Obstacle Tracking in a Dynamic Environment

Application to Robot Soccer

Pedro César Alves Guerreiro Godinho Agostinho

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisor: Professor Pedro Manuel Urbano de Almeida Lima

Examination Committee

Chairperson:
Supervisor:
Members of the Committee:

Month and Year

RESUMO

No contexto de futebol robótico, uma das tarefas fundamentais que cada robot deverá ser capaz de realizar é a deteção de obstáculos que o rodeiam. Para além de um conhecimento instantâneo da posição dos obstáculos, é também de grande utilidade caracterizar e prever a sua evolução no tempo, permitindo assim um melhor planeamento de ações. Adicionalmente deverá ser capaz de classificar os obstáculos detetados, identificando os que correspondem a outros robots e a equipa a que pertencem.

O objetivo desta tese é pois o desenvolvimento de um sistema robusto de deteção, classificação e seguimento de obstáculos para robots da liga média de futebol robótico. O sistema foi separado nos seus constituintes, sendo cada um tratado individualmente.

Inicialmente foi abordado o problema da segmentação por cor, sendo dado enfase à robustez perante alterações na iluminação. De seguida foi desenvolvido um sistema de *ground-truth,* que não participando durante o jogo nas tarefas de deteção e seguimento, é no entanto relevante no âmbito deste trabalho, uma vez que é indispensável para a estimação de estatísticas de erro necessárias para o seguimento. Seguidamente discutem-se as tarefas da deteção e classificação propriamente ditas. Primeiro é abordado o modelo da câmara omnidirecional e seguidamente de que forma a imagem é processada de modo a extrair a posição dos obstáculos. Por último é discutido o procedimento para efetuar o seguimento dos obstáculos. Sendo abordados o filtro de Kalman, o algoritmo Húngaro para associação de dados e as regras de inserção e retirada de um obstáculo.

PALAVRAS-CHAVE: Segmentação por cor; deteção de obstáculos; seguimento de múltiplos objetos; futebol robótico; câmara omnidirecional; Sistema de *ground-truth*.

ABSTRACT

In the context of robot soccer, one of the fundamental tasks that each robot must be able to accomplish is the detection of its surrounding obstacles. Besides the knowledge of the instantaneous position of the obstacles, it's of great utility to characterize and predict their evolution with time, which allows for more sophisticated action planning. Additionally the robot should be able to classify the detected obstacles, identifying the ones that correspond to other robots and which team they belong to.

The objective of this thesis is thus the development of a robust system for the detection, classification and tracking of obstacles for robots of the middle sized robot soccer league. The system has been separated in its subsystems, and each one addressed individually.

Initially it was addressed the problem of color segmentation, with special emphasis on the robustness to illumination changes. Next, it was developed a ground-truth system, that although not directly involved in the detection and tracking during matches, it was indispensable for the estimation of error statistics of which the tracking is dependent. The detection and classification specific tasks are then address. Starting with the omnidirectional camera model followed by the image processing method used to extract obstacle positions. The final part deals with the obstacle tracking. The Kalman filter is presented, followed by the Hungarian algorithm used for data association and finally the rules for insertion and removal of obstacles from the tracking lists.

KEYWORDS: Color segmentation; Ground-truth system; Multiple object tracking; Obstacle detection; Omnidirectional camera; Robot soccer.

LIST OF FIGURES

LIST OF TABLES

TABLE OF CONTENTS

1 INTRODUCTION

1.1 BACKGROUND

Mobile autonomous robotics is a multi-disciplinary field of engineering, encompassing and interfacing a vast range of areas of knowledge. Robot Systems may be very complex and composed of many subsystems that must interconnect and work together seamlessly. It is in this context that robot soccer and in particular RoboCup medium sized league (MSL) robot soccer competitions stand out as an invaluable research and development tool. MSL competitions constitute a challenging and holistic application for mobile robotics and are an exceptional testing ground for the development of new methods and algorithms. The adversarial and competitive nature of robot soccer, not only adds an element of excitement for the researcher but also forces her/him to validate the produced work in an environment other than a controlled laboratory setting. This contributes to bridge the gap between theoretical research and real world applications.

An MSL robot must perform and integrate a vast number of tasks, ranging from low level hardware control, all the way to high level decision making in a cooperative multi-agent environment. The performance of some fundamental tasks such as navigation, obstacle avoidance or ball passing, depends strongly on the ability of the robot to be aware of obstacles in its surrounding environment. Additionally to being aware of the presence of the obstacle, it is also desirable to possess a characterization of its motion over time, which will allow to predict its evolution, even in case of intermittent detection. A detected obstacle should also be classified as to whether it is a robot, and if so, which team does it belong to. This work is concerned with the task of detecting, classifying and tracking of obstacles in the MSL competition environment.

The MSL soccer regulations impose a color structured environment, for example: the robots chassis should be predominantly black, cyan and magenta are used as team identifying color, the ball should be yellow, the field should be green, etc. To take full advantage of this color structure for the purpose of obstacle detection, it was identified the necessity to develop a robust color segmentation solution. Additionally, the development of a ground-truth system was considered relevant for its usefulness in the estimation of noise statistics. The color segmentation and the ground-truth subsystems were designed to be as generic as possible, allowing their utility to extend beyond the immediate scope of this work.

1.2 DEVELOPMENT SETUP

The development of this work demanded a good deal of data collecting and experimentation. For that purpose, the following laboratory equipment was put at disposal: half green playing field with white markings, with dimensions of approximately 12x9m; one fully operational MSL robot; a few other MSL robots at different levels of operability; two ceiling mounted fixed cameras at each end of the field for ground-truth purposes.

Figure 1.1 shows details of MSL robots of the type that was used and the laboratory environment. This robot has a three wheel omnidirectional drive system, with powerful motors that provide high acceleration and the ability to reach top speeds of about 3m/s. Each motor is equipped with an optical incremental encoder. The main sensor that the robot is equipped with is am omnidirectional camera. This camera is fitted with a fish eye lens having a field of view exceeding 180°. It captures frames at an average rate of 15 fps and is interfaced via Gigabit Ethernet. The robot is allowed to be equipped with other types of sensors such as ultrasonic or laser range finders. However, these type of active sensors are vulnerable to cross-talk when used simultaneously by different robots. It was found that problems associated with this type of sensor for this particular application outweigh their utility, and thus the option of their use was discarded.

Figure 1.1: From left to right: MSL robot; Omni-directional camera detail; MSL robots in the laboratory test field, with ground-truth markers installed.

The Robots middleware is based on the ROS framework. The obstacle tracking system was implemented as a ROS package and it relies on the use of ROS topics for data transmission. The following data is used as input: raw camera image topic, linear and angular velocity commands topic, incremental odometry topic. After each iteration the obstacle tracking system publishes to a topic that describes the current tracked obstacles. This information will then be used by higher level subsystems such as obstacle avoidance and others, which are out of the scope of this work.

1.3 SYSTEM OVERVIEW AND THESIS STRUCTURE

In figure 1.2 is depicted a high level diagram of the whole system. There are offline and online components of the system. The online components operate during the course of the soccer match, processing real-time sensor and control information. These components depend on configuration and calibration data generated by the offline components.

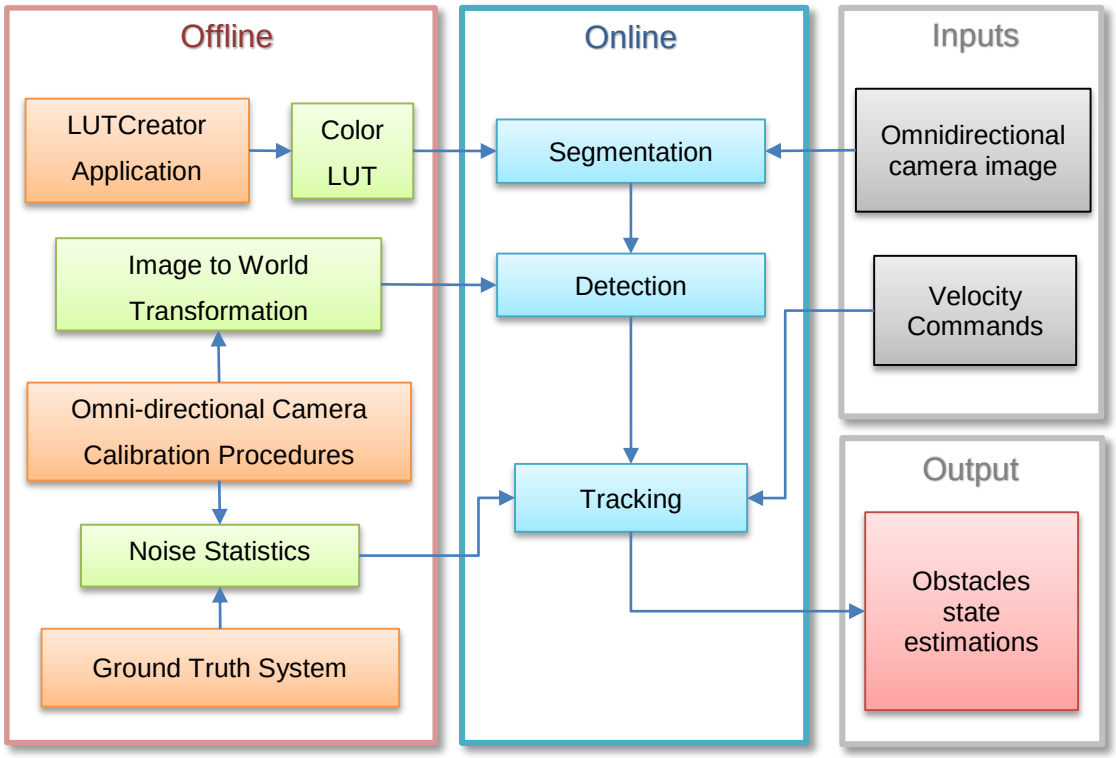

Figure 1.2: System overview

In the next chapters of this work, the system's individual components are addressed in the following manner:

- In Chapter 2 the problem of color segmentation is addressed. The HSV color model is presented and its advantages explained. The LUTCreator, a ROS integrated color segmentation application is presented and compared to previously used method. The problem of color constancy is discussed and some methods are proposed. Experimental procedures are described and the results interpreted.

- Chapter 3 deals with the Ground-truth system. First the transformation from camera coordinates to world coordinates is derived. A procedure for fast estimation of position error covariance using the Kabsch algorithm is suggested. The processing of the images for detection and tracking of the markers is described.

- Chapter 4 deals with the detection and classification of obstacles. The adopted model for omnidirectional camera model is presented. The procedure for estimating the measurement error covariance is described. Next, the image processing tasks are addressed, namely the ray casting detection method and the heuristic filtering strategies.

- Chapter 5 focuses on the obstacle tracking system. First the model of the dynamics is derived. The Kalman filter is discussed, as well as the procedure for estimating the process error covariance. The Hungarian Algorithm and its application in this context are presented. Finally the heuristic rules that control the insertion and removal of tracked obstacles are addressed.

- In Chapter 6 the results from previous chapters are synthetized and final conclusions are presented. A few avenues for future work are proposed, contemplating both the improvement of current solutions and the expansion of functionality.

2 IMAGE SEGMENTATION

2.1 INTRODUCTION

Segmenting an image consists of identifying groups of pixels that share some common attribute. In this particular case, the objective is identifying which pixels represent obstacles in general and in particular MSL soccer robots. MSL rules impose a color structured environment, where robots should be predominantly black with the colors cyan and magenta being used for team identification. Thus it's only natural to take advantage of this environmental constraint and chose color as the attribute used for segmenting an image.

The rest of this chapter is organized as follows: first the HSV color model is formally introduced, and its advantages explained. Methodologies for creating an RGB look-up-table for segmentation are analyzed and compared. The LUTCreator application is presented. The problem of color constancy is addressed and a few methods of implementation are derived. Finally, a set of experimental procedures are described. These are designed to evaluate the performance and robustness of the segmentation methods. The experimental results are presented and interpreted.

2.2 HSV COLOR MODEL

The RGB color model is a convenient machine representation of color, however there is no direct correspondence between RGB values and the psychological perception of color as the relation between red, green and blue channels in not intuitive. A color model more perceptually orientated is the HSV color model, which stands for hue, saturation and value. The HSV model encodes chromaticity in the hue and saturation, whereas brightness is encoded in the value parameter. The hue parameter in particular has great affinity with the visual sensation of color, and this is why the HSV model is particularly relevant for applications that require selection of color through human perception.

The HSV color space results of a geometrical nonlinear transformation of the RGB color space. The RGB color space is represented in a Cartesian space in which the red, green, and blue components are aligned with the coordinate axes. The RGB to HSV transformation will transform this Cartesian space into a cylindrical coordinate space in which the hue parameter will be represented in the angular coordinate, the saturation in the radial distance and the value in the

height. Figures 2.1(a) and 2.1(b) show the gamut of colors in the RGB and HSV spaces respectively.

To obtain the hue coordinate H, the color point in RGB space is projected into a plane designated as chromaticity plane. This plane is perpendicular to the axis of the grays which contains the black $(0,0,0)$ and the white point $(1,1,1)$. This projection of the RGB cube results in a hexagon in the chromaticity plane which is then morphed into a circle. The projected RBG cube and subsequent morphing is depicted in figure 2.1 (c). The way the hue parameter H is obtained from an RGB is expressed in the equations (2.1) to (2.4).

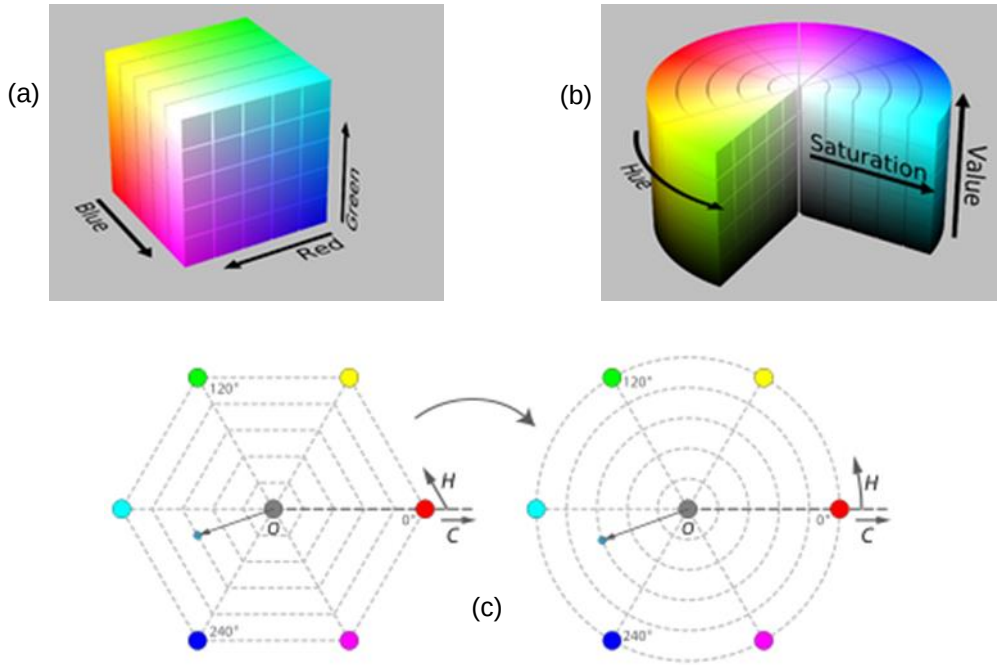

Figure 2.1: (a) RGB color space; (b) HSV color space; (c) Morphing of hexagon in chromaticity plane. Images by Jacob Rus used under CC BY license.

$$M = \max(R, G, B) \tag{2.1}$$

$$m = min(R, G, B) \tag{2.2}$$

$$H' = \begin{cases} undefined, & if\ M = m \\ \dfrac{G - B}{M - m}\ mod 6, & if\ M = R \\ \dfrac{B - R}{M - m} + 2, & if\ M = G \\ \dfrac{R - G}{M - m} + 4, & if\ M = B \end{cases} \tag{2.3}$$

$$H = H' \times 60° \tag{2.4}$$

The saturation parameter S and the value parameter V are given respectively by equations (2.5) and (2.6).

$$S = \frac{M - m}{M} \tag{2.5}$$

$$V = M \tag{2.6}$$

2.3 SEGMENTATION METHOD

The need for real-time performance makes computational efficiency an important concern. An RGB lookup table (LUT) is one of the fastest methods for color segmentation, because for each pixel all it takes is accessing an array position indexed by the pixel RGB component values (some mapping between RGB components and LUT indexes may be necessary depending on the ranges, but that can also be implemented via an array). The problem then lies on how to create the LUT. Prior to this work the method that was being used consisted of selecting the pixels that belong to the area that is desired to be segmented and inserting their RGB values in the LUT, from hereafter this method is referred as *RGB picking method*. This approach however has shown in practice to perform inconsistently. The problem with this method stems from the fundamental misconception that the selected RGB values are a representation of the perceptional notion of color. Instead these selected RGB values are a function of the lighting conditions presented at the time. Because of that, even though the segmentation might be very good in the conditions in which the LUT was created, even a small alteration in the lighting conditions will substantially degrade the quality of the segmentation produced by that LUT. A better alternative to selecting pixels is to select range intervals in HSV color space. As stated before, this allows to make a selection that better represents the visual sensation of color, which being the intended discriminating characteristic results in a more robust segmentation.

As an example consider the case showed in figure 2.2, observe that in figure 2.2(a) two cyan areas are present. Note that the one further away is in a recessed area and therefore appears a little darker than the one closer to the camera.

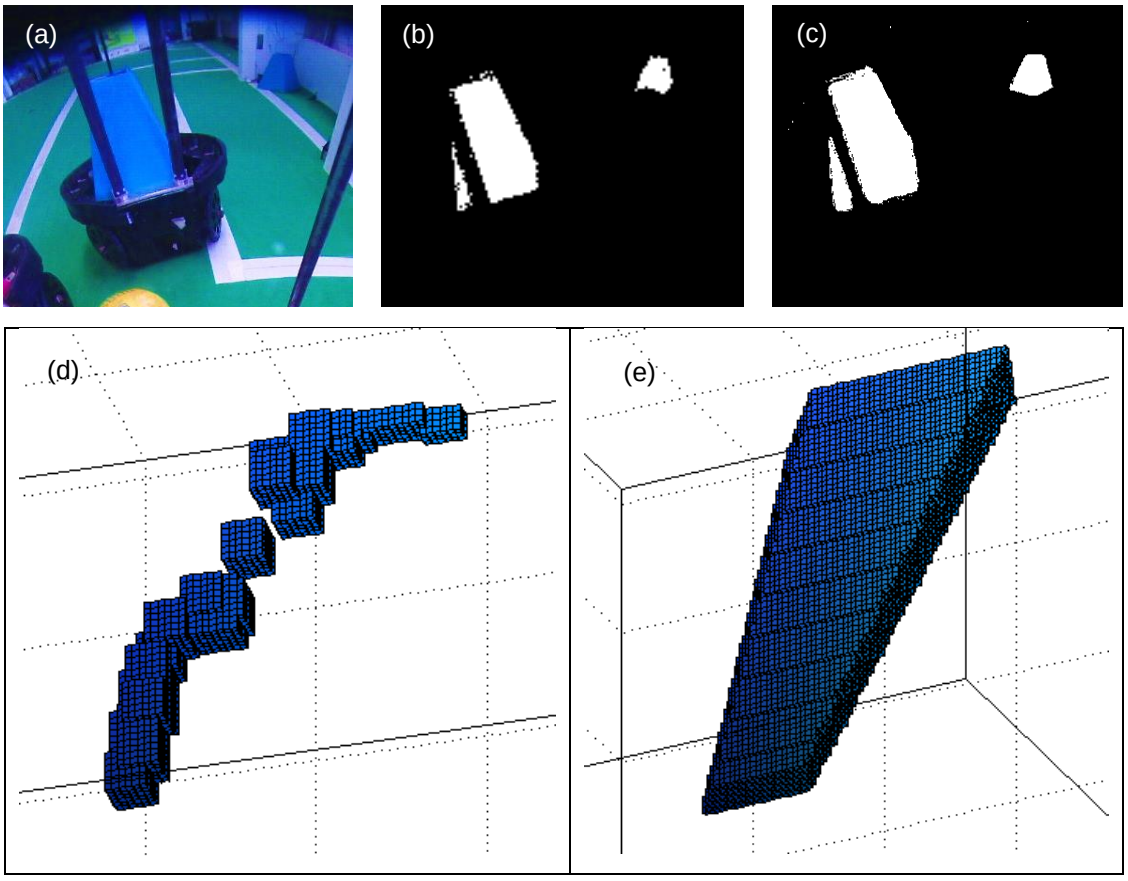

Figure 2.2: (a) Raw image ; (b) binary image resulting from the RBG picking method; (c) binary image resulting from the HSV intervals ; (d) LUT produced with the RBG picking method; (e) LUT produced by HSV intervals.

Figures 2.2(d) and 2.2(e) are each graphical representations of a LUT in RGB color space. The colored volumes represent the RGB values that correspond to the color selected for segmentation. Two LUTs were created, one created with the RGB picking method and the other with the HSV intervals method. In figures 2.2(b) and 2.2(c) it can be observed that both methods seem to produce good results, this observation is however misleading. Analyzing and comparing the morphology of the LUTs created by both methods helps to obtain a better insight on the performance differences. For the LUT created by selecting the HSV intervals, the rectangular volume in HSV space defined by the selected interval values, corresponds to a well-defined wedge shaped volume that resembles a truncated pyramid in the RGB color space. Visualizing the LUT created by the RGB picking it is clear that it is composed of disconnected irregular shapes. It is possible to recognize two distinct areas corresponding to the two cyan areas in the image. So the LUT created with the RGB picking method includes the RGB values visible in the current frame, while in the LUT created with the HSV intervals method those values are included in a bounding region . Furthermore this method allows easy control of the margin of this bounding region which can be expanded (carefully so that it doesn't intersect another color region) so that

the same RGB values will still be bounded by the region, even if the illumination conditions are moderately altered.

2.3.1 LUTCreator application

For the purpose of selecting the HSV intervals, it was recognized the need for a specialized software application. The usefulness of such software spans beyond the application addressed in this work, and therefore this software was designed for general use. The task of interactively tuning the HSV interval ranges for multiple colors would be very cumbersome to do without resorting to a graphical user interface (GUI), and so it was deemed necessary do design one, which culminated in the development of the LUTCreator application.

The LUTCreator GUI uses the QT framework and features custom-made graphical control elements. These widgets consist of colored slider bars with handles for lower and upper values which have shown in practice to provide a very intuitive, quick and controllable way of selecting channel intervals. The application allows an arbitrary number of colors to be selected. These colors are displayed in a list with several features for organization. Being integrated in the ROS middleware, LUTCreator subscribes to an image topic and publishes a segmented image to another topic. This segmented image is an 8 bit grayscale image in which each pixel values is the assigned color code. The working sessions can be saved and later loaded for easy and quick editing. Of course, the LUTCreator application also exports LUT files with options for resolution and format.

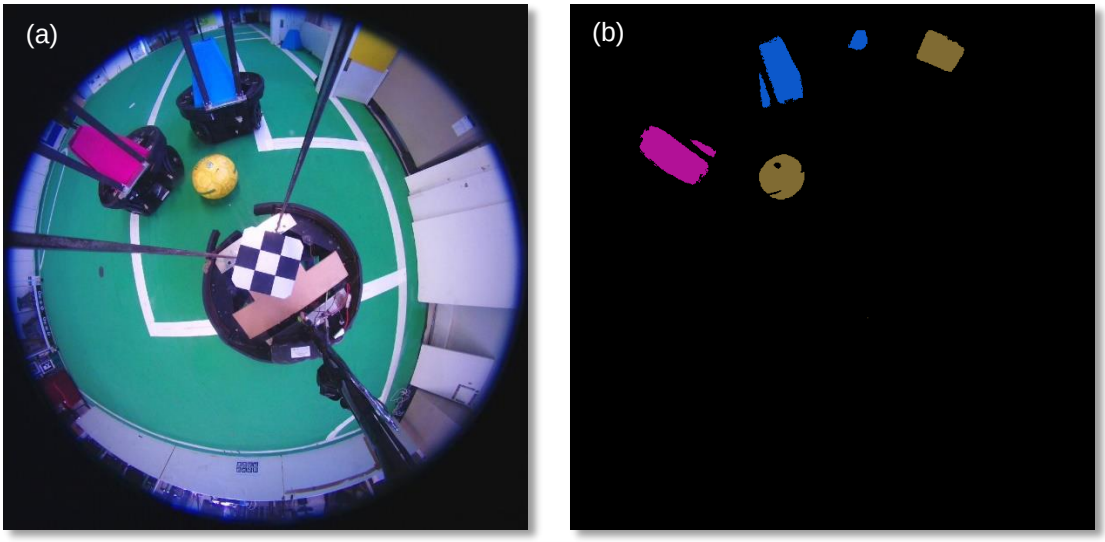

Figure 2.3: (a) Input Frame; (b) Output Frame

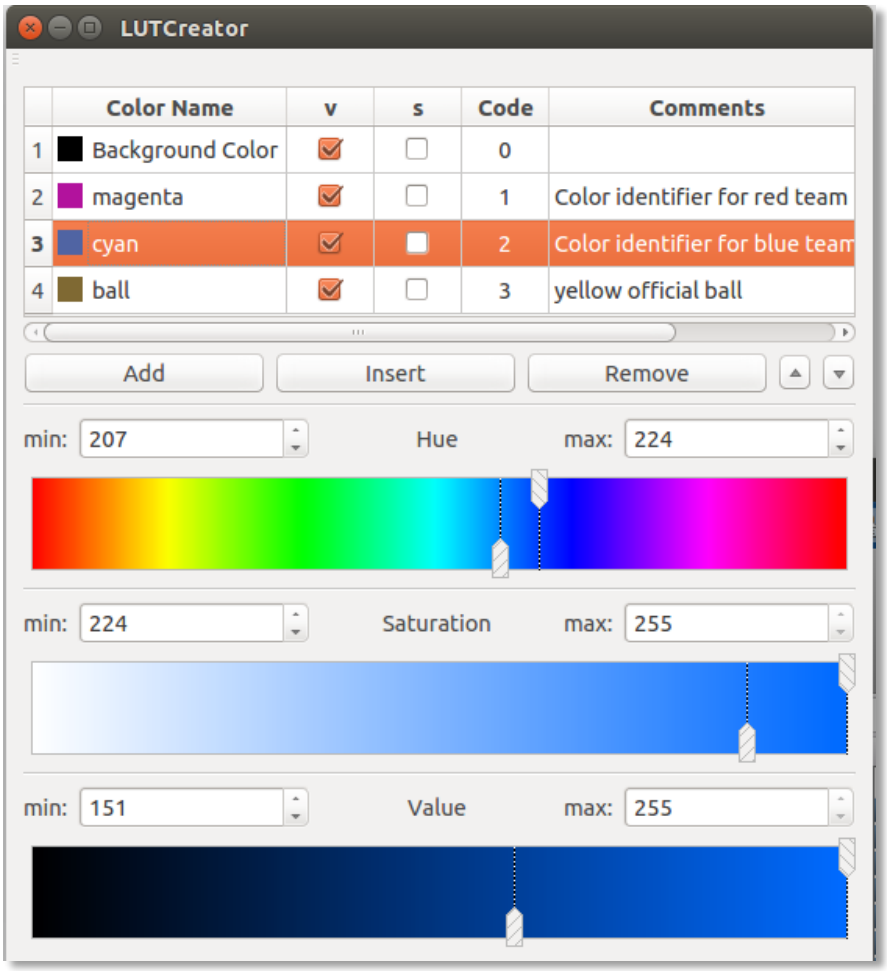

Figure 2.4: LUTCreator control panel.

There are several ways in which the LUTCreator is able to be used in an online application, first it can be used directly in the ROS topic pipeline, that is if the GUI overhead is not a problem for that particular application. In most cases however, CPU is too precious of a resource to be wasted unnecessarily, and a small API can also be used to import and use a LUTCreator LUT file or session file. Using a LUT file provides the fastest segmentation, however it has some minor drawbacks such as having to export (which usually takes a few seconds) and keep large a LUT file, and in case one decides to make the LUT smaller, the decreased resolution will have an impact on the accuracy of the segmentation. It is also possible to use a session file, which is a file that contains only the information that is displayed in the LUTCreator control panel such as the HSV interval values. Being very lightweight and fast to save/load, makes them a convenient albeit less computationally efficient alternative to using a LUT file. The segmentation using the session file requires color conversion from the camera RGB to an HSV representation for every frame, and each pixel has to be checked to see if the value falls in any of the selected intervals.

These are per-pixel operations and can take advantage of GPU parallel processing. In table 2.1 are presented the average segmentation processing times on an *Intel Core i73610QM* CPU and an *NVIDIA GeForce GT 650M* GPU. It can be observed that the average segmentation time using a session file is about an order of magnitude greater than the average segmentation time using a LUT. However, when using the GPU to compute the color conversion, this difference gets significantly reduced. Note that the GPU via the CUDA platform is used in this example only to compute the RGB to HSV conversion using an available OpenCV method from the GPU module. The results from a custom-made implementation are expected to be even better.

Method	LUT file	Session file	Session file with CUDA accelerated color conversion
Average Time (500 samples)	0.002943 s	0.028129 s	0.009993 s

Table 2.1: Average color segmentation processing times

2.4 COLOR CONSTANCY

Color constancy is the ability to distinguish a color of an object, invariant of the color of the illuminant. Many strategies have been proposed to computationally achieve (although not necessarily mimic) this ability of the human visual system. Most approaches to this problem consist of finding an estimate of the scene illuminant color and then perform the chromatic adaptation, so that the output scene colors are perceived as if under some desired illuminant, usually achromatic.

The reason for addressing in this context the topic of color constancy arises from situations that often occur in the MSL environment, which have the potential to degrade the performance of the color segmentation. For example, one situation that may occur due to the relative large dimensions of the field, is non-uniform illumination. Consider for example the situation where there are windows located at one end of the field, in this case depending on type of artificial light sources and on the time of day there may be a considerable difference in color of the illumination from one end of the field to the other. For the sake of simplicity and computational performance, the algorithms presented here make the assumption of the presence of only one illuminant. The previous situation can however be accepted in this category, if the transition of illumination is smooth. In this case, as the illumination color changes progressively in space, the illumination may be considered as locally uniform. Another situation that may occur is a change in the illuminant color over time, this occurs in for natural illumination, in which its color depends on the time of day, but it may also happen with some artificial light sources. Consequently it may happen that a color segmentation system that was calibrated at some point in time, has its performance progressively degraded over time.

There are a vast number of proposed algorithms, however the objective is merely to evaluate the impact of color constancy techniques on the robustness of the color segmentation and not to exhaustively evaluate color constancy methods. Because of that, only two techniques were chosen for testing, the *Max-RGB* method, and the *Gray-World* assumption. The reason this methods were chosen in detriment of others is twofold, first these are perhaps the most popularized methods and are commonly used as a benchmark for testing, second they have the lowest computational cost. For a thorough expose on the topic of color constancy and its applications refer to [1] and [2].

2.4.1 Sensor model

Before introducing the color constancy methods it's necessary to describe the adopted sensor model. The formalism used in the derivation follows as described in [1]. In equation (2.7) is represented the model for the intensity $\boldsymbol{I}$ of a general sensor at position $\boldsymbol{x}_I$ under uniform illumination. The factor $G(\boldsymbol{x}_{Obj})$ represents the object lighting geometry factor, which depends on the position $\boldsymbol{x}_{Obj}$, $R(\lambda, \boldsymbol{x}_{Obj})$ is the reflectance of the object that depends on $\boldsymbol{x}_{Obj}$ and on the wavelength λ , $L(\lambda)$ is the radiance of the light source and finally $\boldsymbol{S}(\lambda)$ is the sensitivity of the sensors.

$$\boldsymbol{I}(\boldsymbol{x}_I) = G(\boldsymbol{x}_{Obj}) \int R(\lambda, \boldsymbol{x}_{Obj}) L(\lambda) \boldsymbol{S}(\lambda) d\lambda \tag{2.7}$$

This model can be greatly simplified if the sensitivity of the sensors is considered to be of very narrow band, such that it can be modeled as a delta function. If we consider three sensors with sensitivity given by a delta function at λ_i with $i \in \{r, g, b\}$ each corresponding to wavelengths in the red, green and blue parts of the spectrum respectively. The simplified model thus becomes:

$$I_i(x, y) = G(x, y) R_i(x, y) L_i \tag{2.8}$$

2.4.2 Max-RGB algorithm

The Max-RGB algorithm relies on the idea that there is some bright patch in the image and that color of the pixels belonging to that patch can be used as an estimate of the color of the illuminant. In practice is not necessary to identify any patch in the image, only to search all pixels for the maximum intensity value for each channel. Those maximum values are assumed to be part of the supposed bright patch, and thus the color they represent should be the color of the illuminant.

A white patch should reflect the maximum light possible for each band, if $R_i(x,y) = 1$ for $i \in \{r,g,b\}$ and let $G(x,y) = 1$ i.e. the patch surface normal is oriented perpendicular to the camera, then the sensor captures the color of the illuminant.

$$I_i(x,y) = L_i \quad , \quad (x,y) \notin Mask \tag{2.9}$$

The omnidirectional image contains a black vignette that correspond to areas of the sensor that don't receive any light from the lens. The values of the pixels in this area should not be included in the calculations. The set $Mask$ contains all the pixel positions that correspond to areas outside the useful part of the image.

Assuming a linear relationship between the response of the sensor and pixel colors, where $c_i(x,y)$ is the pixel color for each channel $i \in \{r,g,b\}$

$$c_i(x,y) = G(x,y)R_i(x,y)L_i \quad , \quad (x,y) \notin Mask \tag{2.10}$$

For every channel the maximum value is found,

$$L_{i,max} = \max_{x,y}\{c_i(x,y)\} \quad , \quad (x,y) \notin Mask \tag{2.11}$$

Each maximum value is used to scale the respective channel values so that the range of values in that channel becomes $[0,1]$, consequently eliminating the effects of the color of the illuminant.

$$o_i(x,y) = \frac{c_i(x,y)}{L_{i,max}} = G(x,y)R_i(x,y) \quad , \quad (x,y) \notin Mask \tag{2.12}$$

In in first row of histograms in figure 2.5 it is possible to observe an example of what happens to the histogram of the RGB channels. On the left is depicted the original histogram for one channel where it can be observed that the maximum pixel value is less than 1. On the right we can observe the histogram after the Max-RGB algorithm is applied, where it appears to have been "stretched" across the entire tonal range. If this is done for the three channels, and assuming that the brightest values in the histogram are indeed the representation of some white object in the scene, the white objects will actually appear white in the image, i.e. will have an RBG value of (1,1,1).

This simplistic approach may work well for most cases, but if sensor noise of even specular reflections are present in the image in the form of bright pixel intensities, these may be incorrectly assumed to represent the white color and this results in an incorrect estimation of the illuminant. This situation is depicted in the middle row of histograms in figure 2.5, in which the small island in the histogram (exaggerated in size for clarity) represents noise. To prevent this, instead of using $L_{i,max}$ as the estimate of the illuminant, the histogram $H_i(k)$ is computed for each channel and the estimate L_i is chosen such that the number of pixels with intensity higher or equal that the intensity at bin j_i is at least a percentage p of the total number of unmasked pixels n. This is expressed (2.13) and (2.14) where n_b is the number of bins in the histogram.

$$L_i = c_i(j_i) \tag{2.13}$$

$$pn \leq \sum_{k=j_i}^{n_b} H_i(k) \quad and \quad pn \geq \sum_{k=j_i+1}^{n_b} H_i(k) \tag{2.14}$$

The example at the bottom row of figure 2.5 illustrates this idea, where the histogram gets scaled beyond the tonal range and the pixels in red have their values clipped. The percentage value p is selected in a way that provides good robustness to noise but at the same time minimizes the clipping, a value of 1% is adequate most of the times.

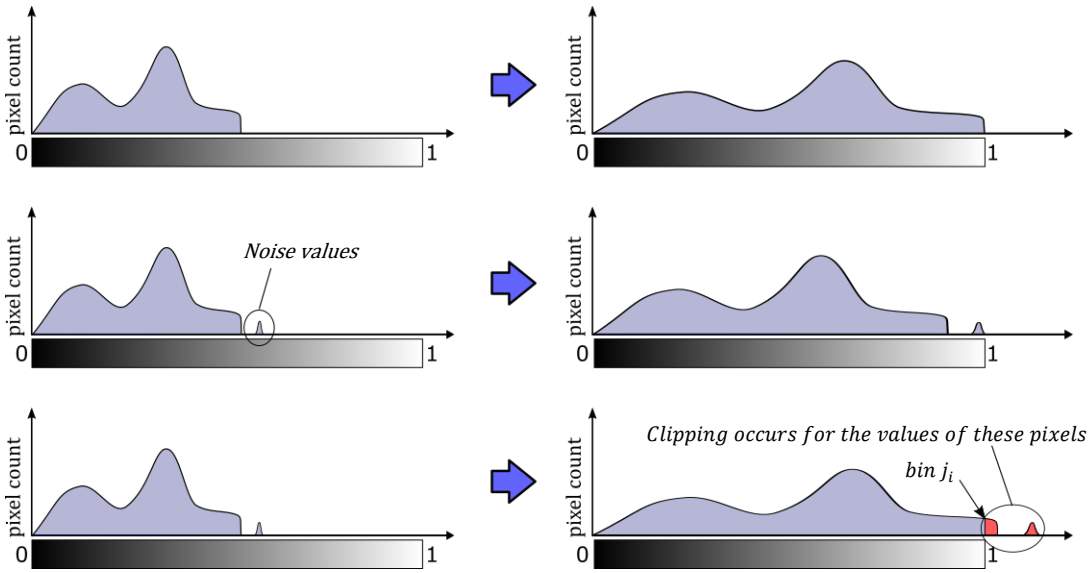

Figure 2.5: Examples of the effect of the Max-RGB algorithm on the histogram.

In this particular application there is a considerable amount of frame area that is occupied by the image of the robot's chassis directly under the camera. Thus another ad hoc variation of this algorithm consists on restricting the search for the maximum pixel values to a selection of pixels that corresponds to the image of a small (about 10x10 cm) checkerboard pattern fixed to the robot in a way that is not visible to other robots and thus still complies with the MSL rules. This version of the algorithm will be hereafter as designated as *Max-RGB on a patch*.

2.4.3 Gray-world assumption

Possibly the most popular white balance method, it is not so much an algorithm as it is an idea from which have spawned several different algorithms, the derivation that ensues is according to the one found in [1] . The idea is that under most conditions the average color of a scene is a neutral gray color. Therefore the average pixel color a_i can be used for obtaining an estimate of the color of the illuminant L_i, as indicated in the following expressions where n is the number of unmasked pixels.

$$a_i = \frac{1}{n}\sum_{x,y} c_i(x,y) \quad , \quad (x,y) \notin Mask \tag{2.15}$$

$$= \frac{1}{n}\sum_{x,y} G(x,y)R_i(x,y)L_i \quad , \quad (x,y) \notin Mask \tag{2.16}$$

$$= L_i \frac{1}{n}\sum_{x,y} G(x,y)R_i(x,y) \quad , \quad (x,y) \notin Mask \tag{2.17}$$

Both the geometry factor G and the reflectance R_i can be considered as independent random variables, as there is no correlation between the shape and the color of an object. If there are many different colors in the scene it's reasonable to assume that the reflectance R_i is uniformly distributed in the range [0, 1].

$$E[GR_i] = E[G]E[R_i] = E[G]\left(\int_0^1 x\,dx\right) = E[G]\frac{1}{2} \tag{2.18}$$

For a large n the average color becomes:

$$a_i = L_i \frac{1}{n} \sum_{x,y} G(x,y) R_i(x,y) \quad , \quad (x,y) \notin Mask \tag{2.19}$$

$$\approx L_i \, E[GR_i] \tag{2.20}$$

$$= L_i \, E[G] \, \frac{1}{2} \tag{2.21}$$

Consequently an estimate of the color of the illuminant is obtained,

$$L_i \approx \frac{2}{E[G]} \, a_i = f \, a_i \tag{2.22}$$

The color of the output pixel can now be calculated by dividing the current pixel color by the estimate of the illuminant,

$$o_i(x,y) = \frac{c_i(x,y)}{L_i} \approx \frac{c_i(x,y)}{f \, a_i} = G(x,y) \, R_i(x,y) \quad , \quad (x,y) \notin Mask \tag{2.23}$$

The f factor affects the intensity of the color by scaling all channels equally, if we assume $E[G] = 1$, then $f = 2$. A sometimes useful alternative way to estimate f is to use a value of f such that clipping of pixel intensities higher than 1 occurs for 1% of all pixels.

2.5 EXPERIMENTAL PROCEDURES

In order to evaluate and compare the segmentation methods, both by themselves and in conjunction with the color constancy methods, a set of three experiments was devised. Each of the experiments is design to evaluate the robustness to a specific type of illumination change. The changes of illumination attempt to replicate possible scenarios that can occur in MSL competitions, which normally take place in spaces where many times it is not possible to attain constant illumination conditions. The types of illumination change that are replicated by the experiments are respectively of intensity, geometry and color.

For each experiment two frames designated as **A** and **B** are captured. There is no change in the scene from frame **A** to frame **B**, only the illumination is modified between the captures. In order to obtain a metrical evaluation of the performance of the segmentation, a reference segmentation image is created for comparison. This reference is created manually so that the selected pixels correspond as closely as possible to the areas that are supposed to be segmented. The results of the segmentations created with the HSV intervals and the RGB picking methods are then compared to this reference. If a pixel in the segmentation corresponds to a pixel in the reference, it is considered a true positive, instead if it corresponds to a pixel that is not part of the reference selection, it is considered a false positive. The results are presented in standard precision and recall measures, where precision is the fraction of segmented pixels that are true positives, while recall is the fraction of reference pixels that were segmented.

While interpreting the experimental results, there are some considerations that should be kept in mind. The reference is manually selected and the boundaries of the selected areas are fuzzy, for that reason the reference selection is somewhat subjective, i.e. it depends on the interpretation of the person that created it. This ambiguity, may cause some false positives to occur near the border of the reference area simply because the reference area might have been selected in a conservative manner, and the perimeter of the segmentation may be a little expanded in comparison to reference area. Also, because the environment is color structured the colors to be segmented are well separated in the chromaticity plane and the number of false positives is always expected to be low. Because of these reasons, a degradation of the segmentation effectiveness will manifest itself primarily as a decrement in the number of true positives with a comparatively small change in the number of false positives. It should also be noted that even though a recall value close to 1 is obviously desired, it doesn't necessarily mean that a higher value, for example 0.85 is much better than say a value of 0.7, as both cases may correspond to a good detection. Considering also that the main purpose of this experiments is to evaluate how the segmentation holds when the illumination changes, the most meaningful indicator that should be considered when evaluating the results is the variation of recall values between frames **A** and **B**. Nevertheless the evaluation of the overall performance should not exclude the observation of the segmented images alongside the numerical results, leading to a better and more insightful interpretation of the overall results.

2.5.1 Experiment 1: Illumination intensity variation

No Pre-Processing			
LUT	Frame	Precision	Recall
HSV interval	A	0.955	0.778
	B	0.894	0.879
RGB picking	A	0.995	0.686
	B	0.842	0.041
Gray-World Assumption			
LUT	Frame	Precision	Recall
HSV interval	A	0.944	0.836
	B	0.857	0.910
RGB picking	A	0.999	0.661
	B	0.807	0.014
Max-RGB Algorithm			
LUT	Frame	Precision	Recall
HSV interval	A	0.927	0.866
	B	0.871	0.928
RGB picking	A	0.997	0.670
	B	0.846	0.053
Max-RGB on a Patch			
LUT	Frame	Precision	Recall
HSV interval	A	0.902	0.844
	B	0.889	0.862
RGB picking	A	0.994	0.695
	B	0.962	0.138

Table 2.2: Experiment 1 results.

In this first experiment, the change in illumination is mostly of intensity. It is the kind of situation that can be experienced if for example the distribution of light sources is not uniform across the field. A combination of natural and fluorescent illumination was used, with the intensity being controlled by the changing the number of lit fluorescent lamps.

The HSV intervals segmentation stayed consistent across the illumination change, as can be attested by the elevated recall values in all cases and a difference in recall values from image **A** to image **B** not higher than 0.101. The segmentation with the RGB picking method degraded to the point of being useless since the recall values in image **B** are very small and the recall values fall an average of 0.62. It can be seen on the figures 2.6 to 2.9 that the detection of the targets would be almost completely lost. The color constancy algorithms did not have a relevant effect in any of the cases.

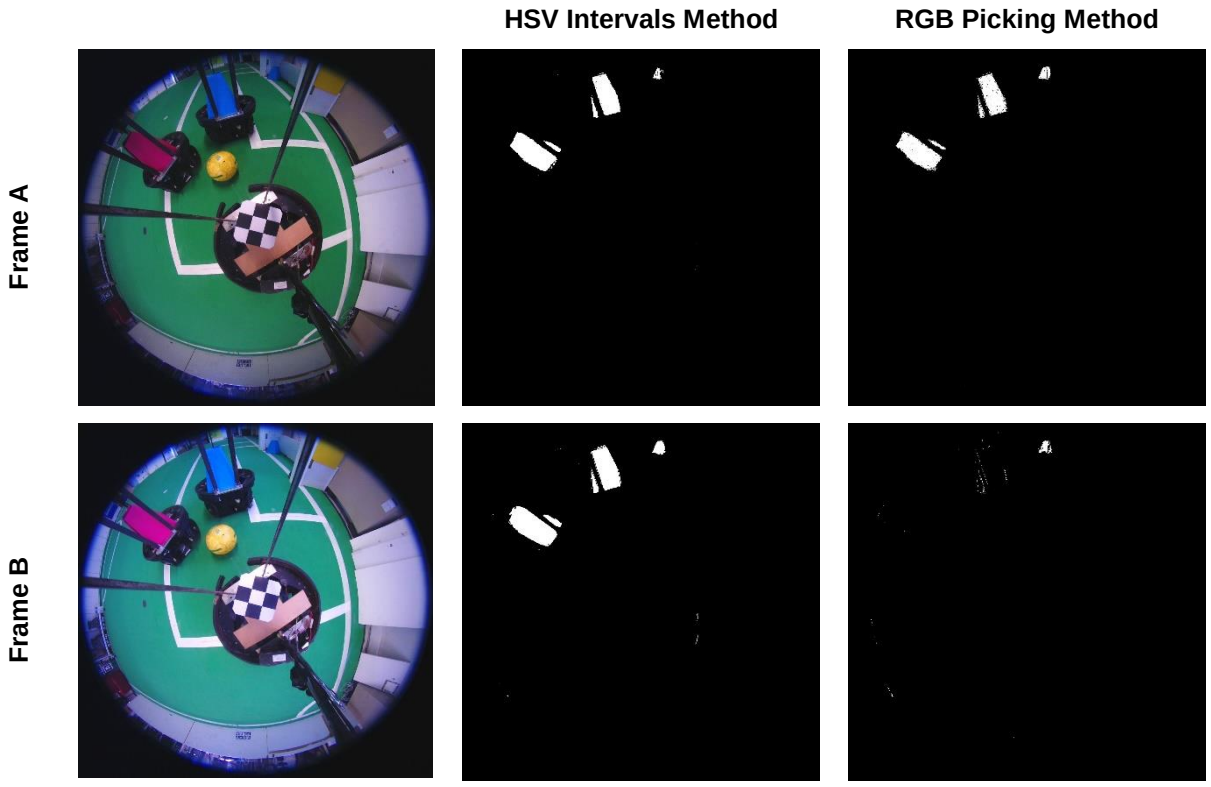

Figure 2.6: Experiment 1 with no color processing.

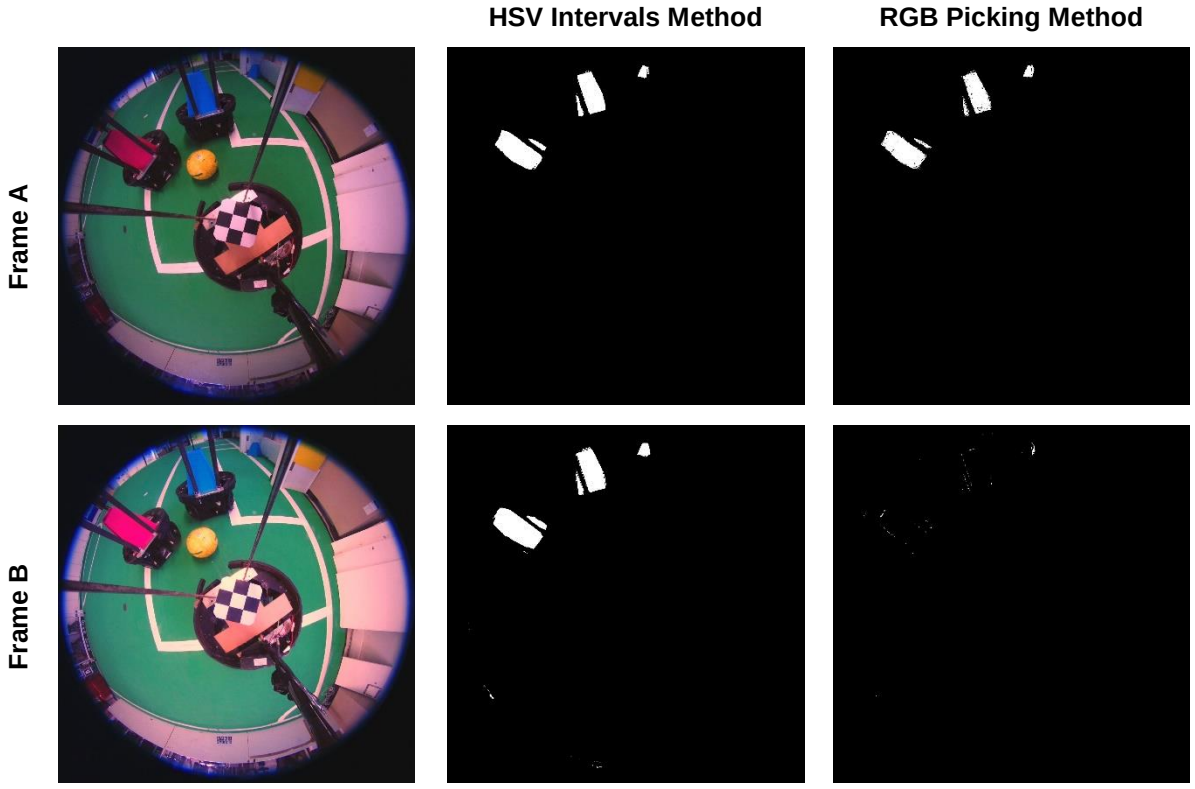

Figure 2.7: Experiment 1 with Gray-World color processing.

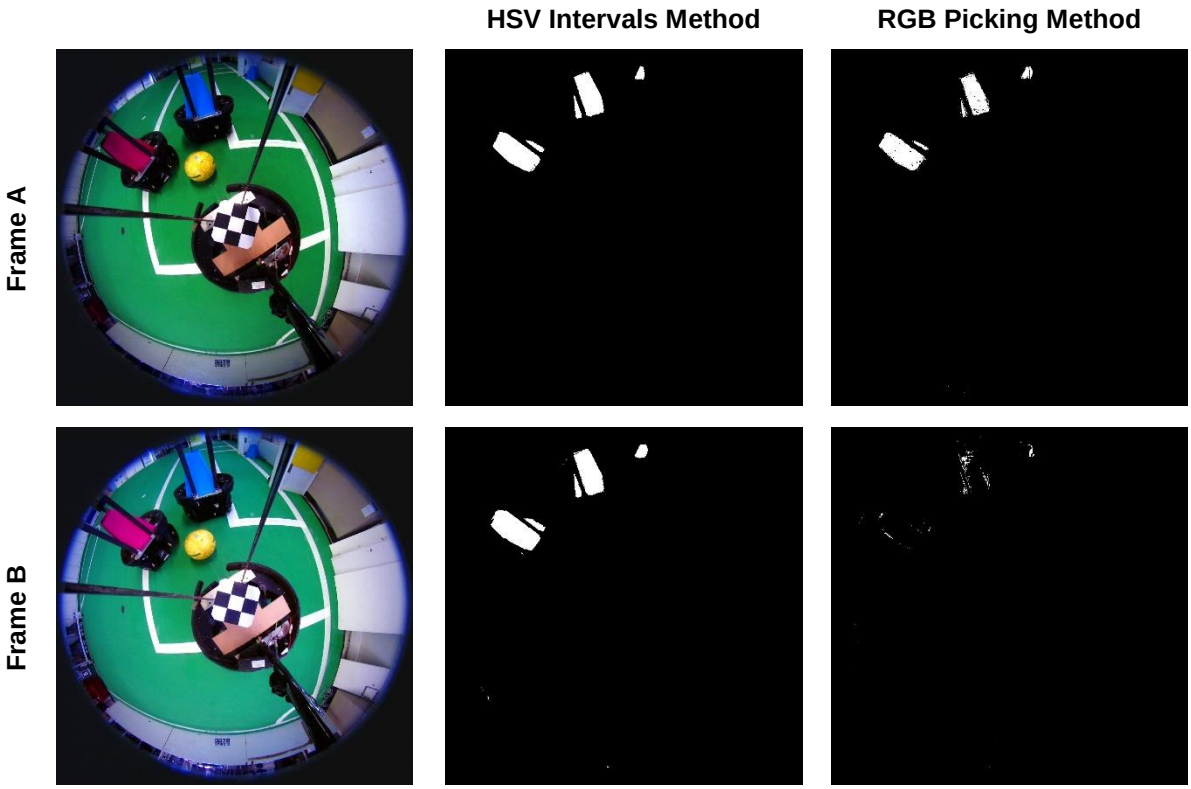

Figure 2.8: Experiment 1 with Max-RGB color processing.

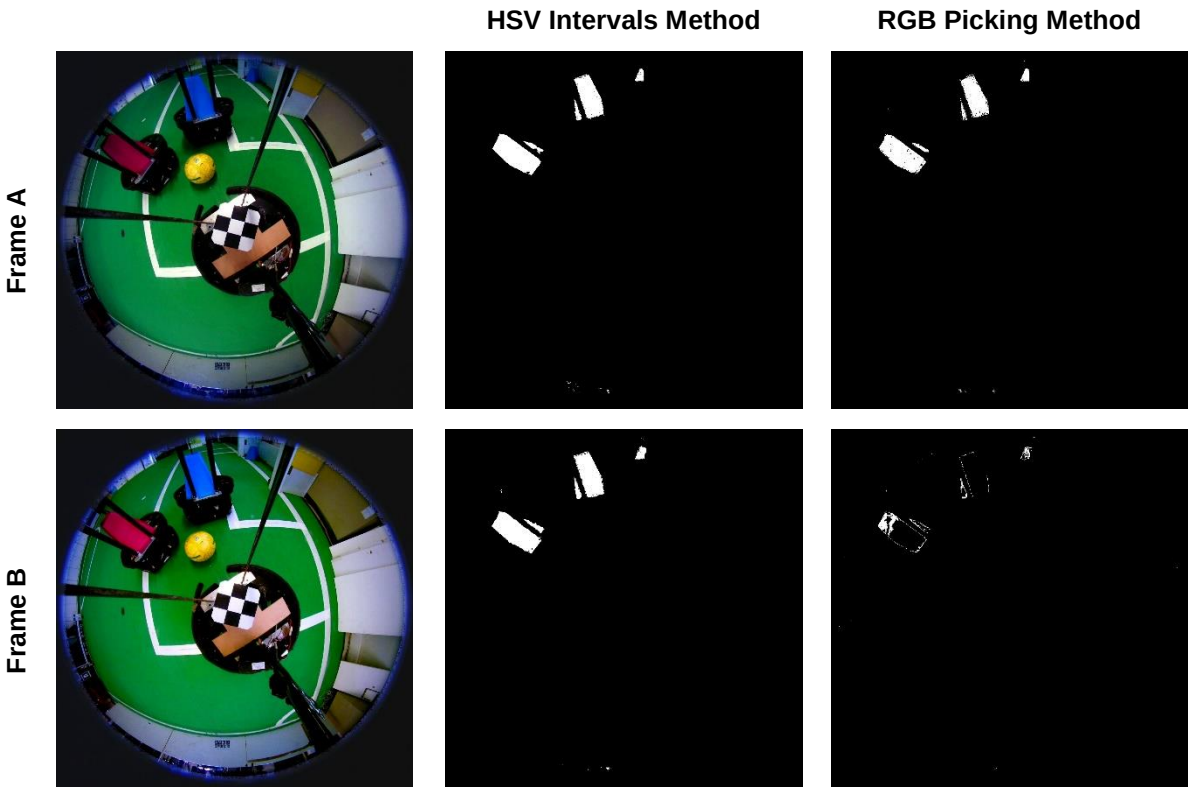

Figure 2.9: Experiment 1 with Max-RGB on patch color processing.

2.5.2 Experiment 2: Illumination geometry variation

No Pre-Processing			
LUT	Frame	Precision	Recall
HSV interval	A	0.915	0.835
	B	0.888	0.872
RGB picking	A	0.994	0.704
	B	0.993	0.492
Gray-World Assumption			
LUT	Frame	Precision	Recall
HSV interval	A	0.912	0.884
	B	0.906	0.901
RGB picking	A	0.996	0.646
	B	0.998	0.521
Max-RGB Algorithm			
LUT	Frame	Precision	Recall
HSV interval	A	0.919	0.856
	B	0.911	0.862
RGB picking	A	0.995	0.646
	B	0.995	0.530
Max-RGB on a Patch			
LUT	Frame	Precision	Recall
HSV interval	A	0.895	0.842
	B	0.856	0.812
RGB picking	A	0.992	0.633
	B	0.992	0.499

Table 2.3: Experiment 2 results.

In this case the changes in illumination are very subtle, in fact by looking at figure 2.10 it is hard to spot the difference between frame **A** and **B**. The illumination is a mixture of natural and artificial (fluorescent), with the artificial lighting kept constant in both frames. The frames **A** and **B** were captured with a time interval of about 1 hour this was enough time so that the angle of incidence of the light changed in a noticeably way, but not the color and intensity.

Again the recall values with the HSV intervals method stay high and suffer little change in all cases. In the case of the RGB picking method the quality of the segmentation did decrease substantially. The fact that even such a subtle change in illumination can deteriorate the performance of the segmentation produced in the RGB picking method, really comes to show how much this method is flawed. For all the color constancy methods the variation of the recall value is lower than in the case were no color processing is applied, however this difference is too small to draw any conclusions.

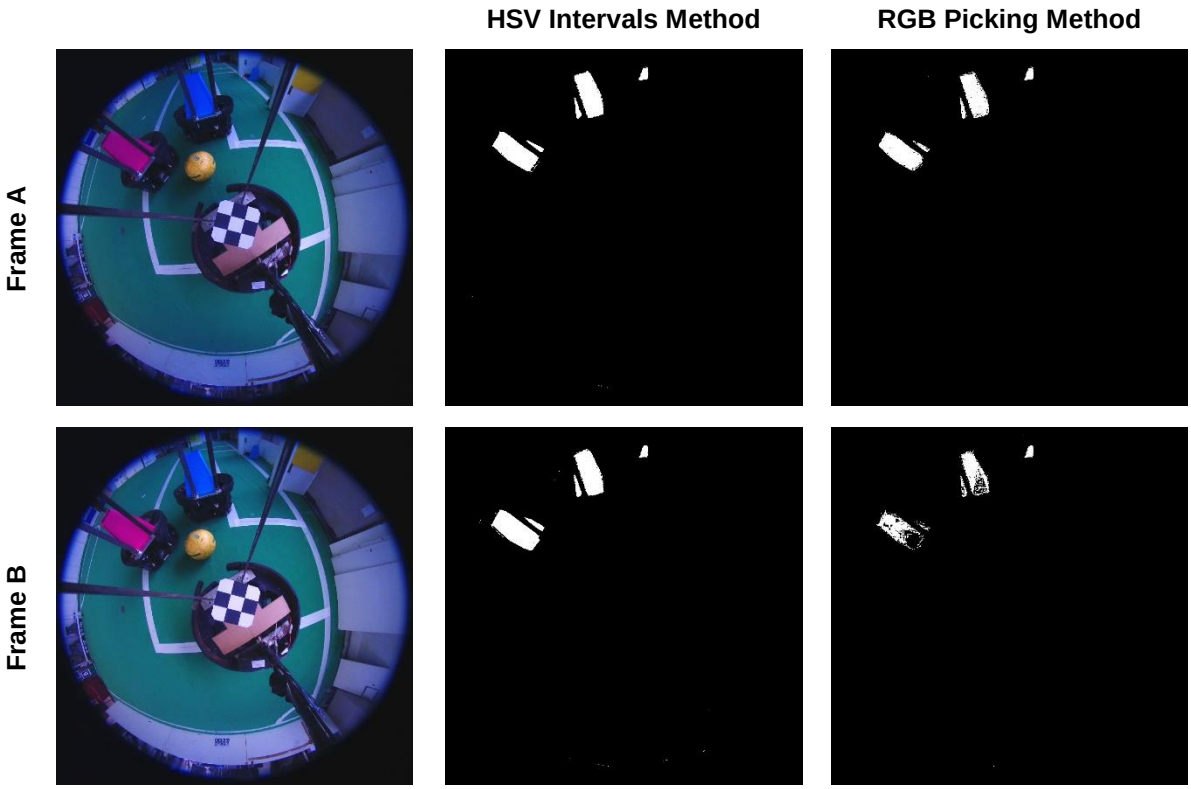

Figure 2.10: Experiment 2 with no color processing.

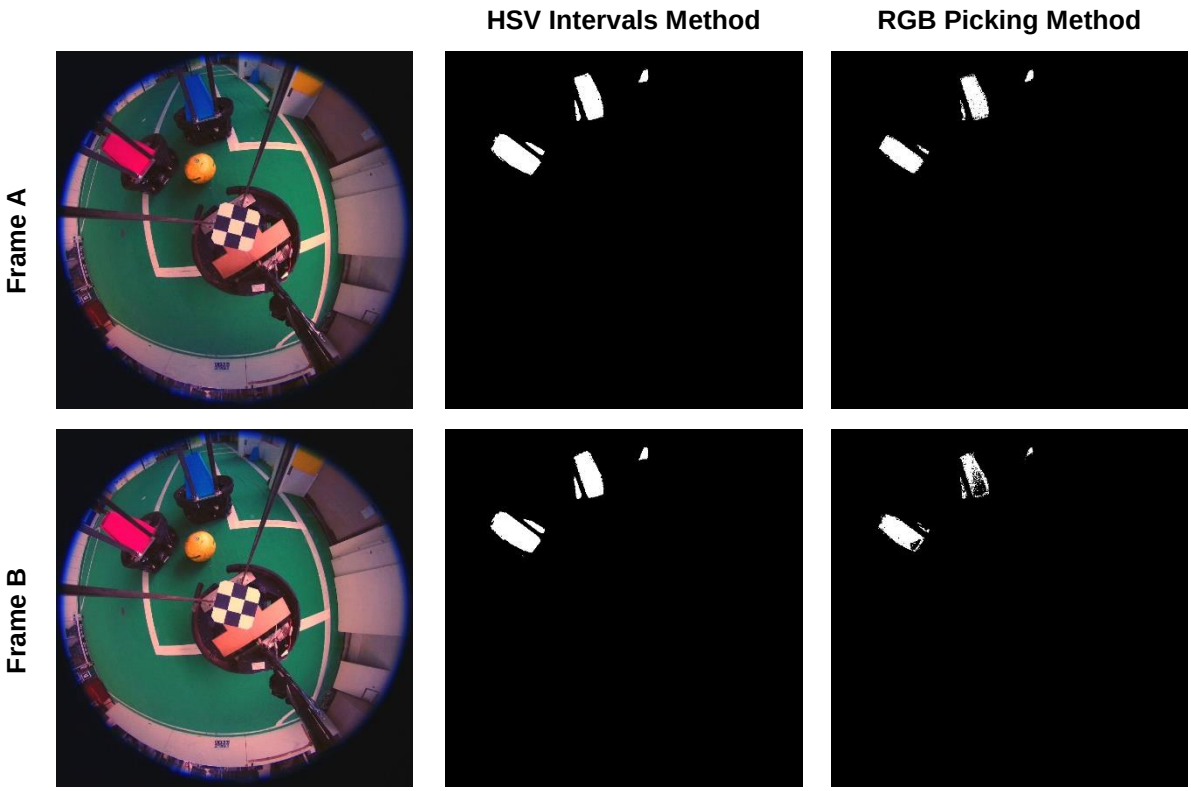

Figure 2.11: Experiment 2 with Gray-World color processing.

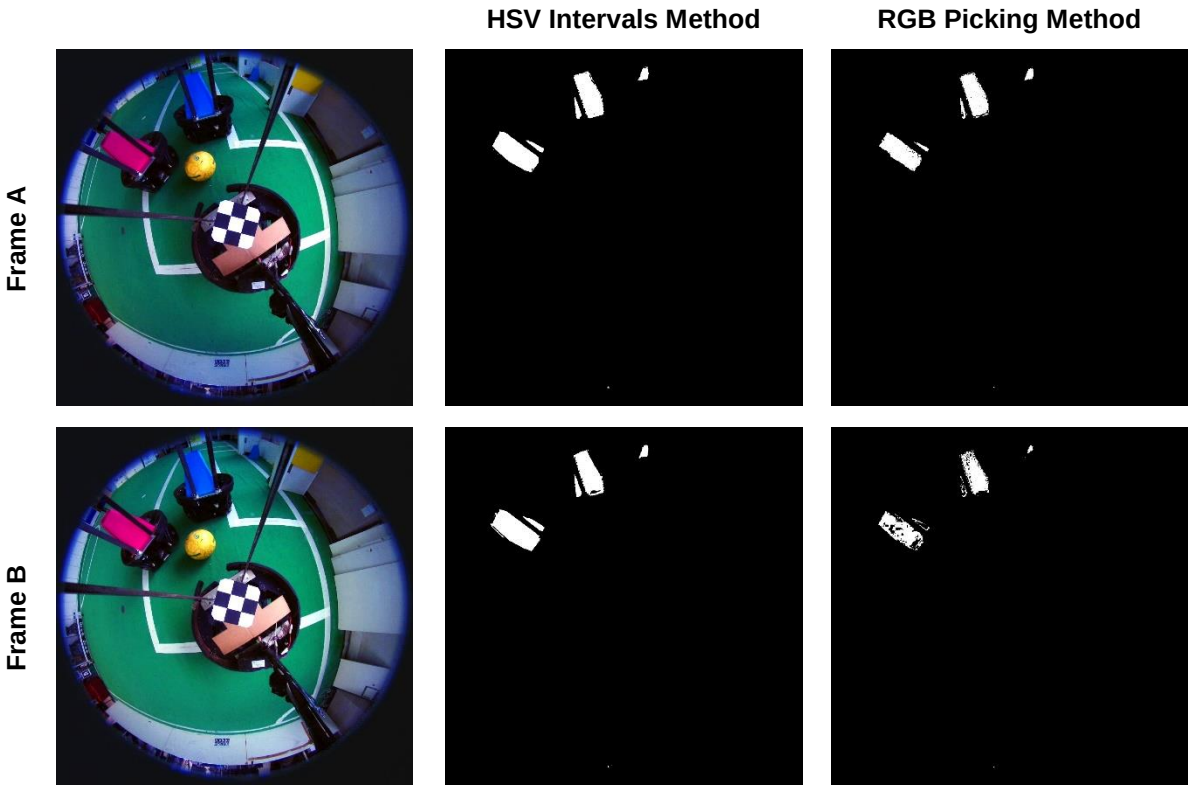

Figure 2.12: Experiment 2 with Max-RGB color processing.

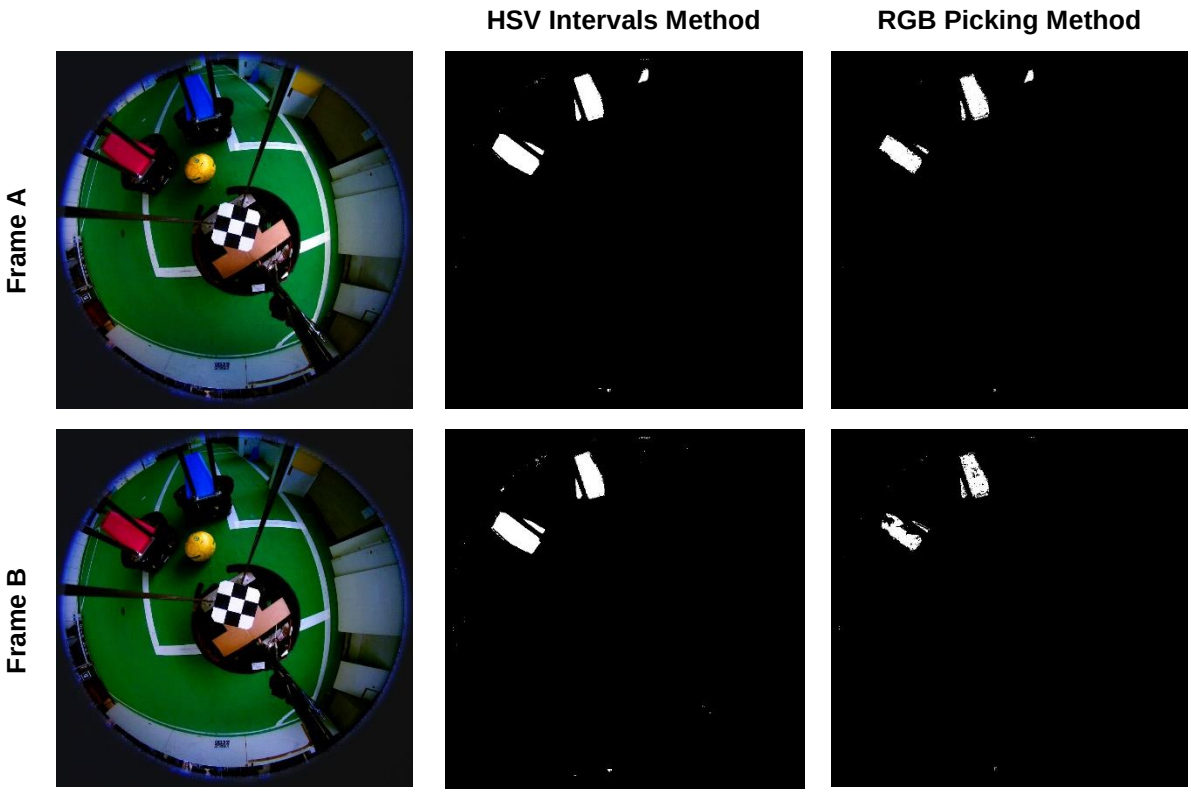

Figure 2.13: Experiment 2 with Max-RGB on patch color processing.

2.5.3 Experiment 3: Illumination color variation

No Pre-Processing			
LUT	Frame	Precision	Recall
HSV interval	A	0.868	0.930
	B	0.746	0.864
RGB picking	A	0.996	0.747
	B	0.323	0.002
Gray-World Assumption			
LUT	Frame	Precision	Recall
HSV interval	A	0.851	0.905
	B	0.837	0.901
RGB picking	A	0.990	0.712
	B	0.981	0.368
Max-RGB Algorithm			
LUT	Frame	Precision	Recall
HSV interval	A	0.912	0.925
	B	0.917	0.911
RGB picking	A	0.996	0.743
	B	0.993	0.331
Max-RGB on a Patch			
LUT	Frame	Precision	Recall
HSV interval	A	0.879	0.928
	B	0.882	0.921
RGB picking	A	0.992	0.789
	B	0.979	0.447

Table 2.4: Experiment 3 results

In this experiment two types of artificial lighting were used, in frame **A** only the fluorescent type was used and in frame **B** lower color temperature tungsten halogen lamps were added. It is easily observed in figure 2.14 that frame **A** has an overall bluish appearance, whereas frame **B** has a distinctive yellowish look. This kind of scenario may occur for example in the case where there are windows at one end of the field. In this case the illumination on that side of the field may be different than on the other side since it is “contaminated” by the sunlight entering the windows.

As in the previous cases, the recall values of the HSV intervals segmentation remain very consistent, especially in the cases where color constancy algorithms were used. In the case of the RGB picking the colors constancy methods proved to be effective, the Max-RGB on a patch performed particularly well, resulting in a usable segmentation, although not nearly as good as the segmentation obtained from the HSV intervals. In fact, when using the HSV intervals method and considering that the effects of all color constancy algorithms, although noticeable are minor, the benefits of using the color constancy may not outweigh the additional computational cost.

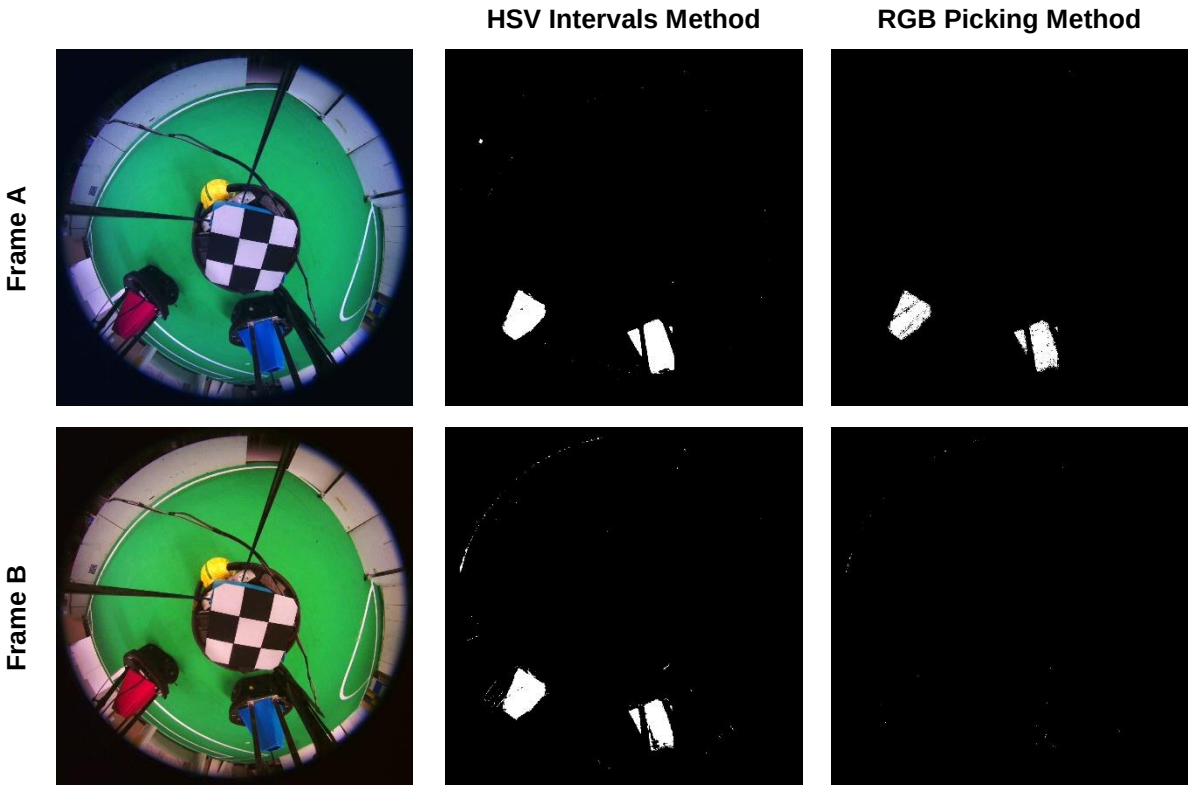

Figure 2.14: Experiment 3 with no color processing.

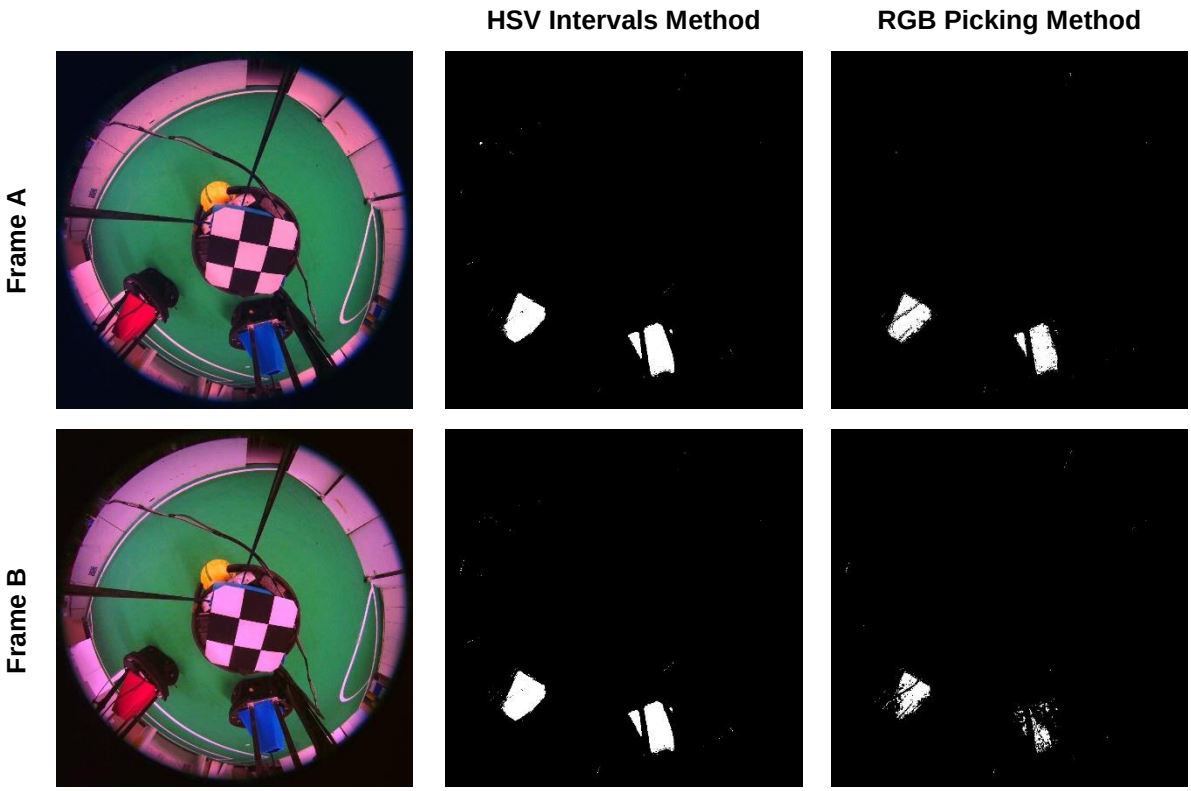

Figure 2.15: Experiment 3 with Gray-World color processing.

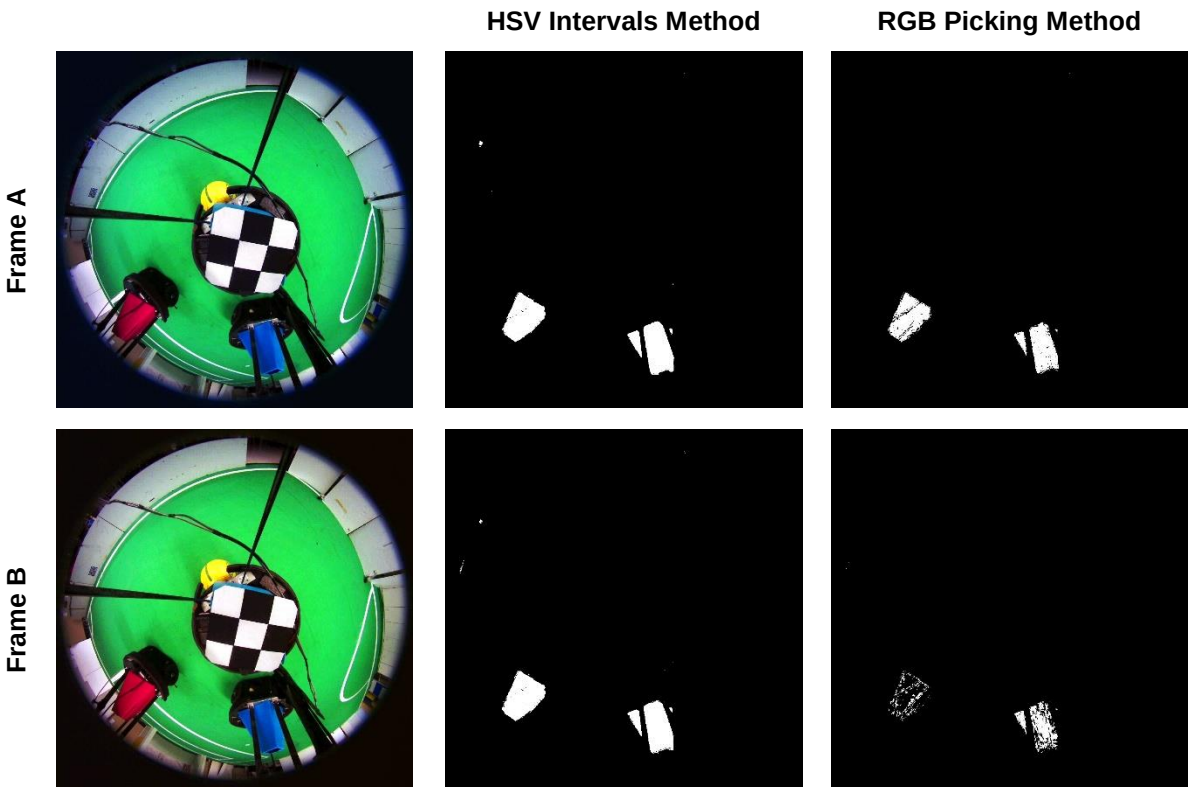

Figure 2.16: Experiment 3 with Max-RGB color processing.

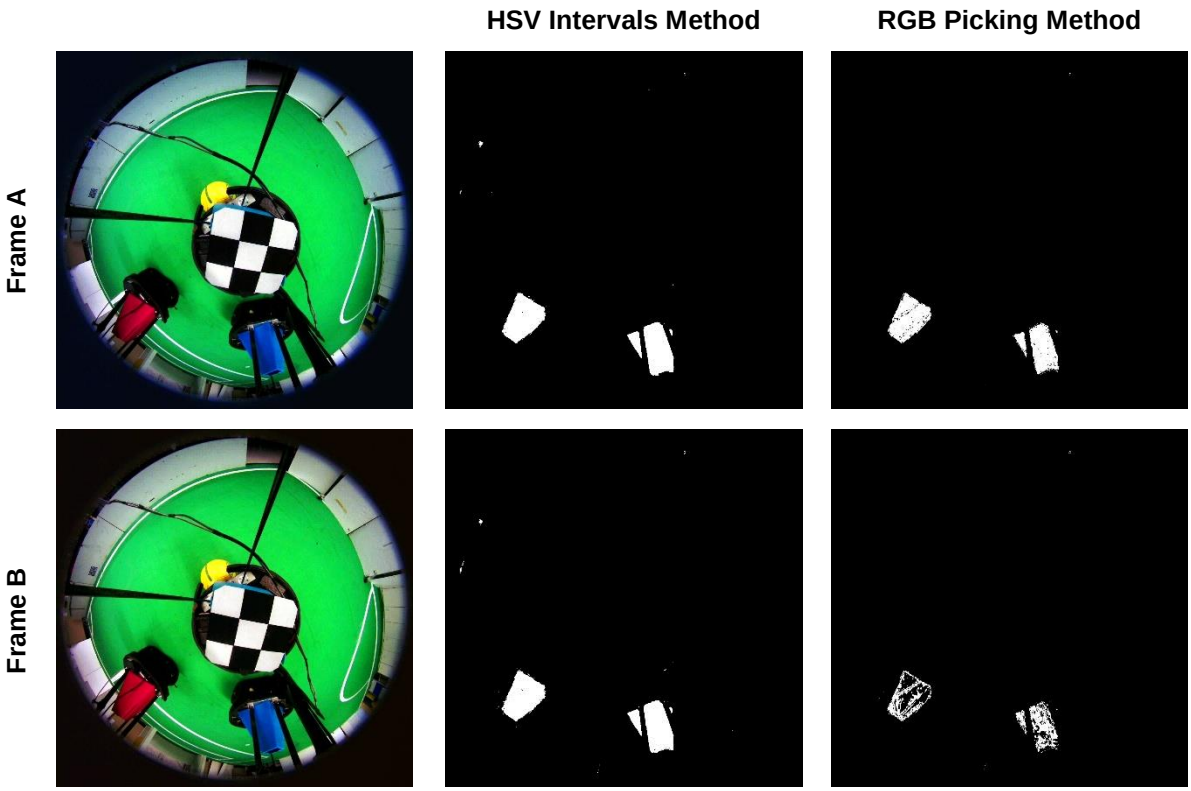

Figure 2.17: Experiment 3 with Max-RGB on patch color processing.

3 GROUND-TRUTH SYSTEM

A ground truth system is undoubtedly an invaluable tool in a mobile robotics laboratory. It is useful for evaluation, testing and data collecting purposes. In this particular case, it was helpful in obtaining an estimate of the process error covariance, which would otherwise be complicated to obtain.

Prior to this work there was already a ground truth system in use at the laboratory. This system however had a number of serious problems. The biggest problem related by its users was the very short useful range, which forced most experiences to be conducted in a small area, not taking advantage of the relatively large testing ground that the soccer field provided. This short usable range resulted mainly from difficulties in detecting the markers attached to the robots. This markers consisted of a surface with different colored shapes mounted on top of the robots. Color segmentation had to be done for each of the colors, and the shapes tended to become flattened and difficult to detect as the distance from the camera increased.

It was clear from the above reasons that the ground truth system needed to be improved, namely by solving the markers detection issues and consequently the detection range, ensuring that the system is useable across the entire field. It was also important to eliminate the need to use so many different colors, which becomes very cumbersome as the number of robots increases. The system should be made simple and quick to deploy, it should demand very little user intervention and should allow to be used with an arbitrary number of robots.

The rest of this chapter is organized as follows: first the hardware components of the system are presented, namely the cameras and the tracking hardware attached to each robot; the transformation that relates a pixel position in the images to a world point constrained to a plane parallel to the ground plane is derived, allowing the measurement of an obstacle's pose; next, an easy and fast procedure for the estimation the measurement error variance using the Kabsch algorithm is proposed; finally the processing of the captured images, namely the detection and tracking of the markers and ultimately the production of the pose estimates is discussed .

3.1 HARDWARE

The developed ground truth solution is comprised of hardware and software components. The hardware consists of fixed position cameras, and marker devices that are mounted on each robot. Two fixed mounted cameras are installed, one at each end of the testing field. These cameras are capable of capturing images at 1294x964 pixel resolution and at a frame rate of about 30 fps. Each robot is equipped with tracking markers that were design to provide easily

detectable features across the whole span of the testing field. The apparatus mounted on each robot consists of a surface covered in black non-reflective material with a blue color high intensity diffuse LED implanted at each end. Experiments with several types of LEDs that were at the time available showed that only diffuse blue color LEDs provided a robust detection across the whole field. The requirements concerning this equipment are that all the robots have the tracking apparatus mounted horizontally at the same height and oriented in accordance to the robot's reference frame.

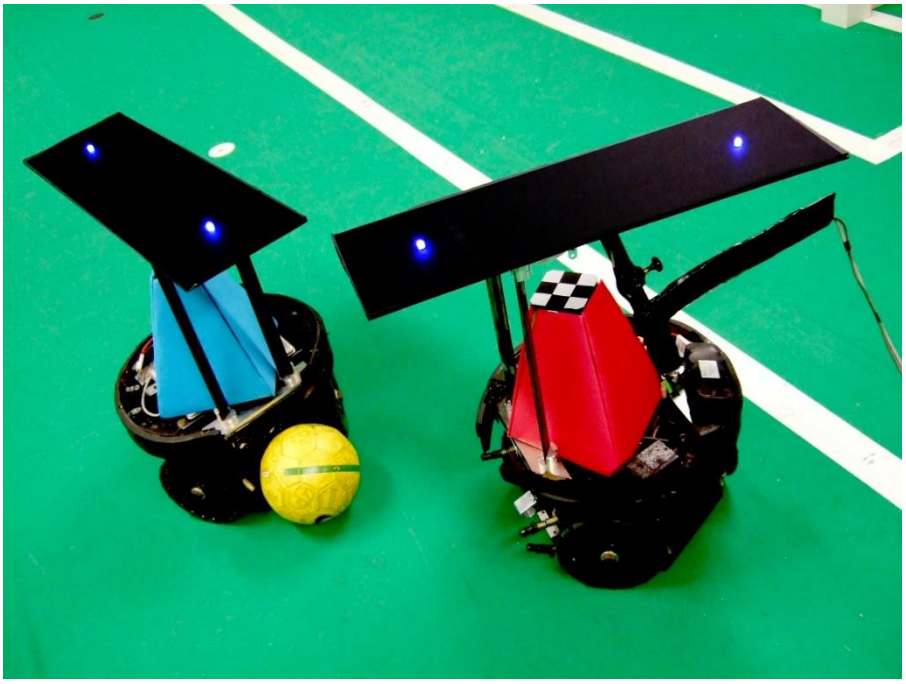

Figure 3.1: MSL robots with tracking marker apparatus.

3.2 CAMERA TO WORLD TRANSFORMATION

Because the motion of the robots is constrained to the ground plane, it's possible to take advantage of that fact to find a unique relationship between the coordinates of point in the world reference frame and its respective image coordinates as long as that point is static relative to the robots reference frame. In this particular case this transformation allows to immediately obtain the world coordinates of a tracking marker by detecting its pixel position, or more precisely the centroid of the pixel region where it is represented.

Consider the equation for a general pinhole camera projection, in which $(x, y, 1)^T$ is an image point in homogeneous coordinates and $(X, Y, Z, 1)^T$ is the corresponding scene point in homogenous world coordinates,

$$\alpha \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & \mathrm{c}_x & 0 \\ 0 & f & \mathrm{c}_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{3.1}$$

The coordinates of the world are transformed by the extrinsic parameters matrix, which describes the rotation and a translation from the world reference frame to the camera reference frame. The camera's geometric properties are represented in the intrinsic parameter matrix, which is a projective mapping from 3D camera coordinates to 2D homogeneous image coordinates.

As stated before the markers motion is restricted to a plane. Because the world reference frame can be chosen arbitrarily as convenient, we can set *Z*=0, allowing the elimination of a column in the extrinsic parameter matrix in equation (3.1). After some simplification we obtain,

$$\alpha \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & \mathrm{c}_x \\ 0 & f & \mathrm{c}_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ r_{31} & r_{32} & t_z \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} \tag{3.2}$$

For this particular application it's not necessary to explicitly estimate the intrinsic and extrinsic parameter matrix values, so their product may be represented by a homography $\boldsymbol{H}$,

$$\alpha \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} \tag{3.3}$$

This transformation is invertible, thus having an estimate of its value allows converting world to image coordinates, or more important for this context, image coordinates to world coordinates using its inverse. From the expression (3.3) we get,

$$x = \frac{h_{11}X + h_{12}Y + h_{13}}{h_{31}X + h_{32}Y + h_{33}} \tag{3.4}$$

$$y = \frac{h_{21}X + h_{22}Y + h_{23}}{h_{31}X + h_{32}Y + h_{33}} \tag{3.5}$$

By multiplying through by the denominator and rearranging, equations (3.4) and (3.5) become,

$$h_{11}X + h_{12}Y + h_{13} - h_{31}Xx - h_{32}Y\mathrm{x} - h_{33}\mathrm{x} = 0 \tag{3.6}$$

$$h_{21}X + h_{22}Y + h_{23} - h_{31}Xy - h_{32}Y\mathrm{y} - h_{33}\mathrm{y} = 0 \tag{3.7}$$

This equations can be expressed in matrix format for a set of corresponding image and world points,

$$\boldsymbol{A}\,\boldsymbol{h} = 0 \tag{3.8}$$

$$\begin{bmatrix} X_1 & Y_1 & 1 & 0 & 0 & 0 & -X_1x_1 & -Y_1x_1 & -x_1 \\ 0 & 0 & 0 & X_1 & Y_1 & 1 & -X_1y_1 & -Y_1y_1 & -y_1 \\ X_2 & Y_2 & 1 & 0 & 0 & 0 & -X_2x_2 & -Y_2x_2 & -x_2 \\ 0 & 0 & 0 & X_2 & Y_2 & 1 & -X_2y_2 & -Y_2y_2 & -y_2 \\ & & & & & \vdots & & & \\ X_n & Y_n & 1 & 0 & 0 & 0 & -X_nx_{\mathrm{n}} & -Y_nx_{\mathrm{n}} & -x_{\mathrm{n}} \\ 0 & 0 & 0 & X_n & Y_n & 1 & -X_ny_{\mathrm{n}} & -Y_ny_{\mathrm{n}} & -y_{\mathrm{n}} \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix} \tag{3.9}$$

For a set of $n \geq 4$ points, the least squares estimate of $\boldsymbol{h}$ can be found by using the singular value decomposition (SVD) transform to decompose $\boldsymbol{A}^T\boldsymbol{A}$,

$$\boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{U}^T \tag{3.10}$$

The least squares estimate of $\boldsymbol{h}$ subject to $\|\boldsymbol{h}\| = 1$ is given by the column of $\boldsymbol{U}$ associated with the smallest eigenvalue in $\boldsymbol{D}$.

Equation (3.1) models an ideal pinhole camera, however in a real camera with lenses there are distortion effects that are often too important to neglect. It is out of the scope of this work to provide a detailed presentation of lens distortion models and how to estimate their parameters. The distortion model used in this work is for convenience the one used in the OpenCV calibration routines. OpenCV offers calibration routines for obtaining the distortion model parameters, and also undistortion routines that correct an image according to the distortion model and its estimated parameters. This model considers two separate distortion components, radial and tangential distortion. The radial component is responsible for the barrel effect, which is more pronounced the wider the field of view of the lens. A pair of undistorted pixel coordinates (x', y') is calculated using the following expressions,

$$x' = x(1 + k_1r^2 + k_2r^4 + k_3r^6) \tag{3.11}$$

$$\mathrm{y}' = \mathrm{y}(1 + k_1r^2 + k_2r^4 + k_3r^6) \tag{3.12}$$

Where r is the distance of the pixel to the optical center (c_x, c_y), which is also obtained from the OpenCV calibration. Tangential distortion occurs when the image plane is not completely parallel to the lenses. The model for tangential distortion is represented in the following expressions,

$$x' = x + (2p_1xy + p_2(r^2 + 2x^2)) \tag{3.12}$$

$$y' = y + (2p_2xy + p_1(r^2 + 2y^2)) \tag{3.13}$$

Figure 3.2 shows the undistorted images used to estimate the homographies for the left and right cameras respectively. Because of the relatively large distance between cameras and the small size of the available checkerboard pattern available, it was not practical to capture a

simultaneous image of the pattern in both cameras. The checkerboard pattern was mounted on top of the robot at a height equal to the height of the features that will be detected. After the capture of the images for the left camera, the pattern was translated along its x direction with the help of guide wires. In this way the world points of the checkerboard points at each position can be expressed in the same reference frame. This means that the pixel coordinates of each camera are related by each respective homography transformation to the same world reference frame.

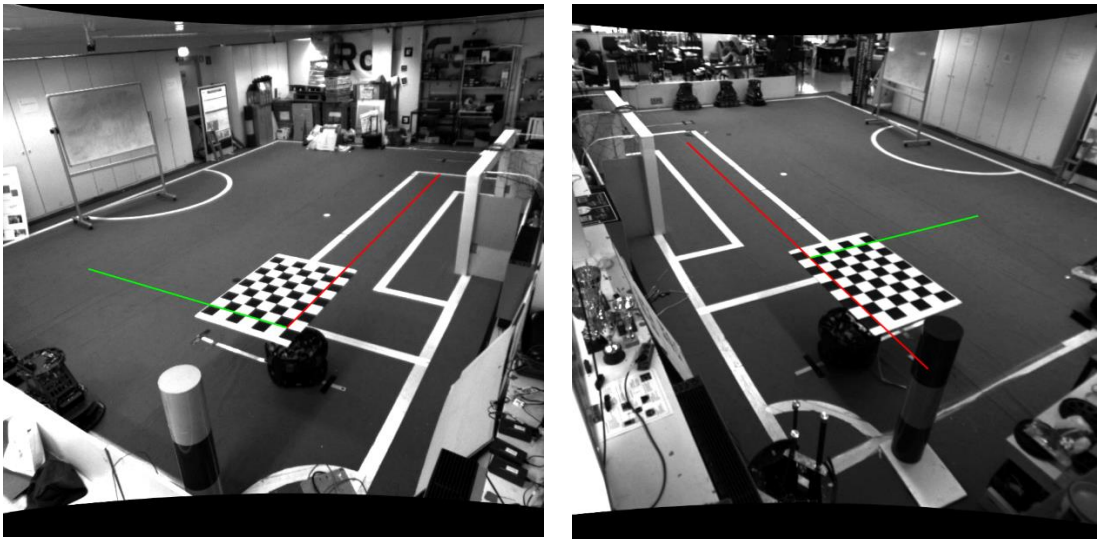

Figure 3.2: Calibration images.

3.3 MEASUREMENT ERROR VARIANCE ESTIMATION

Now that we have a transformation from image points to world points, the camera can be used to take measurements of the world position of markers detected in the image. The camera sensor is not continuous and of infinite precision, the image is made up of discrete pixels and therefore there is an error associated with the measurements. By simple geometrical reasoning and inspection (objects get smaller the further they are) it is easy to conclude that the scene area covered by each pixel increases with the distance to the camera, it is thus natural to assume that the further away the measured position the larger the error. The obvious approach to the estimation of the variance of this error involves the measuring of a good number points across the whole field. The problem with this approach is that precisely positioning those points on a relatively large field is a very time consuming and error prone operation. For applications where it is deemed acceptable to sacrifice some accuracy, the alternative method for estimating the measurement error variance proposed in this work provides a much simpler and faster procedure. A few assumptions are made: it is assumed that the error probability is Gaussian distributed with mean $\mu = 0$, this is the case if it is considered that all of the error is caused by the sensor discretization. Any fixed bias resulting of an imperfect calibration is neglected; it is assumed that the covariance matrix is diagonal with $\sigma_{11} = \sigma_{22} = \sigma$ and that the value of this covariance

parameter σ which for simplicity will be referred simply as variance, is a function of the distance from measured position to the camera.

The method is based on finding the alignment between a set of coordinates corresponding to corner points on a checkerboard of known dimension and a set of coordinates obtained from the image to world transformation of the image coordinates of that checkerboard corners. It is assumed that the error variance can be approximated by the mean squared deviation between the two aligned sets. Several sets of points are obtained by placing the checkerboard at various distances from the camera. Each one is aligned with the known checkerboard coordinates to produce a sample of the variance at a distance given by the average distance of the reconstructed points to the camera. In the next section the method used to align each set of reconstructed coordinates to the set of know corner coordinates is presented.

3.3.1 Kabsch Algorithm

The Kabsch algorithm is a method for solving a constrained orthogonal Procrustes problem that consists of finding the rotation matrix that minimizes the root mean squared deviation between two sets of corresponding points.

Let the $\boldsymbol{P}'$ be the chessboard corner points expressed in its own coordinate frame, and $\boldsymbol{Q}'$ the respective points with world coordinates calculated using the homographic transformation of the image points. The problem consists of finding the transformation which most closely maps one group one points to the other.

$$\boldsymbol{P}' = \begin{bmatrix} x_1 & y_1 \\ \vdots & \vdots \\ x_N & y_N \end{bmatrix} \quad , \quad \boldsymbol{Q}' = \begin{bmatrix} x_1' & y_1' \\ \vdots & \vdots \\ x_N' & y_N' \end{bmatrix} \tag{3.14}$$

Before using the Kabsch algorithm if is first necessary to subtract the centroid from both sets of point, thus taking care of the translation part of the transformation,

$$\boldsymbol{P} = \begin{bmatrix} x_1 - \frac{1}{N}\sum_{i=0}^{N} x_i & y_1 - \frac{1}{N}\sum_{i=0}^{N} y_i \\ \vdots & \vdots \\ x_N - \frac{1}{N}\sum_{i=0}^{N} x_i & y_N - \frac{1}{N}\sum_{i=0}^{N} y_i \end{bmatrix} \quad , \quad \boldsymbol{Q} = \begin{bmatrix} x_1' - \frac{1}{N}\sum_{i=0}^{N} x_i' & y_1' - \frac{1}{N}\sum_{i=0}^{N} y_i' \\ \vdots & \vdots \\ x_N' - \frac{1}{N}\sum_{i=0}^{N} x_i' & y_N' - \frac{1}{N}\sum_{i=0}^{N} y_i' \end{bmatrix} \tag{3.15}$$

The problem of finding the rotation that best aligns $\boldsymbol{P}$ with $\boldsymbol{Q}$ is expressed as follows,

$$\boldsymbol{R} = arg\,\min_{\boldsymbol{\Omega}} \|\boldsymbol{P\Omega} - \boldsymbol{Q}\|_F \ , \qquad subject\ to\ \boldsymbol{\Omega}^T\boldsymbol{\Omega} = \boldsymbol{I} \ \ and \ \ |\boldsymbol{\Omega}| = 1 \tag{3.16}$$

Where $\|.\|_F$ denotes Frobenius norm, and for a matrix $\boldsymbol{A}$ with real entries is defined as,

$$\|\boldsymbol{A}\|_F = \sqrt{trace(\boldsymbol{A}^T\boldsymbol{A})} \tag{3.17}$$

Consider the matrix $\boldsymbol{M}$ defined as follows,

$$\boldsymbol{M} = \boldsymbol{P}^T\boldsymbol{Q} \tag{3.18}$$

Decomposing $\boldsymbol{M}$ using the Singular Value Decomposition (SVD) transform results in,

$$\boldsymbol{M} = \boldsymbol{V}\boldsymbol{S}\boldsymbol{W}^T \tag{3.19}$$

To ensure that the determinant of the transformation is positive, and thus ensure a right handed coordinate system, the following calculation is needed,

$$d = sign(|\boldsymbol{W}\boldsymbol{V}^T|) \tag{3.20}$$

Finally, the rotation matrix that minimizes the root mean squared deviation between the two sets of points is given by,

$$\boldsymbol{R} = \boldsymbol{W}\begin{bmatrix}1 & 0\\ 0 & d\end{bmatrix}\boldsymbol{V}^T \tag{3.21}$$

3.3.2 Estimation procedure

Estimating the variance requires a significant amount of data in the form of world and pixel coordinate pairs. Placing and measuring precise positions in an area of relatively large dimensions as the playing field, makes this a tedious and time consuming task. In this proposed method this task is greatly simplified, because no precise measuring of world positions is necessary. A checker board pattern of known dimension was installed on top of the robot at the same height as the tracking marker apparatus. The robot was then moved across the field and a set of roughly even spaced samples was selected. It is not necessary to sample a precise location, but an effort should be made to get a good distribution of distances as this leads to a better estimation. In figure 3.3, it is possible to observe a few of the samples used to estimate the variance as a function of distance from the camera.

The image coordinates of corner points of the checkerboards were extracted, and transformed into world coordinates. Next the extracted points in each sample are aligned with the known checkerboard corners coordinates. This can be observed in figure 3.4 in which the red points are known position of the corner points and the blue points are the detected corner points. It is clear just by observing the figure that the deviation between the two sets of points increases as the distance from the camera increases, as it was expected. The mean squared error is calculated and used as estimator of the variance. Next, regression is performed on the data, using the average of the points distance from the camera in each sample as the independent variable and the estimated variance as the dependent variable. Figure 3.5 shows the estimate of the

variance as a function of the distance from the camera, which is the result of fitting a fourth degree polynomial to the samples.

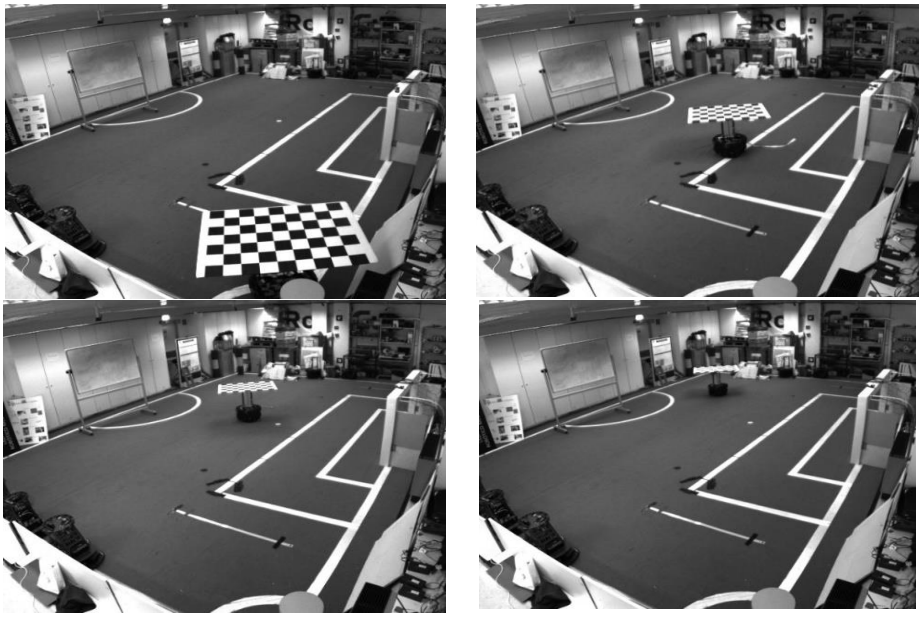

Figure 3.3: examples of checkerboard images used to estimate the error variance.

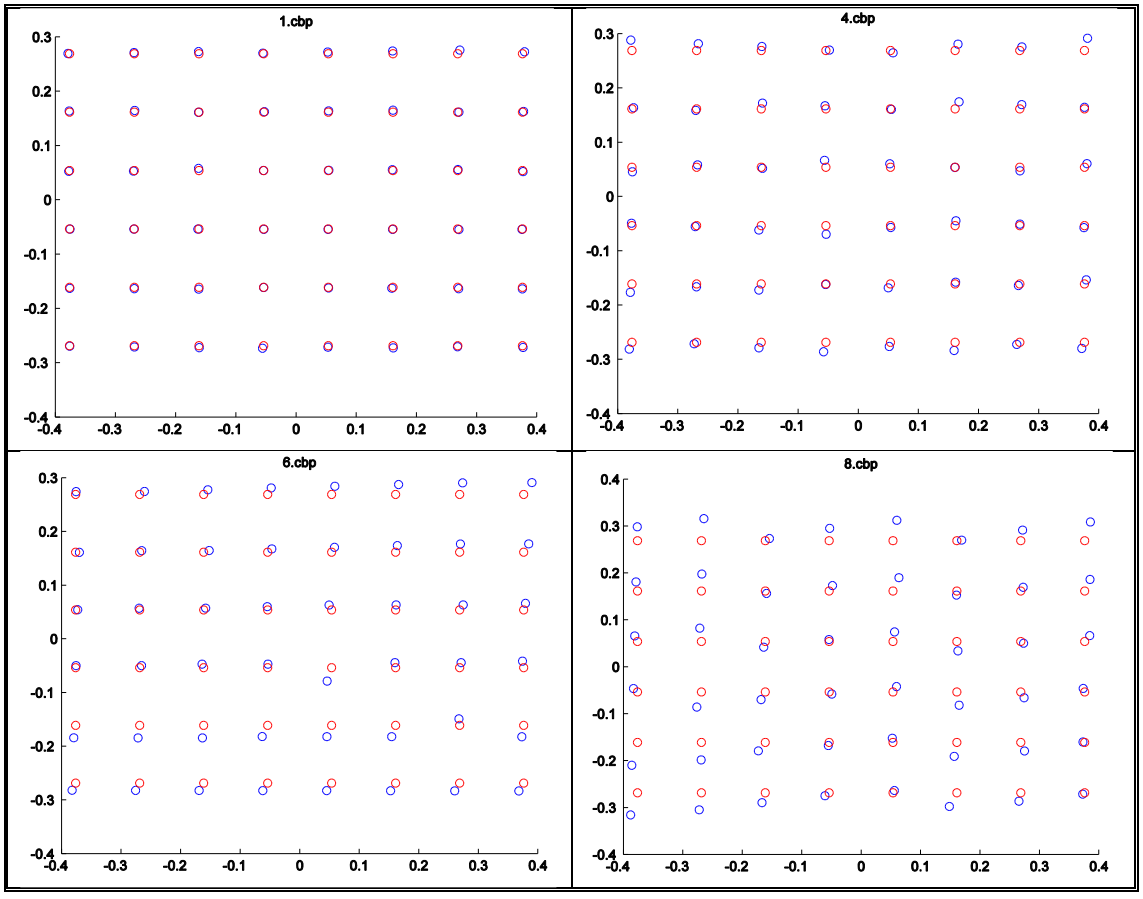

Figure 3.4: Detected corner points aligned with known positions.

The final estimate of the position $\boldsymbol{x} = (x, y)$ and error variance σ^2 are obtained by fusing the estimates from both cameras, by means of an inverse variance weighted average. Which corresponds to the maximum likelihood estimator. Let the position estimates from the left and right cameras be $\boldsymbol{x}_l$ and $\boldsymbol{x}_r$ and their respective variances be σ_l^2 and σ_r^2 then, the final variance and estimate are given in expressions (3.22) and (3.23). Figure 3.6 shows the value of the final variance as a function of the position on the field.

$$\sigma^2 = \frac{1}{\sigma_l^{-2} + \sigma_r^{-2}} \tag{3.22}$$

$$\boldsymbol{x} = \sigma^2 \left(\frac{\boldsymbol{x}_l}{\sigma_l^2} + \frac{\boldsymbol{x}_r}{\sigma_r^2} \right) \tag{3.23}$$

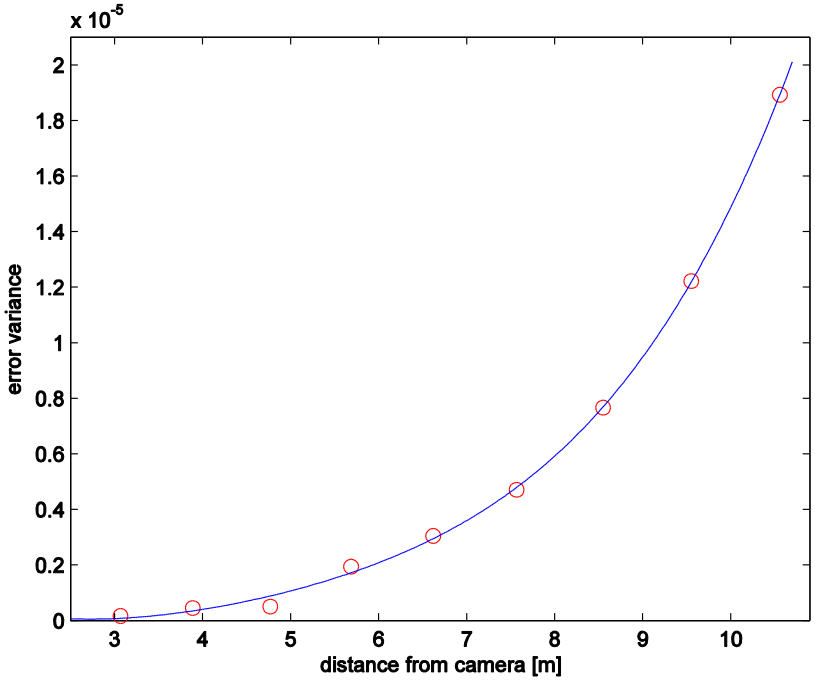

Figure 3.5: Variance as a function of the distance from the camera.

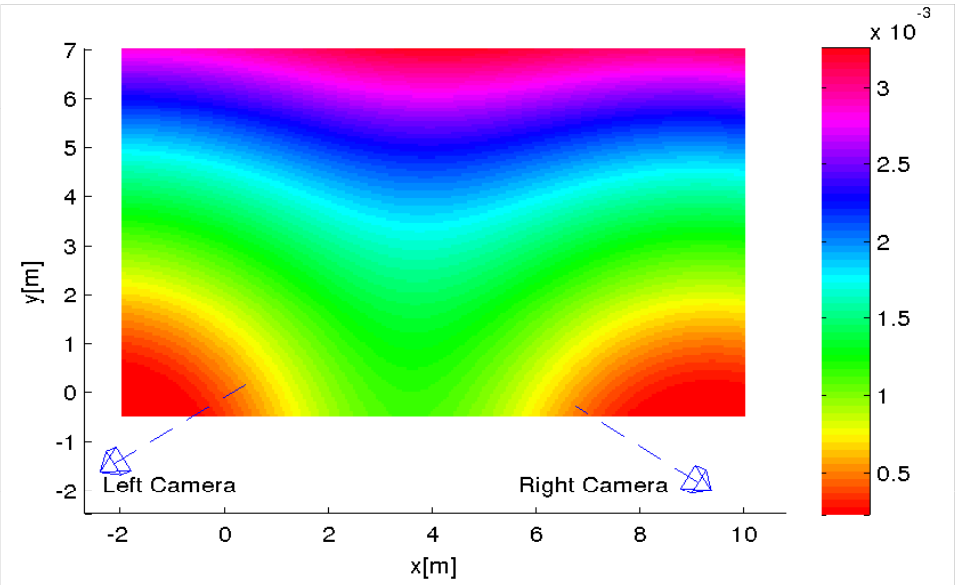

Figure 3.6: Variance as function of the position of the measurement.

3.4 SOFTWARE

This section is dedicated to the processing of the collected image sequences. Some of the methods used in the detection and tracking of markers are similar to the ones that each robot uses for obstacle detection and tracking. The purpose of this section is to provide an overview of the sequence of operations as illustrated in figure 3.7. Nevertheless more detailed explanations of these methods are covered further ahead in this work.

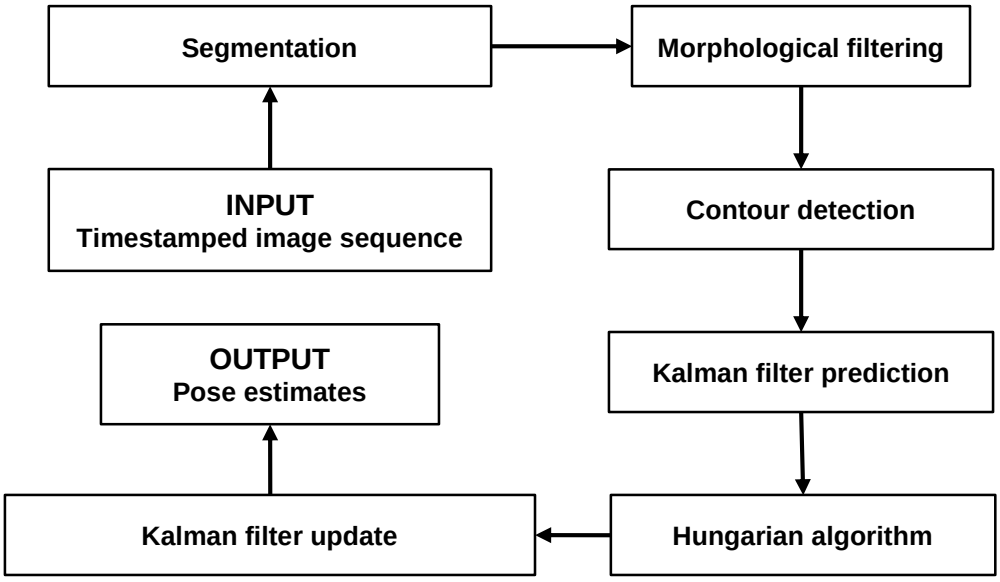

***Figure 3.7:* Sequence of processing operations.**

The first thing being addressed is the detection of the markers. As previously stated, each robot has a blue LED marker at both the front and back. By detecting both markers, one finds the full pose of the robot, its position and orientation. The segmentation system that has been presented in Chapter 2 is used to generate a binary image. A mask is used to eliminate the parts of the image that correspond to robot positions outside the playing field. This masking is particularly important to avoid false positives that can occur near windows. Next the result is subjected to morphological filtering, more precisely an opening operation, thus eliminating small clusters of pixels that represent false positive detections that may exist. A contour finding algorithm [9] is then used to identify each of the blobs that represents a marker and extract the respective centroids. All markers are identical and there is no *a priori* information about the order they're going to be labeled, which is of course dependent on unknown initial conditions. Therefore the user must provide the correspondences between the labels attributed to the detections and the markers for the initial conditions. After the program processes the detection on the first image of the sequence the user is prompted to identify which label corresponds to the back and front of each robot. After this initial identification of marker pairs no further user interaction is required.

Each marker is tracked resorting to a method which uses one Kalman filter per marker and the Hungarian method for associating detections between frames. Both electrical noise in the sensor and illumination noise in the environment cause the measurement to be noisy. Even if the robot is completely motionless, some oscillation is present in each marker's detected position. The unknown noise variances needed for the Kalman filtering, are tuned through trial and error so that the filter produces a smooth output but at the same time ensuring it doesn't respond sluggishly to the robot accelerations. The use of the Kalman filter allows dealing with situations where the detection of a marker is temporarily lost. On which case the position of the marker continues being predicted until the detection is restored. For each iteration the Kalman filter produces a prediction of the state of each tracked marker based on the previous state estimate. The correspondences between the predicted markers and the ones detected are found resorting to the Hungarian method. The algorithm receives as input the distances between each predicted marker position and the position of all detected markers and outputs the correspondences that minimize the sum of the distances between each predicted marker position and the corresponding detected marker position. Finally the Kalman filter update step produces the markers state estimate.

During the processing of the images the program outputs a graphical display of the estimates, allowing the user to verify that no mishaps occur. The user can observe both the detect blobs and their labels on the segmented image or the original image for both cameras. Overlaid arrows display the position and orientation of each tracked robot as can be seen in figure 3.8.

The program also displays the pose estimates in the world reference frame as can be seen in figure 3.9. The estimates obtained for each camera are shown as well as the final estimate. Note that the proximal and distal designation are just alternative nomenclature for the left and right fixed cameras respectively. That nomenclature was meant to be less confusing and

ambiguous than the left and right designation since the proximal and distal are relative to the user workstation.

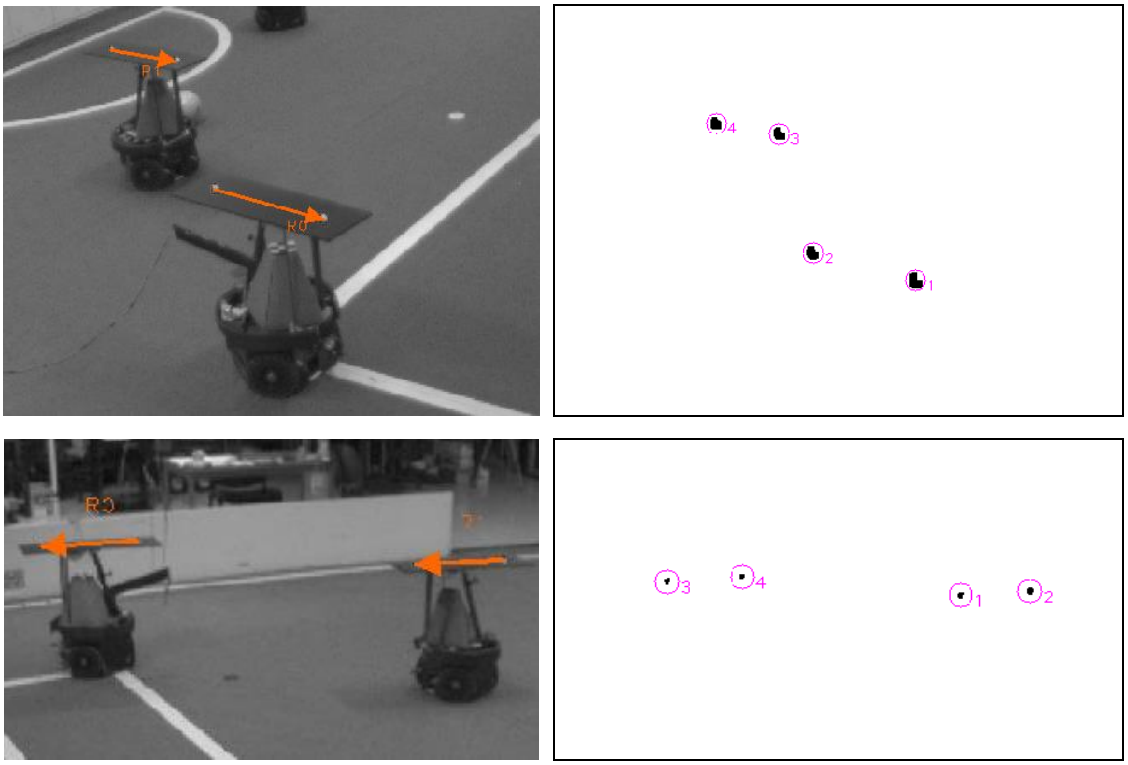

Figure 3.8: Marker detection example.

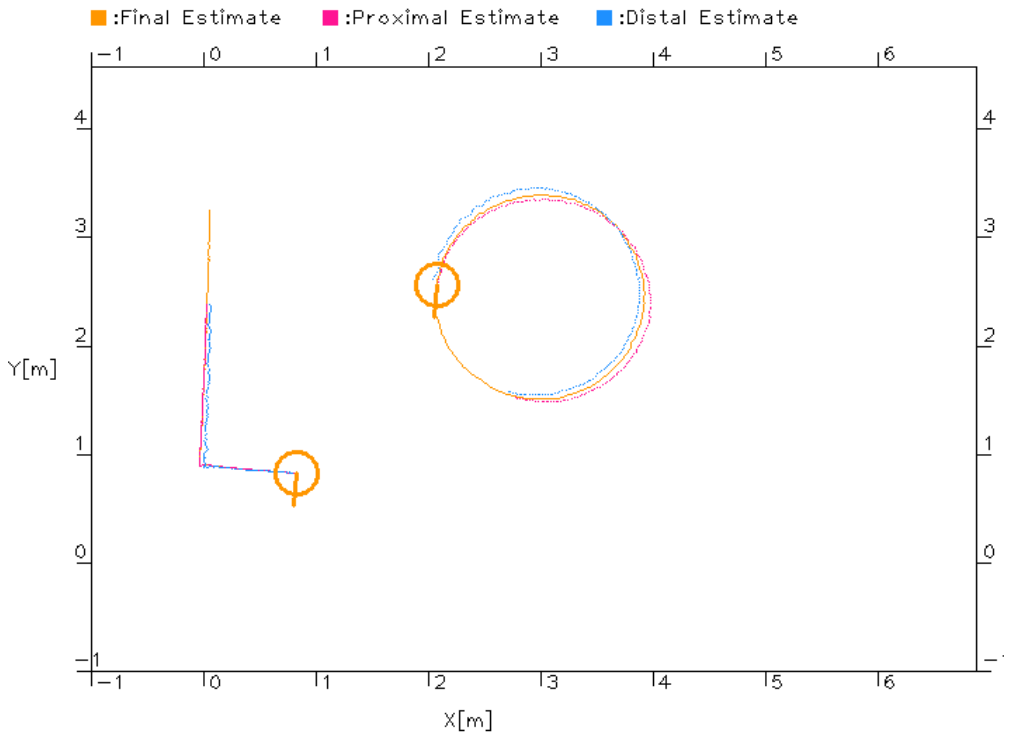

Figure 3.9: Example output window of the ground-truth system.

4 OBSTACLE DETECTION

This chapter addresses the problem of detecting obstacles, namely other robots, using only the image captured by the robots camera. In a multi agent environment such as MSL soccer the choice of sensors is somewhat limited. Active sensors, i.e. sensors that emit energy into the environment, such as sonar, infra-red and laser range finders are widely used in most mobile robotics application, for the purpose of detecting obstacles. Sensors of this type however, are prone to crosstalk when used simultaneously by different robots, thus rendering their measurements very unreliable. The detection of obstacles is completely dependent on being able to extract obstacle information from the camera's image. Because being able to reliably detect obstacles is absolutely fundamental for the overall playing performance of the robots, it is of the utmost importance to have an effective and efficient vision based obstacle detection.

The rest of this chapter is organized as follows: the robot's omnidirectional camera is going to be characterized by explaining how it's modeled and how a pixel in the image is corresponded to a world point in the ground plane; next, it is detailed the procedure that was used to estimate the error variance associated with the camera measurements; after that, the obstacle detection as such is addressed, the image processing methods are presented, as well as the heuristic filtering strategies used to extract useful obstacle information from the segmented image.

4.1 CAMERA MODEL

The robots are equipped with an omnidirectional dioptric vision system composed of a gigabit Ethernet camera with a fisheye lens, providing a field of view slightly larger than 180º. The camera is mounted facing downward on an elevated platform supported by thin struts, so that the view is minimally obstructed in all directions.

When omnidirectional vision systems first started to be used in robotic applications they were most often specially made catadioptric systems, composed of a curved mirror and a perspective camera. This curved mirror had a well-known geometry, and it was usually fabricated to match a desired model. In recent times omnidirectional dioptric systems are more available and are a less costly solution. However, many of these omnidirectional camera systems are marketed for consumer applications, and most of the times no technical data concerning lens geometry is made available by the manufacturer. In the particular case of the cameras used in this work this was no exception.

If it's only taken into consideration the task of detecting obstacles in the ground plane, the problem is equivalent to finding a mapping between pixels coordinates and ground plane

coordinates. Furthermore assuming that the lens is symmetric around its optical axis, this problem is reduced to finding a mapping between a distance in pixels and a distance in the ground plane. This could be solve by collecting data, and performing regression to estimate the mapping function.

Although the primary concern in this work is the detection of obstacles in the ground plane, one should consider the global utility of the omnidirectional camera and be aware of its uses beyond the scope of this work. The choice of a camera model must reflect and account for these necessities. Consider the example of the detection of the ball. Many times the ball travels through the air, as most robots are able to kick it off the ground. To be able to estimate the position of the ball a model more sophisticated than the one suggested in the previous paragraph is needed. An omnidirectional camera model that allows the association of a pixel position with a line in world space in which lie all points that are projected onto that pixel, much like in a normal perspective camera, can be used both for estimating the 3D position of a symmetrical object of known dimensions and the position of a point that belongs to a known plane.

The OCamCalib is an industry proven and widely recognized omnidirectional camera calibration tool-box. It is able to calibrate most types of omnidirectional vision systems, both catadioptric and dioptric, as long as the camera, at least approximately, meets the requirement of having a single effective viewpoint. OCamCalib provides methods to calibrate a camera from a series of captured checkerboard images. It also provides methods to project a scene point into the image and conversely reconstruct the unit vector that emanates from the view point given an image position.

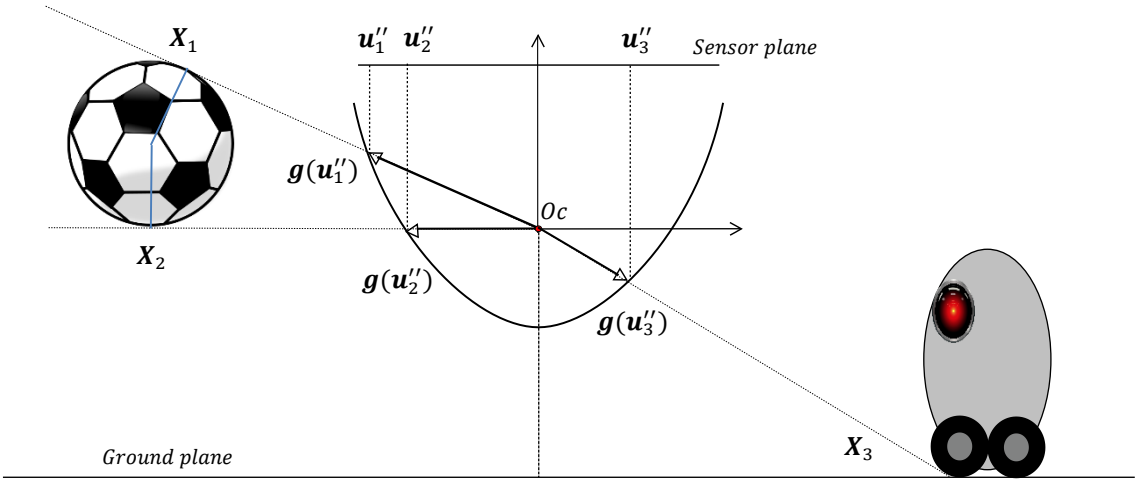

Figure 4.1: Projection of scene points on the sensor plane.

The derivation of the calibration method is beyond the scope of this work and it's described in detail in [3][4][5]. Nonetheless, in order to take full advantage of the OCamCalib calibration it is important for the user to understand how the camera model works.

In figure 4.1 it is possible to observe the projection of scene points into the sensor plane. The sensor plane is a hypothetical plane orthogonal to the lens axis. A point $\boldsymbol{u}''$ in the sensor plane, and expressed in metric coordinates, is related to a point $\boldsymbol{u}'$ in the camera image plane, expressed in pixel coordinates, by means of an affine transformation as expressed in equation (4.1). This transformation, models the discretization process and axis misalignments.

$$\boldsymbol{u}'' = \boldsymbol{A}\boldsymbol{u}' + \boldsymbol{t} \tag{4.1}$$

It be can observed in figure 4.1 that scene points are projected in the sensor plane, by an orthogonal projection of the intersection of the line connecting the optical center Oc to the scene point with a surface defined by the function $\boldsymbol{g}$. The expression (4.2) captures this relation between a scene point $\boldsymbol{X}$ (in homogeneous coordinates) and a point in the sensor plane $\boldsymbol{u}''$. The matrix $\boldsymbol{P} = [\boldsymbol{R} \mid \boldsymbol{t}]$ includes the rotation and translation that relates the scene reference frame with the sensor plane reference frame.

$$\lambda\, \boldsymbol{g}(\boldsymbol{u}'') = \lambda\, \boldsymbol{g}(\boldsymbol{A}\boldsymbol{u}' + \boldsymbol{t}) = \boldsymbol{P}\,\boldsymbol{X}, \qquad \lambda > 0 \tag{4.2}$$

Considering the rotational symmetry with respect to the sensor axis, let the function $\boldsymbol{g}$ be defined as follows,

$$\boldsymbol{g}(u'', v'') = (u'', v'', f(u'', v''))^T \tag{4.3}$$

The function $f(u'', v'')$ is a polynomial of degree N as defined in the following expression, where $\rho = \sqrt{u''^2 + v''^2}$ is the metric distance from the sensor axis.

$$f(u'', v'') = a_0 + a_1\rho + a_2\rho^2 + \ldots + a_N\rho^N \tag{4.4}$$

The relation between a scene point and an image point can be rewritten as,

$$\lambda \begin{bmatrix} u'' \\ v'' \\ w'' \end{bmatrix} = \lambda\, \boldsymbol{g}(A\boldsymbol{u}' + t) = \lambda \begin{bmatrix} A\boldsymbol{u}' + t \\ f(u'', v'') \end{bmatrix} = \boldsymbol{P}\,\boldsymbol{X}, \qquad \lambda > 0 \tag{4.5}$$

The calibration procedure produces the values for the polynomial coefficients a_0 to a_N, and the elements of the affine transform $\boldsymbol{A}$ and $\boldsymbol{t}$.

In this particular case the camera is fixed in relation to the robot's body, and the robot motion is constrained to the ground plane. The robot's reference frame *X* and *Y* axis are made coplanar

with the ground plane and the origin is placed at robot's geometrical center ground coordinates. The gathered data consists of points in the ground plane with coordinates expressed in the robot reference frame and respective pixel coordinates, the extrinsic parameters estimation routines provided by OCamCalib were used to obtain the transformation that relates the robot reference frame to the camera reference frame. Having the complete calibration, we can use the OCamCalib's included methods to find the unit vector emanating from the view point that is associated with a given image pixel. Because the transformation between the camera and the robot reference frame is known, the line defined by the found unit vector and the view point can be intersected with the ground plane to obtain the ground coordinates in respect to the robot's reference frame that correspond to the image pixel coordinates. It is also possible to obtain the 3d coordinates of the ball. By obtaining the lines that are tangent to the ball, as illustrated in figure 4.1, and knowing its diameter, the distance of the ball's center to the view point can be obtained by simple trigonometry.

4.2 ESTIMATION OF THE MEASUREMENT ERROR VARIANCE

The measurements of ground plane positions from pixel positions, obtained as described in the previous section, are obviously affected by noise. One significant source of noise has to do with the fact that the image represents a discretization of a continuous space. It is easy to observe that the furthest a pixel is from the image center, the largest the area of ground plane that is projected onto that pixel. It is therefore natural to assume that the variance of the measurement error ought to increase with the distance from the image center. In order to obtain an estimate of this error variance some assumptions are made: attending to the rotational symmetry of the lens around the optical axis and that the image plane is parallel to the ground plane, it is assumed that the variance depends only on the distance of the pixel to the optical center's projection coordinates on the image (which were estimated during the calibration); it is assumed that the error probability is Gaussian distributed with mean $\mu = 0$. The procedures starts with the collection of images with a checkerboard pattern placed on the ground plane at precisely measured distances from the optical center projection on the ground plane. Figure 4.2 shows a few examples of the used images. From each of this images the estimated coordinates of the pattern corner points are compared with the known positions, as illustrated in figure 4.3, and the mean squared deviation is calculated. This value is used as estimator of the variance. The samples consist of the average of the points distance as independent variable and the estimated variance as the dependent variable. Figure 4.4 shows the result of fitting a fourth degree polynomial to the samples, thus obtaining an estimate of the variance as a function of the distance.

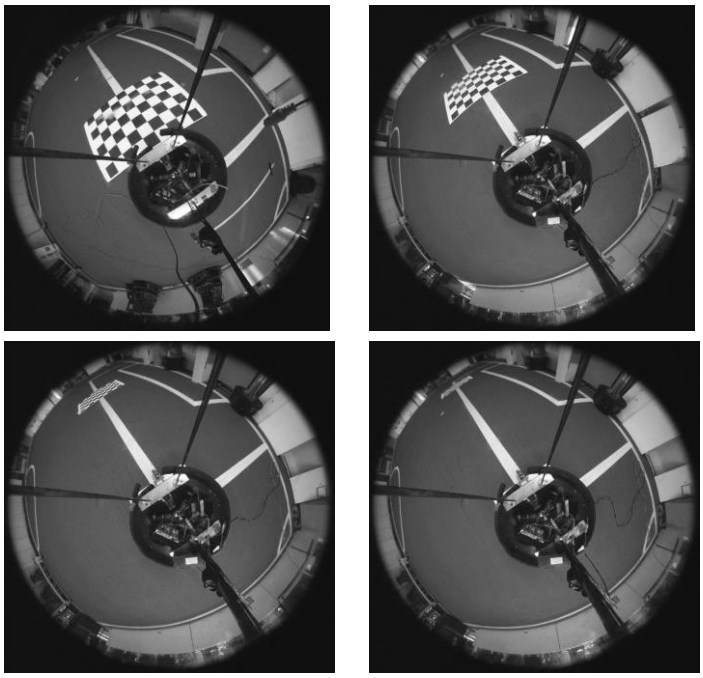

Figure 4.2: A few of the images used for estimation of error variance, patterns at distances of 25cm, 100cm, 250cm and 400cm.

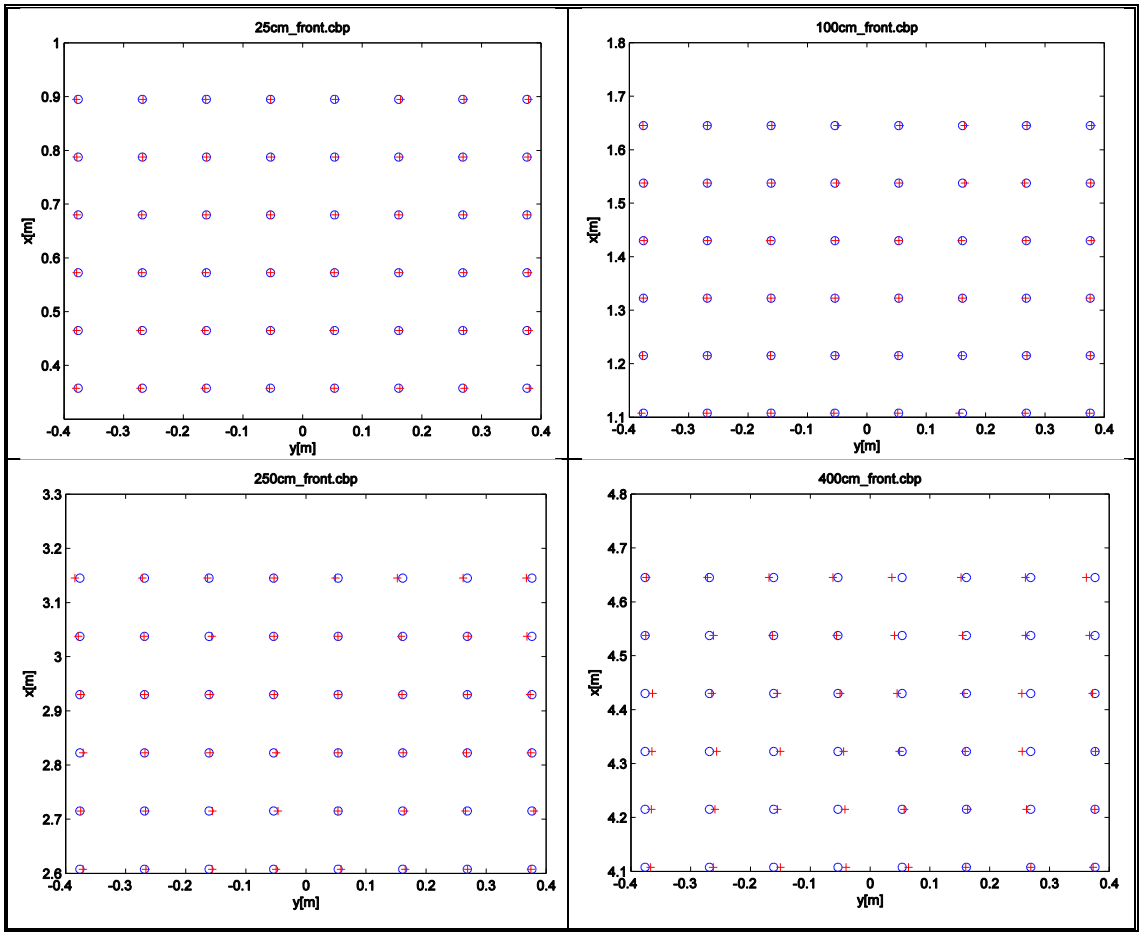

Figure 4.3: Measurements of pattern points obtained for the patterns positioned at multiple distances.

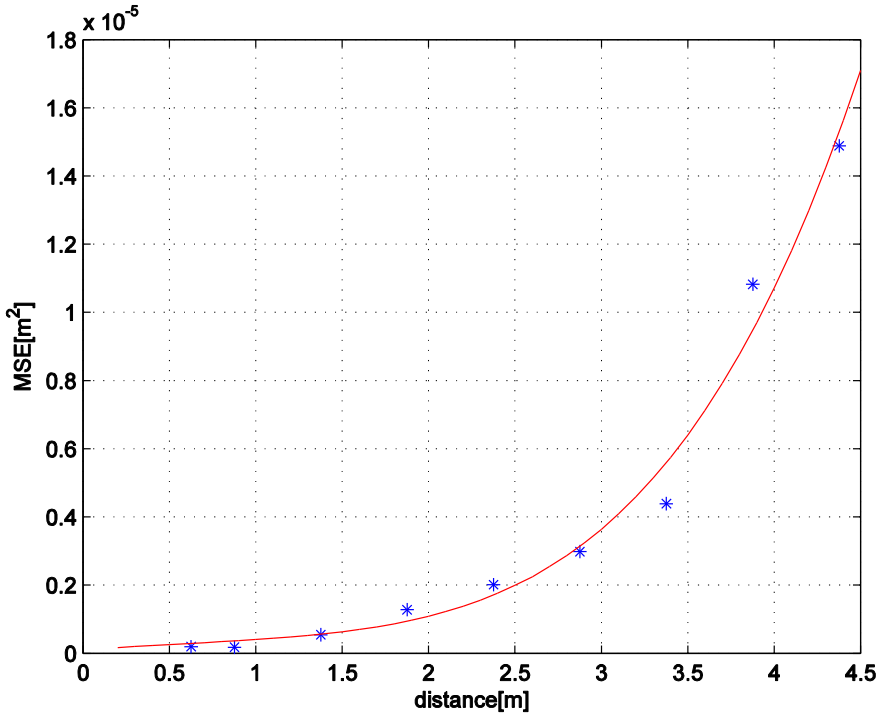

Figure 4.4: Estimate of the ground plane position measurement error variance as a function of distance.

4.3 DETECTION AND CLASSIFICATION.

This section deals with the process of extracting obstacle information, namely position and type of obstacle, from the omnidirectional image. The first step in this process is to segment the image as described in Chapter 2. Three colors are relevant for obstacle detection: black, magenta and cyan. While the robots chassis should be mostly of black color, the magenta and cyan colors are used as team identifiers. After the image is segmented, morphological filtering is performed, more specifically an opening operation. This filtering stage is meant to eliminate small disconnected blobs which merely represent noise and smooth out the significant blobs, and can be loosely interpreted as low pass filtering of obstacle information. A morphological opening operation is composed of an erosion operation followed by a dilation operation, performed on each pixel $P(x,y)$, as defined in equations (4.6) and (4.7) respectively, where $P''(x,y)$ represents the final pixel result. These operations use a small binary image as a structuring element E. A 5x5 pixel square proved adequate for this application.

$$P'(x,y) = \underset{(x',y'):\, E(x',y')\neq 0}{\text{Min}} P(x+x', y+y') \tag{4.6}$$

$$P''(x,y) = \underset{(x',y'):\, E(x',y')\neq 0}{\text{Max}} P'(x+x', y+y') \tag{4.7}$$

The detection scheme relies on a ray casting method instead of explicit identification and labeling of blobs. The former method is a lot more computationally efficient, and in this particular application the pixels significant to the detection of an obstacle's position are the part of the blob's boundary pixels that is visible from the robot's center. This pixels are the ones that correspond to positions in the ground plane, for which a coordinate transformation is available. A small downside of using ray casting is that as rays get further from the robot's center they get further apart, consequently increasing the detection error variance. However, because this happens at distances from the robot center where the measurement resolution is already low, this detrimental effect can for practical purposes be neglected.

Several rays are casted radially from the robot's geometrical center with an angular resolution of one degree, as depicted in figure 4.5. An image mask is used to make sure the rays don't intersect the robots body, including the uprights that support the camera assembly.

Each ray is checked pixel by pixel from the robots center to the outside, until the ray ends or a pixel that was segmented as part of an obstacle is found, in which case the ray continues to be checked until the end or a pixel that was not segmented as part of an obstacle is found. The distance from the first pixel to the second is called the detection length. Adjacent rays in which a detection was registered are then grouped together, resulting in a set of obstacle candidates.

Further filtering to exclude certain candidates is then necessary in order to avoid false positive detections. Some common objects that can be found in the playing field, such as cables, pieces of duct tape, seams in the green field and even trash may be incorrectly identified as an obstacle. Some heuristics were devised to exclude this detections from the set of candidates. First, the extremities of all candidates are trimmed, by excluding the first rays from both sides until a ray is found whose detection length is above a certain threshold. Next, objects in which a large percentage of the rays have a detection length below a certain threshold are discarded. Finally, an area criteria discards candidates for which the sum of the detection lengths is below a certain threshold.

It was defined that each detected obstacle is to be represented by a distance and a width from the robot's center. It can equally be represented by two points corresponding to vertices of an isosceles triangle with two edges collinear with the rays at the extremities of the detected obstacle and with the other edge tangent to the blob, as illustrated in figure 4.6.

At this stage a classification procedure is performed on each of the obstacles to try to identify whether it is a robot and of which team. Three classes of obstacles are defined, one for each team and one undefined obstacle class. An undefined obstacle, is an obstacle that is significant to navigation and avoidance purposes but was not classified as being a robot belonging to either the cyan or magenta teams.

The classification of the detected obstacles, follows a simple procedure. For each obstacle the distance in between the two points that represent it is checked to see if it is approximate, within a certain tolerance, to the expected diameter of a robot which is limited by

MSL soccer rules. If not, the obstacle is immediately classified as an undefined obstacle. On the other hand, if the width of the obstacle is consistent with a robot's width, the obstacle has to be subjected to further classification to try to identify which team it belongs to.

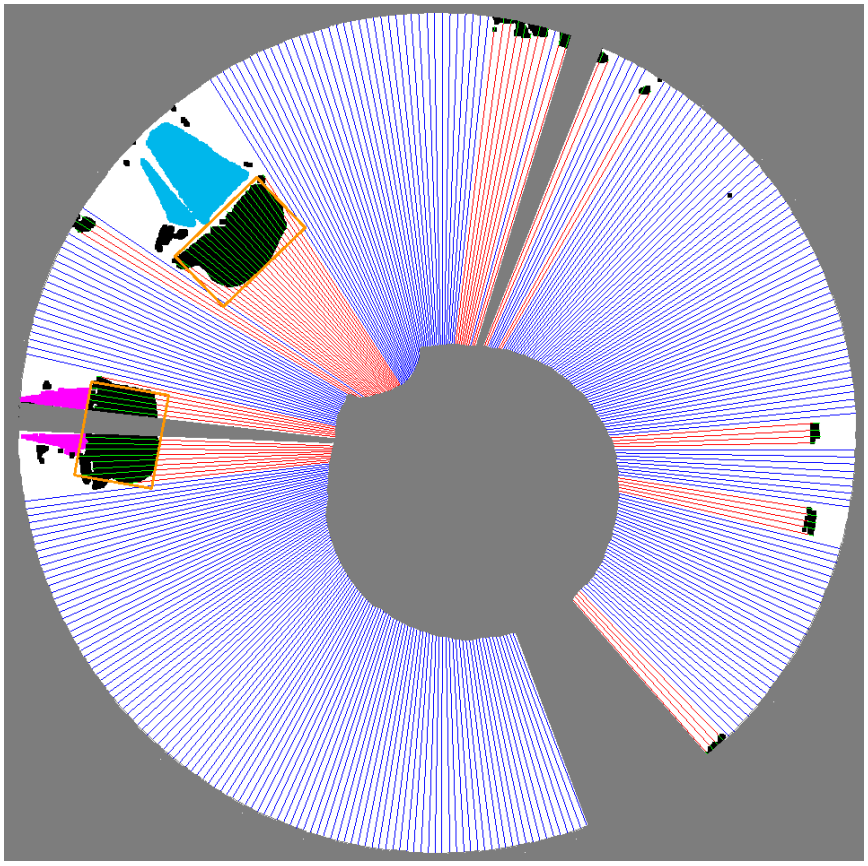

Figure 4.5: Ray-casting detection

A circular area of the image is chosen for each obstacle that is centered near to the centroid of the robots color identifier blob if one exists. Notice the position and shape of the color identifier is not set by the rules and varies from one team to another, so the radius of this circular area must be generous enough to allow for this variation. If the percentage of pixels in that circular area that were segmented to a team color is above a certain threshold then the obstacle is classified as a robot belonging to that team. Otherwise the obstacle classification defaults to undefined. The distance D and radius R of the circular area used for color classification are obtained by the following very simple heuristics,

$$D = \alpha(d + w) \tag{4.8}$$

$$R = \beta w \tag{4.9}$$

Where d and w are the obstacle width and distance respectively, and the linear factors α and β are determined experimentally.

One issue remains to be addressed. It's the situation that occurs when two obstacles are closer to each other than the width of the robot, i.e. the gap between obstacles does not allow the robot to pass. In this case the narrow gap constitutes a *de facto* obstacle to the motion of the robot and so a new obstacle of undefined class in included in the detection output. If any of the obstacles that compose this new merged obstacle has been classified as a robot it still gets to be included in the detection output, otherwise if it was classified as undefined it is excluded. This merging process is done iteratively until there are no more obstacles to be merged.

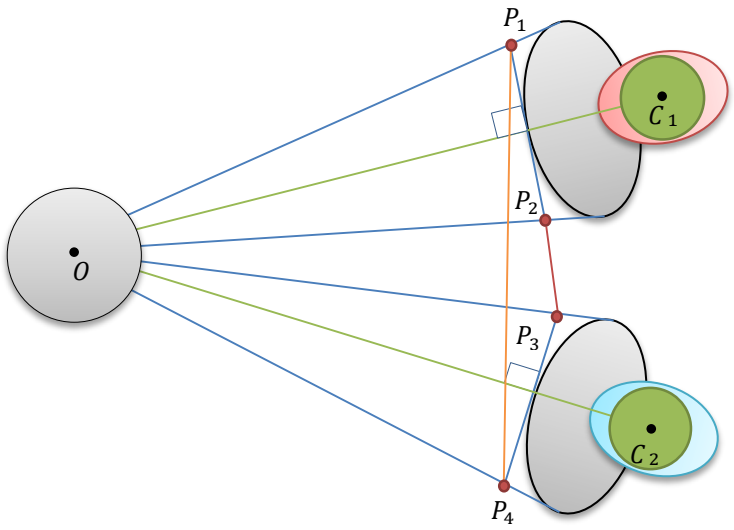

Figure 4.6: Detection example.

As an example, consider the case presented in the previous figure. In this example there are two robot obstacles, represented by the grey ovals on the right hand side. On the left is represented the observer robot and its geometric center O from where the rays are casted. The position of each of the detected obstacles is defined by two points P_i. The green circles are the color classification areas. In this particular case the system would output the following detection results:

- Object delimited by the world coordinates of P_1 and P_2, classified as a robot by the distance between the world coordinates P_1 and P_2 and further classified as a member of magenta team by analysis of the pixels centered around C_1.
- Object delimited by the world coordinates of P_3 and P_4, classified as a robot by the distance between the world coordinates P_3 and P_4 and further classified as a member of cyan team by analysis of the pixels centered around C_2.
- Object delimited by the world coordinates of P_1 and P_4, classified as an undefined obstacle, created after verifying that the distance between the coordinates of P_2 and P_3 is narrower than the width of a robot.

5 OBSTACLE TRACKING

5.1 INTRODUCTION

Up until this point, all the focus has been on extracting obstacle information from an image, which is information concerning the instant the image was captured. This chapter is concerned with the motion of the obstacles over time, as perceived by the observer robot. Many higher level tasks such as obstacle avoidance, ball passing and interception, amongst other, can greatly benefit from having some knowledge about the motion of the obstacles and being able to anticipate their state further in time. This added level of insight, on what is a very dynamic environment, allows the implementation and use of more sophisticated playing behaviors. Additionally, more accurate state estimates can be obtained by combining the instantaneous state observations with the predicted state based on past state estimates and control inputs, provided that there is some previous knowledge on the noise characteristics of both the observations and the predictions. Furthermore, having the ability to predict an obstacle state means that even if an obstacle is temporarily lost from detection, for example when occluded by another, it is still possible to keep tracking it so that when eventually the observations are resumed that obstacle can be correctly identified as the one previously lost from detection.

The remainder of this chapter is laid out as follows. Initially, the model of the dynamics of the obstacles is derived. This model is used to predict the state evolution, by considering the previous state estimate, the velocity controls on the observer robot and the elapsed time. Next, the method used to estimate the error covariance associated with the state predictions generated by the model of the dynamics is addressed. After that, the Kalman filter is presented. This recursive optimal estimator, uses a prediction of the current state, and the current state observations to produce an optimal state estimation. However, to be able to do this it needs to be able to known which predictions correspond to each of the observations. This problem can be modeled as an assignment optimization problem, which is then solved resorting to the well-known Hungarian algorithm. Finally, at the end of this chapter an illustrative example is presented, where all components work together in a game scenario.

5.2 MODEL OF THE DYNAMICS

The velocity of an obstacle in relation to the observer robot can be decomposed in three components: velocity of the target's own motion in relation to the observer, an apparent velocity caused by the linear motion of the observer robots reference frame and apparent velocity caused

by the observer's robot angular velocity. In the example illustrated in figure 5.1 we can observe the velocity component vectors. If the observer robot travels with velocity $\boldsymbol{v_o}$ in relation to the world reference frame, then a static landmark that it observes appears to be moving in the opposite direction with velocity$-\boldsymbol{v_o}$. In a similar way, if the reference frame of the observer robot rotates with angular velocity $\boldsymbol{\omega_0}$, any static landmark it observes will appear to be orbiting around its origin in the opposite direction with tangential velocity $\boldsymbol{v}_\omega$.

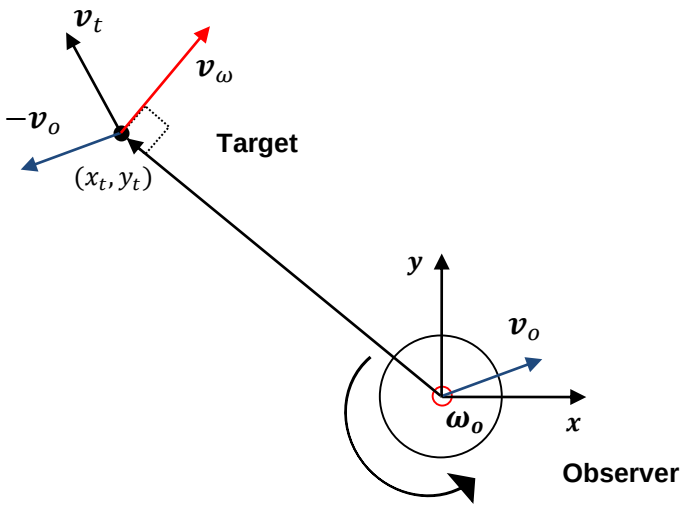

Figure 5.1: Obstacle velocity components.

The velocity of an object moving under uniform circular motion with an angular velocity $\boldsymbol{\omega}$ and radius $\boldsymbol{r}$ is given by,

$$\boldsymbol{v} = \boldsymbol{\omega} \times \boldsymbol{r} \tag{5.1}$$

The component of the target's velocity due to the angular velocity of the observer robot and assuming constant velocity is given by,

$$\boldsymbol{v}_w = -\boldsymbol{\omega_o} \times \boldsymbol{r} \tag{5.2}$$

This equation can be written as the formal determinant,

$$\boldsymbol{v}_\omega = \begin{vmatrix} \boldsymbol{x} & \boldsymbol{y} & \boldsymbol{z} \\ 0 & 0 & -\omega \\ x_t & y_t & 0 \end{vmatrix} = \omega y_t \boldsymbol{x} - \omega x_t \boldsymbol{y} \tag{5.3}$$

Adding all three components the velocity we obtain,

$$\boldsymbol{v} = \begin{bmatrix} v_x \\ v_y \end{bmatrix} = \begin{bmatrix} v_{t_x} - v_{ox} + \omega y_t \\ v_{ty} - v_{oy} - \omega x_t \end{bmatrix} \tag{5.4}$$

Taking the time derivative under the assumption of constant velocities, we obtain,

$$\dot{\boldsymbol{v}} = \begin{bmatrix} \dot{v}_x \\ \dot{v}_y \end{bmatrix} = \begin{bmatrix} \omega v_{yt} \\ -\omega v_{xt} \end{bmatrix} \tag{5.5}$$

The continuous time space-state model, for a state $\boldsymbol{x} = (\, x_t \,, y_t \,, v_{t_x} \,, v_{t_y})^T$ expressed in the observer robot's reference frame is,

$$\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u} \tag{5.6}$$

$$\boldsymbol{A} = \begin{bmatrix} 0 & \omega & 1 & 0 \\ -\omega & 0 & 0 & 1 \\ 0 & 0 & 0 & \omega \\ 0 & 0 & -\omega & 0 \end{bmatrix} \qquad \boldsymbol{B} = \begin{bmatrix} -1 & 0 \\ 0 & -1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{5.7}$$

It is now necessary to obtain a discrete time model. The method used here, for discretization of a continuous system, is described in [12], and is as follows,

$$\dot{\boldsymbol{v}}(k) = \boldsymbol{\Phi}\boldsymbol{v}(k-1) + \boldsymbol{\Gamma}\boldsymbol{u}(k) \tag{5.8}$$

$$\boldsymbol{\Phi} = \boldsymbol{e}^{\boldsymbol{A}T} = \boldsymbol{I} + \boldsymbol{A}T + \frac{\boldsymbol{A}^{\boldsymbol{2}}T^2}{2!} + \frac{\boldsymbol{A}^{\boldsymbol{3}}T^3}{3!} + \cdots \tag{5.9}$$

$$\boldsymbol{\Gamma} = \sum_{\boldsymbol{k=0}}^{\infty} \left(\frac{\boldsymbol{A}^{\boldsymbol{k}}T^{k+1}}{(k+1)!} \right) \boldsymbol{B} = \boldsymbol{A}^{-1}\,(\boldsymbol{\Phi} - \boldsymbol{I})\boldsymbol{B} \tag{5.10}$$

The series (5.9) can be separated into its even and odd components,

$$\boldsymbol{\Phi}_{even} = \sum_{n=0}^{\infty} \frac{(-1)^n T^{2n} \omega^{2n-1}}{(2n)!} \begin{bmatrix} \omega & 0 & 0 & -2n \\ 0 & \omega & 2n & 0 \\ 0 & 0 & \omega & 0 \\ 0 & 0 & 0 & \omega \end{bmatrix} \tag{5.11}$$

$$\boldsymbol{\Phi}_{odd} = \sum_{n=0}^{\infty} \frac{(-1)^n T^{2n+1} \omega^{2n}}{(2n+1)!} \begin{bmatrix} 0 & \omega & (2n+1) & 0 \\ -\omega & 0 & 0 & (2n+1) \\ 0 & 0 & 0 & \omega \\ 0 & 0 & -\omega & 0 \end{bmatrix} \tag{5.12}$$

Identifying in the above equations the Maclaurin's series for the cosine and sine functions and recombining the even and odd parts,

$$\boldsymbol{\Phi} = \begin{bmatrix} cos(\theta) & sin(\theta) & Tcos(\theta) & Tsin(\theta) \\ -sin(\theta) & cos(\theta) & -Tsin(\theta) & Tcos(\theta) \\ 0 & 0 & cos(\theta) & sin(\theta) \\ 0 & 0 & -sin(\theta) & cos(\theta) \end{bmatrix} \tag{5.13}$$

Applying a similar reasoning we obtain the discrete input matrix and also the complete model of the dynamics of the obstacles.

$$\boldsymbol{\Gamma} = \begin{bmatrix} -\frac{T}{\theta}sin(\theta) & \frac{T}{\theta}(cos(\theta)-1) \\ -\frac{T}{\theta}(cos(\theta)-1) & \frac{T}{\theta}sin(\theta) \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{5.14}$$

5.3 PROCESS ERROR COVARIANCE

This section deals with the estimation of the error covariance associated with the model of the dynamics of the obstacles derived in the previous section. The method presented here relies on the ground-truth system presented in Chapter 3. The first step in the process is to collect a significant amount of experimental data. This data is composed of the robot's camera captured images, velocity controls, odometry data, and the images captured from the ground-truth system cameras. All the machines involved in data capture must be synchronized across the network, and all the captured data timestamped. With the concern of statistical significance, it should be ensured that the collected datasets contain a lot of variety. In this particular case, both the observer robot and the target obstacles performed a diverse collection of different motion patterns. These patterns included rectangular and circular figures of varying size, performed at different velocities, and with the robots at different distances.

Each collected dataset is processed by the ground truth system to obtain a list of timestamped positions for each robot. Using this information, and assuming constant velocities, the ground-truth estimate of the obstacle state is then constructed according to the following equation,

$$\boldsymbol{S}(k) = \begin{bmatrix} x_{tar}(k) - x_{obs}(k) \\ y_{tar}(k) - y_{obs}(k) \\ \frac{\big(x_{tar}(k+1) - x_{obs}(k+1)\big) - (x_{tar}(k) - x_{obs}(k))}{t(k+1) - t(k)} \\ \frac{\big(y_{tar}(k+1) - y_{obs}(k+1)\big) - (y_{tar}(k) - y_{obs}(k))}{t(k+1) - t(k)} \end{bmatrix} \tag{5.15}$$

Where x_{tar}, y_{tar}, x_{obs} and y_{obs} are the coordinates in the world reference frame of the obstacle robot and the observer robot respectively, and $t(k)$ is the timestamp associated with the *kth* sample.

The prediction error values are dependent on many variables. It is impractical and unnecessary to take many of this variables into consideration. For convenience only two of this

variables, that were perceived to be the most significant, were considered. This variables are the time period between iterations and the magnitude of the targets velocity in relation to the observer. After each dataset is processed the error values are organized into bins according to the values of this variables.

Notice that the ground-truth values are noisy measurements of real positions. It is assumed that on average the error in the state estimated from ground-truth measurements is much smaller than the error in the state predicted from the model of the dynamics. The ground-truth error is thus considered insignificant, and the measurements are treated as the real position values. The error is then calculated according to equation (5.16). This error is calculated for different sampling time intervals by varying the value of n.

$$\boldsymbol{E_n}(k) = \boldsymbol{S}(kn) - (\boldsymbol{\Phi S}\big((k-1)n\big) + \boldsymbol{\Gamma u}(kn)) \tag{5.16}$$

The error variances for the both coordinates of the position are assumed to be equal. The same is assumed for the velocity. The position and velocity variances are then estimated using the mean squared error (MSE). The results as a function of the obstacle velocity magnitude and the sampling period are present in the tables 5.1 and 5.2. It can be observed in these tables that the behavior of the variances is for the most part what would be expected, the increase of the sampling period as well as the increase in the obstacle velocity, cause an increase in the error variances.

Position Error Variance				
$s \times 10^{-3}\ ;\ ms^{-1}$	$0 \leq \|\boldsymbol{v}\| < 1$	$1 \leq \|\boldsymbol{v}\| < 2$	$2 \leq \|\boldsymbol{v}\| < 3$	$3 \leq \|\boldsymbol{v}\|$
$0 < T < 50$	0.000439532	0.00211673	0.00485432	0.0141425
$50 \leq T < 200$	0.00384609	0.0262091	0.0623425	0.17625
$200 \leq T < 500$	0.0208261	0.151046	0.286312	0.81821
$500 \leq T$	0.0533175	0.375216	0.829724	1.88844

Table 5.1: Position error variance results.

Velocity Error Variance				
$s \times 10^{-3}\ ;\ ms^{-1}$	$0 \leq \|\boldsymbol{v}\| < 1$	$1 \leq \|\boldsymbol{v}\| < 2$	$2 \leq \|\boldsymbol{v}\| < 3$	$3 \leq \|\boldsymbol{v}\|$
$0 < T < 50$	0.0927498	0.199694	0.274756	0.372132
$50 \leq T < 200$	0.0759916	0.219989	0.766296	0.83813
$200 \leq T < 500$	0.0402036	0.284529	1.84588	2.53957
$500 \leq T$	0.0492175	0.406586	2.8862	6.94383

Table 5.2: Velocity error variance results.

5.4 KALMAN FILTER

The state estimation is performed resorting to the Kalman filter, a well know optimal linear estimator. This estimator is optimal in the sense that if the noise is Gaussian distributed it minimizes the mean squared error of the estimated parameters. It is adequate for real time applications since it is a recursive estimator, i.e., new measurements are processed as they arrive. The derivation of the Kalman filter is involved and beyond the scope of this work. However, the brief explanation presented here should be enough for an intuitive and practical understanding on how the Kalman filter works. For an in-depth introduction to the Kalman filter and its applications refer to [6] and [11].

Using a Kalman filter for state estimation is advantageous for multiple reasons. First, by doing a weighted average between the observations and the predicted state, it produces a more accurate state estimate than it would be obtained by the sensor observations alone. Second, it assists in the data association task by allowing the observed obstacles to be associated with the predicted states of the tracked obstacles, instead of associating them with their previous state estimation.

The Kalman filter algorithm can be divided in two steps, the prediction step and the measurement update step. The first two equations constitute the time update or prediction. In equation (5.17) the previously derived model of the dynamics is used to predict the current state. This prediction $\hat{\boldsymbol{x}}_k^-$ is called the prior estimate. The error covariance is also projected ahead in (5.18), where $\boldsymbol{Q}$ represents the process error covariance addressed in the previous section.

$$\hat{\boldsymbol{x}}_k^- = \boldsymbol{\Phi}\hat{\boldsymbol{x}}_{k-1} + \boldsymbol{\Gamma}\boldsymbol{u}_{k-1} \tag{5.17}$$

$$\boldsymbol{P}_k^- = \boldsymbol{\Phi}\boldsymbol{P}_{k-1}\boldsymbol{\Phi}^T + \boldsymbol{Q} \tag{5.18}$$

The next three equations constitute the measurement update step. In equation (5.19) the Kalman gain $\boldsymbol{K}_k$ is calculated. This gain is used to weight the average between the predicted state and the measurement. The matrix $\boldsymbol{H}$ represents the relation between the state and the measurement. In this case only the first two elements of the state, the position of the target, are measured. Matrix $\boldsymbol{H}$ is therefore a 4x4 matrix off zeros except for the first two elements of the diagonal which are ones. Next the posterior estimate $\hat{\boldsymbol{x}}_k$ is calculated in equation (5.20) where the quantity $\boldsymbol{z}_k - \boldsymbol{H}\hat{\boldsymbol{x}}_k^-$ is called the innovation. Finally in equation (5.21) the error covariance is also updated.

$$\boldsymbol{K}_k = \boldsymbol{P}_k^-\boldsymbol{H}^T(\boldsymbol{H}\boldsymbol{P}_k^-\boldsymbol{H}^T + \boldsymbol{R})^{-1} \tag{5.19}$$

$$\hat{\boldsymbol{x}}_k = \hat{\boldsymbol{x}}_k^- + \boldsymbol{K}_k(\boldsymbol{z}_k - \boldsymbol{H}\hat{\boldsymbol{x}}_k^-) \tag{5.20}$$

$$\boldsymbol{P}_k = (\boldsymbol{I} - \boldsymbol{K}_k \boldsymbol{H})\, \boldsymbol{P}_k^- \tag{5.21}$$

5.5 DATA ASSOCIATION

In order to be able to perform the measurement update step of the Kalman filter it's necessary to know which of the observed obstacles corresponds to each of the predicted ones. This can be modeled as an assignment problem which is solvable using the well know Hungarian algorithm. Modern implementations of this algorithm can achieve a time complexity of $O(n^3)$, being adequate for real-time applications in case the number of correspondences is not very large. A thorough description on assignment problems and the Hungarian algorithm can be found in [15].

The data association is done per obstacle class, meaning detected obstacles of one class are matched with the tracked obstacles of the same class. For each class a table of costs is created to be used as the input of the Hungarian algorithm. In this case the Euclidean distance proved adequate to be chosen as cost function. Other cost functions, such as the Mahalanobis distance, which takes into account the error variance, can be used if deemed necessary.

Three different situations need to be account for. First is the case when the number of detected obstacles N is the same as the number of predictions M. This situation results in a cost table of the type show in table 5.3, where d_i denotes the *ith* detected obstacle, p_i the *ith* prediction, and $\|\boldsymbol{P}_\mathrm{i} - \boldsymbol{D}_\mathrm{i}\|$ the distance between their positions. The table 5.4 represents another possible situation, which is to have more detections that predictions. Meaning that new obstacles that weren't previously being tracked are now detected. In this case dummy predictions need to be included as extra columns in the cost matrix so that the matrix becomes square. All costs in the dummy columns are made equal to the largest of the previously calculated costs. The final situation is represented in table 5.5, and corresponds to the case when the number of predictions is higher that the number of detections. This corresponds to the case when some obstacles that were being tracked are lost from detection but have not yet been discarded from the list of tracked obstacles. In this case the matrix needs to be made square by inserting dummy rows with all cost entries equal to the largest calculated cost. In any case the resulting cost matrix is used as input for the Hungarian algorithm, which in turn calculates and outputs the correspondences between observation and predictions, such that it minimizes the sum of the costs

	p_1	p_2	$\cdots$	p_M
d_1	$\|\boldsymbol{P}_1 - \boldsymbol{D}_1\|$	$\|\boldsymbol{P}_2 - \boldsymbol{D}_1\|$	$\cdots$	$\|\boldsymbol{P}_M - \boldsymbol{D}_1\|$
d_2	$\|\boldsymbol{P}_1 - \boldsymbol{D}_2\|$	$\|\boldsymbol{P}_2 - \boldsymbol{D}_2\|$	$\cdots$	$\|\boldsymbol{P}_M - \boldsymbol{D}_2\|$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
d_N	$\|\boldsymbol{P}_1 - \boldsymbol{D}_n\|$	$\|\boldsymbol{P}_2 - \boldsymbol{D}_n\|$	$\cdots$	$\|\boldsymbol{P}_M - \boldsymbol{D}_N\|$

Table 5.3: Cost table with same number of detection and predictions.

	⋯	p_M	$dummy_{M+1}$	⋯	$dummy_N$
⋮	⋮	⋮	⋮	⋮	⋮
d_M	⋯	$\|\boldsymbol{P}_M - \boldsymbol{D}_M\|$	$\max_{i,j}\|\boldsymbol{P}_i - \boldsymbol{D}_j\|$	…	$\max_{i,j}\|\boldsymbol{P}_i - \boldsymbol{D}_j\|$
d_{M+1}	⋯	$\|\boldsymbol{P}_M - \boldsymbol{D}_{M+1}\|$	$\max_{i,j}\|\boldsymbol{P}_i - \boldsymbol{D}_j\|$	…	$\max_{i,j}\|\boldsymbol{P}_i - \boldsymbol{D}_j\|$
⋮	⋮	⋮	⋮	⋮	⋮
d_N	⋯	$\|\boldsymbol{P}_M - \boldsymbol{D}_N\|$	$\max_{i,j}\|\boldsymbol{P}_i - \boldsymbol{D}_j\|$	…	$\max_{i,j}\|\boldsymbol{P}_i - \boldsymbol{D}_j\|$

Table 5.4: Cost table with more detections than predictions.

	⋯	p_N	p_{N+1}	⋯	p_M
⋮	⋮	⋮	⋮	⋮	⋮
d_N	⋯	$\|\boldsymbol{P}_N - \boldsymbol{D}_N\|$	$\|\boldsymbol{P}_{N+1} - \boldsymbol{D}_N\|$	⋯	$\|\boldsymbol{P}_M - \boldsymbol{D}_N\|$
$dummy_{N+1}$	…	$\max_{i,j}\|\boldsymbol{P}_i - \boldsymbol{D}_j\|$	$\max_{i,j}\|\boldsymbol{P}_i - \boldsymbol{D}_j\|$	…	$\max_{i,j}\|\boldsymbol{P}_i - \boldsymbol{D}_j\|$
⋮	⋮	⋮	⋮	⋮	⋮
$dummy_M$	…	$\max_{i,j}\|\boldsymbol{P}_i - \boldsymbol{D}_j\|$	$\max_{i,j}\|\boldsymbol{P}_i - \boldsymbol{D}_j\|$	…	$\max_{i,j}\|\boldsymbol{P}_i - \boldsymbol{D}_j\|$

Table 5.5: Cost table with more predictions than detections.

5.6 INSERTION AND REMOVAL OF TRACKED OBSTACLES

After the data association step, each entry in a class obstacle list represents one of three cases: it can be a new obstacle, such as when the number of detected objects of a given class is higher than the obstacles already being tracked for that class; it can be an already tracked obstacle that was associated with a detected obstacle or it can be an already tracked obstacle that was not associated with any of the detected obstacles, this can happen when the number of detected obstacles is lower that the number of tracked obstacles. A tracked obstacle which is no longer being detected cannot remain indefinitely in the list. The following algorithm contains the rules for the removal of obstacles no longer being detected

$$\begin{aligned} &for\ every\ class\ in\ classes: \\ &\quad for\ every\ obstacle\ in\ list(class): \\ &\quad\quad if\ obstacle\ is\ new: \end{aligned} \tag{5.22}$$

$$
\begin{aligned}
&\quad obstacle.ttl = initial\_ttl(class);\\
&else:\\
&\quad if\ obstacle\ was\ detected:\\
&\qquad if\ obstacle.ttl\ != max\_ttl(class):\\
&\qquad\quad increment(obstacle.ttl);\\
&\quad else:\\
&\qquad decrement(obstacle.ttl);\\
&if\ obstacle.ttl == 0:\\
&\quad list(class).remove(obstacle)
\end{aligned}
$$

Each obstacle has associated with it a *time to live* (TTL) value, that is updated with every iteration of the detection system. On the creation of a new obstacle an initial TTL value is set depending on its class. With each system iteration the TTL value is either incremented (up to a maximum) if the obstacle was detected, or decremented otherwise. If an object TTL reaches 0, that object is removed from the respective list. The initial and maximum TTL values for each class have to be chosen judiciously, so that a good balance between a quick rejection of spurious false positives and retention of tracks during temporary loss of detection is achieved.

5.7 EXPERIMENTAL PROCEDURES

The experiments that were carried out had two main purposes. First it was necessary to adjust the values of the various parameters of the detection and tracking, and second the evaluation of the performance of the system as whole once the parameters were optimally tuned. The followed methodology was informal, consisting mostly on trying to reproduce as good as possible real game scenarios, identifying problems as they occurred and trying to mitigate them by adjusting parameters on a trial and error basis. Special attention was dedicated to cases that were expected to be problematic, such as of heterogeneous lighting conditions and clustering of obstacles. Overall the system has performed very well, producing the desired output in the vast majority of tested cases.

The example that follows was chosen because although simple, it clearly illustrates the working of the various components of system in dealing with a common yet potentially challenging situation. In this example two obstacles are initially detected and classified as members of the cyan team. The obstacles are attributed labels 1 and 2 of the cyan class. This initial configuration can be observed in figure 5.2, where on the left side is shown the omnidirectional cameras image and on the right the system output.

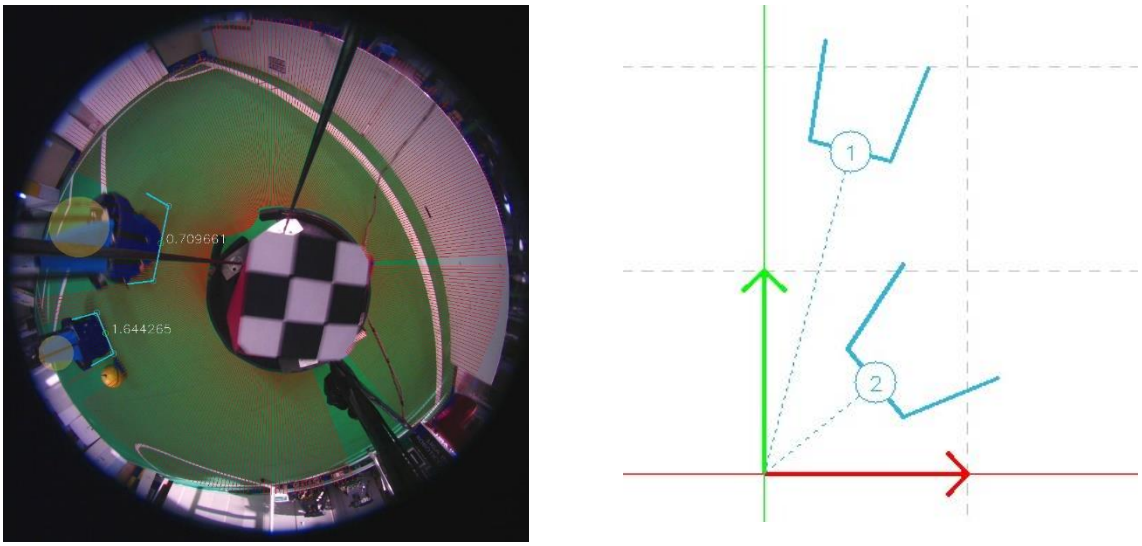

Figure 5.2: Tracking example – Frame 1.

In figure 5.3 it can be observed that as the obstacles move towards each other, the obstacle closer to the observer, cyan 2, starts to occlude the one that is further away, and the system no longer detects two separate robots. Instead an obstacle of undefined class is detected and is given the label 7. Even thou cyan obstacle 1 and 2 are no longer detected they are no discarded and their state continues to be predicted according to the motion model.

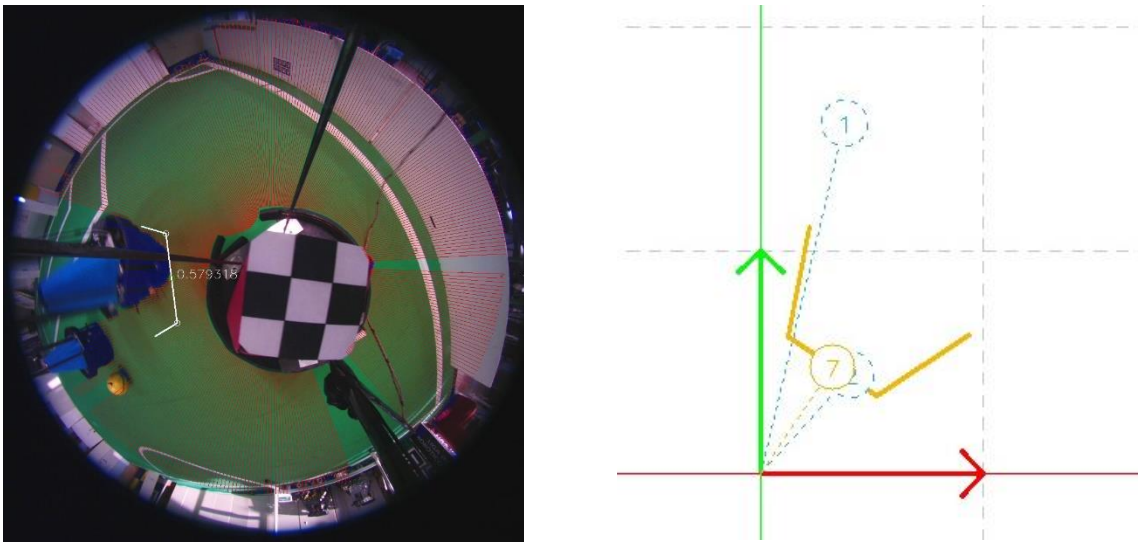

Figure 5.3: Tracking example – Frame 2.

Next, in figure 5.4 the cyan obstacle 1 gets completely occluded by cyan obstacle 2 the systems recognizes the latter and discards the undefined obstacle.

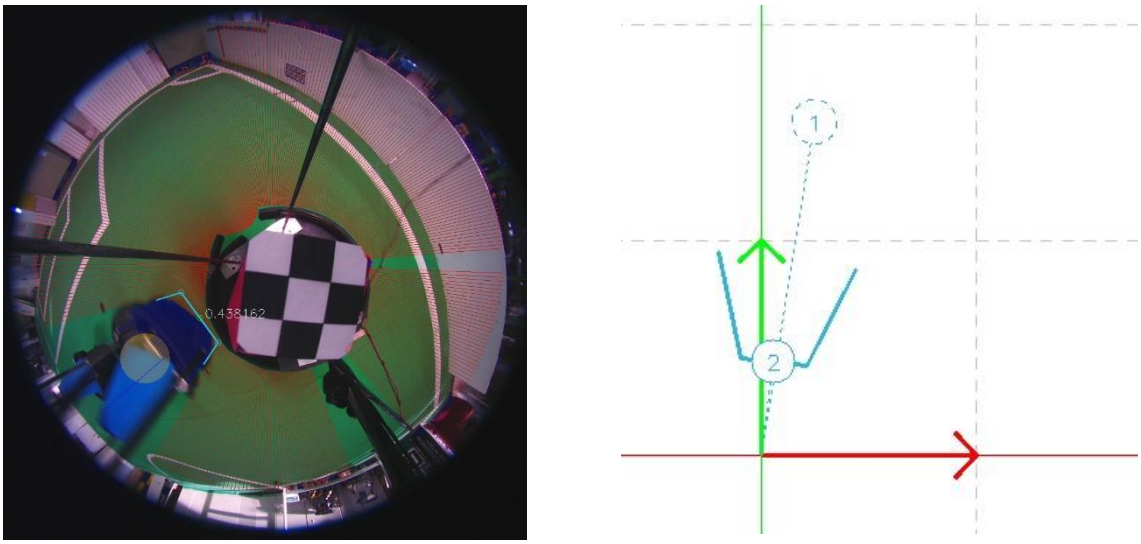

Figure 5.4: Tracking example – Frame 3.

When cyan obstacle 1 starts to pop in from behind cyan obstacle 2, as can be observed in figure 5.5 the system detects once again an undefined obstacle and assigns it label 8. The two cyan obstacles continue being tracked.

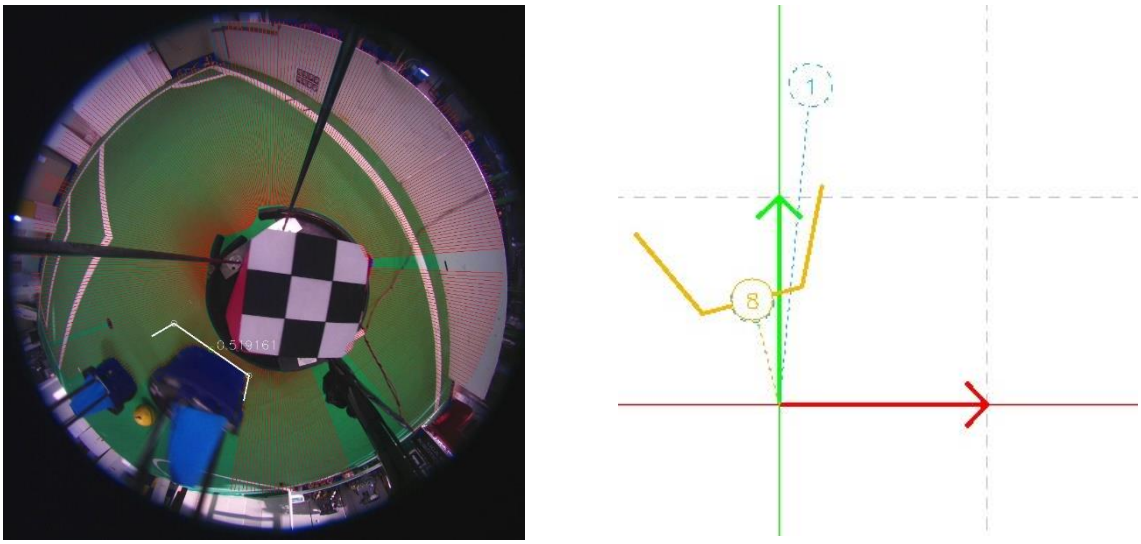

Figure 5.5: Tracking example – Frame 4.

Figure 5.6 represents the final configuration of this example. All undefined obstacles have been discarded. The detection of the two obstacles has been resumed and their label is the same as originally attributed.

Throughout this example the targets have been correctly tracked, even when not detected the state of the object continued to be predicted and when the detection was resumed the obstacles where correctly identified with their original label.

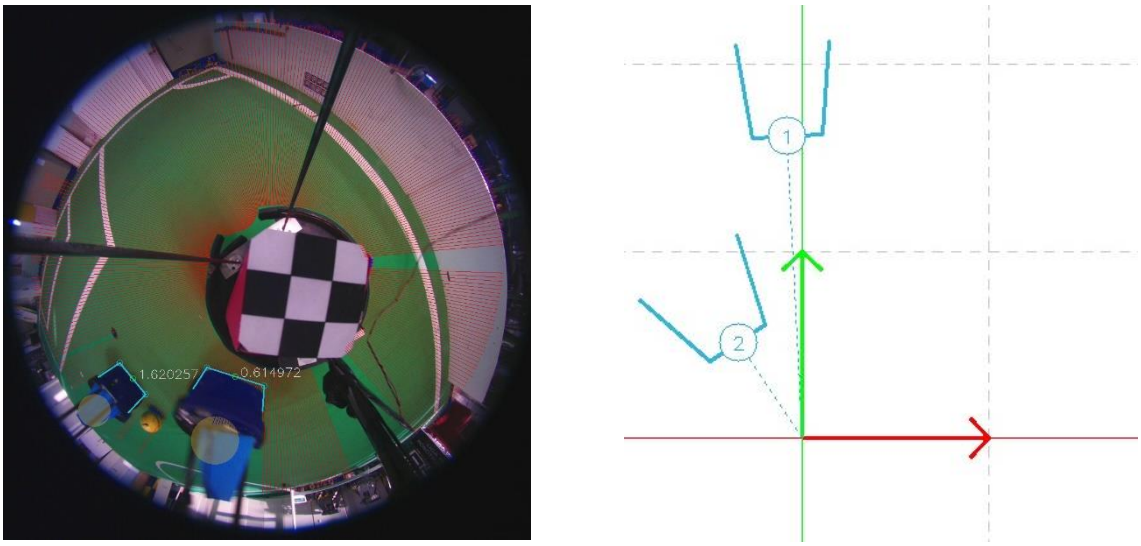

Figure 5.6: Tracking example – Frame 5.

6 CONCLUSIONS AND FUTURE WORK

6.1 CONCLUSIONS

Each of the previous chapters has addressed the development of an individual component of what was the ultimate goal of this work, a complete obstacle tracking system for MSL robots. It is the purpose of this section to elaborate some conclusions and final considerations on both the individual components and the system as a whole. These conclusions, although qualitative for the most part were as much as possible supported on the observation of experimental results. It should be noted, that considering the myriad of possible state configurations and variables found in actual MSL soccer matches and considering also the limited laboratorial resources available during the development of this work, namely the small number of available robots, the tests represent isolated and specific conditions, which may not represent the full spectrum of conditions (robot configurations, light conditions, background noise, etc.) That may be found in actual MSL matches. It is evidently difficult or impossible to predict every situation that may present itself as difficult or problematic to the system. However, an honest effort was made in trying to test scenarios that intuition deemed as potential problematic. In other words the tests were not tailored to accommodate any weaknesses in the system, so that it produced the desired results. Instead, the applied methodology was to use the tests to try to reveal potential weaknesses in the system so they could be corrected whenever possible or at the very least identified and characterized.

Whenever it applies, the presented conclusions take into account the evaluation criteria described on the following points in order of importance:

- Effectiveness of the solution in solving the specified problem.
- Improvement over any previous implemented solutions.
- Quality of the software implementation, namely regarding user experience and extensibility of the solutions to accommodate future improvements and new functionality.

In Chapter 2 the color segmentation system was presented. Being a fundamental component on which many other system components depend, a trustable color segmentation is of paramount importance. An unreliable segmentation systems will propagate its faults to higher level systems, having a big detrimental impact on the global performance. Prior to this work the color segmentation had been a major source of reported problems that were observed during MSL matches. The previous system also lacked functionality and was cumbersome to use (a very limited tool was used to created single color LUTs). The color segmentation system that was developed in this work greatly improves over the previous solution. All of this issues have been

corrected in this new solution, which is more reliable, performant and user friendly. The patent lack of robustness to illumination changes has been largely mitigated and it was clearly shown that it originated from a fundamental misconception that was inherent to the previous system. The LUTCreator, which is the interface of the color segmentation system, is a ROS integrated color segmentation application that was developed to be a generic, functional and uncomplicated tool. It has been welcomed by peers as a useful tool for their color segmentation needs.

The Ground-truth system was presented in Chapter 3. Although not directly involved in the robots online task of detecting and tracking obstacles, a ground truth system is a valuable resource, namely for its use in the estimation of noise characteristics. Therefore the development of a ground-truth system was deemed relevant and pertinent in the scope of this work. Before this work there was already a working ground-truth system. That system however presented several limitations, particularly with regard to the detection range, which vastly reduced its usefulness. The ground-truth system developed for this work greatly improves on the limitations of previous system. It allows for the use of the test field in its full extension, while previously only a small area could be used due to problems in the detection. The adopted hardware markers along with the user friendly interface, makes the system less cumbersome to use, reducing the setup time especially when multiple robots are tracked. In the previous work each robot had a uniquely colored marker and had to be individually segmented. The devised method used for fast estimation the error covariance of the estimates worked well, proving to be a viable alternative, especially under time constrains, and as long as the loss of accuracy is acceptable.

Chapter 4 was dedicated to the detection and classification of obstacles. The previous solution was based on blob detection and an area classifier, which could only crudely classify a detected obstacle as being a robot or not, not being able to identify which team it belongs to. Furthermore it showed problems dealing with blobs corresponding to overlapped robots or very close obstacles. The solution developed in this work differs greatly from the previous solution because it is not based on blob detection instead it relies on ray casting. This method allows for better heuristic filtering strategies and has shown to provide a more robust detection even in high noise environment. The detection system is now also capable of distinguishing an adversary from a team mate using the team identification colors and it is capable of dealing with situations where the gap between detected obstacles is too narrow for the robot to pass in-between. In which case it creates compound obstacles improving the obstacle avoidance algorithms and avoiding unnecessary collisions.

The obstacle tracking task was dealt with in Chapter 5. Its purpose is to provide reliable estimates of the obstacles state by combining the state measurements obtained as described in Chapter 4 with the previously obtained state estimates. At first glance, the adopted solution is similar to the previous implemented solution, however on closer inspection, profound changes become apparent. Both solutions rely on the use of a Kalman filter per target obstacle, however the previous solution used one Kalman filter bank for all the tracked obstacles whereas in this solution three Kalman filter banks are used, one per each class off obstacles, simplifying the data

association task. The approach to data association has also been modified. Whereas before data association was performed in a greedy fashion, it is now performed resorting to the Hungarian method, resulting in more a reliable data association that combined with an effective false positive filtering doesn't have a negative effect on the computational efficiency of the system. This solution has shown experimentally to be able to deal effectively with challenging situations such as temporary occlusion or clustering of obstacles in the image.

Finally, on a global level the above elements integrate into a successful system that performs according to the desired specifications, performing real-time robust detection classification and tracking of obstacles in the MSL environment. The system has perform well under the available testing conditions which aimed to reproduce, to the extent that was possible, real MSL match conditions, not shunning from challenging light conditions and cluttered environment situations.

Overall these conclusions can be summarized in the following observations:

- The initial proposed objectives were achieved, i.e. the solution does solve the problem as it was originally specified.
- The solution improves on previously existing system. Both on the individual component level and as an integrated system the current system works significantly better than before. Considering that this is a relatively low level system on which other important higher level systems are dependent (such as obstacle avoidance and some components of decision making) it is expectable that this improvement is to some degree reflected on the overall playing performance of the robots.
- The development of the software, was mindful from the conception all through the implementation, on respecting the principles of usability, scalability and adaptability of the solution. Through the use of object oriented code architecture and profuse configuration options, the software is easy to use as well to maintain and develop, allowing for further expansion of its feature set.

6.2 FUTURE WORK

From the previous section it is clearly established that the system which was implemented works appropriately and the meets all the specified criteria. Nevertheless there is ample room for improvement and this section provides a brief exploration of some of different avenues for further development. When considering the ways the system can evolve, it can be argued that one can distinguish between work meant to improve the effectiveness and or the efficiency of the current solutions or work that is meant to expand the current system by expanding or altering the current functionality. This distinction is not strict, and it has more to do with the strategic objectives. Whilst the first approach may me more pragmatic in the sense of competition results, the former may be more aligned with the production of academic and research work. Ideally the development effort should as much as possible aim to serve both of this purposes. In the following suggestions I tried to accommodate both of this approaches.

Regarding the color segmentation topic, there is work that can be done to improve some of the computational performance parameters, such as time performance and CPU load. Most of the image processing steps in the segmentation algorithm are per pixel operations. These operations, namely the color conversion and the topological filtering stage, could take advantage of parallel programming processing. Using parallel computing platforms such as CUDA or OpenCL one could harness the power of the computer's GPU. This might result in a significant improvement in speed performance and perhaps more importantly decrease in CPU load, effectively eliminating the potential bottleneck caused by the image processing tasks. In the current implementation some of the OpenCV GPU module functions have been used, namely for the color conversion with very good results. However to take full advantage of the GPU a custom implementation would be necessary. Considering the good performance that is currently achieved it is debatable whether this would be worth the effort.

Regarding new functionality that can be implemented concerning the subject of the color segmentation, the immediate option would be to explore on the topic automatic color segmentation. Even though there is nothing wrong from a practical standpoint with the manually selected segmentation that is used, automatic color segmentation is an interesting problem and an active research topic. The automatic color segmentation methods could also be combined with the manual method. For example, one could use the automatic methods for the tracking of the manually selected color intervals in case of severe illumination changes. This means that besides being interesting from a research perspective, this could have a real-world impact on the robustness of the color segmentation. Common approaches to automatic color segmentation usually rely on the detection and tracking of modes in some chromaticity representation of the image. These could be used to adjust the HSV segmentation intervals in response to a dynamic illumination.

The detection system's purpose is to detect obstacles inside the playing field. In ideal conditions the playing field is completely surrounded by a small white wall that helps to isolate the playing field from the outside environment. Often times however, this wall is inexistent or incomplete or there are objects and clutter in the margins of the field (for example replacement or broken robots). This means that the environment outside the field lines may constitute a source of image noise which can result in false positive detections. One way to alleviate this problem could be to make the obstacle detection system aware of the robot's pose in the field by integrating information from the location system. The current localization system deployed by the robots uses a Monte Carlo Location algorithm, which has shown to work very well in practice. If the robot possesses a good estimate of its location in the playing field, and considering that the detection algorithm detects the contour of the robot in the ground plane, the search range can be narrowed in order to exclude anything outside the field. This would reduce the number of false positive detections, resulting in a more accurate and faster detection and tracking system.

One desirable addition to the detection system, would be the ability to detect the orientation of a detected obstacle, and by doing so obtain its full pose characterization. Generic pose recognition methods usually rely on the detection of some type of robust features intrinsic to the object. However MSL soccer robots usually have a very symmetrical chassis and don't possess very distinct orientation features. Consequently the typical feature detection methods perform poorly in this application. One solution would be to add identifiable features to the robot chassis that would be easily detected. However the robot with these added features must still comply with the MSL competition rules, which seriously limits our options. It is also important to bear in mind that due to the nature of the omnidirectional camera, image resolution diminishes very strongly with the distance from the camera, and distortion effects are very pronounced. The work to be developed would consist of devising some type of hardware features to be added to the robots and the respective detection software. These features should be compliant with MSL soccer rules and be easy to detect across a good distance range. A suggestion to approach this problem is to experiment with very simple shapes such as colored bars of varying length and variable separation so that even if only a few of this bars were detected one could still infer the orientation of the robot. Given that pose measurements would be most likely very noisy and somewhat unreliable, one could use a Kalman filter for the orientation angle alone or possibly extend the model of the dynamics of the target robot to include its full pose.

Quite often, during MSL matches many previously undetected problems arise. The manifestation of this problems may seriously undermine the team's performance and have a deleterious effect on the overall competitiveness. MSL robot teams are complex systems and it's easy for software bugs or flawed algorithms to go undetected during testing. Many times a flaw in one subsystem is only reveled when that subsystem interacts with another and the effects of the flaw are propagated. It is in this context that a comprehensive simulation platform becomes an invaluable tool, provided that it is able to reproduce the robotic systems and game environments with enough degree of realism. Such a platform would bring a new depth of insight

into the development status of the project, aiding in the early detection of potential problems, prioritization of work and the promotion of good development practices. One of the biggest challenges in the development of a simulation platform for MSL is the simulation of the robot's omnidirectional camera. The robot's omnidirectional camera is the main, if not the only, sensor the robot possesses. Not simulating it narrows the scope of the simulation mostly to high level decision making, reducing the worth of the simulation environment. Omnidirectional cameras like the ones the robots possess are non-standard, and most of the established robotics simulation frameworks typically don't offer simulation models for this type of camera, as they mostly only provide models for the common projective type cameras. The camera model that was adopted for this work and introduced in Chapter 4 is particularly suitable to be simulated. One could use the geometric model to generate a revolution surface whose profile would be given by the calibration and then use an orthographic camera to generate the image. Tests conducted in Blender 3D software showed promising results, the only problem being the render time required by the ray-traced reflections. The Blender 3D software has a lot of potential for the development of a robotics simulation platform as this software is open source, it is well document and has an active community. It is also highly configurable and scriptable via the integrated python interpreter. Additionally there is some functionally for dome projection already implemented in Blender [13][14], which could be used as a starting point for adaptation or inspiration.

The suggestions presented in this section are just that, mere suggestions not a comprehensive list off all the possibilities, but merely a few relatively small projects, that are realistically achievable, and that I believe would make useful contributions. One thing is certain, there is ample room for improvement and expansion, for this is a project of an open and flexible nature. The challenge most often lies in finding the optimal compromise between a cornucopia of exiting ideas and the inevitable constraints, both technical and logistical. Anyway, all of that contributes to the charm and enthusiasm of working in robotics. The kind of enthusiasm that I had the privilege to experience in the making of this modest work.

BLIBLIOGRAPHY

[1] Mark Ebner, “Color Constancy”, John Wiley & Sons, 2007.

[2] Theo Gevers, Arjan Gijsenij, Joost van de Weijer, Jan-Mark Geusebroek, “Color in Computer Vision: Fundamentals and Applications”, Wiley-Blackwell, 2012.

[3] D. Scaramuzza, A. Martinelli, R. Siegwart, “A Flexible Technique for Accurate Omnidirectional Camera Calibration and Structure from Motion”, Proceedings of IEEE International Conference of Vision Systems (ICVS'06), 2006.

[4] D. Scaramuzza, A. Martinelli, R. Siegwart, “A Toolbox for Easy Calibrating Omnidirectional Cameras”, Proceedings to IEEE International Conference on Intelligent Robots and Systems (IROS 2006), 2006.

[5] D. Scaramuzza, “Omnidirectional Vision: from Calibration to Robot Motion Estimation”, ETH Zurich, PhD Thesis no. 17635, 2008.

[6] Sebastian Thrun, Wolfram Burgard, Dieter Fox, “Probabilistic Robotics”, MIT Press, 2005.

[7] Roland Siegwart, Illah Reza Nourbakhsh, Davide Scaramuzza, “Introduction to Autonomous Mobile Robots”, 2nd ed. MIT Press, 2011.

[8] Richard Szeliski, “Computer Vision: Algorithms and Applications”, Springer-Verlag, 2011.

[9] Suzuki, S., Abe, K., “Topological Structural Analysis of Digitized Binary Images by Border Following”, CVGIP 30 1, pp 32-46, 1985.

[10] John J. Craig, “Introduction to Robotics: Mechanics and Control”, 3th ed. Prentice Hall, 2004.

[11] Subhash Challa, Mark R. Morelande, Darko Mušicki, Robin J. Evans, “Fundamental of Object Tracking”, Cambridge University Press, 2011.

[12] Gene F. Franklin, J. David Powell, Abbas Emami-Naeini, “Feedback Control of Dynamic Systems”, 5th ed. Pearson Prentice Hall, 2006.

[13] Paul Bourke,Spherical mirror (Mirrordome) – “A new approach to hemispherical dome projection”. Planetarian, Vol. 34 (4), pp 5-9, 2005.

[14] Paul Bourke, “Using a spherical mirror for projection into immersive environments (Mirrordome)”, Graphite (ACM Siggraph), Proceedings of the 3rd international conference on Computer graphics and interactive techniques in Australasia and South East Asia. pp 281-284, 2005.

[15] Rainer Burkard, Mauro Dell’Amico, Silvano Martello, “Assignment Problems”, 2nd ed., Society for Industrial and Applied Mathematics, 2012.

[16] David G. Lowe. “Distinctive image features from scale-invariant keypoints”, International Journal of Computer Vision, 60(2):91–110, 2004.

[17] Z. Zhang, “A flexible new technique for camera calibration”, IEEE Transactions on Pattern Analysis and Machine Intelligence, 22(11):1330–1334, 2000.

[18] João Silva, Nuno Lau, António, J. R. Neves, João Rodrigues, José Luís Azevedo, “Obstacle detection, identification and sharing on a robotics soccer team”, IEETA / Department of Electronics, Telecomunications and Informatics University of Aveiro, 2009.

[19] Naomi Henderson, Robert King, Richard H. Middleton, “An Application of Gaussian Mixtures: Colour Segmentation for the Four Legged League Using HSI Colour Space”, RoboCup 2007: Robot Soccer World Cup XI, Lecture Notes in Computer Science Volume 5001, pp 254-261, 2008.

[20] Xun Li, Huimin Lu, Dan Xiong, Hui Zhang, Zhiqiang Zheng, “A Survey on Visual Perception for Robocup MSL Soccer Robots”, International Journal of Advanced Robotic Systems, 2013.

[21] Raziel Alvarez, Erik Millán, Alejandro Aceves-López, Ricardo Swain-Oropeza, “Accurate Color Classification and Segmentation for Mobile Robots”, Mobile Robots: Perception & Navigation, Pro Literatur Verlag, pp 149-164, 2007.

[22] Richard Hartley, Andrew Zisserman, “Multiple View Geometry in Computer Vision”, 2nd ed., Cambridge University Press, 2004.

[23] Mark Nixon, Alberto Aguado, “Feature Extraction & Image Processing for Computer Vision”, 3rd ed., Academic Press, 2012.

[24] Tijn Schmits, Arnould Visser, “An Omnidirectional Camera Simulation for the USARSim World”, RoboCup 2008, LNAI 5399, pp 296-307, 2009.

[25] Greg Welch, Gary Bishop, “An Introduction to the Kalman Filter”, TR 95-041 Department of Computer Science University of North Carolina at Chapel Hill, 2006.

[26] Kabsch, Wolfgang, "A solution for the best rotation to relate two sets of vectors", *Acta Crystallographica* 32:922, 1976.

[27] Naomi Henderson, Robert King, Stephan K. Chalup, “An Automated Colour Calibration System using Multivariate Gaussian Mixtures to Segment HSI Colour Space”, In Proc. of the 2008 Australasian Conference on Robotics and Automation, 2008.

[28] João Messias, Aamir Ahmad, João Reis, Miguel Serafim, Pedro Lima, “SocRob 2013, Team Description Paper”, 2013.

[29] Fei Liu, Huimin Lu, Zhiqiang Zheng, “A Modified Color Look-Up Table Segmentation Method for Robot Soccer”, In Proceedings of the 4th IEEE LARS/COMRob, 2007.

[30] Xin Li, Kejun Wang, Wei Wang, Yang li, “A Multiple Object Tracking Method Using Kalman Filter”, In Proceedings of the 2010 IEEE International Conference on Information and Automation, 2010.

[31] Rob Farber, “CUDA Application Design and Development”, Morgan Kaufmann, 2011.

[32] Jason M. O’Kane, “A Gentle Introduction to ROS”, CreateSpace independent Publishing Platform, 2013.

[33] Zita Marinho, João Messias, “Multi-object Tracking Based Histogram Classifier and Kalman Filtering”, 2008.

[34] Robocup Wiki, [Online], Available: http://wiki.robocup.org/wiki/Middle_Size_League/

[35] Ros Wiki, , [Online], Available: http://www.ros.org/wiki/