

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO



**FEUP**

# **Calibration of Catadioptric Vision Systems**

**Tiago Raúl de Sousa Pereira**

Mestrado Integrado em Engenharia Electrotécnica e de Computadores

Orientador: Paulo José Cerqueira Gomes da Costa (Prof. Doutor)

Co-orientador: Héber Miguel Plácido Sobreira (Mestre)

July 2012



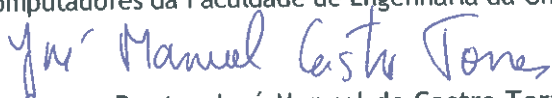
A Dissertação intitulada


“Calibration of Catadioptric Vision Systems”

foi aprovada em provas realizadas em 18-07-2012

o júri

  
Presidente Professor Doutor José Carlos dos Santos Alves  
Professor Associado do Departamento de Engenharia Electrotécnica e de  
Computadores da Faculdade de Engenharia da Universidade do Porto

  
Professor Doutor José Manuel de Castro Torres  
Professor Associado do Faculdade de Ciências e Tecnologia da Universidade  
Fernando Pessoa

  
Professor Doutor Paulo José Cerqueira Gomes da Costa  
Professor Auxiliar do Departamento de Engenharia Electrotécnica e de  
Computadores da Faculdade de Engenharia da Universidade do Porto

  
Mestre Héber Miguel Sobreira  
Investigador do INESC - TEC

O autor declara que a presente dissertação (ou relatório de projeto) é da sua exclusiva autoria e foi escrita sem qualquer apoio externo não explicitamente autorizado. Os resultados, ideias, parágrafos, ou outros extratos tomados de ou inspirados em trabalhos de outros autores, e demais referências bibliográficas usadas, são corretamente citados.

  
Autor - Tiago Raúl de Sousa Pereira



# Resumo

Os sistemas de visão catadióptrica são úteis em muitas aplicações, mas geralmente isso requer uma boa calibração realizada anteriormente.

Nesta tese propõe-se um método de calibração não paramétrico para qualquer tipo de sistema catadióptrico usado no futebol robótico, independentemente do tipo de espelho e sem considerar a restrição que impõe a existência de um único ponto de vista (bastante limitativa relativamente à precisão da montagem do sistema). O método criado usa o ambiente estruturado destes robôs para o processo de calibração, sem precisar de um padrão externo de calibração. Desta maneira, usa a informação, embora incompleta, que as linhas do campo fornecem, usando várias imagens para realizar a calibração, imagens essas adquiridas sobre certas condições específicas.

Portanto, o primeiro passo passa por extrair informação das linhas, o que é conseguido detectando as linhas indivisíveis que constituem as linhas do campo de futebol. Em seguida, identifica-se as linhas através de comparação da estrutura local de conexão das linhas com a estrutura de linhas de um campo, previamente conhecido. Assim, usando uma otimização linear e a conhecimento obtido com as linhas, estima-se os parâmetros da função polinomial que melhor se adapta ao mapeamento real entre pontos da imagem e coordenadas do mundo ao nível do chão.

Finalmente, também se criou uma simulação do sistema de visão, usando a propagação inversa dos raios de luz que atingem a câmara, usando-se um espelho hiperbólico como modelo do reflector de luz. Esta simulação é então usada para testar o processo de identificação de linhas, e também a calibração em geral. Além disso, isto também é utilizado para testar as diferentes possibilidades para a função a usar como mapeamento de pontos na calibração.



# Abstract

Catadioptric vision systems are useful for many applications, but it usually implies the system needs to be accurately calibrated.

In this thesis we propose a non parametric method to calibrate all kinds of catadioptric systems used in soccer robots, independently of the mirror shape, and without using the constraint of the existence of a single view-point. The application uses the structured environment of these robots for the calibration process, without needing any external calibration pattern to do the procedure. Therefore, it uses the incomplete information of soccer field lines and the combination of multiple images acquired under certain constraint to perform the calibration

So, the first step is the extraction of lines information of the image, which is accomplished by detecting the indivisible lines that form the whole soccer field lines. Posteriorly, we identify each line through the matching of the local structure of lines with the well known characteristics of lines structure in a soccer field. Then, using a linear optimization and the knowledge extracted from the lines, we estimate the parameters of a polynomial function in order to fit the mapping from pixel to ground coordinates.

We also created a catadioptric vision system simulation, using the backpropagation ray tracing method. We used the hyperbolic mirror model, which we use as basis for testing the line identification process, and the calibration process itself. Furthermore, we also use it for testing different possibilities of polynomial functions for the calibration.





# Acknowledgments

I would like to thank my supervisor, Prof. Dr. Paulo Costa, for all his explanations and suggestions during these project. I also want to thank Eng. Héber Sobreira for the fruitful discussions and help he provided.

Finally, I want to acknowledge the help and motivation of my family and friends.

Tiago Pereira



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Catadioptric Vision Systems . . . . .	1
1.2	Literature Review . . . . .	3
1.3	Motivation and Methodology . . . . .	8
1.4	Structure of Thesis . . . . .	10
<b>2</b>	<b>Vision System Simulation</b>	<b>11</b>
2.1	The Camera Model . . . . .	11
2.2	Mirror Model . . . . .	14
2.3	Coordinate systems transformation . . . . .	15
2.4	Back-Propagation Ray Tracing . . . . .	17
2.5	Results . . . . .	21
<b>3</b>	<b>Mapping Functions</b>	<b>23</b>
3.1	Polynomial Approximation using Cartesian Coordinates . . . . .	23
3.2	Polynomial Approximation using Polar Coordinates . . . . .	27
3.3	Usage of only lines . . . . .	33
<b>4</b>	<b>Field Lines Identification</b>	<b>37</b>
4.1	Lines Detection . . . . .	38
4.1.1	Main Method . . . . .	39
4.1.2	Post-processing Methods . . . . .	44
4.2	Lines Identification . . . . .	53
<b>5</b>	<b>Parameters Estimation</b>	<b>63</b>
5.1	Optimization process . . . . .	63
5.2	Results . . . . .	66
5.3	Calibration constraints considerations . . . . .	73
<b>6</b>	<b>Conclusions and Future Work</b>	<b>77</b>
6.1	Analysis and Review . . . . .	77
6.2	Future Work . . . . .	78
	<b>References</b>	<b>81</b>



# List of Figures

1.1	Comparison between catadioptric vision system. In (a) the system has a SVP, but in (b) there is no unique intersection for the rays. Image from [1]. . . . .	2
	(a) Central Sensor . . . . .	2
	(b) Non-Central Sensor . . . . .	2
1.2	This image shows the robot and the central part of the soccer field occupy a large part of the image captured . . . . .	2
1.3	Example of conic fitting methods to detect rectilinear lines. From [2] . . . . .	4
1.4	Example of calibration pattern used in [1] . . . . .	5
1.5	Example of checkerboard pattern used by [3] . . . . .	8
1.6	The catadioptric vision system setup used in the soccer robots. From [4] . . . . .	9
2.1	Graphical representation of the ideal pinhole camera model . . . . .	12
2.2	Representation of the mirror model, such as mirror pose and its coordinate system	14
2.3	Possible misalignments between the various components of the vision system . .	15
	(a) Misalignment between camera and mirror . . . . .	15
	(b) Misalignment of camera in relation to world reference . . . . .	15
2.4	Example of back-propagation ray tracing. As shown, from a pixel position and the vision system model, the ray is calculated, and the ground point that originated the ray is estimated. Image obtained from [4]. . . . .	17
2.5	Reflection of a ray in the mirror. Using the surface normal vector and the incident ray direction, the reflection ray direction is estimated. Image from [5] . . . . .	19
2.6	Comparison between different catadioptric vision system simulation. In (a) the system has a SVP, because there are no misalignments, and the ideal pinhole camera model is used (only with principal point offset). In (b) misalignments, pixel skewness and aspect ratio is simulated. . . . .	21
	(a) Ideal Sensor . . . . .	21
	(b) Non-Ideal Sensor . . . . .	21
2.7	Simulation of ideal catadioptric vision system with robot in a different field position.	21
3.1	Calibration results for the non-ideal sensor using a cartesian coordinates $5^{th}$ degree polynomial function. In (a) we represent the reverse mapping of points, and in (b) we represent the normalized calibration error. . . . .	26
	(a) Reverse mapping . . . . .	26
	(b) Normalized error . . . . .	26
3.2	Calibration results for the non-ideal sensor using a cartesian coordinates $10^{th}$ degree polynomial function. In (a) we represent the reverse mapping of points, and in (b) we represent the normalized calibration error. . . . .	27
	(a) Reverse mapping . . . . .	27

	(b) Normalized error . . . . .	27
3.3	Calibration results for the ideal sensor using a cartesian coordinates $5^{th}$ degree polynomial function. In (a) we represent the reverse mapping of points, and in (b) we represent the normalized calibration error. . . . .	27
	(a) Reverse mapping . . . . .	27
	(b) Normalized error . . . . .	27
3.4	Calibration results for the ideal sensor using a cartesian coordinates $10^{th}$ degree polynomial function. In (a) we represent the reverse mapping of points, and in (b) we represent the normalized calibration error. . . . .	28
	(a) Reverse mapping . . . . .	28
	(b) Normalized error . . . . .	28
3.5	Calibration results using various approaches: (a) uses a linear optimization, (b) a combination of one linear and one nonlinear optimizations, and (c) uses a multi-step combination of both linear and nonlinear parameter estimations, with repetition of steps 1 and 2 until no further improvement. It uses a five degree polynomial function. The optimization steps are presented in 3.3. . . . .	30
	(a) Linear Optimization (step 1) . . . . .	30
	(b) 2 step Optimization - steps 1 and 3 . . . . .	30
	(c) Multi-step Opt. (steps 1 to 3) . . . . .	30
3.6	Calibration results using various approaches: (a) uses a linear optimization, (b) a multi-step combination of linear and nonlinear optimizations. Step 3 is not used because, with a $11^{th}$ degree polynomial function, it takes a long time. The optimization steps are presented in 3.3. . . . .	30
	(a) Linear Optimization (step 1) . . . . .	30
	(b) Multi-step Opt. (repetition of steps 1 and 2) . . . . .	30
3.7	Calibration results for non-ideal sensor using a polar coordinate $5^{th}$ degree polynomial function. In (a) we represent the reverse mapping of points, and in (b) we represent the normalized calibration error. . . . .	31
	(a) Reverse mapping . . . . .	31
	(b) Normalized error . . . . .	31
3.8	Calibration results for non-ideal sensor using a polar coordinate $10^{th}$ degree polynomial function. In (a) we represent the reverse mapping of points, and in (b) we represent the normalized calibration error. . . . .	32
	(a) Reverse mapping . . . . .	32
	(b) Normalized error . . . . .	32
3.9	Calibration results for an ideal sensor image using a polar coordinate $5^{th}$ degree polynomial function. In (a) we represent the reverse mapping of points, and in (b) we represent the normalized calibration error. . . . .	32
	(a) Reverse mapping . . . . .	32
	(b) Normalized error . . . . .	32
3.10	Calibration results for an ideal sensor image using a polar coordinate $10^{th}$ degree polynomial function. In (a) we represent the reverse mapping of points, and in (b) we represent the normalized calibration error. . . . .	32
	(a) Reverse mapping . . . . .	32
	(b) Normalized error . . . . .	32

3.11	Comparison between calibration using only line points and using all points (inside a predetermined region), using a 5 degree polynomial function. The result is presented in the form of a reverse mapping of the calibration image. The color code for (a) is in table 3.2, and for (b) is in table 3.5. The test is done in the image used for calibration. . . . .	33
	(a) Reverse Mapping (calibration using all field points) . . . . .	33
	(b) Reverse Mapping (calibration using only field lines) . . . . .	33
3.12	Comparison between calibration using only line points and using all points (inside a predetermined region), using a 5 degree polynomial function. The result is presented in the form of a reverse mapping of the calibration image. The color code for (a) is in table 3.2, and for (b) is in table 3.5. The test is done using an image different from the calibration image. . . . .	34
	(a) Reverse Mapping (calibration using all field points) . . . . .	34
	(b) Reverse Mapping (calibration using only field lines of other image) . . . . .	34
3.13	Comparison between calibration using only line points and using all points (inside a predetermined region), using a 5 degree polynomial function. The result is presented using the calibration error of each one. . . . .	34
	(a) Using all field points . . . . .	34
	(b) Using only field lines . . . . .	34
3.14	Comparison between calibration using only line points and using all points (inside a predetermined region), using a 10 degree polynomial function. The result is presented in the form of a reverse mapping of the calibration image. The color code for (a) is in table 3.2, and for (b) is in table 3.5. The test is done in the image used for calibration. . . . .	35
	(a) Reverse Mapping (calibration using all field points) . . . . .	35
	(b) Reverse Mapping (calibration using only field lines) . . . . .	35
3.15	Comparison between calibration using only line points and using all points (inside a predetermined region), using a 10 degree polynomial function. The result is presented in the form of a reverse mapping of the calibration image. The color code for (a) is in table 3.2, and for (b) is in table 3.5. The test is done using an image different from the calibration image. . . . .	36
	(a) Reverse Mapping (calibration using all field points) . . . . .	36
	(b) Reverse Mapping (calibration using only field lines of other image) . . . . .	36
3.16	Comparison between calibration using only line points and using all points (inside a predetermined region), using a 10 degree polynomial function. The result is presented using the calibration error of each one. . . . .	36
	(a) Using all field points . . . . .	36
	(b) Using only field lines . . . . .	36
4.1	Example of a soccer field and the result of lines detection and classification. . . . .	37
	(a) Soccer field . . . . .	37
	(b) Soccer field lines identified . . . . .	37
4.2	The soccer field lines. . . . .	38
4.3	Corner detection using the Harris method. . . . .	39
4.4	Meta-code for the main method of lines detection . . . . .	40
4.5	Example of line tracking used in line detection. . . . .	40
	(a) First Step . . . . .	40
	(b) Second Step . . . . .	40
	(c) Third Step . . . . .	40

4.6	Behavior of main line detection method - line tracking - when passing by a corner (lines intersection). It is shown the line tracking tries to maintain the actual line direction. . . . .	41
	(a) Before corner . . . . .	41
	(b) After corner . . . . .	41
4.7	Result of lines intersection when following a line . . . . .	41
	(a) First step . . . . .	41
	(b) Second step . . . . .	41
	(c) Third step . . . . .	41
	(d) Fourth step . . . . .	41
4.8	Example of situation in which a corner would be wrongly detected. . . . .	42
4.9	Meta-code for the detection of dashed lines . . . . .	43
4.10	The result of applying the main method of line detection to a soccer field image . . . . .	44
4.11	Merging multiple lines into one . . . . .	45
	(a) Before Merging . . . . .	45
	(b) After Merging . . . . .	45
4.12	Meta-code for the extension of already detected lines . . . . .	45
4.13	Meta-code for the elimination of lines mainly composed of invisible points . . . . .	45
4.14	Elimination of lines with almost all points invisible . . . . .	46
	(a) Before Lines Elimination . . . . .	46
	(b) After Lines Elimination . . . . .	46
4.15	Meta-code for the creation of lines using checked but unclassified pixels . . . . .	46
4.16	Creation of lines from unclassified pixels . . . . .	47
	(a) Before creating new lines . . . . .	47
	(b) After creating new lines . . . . .	47
4.17	Meta-code for the post-processing methods used in line detection . . . . .	48
4.18	Meta-code for the detection of corners in already detected lines . . . . .	49
4.19	Detection of corner in middle of line, for cases in which the corner position is clear . . . . .	50
	(a) Before corner detection . . . . .	50
	(b) After corner detection . . . . .	50
4.20	Detection of corner in middle of line, for cases in which the corner position is unclear . . . . .	51
	(a) Before corner detection . . . . .	51
	(b) After corner detection . . . . .	51
4.21	Meta-code for merging curvilinear (center circle) lines, wrongly separated in two . . . . .	52
4.22	The result of applying the complete line detection procedure to a soccer field image . . . . .	53
4.23	Meta-code for identification of the detected field lines . . . . .	55
4.24	This shows how we can detect the center circle lines, using the inner product of lines' extremities local direction. For curvilinear lines the inner product is maximum (V1 and V2), for rectilinear lines it is minimum (V3 and V4). . . . .	56
4.25	Due to problems in the lines thinning process, extremities of center circle lines may not intersect each other. The nearest corners are found, and the line connecting them is considered a corner (red pixels) after lines identification. . . . .	56
	(a) Before Identification . . . . .	56
	(b) After Identification . . . . .	56



4.26	This shows how to identify each extremity of Line 1 (line is identified, corners are not). Using the line's extremities local direction (V1 and V2), we calculate the average and use it as an estimation of the x axis orientation. Then we use the cross product of estimated x and V3 (estimated y axis) to determine z axis direction. If the result is not the expected, we know V3 is inverted, then clarifying which corner (1 or 2) corresponds to each extremity of Line 1. . . . .	57
4.27	Meta-code identification of penalty and goal area lines and corners . . . . .	58
4.28	Using the already identified corners C1 and C2, and the ordered 3-line corners, we can calculate V1 and V2. Using their average, we use the inner product of that average with the estimated x axis direction (4.26). Analyzing the signal of the result obtained, we can distinguish the side we are dealing with. . . . .	59
4.29	For identification of penalty and goal areas, we detect corners a, b, c and d. Then we differentiate them using corners a and d, and checking if they are C1 and C4 ((a)), or C2 and C3 ((b)) . . . . .	60
	(a) Identification of penalty area . . . . .	60
	(b) Identification of goal area . . . . .	60
4.30	Result of lines identification using non-ideal sensor in (a). If compared with (b), we see the colors are the same, meaning all lines are well classified. . . . .	61
	(a) After lines identification . . . . .	61
	(b) Soccer field lines model . . . . .	61
4.31	Result of lines identification. If compared with 4.1b, we see the colors are the same, meaning all lines are well classified. . . . .	61
	(a) Robot aligned with x axis (world reference) . . . . .	61
	(b) Robot aligned with y axis (world reference) . . . . .	61
4.32	Result of lines identification. If compared with 4.1b, we see the colors are the same when lines were identified (not white), but sometimes not all lines are identified. . . . .	62
	(a) Image with some lines not identified . . . . .	62
	(b) Image with some lines not identified . . . . .	62
5.1	Calibration using only one image, a second degree polynomial for fitting, and the same image for testing . . . . .	70
5.2	Calibration using one image, a fitting polynomial function of degree 5, and testing the calibration error using: the calibration image for the testing (a), and a different image for the validation (b). . . . .	70
	(a) Reprojection of test-image, which is the one used in calibration . . . . .	70
	(b) Reprojection, with different images being used for calibration and validation . . . . .	70
5.3	Calibration error when using two calibration images (robot in the central field position, with orientation equal do 0 and 90 degrees.), and a fifth degree polynomial . . . . .	71
5.4	Calibration using the two images set, a polynomial of degree 5, and testing it with two images different from the ones used for calibration. . . . .	71
	(a) Reprojection of test-image different from calibration images, but with robot position in the field similar to the one in calibration images . . . . .	71
	(b) Reprojection of test-image with robot position far from the position of the robot in the calibration images . . . . .	71
5.5	Calibration using 39 images, a polynomial of degree 9, and a test image not used in calibration. . . . .	72
	(a) Test-image reprojection (image is not any of the 39 images used in calibration) . . . . .	72
	(b) Calibration Error . . . . .	72

5.6	Example of virtual aid in the image, that could allow the implementation of the calibration process, considering the restrictions of this method . . . . .	75
6.1	Example of lines extraction using color segmentation in (a), as presented in [6]. In (b) it is possible to observe the difficult line extraction problem in this images, because further lines almost do not have white pixels. . . . .	79
	(a) Line extraction using color segmentation . . . . .	79
	(b) Image showing difficulty in extraction of lines far from the robot . . . . .	79

# List of Tables

3.1	Fitting error of polynomial function estimation (cartesian coordinates) . . . . .	25
3.2	Color code of image reprojection (calibration using both field lines and non field lines points) . . . . .	26
3.3	Multi-step optimization . . . . .	29
3.4	Fitting error of polynomial function estimation (polar coordinates) . . . . .	31
3.5	Color code of image reprojection (calibration using only field lines) . . . . .	33
5.1	Calibration error of all points using polynomial function (cartesian coordinates) with various polynomial degrees and two sets of calibration images . . . . .	67
5.2	Error of calibration points using polynomial function (cartesian coordinates) with various polynomial degrees and two sets of calibration images . . . . .	68
5.3	Color code of image reprojection . . . . .	69
5.4	Running times for the various constituent parts of the calibration process . . . . .	73
5.5	Running times for the optimization process considering 2 and 39 image sets, and varying polynomial degrees . . . . .	73



# Acronyms and Symbols

## List of Acronyms

FEUP	Faculdade de Engenharia da Universidade do Porto
SVP	Single View-Point

## List of Symbols

$u$	first coordinate of the image plane
$v$	second coordinate of the image plane
$\lambda$	parameter of camera model describing the correspondence of one image point to infinite points in the camera reference
$f$	focal length of camera
$x_c$	$x$ coordinate on the camera reference
$y_c$	$y$ coordinate on the camera reference
$z_c$	$z$ coordinate on the camera reference
$u_0$	first coordinate of principal point
$v_0$	second coordinate of principal point
$\alpha$	skewness of pixels
$\lambda_u$	ray without considering distortion
$\lambda_x$	$x$ coordinate of ray in the camera reference without considering distortion
$\lambda_y$	$y$ coordinate of ray in the camera reference without considering distortion
$\lambda_c$	distortion center
$\lambda_d$	ray after considering distortion
$x_m$	$x$ coordinate on the mirror reference
$y_m$	$y$ coordinate on the mirror reference
$z_m$	$z$ coordinate on the mirror reference
$a$	mirror parameter
$b$	mirror parameter
$R_m^c$	mirror-to-camera rotational transformation
$\theta_m$	angle of mirror rotation along $y$ camera axis
$\psi_m$	angle of mirror rotation along $x$ camera axis
$T_m$	distance between camera and mirror references, expressed in camera coordinate system
$H_m^c$	mirror-to-camera homogeneous transformation
$R_c^m$	camera-to-mirror rotational transformation
$H_c^m$	camera-to-mirror homogeneous transformation
$R_c^w$	camera-to-world rotational transformation

$\theta_c$	angle of camera rotation along y world axis
$\psi_c$	angle of camera rotation along x world axis
$\phi_c$	angle of camera rotation along z world axis
$T_c$	distance between camera and world references, expressed in world coordinate system
$H_c^w$	camera-to-world homogeneous transformation
$\lambda_{cm}$	ray points in camera reference
$\lambda_m$	ray points in mirror reference
$\lambda'_m$	ray orientation in mirror reference
$\lambda''_m$	camera optical center in mirror reference
$\vec{N}$	surface normal vector in mirror reference
$\vec{r}_{im}$	incident ray orientation in mirror reference
$\vec{r}_{rm}$	reflected ray orientation in mirror reference
$\vec{r}_{rc}$	reflected ray orientation in coordinate system of camera
$\vec{r}_{rw}$	reflected ray orientation in world reference
$P_w$	ray intersection with mirror in world coordinates
$x_{fw}$	x coordinate of ray intersection with ground in world reference
$y_{fw}$	y coordinate of ray intersection with ground in world reference

# Chapter 1

## Introduction

In this introductory chapter we present and explain the various types of catadioptric vision systems, also introducing the most recent solutions for their calibration. Furthermore, we show the main reasons for the development of this thesis and the approach used to calibrate the catadioptric vision system used in the robot soccer team at FEUP. Finally, we provide an outline of the various parts that constitute this work.

### 1.1 Catadioptric Vision Systems

A catadioptric vision system consists of a combination of mirrors with conventional cameras to enhance the sensor field of view. The most common mirror types are the parabolic, hyperbolic, elliptical, conic, fisheye, and planar, although any other kind of mirror can be used. The camera usually points towards the mirror, reflecting the environment in the opposite direction of the camera.

As stated in [1], the catadioptric vision system can be classified in two categories: single view-point (or central) imaging system, or non-SVP (non-central) imaging systems. In the first case, lines of incoming rays intersect at a single point, and are reflected to the image plane (Figure 1.1). The only mirror shapes with this property are hyperboloidal, ellipsoidal and planar mirrors (coupled with perspective cameras) and paraboloid mirrors (coupled with an orthographic camera) [7]. With SVP imaging systems it is possible to obtain a mapping of every scene point on the image plane. On the other hand, non-SVP sensors can overcome the alignment constraint, without limitations regarding mirror shape and position. One can even use multi-part mirrors and single-camera stereo-vision. However, images taken from these systems cannot be rectified, and there is no exact closed-form function that can transform an omnidirectional image into a perspective one.

Although omnidirectional vision gives a better perception of all major objects in the robots surrounding area, there is a higher resolution degradation with growing distances away from the robot [8, 5]. One disadvantage of omnidirectional cameras is then badly distributed spatial resolution: the camera and robot themselves usually occupy a big part of the image, leaving only a

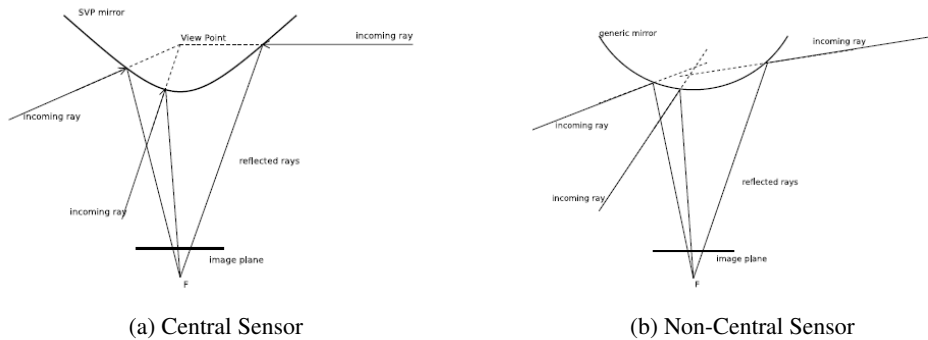


Figure 1.1: Comparison between catadioptric vision system. In (a) the system has a SVP, but in (b) there is no unique intersection for the rays. Image from [1].

small space in the peripheral part for important surrounding environment [1], which is shown in Figure 1.2.

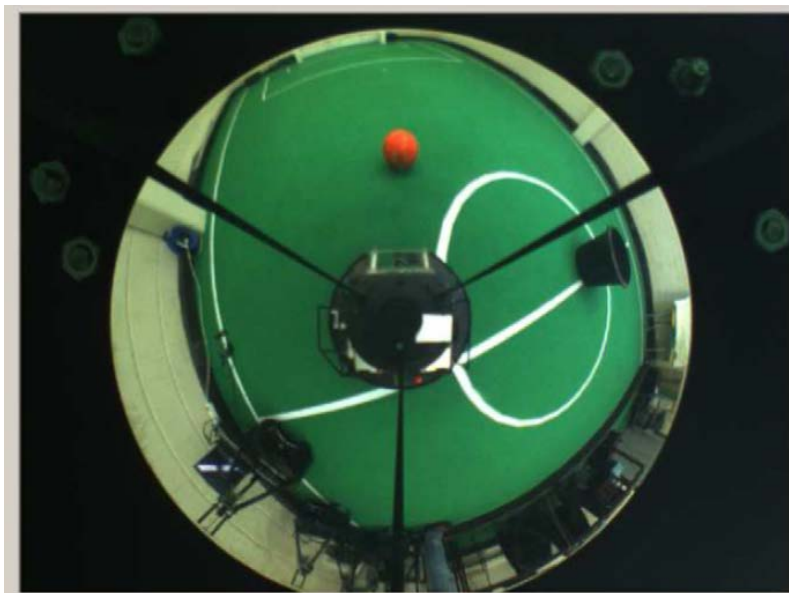


Figure 1.2: This image shows the robot and the central part of the soccer field occupy a large part of the image captured

In [9] they show the potential of omnidirectional catadioptric vision systems for mobile robots moving within structured environments, which directly applies to soccer robots, the main focus of this work. It is shown the advantages of this vision systems for self-localization.

As asserted in [5], although alignment between camera and mirror is desired given its useful optical properties, such as SVP, equi-resolution, and equi-areal, it is difficult to obtain because alignment correlates with camera intrinsic parameters, or even because of a simple mechanical setup. Nevertheless, most camera calibration methods assume SVP.

Catadioptric camera systems can be used in many applications, including but not limited to security systems, surveillance, robot navigation, 3D reconstruction, teleconferencing, advertising



[10, 1, 2]. However, for practical applications, this vision system requires a non-linear transformation to map the image points onto real world coordinates. Therefore, accurate calibration is necessary for extracting metric information from the images. The calibration may be used in many kind of systems, and tries to create direct or indirect mappings from image pixels to real world coordinates.

## 1.2 Literature Review

In this section, we discuss the most recent developments of catadioptric systems calibration. There are various approaches that differ mainly by the type of mirror, the projection model used (skewness, alignment errors), the information that is already known beforehand (the mirror and/or the camera intrinsic parameters) and the method used (auto-calibration, grids,...). These are generally parametric approaches, but there are also some non-parametric methods that do not try to estimate camera and mirror parameters, but only a mapping function between image points and world points. [11]

The main research focus of literature on this subject is based on parametric calibration, which tries to extract parameters of models that describe the system, such as camera and mirror models, and matrices describing the relative positions between the various coordinate references - camera, mirror, and world coordinate system. Using this methodology, we can simplify the problem by assuming it is a SVP system, which states camera and mirror must be aligned, therefore eliminating the camera to mirror coordinates transformation, and limiting the degrees of freedom of the camera to world transformation, as the camera must be perpendicular to the ground. However, SVP methods assume a precise assembly of the setup.

On the other hand, non-parametric calibration methods do not assume any specific model of camera, mirror, and misalignments. With these methods we only estimate the parameters of a certain function that best describes the pixels to world points mapping, minimizing the fitting error.

Parametric methods are the most commonly found, as they find parameters that, when well estimated, allow a good calibration of all image regions, because distortions are implicit to the model. Knowing the intrinsic parameters - mirror pose with reference to the camera, mirror shape, and camera model parameters - and the extrinsic parameters of the catadioptric system - the camera pose in the world reference frame - it is possible to map every pixel to a projection ray in the world.

On the other hand, non-parametric estimations usually perform worse, not being able to make good estimates of all image regions. One example is polynomial approximations, which are often valid only locally and badly approximate the projection around the edges of the mirror [11]. Nevertheless, parametric techniques are strongly dependent on the omnidirectional sensor model they use, while non-parametric approaches usually work with any kind of mirror and sensor.

Although most methods use a calibration pattern, as a grid or checkerboard, there are also self-calibration methods that require no calibration pattern, nor any *a priori* knowledge about the scene, only assuming the capability to automatically find point correspondences in a set of images

of the same scene. These methods may suffer in case of tracking difficulties and of a small number of features points. [3]

The calibration of SVP systems is a well known and studied problem. On the other hand, calibration of non-SVP sensors is more difficult to find in the scientific literature, and often solves only part of the problem, with some common assumptions, as an *a priori* camera calibration, or known mirror shape. [1]

Regarding the SVP constrained approaches, in [2], a unifying theory for central projection systems calibration is presented. It provides a model that maps points in 3D world in the image plane, and it can describe various kinds of SVP vision systems, such as the combination of an orthographic camera with a parabolic mirror, or a perspective camera with hyperbolic, elliptical or planar mirrors. This model is fully described in [12], where the equivalence between catadioptric and spherical projections is shown. Only systems with the SVP constraint can be used, and so the camera and mirror axis must be aligned in the case of orthographic cameras, or the center of the camera must be coincident with the outer focus of the mirror in the other cases.

Considering this, they derive several invariant properties of catadioptric line images. Lines are projected to conic curves, and then they use three or more line images, which are determined using conic fitting techniques, to calibrate the system, recovering the calibration parameters of the model they presented. These are camera intrinsic parameters: relative pose between camera and mirror, and the type of system. After experimental tests, they conclude curve estimation tends to fail due to only small arc being visible in the image, which may affect the overall calibration process as it is the starting point of the procedure.



Figure 1.3: Example of conic fitting methods to detect rectilinear lines. From [2]

Another approach is proposed in [5], where a calibration process for misaligned catadioptric systems, in other words, non-SVP systems, is discussed, which also has the advantage of not using nonlinear optimization. It proposes a calibration method for systems composed of a surface

of revolution mirror and a perspective camera, giving complete freedom to their relative position. The calibration process itself is based on the mirror boundary, and copes with misalignment considering separate models for the camera and the mirror. The correspondence between points in the different coordinate systems is calculated using mirror posture parameters, and reflection is calculated using ray tracing. It uses a common pinhole model for the camera to transform points in the image coordinate system into the camera coordinate system. To estimate the mirror posture, they use a method based on conic fitting along the ellipse in the image made from the mirror boundary. For that, they assume the camera is already calibrated and the radius of the mirror boundary is known. Using this method they obtain four solutions, and then they choose one with a method that uses rays from pixels far from the camera, which are used because with those ones differences of viewpoints can be ignored.

In [1], another method to calibrate non-SVP sensors is proposed. They also use the mirror border and a set of points to determine the mirror pose with respect to the camera, as well as the camera pose with respect to a world reference. In this case they assume the camera is already calibrated and that the mirror shape is known. For estimating the mirror localization they use the circle-ellipse correspondence due to the projection of a circle with known radius - mirror border - on the image plane. This method calculates the mirror position with ambiguity, which is solved only when estimating the extrinsic parameters (camera position in the world coordinate frame). They then calculate these last parameters using a set of points with known coordinates in both image and world coordinate frame and use ray tracing to calculate the reflected interpretation lines of those points. After that, one knows the position of these lines in both the world reference and camera reference, so one can compute the transformation between the two coordinate systems. Due to the orthonormality constraint of the rotational matrix, this becomes a non-linear system of equations. The intrinsic and extrinsic parameters estimation is done separately, and so the errors of the first one are propagated to the second estimation.

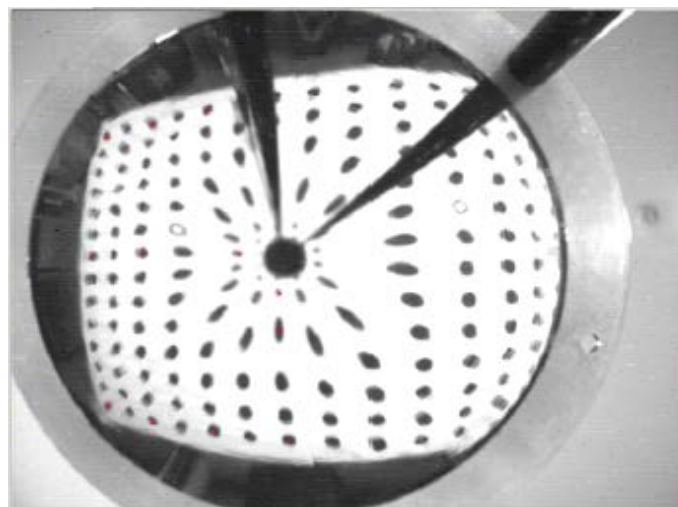


Figure 1.4: Example of calibration pattern used in [1]

In [8], a distance map around the robot's center is calculated using a back-propagation ray-tracing approach, with compensation for possible misalignments. In this case the mirror shape is already known. It is also stated that a defocus blur problem does not exist due to low spatial resolution at higher distances, so the main problem is the low-resolution itself. For the camera, they use the pinhole model, disregarding distortion of the lens. In the beginning some assumptions are used, such as the CCD axis being aligned with the axis of world coordinate system. Using the backpropagation ray tracing methodology, they can calculate the ray corresponding to each pixel, its intersection with the mirror is calculated, then obtaining the reflection ray, and finally the intersection with the ground as well. Furthermore, they generalize this approach, by withdrawing some of the assumptions initially used; now the CCD does not need to be aligned with the robot's coordinate axis, and the axis of revolution is not considered perpendicular to the ground anymore. To accomplish this, they assume a translation offset of the CCD center point, and three rotation angles applied to the three axis of the sensor, giving it complete freedom of position. The same approach is used to simulate the mirror misalignment, with the difference being that only two rotations are applied to the axis of the mirror. This is due to the fact that a rotation along its axis of revolution has no overall effect in the resulting image acquired by the system.

To get some of the parameters needed to construct a distance map, they measured it from the setup itself - the soccer robot - while others were extracted through algorithmic analysis of the image. The main image features used were the mirror outer rim diameter and eccentricity, the center of the mirror image, center of robot image, and both radii, as well as the distance and eccentricity of the game field lines. They also state the extracted parameters can be non-optimal due to the degradation of resolution with distance and its effect on feature extraction fidelity. To overcome this problem, they provide image feedback tools with which is possible to manually and interactively adjust the misalignment parameters.

Considering self-calibration methods, there is [10], in which they use omnidirectional images as input to a stereo algorithm in order to reconstruct the surrounding environment. They can obtain all the intrinsic pinhole camera model parameters, and the mirror parameters. However, the mirror must be paraboloid.

They first present a method that uses the mirror boundary to calibrate the system, hence only needing one image to execute this method. The method assumes the boundary is a circle, which means it is specially suitable for SVP vision systems. After fitting the circle to the boundary, the image principal point is determined as being its center. Then, using the boundary radius and the system known field of view, they calculate the remaining mirror parameter.

After that, they propose a second method using point feature tracks across an omnidirectional image sequence, and using consistency of pairwise correspondence, which satisfies the epipolar constraint. First, it generates point tracks, using relatively short sequences, otherwise tracking would be more difficult due to the highly distortion in catadioptric systems. Moreover, they use the least-median error metric to estimate the proposed unknown parameters, minimizing an objective function which can be calculated using the algebraic error metric or the image error metric, the latter having better results. The error relates to the epipolar constraint, and the estimation uses

the non-linear process with initial values being the ones obtained with the circle based technique - mirror parameter and principal point - and setting the initial values of aspect ratio and skew as one and zero, respectively.

Paper [11] presents an approach to calibrate omnidirectional vision systems using planar grids, generalizing the typical method using grids to calibrate perspective cameras. With this technique, four points need to be selected for each calibration grid by the user. The method makes use of the unified projection model [13, 12]. In this method, the camera and mirror are not considered separately, but as a global device. Moreover, an extra distortion function is added to model the misalignment between mirror and camera and the imperfection of the lens shape, which translates in radial and tangential distortions. They use a non-linear minimization to decrease the distance between the projection of the grid and the extracted values in the image. Therefore, it needs to have a good set of initial values of the parameters. For this, they start by ignoring the extra distortions and assuming aspect pixel ratio equal to 1 and skewness equal to zero. To initialize the principal point they use either the image center or the center of the mirror border, which is considered to be a circle. They then estimate the focal length using at least three image points belonging to a non-radial line image. Furthermore, the extrinsic parameters are estimated using four points of a grid of known size.

In [14] a parametric approach is also used, modeling the various elements of the vision system and their relative positions. However, this method is restricted to conic mirrors.

It is also possible to use a model with complete freedom regarding the relative position between camera and mirror, which is determined through nonlinear optimization. [15]

Moreover, [16] uses a large dot calibration pattern that recovers mirror parameters, and principal point.

Another method was proposed in [17], which uses a homography-based calibration. However, it assumes a parabolic mirror and an already calibrated camera, considering the intrinsic parameters of the camera are known *a priori*. Therefore, it just estimates mirror parameters and its position.

In [3], they propose a method to calibrate SVP systems, using a planar pattern located in different positions. This is a non-parametric method, which assumes the image projection function can be described by a Taylor series expansion. The coefficients are calculated using a two-step least squares linear minimization. This method assumes the circular external boundary of the mirror is visible in the image.

Both [3, 18] use a non-parametric calibration, which creates an association of a projection ray in 3D world to every pixel in an image. This is a highly generic calibration method, and so a specific model of the sensor is not required. [18] uses multiple images of overlapping calibration grids to obtain an initial calibration.

Another approach is proposed in [19], which considers the mapping of 2D image pixels to 3D light rays, creating a generic imaging model that can be used to calibrate many types of vision system using only one algorithm.



Figure 1.5: Example of checkerboard pattern used by [3]

Finally, we must consider the optimization technique used to estimate the parameters that describe the system (either parameters of the system model, or parameters of the mapping function). We can separate it in two groups again: nonlinear and linear optimization. Although nonlinear methods are more flexible, they have the disadvantage that the resulting accuracy is strongly dependent on the initial values, as the solution obtained might be a local minimum.

The fact that the mapping from pixel to ground coordinates may have some image line in which ground points are at infinity shows the polynomial approximation will never perfectly match the true mapping. Therefore, we can only hope to estimate the central region of the image, the most extreme points being the view of soccer field corners with the robot in the opposite corners. Any distances longer than that are not necessary, because localization algorithms only will use points as far as those are, and so calibrating the rest of the mirror is not necessary, or even possible. If, instead of mapping the image pixels to ground coordinates, we mapped them to the ray vectors at each point, the problem would not have such a strong non-linearity, making it possible to better calibrate the peripheral regions of the image.

### 1.3 Motivation and Methodology

As explained previously, omnidirectional vision system are used in soccer robots due to its possible 360 degrees field of view. Although the resulting images are strongly distorted, its wide field of view is an important feature for self-localization in robots, because it can get an overall perception of its surrounding environment. When matching the image obtained through the vision system with a previously known map of the surrounding scenario, the probability of correctly estimating its position can be increased with the knowledge of what is around the robot. Besides, it gives



another important advantage: being able to see what is behind itself, which may prove to be important for the soccer applications. Nevertheless, in order for self-localization to work properly, these systems need to be properly calibrated, especially considering the nonlinearities introduced by catadioptric vision systems.

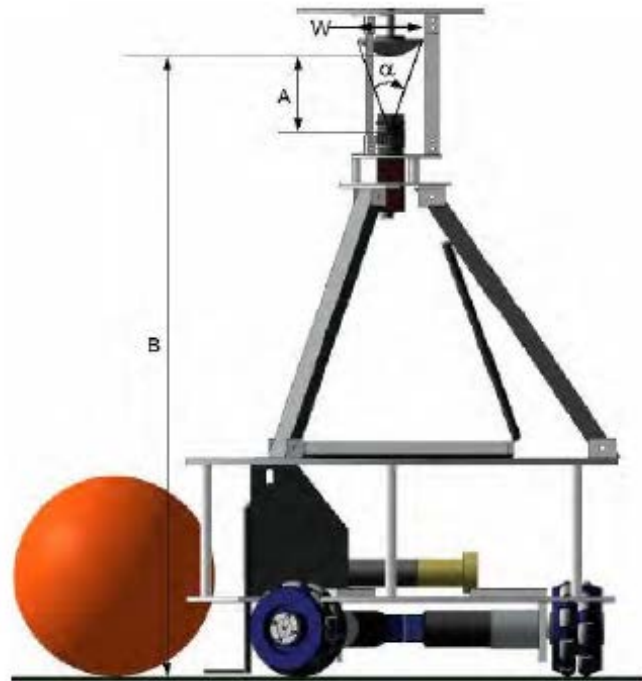


Figure 1.6: The catadioptric vision system setup used in the soccer robots. From [4]

Therefore, with this work we intend to study some calibration processes, in order to improve the localization of soccer robots. Thus, an accurate calibration is essential for self localization to work properly. In this project we do not concern ourselves with color balance, assuming that the colors are previously calibrated.

Usually, a checkerboard pattern is used to calibrate the system, as it is easy to obtain many points in the world referential whose position is known. However, in this specific case we try to calibrate the system only using the lines of the soccer field where the robot is located, because it introduces some advantages useful in this scenario. For example, these robots are subject to collision with other robots that could cause slight changes in the mechanical structure, thus negatively affecting the quality of the previous calibration given the new mechanical state of the robot. Given these conditions, the use of field lines allows the creation of a calibration process that does not need any exterior setup.

The ultimate application of this master thesis is the soccer robot used at FEUP, and so we aim to create a calibration process for the camera-mirror vision system of these robots. The setup currently uses an hyperbolic mirror, but this work is extendable to all kinds of mirrors.

We use a mapping function to calibrate the system, instead of a process to determine all the intrinsic and extrinsic parameters of both camera and mirror, because the first gives the possibility

of changing the type of camera and mirrors while still being able to use this calibration process. Moreover, for the soccer application, we need calibration to obtain a mapping between pixels and world points (ground), used in the localization process. So the non-parametric approach is much more suitable, because it finds the parameters of a function that best fits that mapping. In this particular case, the intrinsic parameters of camera, mirror, and their relative positions are irrelevant. However, using this approach, we also need to find the type of function that best fits the mapping, and the calibration process itself, namely its constraints. Furthermore, the parametric method have been the major focus of research, and many viable solutions are already available.

Lastly, we need to choose the optimization method that retrieves the desired parameters from the calibration data, as it can have a great influence in the overall result of calibration. The optimization can be linear or non-linear. This choice is highly dependent on the calibration procedure and data extraction.

In conclusion, robotic soccer is the main application of this algorithm, and so we propose a calibration process that uses the domain area of these robots to calibrate the vision system. Therefore, instead of using a non-soccer calibration pattern, such as a checkerboard, we use the soccer field, more precisely, the soccer field lines, to calibrate the system. For this purpose, we chose the non-parametric approach for our calibration process. Therefore, to accomplish that, we need to first get calibration images from robot's vision system. Then we extract field lines (using color segmentation) and identify its structure, to retrieve information from the lines position. Finally, we apply an optimization method that fits a function to the pixels to world coordinates mapping.

In all the work we use *Matlab* as a programming language. This choice was motivated by the language flexibility and all the toolboxes already available. In particular, it already has implemented functions to deal with image processing, and various linear and nonlinear optimization methods, what makes it especially suitable for this thesis.

## 1.4 Structure of Thesis

The next chapter describes a simulation of general catadioptric vision systems, explaining how we modeled each part of the system.

In chapter 3, we study various kinds of functions that can be used in non-parametric calibration, testing the fitting properties of each one on the simulation.

The fourth chapter covers the main methods developed to extract information from the field lines. We also describe the line extraction methods, that separate the field lines in indivisible lines, and the identification model that matches each extracted line with the correspondent line of a model.

In the following chapter we describe the calibration process used, specifically the optimization method used, considering the information retrieved from field lines. We show the results obtained with the proposed calibration method.

Finally, chapter 6 is dedicated to the main conclusions of the obtained results, as well as the future work.



## Chapter 2

# Vision System Simulation

We started by building a simulation, because with that it is easier to study the system and pre-validate the solution obtained. Although in the real process the correspondence between a pixel in the image and a point in the world is not known, with the simulation we can use it to validate calibration, even without using that information in the calibration process itself. So, we start by giving the parameters of the vision system, such as intrinsic parameters of camera, transformation matrices from camera to mirror referential and also between camera and world referential.

Then, we simulate the image obtained with the robot in different positions in the soccer field. Although we know the mapping from image points to world points, we won't use it in the calibration process. Given the image obtained, we proceed to line extraction, and then line detection. After that, we use the calibration process to obtain the intended mapping and then we compare it with the known original mapping to calculate the error of the calibration process.

The simulation implements all the misalignment possibilities, using a generic hyperboloid function to describe the mirror shape, and a pinhole model for the camera that models focal distance, aspect ratio, pixel skewness and image principal point. We can adapt the generic model to a specific setup by changing the values of the described parameters.

We did not implement the radial and tangential lens distortions, as it would imply solving an iterative process for each image pixel, making the simulation extremely slow. However, this is not considered to introduce a relevant difference in the results shown in the following sections, regarding the calibration procedure. We assume this because the mirror already introduces a high degree of distortion/non-linearity. Given that, the camera lens distortion is negligible, unlike common vision system in which the lens is the main cause of image distortion.

We use the assumption that the mirror surface is a surface of revolution, which means its position and orientation only has five degrees of freedom in relation to the camera referential.

### 2.1 The Camera Model

To describe the camera we use the common pinhole model, which describes the mathematical relationship between the coordinates of a 3D point and its projection onto the image plane of an

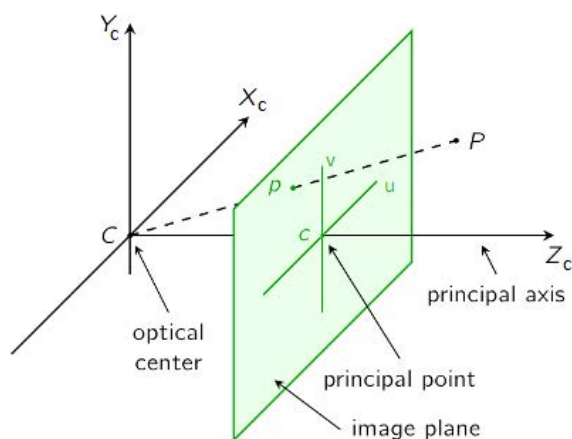


Figure 2.1: Graphical representation of the ideal pinhole camera model

ideal pinhole camera. This is a simple model that does not include the lenses, therefore not considering their effects, such as geometric distortions or blurring. However, to increase the validity of this model, we can still add to the simulation the Brown distortion equations, therefore taking into account the geometric distortions. Applying also some suitable coordinate transformations on the image coordinates, and neglecting some of the less relevant effects, the pinhole camera model often can be used as a reasonable description of how a camera depicts a 3D scene, being commonly used for example in computer vision and computer graphics. [20]

The ideal pinhole camera model describes the perspective projection of a 3D point in the camera coordinate system  $(x_c, y_c, z_c)$  into an image plane point  $(u, v)$ . The optical center and principal point are defined as shown in Figure 2.1. [21, 22]

Defining  $f$  - focal length - as the distance between optical center and image plane, we have

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} \quad (2.1)$$

This model uses the origin of pixel coordinates as principal point, but considering most systems use the upper left corner of an image as origin of image reference, a principal-point offset must be added to the model, which gives

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & u_0 \\ 0 & f & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} \quad (2.2)$$

Besides, this model still assumes pixels are square and not skewed, which may not be always valid. These changes can also be incorporated in the model and we obtain the matrix that describes

the intrinsic camera parameters model used in this work.

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_u & \alpha & u_0 \\ 0 & f_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} \quad (2.3)$$

However, assuming the vision system employs a recent digital camera, we could consider pixels are square ( $f_u = f_v$ ) and non-skewed ( $\alpha = 0$ ).

Moreover, with this simulation we do not want to calculate the projection of a 3D point, but the reverse process, calculating the equation of rays that project in each pixel of the image. Therefore, inverting the camera model and defining  $\lambda_x$  as  $x_c/z_c$  and  $\lambda_y$  as  $y_c/z_c$  we obtain

$$\begin{bmatrix} \lambda_x \\ \lambda_y \end{bmatrix} = \begin{bmatrix} \frac{1}{f_u} & -\frac{\alpha}{f_u f_v} & -\frac{u_0}{f_u} + \frac{\alpha v_0}{f_u f_v} \\ 0 & \frac{1}{f_v} & -\frac{v_0}{f_v} \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \quad (2.4)$$

We can also model the distortions using the Brown's distortion model [23].

The undistorted rays are expressed as

$$\lambda_u = \begin{bmatrix} \lambda_x \\ \lambda_y \end{bmatrix} \quad (2.5)$$

and the distortion center is

$$\lambda_c = \begin{bmatrix} \lambda_{cx} \\ \lambda_{cy} \end{bmatrix} \quad (2.6)$$

Considering  $r^2 = (\lambda_x - \lambda_{cx})^2 + (\lambda_y - \lambda_{cy})^2$ , each ray after distortion is given by

$$\begin{aligned} \lambda_d = & \lambda_u + (\lambda_u - \lambda_c) \sum_{i=1}^{N_r} (R_i \cdot r^{2i}) \\ & + \begin{bmatrix} T_1(r^2 + 2(\lambda_x - \lambda_{cx})^2) & 2T_2(\lambda_x - \lambda_{cx})(\lambda_y - \lambda_{cy}) \\ T_2(r^2 + 2(\lambda_y - \lambda_{cy})^2) & 2T_1(\lambda_x - \lambda_{cx})(\lambda_y - \lambda_{cy}) \end{bmatrix} \left(1 + \sum_{i=3}^{N_t} (T_i \cdot r^{2(i-2)})\right) \end{aligned} \quad (2.7)$$

where  $T_i$  is the  $i^{th}$  tangential distortion coefficient,  $R_i$  the  $i^{th}$  radial distortion coefficient, and  $N_t$  and  $N_r$  the orders of the radial and tangential distortions, respectively. The distortion effect on the rays is a change of direction. After that we use the camera intrinsic parameters to calculate the pixel to rays (after distortion) correspondence:

$$\lambda_d = \begin{bmatrix} \frac{1}{f_u} & -\frac{\alpha}{f_u f_v} & -\frac{u_0}{f_u} + \frac{\alpha v_0}{f_u f_v} \\ 0 & \frac{1}{f_v} & -\frac{v_0}{f_v} \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \quad (2.8)$$

As we are not calibrating the various parts separately, lens distortion is not expected to create

differences in the process, as for the mapping optimization it is the same to fit only the catadioptric distortion, or both catadioptric and lens distortion. Besides, catadioptric distortion has a much greater effect, because with this system the mapping from pixels to ground coordinates usually has a line in the image that corresponds to points at infinity. From those points to the periphery the camera captures the environment above ground. With lens distortion, this does not happen, and the distortion is much softer, sometimes even being invisible to unaided eye.

## 2.2 Mirror Model

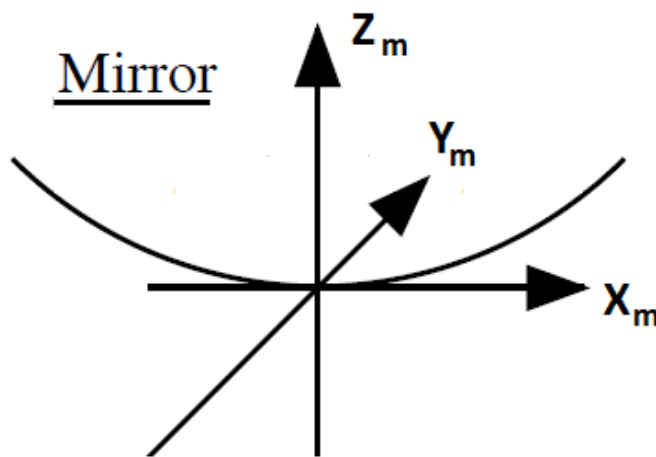


Figure 2.2: Representation of the mirror model, such as mirror pose and its coordinate system

To model the mirror shape we use the equation of a two sheet hyperboloid function, the mirror being one of them. We also consider the radius of the mirror when modeling the vision system. However, this describes an infinite mirror, and in reality the mirror is limited. In those cases where a pixel corresponds to a ray that does not intercept the mirror, and considering the robot setups this work aims at, it will intercept a black slab above the mirror, and so it will be a black point in the image. We use the hyperboloid mirror because it is the kind of mirror used in the soccer robots. Nevertheless, this does not imply the calibration process described in the following sections only works with that type of mirrors, being just the one used in the simulation for tests. Considering the calibration is non-parametric, it has no major relevance in the overall calibration method, although the results may be different.

$$\frac{(z_m + a)^2}{a^2} - \frac{x_m^2 + y_m^2}{b^2} = 1 \quad (2.9)$$

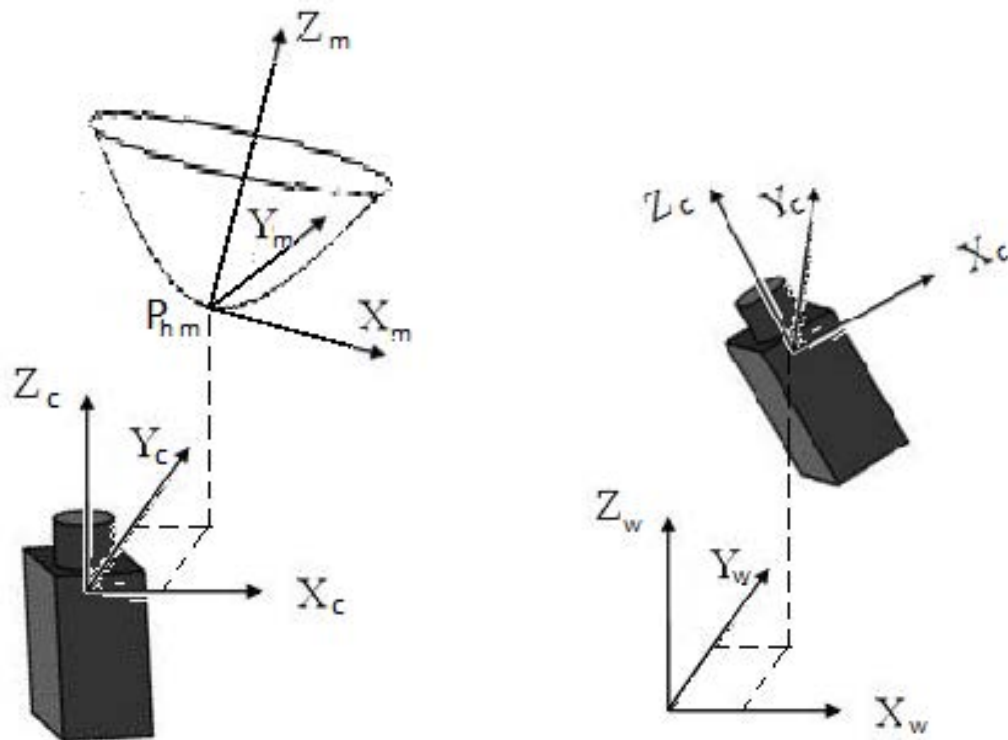
with  $a$  being positive, so the hyperboloid has two sheets.

There can also be some misalignment between the camera and the mirror, which is modeled in the next section.

One example of mirror misalignments can be found in Figure 2.3a.

## 2.3 Coordinate systems transformation

We use this to model both the misalignments and extrinsic parameters. The last ones refer to the change of position of the robot in the soccer field, which in the simulation is the world reference.



(a) Misalignment between camera and mirror

(b) Misalignment of camera in relation to world reference

Figure 2.3: Possible misalignments between the various components of the vision system

As we see in Figure 2.3, there can be misalignments between the various parts that are part of the vision system. In Figure 2.3a we have a non-ideal sensor, with translational and rotational misalignments. In order to have an ideal sensor, the  $x$  and  $y$  coordinates (in the camera reference) of  $P_{hm}$  must be zero. The  $z$  coordinate is non-zero, and represents the distance between the camera and the mirror. In the ideal case, the  $z$  axis of both mirror and camera coordinate systems must be aligned. Rotations along  $z$  axis (mirror reference) have no effect, because the mirror is a curve of revolution. Moreover, in order to have an ideal sensor, the  $z$  axis of camera and world references must be aligned too. However, in this case rotation of camera reference along  $z$  axis is relevant, because it changes robot orientation (extrinsic parameter). However, the camera orientation is not the robot orientation, because camera and robot axis might not be aligned. The  $z$  world projection of the central point of camera coordinate system is an intrinsic parameter, however, its value may change without affecting the type of sensor (ideal or non-ideal). The  $x$  and  $y$  projections of the same point are extrinsic parameters, representing the position of the robot in the soccer field.

Although it is important to distinguish the world reference (soccer field) from the robot reference (coordinate system centered in the robot), in this chapter we do not consider it because it does not affect the simulation procedure. However, for the calibration procedure, it is an important aspect and we deal with it in Chapter 5. Their  $z$  axis are always aligned, being the rotation along this axis between them the real representation of the robot orientation. There's no  $z$  translation between them, and  $x$  and  $y$  translation represents the position of the robot in the field.

If the vision elements are aligned, their reference's axis are aligned too, except for translation. Thus the rotational transformation between them is the identity transformation.

To model the camera-to-mirror coordinate transformation, we use homogeneous transformations, which are composed of a rotation matrix and a translation vector. To build the rotation matrix for this case, we assume a composition of extrinsic rotations (rotations about the fixed reference frame axes, which are the camera axes). Using yaw, pitch, and roll angles, applied to  $x$ ,  $y$  and  $z$  axis, respectively, we obtain the following rotation matrix, relating the mirror referential to the camera coordinate system. Roll angle is not used because  $z$  axis is considered to be the axis of revolution, hence rotations along this axis have no effect.

$$R_m^c = R_{y,\theta_m} R_{x,\psi_m} = \begin{bmatrix} \cos \theta_m & \sin \theta_m \cdot \sin \psi_m & \sin \theta_m \cdot \cos \psi_m \\ 0 & \cos \psi_m & -\sin \psi_m \\ -\sin \theta_m & \cos \theta_m \cdot \sin \psi_m & \cos \theta_m \cdot \cos \psi_m \end{bmatrix} \quad (2.10)$$

with  $\theta_m$  being the pitch angle, and  $\psi_m$  the yaw angle

If we multiply this matrix by a vector in the mirror reference, we obtain the vector orientation given in camera coordinates. For the inverse transformation  $R_c^m$ , we just have to transpose the matrix, because it is orthogonal.

The distance between referentials given in the camera coordinate system is

$$T_m = \begin{bmatrix} t_{mx} \\ t_{my} \\ t_{mz} \end{bmatrix} \quad (2.11)$$

We obtain the following homogeneous matrix describing the mirror-to-camera transformation

$$H_m^c = \begin{bmatrix} R_m^c & T_m \\ 0_{1,3} & 1 \end{bmatrix} = \begin{bmatrix} \cos \theta_m & \sin \theta_m \cdot \sin \psi_m & \sin \theta_m \cdot \cos \psi_m & t_{mx} \\ 0 & \cos \psi_m & -\sin \psi_m & t_{my} \\ -\sin \theta_m & \cos \theta_m \cdot \sin \psi_m & \cos \theta_m \cdot \cos \psi_m & t_{mz} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.12)$$

The inverse homogeneous transformation will be

$$H_c^m = \begin{bmatrix} R_c^m & -R_c^m T_m \\ 0_{1,3} & 1 \end{bmatrix} = \begin{bmatrix} R_m^{cT} & -R_m^{cT} T_m \\ 0_{1,3} & 1 \end{bmatrix} \quad (2.13)$$

To model the system extrinsic parameters, we use the same approach, modeling the camera-to-world transformation with a homogeneous matrix. However, in this case we use the roll angle, as

it has effect in the relative position between ground and camera when acquiring images for various roll angles. In this case the fixed reference is the world. Considering  $\theta_c$  the pitch angle, and  $\psi_c$  the yaw angle, we have

$$R_c^w = R_{z,\phi_c} R_{y,\theta_c} R_{x,\psi_c}$$

$$= \begin{bmatrix} \cos \phi_c \cdot \cos \theta_c & -\sin \phi_c \cdot \cos \psi_c + \cos \phi_c \cdot \sin \theta_c \cdot \sin \psi_c & \sin \phi_c \cdot \sin \psi_c + \cos \phi_c \cdot \sin \theta_c \cdot \cos \psi_c \\ \sin \phi_c \cdot \cos \theta_c & \cos \phi_c \cdot \cos \psi_c + \sin \phi_c \cdot \sin \theta_c \cdot \sin \psi_c & -\cos \phi_c \cdot \sin \psi_c + \sin \phi_c \cdot \sin \theta_c \cdot \cos \psi_c \\ -\sin \theta_c & \cos \theta_c \cdot \sin \psi_c & \cos \theta_c \cdot \cos \psi_c \end{bmatrix} \quad (2.14)$$

If we multiply this matrix by a vector in the camera reference, we obtain the vector orientation given in world coordinates.

The translational difference between referentials is given in the world coordinate system as

$$T_c = \begin{bmatrix} t_{cx} \\ t_{cy} \\ t_{cz} \end{bmatrix} \quad (2.15)$$

We obtain the following homogeneous matrix describing the camera-to-world transformation

$$H_c^w = \begin{bmatrix} R_c^w & -R_c^w T_c \\ 0_{1,3} & 1 \end{bmatrix} \quad (2.16)$$

## 2.4 Back-Propagation Ray Tracing

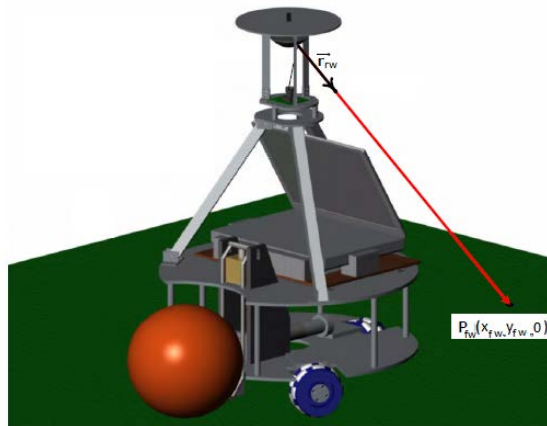


Figure 2.4: Example of back-propagation ray tracing. As shown, from a pixel position and the vision system model, the ray is calculated, and the ground point that originated the ray is estimated. Image obtained from [4].

To calculate the pixel-ground position correspondence, we apply back-propagation ray tracing using the models described previously (following the approach used in [8]) Starting with Equation 2.4 to determine the incident ray on each pixel in the camera reference, described by  $\lambda_{cm} = s(\lambda_x, \lambda_y, 1)$ , we then apply a homogeneous transformation to get the ray equation in the mirror reference

$$\lambda_m = H_c^m \times (s\lambda_x, s\lambda_y, s, 1) = R_c^m \times s(\lambda_x, \lambda_y, 1) - R_c^m \times T_m = s\vec{\lambda}'_m - \lambda''_m \quad (2.17)$$

$$\vec{\lambda}'_m = \begin{bmatrix} \lambda'_{mx} \\ \lambda'_{my} \\ \lambda'_{mz} \end{bmatrix} \quad (2.18)$$

$$\lambda''_m = \begin{bmatrix} \lambda''_{mx} \\ \lambda''_{my} \\ \lambda''_{mz} \end{bmatrix} \quad (2.19)$$

Substituting the ray equation in the hyperbole  $(x_m, y_m, z_m)$  coordinates, we have

$$\frac{(s\lambda'_{mz} - \lambda''_{mz} + a)^2}{a^2} - \frac{(s\lambda'_{mx} - \lambda''_{mx})^2 + (s\lambda'_{my} - \lambda''_{my})^2}{b^2} = 1 \quad (2.20)$$

$$\left( \frac{\lambda'_{mz}{}^2}{a^2} - \frac{\lambda'_{my}{}^2 + \lambda'_{mx}{}^2}{b^2} \right) s^2 + \left( \frac{2\lambda'_{mz}(a - \lambda''_{mz})}{a^2} - \frac{2\lambda'_{my}\lambda''_{my} + 2\lambda'_{mx}\lambda''_{mx}}{b^2} \right) s + \left( \frac{(a - \lambda''_{mz})^2}{a^2} - \frac{\lambda''_{mx}{}^2 + \lambda''_{my}{}^2}{b^2} - 1 \right) = 0 \quad (2.21)$$

$$s_a = \frac{\lambda'_{mz}{}^2}{a^2} - \frac{\lambda'_{my}{}^2 + \lambda'_{mx}{}^2}{b^2} \quad (2.22)$$

$$s_b = \frac{2\lambda'_{mz}(a - \lambda''_{mz})}{a^2} - \frac{2\lambda'_{my}\lambda''_{my} + 2\lambda'_{mx}\lambda''_{mx}}{b^2} \quad (2.23)$$

$$s_c = \frac{(a - \lambda''_{mz})^2}{a^2} - \frac{\lambda''_{mx}{}^2 + \lambda''_{my}{}^2}{b^2} - 1 \quad (2.24)$$

$$s_m = \frac{-s_b \pm \sqrt{s_b^2 - 4s_a s_c}}{2s_a} \quad (2.25)$$

In case  $s_a = 0$  or  $(s_b^2 - 4s_a s_c) < 0$ , the ray does not intersect the mirror.

Next, we calculate the z axis mirror intersection coordinate of the two solutions, as only the positive ones are valid (solutions with negative z coordinate intersect the hyperboloid sheet that do not exist). If there are two possibly valid solutions, the one with lower z is chosen. Considering any possible relative position between camera and mirror, it might happen that the correct solution (first interception of ray with mirror) is the one with greater z mirror coordinate. However, considering



the common position of camera pointing upwards to the mirror, the solution presented will always be right.

Moreover, we can also choose the right solution calculating the vector from the optical camera center and intersection with the mirror using other approach. The shorter one, and in the same direction as  $\lambda_m$  will be the first hyperboloid intersection corresponding to the real mirror-ray intersection. Nevertheless, this approach takes more calculations - the first presented just uses comparisons and will have, in common catadioptric systems, the same result. The solution is  $s_m$ .

The intersection with the hyperboloid in the mirror and camera references are, respectively

$$P_m = s_m \vec{\lambda}'_m - \lambda''_m \quad (2.26)$$

$$P_c = s_m(\lambda_x, \lambda_y, 1) \quad (2.27)$$

and the mirror incident light ray, from camera, is

$$\vec{r}_{im} = \vec{\lambda}'_m \quad (2.28)$$

To calculate the incident ray direction, we use the law of reflection, which states the angle of incidence equals the angle of reflection, and incident, normal and reflected directions are coplanar, the angle of incidence/reflection being the angle of incoming/outgoing light with respect to the normal of the surface.

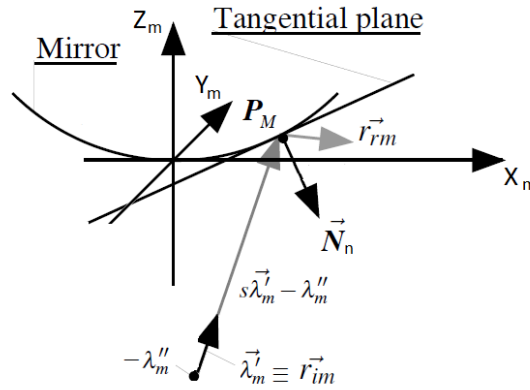


Figure 2.5: Reflection of a ray in the mirror. Using the surface normal vector and the incident ray direction, the reflection ray direction is estimated. Image from [5]

In order to calculate vector normal to the mirror surface, which is defined implicitly as the set of points  $(x_m, y_m, z_m)$ , we define a 4-D function  $F(x_m, y_m, z_m)$  such that  $F(x_m, y_m, z_m) = K$  corresponds to the mirror equation. Considering this, the normal at a point  $(x_m, y_m, z_m)$  on the surface is given by the gradient  $\nabla F(x_m, y_m, z_m)$ , since the gradient at any point is perpendicular to the level set, and  $F(x_m, y_m, z_m) = K$  (the mirror equation) is a level set of  $F$ . Therefore, with  $F$  being

$$F(x_m, y_m, z_m) = -\frac{(z_m + a)^2}{a^2} + \frac{x_m^2 + y_m^2}{b^2} + 1 \quad (2.29)$$

the surface normal is

$$\vec{N}(x_m, y_m, z_m) = \nabla F(x_m, y_m, z_m) = \begin{bmatrix} \frac{2x_m}{b^2} \\ \frac{2y_m}{b^2} \\ -\frac{2(z_m+a)}{a^2} \end{bmatrix} \quad (2.30)$$

pointing away from the mirror. We then normalize this vector to make it unitary.  $\vec{N}_n = \vec{N} / \|\vec{N}\|$

Using the intersection point, we calculate the surface normal at that point  $\vec{N}_p$ , and we calculate the incident ray as

$$\vec{r}_{rm} = \vec{r}_{im} - 2 \langle \vec{N}_p, \vec{r}_{im} \rangle \vec{N}_p \quad (2.31)$$

Then, we need the reflected ray orientation in the camera coordinates, so

$$\vec{r}_{rc} = R_m^c \vec{r}_{rm} \quad (2.32)$$

And the mirror intersection point in the camera reference is

$$P_c = H_m^c \times P_m \quad (2.33)$$

We also checked if  $x_m^2 + y_m^2 \leq hr_m^2$ , with  $hr$  being the mirror radius, to check the ray really intersects the mirror or not.

Then we just need to get the mirror intersection point and reflected ray direction into world coordinates.

$$\vec{r}_{rw} = R_c^w \vec{r}_{rc} \quad (2.34)$$

$$P_w = H_c^w \times P_c \quad (2.35)$$

Finally, to calculate the ground intersection, we just need do confirm the z coordinate of  $\vec{r}_{rw}$  is positive, and then the ground intersection is given by

$$P_w = (P_{wx}, P_{wy}, P_{wz}) \quad (2.36)$$

$$\vec{r}_{rw} = (r_{wx}, r_{wy}, r_{wz}) \quad (2.37)$$

$$k = -P_{wz} / r_{wz} \quad (2.38)$$

$$x_{fw} = P_{wx} + k \cdot r_{wx} \quad (2.39)$$

$$y_{fw} = P_{wy} + k \cdot r_{wy} \quad (2.40)$$

## 2.5 Results

Applying the method described previously for each pixel of the image we obtain the correspondent ground point. Using that approach, we can simulate the images acquired by the vision system. In the next pictures we show some results of the vision system simulation.

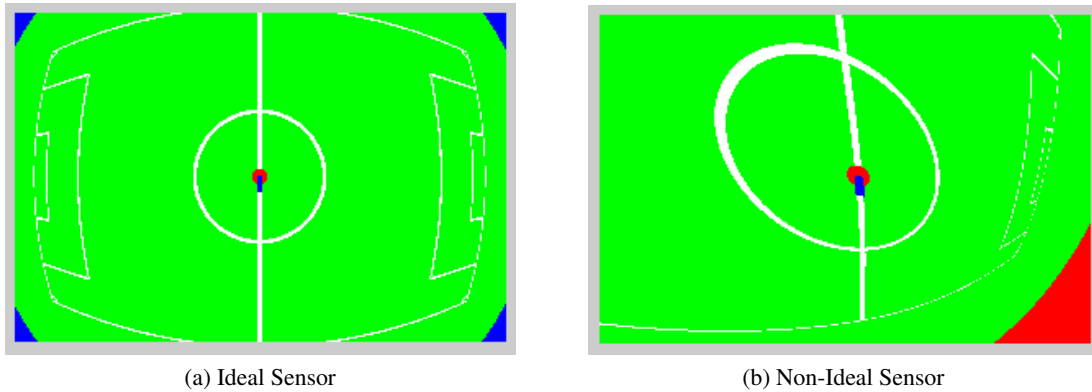


Figure 2.6: Comparison between different catadioptric vision system simulation. In (a) the system has a SVP, because there are no misalignments, and the ideal pinhole camera model is used (only with principal point offset). In (b) misalignments, pixel skewness and aspect ratio is simulated.

In Figure 2.6 we show the difference in the results when using simulation of vision systems with different parameters. While in the first case we use a central vision system, thus having only radial distortion, in the second one there are strong misalignments, and so the vertical field of view is not the same in all directions. In this images, red regions correspond to points above ground, and blue ones are non-valid regions due to not intersection of the mirror by the rays corresponding to that pixels. This effect is due to the limited radius of the mirror.

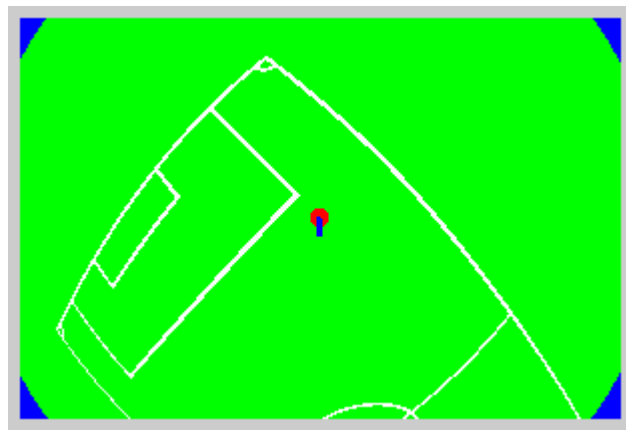


Figure 2.7: Simulation of ideal catadioptric vision system with robot in a different field position.



## Chapter 3

# Mapping Functions

After building the simulation, we tested some types of functions to know how well they would fit the kind of distortion of a catadioptric vision system. The polynomial functions are the obvious idea, as they are specially suited for linear optimization. Therefore, we firstly tried to estimate two polynomial functions for  $x$  and  $y$  world coordinate, as a function of pixel coordinates  $u$  and  $v$ .

However, considering the type of system we are dealing with, we decided to tests this also with another polynomial function, now using polar coordinates. However, this makes it a non-linear optimization, because we need to calculate the center pixel that is the origin of the polar reference coordinate system. To overcome this problem we propose a multi-step optimization.

Considering the geometry of the catadioptric system, the last type of functions was expected to give the best results, even for non-SVP systems. However, it only showed to be better when fitting SVP vision systems, in which all the parts are aligned. But for the other cases, namely the ones we are assuming we want to calibrate, the usage of Cartesian coordinated seems to have better results.

It is important to realize this tests were done using all image pixels, and not only the field lines. This was done because at this phase we only want to analyze which functions better fit the calibration of catadioptric systems, by comparison of the errors obtained for each one with different orders (polynomial degrees). However, we are not using all points of image, as there are regions that we do not want to calibrate, for example, regions corresponding to infinity in the ground reference. Besides, those points would make the fitting much more difficult, and to improve in those regions, in the main part of the image it would became worse.

Besides, when all points are used in the calibration process, we only need to get information from one test image. However, if only lines are used, this can not be done because the mapping will fit reality better in the field lines. As field regions without lines are only extrapolated, the function is not expected to be so close to the true mapping as it is in the lines region.

### 3.1 Polynomial Approximation using Cartesian Coordinates

For each calibration point  $l$ , we have

$$\hat{x}_l(u_l, v_l) = \sum_{i=0}^O \sum_{j=0}^i \beta_{ij} u_l^{i-j} v_l^j = \sum_{k=1}^m \beta_k \phi_k(u_l, v_l) = \sum_{k=1}^m S_{lk} \beta_k = S_l \beta \quad (3.1)$$

$$\hat{y}_l(u_l, v_l) = \sum_{i=0}^O \sum_{j=0}^i \alpha_{ij} u_l^{i-j} v_l^j = \sum_{k=1}^m \alpha_k \phi_k(u_l, v_l) = \sum_{k=1}^m S_{lk} \alpha_k = S_l \alpha \quad (3.2)$$

where for each  $\beta_{ij}$  corresponds a parameter  $\beta_k$ ,  $O$  is the order of the polynomial function, and  $m$  the total number of parameters.  $S_l$  is a line vector, with  $S_{lk}$  being its elements, and  $\beta$  is a vector with all the parameters  $\beta_k$ .

We should also emphasize that, instead of using  $u$  and  $v$ , if we use  $u - u_0$  and  $v - v_0$ , the results, theoretical should be the same, except for the values of constants  $k_{ij}$ , but the mapping should result the same. However, for numerical reasons, when only using  $u$  and  $v$  the least square method would return a warning saying the matrix used for the calculations was “singular to working precision”, and the resulting mapping would not properly fit the distortion. For  $u_0$  and  $v_0$ , we just choose the center of the image.

As we have the real values for each calibration point,  $x(u, v)$  and  $y(u, v)$ , we can estimate  $\beta$  and  $\alpha$  parameters. From now on we only demonstrate how to calculate  $\beta$ , but the same applies to  $\alpha$ . In order to find the coefficients  $\beta$  that fit the known data best, we use the quadratic minimization approach [24]

$$\hat{\beta} = \arg \min_{\beta} J_x(\beta) \quad (3.3)$$

where the objective function  $J_x$  is given by

$$J_x(\beta_x) = \sum_{l=1}^{N_p} |x_l - \hat{x}_l|^2 = \sum_{l=1}^{N_p} \left| x_l - \sum_{k=1}^m S_{lk} \beta_k \right|^2 = \|x - S_x \beta\|^2 \quad (3.4)$$

where  $x$  is a vector (size  $l$ ) with the x coordinate of each calibration point,  $S_x$  is a matrix with  $l$  lines, each being  $S_l$ , and  $N_p$  is the number of calibration points.

Given this formulation of the problem, and considering  $\hat{y}$  and  $\hat{x}$  functions are linear combinations of partial functions, we can use a linear method to estimate the parameters - linear least squares optimization. The solution in this case is them given by

$$\hat{\beta} = (S_x^T S_x)^{-1} S_x^T x \quad (3.5)$$

For that purpose we used *Matlab* method *lscov*, which instead of equation 3.5 uses the QR factorization (slower than the normal equations method but more numerically stable), resulting in

$$R \hat{\beta} = Q^T x \quad (3.6)$$

Being  $R$  an upper triangular matrix and  $Q$  an orthogonal matrix such as  $S_x = QR$ . The solution can be easily found with backward substitution because  $R$  is upper triangular.

In 3.1 we show the fitting results of the calibration, for polynomial degree ranging from five to twenty. We use two case studies, the ideal sensor - no misalignments and ideal pinhole camera model - and non-ideal sensor - with strong misalignments. We show the average and maximum error for each case, for both the ideal and non-ideal sensor simulated in 2.6. As expected, when increasing the order of the function the error decreases. It is also easy to note that errors are lower in the SVP case, which can be explained by the lower degree of distortion in the second case systems.

Table 3.1: Fitting error of polynomial function estimation (cartesian coordinates)

Order	Non-ideal Sensor		Ideal Sensor	
	Maximum error (cm)	Average error (cm)	Maximum error (cm)	Average error (cm)
5	222.25	14.82	21.83	2.34
6	150.25	9.00	22.21	2.34
7	100.76	5.54	3.93	0.26
8	67.02	3.44	4.01	0.26
9	44.18	2.14	1.44	0.13
10	29.19	1.33	1.47	0.13
11	19.37	0.83	0.10	0.01
12	12.93	0.52	0.10	0.01
13	8.64	0.33	0.11	0.01
14	5.76	0.21	0.11	0.01
15	3.83	0.13	0.01	0.00
16	2.54	0.08	0.01	0.00
17	1.68	0.05	0.01	0.00
18	1.11	0.03	0.01	0.00
19	0.73	0.02	0.00	0.00
20	0.48	0.01	0.00	0.00

As we can see from the error image in Figure 3.1b, the error is greater at the calibration frontier, and quite lower in the rest of the image, which explains the big difference present in Table 3.1 between maximum and average error. We use Figure 3.1b and Figure 3.1a to analyze the error, because they give a different perspective of the error. In the first we calculate the estimated world position corresponding to each calibration pixel. If it belongs to a line, we print a black pixel. If not, it corresponds to a field point that is not line. For that case, it will be a pink point. We make it different from green, to differentiate points that belong to the calibration, and those that do not. Finally, the world re-projection might not be accurate, and so if it corresponds to a line, we print it yellow instead of pink, in order to make the true lines position visible. In the second image, we use our knowledge of the current world correspondence between pixel and ground coordinate, and use it to calculate the calibration error of each pixel. Then, we normalize it dividing it by its maximum value. Therefore, black points correspond to no error, and white (value of one) corresponds to maximum error. While the first one gives a better feedback of the fitting quality (which is not visible in the second as it is always normalized), the second one gives a better idea of the error spatial distribution. For example, in the first one a black point in a the lines position,

or a pink point, might give the idea that region is well calibrated, but that might not be true, as we only know the mapping still corresponds to a line/field point, but do not know if it is the correct position. This problem arises because we are representing a four dimensional function in a plane. The color code is presented in Table 3.2. In order to know the quality of the calibration procedure, it is important to have information relative to the field size. In this case we simulated a field with  $10\text{m}\times 6\text{m}$ , with line width of 6cm. Considering this and choosing an appropriate order for the estimating function, the calibration procedure can have relatively low error.

Table 3.2: Color code of image reprojection (calibration using both field lines and non field lines points)

Color	Meaning
Blue (periphery)	Rays from camera do not intersect the mirror due to its limited radius
Red (periphery)	Rays from camera intersect the mirror, but direction of reflected ray points upward, not intersecting the floor
Black	Pixels reprojected into the soccer field lines
Green	Field pixels (not field lines) that do not reproject into field lines (pixel not used for calibration)
Pink	Field pixels (not field lines) that do not reproject into field lines (pixel used in calibration)
White	Field lines in the test image whose reprojection is not a line (pixel not used for calibration)
Yellow	Field lines in test image whose reprojection is not a line (pixel used in calibration)

We start by testing it in a non-ideal sensor, as shown in Figure 3.1.

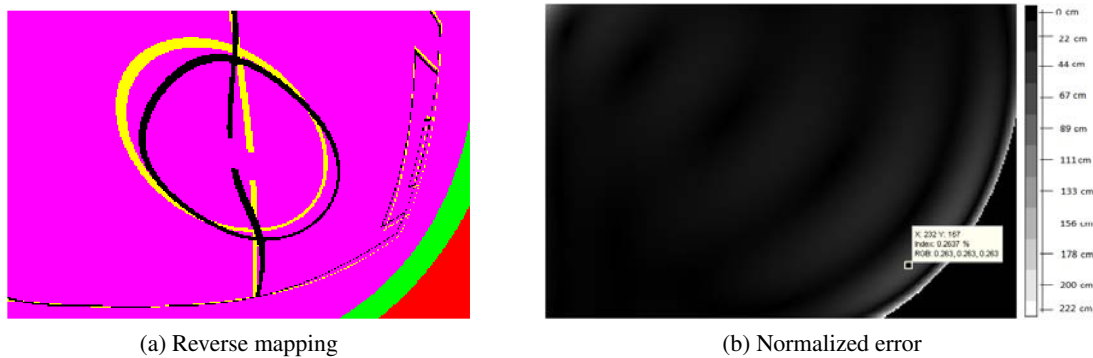


Figure 3.1: Calibration results for the non-ideal sensor using a cartesian coordinates  $5^{th}$  degree polynomial function. In (a) we represent the reverse mapping of points, and in (b) we represent the normalized calibration error.

Besides, we can also confirm that while five degree polynomials are a poor fit to the real mapping, with a ten degree function the calibration results are already completely acceptable results, being the error in the center less than one centimeter, and the maximum error around thirty centimeters. However, as we can see in Figure 3.2, that only happens in the calibration frontier, with local maximum errors in the rest of the image being quite low. Finally, it is important to notice



that, when increasing the order, and considering the true mapping has no oscillations but a curvature always with the same orientation, the oscillations of the polynomial get a higher frequency, which results in error oscillations. In this case the results are better, because we are using well distributed calibration points. However, with fewer points in the calibration, a image region might have big oscillation, thus great error, so calibrated areas can be closely mapped.

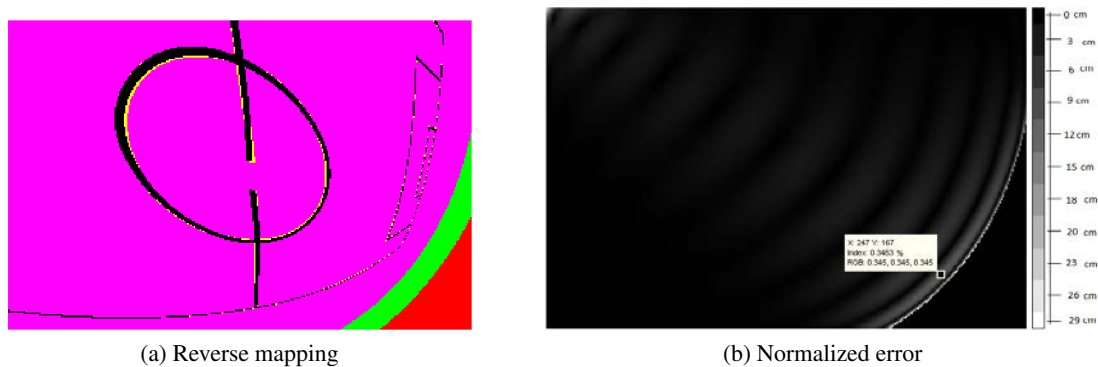


Figure 3.2: Calibration results for the non-ideal sensor using a cartesian coordinates  $10^{th}$  degree polynomial function. In (a) we represent the reverse mapping of points, and in (b) we represent the normalized calibration error.

Another interesting test is the calibration of ideal sensors. Although we do not have those in real setups, the resulting can still be closer to this example than the obtained from the non-ideal sensor simulation. As shown in Table 3.1, Figure 3.3 confirms the better quality calibration results. Although a 5-degree polynomial was not a good choice in the last case, in this the results are quite acceptable, even similar to the ones obtained for the last case with a ten degree function.

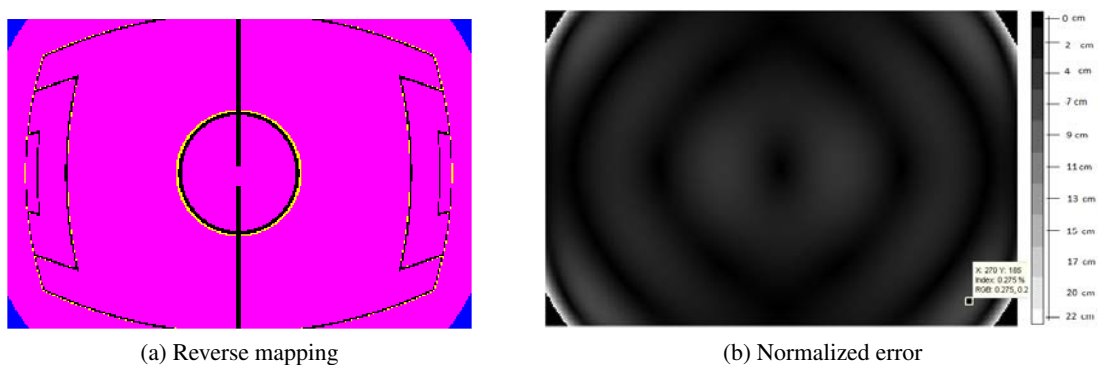


Figure 3.3: Calibration results for the ideal sensor using a cartesian coordinates  $5^{th}$  degree polynomial function. In (a) we represent the reverse mapping of points, and in (b) we represent the normalized calibration error.

## 3.2 Polynomial Approximation using Polar Coordinates

Again, the same as Equations 3.7 and 3.8 is applied with polar coordinates.

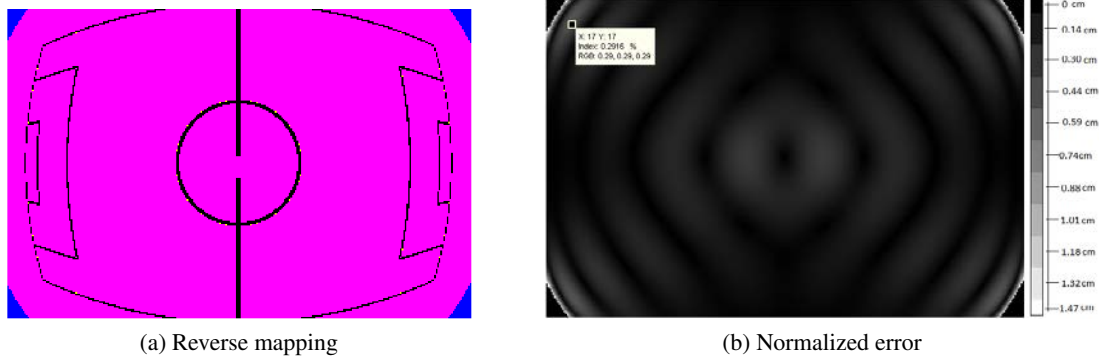


Figure 3.4: Calibration results for the ideal sensor using a cartesian coordinates  $10^{th}$  degree polynomial function. In (a) we represent the reverse mapping of points, and in (b) we represent the normalized calibration error.

$$\hat{\rho}_{wl}(\rho_{al}, \theta_{al}) = \sum_{i=0}^O \sum_{j=0}^i \beta_{ij} \rho_{al}^{i-j} \theta_{al}^j = \sum_{k=1}^m \beta_k \phi_k(\rho_{al}, \theta_{al}) = \sum_{k=1}^m S_{lk} \beta_k = S_l \beta \quad (3.7)$$

$$\hat{\theta}_{wl}(\rho_{al}, \theta_{al}) = \sum_{i=0}^O \sum_{j=0}^i \alpha_{ij} \rho_{al}^{i-j} \theta_{al}^j = \sum_{k=1}^m \alpha_k \phi_k(\rho_{al}, \theta_{al}) = \sum_{k=1}^m S_{lk} \alpha_k = S_l \alpha \quad (3.8)$$

with

$$\hat{\rho}_{wl} = \sqrt{(x - x_0)^2 + (y - y_0)^2} \quad (3.9)$$

$$\hat{\theta}_{wl} = \arctan \frac{y - y_0}{x - x_0} \quad (3.10)$$

$$\hat{\rho}_{al} = \sqrt{(u - u_0)^2 + (v - v_0)^2} \quad (3.11)$$

$$\hat{\theta}_{al} = \arctan \frac{v - v_0}{u - u_0} \quad (3.12)$$

$$(3.13)$$

Although in the cartesian coordinates case we would have  $u_0$  and  $v_0$  just for numerical reasons, in this case it is part of the parameters we have to estimate. Therefore, this is not a linear optimization problem anymore. As we don't have an initial guess for the calibration parameters, a non-linear optimization would not be accurate, thus we use a multi-step optimization. In the first optimization (linear), we consider the center of the image referential the center of the image and the center of the world reference the point  $(0, 0)$ . Then, we perform a least square method to calculate two polynomial functions  $\rho_w$  and  $\theta_w$  as a function of  $\rho_a$  and  $\theta_a$ . We then have an initial solution, so we can use a non-linear optimization to tune all the parameters - the references' centers and the polynomial parameters. However, with high order polynomial functions the last process can take a long time, and so we instead use a three step optimization, as shown in 3.3.

The first step is the same, but in the second we use the former results as initial solution to get an estimate of the following parameters:  $u_0$ ,  $v_0$ ,  $y_0$  and  $x_0$ . Only after that we use a third optimization to tune all the parameters. For the non-linear optimization the Trust Region Reflective algorithm was used.

Table 3.3: Multi-step optimization

Step	Optimization Description
1	$u_0 = size_u/2$ , $v_0 = size_v/2$ , $y_0 = 0$ and $x_0 = 0$ . $\beta$ and $\alpha$ parameters obtained using linear optimization (least squares, using QR factorization)
2	Using $\alpha$ and $\beta$ obtained previously, $u_0$ , $v_0$ , $y_0$ and $x_0$ are estimated using nonlinear optimization (Trust Region Reflective algorithm)
3	All parameters are tuned using nonlinear optimization (Trust Region Reflective algorithm)

In order to improve the overall result, we can repeat steps 1 and 2 (Table 3.3 until there are no further improvement), and only then use step 3. The last step is important because a minimum when estimating parameters separately may not be the global optimum when estimating all the parameters at the same time.

However, now there are four more parameters describing the center of each polar coordinate system in the world and image plane references. Their orientation is not changed, as that can be modeled using the equation above.

For this case, we need a multi-step calibration. In Figure 3.5 we show the results for different approaches of five degree polynomial calibration. We start with only the linear optimization, then combining it with a non-linear method to estimate all parameters, and finally we test a multiple step approach. In the latter, it follows the linear optimization to find the polynomial parameters with a non-linear method to estimate the references centers, doing it until the error can't be decreased. Finally, as this method estimates the best parameters separately (which may lead to non minimal solutions), we use a global non linear estimation. As we can conclude from Figure 3.5, the results are not promising, and the increased cost of nonlinearity is not worth the difference in the results.

In Figure 3.6 we did the same test, but with higher degree polynomials. As expected, there is little difference between only linear and multi-step optimizations, and so we decided to use only linear calibration in the following tests. Nevertheless, we noticed that even with the system tested, which copes strong misalignments, with an order increase of the function, the results can be relatively good.

As we did previously, we again present in Table 3.4 the results of polar coordinates polynomial function, with varying degrees and using two different types of catadioptric vision systems. While for SVP systems the results are quite good, with average error lower than one centimeter even for five degree functions, the same does not happen with the other system. In that one the error is never has satisfying has obtained with cartesian coordinates. Besides, we noticed that after order ten, the error is quite unstable, seeming to increase and decrease almost randomly.

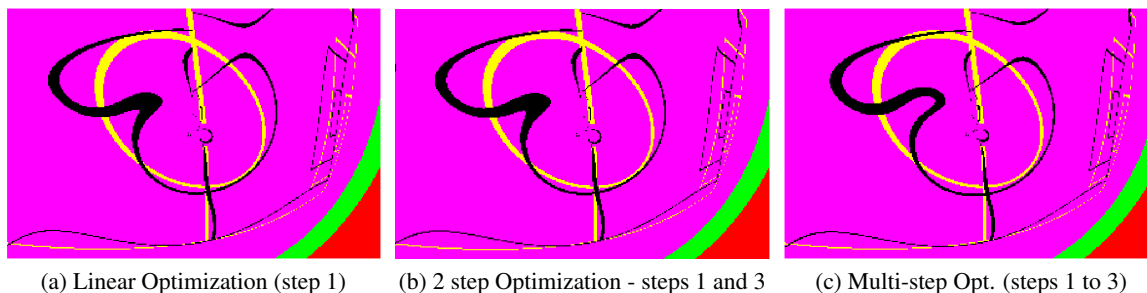


Figure 3.5: Calibration results using various approaches: (a) uses a linear optimization, (b) a combination of one linear and one nonlinear optimizations, and (c) uses a multi-step combination of both linear and nonlinear parameter estimations, with repetition of steps 1 and 2 until no further improvement. It uses a five degree polynomial function. The optimization steps are presented in 3.3.

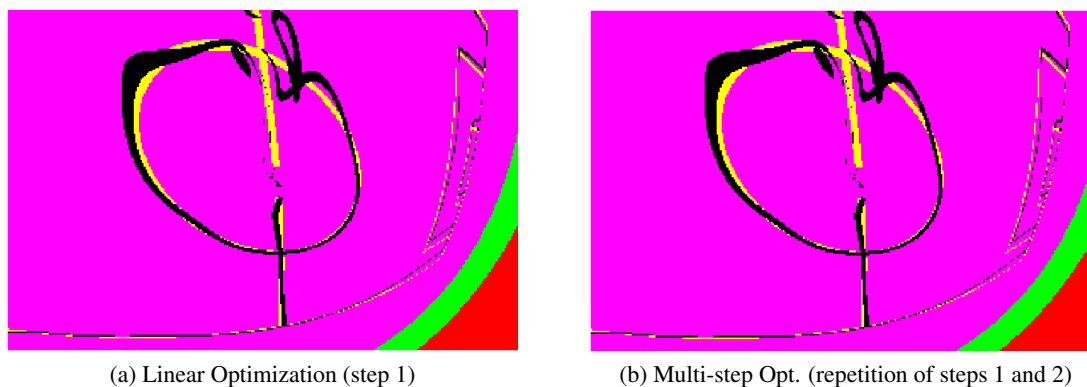


Figure 3.6: Calibration results using various approaches: (a) uses a linear optimization, (b) a multi-step combination of linear and nonlinear optimizations. Step 3 is not used because, with a 11<sup>th</sup> degree polynomial function, it takes a long time. The optimization steps are presented in 3.3.

We explain the difference in the results for the two sensors as follows. In the ideal case, our first estimate of the coordinate system centers - the center of the image - is correct. Besides,  $\theta_w = \theta_i$ , as the image only has radial distortion. Therefore, we only need a function to estimate  $\rho_w$  as function of  $\rho_i$ , and our linear optimization is enough to find a polynomial that fits that. However, in the other case, that does not hold, hence the need of multi-step optimization. Nevertheless, due to common local minimum problem of non linear optimization, that does not have any result, and so the results are poorer.

As with cartesian coordinates, there is usually greater errors in the frontier of the calibration region. However, in this case, besides further regions (big  $\rho$ ), there is also another frontier - the  $-180/180$  degrees angle. As it is another frontier, the error in that region is greater than in the rest of the image.

However, for the SVP system case, the angle frontier does not apply any more. This can be explained due to the perfect alignments, that, as said before, make  $\theta_w = \theta_i$  and  $\rho_w$  function of  $\rho_i$ , and so the results for  $\theta$  equal to 180 or -180 degrees are the same. It is important to notice the

Table 3.4: Fitting error of polynomial function estimation (polar coordinates)

Order	Non-ideal Sensor		Ideal Sensor	
	Maximum error (cm)	Average error (cm)	Maximum error (cm)	Average error (cm)
5	536.12	38.99	5.78	0.45
6	487.57	31.21	2.57	0.16
7	387.90	26.09	1.03	0.05
8	371.13	23.20	0.34	0.02
9	322.75	20.23	0.14	0.01
10	332.68	16.73	0.07	0.00
11	227.23	13.66	0.03	0.00
12	241.32	11,71	0.03	0.00
13	292.81	11.50	0.03	0.00
14	1648.80	25.11	0.00	0.00
15	907.42	12.64	0.04	0.00
16	491.06	10.26	0.01	0.00
17	257.81	8.60	0.01	0.00
18	463.37	10.80	0.40	0.01
19	699.54	13.55	0.00	0.00
20	1965.35	50.40	0.00	0.00

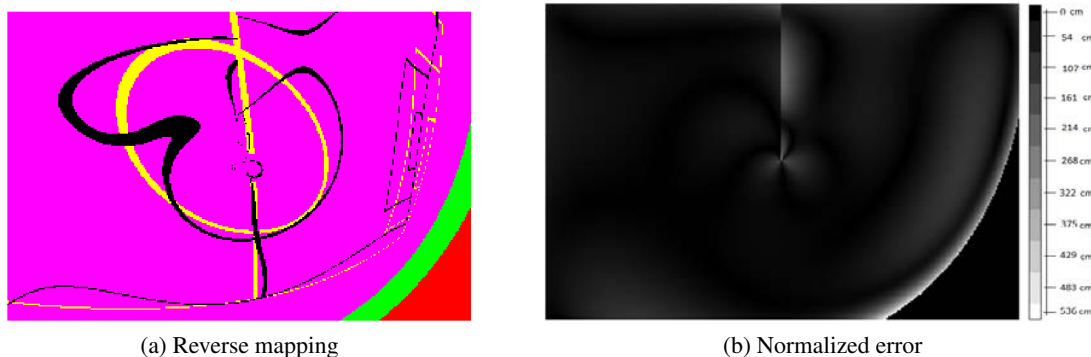


Figure 3.7: Calibration results for non-ideal sensor using a polar coordinate  $5^{th}$  degree polynomial function. In (a) we represent the reverse mapping of points, and in (b) we represent the normalized calibration error.

difference in the calibration quality between ideal (Figure 3.7) and non-ideal sensors (Figure 3.9). While using the same order for polynomial function, in the first one the result is not acceptable, but in the latter the result is almost perfect. This is due to the correct initial values used for the function parameters in the ideal case (which are incorrect in the other case). Moreover, another reason to explain this is that for ideal sensors this kind of functions is a good fit for the mapping, but not for non-ideal sensors.

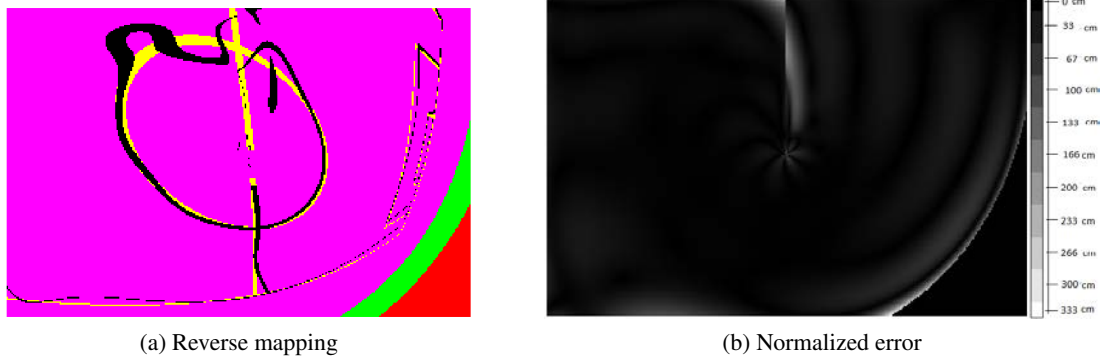


Figure 3.8: Calibration results for non-ideal sensor using a polar coordinate  $10^{th}$  degree polynomial function. In (a) we represent the reverse mapping of points, and in (b) we represent the normalized calibration error.

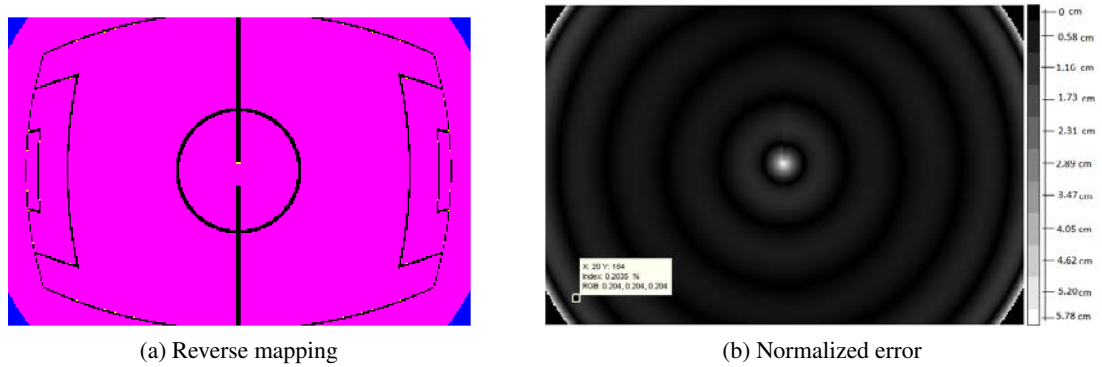


Figure 3.9: Calibration results for an ideal sensor image using a polar coordinate  $5^{th}$  degree polynomial function. In (a) we represent the reverse mapping of points, and in (b) we represent the normalized calibration error.

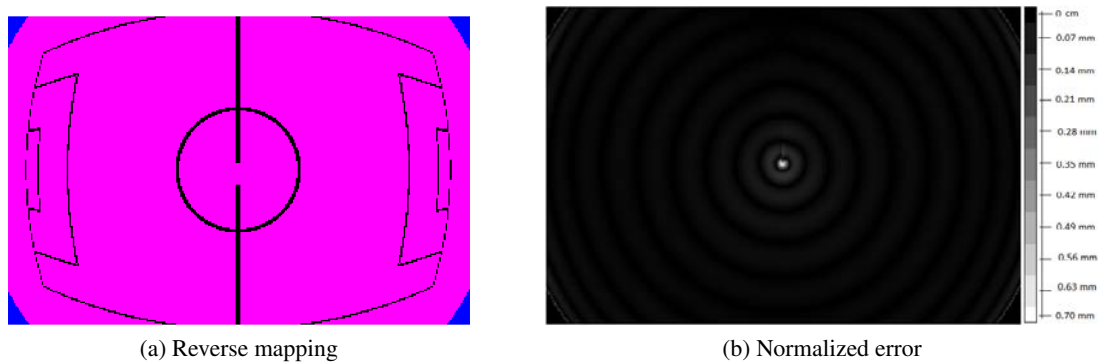


Figure 3.10: Calibration results for an ideal sensor image using a polar coordinate  $10^{th}$  degree polynomial function. In (a) we represent the reverse mapping of points, and in (b) we represent the normalized calibration error.

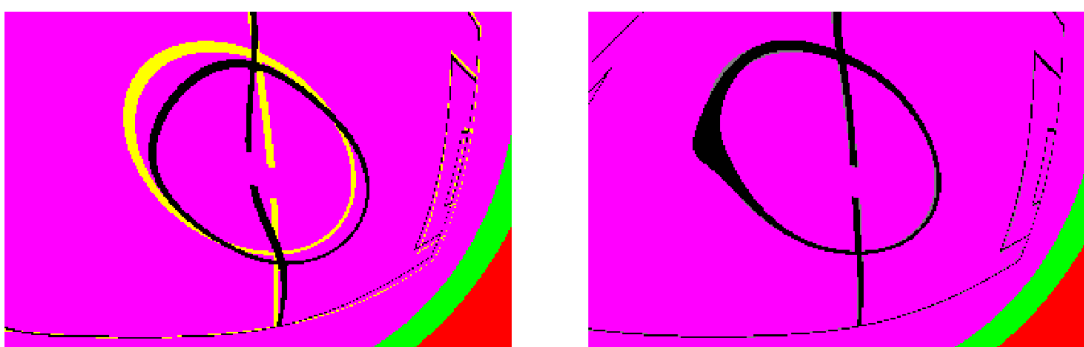
### 3.3 Usage of only lines

Although we decided to analyze the functions using all points, we know that in reality we can not do it, as it will affect the final result. Therefore, in this section we compare the results of calibration when using only lines or using all points. However, this is still not the real case, as we use all points belonging to lines, and their  $x$  and  $y$  world coordinates. So, all the results in here must be accentuated in order to apply them in the real case. For this tests we use the cartesian coordinates, as it has the overall best results considering the two types of systems.

The reverse mapping color code for this specific case is presented in Table 3.5.

Table 3.5: Color code of image reprojection (calibration using only field lines)

Color	Meaning
Blue (periphery)	Rays from camera do not intersect the mirror due to its limited radius
Red (periphery)	Rays from camera intersect the mirror, but direction of reflected ray points upward, not intersecting the floor
Black	Pixels reprojected into the soccer field lines
Green	Field pixels (not field lines) that do not reproject into field lines (pixel not used for calibration nor validation)
Pink	Field pixels (not field lines) that do not reproject into field lines (pixel not used in calibration, but used in validation with test image)
Cyan	Field pixels (not field lines) that do not reproject into field lines (pixel used in calibration and validation)
White	Field lines in the test image whose reprojection is not a line (pixel not used for calibration, neither for testing)
Yellow	Field lines in test image whose reprojection is not a line (pixel not used in calibration, but used in validation)
Grey	Field lines in test image whose reprojection is not a line (pixel used in calibration and validation)



(a) Reverse Mapping (calibration using all field points)

(b) Reverse Mapping (calibration using only field lines)

Figure 3.11: Comparison between calibration using only line points and using all points (inside a predetermined region), using a 5 degree polynomial function. The result is presented in the form of a reverse mapping of the calibration image. The color code for (a) is in table 3.2, and for (b) is in table 3.5. The test is done in the image used for calibration.

As we see in Figure 3.11, when using a five degree polynomial function to fit only the lines, the calibration seems to get way better. However, this happens because that figure is specially suited to analyze the reprojected lines position. Considering only lines were used for calibration, that was expected to happen. That is the reason why in this cases we should use different images for the calibration and validation process, as only the last ones can tell us the extrapolation quality in the regions not calibrated. However, even in 3.11 we can see that non calibrated regions have reprojected lines (in black) that should not be there.

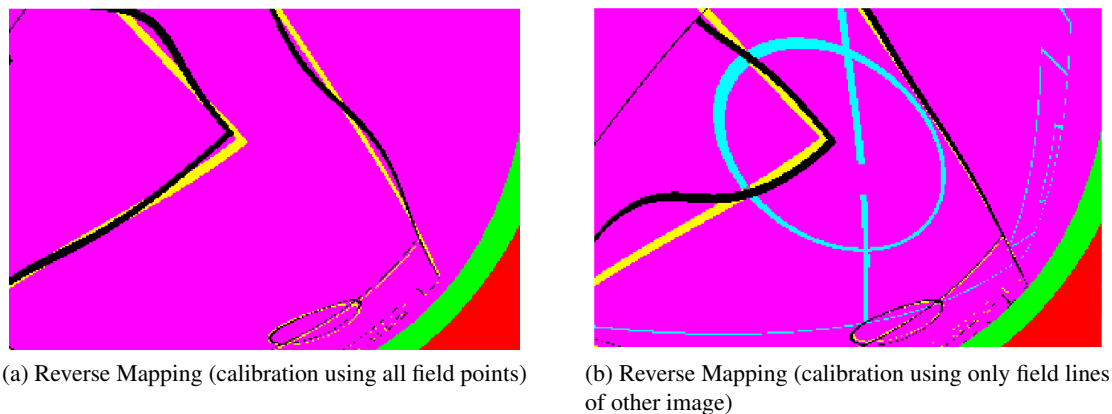


Figure 3.12: Comparison between calibration using only line points and using all points (inside a predetermined region), using a 5 degree polynomial function. The result is presented in the form of a reverse mapping of the calibration image. The color code for (a) is in table 3.2, and for (b) is in table 3.5. The test is done using an image different from the calibration image.

In Figure 3.12 we confirm that by testing the system with a different image. As expected, there are lines that are well fitted (in regions calibrated), but there are also some lines that move away from its correct position, because it was not calibrated properly. In this case, we still see regions with lines that should not have anything in there.

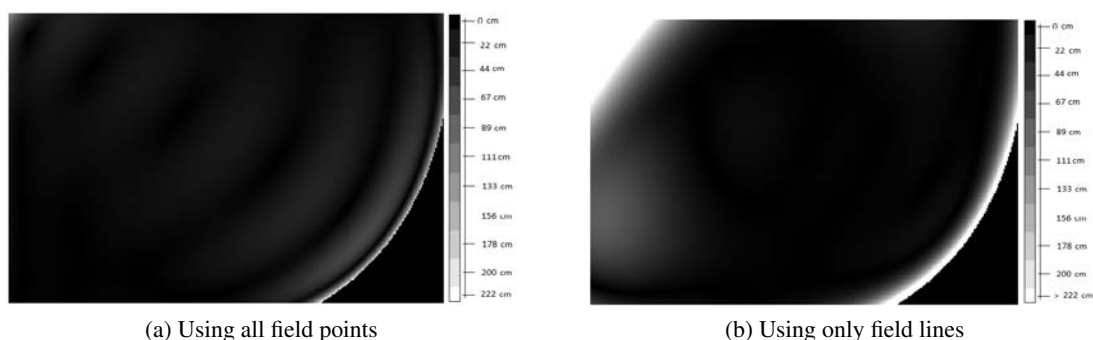


Figure 3.13: Comparison between calibration using only line points and using all points (inside a predetermined region), using a 5 degree polynomial function. The result is presented using the calibration error of each one.

Again, to analyze the error we can also use Figure 3.13. In Figure 3.13b, the error is not



normalized to be between 0 (no error) and 1 (maximum error), but normalized so the same gray intensity in both images corresponds to the same error, making it possible to compare the results. An interesting feature is that error seems to get lower even in regions not used for calibration, as long as they are in the central part of the image and relatively near some line used for calibration. However, in regions where this condition is not satisfied, the error is much higher, which makes the maximum and average error worse (maximum error is 939.68 centimeter and average error is 31.247 centimeter).

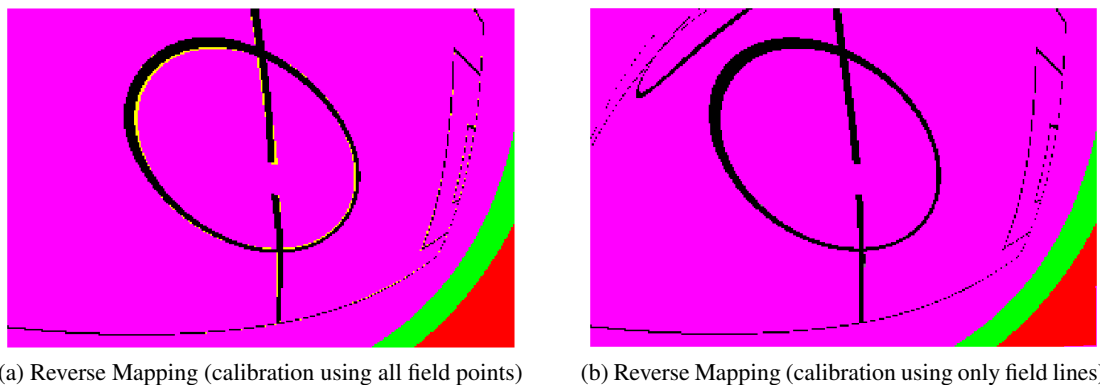
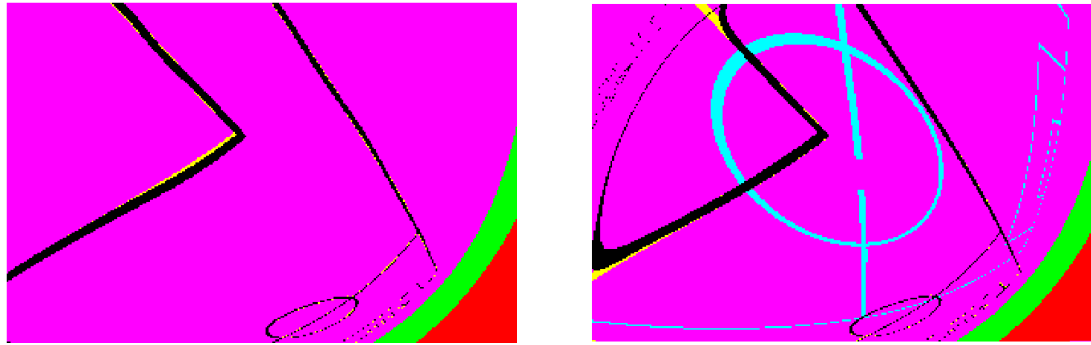


Figure 3.14: Comparison between calibration using only line points and using all points (inside a predetermined region), using a 10 degree polynomial function. The result is presented in the form of a reverse mapping of the calibration image. The color code for (a) is in table 3.2, and for (b) is in table 3.5. The test is done in the image used for calibration.

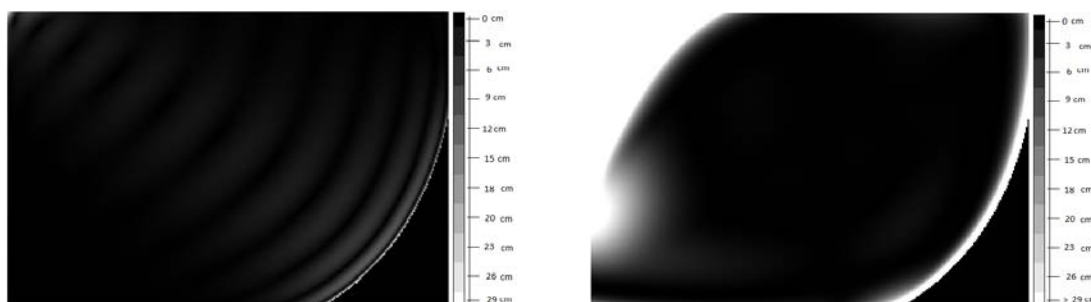
Using the 10 order polynomial function, we get similar results, but know maximum error is 11878 centimeter and average error is 73.24 centimeter. Therefore, instead of getting better when increasing the order, the reverse happens. This is a effect we discussed earlier, the usage of high order function in regions with low calibration points. In regions used for calibration the function fits better (Figure 3.14), but in other regions the extrapolation gets out of control (easily seen in Figure 3.16).



(a) Reverse Mapping (calibration using all field points)

(b) Reverse Mapping (calibration using only field lines of other image)

Figure 3.15: Comparison between calibration using only line points and using all points (inside a predetermined region), using a 10 degree polynomial function. The result is presented in the form of a reverse mapping of the calibration image. The color code for (a) is in table 3.2, and for (b) is in table 3.5. The test is done using an image different from the calibration image.



(a) Using all field points

(b) Using only field lines

Figure 3.16: Comparison between calibration using only line points and using all points (inside a predetermined region), using a 10 degree polynomial function. The result is presented using the calibration error of each one.

## Chapter 4

# Field Lines Identification

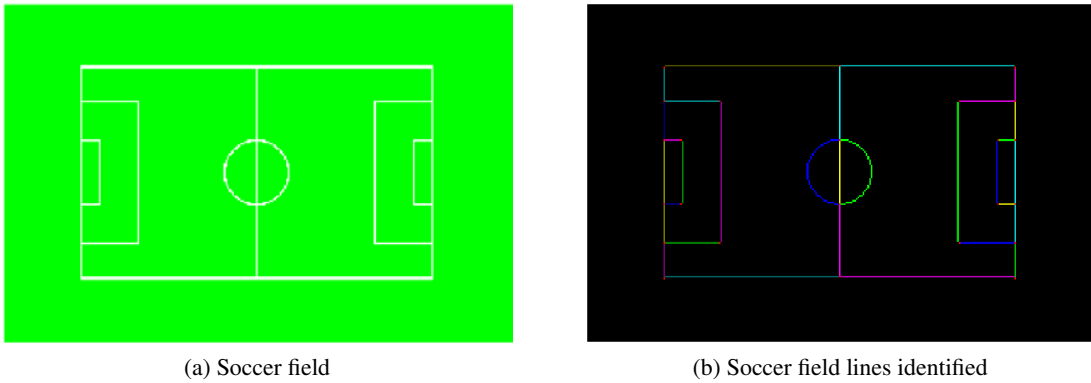


Figure 4.1: Example of a soccer field and the result of lines detection and classification.

To use the field lines in the calibration process, we need to extract some information from them. We define *lines* as the indivisible segments of field lines constituting the soccer field, and corners the points connecting two or more of those segments. Therefore, we need to extract the lines and match the extracted information with a soccer field model, in order to obtain a correspondence between the line extracted and a specific line of the model. For that purpose, we start by separating the white field lines from the rest of the image, obtaining an image with only the unidentified white lines. With the line extraction process, we intend to separate the white lines that form a whole (soccer field) into indivisible lines that connect each other at corners, as shown in Figure 4.1.

We define the type of a corner as a corner that connects a certain number of lines. Therefore, we can have several corner types: one-line corner connects one lines, 2-line corner connects two, and so on.

Moreover, we do not need to have line width of more than one pixel, as it does not add any valuable information to the process of identifying the lines in the system. For that reason, we apply a skeletonization algorithm to obtain one pixel width lines. Although we can use the wider lines' edges for the calibration, we can only do that with a few part of the lines - the ones in the region with good spacial resolution. Anyway, for the *lines* detection that is irrelevant.

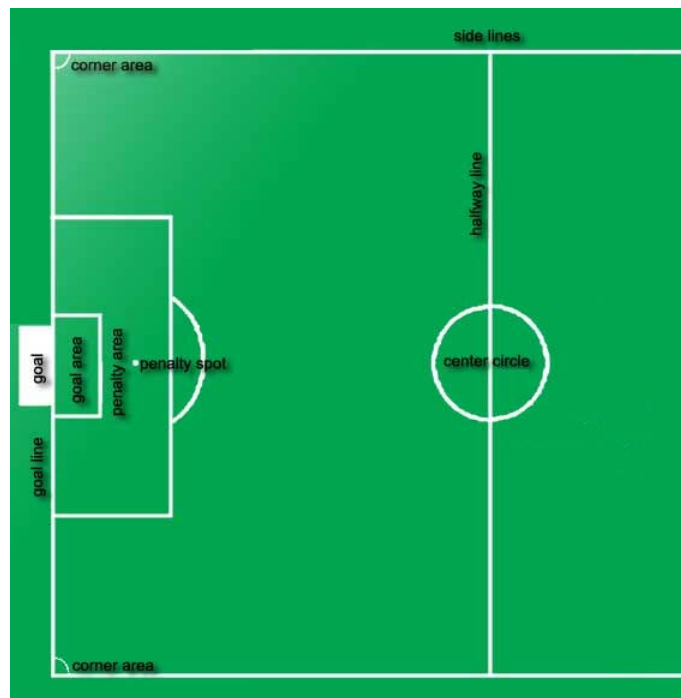


Figure 4.2: The soccer field lines.

## 4.1 Lines Detection

Using a common perspective vision system, the field lines in the image would still be linear, and so a simple Hough transform would be enough to detect the lines. However, using a catadioptric system, there is high distortion, making the field lines curve, and making it impossible to use this kind of algorithms.

A good approach to detect lines would then be to detect the corners first, and then the lines would just be the group of pixels connecting the various corners. A corner in this kind of images can be defined as the end or beginning of a line, the intersection of 2 or more curved lines, and an abrupt change of direction of a curved line - in this last case, there is an intersection of 2 lines, but the resulting image seems to be one unique line, because each pixel only has 2 neighbors, the previous and following pixel of the image line.

Firstly, we tried to use a corner detection algorithm based on the Harris method, but it does not have the expected result. As we can see in 4.3, many corners are wrongly detected, and with the downside of some other corners not being detected. For example, the intersection of the goal line with the side line is not detected, because the change of direction is not that abrupt. This can be explained by the distortion caused by the system, increasing the curvature of the lines in this region. Another problem occurs in the center circle, caused by the natural curvature of this line, which is curve even in the world reference. To overcome this problem, a conic fitting method [2] could be used. However, this would not be an appropriate solution because it assumes SVP systems, which may not be the case. Besides, some of the lines, specially in the peripheral regions of the image, may be too short to make an appropriate fitting. Moreover, this would not properly

detect the center circle, and so a separate method would be needed for that purpose.



Figure 4.3: Corner detection using the Harris method.

#### 4.1.1 Main Method

We then propose a method for detection of lines, that uses as base assumption the fact that, even in omnidirectional vision systems, the rectilinear lines are locally linear in the image.

With this approach, the base technique consists of selecting a line pixel and mark it has a corner (beginning of line), then searching for the nearest pixel corresponding to a white line. Only the adjacent pixels that belong to a white field line are considered neighbors. Furthermore, we save in memory the latest direction obtained - position of last pixel minus position of previous one, normalized in order to have absolute value of one. Using the last  $k$  saved directions we calculate the average to estimate the local direction of the line. This can then be used to forecast the position of the following pixel of the line. Not finding a pixel in the expected position, other neighbors of the last pixel are searched and then the nearest one to the estimated direction and corresponding to a white line is selected.

In case of finding an intersection of undetected lines, the method continues following the direction closest to the previous one. Finally, when finding one pixel with no more neighbors, it is marked as end of line. All the detected pixels of that line are saved in a new matrix - with the same size of the original image - with a line ID  $x$ . When the end of line is reached, and a new line begins, the new found pixels are registered with the line ID  $x + 1$ . The line ID number starts at 1.

All lines are also marked in an adjacency matrix *lines*, with the nodes being the beginning and end of lines. For example, if line  $m$  connects corner  $i$  to corner  $j$ , then  $lines(i, j) = m$ . However, this representation proved not to be enough, because in some cases there are more than one line connecting the same pair of corners. For example, there are two corners in the center circle, and connecting them there are three lines: each half-circle, and the halfway line. To take this into account, we just need to add a third dimension to the adjacency matrix, from now on called *layer*. Thus, to save the information we would instead use  $lines(i, j, l) = m$ , with  $l$  being the first empty layer. In any case, the multi-layer solution is also important for a robustness point of view, because even in situations in which there should not be more than one line between two corners, that may

```

1  function Line_detection–MainMethod
2  %builds 5 matrices:
3  %–checked: attributes for each image pixel a line ID
4  %–corners: attributes for each image pixel a corner ID
5  %–lines: adjacency matrix, with nodes=cornersID and values=lineID
6  %–lines_len: matrix with size of each line
7  %–corners_order: information of direction of intersecting lines
8  for Each_image_pixel(i)
9      if( Pixel(i)_is_undetected_field_line )
10         while(1)
11             W=Direction_calculated_using_line_last_directions_between_pixels;
12             for W : further_directions_from_W
13                 if(Neighbor_pixel(j)_is_first_belonging_to_undetected_field_line)
14                     Saves_direction;
15                 end
16             end
17             if No_line_pixel_found %(no pixel j)
18                 if Actual_line_is_empty (only isolated pixel (i) detected)
19                     break;
20                     %no line detected – pixel (i) does not belong to any line
21                 else
22                     Invisible_line_detection; %using direction W
23                 end
24             else
25                 Pixel(i)_is_marked_as_line_point;
26                 Pixel(j)_is_marked_as_NextPixel;
27                 % continues to next while(1) cycle from NextPixel
28                 Update_of_(lines_len)_matrix;
29             end
30         end
31     end
32 end
33 end

```

Figure 4.4: Meta-code for the main method of lines detection

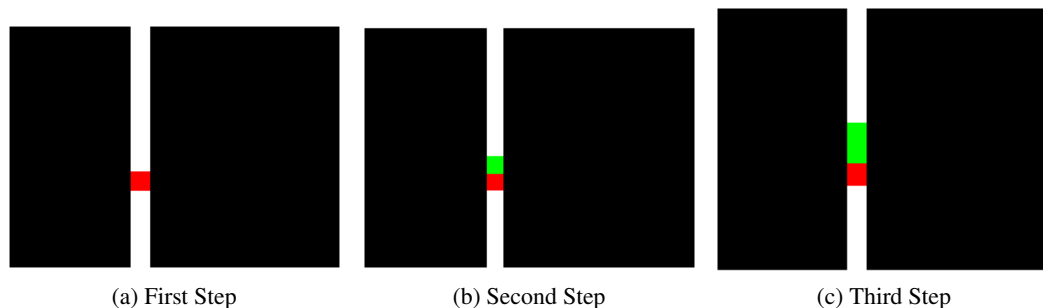


Figure 4.5: Example of line tracking used in line detection.

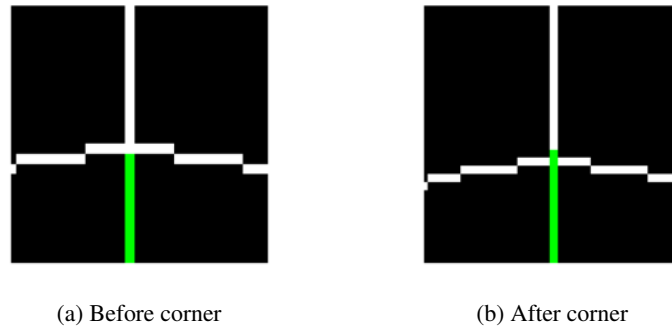


Figure 4.6: Behavior of main line detection method - line tracking - when passing by a corner (lines intersection). It is shown the line tracking tries to maintain the actual line direction.

happen due to a temporary incorrect detection of lines, that may disappear when deleting lines (which will be explained below). Finally, when using the adjacency matrix one should take into account that the order of the corners is irrelevant, and so one can either save the information in both places,  $lines(i, j, l) = m$  and  $lines(j, i, l) = m$ , or using only one of the triangular parts of the matrix, which can be achieved with  $lines(\min(i, j), \max(i, j), l) = m$  or  $lines(\max(i, j), \min(i, j), l) = m$ .

When following a line, it can intersect an already detected line. In this case, the intersection point is saved as a new corner, and the intersected line is separated into two lines. On the other hand, if the intersection occurs in a corner, this one will then be considered the end corner of that line.

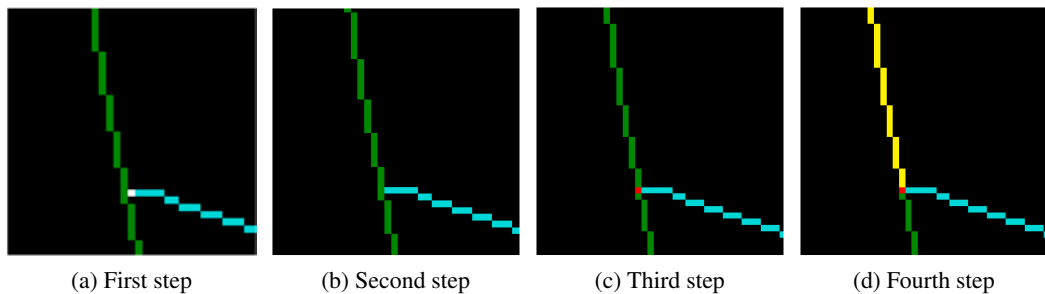


Figure 4.7: Result of lines intersection when following a line

Another approach was tried, in which the corners were detected when following a line. When using this method, whenever a point had more than two neighbors it would be considered a corner. Moreover, even when the pixel had only 2 neighbors, depending on its positions it could be considered a corner. This approach generally had good results, but in image regions with lower spatial resolution it would cause some errors, because sometimes it would incorrectly assume the existence of corners.

Besides saving the corners ID information in a matrix, we also save the information of each corner in a different format, a vector with size equal to the number of corners. In each position there is a structure with information about the corner, as its pixel location, and the direction from

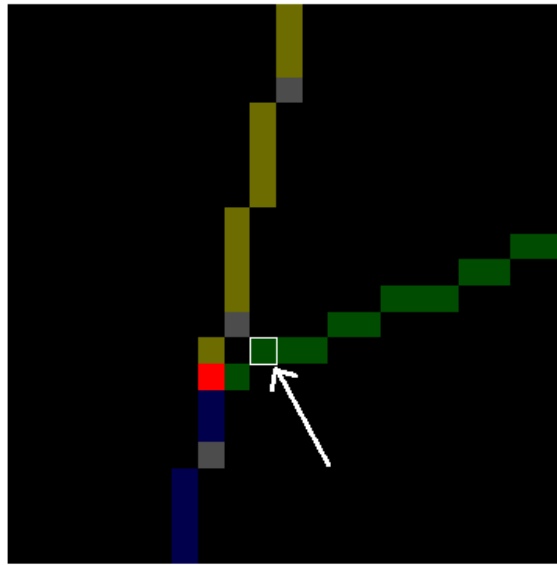


Figure 4.8: Example of situation in which a corner would be wrongly detected.

which the lines connected to the corner depart from it. This information is useful because it provides faster access to the position of the corner, the direction from which the lines depart it, and the number of lines intersecting themselves at that point, information that proves to be interesting for another posterior operations.

Another feature of the developed algorithm is being able of detecting dashed lines. This is quite important considering in many occasions the lines in the peripheral region of the image, due to the low spatial resolution, have a one-pixel width, or even less. This means sometimes the line will be interrupt, and so its essential to develop a method that can cope with this problem. Again, it uses the last  $k$  directions between pixels, and with that estimating the local direction of the line. Then, we just search in that direction for any pixel corresponding to a line. We start at the nearest neighbor of the last pixel of the line, and then move pixel by pixel until reaching a point of a line. For each step, not only the one pixel in the estimated direction is checked, but also the two closest neighbors in that direction. Although this method had results good enough, it would be an interesting idea to used the curvature of the line to estimate further curvature of the local direction along the invisible part of the line. Each time a pixel has no neighbors, this process is used until a pixel is found or a maximum number of invisible pixels is reached ( $n \times k$  pixels). In this last case, the invisible line is ignored and the last visible pixel is considered the end of line. If it encounters a still non checked pixel, it proceeds in the process of following the line as described previously. In case it is a pixel of another line or a corner, it must proceed as described previously in a similar situation. The invisible points can be seen in Figure 4.14 as grey pixels, and the visible points are represented with color. Moreover, the corners are represented with red pixels.

The invisible points do not strictly correspond to a field line point, otherwise it would be a white point in the image acquired by the sensor. However, using this concept it is useful to keep track of lines in regions where the spacial resolution is low. It is important to say that both  $n$  and  $k$



```

1 function Invisible_line_detection
2   %There is no line pixel in the closest neighborhood so we
3   %search further in the actual direction (W) from pixel (i) in order
4   %to continue to detect line, even if dashed
5   while( Distance_without_line_points < Threshold)
6     %In each step, we search the pixel closest to the actual line
7     %direction (Pixel1), and the two closest neighbors (Pixels 2 and 3).
8     for each_Pixel (1 to 3)
9       if pixel_is_undetected_field_line
10        Case_1; break;
11      elseif pixel_is_corner
12        Case_2; break;
13      elseif pixel_is_other_line
14        Case_3; break;
15      end
16    end
17    %still no line detected
18    Next_pixel=Pixel_1;
19  end
20  if Case_0 %no more points detected in that direction
21    Pixel(i)_marked_end_corner_of_actual_line;
22    Update_of_matrices; %lines ,corners ,lines_len and corners_order
23    break;
24  elseif Case_1
25    Pixel(i)_is_marked_as_line_point;
26    New_detected_pixel_is_marked_as_NextPixel;
27    %New detected pixel is last Pixel 1/2/3 belonging to field line
28    Invisible_pixels_of_dashed_line_marked_as_line;
29    %marked differently from visible line points
30    Update_of_matrices;
31    % No break – continues to next while(1) cycle from NextPixel
32  elseif Case_2
33    Pixel(i)_is_marked_as_line_point;
34    Corner_detected_is_end_corner_of_line;
35    %Corner detected is last Pixel 1/2/3 belonging to field line
36    Invisible_pixels_of_dashed_line_marked_as_line;
37    Update_of_matrices;
38    break;
39  elseif Case_3
40    Pixel(i)_is_marked_as_line_point;
41    Detected_line_pixel_is_marked_corner;%intersection of 2 lines;
42    %Detected line pixel is last Pixel 1/2/3 belonging to field line
43    Invisible_pixels_of_dashed_line_marked_as_line;
44    New_corner;%method separates intersected line in two;
45    Update_of_matrices;
46    break;
47  end

```

Figure 4.9: Meta-code for the detection of dashed lines

were chosen manually, as the framework must be taken into account. For example, the resolution of the camera, as the overall distortion, may have some impact on how locally the direction should be determine.

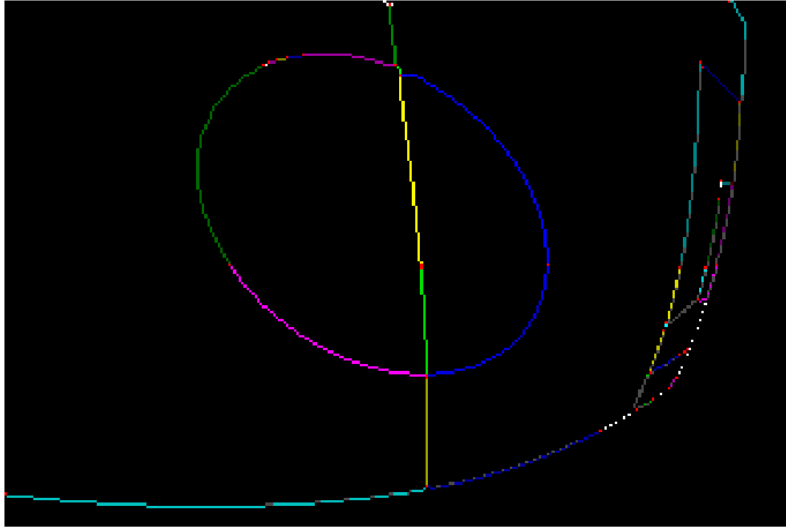


Figure 4.10: The result of applying the main method of line detection to a soccer field image

#### 4.1.2 Post-processing Methods

Using this information, and after running the base method to find individual lines, it is possible to, for example, easily merge lines. If there is a corner connecting only two lines, it can mean it is only one line. Therefore, when those situations occur and lines depart the corner with directions with an angle between them of at least 90 degrees, then the two lines will be merged into only one. We chose a threshold of 90 degrees, possibly eliminating correct corners, because we want to eliminate all possible wrong corners, as in the end a more specific function to estimate corners position will be used. All the corners are saved in a matrix with the same size of the original image, being registered with a corner ID that starts at one, and increases when a new corner is found. Besides storing the information presented previously, we also save the length of lines, more specifically, the total length and the number of “invisible” points that make up part of the line.

The merge situation occurs some times because it is common for a detected line to start in the middle of the real line - only with luck it will start in the beginning of the line, usually starting at a random point of the line.

Furthermore, when a line is detected, it starts in a point and follows the line in one direction. However, the opposed direction must be checked too. Therefore, after checking all pixels corresponding to a field line, we check which lines start at a corner without any other connection besides the line in question. This may mean that opposite direction needs to be checked. Thus we start following the line from the corner and calculate the first  $k$  detected directions, then using the average as the value of the line local direction at the corner. After that, we use the method to

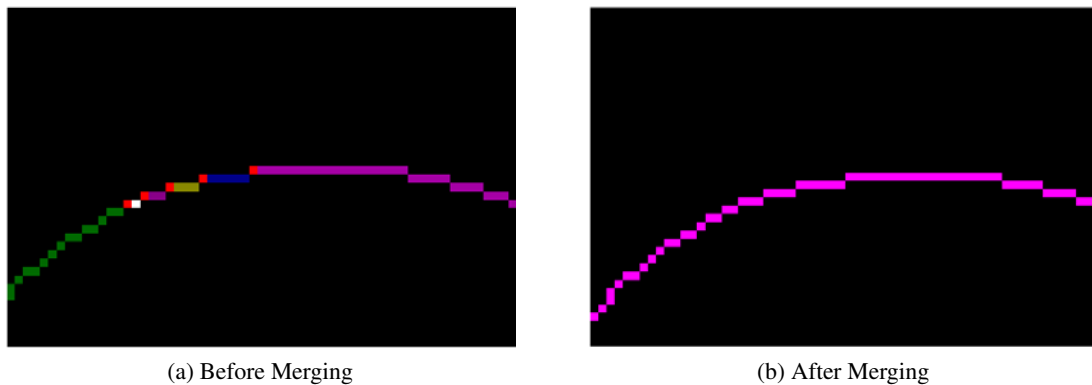


Figure 4.11: Merging multiple lines into one

```

1 function Extend_lines
2 for each_corner
3     if corner_is_extremity_of_only_1_line %corner does not connect lines
4         W=local_line_direction;%at extremity – the actual corner
5         Invisible_line_detection; %using direction W – extend line.
6     end
7 end

```

Figure 4.12: Meta-code for the extension of already detected lines

complete dashed lines to search for new points from the corner in the opposite direction previously estimated.

```

1 function Delete_lines
2 for each_line
3     if (Percentage_of_Invisible_point_of_line > Threshold )
4         Delete_line_from_matrices;
5         %checked , lines , corners , corners_order , lines_len
6     end
7     if (Line_size==1 and One_line_extremity_does_not_connect_other_lines)
8         Delete_line_from_matrices;
9     end
10    if (Two_corners_connected_by_two_lines , and_corners_are_adjacent)
11        Delete_larger_line_from_matrices;
12    end
13 end

```

Figure 4.13: Meta-code for the elimination of lines mainly composed of invisible points

After applying these methods, it is also important to check if there is lines completely, or almost completely made up of invisible points. This is not a desirable situation, because it would mean non existing lines are being wrongly detected, although it may be truly a line which posing

in a low definition area. To detect this kind of situations we use the ratio  $(length_{inv}/length_t)$ , which represents the percentage of line that is invisible ( $length_{inv}$  is the number of invisible pixels of the line, and  $length_t$  is the total number of pixels of that line).

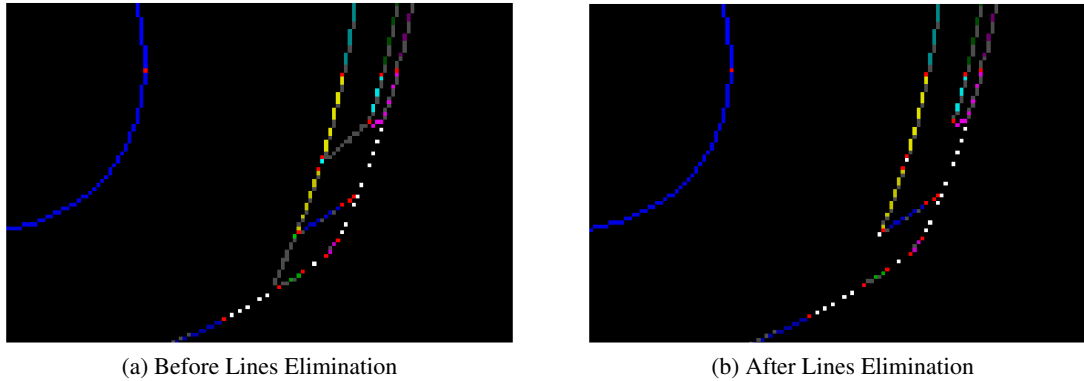


Figure 4.14: Elimination of lines with almost all points invisible

The threshold used was chosen manually after several different trials. However, either an heuristic calculation or a more comprehensive examination of its effect should be done in future work. Furthermore, its value is not expected to be optimal to all systems, and so it must be properly chosen for each application.

```

1 function New_lines
2 for each_pixel_not_belonging_to_any_line %isolated pixels
3   for r=1:MaxRadius %search the neighborhood of the pixel
4     for ang=0:(8*r-1) %search the neighborhood of the pixel
5
6       Isolated_pixed_coordinates=[ ci cj ];
7       Dir=round([ cos(ang/(8*r)*2*pi) sin(ang/(8*r)*2*pi)]*r);
8       if Pixel[ci+Dir(1),cj+Dir(2)]_is_corner_or_line
9         Stop_search_neighborhood;
10      elseif Pixel_is_another_isolated_pixel
11        Invisible_line_detection; %using direction Dir
12        Stop_search_neighborhood;
13      end
14    end
15  end
16 end

```

Figure 4.15: Meta-code for the creation of lines using checked but unclassified pixels

Another final processing method used in this approach analyzes all checked points that could not be assigned to any line. For each of them it searches the nearest point in the same conditions, at the same time checking if there is any line point or corner in between. If there is a clear region connecting them, a new line is registered in the respective matrices.

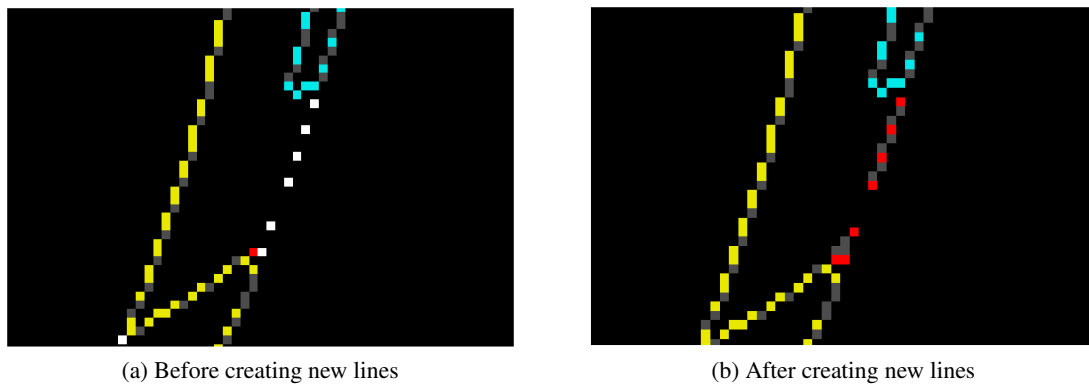


Figure 4.16: Creation of lines from unclassified pixels

Another approach would be to try to connect that kind of point to any closer point with a free path, but that was not used because it would cause the creation of lines that do not really exist, which could happen due to the strong distortion the image undergoes. For example, in the peripheral region of the image, a point not recognized in the goal line between the goal posts would be easily connected to a goal area line, instead of being connected to another similar point in the goal line. Due to the reduced spacial resolution in that area, the goal line and goal area lines would be quite close, possibly closer than points in the same goal line, this because we are considering that is a dashed line. However, only connecting points in the same condition, we are connect two near points with a free path in between, which are certain to belong to field lines that were not detected in that region. Consequently, there is a high probability that those points are part of the same undetected line.

The last post-processing functions cannot be executed in a random order, which would cause this method to not work properly. Therefore, we start with the new lines creation method (connecting unidentified line point), a process that creates always a completely invisible line. After, we use the expansion method (expands the line in the opposite directions it departs its extremities). This has to be the second method, in order to expand the newly created lines. In this phase it is still possible to lines with only invisible points, and so we cannot delete lines at this point. Thus we follow by merging the lines interconnect each other without intersection of other lines, and only after that we remove the invisible lines. This cycle of operations is run until there is no changes in the lines detected. Furthermore, the order of the operations applied must be this one, as it is essential for the method not to enter an infinite cycle.

After deleting corners or lines, those are never used again, staying as temporary elements that were deleted. This is used for the simplicity of the process, as this way we only have to increment the IDs when we want to create a new line or corner, and not search for available ones.

Finally, we apply a method to detect corners in the middle of detected lines, because of abrupt changes of the local line direction. This is the more sensitive part, due to the existence of non-linear parts - center circle - in the soccer field, which may have a change of direction along the line similar to the one present in corners in highly distorted regions. One example that shows this

```

1 function Line_detection-PostProcessing
2 Delete_lines;
3 Extend_lines;
4 if Possible_to_merge_lines
5     Merge_lines;
6 end
7 for 1:2
8     New_lines;
9     Extend_lines;
10    if Possible_to_merge_lines
11        Merge_lines;
12    end
13    Delete_lines;
14    if Possible_to_merge_lines
15        Merge_lines;
16    end
17    while(Changes_are_made)
18        Add_corners(Patameters1);%uses few line pixels in the detection
19        %high threshold -> clear corners detected
20    end
21    while(Changes_are_made)
22        Add_corners(Patameters2);%uses more line pixels in the detection
23        %lower threshold -> less clear corners detection
24    end
25    Merge_lines_spec;
26    %merge center circle lines if separated in two
27 end

```

Figure 4.17: Meta-code for the post-processing methods used in line detection

```

1  function Add_corners
2  % – Local horizon (SS) – the line length used to
3  %calculate local direction
4  % – Corner detection threshold (TH) – the minimum peak amplitude
5  %to consider existance of corner
6  % – Curvature threshold (TH2) – the curvature value to distinguish
7  %rectilinear and curvilinear lines
8  for each_line
9      Dir=Line_Direction_between_pixels;
10     %searches all the line pixels from one extremity to the other, saving a
11     %vector with all the changes in direction from pixel to pixel.
12     for each_viable_Dir_position
13         Dir1=Direction_of_previous_SS_points_of_line;%unitary 2d vector
14         Dir2=Direction_of_follwoing_SS_points_of_line;%unitary 2d vector
15         ChDir=arccos( Internal_product(Dir1,Dir2) );
16     %ChDir is a vector that represents, for each line point, the change of
17     %direction as an angle (curvature)
18     end
19     [val ,p]=max(ChDir);
20     %the position (p) of the maximum curvature is a good estimate
21     %of a possible corner position val is the maximum curvature
22     mc=Mean_curvature; %after extracting all curvatures in the
23     %maximum peak region (p-SS/2:p+SS/2)
24     val=val-mc;%to identify a corner the interesting value is the peak
25     %amplitude in relation to the mean value
26     if (val>TH && mc<TH2)
27         %high probability of corner existance in a rectilinear line
28         Corner_position=round(ss/2)+(p-1);
29         Estimated_corner_position_is_marked_corner;
30         New_corner;%method separates line in two;
31         Update_of_matrices;
32     end
33 end

```

Figure 4.18: Meta-code for the detection of corners in already detected lines

effect is the angle in the image between two rectilinear lines, depending on the image region. In regions where at least one of the pixel coordinates  $(u, v)$  is close to the distortion center (which may be different from the image center), the angle between image those lines is approximately 90 degrees. On the other hand, in regions far from the center, the angle is gradually bigger than 90 degrees. In typical catadioptric images with a wide vertical field of view, the angle can reach angles as big as 135 degrees.

In the region where the side line intercepts the goal line, there is a corner and the local directions in each direction is different, but in the remaining part of the line the local tangent to the line is relatively constant and approximately zero, specially considering a local domain of only some pixels of the line. However, after the last explained phases, this two lines will in principle be merged. Therefore, the change of direction of tangent to the line will be approximately zero, aside from a peak in the corner position. On the other hand, in the center circle the direction of the line tangent will constantly change along the line, thus resulting in an approximately flat but different from zero change in direction, without any relevant peaks in the direction change. Consequently, this seems to be a good approach to detect corners in the middle of lines. Considering that, for each line we move from pixel to pixel estimating the average direction of the line fragment with  $k'$  ahead, and the same for the fragment with  $k'$  behind. This means we start searching the line  $k'$  pixels ahead of one of its extremities, and finish when missing only  $k'$  to the other end of the line. For each pixel position of the line we calculate the intern product of the two estimated tangent directions (ahead and behind), which result is  $\cos(\theta)$ , with  $\theta$  being the angle between the two directions.

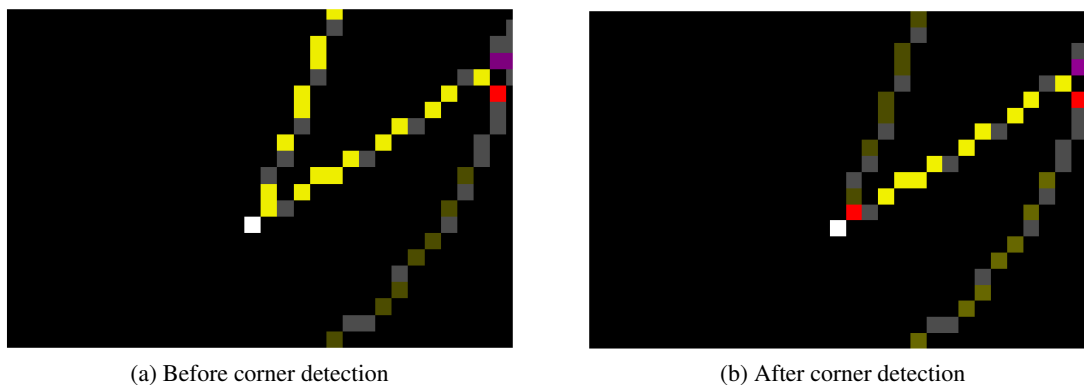


Figure 4.19: Detection of corner in middle of line, for cases in which the corner position is clear

Then we could analyze the result of the internal product along the line, which would be approximately one in rectilinear areas, and lower than one around corner positions, which would be a considerable different value minimum. In the curvilinear lines case, the product would be always different from one, but without any relevant minimum. However, the internal product is not the chosen feature to estimate corners, as it is highly non-linear in relation to the angle, which gives a better and relevant understanding of the direction changing of the line. Thus we use the inverse trigonometric function of cosine to calculate the angle. Having applied this transformation, we



just search for the maximum angle, which will be the corner position. Furthermore, it is easy to understand that this maximum will not be an abrupt peak, and the angle will start increasing as soon as the corner enters the a direction estimation line fragment. Hence, after finding the peak we will eliminate the  $k'$  surrounding points in each side and then calculate the mean angle of the remaining points. The peak value minus the average angle calculated before will then be compared with a threshold value in order to decide if the point is a corner or not.

This approach is only theoretically valid, and in the real case there are distortions of the lines, caused by the catadioptric distortions themselves, but also due to skeletonization, and low spatial resolution in certain image areas. Therefore in the practical cases the corners and the curves are not so easily distinguishable, needing a well chosen threshold to separate both cases. A further improvement that can be done in the future is use the region of the image to change the threshold of corner detection

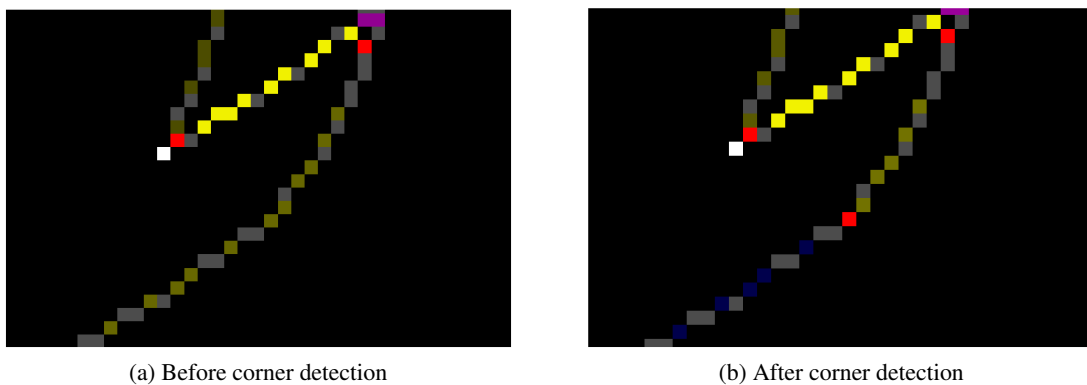


Figure 4.20: Detection of corner in middle of line, for cases in which the corner position is unclear

When estimating the local direction of a line at one of its extremities, we use several pixels of that region and calculate the average of all pixel directions relative to the previous line pixel. Another approach to calculate that local direction would be to use only the first and last pixels of the line fragment. But, although we assume the line is linear in that region even considering the catadioptric distortion, there may be some curvature. In that case using the average of pixel directions is a better estimate of the line local direction in the point, although not being the optimal value due to the existence of curvature. It is important to note that sometimes the existence or not of a corner may not be observable using this kind of feature, which may provide only information of the probability of occurrence of each state (corner or no-corner). Therefore, in some range of values of the feature used there may be an overlap of probability densities of the two states, making them not mutually exclusive, and therefore not perfectly distinguishable. The feature threshold used is then chosen in a way that, instead of reducing the error obtained, only tries to optimize it by making it minimal. That would not perfectly distinguish the two possible states, but would be a solution that reduces the error of misclassification of each state. This problem brings the subject of statistical pattern recognition, which could be used in order to improve the overall results. Using more features to clearly identify the states, and training classifiers to optimally decide of

the existence of corners, is probably an approach worth investigating in the future. This method is used multiple times, until there is no difference between the input and the output. We do this because after separating a line in two, it may be possible to find corners in each of the new lines created.

```

1 function Merge_lines_spec
2 for each_corner(i)
3     if Possible_to_merge_lines
4         %corner connects lines 1 and 2
5         dir1=Direction_at_one_extremity_of_line_1;%not corner i
6         dir2=Direction_at_one_extremity_of_line_2;%not corner i
7         dir3=Direction_of_line_1_at_corner(i);
8         dir4=Direction_of_line_2_at_corner(i);
9         if(dir1'*dir2>Threshold_1 and (dir1'*dir3>Threshold_2
10            or dir2'*dir4>Threshold_2))
11             Merge_lines;%lines 1 and 2
12         end
13     end
14 end

```

Figure 4.21: Meta-code for merging curvilinear (center circle) lines, wrongly separated in two

Lastly, we implemented a procedure that merges lines in certain specific conditions. The purpose of this method is to avoid center half-circle lines from being separated in two new lines, which may be caused by the last described functions. More specifically, this method tries to find lines that are curvilinear in both the distorted image and the real world (center circle) and were previously separated in two due. To some extent, this method and the last one, corner detection, have similar objectives, as both try to correctly estimate corners positions. While the first one creates corners, this one tries to find wrongly created corners and eliminate them. Although seeming to have opposite effects that cancel each other, that does not happen because the techniques used in each one are different, and this last one serves as a refinement of the other. In a certain way, it works as a second feature used to detect corners, as described previously in reference to pattern classification.

In this special case merge situation, the merging constraints formerly explained still apply, but a new condition is added, based in the following reasoning. If we analyze the local tangent directions of a line at its extremities, we conclude that in a half-circle those directions are parallel, and in a rectilinear line they are opposed. Nevertheless, in the case of a catadioptric system image there is some distortions that cause each of this statements not to be always true, hence the 90 degrees threshold is used to separate both cases. To apply this idea we check the 4 local directions of the possible lines to merge. The relation between the direction of the lines extremities further from each other, albeit bringing useful information, is not enough to distinguish a wrong corner in the middle of a semicircle line from correctly placed corners. This can be proven considering this result applied to a corner separating perpendicular lines, which may be even more than 90

degrees in a distorted image. For that reason, to detect corners in semicircles we also use the two extremities direction difference of each line. In a perpendicular lines, it will be approximately 180 degrees in each line, but in the semi-circle one of the fragments will still have a high curvature, and so the angle difference will be much less than a straight angle. With this approach only highly distorted rectilinear lines will be misclassified, and the corner will be wrongly deleted. It is important to clarify that this technique will only work in cases the line is separated in two. If there is more than one wrongly placed corner in the semicircles, it might fail. Yet that is not probable to happen, as after placing a first corner in the semicircle, a second one is not expected. This comes from experience dealing with this kind of images, that shows the semicircle region with the most abrupt change of direction - what happens specially due to image distortion - disappears creating the first wrong corner, therefore highly reducing the possibilities of a second one.

This approach is rather inefficient, deleting and creating the same corners multiple times, and running relatively big blokes of code just to check there is no change in the result after applying some post-processing functions. After some tests using *Matlab*, we confirm that fact analyzing the run time, which will often surpass 5 seconds. Obviously, this time depends from image to image, thereby depending on the specific catadioptric systems used. Thus, it would be a good idea to improve this aspect in the future, making the code more efficient without using it blindly, but rather using an approach that analyzes the last results in order to know which operations still need to be done. One simple approach for that purpose would be to rerun the operations only in the lines that were affected by the previous methods.

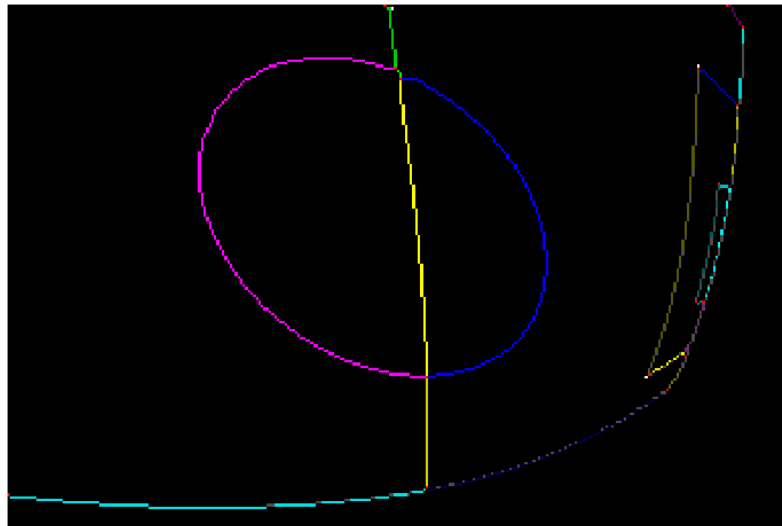


Figure 4.22: The result of applying the complete line detection procedure to a soccer field image

## 4.2 Lines Identification

After extracting all the indivisible lines from the original image, we have some matrices that give us some information about the structure of the lines network of the soccer field image, the most

important one being the lines adjacency matrix. After this step, we need to match it with a soccer field. A possible solution would be to use adjacency matrix matching, as described in [25, 26]. However, this methods, albeit being a more general and probably a more interesting approach to the problem, would only use part of the information we have, with a higher probability of failing the operation. Besides, we must take into account that in this problem we do not seek a good match, but a perfect one. If we misclassify a line in each image, it will have a enormous effect in the calibration process. If that happens, the mapping function will be optimized so as all points of that line will be belong to another one, creating a completely wrong map in that region . Because of that, it is preferable not to match all lines, but do it correctly for the matched ones. Consequently, we use our knowledge of the structure of the soccer field in order to match the lines.

Firstly, we detect the center circle, assuming it is observable from every possible robot position in the field. Although this assumption's truth depends on the vision system itself, it is expected the robot is able to see at least half of the field. The central region of the field is important to be extracted, as this lines provide calibration information for the center and middle radial region of the image, considering a varied set of robot positions. Besides, the successful extraction of this lines information makes easier the process of recognizing the other lines, in other words, goal line, goal area lines, and penalty area lines. This is true because we already known certain points correspond to that lines, only needing to recognize which one is each line/corner, otherwise we would still need to identified the corners and lines from a full set of options, with all field lines still undetermined. To achieve that goal, the functions in the previous section must be tuned so as the center circle is always detected, even at the cost of not recognizing some corners.

To determine the semicircles, every line is analyzed regarding its extremities tangent directions (Figure 4.24). We then assume that semicircle lines, besides being visible in the image, are the ones with the greater similarity between extremity directions, meaning its internal product is the bigger and closer to one - the direction vectors are normalized to absolute value of one. These two lines are supposed to intercept each other at their ends. However, due to skeletonization effects, it may not happen as shown in Figure 4.25.

For that reason, we have to find which extremity of each line is closer to another. If the semicircles intercept each other, the lines connecting the extremities of two semicircle lines, they are marked as corners. But if there is no line connecting them, then it is not the center circle. This may happen if the center circle line is divided in more than two lines. Therefore, in this special case we apply a merging procedure to correct that situation. Then, if the center circle is correctly extracted, the algorithm proceeds to the extraction of other lines, otherwise aborts the extraction procedure. Afterwards, the mean pixel position of those points will be used as the calibration point correspondent to that corner. Then, the halfway line is marked too, and its interceptions with the side lines are only marked if the extremities of the halfway line are corners connecting 3 lines, otherwise those corners cannot be the intersection between those lines.

At this point, all identified lines will be deleted from the adjacency matrix, so the other lines can be easily extracted.

One relevant aspect to consider is the orientation of the field. In localization for example it

```

1 function Line_Identification
2 for each_line(i)
3     dir1=Direction_at_one_extremity_of_line_i;
4     dir2=Direction_at_another_extremity_of_line_i;
5     dir(1,i)=dir1 '* dir2;
6 end
7 % max(dir) – line with greater curvature – center circle line
8 Extraction_of_center_circle_lines; % using vector dir
9 Definition_of_field_orientation_using_one_of_center_circle_lines;%line 1
10 Identification_of_line(1)_corners;
11 %accordingly with world x and y axis convention
12
13 %corners of center circle lines may not intersect each other, due to
14 %skeletonization. In that situation, they are connected by small lines.
15 Identification_of_closest_corners_of_2_center_circle_lines;
16 Identification_of_halfway_line;
17 %after identification of center circle, and considering line
18 %detection in this region of the field is correctly done, the halfway
19 %line identification is an easy and direct process
20
21 Cor=Detection_of_all_3-line_corners;
22 %detection of corners that connect 3 lines, except the ones
23 %intersecting halway and side lines
24 for a=1:2 %for each side of the field
25     Cor1=Grouping/Ordering_of_four_3-line_corners;%vector with corners
26     %of goal lines. ordering must be accordingly with world x and y axis
27     Detection_of_corners(Cor1)_side_in_the_field;
28     Identification_of_goal_and_penalty_area_lines;
29     %After this, goal and penalty area lines are identified.
30     %On that side, the only lines missing identification are the ones
31     %intersecting the field corners – intersection of goal and side line.
32     if penalty_and_goal_areas_detected
33         Identification_of_repective_field_corners_and_goal_line;
34         %An easy and direct process after the last one.
35     end
36 end
37
38 if field_corner_detected
39     Identification_of_respective_side_lines;
40 elseif halwayline_detected
41     Identification_of_side_lines;
42 end

```

Figure 4.23: Meta-code for identification of the detected field lines

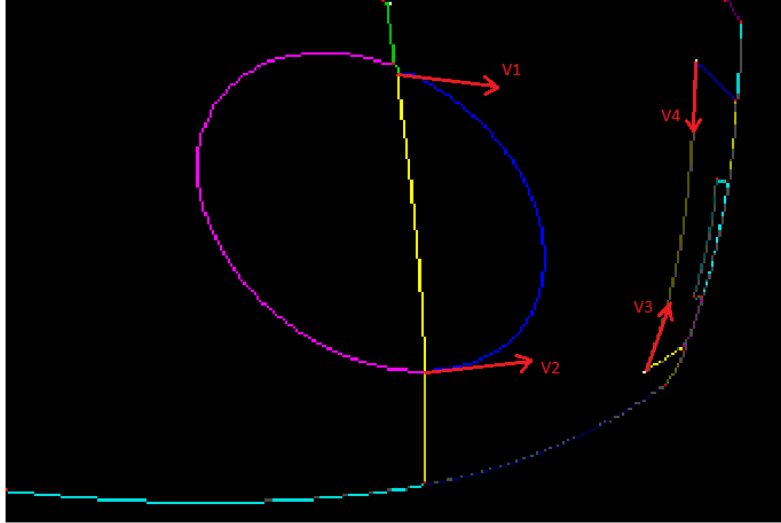


Figure 4.24: This shows how we can detect the center circle lines, using the inner product of lines' extremities local direction. For curvilinear lines the inner product is maximum (V1 and V2), for rectilinear lines it is minimum (V3 and V4).

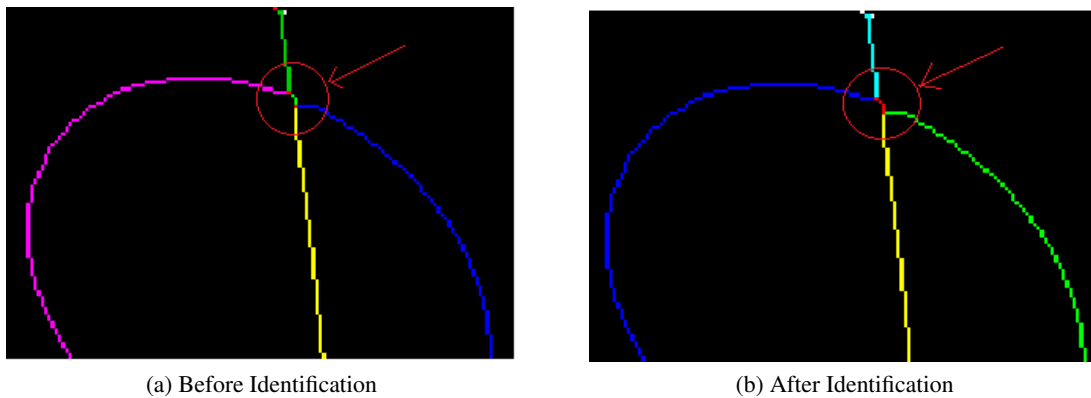


Figure 4.25: Due to problems in the lines thinning process, extremities of center circle lines may not intersect each other. The nearest corners are found, and the line connecting them is considered a corner (red pixels) after lines identification.

is important to know which side the robot is looking at, and for that usually soccer robots use a compass. However, in the calibration process that does not matter, which is proved with the following: considering the kickoff mark as the  $(0,0)$  position, in regard to the  $x$  and  $y$  coordinates, an image obtained from a given  $(x_1, y_1)$  position with robot orientation  $\theta_1$  is equivalent to the image obtained from  $(-x_1, -y_1)$  and orientation  $-\theta_1$ .

We will show this has no effect in the calibration procedure, meaning we can attribute a semi-circle line to any one of the sides. Another question to take into account is which corner is at each extremity of the identified semi-circle line. For that purpose we must consider that with the current setup used, the camera points upwards, thus the image obtained will have one of the world axis inverted. Therefore, considering a world coordinate system in which the  $z$  axis points upward, in the resulting image we have this axis pointing downwards, as if the camera was taking a shot from under the field. Therefore, having the center circle lines detected, we estimate the  $x$  axis position, and then we choose the corners so as the  $z$  axis points downward (Figure 4.26).

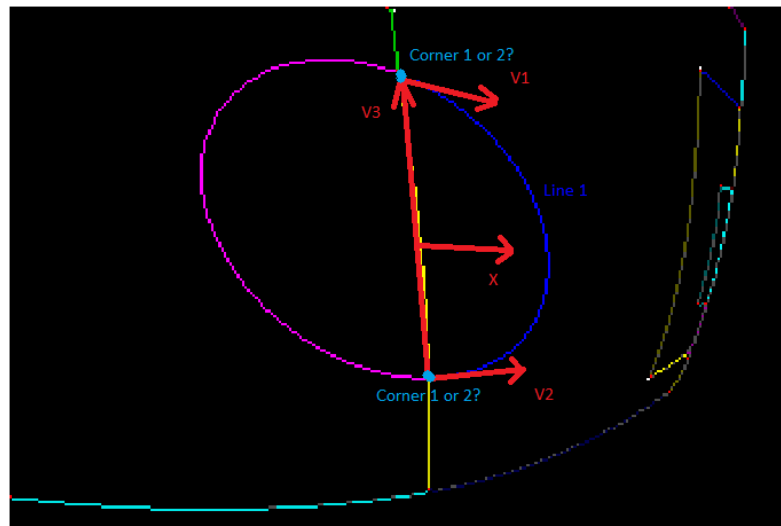


Figure 4.26: This shows how to identify each extremity of Line 1 (line is identified, corners are not). Using the line's extremities local direction ( $V1$  and  $V2$ ), we calculate the average and use it as an estimation of the  $x$  axis orientation. Then we use the cross product of estimated  $x$  and  $V3$  (estimated  $y$  axis) to determine  $z$  axis direction. If the result is not the expected, we know  $V3$  is inverted, then clarifying which corner (1 or 2) corresponds to each extremity of Line 1.

Another problem of skeletonization causes a simple corner connecting 2 lines to be mistaken as a 3 lines connection. However, this problem is solved by detecting and eliminating the small and incorrectly created lines from the adjacency matrix. To detect them, we just search for small lines that do not connect anything in one extremity, and have a 3 line connection in the other.

Then, the only expected remaining corners connecting 3 lines are the interceptions between penalty and goal area lines and goal line. Considering both sides of the field, this adds up to 8 possibilities, although we do not need all of them to proceed with this method. In each side, this type of corners are connected to each other, but only with one connection between them. Using

```

1 function Identification_of_goal_and_penalty_area_lines
2 for each_corner(a)_of_Cor1 %Cor1 is vector with 3-line corners
3   %of one goal line , properly ordered
4   for other_corners(b)_connected_to_corner(a)_through_line
5     if (corner(b)_is_not_present_in_Cor1)
6       for other_corners(c)_connected_to_corner(b)_through_line
7         if (corner(c)_is_not_present_in_Cor1 and corner(c)_not=_corner(a))
8           for other_corners(d)_connected_to_corner(c)_through_line
9             if (corner(d)_present_in_Cor1 and corner(d)_not=_corner(b) )
10              if (corner(d)_position_in_Cor1 ==(corner(a)_position_in_Cor1+1))
11                Identification_of_goal_area_lines_and_corners ;
12              elseif (corner(d)_position_in_Cor1 ==(corner(a)_position_in_Cor1+3))
13                Identification_of_penalty_area_lines_and_corners ;
14            end

```

Figure 4.27: Meta-code identification of penalty and goal area lines and corners

the example of Figure 4.28, we can have a vector with all 3-line corners

$$Cor = \begin{bmatrix} 12 & 6 & 9 & 55 & 44 & 23 \end{bmatrix} \quad (4.1)$$

In this case too we select the corners from one side, ordering them alongside the inverted y axis.

$$Cor1 = \begin{bmatrix} 23 & 44 & 12 & 6 \end{bmatrix} \quad (4.2)$$

After that we need to determine which side this corners belong to, using the direction of the already detected lines (as explained in Figure 4.28, if inner product is positive it is at the same side as the semicircle used to calculate it, otherwise it is in the opposite side).

So, considering the vector with the 3-line corners, we start checking which of them are connected, and putting them into a vector within they are in the order they are connected in reality. After running that process one time, all the visible 3 lines corners of one side will be extracted and put into order. After that, we check each corner to see if it connects other one that is not part of the vector, itself connecting a second corner not belonging to the corners vector, which finally connects a new corner from vector that vector, forming the characteristic structure of penalty and goal area lines. To distinguish these two, we just have to check the index difference in the vector between the first and last corners used. If the difference is one, then the goal area was detected, contrariwise, if the difference is three, it corresponds to the penalty area. A better understanding is accomplished when observing figure 4.29. In each of the mentioned structures the second and third corners detected (b and c) may be 3 line connecting corners, a problem due to the skeletonization process explained previously. In that case, the wrongly detected lines are marked as part of the corner to which they are connected.

If the penalty area is detected, then the method proceeds to the field corners detection. There



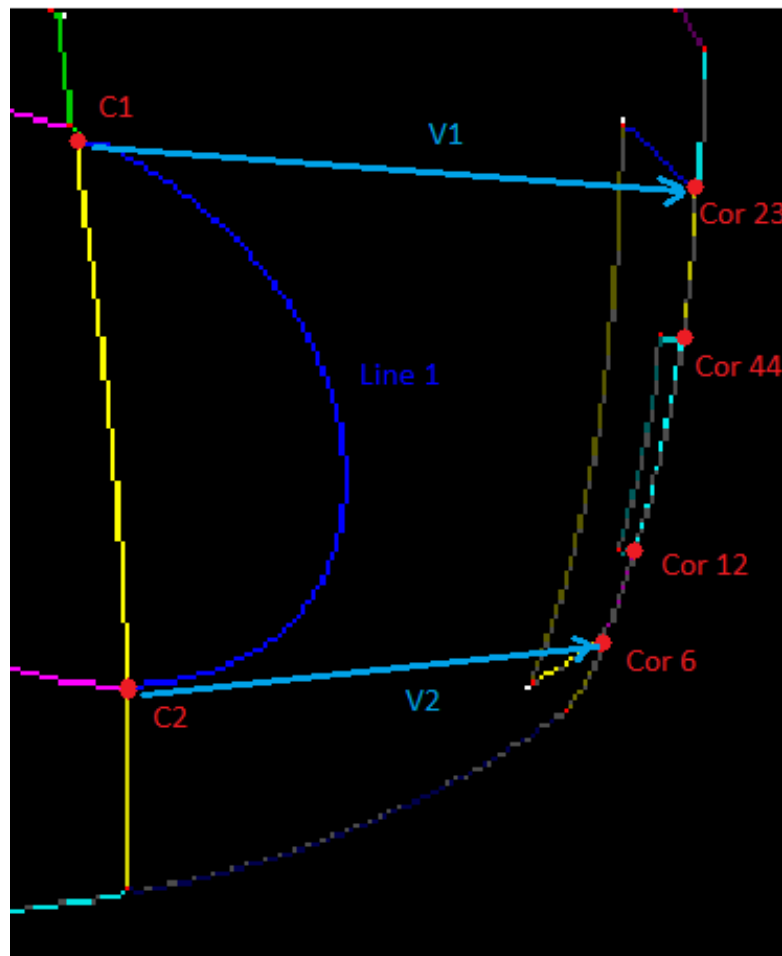


Figure 4.28: Using the already identified corners  $C1$  and  $C2$ , and the ordered 3-line corners, we can calculate  $V1$  and  $V2$ . Using their average, we use the inner product of that average with the estimated  $x$  axis direction (4.26). Analyzing the signal of the result obtained, we can distinguish the side we are dealing with.

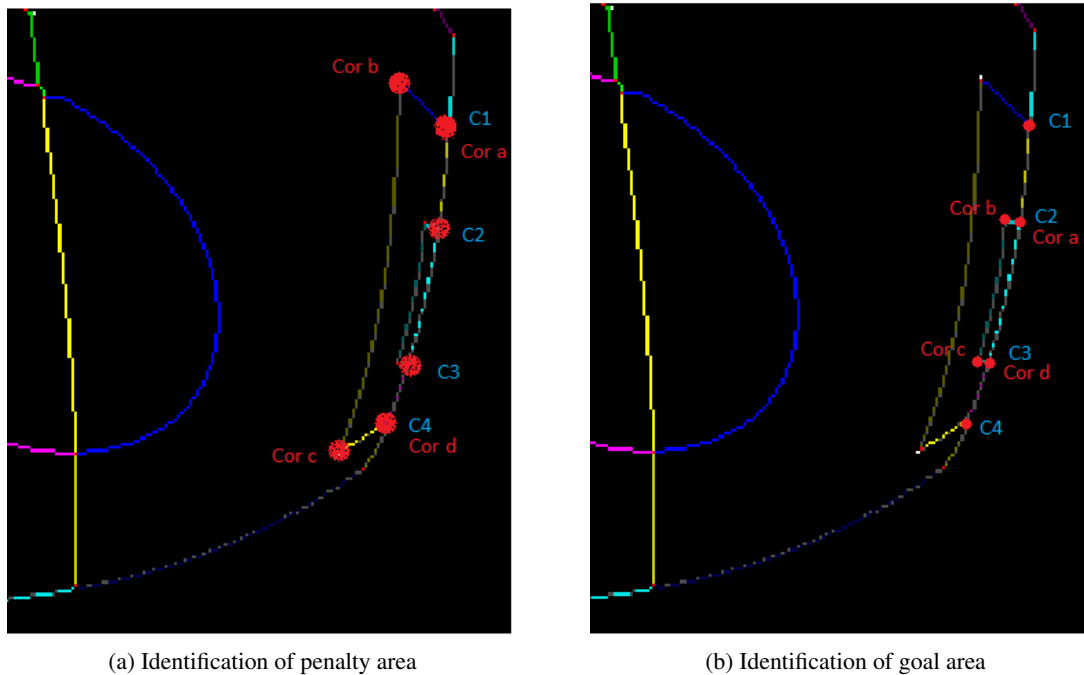


Figure 4.29: For identification of penalty and goal areas, we detect corners a, b, c and d. Then we differentiate them using corners a and d, and checking if they are C1 and C4 ((a)), or C2 and C3 ((b))

may be several possible cases in this situation. If the line ends in an one-line corner, then it matches that line with the goal line. If it is another type of corner, there is two types of situations: being the sideline and halfway line interception corner, but also being a wrongly detected corner or another corner because field corner was not detected.

If the side lines were not detected with the last described method, and any corner interception with the halfway line exists, then the side lines are matched using the halfway line extremities.

In Figure 4.30 we can observe the final result after applying the complete identification method (using image simulated for the non-ideal sensor). In this image we can see all lines were correctly classified.

In figures 4.31 we can see the result for the ideal sensor case, and we show that even with different robot positions the lines identification performs well.

Finally, it is possible to conclude from Figure 4.32 that field lines are not always perfectly identified. However, if identified, they tend to be well classified.

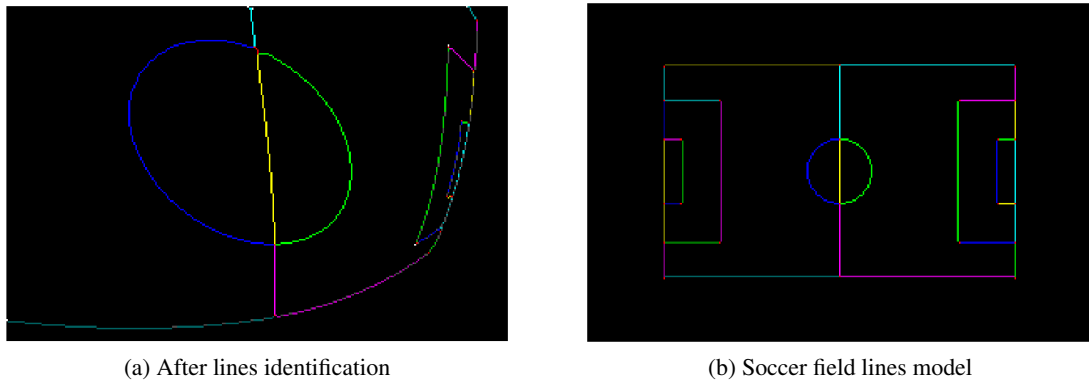


Figure 4.30: Result of lines identification using non-ideal sensor in (a). If compared with (b), we see the colors are the same, meaning all lines are well classified.

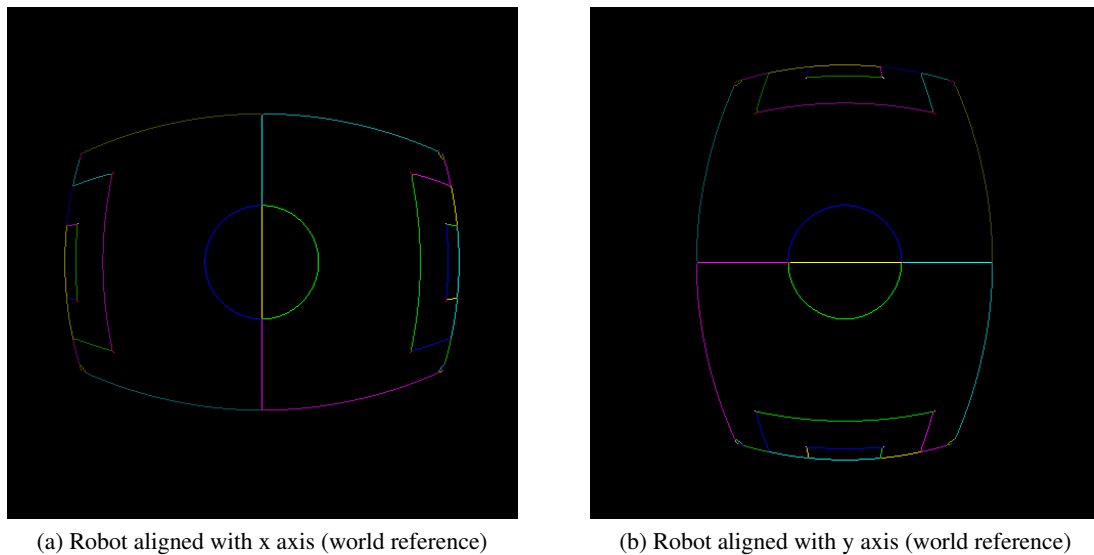
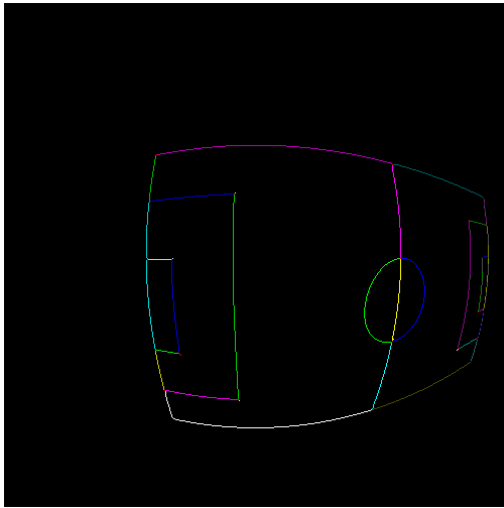
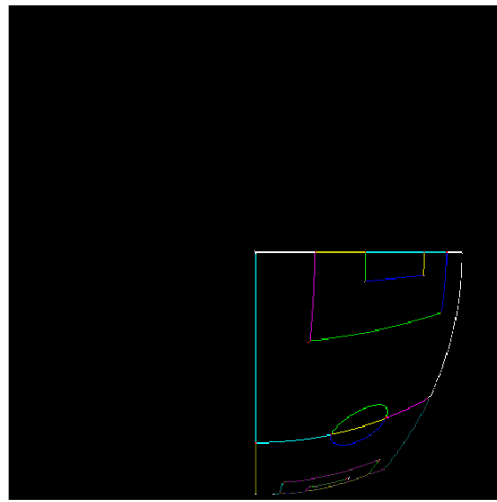


Figure 4.31: Result of lines identification. If compared with 4.1b, we see the colors are the same, meaning all lines are well classified.



(a) Image with some lines not identified



(b) Image with some lines not identified

Figure 4.32: Result of lines identification. If compared with 4.1b, we see the colors are the same when lines were identified (not white), but sometimes not all lines are identified.

## Chapter 5

# Parameters Estimation

After identification of field lines we have the information of all extracted lines and corners. We then use an optimization process which estimates the parameters of a function from the data obtained previously.

This method could use linear or nonlinear optimization, although linear methods are desired due to lack of initial parameter values and local minimum problems when using non-linear approaches. While for parametric calibration this is not as important, because with some basic knowledge of the setup we can have good estimates of the parameters, with non-parametric methods we use a function to map pixels to ground points, and so we cannot have a good estimate of its parameters only based on the robot and vision system characteristics. Therefore, considering we use non-parametric calibration, the linear optimization is even more desirable than most cases. Consequently, a least-square approach to estimate the parameters of a polynomial function, as shown in Chapter 3, is the desired optimization method.

### 5.1 Optimization process

While corners give us complete information - both ground coordinates  $x$  and  $y$ , lines only provide part of that information. For example, each image pixel of a identified rectilinear line brings information of one of the ground coordinates,  $x$  or  $y$ , but not both. On the other hand, curvilinear lines as center circle correspond to  $(x,y)$  points such as  $(x - x_0)^2 + (y - y_0)^2 = r^2$ . However, using this coordinates to describe the information retrieved is not useful, as it brings non-linearity. Therefore, for these lines it would be preferable to use polar coordinates, as it could describe the information in a similar way as it was done with rectilinear lines, making  $\rho$  a known variable and  $\theta$  unknown.

However, we can not use both coordinate systems in one linear optimization, and so we have to choose one of them and discard the lines using the other. Obviously, we keep the rectilinear lines, as there are much more of this kind in a soccer field. The information lost because of not using the center-circle can be compensated with the usage of multiple images with different robot positions in the field. Anyway, the multi-image approach would always be necessary due

to the intrinsic characteristics of non-parametric calibration. While using parametric techniques, only some few points of image might be used, as they provide enough information to estimate the model parameters. Nevertheless, with non-parametric calibration we estimate a function that maps pixels to ground points, meaning only the pixel regions used in the optimization process will be a good estimate of the mapping, and the other regions will only be extrapolated by the mapping function.

Besides, there is also a problem related to the order of the system. Due to strong image distortion, the order of the polynomial approximation has to be relatively high to make a good estimate of most regions of the image. However, with a higher order, there are two consequences. Firstly, there is an increase in the function oscillations, which means the estimation is correct in the calibration points, but there might be great deviations from the mapping in the other regions. Therefore, calibration points must be well distributed in the whole image. This also relates to the second aspect, that is the need of more calibration points to be used in order to estimate a higher order polynomial. This aspects, in turn, also relates to the need of using various different images in the optimization process, to overcome that problems.

Considering the previous explanations, we estimate two mapping functions: one for the pixel  $(u, v)$  mapping to the ground  $x$  coordinate, and a second mapping  $(u, v)$  to the ground  $y$  coordinate. While the first uses corners and lines with exact world  $x$  information, the second uses lines retrieving  $y$  coordinate information, plus corners.

Moreover, with this approach we use the ground coordinates, not estimating the robot-centered mapping that is useful for localization and many other applications, with  $(0, 0)$  being mapped to the field kickoff mark. To create the desired mapping with the robot center corresponding to the  $(0, 0)$  point of the mapping and the robot front aligned with the  $x$ -axis, we only need to apply a  $z$ -axis rotation (world reference) and a translation to the previously obtained mapping - the world reference to robot coordinate system homogeneous transformation. For this we need to estimate the direction and position of the robot in the field, which is described in Section 5.3.

Then, after line extraction we have some points for modeling the two pixel to ground coordinates correspondence. We want to estimate  $x$  as function of  $u$  and  $v$ , and  $y$  as function of  $u$  and  $v$ . For each calibration point  $l$  (calibration points for  $x$  and  $y$  may be different), we have

$$\hat{x}_l(u_l, v_l) = \sum_{i=0}^O \sum_{j=0}^i \beta_{ij} u_l^{i-j} v_l^j = \sum_{k=1}^m \beta_k \phi_k(u_l, v_l) = \sum_{k=1}^m S_{lk} \beta_k = S_l \beta \quad (5.1)$$

$$\hat{y}_l(u_l, v_l) = \sum_{i=0}^O \sum_{j=0}^i \alpha_{ij} u_l^{i-j} v_l^j = \sum_{k=1}^m \alpha_k \phi_k(u_l, v_l) = \sum_{k=1}^m S_{lk} \alpha_k = S_l \alpha \quad (5.2)$$

where for each  $\beta_{ij}$  corresponds a parameter  $\beta_k$ ,  $O$  is the order of the polynomial function, and  $m$  the total number of parameters.  $S_l$  is a line vector, with  $S_{lk}$  being its elements, and  $\beta$  is a vector with all the parameters  $\beta_k$ .

As we have the real values for each calibration point,  $x(u, v)$  and  $y(u, v)$ , we can estimate  $\beta$  and  $\alpha$  parameters. From now on we only demonstrate how to calculate  $\beta$ , but the same applies to  $\alpha$ .

In order to find the coefficients  $\beta$  that fit the known data best, we use the quadratic minimization approach [24]

$$\hat{\beta} = \arg \min_{\beta} J_x(\beta) \quad (5.3)$$

where the objective function  $J_x$  is given by

$$J_x(\beta_x) = \sum_{l=1}^{N_p} |x_l - \hat{x}_l|^2 = \sum_{l=1}^{N_p} \left| x_l - \sum_{k=1}^m S_{lk} \beta_k \right|^2 = \|x - S_x \beta\|^2 \quad (5.4)$$

where  $x$  is a vector (size  $l$ ) with the  $x$  coordinate of each calibration point,  $S_x$  is a matrix with  $l$  lines, each being  $S_l$ , and  $N_p$  is the number of calibration points.

Given this formulation of the problem, and considering  $\hat{y}$  and  $\hat{x}$  functions are linear combinations of partial functions, we can use a linear method to estimate the parameters - linear least squares optimization. The solution in this case is then given by

$$\hat{\beta} = (S_x^T S_x)^{-1} S_x^T x \quad (5.5)$$

For that purpose we used *Matlab* method *lscov*, which instead of equation 5.5 uses the QR factorization (slower than the normal equations method but more numerically stable), resulting in

$$R\hat{\beta} = Q^T x \quad (5.6)$$

Being  $R$  an upper triangular matrix and  $Q$  an orthogonal matrix such as  $S_x = QR$ . The solution can be easily found with backward substitution because  $R$  is upper triangular.

This minimization problem has a unique solution, provided  $S_x$  columns are linearly independent, which is true with the chosen functions if the calibration points are well distributed in the image plane.

One useful approach would be to estimate the reliability of the information extracted from lines, and use it in a weighted linear least squares optimization, minimizing the weight of less reliable observations.

However, we still have to consider that in each calibration image the point correspondence is done using the world coordinates, and so we have different mappings for each image due to different position of robot in the field (for example, using different images, different pixels will correspond to the ground position  $(0,0)$ , which must not happen). To overcome this problem and be able to use all images in one optimization process, we must use the extrinsic coordinates to convert the points of every image to the same coordinate system, such as all obtained points belong to one unique mapping function. Therefore, choosing the first image as reference, we must use the relative translational and rotational difference between this image and the others to transform the ground points correspondence of the second image to the first image reference.

For example, if a second image is obtained with a change in orientation of  $\theta$  degrees, and the robot position distance to the one of the first image is  $(t_x, t_y)$ , then to every calibration point of

the second image  $(x', y')$ , we will apply rotational and translational transformation to those points, symmetrical to the transformations relating the two robot coordinate systems.

The problem of this approach is the following: points that previously had one known and other unknown variables ( $x$  or  $y$ ) could be used to improve one of the mappings and not the other, but now, after the coordinates transformation, the same point corresponds to the following line  $(x\cos(teta) + \lambda\sin(teta) + t_x, -x\cos(teta) + \lambda\sin(teta) + t_y)$ , which does not bring any additional information that can be used in the linear optimization method proposed.

Nevertheless, if we assume the robot only changes orientation with multiples of 90 degrees, then the known and unknown variables interchange and we can use it in the calibration. Although we have this constraint, it does not mean the robot has to be aligned with  $x$  or  $y$  axis of the world reference, only that robot position has to have a 90, 180, or 270 degrees rotation relative to the position of the image chosen as reference.

Considering this assumption, other problems arise. For example, we need to capture the calibration images with the robot in a specific position. And even considering that is possible, explained in section 5.3, we have to be able to detect what is the direction on each calibration image. For that, we just need to analyze the lines information provided by the methods described in last chapter. Considering each line detected we use its corners to create a vector. The field lines always have the direction of world reference  $x$  or  $y$  axis, but that is not possible to be accurately calculated in the image (lines are curvilinear). However, it is possible to calculate an approximate field orientation in the image. Although it is not precise, it does not matter, because we only have 4 possible classifications, and the orientation classification will always be correct if the image distortion is similar to the one found in common catadioptric vision systems.

To know the orientation of the field, we compare the field estimated orientation with the one that is the reference, using the inner product with both the orientation for the reference  $x$ -axis and also with  $y$ -axis

Using the absolute value, we chose the one with the greater value, meaning it is the closest direction. Besides, this can only distinguish between perpendicular direction, but we can also use the signal of the inner product to distinguish between opposite directions.

Therefore, with this we show that the orientation detection is not important in the line extraction methods, as long as the robot positions are correspondent to the chosen orientation.

Besides, with this methodology it is possible to have information of various images for one pixel. Instead of using it multiple times, which would give more importance to that pixel than the others, we calculate the average for all that on each pixel, so it only counts once on the optimization, in order to have a distributed error in whole image, and not concentrated in certain regions.

## 5.2 Results

In Table 5.1 we have the results of this calibration methodology, with the polynomial degree ranging from five to fifteen. We also tested the procedure with two sets of images, one only with



two, and another with almost forty images. In this kind of images there is a strong distortion, and the true mapping is not polynomial. Therefore, as we are estimating the mapping using a polynomial function, we need a high order polynomial in order to have a function that fits well the true non polynomial mapping. So, the more calibration points used the better, because with more points we can estimate better a higher order polynomial function (especially if the points used belong to regions that require a higher order function).

Moreover, using Table 5.1 we can see there is no need to increase the order of the system indefinitely, because from some point the order is excessive considering the number of calibration points used (overfitting), as discussed previously. In Table 5.1 we highlighted the optimal order for each calibration set. As we can see, the optimal order increases with the number of images used, what is expected because using more points (distributed in the whole image) we can apply a more complex function to fit the mapping.

Table 5.1: Calibration error of all points using polynomial function (cartesian coordinates) with various polynomial degrees and two sets of calibration images

Order	Two images		39 images	
	Maximum error (m)	Average error (m)	Maximum error (cm)	Average error (cm)
5	1.714	0.1687	100.1	9.15
6	1.757	0.1690	142.6	10.18
7	<b>1.043</b>	<b>0.0696</b>	51.03	3.09
8	1.792	0.0777	78.81	3.45
9	4.853	0.3074	<b>64.71</b>	<b>2.22</b>
10	14.19	0.5068	65.55	2.69
11	428.2	21.08	172.3	3.85
12	433.0	20.98	123.8	3.46
13	3298	139.6	300.3	4.39
14	4019	141.4	755.3	7.83
15	14196	567.6	1312.5	10.43

Comparing Table 5.1 and Table 5.2 we show the increase of error while increasing the order is due to overfitting. In Table 5.2 we only show the error of points used for calibration, and as expected it decreases while increasing the polynomial degree, besides being lower than when error is tested with all image points (Table 5.1). So, in Table 5.1 the error starts increasing because we test not only the calibration points error, but also the error in other regions of image, and from a certain order the function starts fitting too well the points used for calibration, although losing the power of extrapolation for other regions.

Moreover, it is interesting to notice that in Table 5.1 error is lower when using a 2-image calibration set than when using more images. This is easily explained, because in that table we only show the error in the calibration points. Therefore, when using the same function complexity, it is obvious the function will fit better less points, especially when all of them are in the central part of the image - which has less distortion. So, when using 39 images and the same order, there is more points in the periphery of the image. It is more difficult to fit the true mapping on those

points, thus having greater errors. However, when using more calibration points the extrapolation is better, and so we show that true error (Table 5.1) is smaller when using more images.

So, we conclude we should use more calibration images in order to get better results when increasing the number of parameters (increasing the order of the estimation function). Nevertheless, we were expecting a big difference in the results of the two image sets, as one has a lot more images than the other. However, the optimal order is only slightly higher, and the maximum error in the second is not even half of the first (considering the optimal polynomial order). Using the second set, with a reasonable number of images, and choosing a 9<sup>th</sup> degree polynomial function, the average error is just above two centimeters. As in Chapter 3, the maximum error is 70 centimeters, but it is the error at the frontier, and even points further from the robot - but not in the frontier of calibration - have an error that is only fifteen percent of the maximum. Therefore, the field is well estimated in most regions of the image, and so we consider the calibration successful.

Table 5.2: Error of calibration points using polynomial function (cartesian coordinates) with various polynomial degrees and two sets of calibration images

Order	Two images		39 images	
	Maximum error (cm)	Average error (cm)	Maximum error (cm)	Average error (cm)
5	8.59	0.91	19.28	4.27
6	9.03	0.90	19.12	3.81
7	7.76	0.89	18.77	1.61
8	7.61	0.89	18.83	1.53
9	7.30	0.88	18.80	1.40
10	7.15	0.87	18.81	1.39
11	6.27	0.73	18.80	1.37
12	6.14	0.73	18.76	1.37
13	3.20	0.56	18.76	1.36
14	3.13	0.56	18.74	1.36
15	3.00	0.53	18.76	1.35

As in chapter 3, to analyze the distribution of the calibration error in the image we use two types of images: one showing the the error in each pixel as a percentage of the maximum error, using greys scale; the other uses the reprojection of pixels into world coordinates overlapped with the real image, showing an immediate feedback of the calibration quality. In the latter, we use a color code described on table 5.3.

Again, in order to know the quality of the calibration procedure, it is important to have information relative to the field size. In this case we simulated a field with 10m×6m, with line width of 6cm. Considering this and choosing an appropriate order for the estimating function, the calibration procedure can have relatively low error.

Moreover, we use again the concept of calibration images and test images. The latter ones are not used in calibration, only in the validation process that calculates the estimation error. The first ones are used to calibrate the system, and may be used or not to test the calibration, and so they can be part of the test images set.

Table 5.3: Color code of image reprojection

Color	Meaning
Black (periphery)	Rays from camera do not intersect the mirror model (not even with infinite radius), or in test images it corresponds to regions in which calibration is not tested
Blue (periphery)	Rays from camera do not intersect the mirror due to its limited radius
Red (periphery)	Rays from camera intersect the mirror, but direction of reflected ray points upward, not intersecting the floor
Black	Pixels reprojected into the soccer field lines
Green	Field pixels (not field lines) in the test image that do not reproject into field lines (pixel not used for calibration)
Pink	Field pixels (not field lines) in the test image that do not reproject into field lines (pixel used to calibrate y world coordinate)
Blue	Field pixels (not field lines) in the test image that do not reproject into field lines (pixel used to calibrate x world coordinate)
Red	Field pixels (not field lines) in the test image that do not reproject into field lines (pixel used to calibrate both x and y world coordinates)
White	Field lines in the test image whose reprojection is not a line (pixel not used for calibration)
Yellow	Field lines in test image whose reprojection is not a line (pixel used to calibrate y world coordinate)
Grey	Field lines in the test image whose reprojection is not a line (pixel used to calibrate x world coordinate)
Cyan	Field lines in the test image whose reprojection is not a line (pixel used to calibrate both x and y world coordinates)

It is interesting to note that when the order is two, the calibration result is quite poor (figure 5.1), even using the same unique image for testing and calibration. This means a second order polynomial function does not properly fit the catadioptric vision systems distortion.

However, when using order five the results get a lot better, even still using only one image for calibration. As we see in figure 5.2, when testing with the same image used in calibration, the result is good. However, using a test image with points in regions not calibrated, the fitting in those regions is quite bad again.

Nevertheless, if we start using the 2 images set in the fitting procedure, we see the error now is quite low in all the central region of the image, only increasing in areas further from the robot (figure 5.3).

We present in figure 5.4 the reprojection of field lines when calibration is done with the 2 images calibration set and fifth order function, confirming that in the central region the error is low (figure 5.4a), although it is visible the increase of error in non calibrated regions. Moreover, in figure 5.4b we see the furthest regions are not well calibrated.

Nevertheless, we conclude that using only two images and a 5<sup>th</sup> degree polynomial function the fitting in the central region is quite accurate, with good extrapolating properties (regions not calibrated are still a good approximation of the real mapping). Only the furthest regions are not

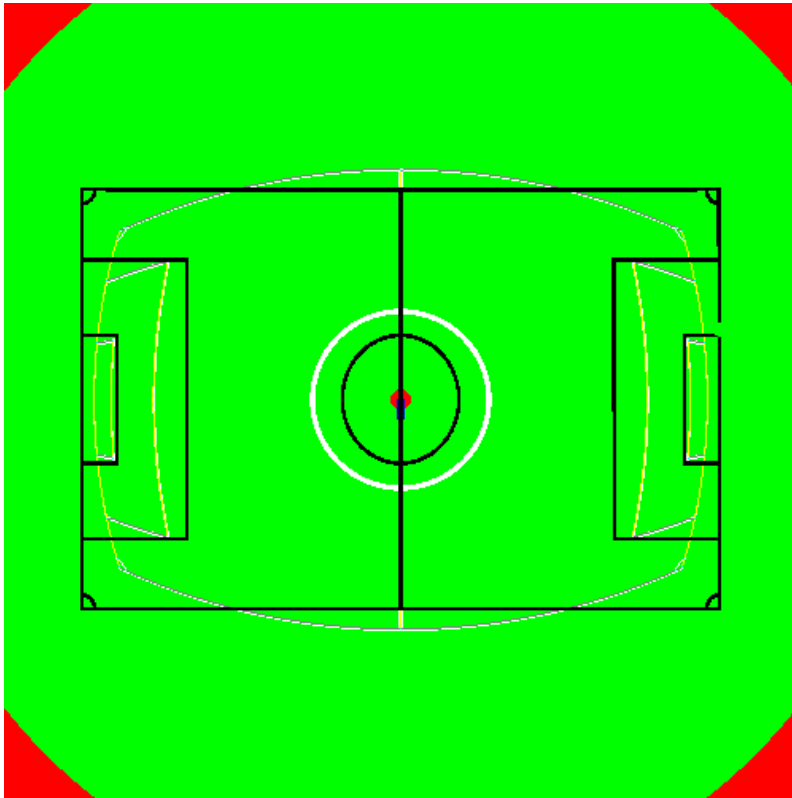
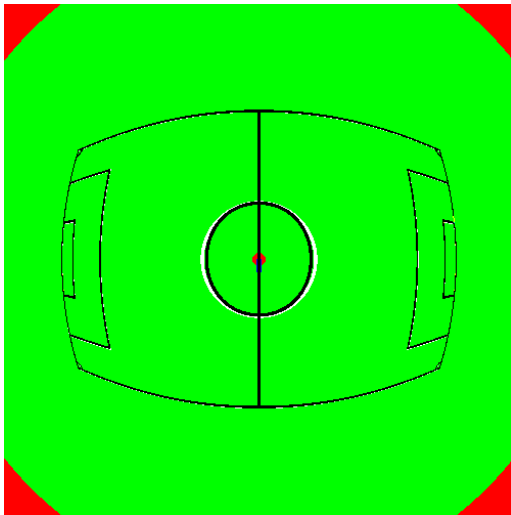
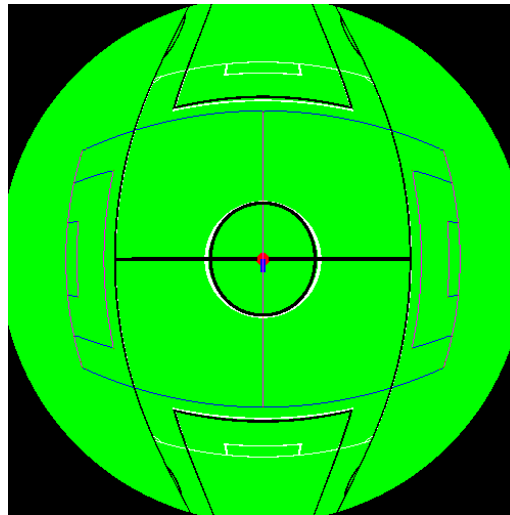


Figure 5.1: Calibration using only one image, a second degree polynomial for fitting, and the same image for testing



(a) Reprojection of test-image, which is the one used in calibration



(b) Reprojection, with different images being used for calibration and validation

Figure 5.2: Calibration using one image, a fitting polynomial function of degree 5, and testing the calibration error using: the calibration image for the testing (a), and a different image for the validation (b).

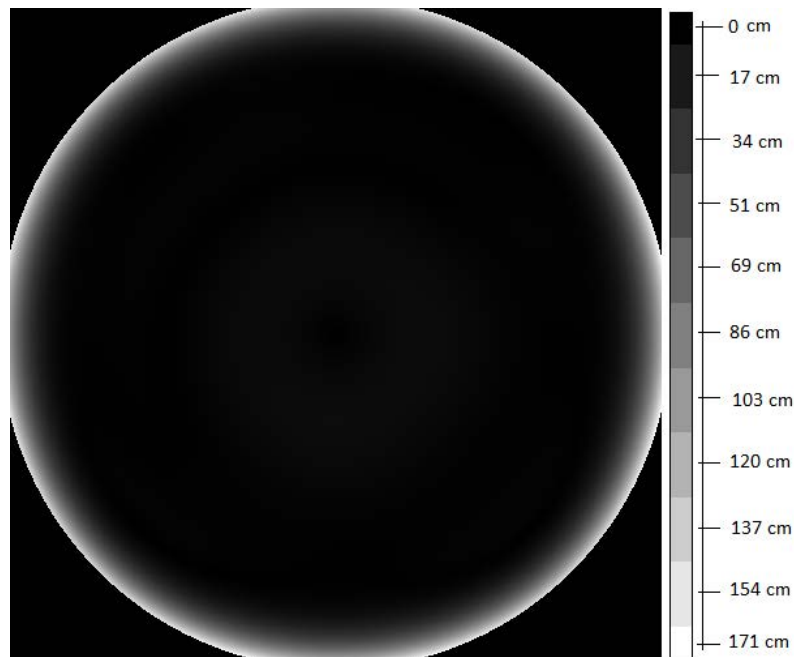
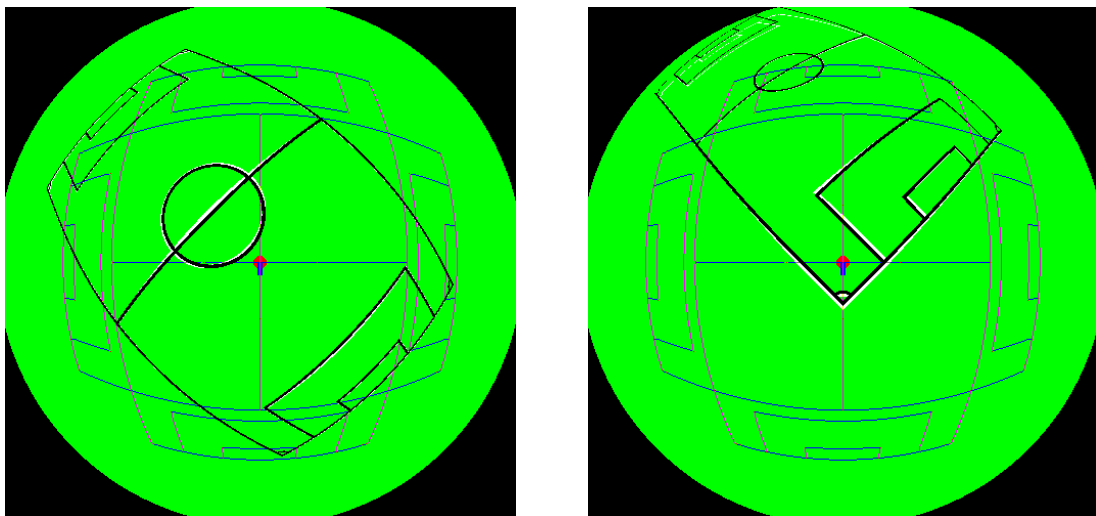


Figure 5.3: Calibration error when using two calibration images (robot in the central field position, with orientation equal do 0 and 90 degrees.), and a fifth degree polynomial



(a) Reprojection of test-image different from calibration images, but with robot position in the field similar to the one in calibration images

(b) Reprojection of test-image with robot position far from the position of the robot in the calibration images

Figure 5.4: Calibration using the two images set, a polynomial of degree 5, and testing it with two images different from the ones used for calibration.

well calibrated. However, we did not use calibration images for that region, and we show that using more images, specifically some that calibrate the further regions of the image, we can have even better results. For example, in figure 5.5a we have the robot near one of the field corners, but the opposite side of the field is still well calibrated, and we can see the error is almost null in every

region of the image (figure 5.5b), except some regions with less points used in the calibration procedure.

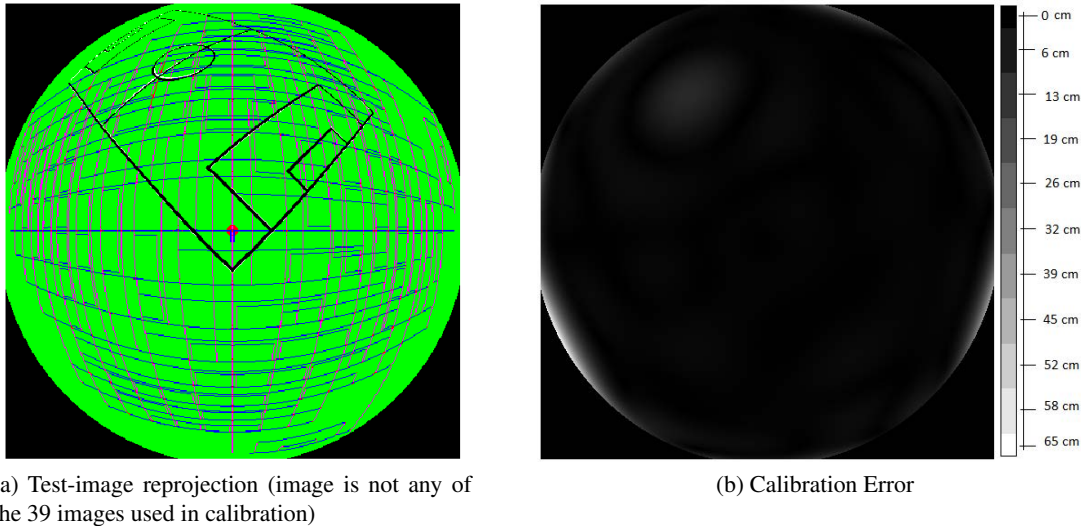


Figure 5.5: Calibration using 39 images, a polynomial of degree 9, and a test image not used in calibration.

With figure 5.4 we see we can calibrate well the central region with only two images and a  $5_{th}$  degree function. However, increasing the number of images (with the robot in different field positions and orientation) increases the calibration quality, and we have low errors in the important parts of the image. However, in order to achieve even better results, we suggest as future work the use of a spline with two functions, one to calibrate the central region and other for the furthest areas of the image. Therefore, with few images and still a low order we might be able to have a good calibration, because in this case we do not *force* one single function to be a good fit to every image pixels.

In Table 5.4 we present the average running times for the calibration process, using the 39 images set. As we can see the the whole method takes some time, being the lines detection procedure the one with the greatest contribution. However, Matlab is not the best framework for time efficiency, thus being important to export this methodology to another language if we want to reduce the running times.

In Table 5.4 we use a  $9^{th}$  degree function to calculate the parameters estimation times, because it gives the best calibration results (optimal order). The coordinates transformation method refers to the transformation that allows the use of multiple lines in the calibration. Besides the parameters estimation, in table 5.4 we present the time spent processing each calibration image. In Table 5.5 we present the parameters estimation times for the two image sets and varying polynomial degrees.

Table 5.4: Running times for the various constituent parts of the calibration process

Method	Average time (s)	Standard deviation (s)
Color Segmentation	0.820	0.089
Skeletonization	0.026	0.024
Lines Detection	12.53	1.26
Lines Identification	0.126	0.060
Coordinates transformation	0.427	0.027
Parameters estimation	0.592	-

Table 5.5: Running times for the optimization process considering 2 and 39 image sets, and varying polynomial degrees

Order	Running time with 2-image set (s)	Running time with 39-image set (s)
5	0.055	0.504
6	0.032	0.169
7	0.039	0.272
8	0.055	0.421
9	0.080	0.592
10	0.107	0.839
11	0.137	1.16
12	0.177	1.54
13	0.228	2.03
14	0.305	2.61
15	0.360	3.11

### 5.3 Calibration constraints considerations

To transform the mapping from world coordinate system onto the robot reference, we need to calculate robot position and orientation. However, instead of estimating the initial position, an easier approach would be to constraint it to a known position, possibly a corner, with the user positioning the robot. To estimate the robot orientation, there is only an approach possible, due to the possible misalignment between camera and robot orientation, that involves moving the robot. Therefore, we would move the robot in a straight line and analyze the  $(x,y)$  variations for various image pixels. Consequently, this can only be done after calibration, and that's the reason we consider the first calibration image the reference, and not the robot itself, which would eliminate the need of this last calibration step. Besides, the pixels on the peripheral regions of the image would not be used, as the calibration error is bigger in these areas. Finally, we are not sure the robot moves in a real straight line, hence we would only use the initial average derivative (average of derivative of vector  $(x,y)$  for every pixel used) to estimate the robot orientation.

Moreover, there is also the constraint of knowing the relative positions of calibration images, which can be solved using the method above, which is to locate the robot in known positions. Another method could be to estimate its position. It is also important to notice that even in robot regions of the image, in which the lines are not visible, it is possible to detect lines, due to the dashed lines detection property. Therefore, we can also detect the corners in that region, hence

being able to calculate the nearest corner - which is useful to estimate the robot position problem - and use it to extract lines information. This is specially important to make better estimates of the robot position in order to reduce its uncertainty.

However, to apply this method we would need to have already calibrated the system. To overcome this problem, a multi-step optimization could be used. Firstly, assuming we have the robot positions roughly estimated, we could still use the least squares approach to calculate the function parameters. After having that, we could use it as initial values for the parameters in a second non-linear step that estimates both the function parameters and the robot positions

This calibration procedure also creates the problem of acquiring images with the robot orientation restricted to multiples of 90 degrees, otherwise the calibration would have catastrophic results. To ensure that constraint is verified, we propose the following approach. The user must put the robot on a 3-line corner with any orientation, and then one should run a program to identify the lines departing from the robot. After that, the robot must be moved along the intercepting line, so as the the only remaining possible line direction departing from the robot still not identified is visible. After that moment, we have a way of determining if the robot is in one of the only four possible orientations. Then, the user can remotely control the robot, moving it to corners, in which it is possible to verify the orientation just by using comparing the predetermined line positions with the actual lines departing from the corner. This verification could be done automatically, and the user would only have to move it around. Using this approach, any kind of feedback would be useful for the user to know when he had placed the robot in a corner with the right orientation. Besides, in the initial position the user chose there would be no need to precisely put it in the corner, as long as in the end he can define the four line possible positions. For the calibration process, the position could then be estimated as described previously. The position deviation from the corner would then be the same in every corner. With this approach only corners can be used as locations for the robot, although it still give the possibility of using  $14 \times 4$  different calibration images. This procedure is exemplified in Figure 5.6.



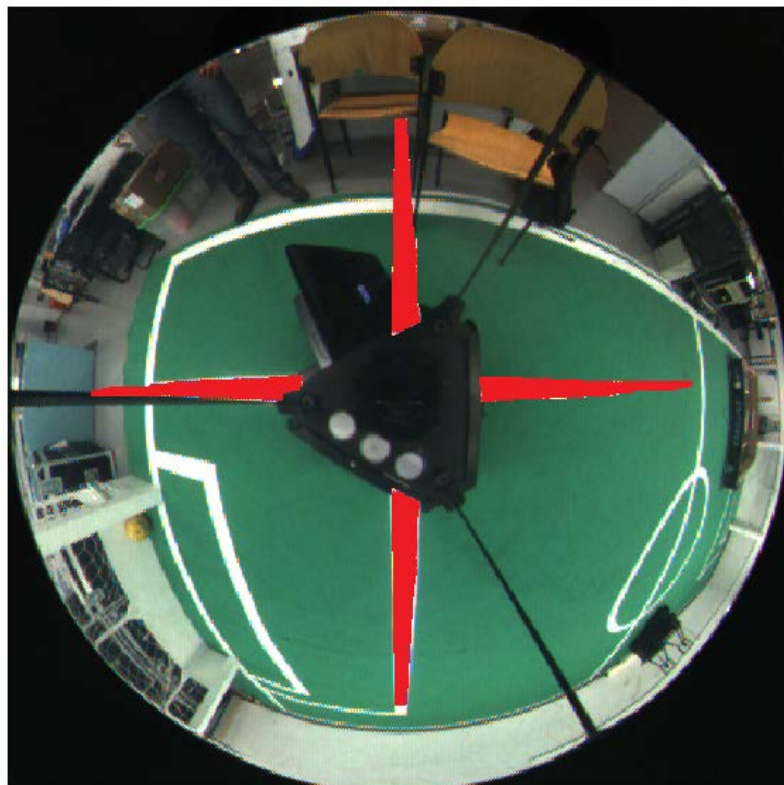


Figure 5.6: Example of virtual aid in the image, that could allow the implementation of the calibration process, considering the restrictions of this method



## Chapter 6

# Conclusions and Future Work

### 6.1 Analysis and Review

The objective of this thesis was to create a calibration procedure for the soccer robots. We simulated the vision system and calibration procedure, and used the simulation to validate the calibration method. However, we did not implement this method using the real soccer robots, thus not being able to validate the procedure with data extracted from real images.

In this work we started by creating a simulation of the catadioptric vision system, in order to test the calibration method proposed. We use the pinhole model for the camera, and simulate the mirror using an hyperbolic function with two sheets. We also simulated the misalignments of the system by defining a coordinate system for each part that constitutes the global system. Therefore, we have the image plane, and the camera, mirror and world coordinate system. The camera model creates the correspondence between image plane points, and points in the camera reference. Moreover, we created two homogeneous transformations (and the respective inverses) to define the relative position between referentials: camera to mirror and camera to world transformation.

Then, we calculate the world points corresponding to each pixel using backpropagation ray tracing. With this technique, we start at each image pixel and, using the camera model, we obtain the ray of light that comes from the world to that pixel. With that ray equation we find its intersection with the mirror, then calculating the normal vector of the surface at that point. Finally, using the normal vector and the incident ray, we calculate the direction of the reflected ray. Then, we just transform it in world coordinates, and calculate the intersection of the ray with the ground. Considering the ground point it intersects, it can be a white line or just the green soccer field. We also consider the possibilities of the ray not intersecting the mirror (for example, because the mirror is finite) and the cases in which the ray points upward in the world reference, thus not intersecting the ground.

After that, we studied which functions would fit better the mapping from pixels to ground coordinates. We used polynomial functions with both cartesian and polar coordinates, testing it with two simulations of vision sensors, one ideal and the other non-ideal. We concluded that for SVP systems the polar coordinates have a better result, however, in the non-ideal case with

misalignments, the first one had better results, due to usage of only linear optimization for the parameters estimation. On the contrary, the use of polar coordinates means the optimization has to be nonlinear.

We also proposed a method to detect the individual lines that constitute the soccer field white lines. For that we assume that rectilinear lines, although generally being curvilinear in catadioptric systems, are locally rectilinear. In order to detect lines, we search all pixels and use a tracking technique to follow and detect the entire line. We even assume lines can be dashed, using its local direction to continue searching the line even when we stop detecting it.

We apply some post-processing methods to accomplish the optimal line detection, such as merging, extension and elimination of lines, and also corner detection. For the latter, we use the change of local line direction as an indicator of corner probability.

Following that, we presented a technique to classify the lines detected, in order to be able to extract information from them. For that, we used the well known structure of lines connection in a soccer field, and compared it with the unidentified lines in order to classify each detected line. This approach has the benefit of not having to attribute a classification to every line, only the ones it is certain can be correctly classified. Therefore, we can have images in which only part of the lines are classified, thus reducing the calibration error due to misclassified lines.

Finally, we used a linear optimization combined with the information extracted from the lines to estimate the function parameters that calibrate the system. We also presented the results obtained using that procedure for various polynomial degrees, calibration images and test images. Moreover, we show it is possible to obtain a good calibration with this process, when using multiple images for the calibration (with varying robot positions) and an appropriate order for the estimation function. Although the proposed calibration procedure has some constraints on the images acquiring process, we suggested some solutions that make this method valid and applicable to real situations. Considering the order of the polynomial function used, it may have to be relatively high in order to have good calibration results. As a consequence, it can be difficult to calculate in real-time the function value for a given pixel. A possible solution is to use a lookup table to save those values. This reduces the time consumed to get the correspondence between pixel and world point, although it necessarily increases the amount of memory needed to store the lookup table. Nevertheless, considering the usual capacities of computers, and that the image has  $500 \times 500$  pixels, this is not expected to be a problem.

## 6.2 Future Work

In this section we indicate some of the features we did not implement, but consider important for the success and accuracy of the calibration method, thus being part of the future work plans.

The implementation of the proposed techniques in real robots using non simulated images is the first step, as it is essential to evaluate the calibration accuracy we can obtain with this method. This assessment is true considering some of the differences we have between simulation and reality. Firstly, the simulation does not consider surrounding objects in the field, possibly

covering some field lines. Moreover, the goal was not considered, and being white it can be wrongly classified as field lines. Finally, with some simple tests we concluded lines far from the robot are even more difficult to detect than in the simulation, possibly due to blur.

Regarding the extraction of field lines from the image, a technique is presented in [6], based on color segmentation. This method was already implemented and used by the soccer robots before this work. However, it can only detect lines near the robot. For example, with the robot in the center of the field, this method will only detect the center circle, the halfway line, and part of the side lines, as seen in Figure 6.1a. Nevertheless, this is not enough for the calibration procedure, because it implies only the central region will be calibrated, which shows no improvement in the calibration quality over the calibration procedure the robots are already using. Therefore, we desire to develop a method that can better cope with the detection of lines further from the robot.

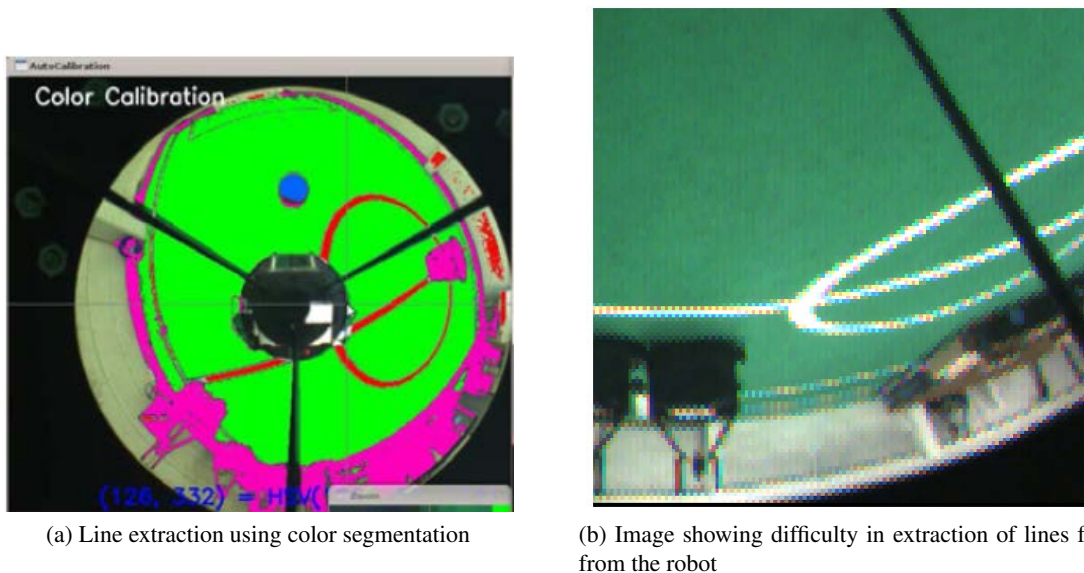


Figure 6.1: Example of lines extraction using color segmentation in (a), as presented in [6]. In (b) it is possible to observe the difficult line extraction problem in this images, because further lines almost do not have white pixels.

In the simulation we can add extra feature, as an estimate of the blur, lens distortion, and more types of mirror models. In order to properly simulate the difficulty of line detection in the periphery of the image, we used low definition simulations and implemented detection of incomplete lines. However, with real images other problems arise. For example, we see lines are even more than one pixel wide, but its color is quite dissimulated, probably due to blur. This happens because cameras cannot focus all distances at the same time, and while focusing the central region, the peripheral areas will defocus, and therefore become blurred.

Currently, the line detection and identification operations are executed separately, but in the future it may be worth trying to use both of them together. Although we can only classify lines that were already detected, we could, instead of detecting all of them and only in the end identifying them, identify some lines, and with those classified try to detect more lines correctly. This

approach would only be useful to try to detect lines with invisible parts, which may be caused by objects covering the lines (ball, other robots, ...), it can be due to low resolution of regions further from the robot, or even because of blur that interferes with lines segmentation.

Anyway, even without using this approach, the methods must be improved in order to have better line identification, and decreasing the number of lines misclassified. Moreover, we must also increase the robustness of this methods regarding the wrongly detected lines.

Regarding the optimization process, we want to experiment with other functions, such as splines. This kind of mappings may facilitate the good calibration of both central and peripheral image regions.

We also want to study other kinds of non-parametric calibration methods. For example, instead of trying to create a pixel to ground point mapping, it would be interesting to try to estimate a function that relates the pixel position with the rays in the world coordinate system. This would be similar to the backpropagation ray tracing methodology, but instead of using the camera, mirror and misalignments models to calculate it, we would just estimate a function to fit that transformation.

Finally, we would like to export the code developed to *Lazarus*, because this programming language is used for almost all applications of the soccer robot team at FEUP.

# References

- [1] A. Colombo, M. Matteucci, and D. Sorrenti. On the calibration of non single viewpoint catadioptric sensors. *RoboCup 2006: Robot Soccer World Cup X*, pages 194–205, 2007.
- [2] J.P. Barreto and H. Araujo. Geometric properties of central catadioptric line images and their application in calibration. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(8):1327–1333, 2005.
- [3] D. Scaramuzza, A. Martinelli, and R. Siegwart. A flexible technique for accurate omnidirectional camera calibration and structure from motion. In *Computer Vision Systems, 2006 ICVS'06. IEEE International Conference on*, pages 45–45. IEEE, 2006.
- [4] B. Cunha, J. Azevedo, and N. Lau. Calibration of non-svp hyperbolic catadioptric robotic vision systems.
- [5] T. Mashita, Y. Iwai, and M. Yachida. Calibration method for misaligned catadioptric camera. *IEICE TRANSACTIONS ON INFORMATION AND SYSTEMS E SERIES D*, 89(7):1984, 2006.
- [6] A.J.R. Neves, A.J. Pinho, D.A. Martins, and B. Cunha. An efficient omnidirectional vision system for soccer robots: From calibration to object detection. *Mechatronics*, 2010.
- [7] D. Scaramuzza. *Omnidirectional vision: from calibration to robot motion estimation*. PhD thesis, ETH, 2007.
- [8] B. Cunha, J. Azevedo, N. Lau, and L. Almeida. Obtaining the inverse distance map from a non-svp hyperbolic catadioptric robotic vision system. *RoboCup 2007: Robot Soccer World Cup XI*, pages 417–424, 2008.
- [9] P. Lima, A. Bonarini, C. Machado, F. Marchese, C. Marques, F. Ribeiro, and D. Sorrenti. Omni-directional catadioptric vision for soccer robots. *Robotics and Autonomous Systems*, 36(2):87–102, 2001.
- [10] S.B. Kang. Catadioptric self-calibration. In *Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on*, volume 1, pages 201–207. IEEE, 2000.
- [11] C. Mei and P. Rives. Single view point omnidirectional camera calibration from planar grids. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 3945–3950. IEEE, 2007.
- [12] C. Geyer and K. Daniilidis. A unifying theory for central panoramic systems and practical implications. *Computer Vision ECCV 2000*, pages 445–461, 2000.

- [13] J.P. Barreto and H. Araujo. Issues on the geometry of central catadioptric image formation. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 2, pages II–422. IEEE, 2001.
- [14] C. Cauchois, E. Brassart, C. Drocourt, and P. Vasseur. Calibration of the omnidirectional vision sensor: Syclop. In *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, volume 2, pages 1287–1292. IEEE, 1999.
- [15] D. Strelow, J. Mishler, D. Koes, and S. Singh. Precise omnidirectional camera calibration. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–689. IEEE, 2001.
- [16] C. Geyer and K. Daniilidis. Catadioptric camera calibration. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 1, pages 398–404. IEEE, 1999.
- [17] B. Zhang and Y. Li. Homography-based method for calibrating an omnidirectional vision system. *JOSA A*, 25(6):1389–1394, 2008.
- [18] S. Ramalingam, P. Sturm, and S.K. Lodha. Towards complete generic camera calibration. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 1093–1098. IEEE, 2005.
- [19] P. Sturm and S. Ramalingam. A generic concept for camera calibration. *Computer Vision-ECCV 2004*, pages 1–13, 2004.
- [20] Wikipedia. Pinhole camera. [http://en.wikipedia.org/wiki/Pinhole\\_camera](http://en.wikipedia.org/wiki/Pinhole_camera), June 2012.
- [21] Y. Morvan. Pinhole camera model. <http://www.epixea.com/research/multi-view-coding-thesis.html#multi-view-coding-thesisch2.html>, June 2012.
- [22] M.A.P. Cisneros. *Intelligent Model Structures in Visual Servoing*. PhD thesis, the University of Manchester, 2004.
- [23] Wikipedia. Lens distortion. [http://en.wikipedia.org/wiki/Distortion\\_\(optics\)](http://en.wikipedia.org/wiki/Distortion_(optics)), June 2012.
- [24] Wikipedia. Linear least squares. [http://en.wikipedia.org/wiki/Linear\\_least\\_squares\\_\(mathematics\)](http://en.wikipedia.org/wiki/Linear_least_squares_(mathematics)), June 2012.
- [25] B. Luo and E.R. Hancock. Structural graph matching using the em algorithm and singular value decomposition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 23(10):1120–1136, 2001.
- [26] S. Umeyama. An eigendecomposition approach to weighted graph matching problems. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 10(5):695–703, 1988.