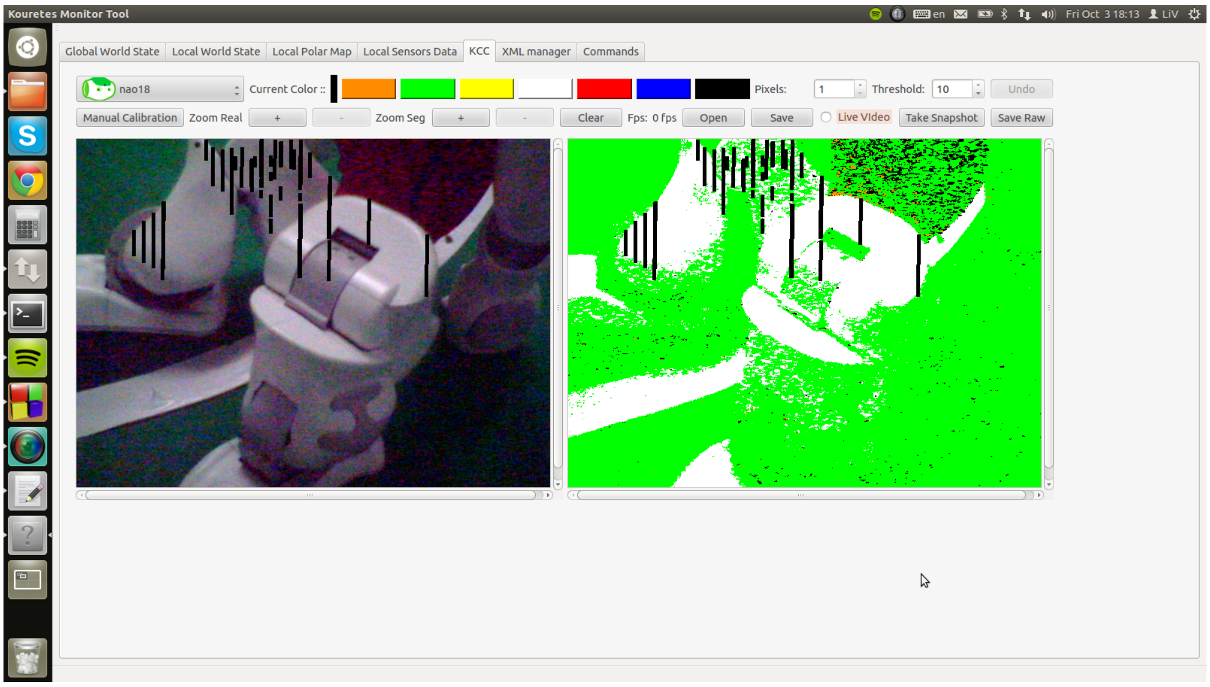


Figure 4.9: Example of "groups" being disposed

    - from right to left

- `Bottom to Top`
    - `from left to right`
    - `from right to left`

Given those directions, a problem arises with some of the possible ways we see a line in a frame. For example, if we see a corner in a complete symmetrical way we then read the "groups" one by one alternating from right line to left line. This is a problem for our work, thus we take the first "group" we see and we do not allow to add next to it a "group" that has a distance greater than its length. If this criterion is met, we form a line with those two "groups". This procedure goes on until we used all "groups". If this criterion isn't met, we start a new line. At this point we have many small lines that need to be merged or deleted. This procedure is described in the Algorithm 1.

In order to merge all lines we reorganize all small ones by their x variable so that we have a common referring point. Then using the "Least squares fit" theory we find the three coefficients for each small line C_0, C_1 and C_2. At this point probably all lines,

Algorithm 1 Lines Formation

```
Input: $Groups = \{g_1, \dots, g_{n-1}\}$
    Groups have as variables there upper pixel $g_1(1)$, bottom pixel $g_1(2)$ only
Output: Lines or parts of a line: $lines_{atstart}$
 1: for all $Groups_n$ do
 2:     $Length_{g_k} = g_1(1) - g_1(2)$
 3:     $Mid_{g_k} = g_1(1) + \frac{g_1(1)-g_1(2)}{2}$
 4: end for
 5: $lines_{atstart}(1) = g_1$
 6: for all $Groups_n, n > 1$ do
 7:     if $abs(Mid_{g_k} - Mid_{g_{k+1}}) > Length_{g_k}$ then
 8:         end of line
 9:         new line
10:     else
11:         $lines_{atstart}(k) = g_k$
12:         k=k+1;
13:     end if
14: end for
15: return $lines_{atstart}$
```

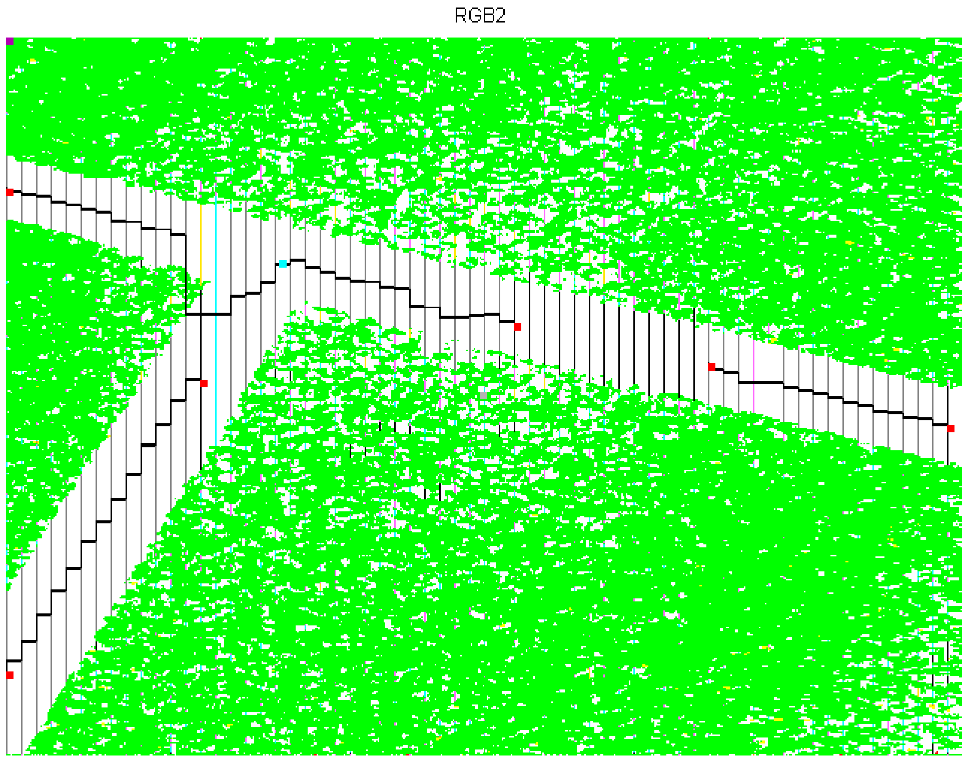


Figure 4.10: Example of a vertical "group" not being disposed

straight or not, must be in parts of smaller ones. There is a slight chance that we have already in our disposal a corner line, which generates the problem of considering this corner line as a curve. To solve this problem, we examine all lines found until now, with regard to their direction, up or down. If one of the lines starts from bottom-to-top (up) and ends, or has in its' length, a change of direction to top-to-bottom (down), we consider it a corner, and we separate it at the point of its' direction change. In order to avoid confusion between corners and curves or straight lines that simply have an up or down element, we give to this direction check procedure a small threshold. The biggest problem we faced in this thesis was that some times the curves were considered as lines and vice versa. In theory all lines have $C_0 = 0$, though in reality this doesn't match our expectations. This is happening because the robot never sees any line as a complete straight one. Given the noise and the poor analysis, best case scenario is that C_0 will be close to zero in a difference of E-4 class. After many experiments we found this thick threshold in our simulation to be 0.0006. In our on-line implementation on the Nao robot this was unrealistic, giving us a threshold expectation between 0.007. This was 1000 times bigger and many times a line exceeded this value, meaning that some lines will be considered as curves. Nevertheless we managed to have good results in merging all small lines into bigger ones and losing only a small group of them. At this point we faced a dilemma; putting the threshold high enough and consider some curves as lines or vice versa. Since curves are more important, but we can manage to see them if we have a longer point of view, and lines can form many other landmarks, we added a second threshold in order to have a "blind area" between those two types of line. Moreover, we marked as "blind area" the area between 0.009 and 0.02. If a line has its C_0 in this area it's been ignored as part of the solution in problem that we just stated. It's important to notice that all that we mentioned before, work the same way with positive or negative C_0. Afterwards, we just merge all small lines that can be a part of a greater one depending on their Coefficients values. There are two types of merges:

- `Curved lines`
  - $|C_0|$`>0.02 for both lines`
- `Straight lines`
  - $|C_0| \leq 0.009$ `for both lines`

This procedure is described more detailed in the Algorithm 2.

4.2.4 Forming Circles, Corners and T-lines

After this procedure, we may hold one or more lines and one or more curves. Given the importance of any curve found in a frame, we decided that if so, we will ignore all other lines found in this frame. Additionally, if more than one curves are found, we examine the one with the biggest length size and ignore all others. The goal of this thesis is to recognize all four type of landmarks, as we mentioned before, which are circles, corners, T-lines and straight lines. To do so, we have to find the meeting point, if there is one, of all straight lines, or the center of the circle that any curved line is part of in this frame. We cite below, in order of importance, all forming types. If one is found, we ignore all others below it.

Algorithm 2 Blind Area

```
Input: $lines_{atstart} = \{l_1, \ldots, l_{n-1}\}$
Output: Lines: $lines_{merged}$
 1: for all $lines_{atstart(k)}$ do
 2:     find $C_{0(k)}$, $C_{1(k)}$, $C_{2(k)}$
 3: end for
 4: if $C_{0(k)} > 0.002$ then
 5:     $lines_{atstart(k)} = curvedlineconcaveupwards$
 6: else if $C_{0(k)} < -0.02$ then
 7:     $lines_{atstart(k)} = curvedlineconcavedownwards$
 8: else if $-0.009 < C_{0(k)} < 0$ then
 9:     $lines_{atstart(k)} = straightlineupwards$
10: else if $0 \leq C_{0(k)} < 0.009$ then
11:     $lines_{atstart(k)} = straightlinedownwards$
12: else if $0.009 < abs(C_{0(k)}) < 0.02$ then
13:     delete $lines_{atstart(k)}$
14: end if
15: return $lines_{final}$
16: for all $lines_{final(k)}$ do
17:     if $lines_{final(k)}$ and $lines_{final(k+1)}$ = curved with same concave type then
18:         Merge
19:     else if $lines_{final(k)}$ and $lines_{final(k+1)}$= straight line with same direction then
20:         Merge
21:     end if
22: end for
23: return $lines_{merged}$
24: for all $lines_{merged}$ do
25:     if Change of direction then
26:         Split in the exact pixel
27:     end if
28:     for all for any of two $lines_{split}$ do
29:         if $line_{length} < 4$ then
30:             delete
31:         end if
32:     end for
33: end for
34: return $lines_{split}$
```

- `Forming Circles:`For the purpose of finding the center of the circle, we examine the curved line that we found. As mentioned in Section 2.5.3, we can easily find the center of the circle, given three points on its perimeter. To do so, we extract the first, the middle and the last pixel in our curve and project them, using "Kinematix", on the field. Afterwards, we compute the center of the circle using the method described in Section 2.5.3. Finally, for our own convenience, we project those coordinates in our image so that we can see the results.

- `Forming Corners:`There is a slight chance that in our lines, as we mentioned before, there is already a corner formed and if so, we have to split it. If we haven't found a corner yet, we need to find the meeting points of two, perpendicular to each other, straight lines that we may have already identify. If their meeting point, is outside the boundaries of one of those lines (Figure 4.11) then it's consider as a corner.

- `Forming T-Lines:`As in Corners we work in the same way, but on this case, if the meeting point is within the boundaries of one of those lines (Figure 4.11), then it's consider a corner.

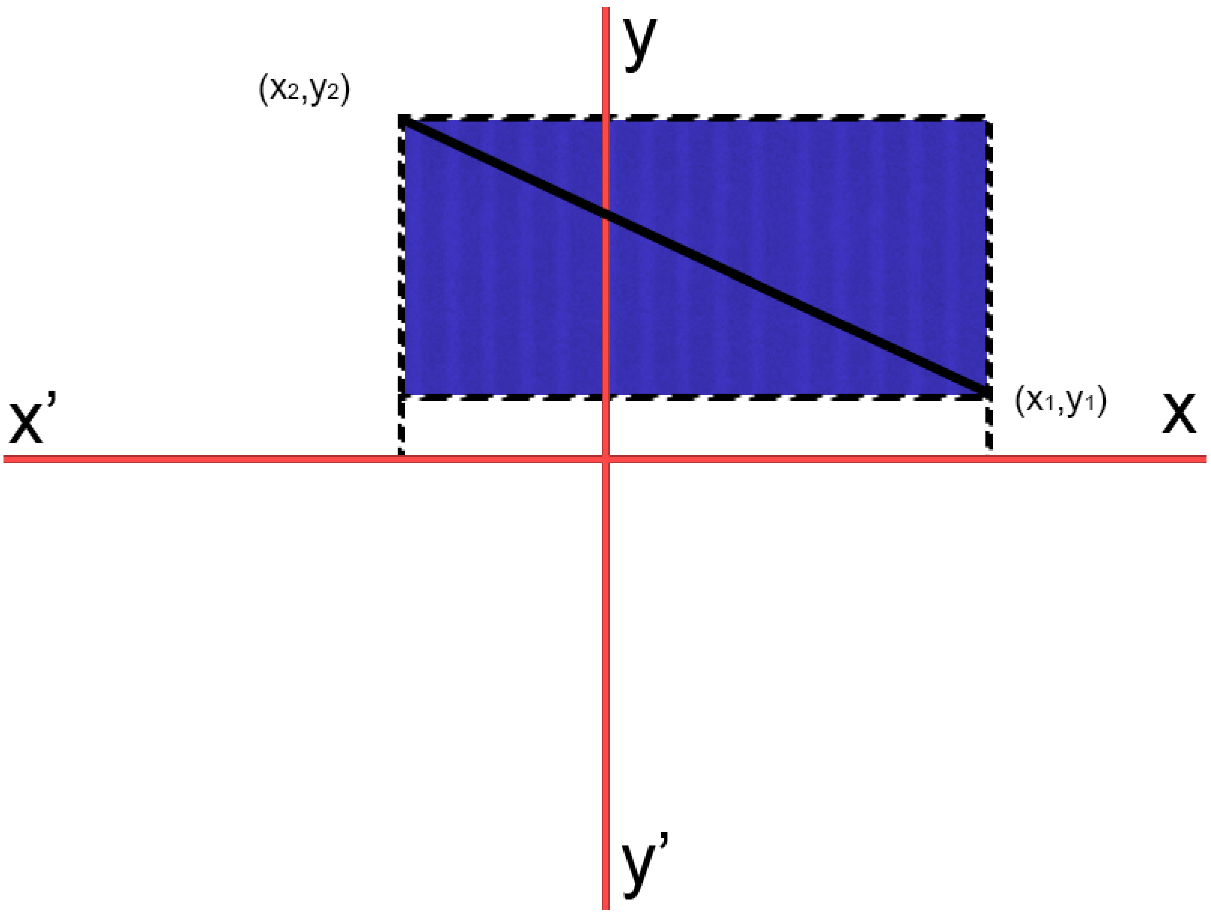


Figure 4.11: Boundaries of a Line

Chapter 5

Implementation

5.1 Matlab Simulation of Line Recognition

Before integrating our coordination mechanism on our Monas software architecture for execution on the real robots, we implemented a simulation in Matlab, to test the feasibility and the effectiveness of our approach.

We initially implemented the image process in random pictures of many possible scenarios in a real RoboCup game (Figure 5.1 , Figure 5.2), using for color recognition our own colortable mechanism. This simulation enabled us to visualize our ideas, evaluate the outcome, and tune the parameters, constants, and percentages used in the algorithm. Thereafter, we implemented the line acceptance algorithm with the criteria that we mentioned before and the line merging algorithm to observe and evaluate their behavior. Finally we implemented the line classification algorithm and visualized the outcome. This Matlab simulation is certainly not fully representative of the actual implementation on the real robots, since it assumes an ideal environment without noise and makes use of ample computational resources, when in fact this is not true. Nonetheless, it proved to be sufficient to get a first glimpse of whether our coordination approach would work.

Figure 5.3 (left) shows the visual output of the ”groups” recognition, after discarding those that didn’t match the first criteria. Every different color, match a different criterion and black color represents the acceptable ”groups”. In Figure 5.3 (right) after using the merging algorithm the ”groups” form lines. Additionally, Figure 5.4 depicts the results of a circle, Figure 5.5 the results of a Corner, and Figure 5.6 the results of a T-line. In all those figures, red squares represent the line’s pixels that matter for the localization agent (starting, maybe mid and ending pixel). The cyan squares are where we found a T-line or a Corner and finally the blue squares are where we found the center of a circle. While in our Matlab simulation, we were obligated to encounter the circle as an ellipse formed by two other circles in order to find the center of the circle, on our Nao's on board simulation this wasn't necessary due to the appliance of "Kinematix". A careful observer can easily notice through this simulation and graphical user interface that the outcome of our approach yields extremely good results.

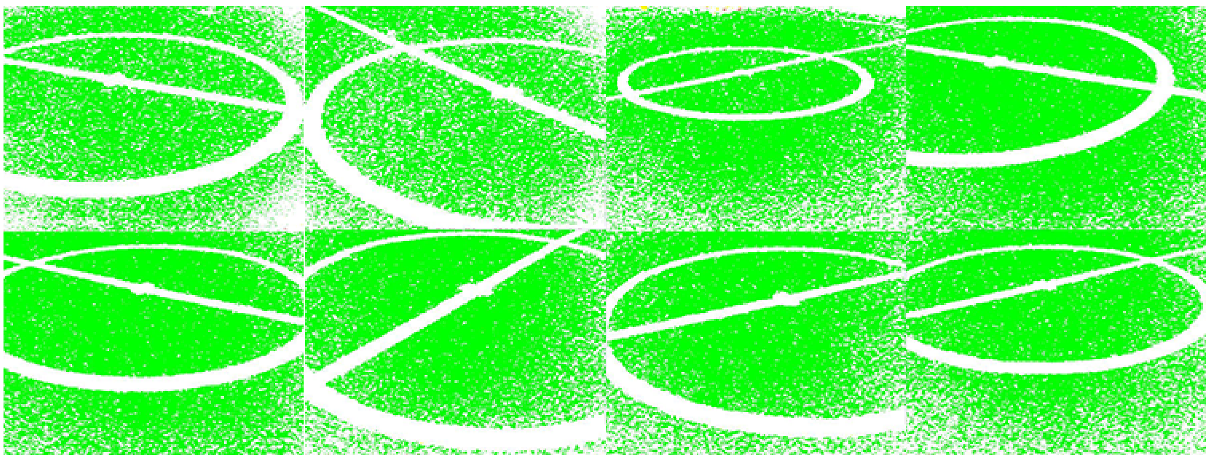


Figure 5.1: Circle scenarios

5.2 The `Vision` Monas Activity

In order to implement our approach on the real NAO robots, we developed our line recognition mechanism and integrated it on the Monas `Vision` activity. This activity is responsible for the sight and objects recognition of our robots and operates on the frames that the robot takes on each time cycle. In coordination with others activities it takes output results, such as poses or beliefs about the environment, and produces outputs to other activities such us the `Behavior` activity or the `Localization`. `Vision` is currently executed on the `KVision` agent and is specified inside our `agents.xml` configuration file that determines which activities will be running on the robot and at what frequency. Currently is being executed at a frequency of 25Hz (25 times per second).

The high frequency of execution implies that vision module must take place in less than 20 milliseconds. To understand the significance of this constraint we must take into

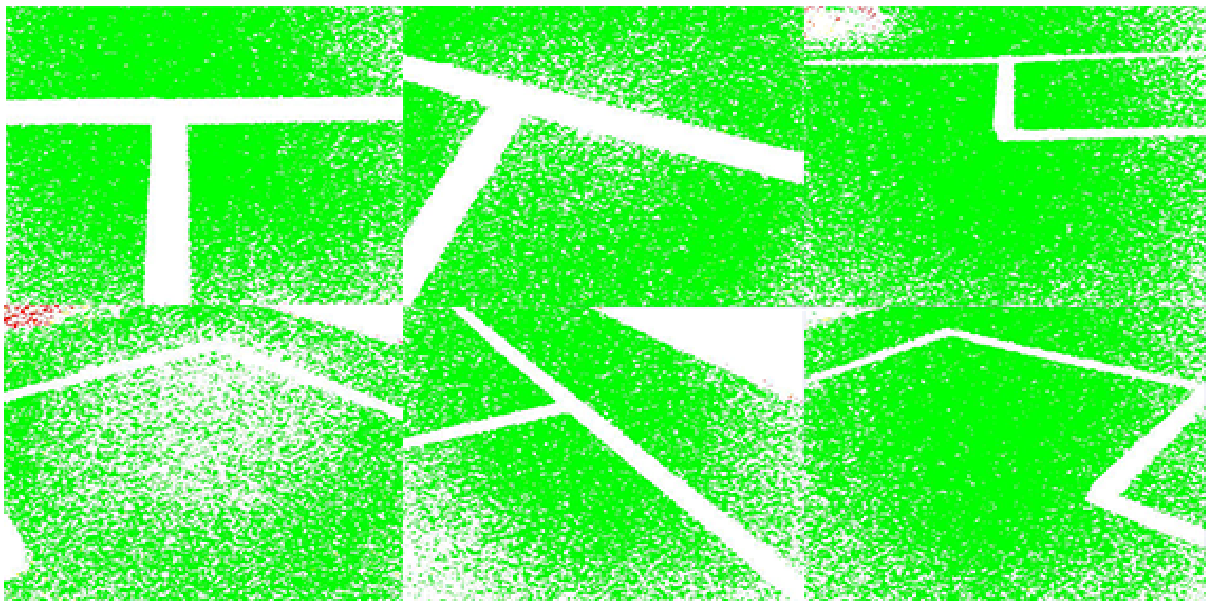
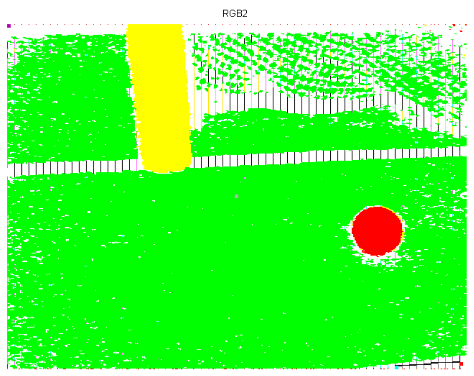


Figure 5.2: Corners and T-lines scenarios



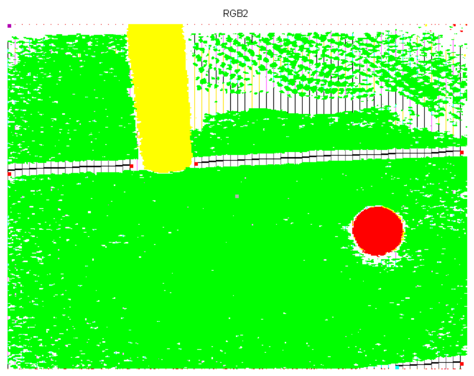


Figure 5.3: Goal's area "Groups" recognition (left) and line merging (right)

account the limited processing power available on the robot. We need to keep in mind that we are developing software for an embedded platform with limited computational resources, which are shared among all programmed operations, as well as middleware operations. Hence, the workload of each activity must be in accordance with the total workload on the robot. Given these constraints, we paid attention to the real-time performance of our implementation. It turns out that the execution was out of limits before we even added our lines recognition process and exceeded those limits even more

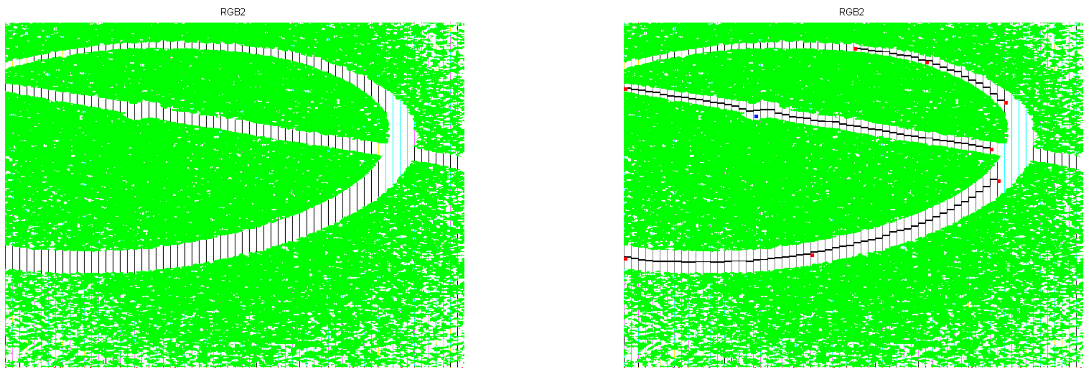


Figure 5.4: Center's "Groups" recognition (left) and line merging (right)

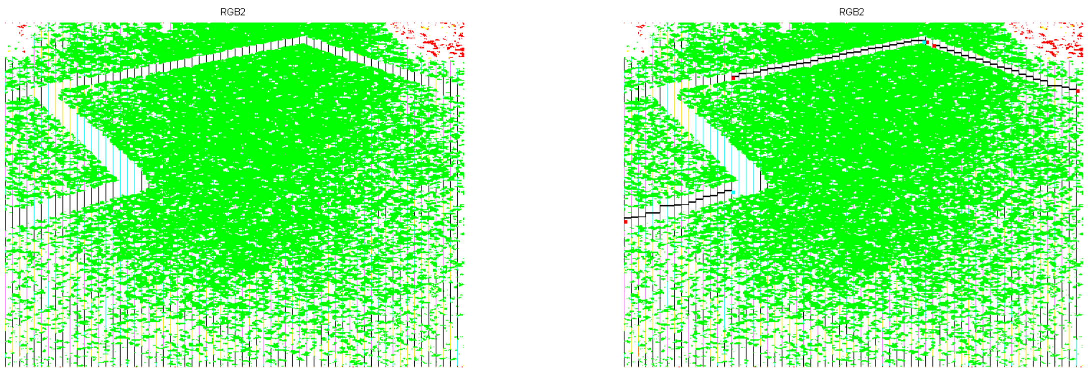


Figure 5.5: Corner's "Groups" recognition (left) and line merging (right)

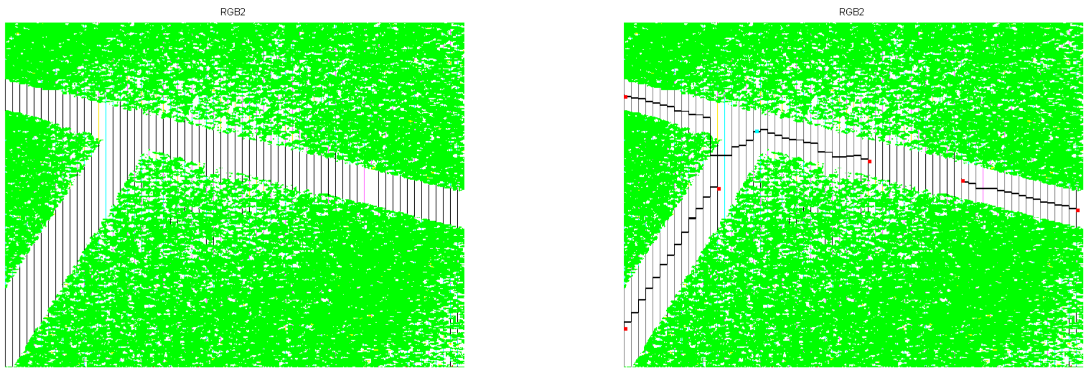


Figure 5.6: T-line's "Groups" recognition (left) and line merging (right)

after applying it. On the other hand, the execution is inextricably linked with how many pixels we decide to obtain and our mathematical background which can't be thrown away in a real-time performance. Thus, to lower the real-time requirements stated above we had to simplify every algorithm and manage to use as less pixels as possible. In addition to the cost of increasing the computational load and thus the overhead of the whole robot,this also results in under-activity of the other activities.

Chapter 6

Results

In this chapter we present the results of our work and show how it will benefit the rest of our RoboCup team. The best approach to show that a framework, like the one proposed, works well would be to compare the functionality and effectiveness of a team using our color table and line recognitions mechanism to a team which has no means such as them. For example, in a number of games between the two teams, one can record how many times each team has won. Unfortunately, in our case this comparison cannot be performed, because we do not have the means to conduct such large-scale experiments, i.e. enough functional robots to form two teams and sufficient time to run a large number of games. Thus, we could not provide such quantitative results, yet we will provide qualitative results by presenting certain scenarios that could occur during a robotic soccer game and compare them to our Matlab simulation output on the same scenarios. This way we can show that our approach can be really useful and works well even in the real, noisy and dynamic world.

6.1 Color recognition Results

The first step in our evaluation was to compare the two colortables results in a sample picture and in real time execution. To do so, we must have in our disposal many different light conditions in which we could perform a full game or a simulation at least. Unfortunately we had only two conditions to compare with, our ”Kouretes” lab and the amphitheater, that we perform our presentations for schools or other individuals that require to see our team. As we couldn’t form a full game we couldn’t see the longterm

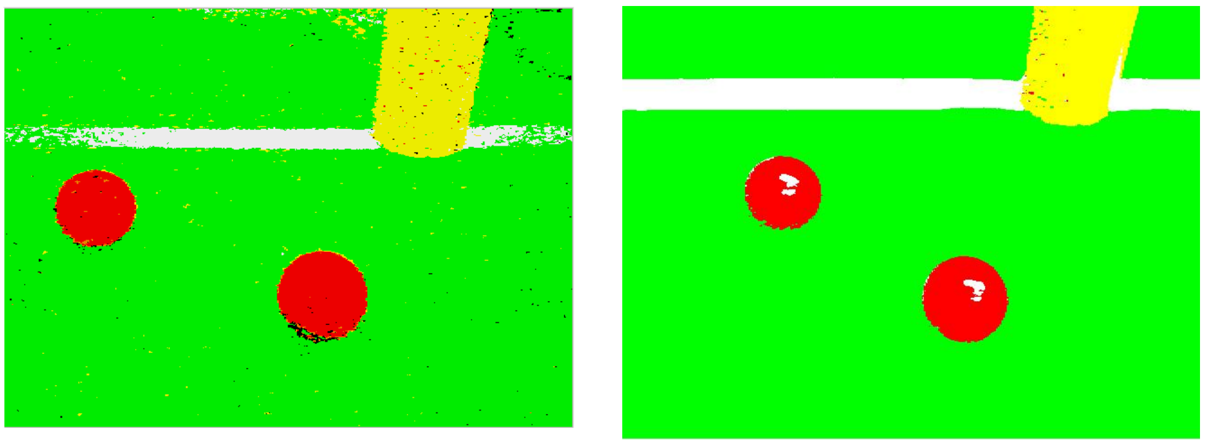


Figure 6.1: Colortable from the current mechanism in "Kouretes" (left) and from this thesis' mechanism (right)

results, and the short term results were quite the same. Although we didn't form a full game the results in a noisy environment are way better with our color recognition model, which we can see in the pictures from our matlab simulation Figure 6.1, as we do not accept shadows as colors and thus have a better result. In Figure 6.2, we present the result of color recognition side by side with the real RGB color model photo as the gain changes from 0 to 300.

6.2 Line recognition Results

Our first goal is to check the functionality of our lines recognition mechanism on some simple scenarios that can simulate real game phases. In this scenario (Figure 6.3) there are eight different angles/ways to see the circle and in all of those, we show the ability of the robot to recognize curves and lines. Unfortunately giving the complexity of the circle center finding mechanism we weren't able to visualize the results but through the consul of the robot we checked the results by values and were good as expected. The second scenario (Figure 6.4) represents the Corners recognition and consists of ten pictures of lines and corners recognition. The first six pictures are simple line recognitions and the last two also show the result of the corner that has been recognized. The third, and last scenario (Figure 6.5), represents the T-Lines recognition and consists of eight pictures of lines and T-lines recognition. Like the corners scenario the first four pictures are simple line recognitions and the last two show the result of the T-line that has been recognized.

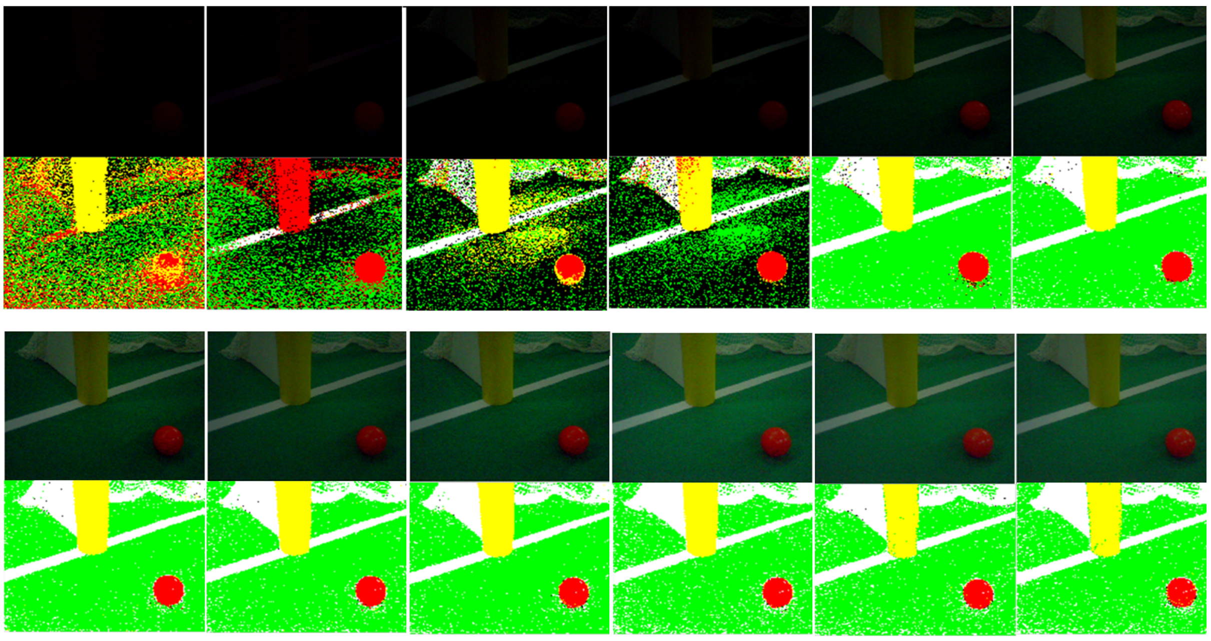


Figure 6.2: Real RGB photo side by side with our Color Recognition mechanism as the gain changes from 0 to 300

To visualize all this scenarios and also in order to check the color recognition as well as the lines recognition mechanism, we used our graphical tool KMonitor [17] . As we can see during this scenario the robot produces efficient results even if some times it doesn't merge all lines that can be merged.

In Figure 6.7 we present some goal lines recognition and Figure 6.7 the problem of an object recognition as well as background recognition, as was produced in our lab. As shown by these figures, we handle well all environment interference as well as obstacles. This is highly important because it indicates that our lines recognition model will not be confused by its' environment. Even in situations where it recognized some lines in the environment and merged them together there were no corners, T-lines or circle centers found thanks to our deleting mechanism.

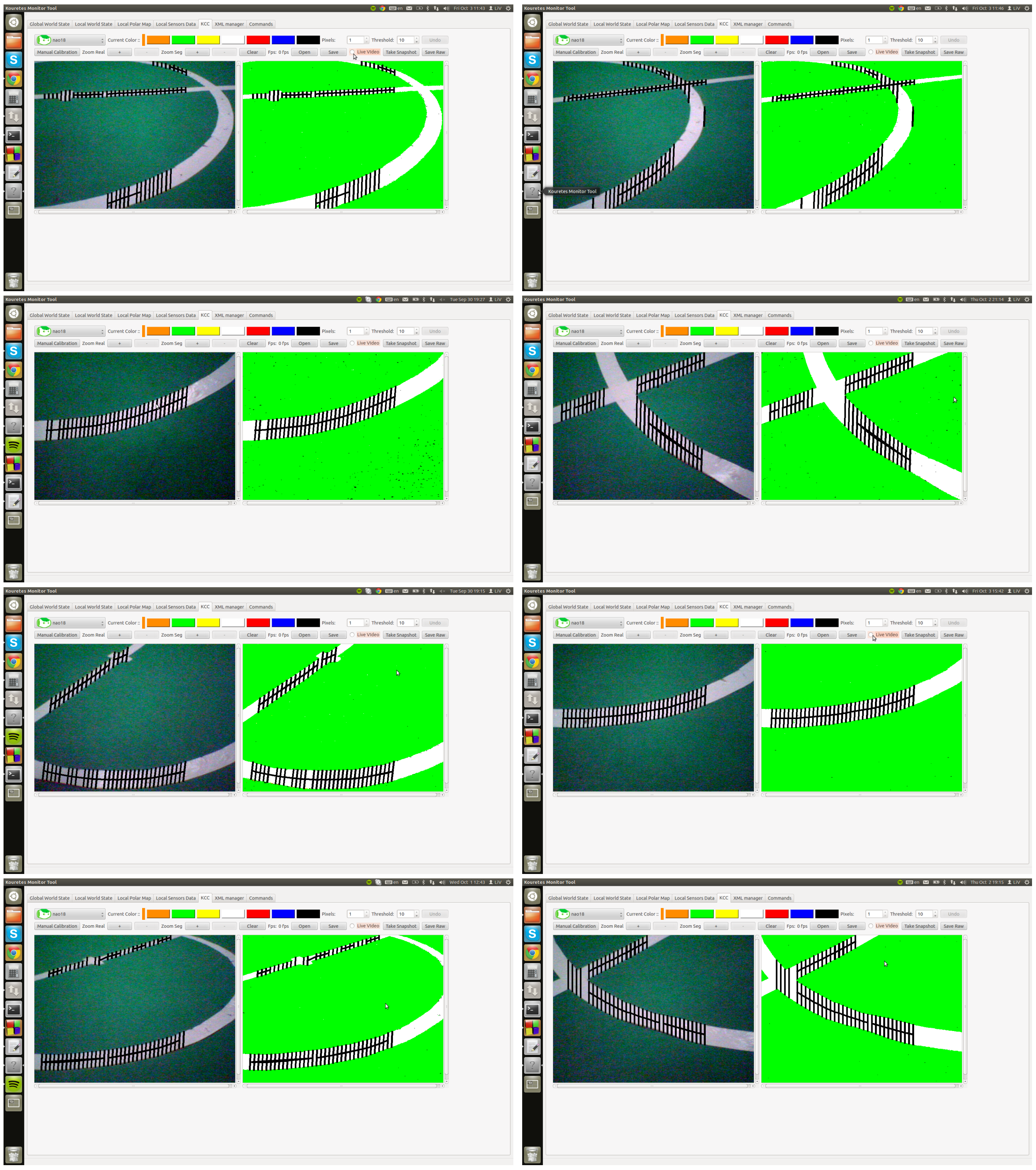


Figure 6.3: Scenario I: Recognizing circles and lines

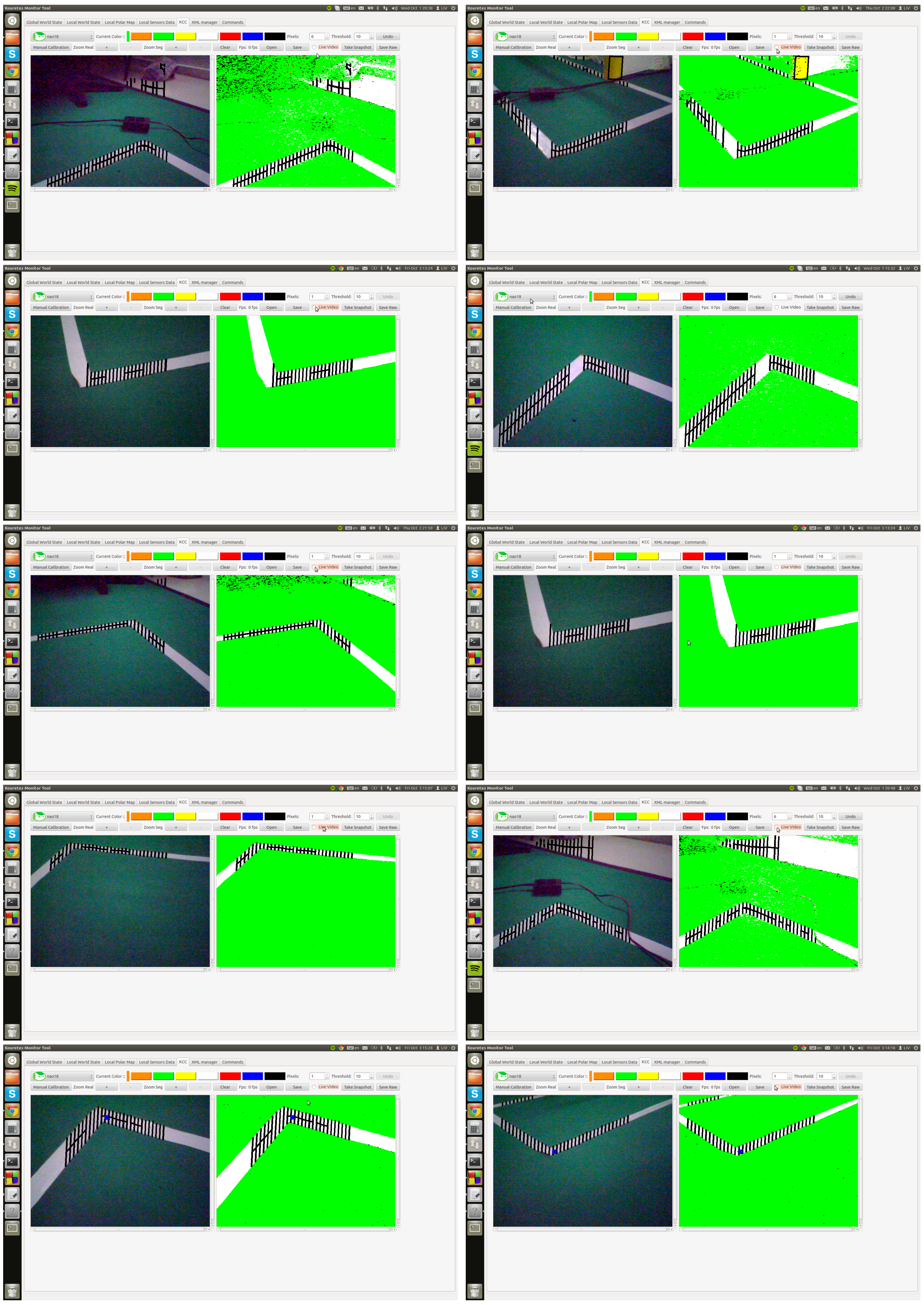


Figure 6.4: Scenario 2: Recognizing corners and lines

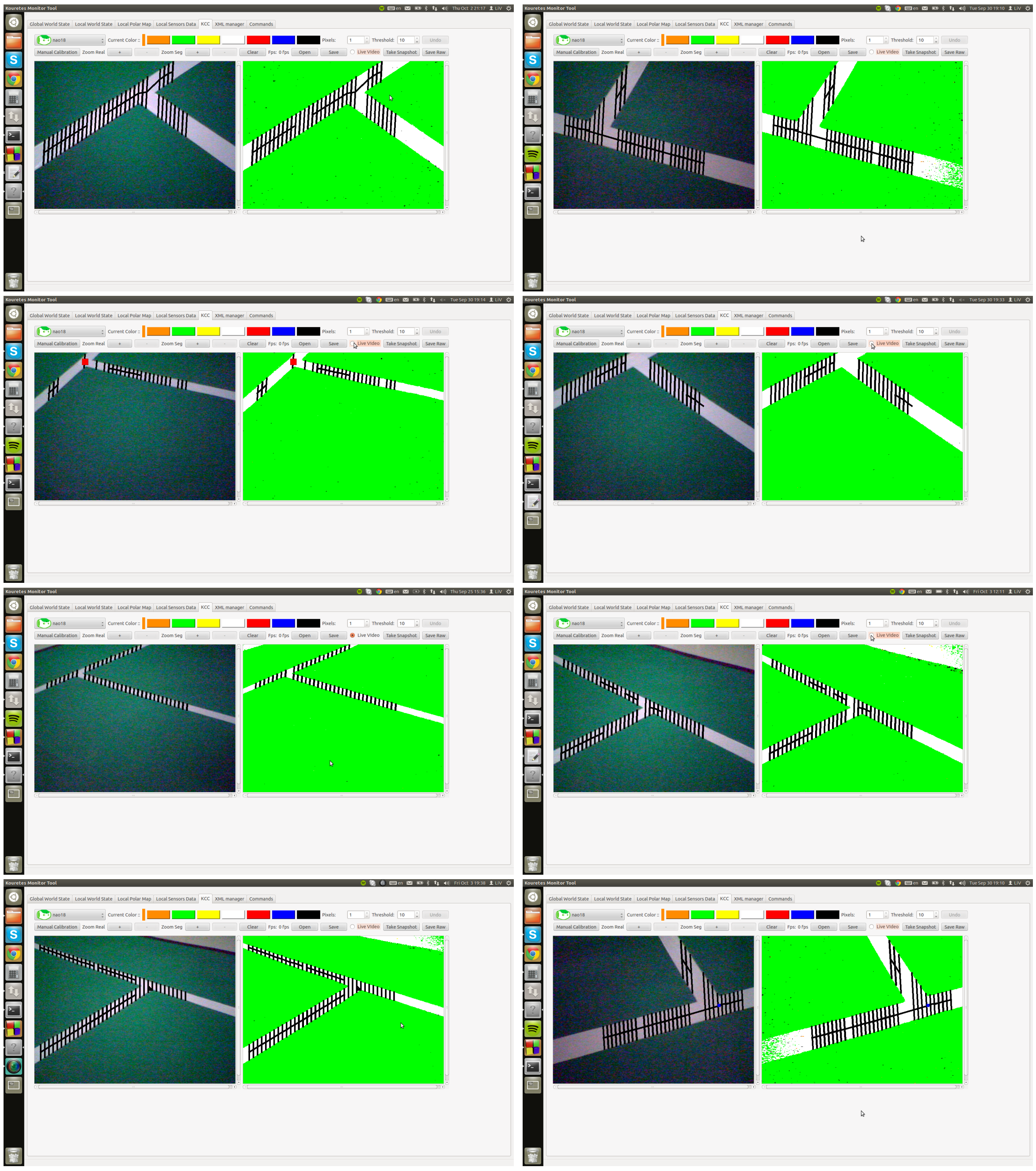


Figure 6.5: Scenario 3: Recognizing T-lines and lines

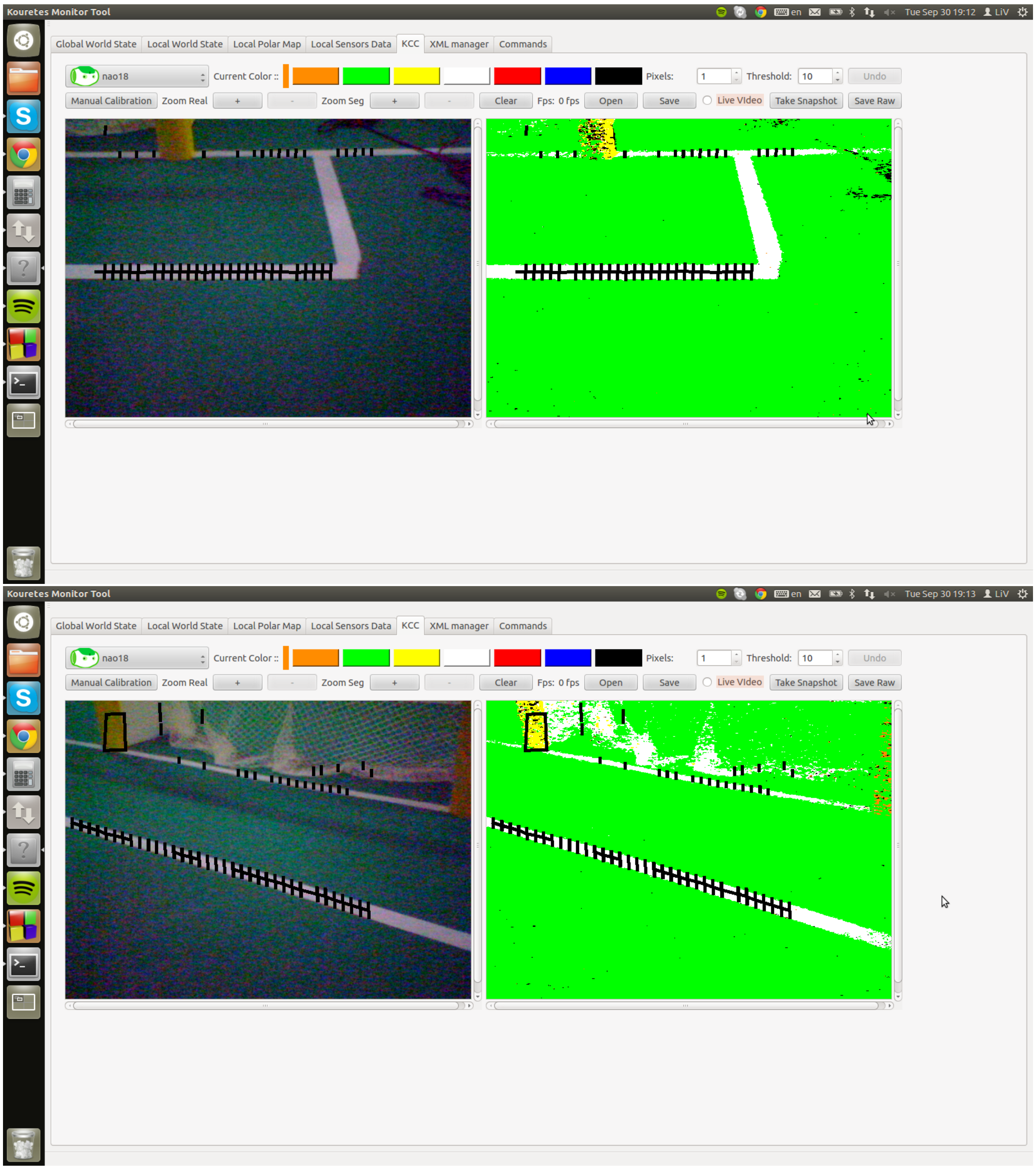


Figure 6.6: Goal lines recognition

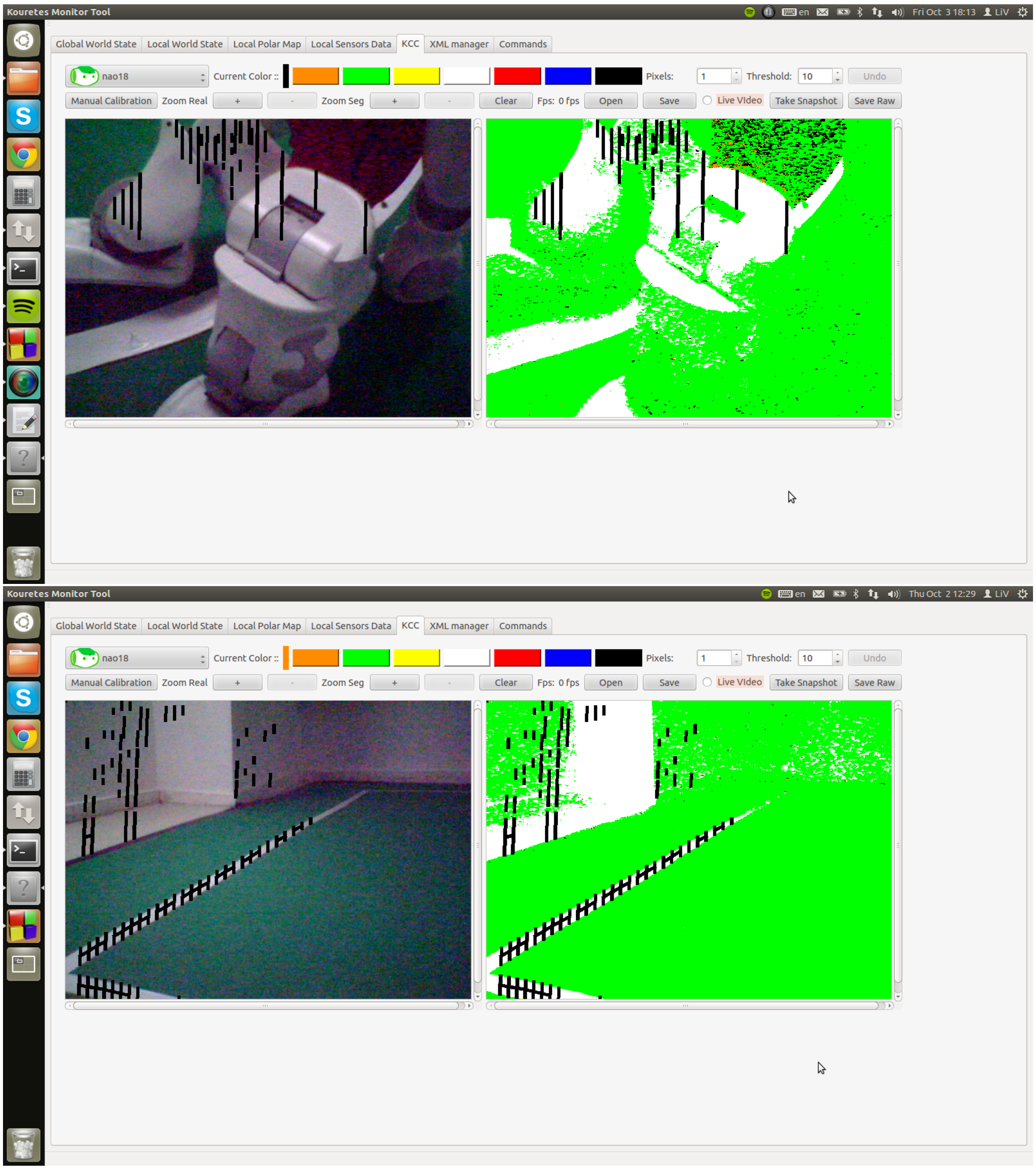


Figure 6.7: Obstacles recognition

Chapter 7

Conclusion and Future Work

7.1 Conclusion

This thesis summarizes the colors and lines recognition into the software architecture and repository of the RoboCup team Kouretes. As explained in Chapter 4, the end result of our work are both, a new more effective way of producing color tables and thus color recognition despite the light conditions, as well as recognizing corners, T lines and finding the center of the circle. As shown in Chapter 6, our work provides useful informations for our localization agent during RoboCup games and provides the opportunity of obtaining a better belief about our current position and our decision making mechanism.

7.2 Future Work

Regarding color recognition and color table production that we stated in this thesis, as we said in Chapter 4, this mechanism is almost completed but still needs consistent work, in order to provide more color tables in more light conditions. We expect our team in the future to add more samples in every event and give to our color recognition model more freedom. If so, in a short amount of time it would be able to fully recognize all light conditions. Another significant improvement would be to implement this mechanism into our newest version of Nao robot which has and HD camera and can produce far better results.

This thesis approached, for the first time in our team "Kouretes", a line recognition model. As mentioned in the color recognition future work the newest version of NAO robot has an HD camera which, given our 640x480 poor resolution, will add a significant improvement to our frame quality and will produce better results if added on it. For example, one pixel noise in our camera can change a line into curve or vice versa, if we hadn't add the "blind area". With better quality camera this wouldn't be an issue and we wouldn't lose some lines in order to avoid confusion between lines and curves. The new Nao has also a higher level processor, which will gain time for every thread in "Monas" and will give us the opportunity to add more complex mathematical procedures in order to identify objects or landmarks. Further improvement in our "Vision" agent, such as recognizing opponent robots, would produce more accurate results regarding the obstacles delete model that we stated in this thesis.

One of the most important features that could be implemented is a machine learning approach, like in our colors recognition model, that could benefit of past results and produce better ones in the future. This could also give the ability to reject our results if, by its' knowledge of the environment, can decide that they are inaccurate. Thus, it could possibly improve the objects and landmark recognition and eventually our team's overall performance leading to winning games.

References

[1] Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., Osawa, E., Matsubara, H.: RoboCup: A challenge problem for AI. AI Magazine **18**(1) (1997) 73–85 5

[2] RoboCup SPL Technical Committee: Standard Platform League rule book (2013) Only available online: www.tzi.de/spl/pub/Website/Downloads/Rules2013.pdf. 6

[3] Gouaillier, D., Blazevic, P.: A mechatronic platform, the Aldebaran Robotics humanoid robot. In: Proceedings of the 32nd IEEE Annual Conference on Industrial Electronics (IECON). (2006) 4049–4053 6

[4] Aldebaran Robotics: NAO documentation (2012) Only available online: www.aldebaran-robotics.com/documentation. 7

[5] Paraschos, A.: Monas: A flexible software architecture for robotic agents. Diploma thesis, Technical University of Crete, Greece (2010) 11

[6] Orfanoudakis, E.: Reliable object recognition for the RoboCup domain. Diploma thesis, Technical University of Crete, Greece (2011) 11, 25, 27, 39

[7] E., C.: Reliable object recognition for the RoboCup domain. Diploma thesis, Technical University of Crete, Greece (2011) 12

[8] Kargas, N.: Robust localization for the RoboCup standard platform league. Diploma thesis, Technical University of Crete, Greece (2013) 12, 30

[9] N., P.: Cooperative global game state estimation for the robocup standard platform league. Diploma thesis, Technical University of Crete, Greece (2013) 12, 30

[10] I., K.: Path planning for nao robots using an egocentric polar occupancy map. Diploma thesis, Technical University of Crete, Greece (2012) 12

[11] D, T.: Interleaving of motion skills for humanoid robots. Diploma thesis, Technical University of Crete, Greece (2012) 13

[12] Kofinas, N., Orfanoudakis, E., Lagoudakis, M.G.: Complete analytical inverse kinematics for NAO. In: Proceedings of the 13th International Conference on Autonomous Robot Systems and Competitions (ROBOTICA). (2013) 13

[13] Kofinas, N.: Forward and inverse kinematics for the nao humanoid robot. Diploma thesis, Technical University of Crete, Greece (2012) 13

[14] Paraschos, A., Spanoudakis, N.I., Lagoudakis, M.G.: Model-driven behavior specification for robotic teams. In: Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS). (2012) 13

[15] Paraschos, A., S.N.L.M.: Model-driven behavior specification for robotic teams. In: Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS). (2012) 13

[16] Topalidou-Kyniazopoulou, A., S.N.L.M.: A case tool for robot behavior development. In: RoboCup 2012: Robot Soccer World Cup XVI. Volume 7500 of Lecture Notes in Computer Science. Springer. (2013) 13

[17] Karamitrou, M.: Kmonitor: Global and local state visualization and monitoring for the robocup spl league. Diploma thesis, Technical University of Crete, Greece (2012) 13, 27, 59

[18] Wikipedia: Visual system— wikipedia, the free encyclopedia (2014) http://en.wikipedia.org/wiki/Particle_swarm_optimization. 14

[19] Goldstein., E.B.: Sensation and perception. In: Wadsworth Pub Co., eighth edition. (2010) 15, 16

[20] Wikipedia: Hsl and hsv — wikipedia, the free encyclopedia (2014) http://en.wikipedia.org/wiki/HSL_and_HSV. 16, 17

[21] Wikipedia: Rgb color space — wikipedia, the free encyclopedia (2014) http://en.wikipedia.org/wiki/RGB_color. 18

[22] Wikipedia: Normal distribution — wikipedia, the free encyclopedia (2014) http://en.wikipedia.org/wiki/Normal_distribution. 19, 31

[23] Committee, R.T.: Robocup standard platform league (nao) rule book (2014) http://www.informatik.uni-bremen.de/spl/pub/Website/Downloads/Rules2014.pdf. 19

[24] Georgakis, G.: Field landmark recognition and localization for the robotstadium online soccercompetition. Diploma thesis, Technical University of Crete, Greece (2012) 19, 31

[25] J., W.: Data analysis using the method of least squares: Extracting the most information for experiments. (Springer (2005)) 19

[26] Casey, J.: ”the circle. In: Ch. 3 in A treatise on the analytical geometry of the point, line, circle, and conic sections, containing an account of its most recent extensions, with numerous examples. (University of Michigan Library (2001)) 20

[27] Weisstein, E.: Circle-circle intersection 22

[28] E., M.: Dynamic multi-robot coordination for the robocup standard platform league. Diploma thesis, Technical University of Crete, Greece (2013) 30

[29] Wikipedia: Yuv — wikipedia, the free encyclopedia (2014) http://en.wikipedia.org/wiki/YUV. 34

[30] Wikipedia: Yuv — wikipedia, the free encyclopedia (2014) http://en.wikipedia.org/wiki/Covariance_matrix. 34