AUTOMATIC EXTRACTION OF THE CAMERA
POINT OF VIEW IN FOOTBALL SCENES

Master’s Final Project Dissertation

by JORDI PONT TUSET

Advisor: Prof. Ferran Marques Acosta
Barcelona, July 2009
Abstract

Automatic soccer scene analysis is the order of the day and the extraction of the point of view may play a central role in the process. In other words, knowing where the camera is located and where it points may give us priceless information towards understanding the scene content at a high level in soccer matches.

We propose an algorithm that is based on the matching of a field line model to the detected lines of the image to calibrate a simple camera model. Instead of pursuing the best accuracy possible, which is in general the aim of camera calibration algorithms, we present a new point of view that simplifies all the steps of the process to gain in efficiency, feature asked to the majority of scene analysis algorithms. The finally reached accuracy is shown to be sufficient for many of the applications demanded. In the remaining cases, the obtained results can be used as initial point for a more precise (and time-consuming) refinement step.

Specifically, the algorithm first extracts the grass pixels in a simple yet robust manner. Lines are then detected using a single Hough transform, which is further exploited to calculate the vanishing points and detect the pieces of ellipses of the field. Once the lines detected are recognized, the point of view is extracted in a two-step algorithm that allows us to define different levels of quality processing. Combining these levels, we are capable of extending the algorithm to sequences, minimizing the computational cost towards a real-time performance.

In parallel, the author has advised a Degree’s Final Project focused on the shot classification of soccer scenes based on low-level features of the grass layout, which fits seamlessly with the current work.
List of Figures

1.1 Algorithm block diagram and dissertation organization .......................... 4
2.1 Original images where different situations for grass detection are illustrated 6
2.2 Grass mask draft result for three different situations .............................. 7
2.3 Grass detection result for three different situations ............................... 9
2.4 Line draft extraction process results .................................................. 10
3.1 Line parameterizations for the Hough transform .................................. 12
3.2 Theoretical example of Hough transform of a line ................................ 14
3.3 Vanishing points and vanishing line in the Hough domain ....................... 16
3.4 Theoretical example of Hough transform of a circle ............................ 17
3.5 Theoretical example of Hough transform of an ellipse .......................... 18
3.6 Comparison of the Hough transform of a circle and an ellipse ............... 19
3.7 Illustration of the process of line sweeping for circles and ellipses ........... 19
3.8 Hough transform real example ...................................................... 20
3.9 Filtering of the Hough transform .................................................... 21
3.10 Thresholding of the Hough transform ............................................. 22
3.11 Hough instances classification for goal shots .................................... 25
3.12 Hough instances classification for middle-field shots ......................... 25
3.13 Computation of vanishing points .................................................... 27
4.1 Intersection point recognition and incidence structures ......................... 30
5.1 3D global frame of reference ......................................................... 32
5.2 Rotation angles that define the orientation of the camera ..................... 33
5.3 Pinhole camera model and its frame of reference ............................... 34
5.4 Calibration results ........................................................................ 37
6.1 First level processing scheme ........................................................... 39
6.2 Second level processing scheme ....................................................... 40
6.3 Third level processing scheme

6.4 Illustration of the line tracking procedure

6.5 Fourth level processing scheme

7.1 Grass detection wrong results

7.2 Line detection wrong results

7.3 Level 1 results

7.4 Level 2 results

7.5 Level 3 results

7.6 Level 4 results

7.7 Player markers demonstration

7.8 Outside line plotting demonstration

A.1 Morphological refinement of the grass mask
Notation

\(T_R, T_B\) Thresholds applied in the grass extraction algorithm for the red and blue channels, respectively. 6

\([u, v]\) Image pixel coordinates used in this work, where the origin is placed at the up-left corner of the image, \(u\) is the horizontal direction, and \(v\) the vertical one. 8, 36, 59

\(r_U\) Horizontal resolution of an image, i.e., its size in the \(u\) axis. 36, 38, 64, 66

\(r_V\) Vertical resolution of an image, i.e., its size in the \(v\) axis. 36, 38, 64, 66

PTZ Acronym that stands for Pan (\(a\)), Tilt (\(\beta\)), and Zoom (Focal length \(F\)), which are three of the pinhole camera model parameters. 42

RGB Acronym that stands for Red, Green, and Blue, one of the most well-known color spaces. 6

xyz Acronym referring to the 3D-world coordinates of the camera \((C_x, C_y, C_z)\), three of the pinhole camera model parameters. 42
Chapter 1

Introduction

Soccer is among the sports with largest audience, so any added value that enhances the standard broadcast is priceless for production and advertising companies. Augmented reality, automatic summarization, or game statistics may be listed among the features one could think of; but in turn, they usually entail laborious manual tasks or using expensive dedicated hardware. Special effort has therefore been made to automatically provide as much information as possible. Automatic camera viewpoint extraction plays a central role in this process, laying the basis for promising and ground-breaking algorithms.

In the context of the i3media project\textsuperscript{1}, there is the need of computing the type of shot for each frame towards numerous applications: whether the advertisement of the field is visible at every moment, event detection to produce relevant summaries, content analysis to provide match statistics, among many others. Computing the point of view of the scene, i.e., the point from where the image is taken and to which spot the camera points, may provide a common framework for all these objectives to be achieved.

Point of view extraction can also be understood as camera calibration, i.e., to provide a mathematical description, or camera model, of how the scene is projected to the image. In this case, the aim is usually to provide the highest level of accuracy in order to be able to precisely locate the image objects in the real world. Accordingly, camera calibration algorithms are usually computationally expensive.

The pursued applications in the i3media project, however, are intended to have the least delay possible or even work in real time. On the plus side, many of them do not need very accurate camera calibrations and can be achieved with an approximation of the position

\textsuperscript{1}CENIT-2007-1012: Tecnologías para la creación y gestión automática de contenidos audiovisuales inteligentes, I3MEDIA.
of the camera or where it points at.

Thus, the main objective of this Master Thesis is to prove that an efficient and simple algorithm can be designed yet obtaining enough accuracy for the objectives presented. It is not the aim of the Master Thesis to provide an algorithm that meets the demand of a real-world scenario, since we believe that it is out of the reach of a Master Thesis: all the particular cases that one can found in a real-world broadcast requires a hard work of tuning and adaptation that is beyond the research objectives. Despite this, we believe the final result is not that far from it and can have promising applications in short.

In other words, this Master Thesis is a proof of concept: a demonstration that another approach can be taken to camera calibration in which a better performance is achieved, while providing accurate enough results for the applications demanded.

Specifically, our algorithm relies on the detection and recognition of the visible field lines, whose layout in the image, along with the a priori knowledge of the field line model, may be used to achieve our objective. To do so, we have designed the whole process from scratch: from the original image taken from a TV broadcast to the final result, involving various types of techniques from Image Processing and Computer Vision. All the links of the chain are fully exploited to take full advantage of the operations performed, in order to improve the algorithm's efficiency.

The input of our algorithm is a video sequence taken directly from the television signal. As so, we have to deal with any type of frames that may be found on a soccer broadcast: advertisement sections, close-up views of the stand, trainers, or players. Looking for field lines in these type of scene not only may be misleading but is a waste of computational time. Apart from that, there might be close-up views of the field in which there are lines but there is not enough perspective to recognize them.

In order to tackle this problem, this work has been done in parallel with the Degree's Final Project of David Varas, which Jordi Pont has advised during six months. In it, the author performs a shot detection and classification using only low-level features of the image to discard the shots in which lines should not be looked for. Specifically, its input is the grass layout, whose extraction is in turn the first step towards line detection. Image features are reused between the two algorithms providing an efficiency improvement.

Thus, the current Project assumes that it receives a whole shot in which field lines are visible from a wide view.
1.1 State of the Art

Many approaches have been presented that rely on the detection of the field lines or points to extract the point of view of sports scenes.

In [FKdWE04], a camera calibration algorithm for sport sequences is presented, that relies on the matching of the detected lines and points with the field model. Initially, a draft of the field lines is obtained from a frame, via a white pixel detector, and the Hough transform is used to get a large set of candidate lines that are refined in a combinatorial search within the model lines. This search can be computationally expensive. The approach takes advantage of the temporal redundancy of video sequences to update the model frame to frame and avoid recomputing the calibration.

A similar global approach, focused on soccer, may be found in [HPV04]. In this case, lines are extracted using recursive polynomial approximations of contour chains after a Canny's edge detector. They are classified into two groups provided that, in a soccer field, two main line directions are visible. The vanishing points are then computed using a RANSAC-type algorithm. Again, this approach is computationally expensive and needs a priori approximations of some parameters.

A different approach to soccer video analysis is presented in [ET03], where the authors make use, among other features, of the percentage of grass pixels in the image to detect shot boundaries, classify the type of view, etc. Bayesian classifiers are employed, which have to be previously trained. Grass pixel detection is performed looking for a dominant color in the histogram, so hard color transitions (e.g. due to shadows) can cause the algorithm to fail.

Our approach may be understood as a meeting point between the abovementioned methods, simplifying and fully exploiting each step of the algorithm towards an efficient and robust algorithm.

1.2 Algorithm Overview and Organization of the Dissertation

This section gives an overview of the whole process and links each step with the chapter of this Dissertation in which it is explained. Figure 1.1 depicts the block diagram of the algorithm presented, from the original image frame to the point of view parameters. Note that each block is linked to the chapter of this dissertation in which it is detailed.

First, Chapter 2 expounds on the image preprocessing. In this block, an original image is read and the grass area is detected. Then, white areas are searched among the grass and
so the output is the line draft mask.

Then, Chapter 3 is focused on the **Hough domain exploitation**. This block is the core of the algorithm, since, apart from detecting the lines in the image, it provides a valuable stream of information about the visible ellipses and vanishing points of the field lines.

Chapter 4 describes how the detected lines and points are **recognized**, i.e., matched with a field model, in order to know their 3D-world coordinates.

Chapter 5 shows how, gathering the instances recognition and the vanishing points coordinates, the **point of view** is computed.

Next, Chapter 6 explains how the algorithm takes advantage of the **temporal redundancy** in video sequences, reusing information between frames towards speeding the process up.

Chapter 7 is devoted to present and discuss the results of the **experiments** performed to assess all the blocks involved in the algorithm.

Finally, **conclusions** are drawn and the **future lines of work** are described in Chapter 8.

The whole process is illustrated with three typical scenes, where the results of each step of the algorithm may be analyzed and the flow of the algorithm may be followed. In order to facilitate the reader's understanding of the text, a Notation section has been provided, where some of the terms used in the work are briefly described and linked to the page where they appear.
Chapter 2

Image Preprocessing

The first step of our algorithm is the image preprocessing, which is aimed at obtaining a draft of the field lines visible in the image. In other words, we want to obtain a binary mask where each pixel is classified as being part of a line or not. As we will describe in next chapters, this mask will be used to detect and recognize the field lines.

This process is divided into two main blocks. First, the algorithm detects the grass pixels in the image via a color filtering and a refinement step. As introduced before, this part of the algorithm has been the base of the degree’s final project of David Varas. It has been improved in his work and a shot detector has been designed based only on low-level features of the grass mask.

Second, the process tries to extract the line pixels among the grass ones. Following we describe both processes, motivating the decisions made, illustrating the process with examples, and proving the correctness of the approach with some challenging situations.

2.1 Grass Detection

The detection of the field grass is the first step in our algorithm. The key idea behind it is to focus the search of lines to a reduced area, in order to minimize both the false detections due to the view of the stand and the computational effort of the algorithm. In other words, we mainly want to remove the spectators and the players in order for the line detection algorithm to minimize the errors.

The detection of the field grass as a base for more complex algorithms in sports is widely used in previous works. This detection is usually performed via the detection of a single
dominant color, as done in [ET03]. They compute the color histogram in the HSI space and look for the peak in each channel. A pixel is considered as grass if its color value lies in an interval around these peaks. The statistics of the grass is learned at start up and automatically updated through the match. This methodology is able to handle slow variations of the grass color when, for instance, the sun is setting. A challenging situation for this algorithm, however, is the presence of hard color transitions within the field, usually due to shades caused by the sun. Since the process looks for a single dominant color, only a part of the grass is correctly detected.

Figure 2.1 shows three original images where different situations for grass detection are illustrated, particularly the challenging situation where a hard color transition is found on the grass.

![Figure 2.1: Original images: (a) typical image of the goal, (b) typical image of the field center, and (c) grass with dark shadows where the single dominant color detection would fail](image)

The approach followed in this work is much simpler yet robust and efficient. In short, the main idea is to consider grass anything with a dominant value in the green channel. In order to make it robust, a refinement of the grass mask is performed taking advantage of the fact that the grass area is usually formed by big connected components.

Following, we give a formal description of both the grass mask draft computation and its further refinement.

### 2.1.1 Grass mask draft

Given an image represented in the RGB space, we select those pixels whose green channel value is dominant, i.e., a pixel $p$ with color values $(R_p, G_p, B_p)$ is said to be grass if:

$$G_p > (1 + T_R)R_p \quad \text{and} \quad G_p > (1 + T_B)B_p$$

where $T_R$ and $T_B$ are color tolerance factors.
2.1 Grass Detection

This computation is performed for all the pixels of the image, resulting in a binary mask, whose *activated* values refer to the grass. Recalling the objective of the grass mask computation, to search the field lines, we need for the white pixel lines to remain in the mask. Having a close look to the images we are working with, we noticed that, although the lines seem white, they are a very saturated green tone, maybe due to the coding process, or because of the transparency of the chalk used to mark the lines.

Figure 2.2 show the resulting mask for the three cases presented above, where we depict the non-grass pixels in pale yellow. As may be observed in Image (c), the algorithm handles the hard transitions properly. As drawback, the mask includes some non-grass green areas present among the supporters, which result in numerous false detections that motivate the further refinement, as presented in the next section.

![Figure 2.2: Grass mask draft, where non-grass area is highlighted in yellow: (a) typical image of the goal, (b) typical image of the field center, and (c) challenging situation due to the grass shades](image)

2.1.2 Grass mask refinement

Having a closer look to the results in Figure 2.2, we may notice three types of error we would like to solve:

- Some false positive detections of non-grass areas in the stand due to some green spots among the supporters.
- Some false negative detections of grass areas that are not classified as grass due to its non-green color (e.g. a brown spot).
- The players are not completely removed. This effect may be caused by the coding algorithm of the images, that merges the green color of the grass with the boundary of the player, resulting in a gradation between green and the player colors.

In other words, with this refinement we are trying to focus the line search only on the grass area while minimizing the potential false detections caused by the presence of players or brown spots on the grass.
2.1 Grass Detection

Our first approach to this process of refinement was based only on morphological filtering. A combination of opening, closing, and erosion processes with different structuring elements solved the three items listed above correctly in the majority of the views. When trying the algorithm extensively, however, its limitations were made evident and so we changed the paradigm. Despite this, we consider this first approach interesting and clean, so we have included a description of the process in Appendix A.

The final approach is mainly based on area filtering. First of all, in order to remove the spots among the supporters, we proposed to remove all the connected components of the mask but the one with largest area. This is based on the fact that the grass is usually a single large connected component in the mask. This is not always true, however. In some cases, the players, the scoreboard, or wide field lines, disconnect the grass area into two or more pieces. In order to overcome this problem, the first step of the refinement consists in gathering a given percentage of the mask area, consecutively merging the biggest remaining chunk available. Let us give a formal description of this algorithm.

Let \( m[u, v] \) be the grass mask draft, where \( m[u, v] = 1 \) if the pixel with coordinates \([u, v]\) is grass and \( m[u, v] = 0 \) otherwise. We first label the connected components of value 1 and compute their area. Let us assume there are \( N \) connected components in the image. Let \( R_i \) be the connected component with \( i \)th biggest area \( A_i \), i.e., \( R_1 \) is the connected component with largest area and \( R_N \) the smallest. Then, we remove all the connected components \( R_i \) with \( i > n \), where:

\[
n = \min \left\{ 1 \leq i \leq N \text{ s.t. } \sum_{k=1}^{i} \frac{A_k}{A} > 0.8 \right\}
\]

being \( A = \sum_{k=1}^{N} A_k \) the total area of the draft mask. In short, we gather the first biggest regions that add at least the 80% of the draft mask area. Note that in the majority of the cases, a single connected component represent far more than 80%, and when not, two components already do.

At this stage we have correctly removed the spectators noise, but we still may have small holes in the grass mask we want to remove. They are usually caused by the lack of grass in some small parts of the field. The player holes, however, must remain. The procedure to achieve this goal is to remove those holes whose area are smaller than a threshold. In order for the algorithm to work independently of the image size, this threshold is defined as a percentage of the global image area. In our case we set this percentage to 0.03%. Formally, if the global image area is \( A_I \), we remove the holes whose area \( A_I < 0.0003A_I \).

The last step, in order to remove completely the players, is to dilate the resulting mask using the following structuring element \( se_1 \):
2.2 Line Draft Extraction

2.2 Line Draft Extraction

Figure 2.3 depicts the resulting grass mask after the refinement process for the three selected cases. Notice how the refinement algorithm do not erase some grass holes due to the lines in the image (c), because they are bigger than the selected threshold. Comparing this result with the image (b), however, we realize that if we set a bigger threshold, then some player holes would be remove. Since it is preferable to remove a piece of line to having a player in the resulting mask, we used a small threshold.

\[ se_1 = \begin{bmatrix}
0 & 0 & 1 & 0 & 0 \\
0 & 1 & 1 & 1 & 0 \\
1 & 1 & 1 & 1 & 0 \\
0 & 1 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 & 0 
\end{bmatrix} \]

\( se_1 \)

(a)

(b)

(c)

Figure 2.3: Grass detection result, where non-grass area is highlighted in yellow: (a) typical image of the goal, (b) typical image of the field center, and (c) challenging situation due to the grass shades

2.2 Line Draft Extraction

Once we have the grass mask, the next step is aimed at extracting a draft of the lines in the images, i.e., obtaining a binary mask with ‘1’ at the pixels where we consider there may be a line and ‘0’ at the other pixels.

First of all, we convert the masked image, i.e. the one without stand nor players, to grayscale, as depicted in Figure 2.4(a). Let \( BW_1[u, v] \) be the resulting image.

Next, we want to highlight the lines among the grass. Theoretically, since they are white on dark colors, lines should be the areas with highest contrast in the grass zones (recall that the players have been removed). In order to highlight these areas, therefore, we apply a tophat filter using a small structuring element:
2.2 Line Draft Extraction

The shape of this structuring element was chosen to have a high response on thin lines, while minimizing the output due to noise. The filtered image is, therefore:

\[
BW_2[u, v] = \text{tophat}(BW_1[u, v], se_4)
\]

The resulting image is shown in Figure 2.4(b), where we may check that the homogeneous zones of the grass are almost removed, while the lines produce a high response.

The last step is to **binarize** the grayscale filtered image by a threshold \(th\). Formally:

\[
B[u, v] = \begin{cases} 
1 & \text{if } BW_2[u, v] > th \\
0 & \text{otherwise.}
\end{cases}
\]

In order for the threshold to be independent of the image being processed, the value of \(th\) is chosen as the percentile 98 of the grayscale filtered image \(BW_2[u, v]\). Formally, if 
\(H(t)\) is the normalized histogram of the values of \(BW_2[u, v]\), and \(m = \min\{BW_2[u, v]\}, M = \max\{BW_2[u, v]\}\):

\[
th = \min \left\{ s \in [m, M] \text{ s.t. } \sum_{t=m}^{s} H(t) dt \geq 0.98 \right\}
\]

Figure 2.4(c) depicts the final thresholded image \(B[u, v]\), where the lines are clearly highlighted, and the noise due to players, grass, or spectators is acceptable.

\[
se_4 = \begin{bmatrix}
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0
\end{bmatrix}
\]

\(BW_2[u, v] = \text{tophat}(BW_1[u, v], se_4)\)

Figure 2.4: Line draft extraction process: (a) grayscale masked image, (b) image after tophat by \(se_2\), (c) binarized image. Please note that images (b,c) have been inverted for the sake of intelligibility.
Chapter 3

Hough Domain Exploitation

As introduced previously, the key idea of our approach is to compute the camera viewpoint based on the shape and configuration of the field lines. A soccer field is formed by straight lines and circles or arcs (ellipses or segments of ellipses, due to the projection), so we are interested in detecting both types of shapes. Once we have a draft of the field lines and ellipses, the next steps are aimed at detecting and characterizing them in the mask, and, if possible, computing the vanishing points of the field lines.

The Hough transform is a widely used tool to detect lines and curves in images [DH72]. Ideally, peaks in the transformed domain represent instances of the parameterized curve in the original image. Since we are interested in two types of shapes, a first attempt could be to compute two Hough transforms: one for straight lines, and another one for ellipses. This option, however, should be avoided due to its computational load.

Our approach is based on performing the Hough transform for straight lines in normal form and detecting both lines and ellipses. In the transformed domain, the former will be detected as peaks and the latter as thin long crests.

The following four sections are devoted to providing the theoretical explanations about the Hough transform in order to understand the algorithm presented. Specifically, the Hough transform basis for line detection are laid in the first section. Then, two sections are devoted to showing useful properties of the Hough domain. Finally, we expound on the theoretical properties that allow the detection of ellipses in the Hough domain for straight lines.

In the second part of this section, in the last three sections, we describe how our algorithm takes advantage of the theoretical basis provided previously. First, we give details about
the filtering of the result in real cases and we show how we exploit its properties to characterize the ellipses and lines detected. Finally, when possible, we present how we estimate the vanishing points of the field lines based on the Hough domain’s properties.

3.1 Hough Transform Basis

The Hough transform is one of the most well-known techniques for line and curve detection in images. The first step consists in parameterizing the shape or curve we want to search in the image. When detecting lines, for instance, one could think of parameterizing them in slope-intercept form, i.e., if taking the usual $u - v$ representation of the image plane, a line could be represented as:

$$v = mu + n$$

where $m$ is known as slope, and $n$ as intercept, due to the fact that it is the $v$ value in the interception with this axis. Figure 3.1(a) depicts the meaning of these values, where $m = \tan \beta$.

![Figure 3.1: Line parameterizations for the Hough transform: (a) slope-intercept form and (b) normal form](image)

This parameterization, however, is not the most suitable for line detection, mainly because a vertical line cannot be represented, or almost-vertical lines have very large slope values. As motivated in [DH72], a better line parameterization for line detection is the normal form. Note that, although not used in the process of line detection itself, the properties of the Hough transform in slope-intercept form has very interesting properties. That is why we will devote the following sections to show them and motivate the use this form will have in our algorithm.

A line is parameterized in normal form as:

$$ucos \theta + v \sin \theta = \rho$$
3.1 Hough Transform Basis

where $\rho$ is the distance from the line to the origin, and $\theta$ is the slope of the line passing through the origin and perpendicular to the parameterized line. Figure 3.1(b) shows the meaning of these values.

Note that there is an ambiguity in this parameterization, since if the point $(\theta, \rho)$ belongs to the line, so does $(\theta + \pi, -\rho)$. This forces us to bound the parameter values for the transform to be unambiguously defined. In our case, we restrict the line parameters to be in:

$$[\theta, \rho] \in [0, \pi) \times [-D, D]$$

where $D$ is the diameter of the image, i.e., the longest distance from the origin a line can be. This way, each line in the image corresponds to a unique transformed point. The space consisting of all the possible values of the parameterized shapes is known as Hough domain.

In practical uses of the Hough transform, the Hough domain is discretized in a uniform grid of steps $\Delta \theta$ and $\Delta \rho$, dividing it in rectangular bins. Having a binary mask, each of its points $\{u_i, v_i\}$ is transformed into its equivalent sinusoidal curve in the $\theta - \rho$ plane:

$$u_i \cos \theta + v_i \sin \theta = \rho$$

The procedure of the Hough transform is to compute the sinusoidal for each point in the original mask curve and vote each bin of the Hough domain it passes through. It is straightforward to prove that the sinusoidal curves of two points in the original image intersect in the point corresponding to the line passing through the two original points. A set of aligned points in the original mask create, therefore, an accumulation of votes in the corresponding line. This way, ideally, a large set of aligned points in the original mask correspond to a large peak in the Hough transform, whose coordinates refer to the parameters of the line in the original image.

To sum up, the key idea is to transform a binary image, look for the peaks in the Hough domain, and their coordinates correspond to the parameters of the lines found in the original image.

Figure 3.2 depicts an example of Hough transform. On the left-hand side, we show an original image with 5 collinear points, and on the right, their respective sinusoidal curves in the same color. Observe how the sinusoidal curves of collinear points intersect in a single point in the Hough domain. Its bin will receive a large number of votes, creating a peak in the Hough transform. As can be seen, the coordinates of the intersection points correspond to the parameters of the line passing through the five points.
3.2 Slope-Intercept Hough Transform Duality

As introduced in the previous section, the form we will use to compute the transform itself and detect the line is the normal form. Once computed the instances, however, we translate their coordinates into slope-intercept form to take advantage of the good properties of this form. This motivates the study of the slope-intercept form properties that follows, in which our algorithm is based.

**Theorem 3.1.** The set of parameters of all the lines passing through a point in the image plane forms a line in the slope-intercept plane \((m, n)\).

**Proof.** Let \((u_i, v_i)\) be a point in the image plane. Let us compute the transformed curve in the Hough domain in slope-intercept form. First of all, the set of lines passing through this point can be expressed as:

\[
     v - v_0 = \lambda (u - u_0) \quad \Leftrightarrow \quad v = \lambda u + (v_0 - \lambda u_0)
\]

For each \(\lambda\), i.e., for each line passing through the point, we have a transformed point in the Hough domain, whose coordinates are the slope and intercept of the line. In our case, these coordinates are:

\[
\begin{align*}
    m &= \lambda \\
    n &= v_i - \lambda u_i
\end{align*}
\]

\[
\implies n = -mu_i + v_i
\]
That is, the transformed curve of a point in the image plane is a line in the Hough domain \((m, n)\) with slope \(-u_i\) and intercept \(v_i\).

**Theorem 3.2.** **Collinear points in the image plane correspond to intersecting lines in the slope-intercept plane.** The coordinates of the intersection point correspond to the line passing through all the original points in the image plane.

*Proof.* Let \(\{(u_i, m_0u_i + n_0)\}_i\) be a set of collinear points in the image plane, where \(v = m_0u + n_0\) is the line passing through the points. Recalling the former expression of the transformed line of each point, we have that the set of lines in the Hough domain is:

\[ n = -mu_i + m_0u_i + n_0 \iff n - n_0 = -u_i(m - m_0) \]

That is, a set of lines passing through the transformed point \((n_0, m_0)\).

In other words, the line passing through a set of points in the image plane corresponds to the intersection point in the Hough domain of the transformed lines.

**Theorem 3.3** (Slope-Intercept Hough Duality). **The intersection point of two lines in the Hough domain corresponds to the line passing through the anti-transformed points.** The line passing through a set of collinear points in the slope-intercept plane corresponds to a set of intersecting lines in the image plane, whose intersection point corresponds to the line passing through the points in the Hough domain.

*Proof.* Given that the correspondence between a point in the image and a line in the transformed plane is bijective, the inverse of the theorems presented before proves both assertions.

To sum up, a point in one domain corresponds to a line in the other domain. Intersecting lines in one domain correspond to joining points in the other one.

### 3.3 Perspective and Duality: Vanishing Points in the Hough Domain

This section expounds on the exploitation of the duality properties presented before for the slope-intercept form and some perspective principles that will provide useful information in soccer scenes, as we will present below.

When projecting a set of parallel lines into an image, the projected lines meet in a point known as **vanishing point**. Thanks to duality, the line that passes through the transformed
points in the Hough domain corresponds to the vanishing point of the lines. This way, the computation of a vanishing point of a set of lines can be performed in the Hough domain computing the line between the transformed points.

Each set of parallel lines projected in an image creates a different vanishing point, having the property that they are aligned forming the **vanishing line** of the image.

Let us illustrate all these properties with a soccer example. Figure 3.3(a) shows a sketch of the lines of the goal area. The lines of the soccer field, and many other sports, may be separated into two sets of parallel lines, which we represented in two different colors: in blue the ones we define as vertical \( (v_i) \) (they are usually seen closer to the vertical) and in green the horizontal ones \( (h_i) \). Each line of both sets has been lengthened until meeting the corresponding vanishing point \( (\infty_h) \) and \( (\infty_v) \). The broken line between both vanishing points is the vanishing line \( (\infty) \).

Figure 3.3(b) shows the dual representation of the lines in (a), i.e., the Hough transform in slope-intercept form. We may first observe how each set of parallel lines is transformed into collinear points. The transform of each vanishing point corresponds to the line passing through the points whose coordinates refer to the set of lines in the image meeting that specific vanishing point. The coordinates of the intersection point between lines coming from the transformed vanishing points are the parameters of the vanishing line.

**Figure 3.3:** Vanishing points and vanishing line in the Hough domain in slope-intercept form. (a) represents the image plane with two sets of parallel lines, their vanishing point and the vanishing line of the image. (b) depicts the dual representation of all these instances in the Hough domain in slope-intercept form.

Section 3.7 shows how we take advantage of all these properties to compute the vanishing points in a real scenario in a robust manner.
3.4 Detecting Ellipses in the Hough Transform for Straight Lines

This section is devoted to showing how ellipses can be detected in the Hough transform for straight lines. We will base our explanation on the development of theoretical examples, which will guide us towards the final algorithm. Let us start with the simplest case of ellipse: a circle. Figure 3.4(a) shows a set of points on a circle and in (b) the sinusoidal curves for each of them in the Hough domain.

Comparing this result with the one of a straight line in Figure 3.2, the first thing we may note is that there is no common point for all the sinusoidal curves, since in the original image there exists no line passing through all the points of the circle. This means that a circle does not, in principle, create a peak in the Hough domain; as we forecast.

A noteworthy feature of the transform of the circle is the fact that its diameter corresponds to the width of the strip formed by the sinusoidal curves in the Hough domain, as highlighted in Figure 3.4(b). This fact is easily understandable. Given a direction $\theta$, let us imagine a line moving through the circle in this direction, i.e., moving it in parallel. In a particular $\rho$, if the line intersect the circle, the bin corresponding to that line will have at least one vote. In contrast, if the line does not intersect the circle, the bin will have no vote. Therefore, if we sweep all possible $\rho$ values for a particular direction, the set of values that create at least a vote in the transform is exactly the diameter of the circle. Translating this explanation into the Hough domain, it is clear that, for a particular value of $\theta$, the width of the strip in $\rho$ corresponds exactly to the diameter of the circle in that direction.

Let us move on to a more general case, an ellipse. Figure 3.5(a) shows a set of points on an
ellipses and in (b) the sinusoidal curves for each of them in the Hough domain.

![Diagram showing Hough transform of an ellipse](image)

Figure 3.5: Theoretical example of Hough transform of an ellipse: (a) original image and (b) Hough domain with the sinusoidal curves for all the points.

The result is very similar to the case of the circle, except that the width of the strip in the Hough domain is not constant in this case. This is related to the fact that the diameter of the ellipse is not constant in all directions. Generalizing the result of the circle, we can prove that:

- The width in $\rho$ of the strip in the Hough domain, at a given direction $\theta$, corresponds to the diameter of the ellipse in that direction.

This way, in the example of Figure 3.5, the horizontal diameter corresponds to the width of the strip for $\theta = \pi$, which is 55. The same way, the width of the strip for $\theta = \pi/2$ corresponds to the vertical diameter, which is 20.

Having a closer look to the transform, we notice that there is an accumulation of votes around the strip borders in $\theta = \pi/2$, and in general, around the strip borders. Recalling the case of the circle, we also notice how there is an accumulation of votes in the borders, but equally distributed among all directions.

To study this effect deeper, let us illustrate the Hough transform in the usual way, i.e., with an image where each color represents the number of votes received in a particular bin. Figure 3.6 depicts the Hough transform of the circle and ellipse presented before, using the representation explained. Note that the images transformed correspond to the shapes presented before but instead of having just a few points, they are completely outlined.

The first thing to note is that votes are clearly accumulated along the border of the strip,
3.4 Detecting Ellipses in the Hough Transform for Straight Lines

![Figure 3.6: Comparison of the Hough transform of (a) a circle and (b) an ellipse](image)

as we forecast in the previous discussion. This fact has also an intuitive explanation.

Before starting the elucidation, let us recall that we discretize the Hough domain in a grid of sizes $\Delta \theta$ and $\Delta \rho$. This way, the number of votes of a bin does not correspond exactly to the number of points of the original shape a particular line passes through, but all the set of lines contained in the range of $\Delta \theta$ and $\Delta \rho$ in that bin. Assuming $\Delta \theta$ negligible, which can be acceptable if we make it small, the process we used to illustrate how the diameter of the circle was related to the size of the strip in the Hough domain can be translated to, instead of moving a line through the original image, moving a strip of width $\Delta \rho$, as depicted in Figure 3.7.

![Figure 3.7: Illustration of the process of line sweeping for circles and ellipses: Note the different amount of points the strip passes through, depending on its distance to the origin of the shape](image)

This way, for a particular $\theta$, let us imagine a strip of width $\Delta \rho$ passing through the center of a circle and through its border (left hand-side of Figure 3.7). The amount of points the line passes through when it is in the center is considerably lower than the amount when the strip is positioned on the border of the circle. This way, the number of votes in the borders of the Hough strip is higher than the number of votes in its middle, as has been observed in Figure 3.6(a).
This effect is even more noticeably for the case of the ellipse in Figure 3.6(b). In this case, for \( \theta = \pi / 2 \), the size of the border is significantly larger than in the case of the circle (note the limits of the color bar). For other values of \( \theta \), however, the value of the border is much lower. Note that this is not the case in the circle, where all the border has exactly the same value.

The explanation may be found on the right hand-side of Figure 3.7. When we sweep using a strip in the direction of the largest axis of the ellipse, the number of points passed through in the border is considerably larger than in the case of the circle, and specially larger than in its center. When sweeping with the strip in the direction of the smallest axis, however, the variation of the number of points is less significant. In contrast, since the shape of the circle is the same for all directions, the size of the border of the strip in the Hough domain of a circle is the same for all \( \theta \).

To sum up, ellipses create a strip in the Hough domain, whose borders form two crests with their maxima in the direction of the largest axis of the ellipse.

It is straightforward to show that, the Hough transform of half an ellipse or circle creates only one of these crests. Reproducing the sweep procedure as done before, imagining that the ellipse is cut by its largest axis, it can be seen that one of the crest is the same than in the whole ellipse. The other border, however, will not be in the Hough domain.
3.5 Hough Transform Filtering

Line and ellipse detection is not that easy when dealing with *real* images, due to the noise that affects the line draft masks and the interaction between different lines and ellipses. Searched peaks or crests in the Hough domain are potentially flattened and masked, and spurious peaks appear. Figure 3.8 depicts an example in a real-word scenario.

In order to understand what we are dealing with, let us plot with better detail an area of interest of the Hough transform in Figure 3.8(b). A cut of the Hough transform in the area where some peaks are found is shown in Figure 3.9(a). As we may observe, a large amount of noise corrupts the actual peaks, and part of it could be easily miss-classified as peaks.

In order to solve this issue, the proposed algorithm *cleans* the Hough transform using a high-pass filter, which was manually adjusted to the following impulsional response:

\[
1 - \frac{1}{10}
\]

In short, each filtered value corresponds to the original value at that position minus the double of the mean of a neighborhood in the $\rho$ direction. This way, a peak that is high but its neighborhood is comparably high will be almost erased. On the other hand, a peak that is higher that its neighborhood will be less reduced, so it will be highlighted with respect their neighbors. Note that the filter is unidimensional and is applied in the $\rho$

![Figure 3.9](image)

**Figure 3.9:** Hough transform detail in an area of interest: (a) before filtering and (b) after filtering
direction, given that the peaks, and specially the crests, are more abrupt in this direction and therefore easily detectable.

Figure 3.9(b) depicts the Hough transform of the example after being filtered. The noise has been reduced noticeably while keeping the peaks of interest almost unmodified. Note how the second peak, for instance, is clearly more contrasted than before with respect the noise in its neighborhood.

The filtered result is thresholded, obtaining a binary image where ideally there should be a point for each detected line and a crest for each piece of ellipse. In a real case, however, this image contains a set of point clouds, each of which refers to a single line or ellipse. Figure 3.10(a) depicts, in black, the resulting binary image of the Hough transform in Figure 3.8(b), in the area of interest, after thresholding. Note that there are different clouds, each of which formed by a number of points.

We must therefore cluster these clouds in order not to have many detected lines for each original line. The approach we follow is based on a simple morphological filtering. We dilate the binary transformed image aiming at grouping all the points of a cloud, but without merging points from different lines or ellipses. The structuring element used is an

![Figure 3.9(b)](image_url)

**Figure 3.9:** The Hough transform of the example after being filtered. The noise has been reduced noticeably while keeping the peaks of interest almost unmodified.

![Figure 3.10](image_url)

**Figure 3.10:** Thresholding of the Hough transform: (a) black spots correspond to the binary thresholded image while the red areas represent the clustered clouds; (b) Hough transform in the thresholded areas where we depict the peaks and the crests.
3.6 Hough Instances Classification

ellipse rotated 45°, given the characteristics observed in the Hough transform of the soccer field lines. Figure 3.10(a) show, in red, the cluster made out of the point clouds, after the morphological filtering. Figure 3.10(b) may give us a clear view of the data. It represents the Hough values in the clusters presented before. As we will see below and is clearly noticeable, there are six peaks and a crest, according to the fact that the original image (Figure 3.8(a)) has six different lines and a piece of ellipse. Note the difference in shape between a crest and a peak.

Let us denote each cloud as a Hough instance, since they are originated by an instance of a line or ellipse in the original image. For each instance, a representative is chosen, as the coordinate of the maximum value within the cluster.

Summarizing, we filter the Hough transform to remove as much noise as possible, we binarize it and we cluster the result forming a set of instances, each of which represented by the coordinates of its maximum.

In principle, the representatives of the clusters refer to every line and piece of ellipse in the image.

3.6 Hough Instances Classification

Until now, we have detected a set of instances in the Hough domain, i.e., a set of point clouds and its representatives. Each of these instances refer to a line or an ellipse. The following step is, therefore, classifying each instance between coming from a line or referring to a piece of ellipse. Obviously, there might also appear false detections we should discard. On top of that, we are interested in separating the set of straight lines between two sets of parallel lines, as we will motivate later.

Let us start tackling a specific type of shot of soccer scenes: the wide shot of the goal area (see an example in Figure 2.1(a)) taken from the side of the field. The reason for concentrating on this case is twofold. First, the large number of lines visible in this situation makes it easier to classify the instances and so it is a good starting point. Apart from that, this type of shot is clearly the one that contains the most relevant information about the match: the goals. From a point of view of content understanding, therefore, it is a good idea to devote our efforts to this type of shot. Once this situation is solved, the algorithm will be improved to consider more general situations.

A soccer field has a very distinctive pattern of lines, as presented in Section 3.3. In the case of shots of the area goal we may take advantage of two distinctive properties: the vanishing line is usually above the image, and the vanishing points for each set of lines have different
3.6 Hough Instances Classification

signs in its $u$ coordinate. Translated into the Hough domain in slope-intercept form, this means that the two sets of lines form two lines that intersect in the negative part of the $n$ axis and their slopes have different sign. In the Hough domain in normal form, this pattern also holds as: two sets of points, one with decreasing values of $\rho$ when increasing $\theta$ (those with $\theta < 90$) and another with decreasing values of $\rho$ when decreasing $\theta$ (those with $\theta > 90$).

Having this in mind, the algorithm to extract the lines among all the instances is as follows. First, instances are separated into two groups: those with $\theta < 90$ and with $\theta > 90$ (One of them will refer to the vertical lines of the field, and the other to the horizontal ones, apart from the false-detection or ellipse instances). After that, we sweep the former group in increasing values of $\theta$. Each instance of this group must have, if it is a line, a lower value of $\rho$ than the previous instance. If it does not, it is classified as non-line, i.e., being caused by a false detection or by an ellipse. For those instances with $\theta > 90$, the algorithm is the same but the sweep is done in decreasing $\theta$.

Finally, the set of lines with a mean $\theta$ value closer to $90^\circ$ is considered horizontal, and the other, vertical. This fact clearly holds in the majority of the shots taken from the side of the field. This classification allows us to already know which goal we are focused on. If the horizontal lines have $\theta > 90$, the focus is on the right goal. In the other case, the focus is on the left goal.

Figure 3.11(a) depicts the detected instances in the Hough domain in normal form, classified between vertical lines in blue (in this case $\theta > 90$ - so we are focusing on the right goal), horizontal lines in green (in this case $\theta > 90$), and non-lines in black. Note that this classification has allowed us distinguishing an instance created from a piece of ellipse, without further analysis of the instance itself. Figure 3.11(b) shows the original image with the overlaid instances in the same color. We corroborate that the non-line instance comes from a little piece of ellipse.

Note that the resolution of the image is $r_U=720$ pixels and $r_V=576$ pixels, so one can check the correctness of the parameters detected approximately. The black line, for instance, has a $\rho$ value near to $r_V$, since it is very close to the image bottom. Recall the definition of $\theta$ and $\rho$ in Figure 3.1(b) to fully check the coherence of the line parameters.

Let us move on to another type of shot of soccer scenes: the wide shot of the middle field (see an example in Figure 2.1(b)) taken from the side of the field. In this case, the influence of the central circle is remarkable. The feature in which we base our procedure to distinguish this type of shot from the goal area is the fact that there is an almost-vertical line. This way, we check whether there is a line around $\theta = 180$ (and $\theta = 0$). If so, the shot...
is considered as middle field. We base our search on the vertical line because those are never caused by ellipses, since their main axis is always almost horizontal.

The algorithm then follows looking for the horizontal line. In order to distinguish it from the instances coming from the ellipse, we look for the lines in a small strip around $\theta = 90$. Among these, the horizontal line is taken as the one with lowest $\rho$ value, i.e., the one in the highest position in the image.

Figure 3.12(a) shows the results of this algorithm on a middle-field shot, and (b) the orig-
inal image with the detected instances overlaid. If we observe the instances classified as *non-lines*, we clearly distinguish the crest pattern we introduced previously. In particular, if we intersect each crest with the line $\theta = 90^\circ$, and compute their distance in $\rho$, we will obtain the size of the smallest axis.

### 3.7 Vanishing Points Computation

When at least three lines are detected, two from a direction, and another one from the other direction, we are able to compute the vanishing point of each direction as presented below.

First of all, we translate the coordinates of the detected horizontal and vertical lines to the slope-intercept form. As introduced in the example of Figure 3.3, the points coming from the set of parallel lines should be on the line coming from the vanishing point. In a real case, however, these instances will not be exactly collinear. Thus, we could compute the vanishing point of both group of lines as the linear regressions of their coordinates in the Hough domain.

In most of the cases of real soccer broadcast, the camera is assumed not to swing. As explained in [WT91], the vanishing line in this case must be horizontal. Translated into the Hough domain, this means that the point referring to the vanishing line must have $m = 0$. Given that this point comes from the intersection of the lines referring to the vanishing lines, this intersection must lie on the axis $m = 0$.

This way, the computation of the vanishing points will be performed as two linear regressions of the two sets of parallel lines parameters in the Hough domain, restricted to having the intersection point on the $m = 0$ axis.

Following we illustrate the process of computation of the vanishing points of the example in Figure 3.11. The detected instances are first translated into slope-intercept form $(m, n)$. Two linear regressions are then performed restricted to having its intersection point in $m = 0$, to ensure that the vanishing line is horizontal. Figure 3.13 depicts the result, with the six points referring to the six detected lines, and two linear regressions intersecting in $m = 0$. The vanishing line would be, therefore, $v_\infty = -533$; which is intuitively coherent with the image content, since it is a line above the image at a reasonable distance (recall that the image resolution is 720×576). The slope of the green line is 1505, so according to the result presented in Theorem 3.1, the vertical vanishing point is at $u_v = -1505$. The horizontal vanishing point would be at $u_h = 9377$, since the slope of the blue line is -9377. This result is also coherent, since the vertical vanishing point is at the left of the image.
(u < 0) and the horizontal one is at the right (u > 0), considerably farther.

**Figure 3.13**: Vanishing points computation via two linear regressions restricted to intersect in $m = 0$. 
Chapter 4

Recognition of Detected Instances

The output of the previous blocks is a set of instances, classified into vertical and horizontal lines, and false detections or pieces of ellipses. Recalling that our final goal is to obtain the point of view of the scene, i.e., the coordinates where the scene is taken from; it is clear that the following step is to recognize the detected instances. In other words, we need to match the detected lines to the field line model we have a priori. As we will discuss in the following chapter, this will allow us to calculate the camera point of view.

The cornerstone of our procedure is the computation of the so-called incidence structure for each intersection point between two lines. This allows us to recognize each visible intersection point in the image, and therefore, also the lines involved. Following, we expound on the procedure to classify and recognize the detected lines and their intersection points, according to a field model, and taking advantage of a compact and useful way to represent these data.

4.1 Intersection Point Classification

The first step of this block is the detection and classification of visible field-line intersections. In order to do so, all the intersections between horizontal and vertical detected lines are considered as potential intersection points. Note that among these intersections we may find virtual points coming from the intersection of the lines but not from the actual segments that originated the lines [FKdWE04]. Our procedure follows by checking whether the segments along the lines are real, i.e., for each pair of consecutive intersection points along each line, the segment between them is said to be real if a significant percentage of it actually appears in the line mask. Once we have all the real segments,
we calculate, for each potentially real point, their incidence structure, i.e., we compute the number of real segments that meet the point and from where they do it (vertically or horizontally, and from below or above). This way, we can assign each point an incidence structure among the following ones, where each graph depicts the directions from where the real segments meet the point:

These incidence structures are the only ones that may be found in a soccer field, and therefore, the points that do not match any of the structures presented are considered virtual points and ignored. Note that, since the detection of the real points is not performed locally, but via the intersection of real segments, the algorithm is very robust to occlusions. Figure 4.1(a) shows the detected real intersection points in a case of goal view and their incidence structure. The robustness of the algorithm is illustrated in this example, since two of the detected points are completely hidden by players.

### 4.2 Representation of Points, Lines, and Incidence Structure

All the detected points come, as explained before, from the intersection between a horizontal and a vertical line. Thus, we designed a compact representation of a set of lines and intersection points via a matrix, whose columns (rows) refer to the vertical (horizontal) lines. The elements of the matrix are the incidence structure if the two lines actually intersect and blank if there is no real point at this position in the image. Note that this representation is possible thanks to the line classification and sorting we have obtained directly from the Hough domain. Figure 4.1 depicts: (a) the detected intersection points of the example image and their incidence structure, (b) the compact matrix representation of the detected lines and points, and (c) the complete field representation, which we denote as field matrix.

### 4.3 Recognition of Points and Lines

As introduced, the next objective is to recognize the detected lines and points, i.e., we want to know to which points and lines of the real soccer field model they refer. Using the representation presented above, this problem may be informally stated as finding the best matching between the columns and rows of the detected matrix and a subset of the ones of the field matrix (in the example: Figure 4.1(b) and (c)).
4.3 Recognition of Points and Lines

Let us assume we have detected $n_h$ horizontal and $n_v$ vertical lines, and so the detected matrix has $n_h$ rows and $n_v$ columns. Then, this matrix is compared with all the sub-matrices of the field matrix of the same size. The comparison is performed via a similarity measure that penalizes the number of different incidence elements between them and the separation of the rows and columns. The latter factor is added to prioritize those configurations in which the detected lines are consecutive in the field, since detecting consecutive lines is more probable than detecting separated lines.

The column and row selection that provides the most similar sub-matrix is considered as the resulting matching of lines and points, i.e., we link each point and line of the image with one of the field model. This allows us, for instance, to know the real world coordinates of the detected points. Figure 4.1(c) presents, in red, the matched lines and points from the field matrix. Note that the matching is correct, although it is not a perfect matching, since the corner of two lines is not visible from this perspective.

**Figure 4.1:** Intersection point recognition and incidence structures: (a) Original image with the detected points and their incidence structure. (b) Compact representation of the detected lines and the incidence structure of the intersection points. (c) Matrix representation of all the lines and intersections of a soccer field; in red, the matched lines and points
Chapter 5

Point of View Extraction

At this point, given an image, we have presented an algorithm to detect a set of instances in the soccer field, namely straight lines and their intersection points. We have also provided a way to recognize them, i.e., to match them with a field model, providing us with the real-world coordinates of each detected point and line in the image. The vanishing points of the two sets of parallel lines in the soccer field may also be obtained.

Gathering all these results, our objective is to calculate the point of view of the scene, or, in other words, we want to know from where the image was taken and where it was focused. Technically, we want to calibrate a camera model automatically.

In this chapter, we first define all the needed frames of reference to formalize the process of point of view extraction, and deduce the projection equations, i.e., the equations that relate the coordinates of the 3D-world points with the ones of their image projections. Finally, we present the algorithm that achieves our goal of automatic camera calibration, along with its theoretical derivation.

5.1 Frames of Reference and Projection Equations

This section defines the coordinate systems we use in this work, including the 3D field and camera coordinates, and the 2D image plane coordinates and sampling. These frames of reference are used later to describe the projection of the image and are tailored to a football stadium scenario.

All the frames of reference are described as Euclidean frames of reference, i.e., containing a point known as origin and \( n \) vectors that are commonly referred as axes. This allows us
5.1 Frames of Reference and Projection Equations

Figure 5.1: Scheme depicting the frame of reference considered, their position on the field, and the position of the points of interest

grasping the meaning of all the coordinates, although in some parts we will be working with the corresponding coordinates of the projective space for convenience, as we will and motivate later.

5.1.1 3D global frame of reference

Let us define a frame of reference with its origin $O$ placed at the intersection between the line at the middle of the field and one of the side lines. The axes $(x, y, z)$ are orthonormal and oriented in a way that the field is the $x – y$ plane and the $z$ axis represents the height with respect to the field. Figure 5.1 depicts the scenario we are presenting. We will refer to this frame of reference as $[O, (x, y, z)]$.

Let $C$ be the position of the camera (specifically, its focal point). In terms of the reference introduced before, we denote $C = (C_x, -C_y, C_z)$. The reason to introduce the minus sign in the $y$ component is to keep the $C_y$ parameter positive, since the camera will usually be placed in the negative part of the $y$ axis. In plain words, $C_x$ is the displacement of the camera with respect to the middle of the field, $C_y$ its distance to the side of the field, and $C_z$ the height at which the camera is placed.

Next, we define $f$ as the point on the field on which the camera is focused, or, in other words, where it is pointing at. Its coordinates are defined as $f = (x_f, y_f, 0)$. Note that we only consider focus points with $z = 0$, i.e., being on the field plane. According to this definition, $x_f$ is the distance between the focus point and the middle of the field and $y_f$ its distance to the side. Figure 5.1 graphically represents all the parameters defined before.
5.1 Frames of Reference and Projection Equations

5.1.2 Camera model and its frames of reference

The next step we take is to define the camera model, with respect to a frame of reference adapted to this model.

In the same 3D scenario considered before, the camera frame of reference has its origin at the focal point of the camera \(C\) and its axes \((x', y', z')\) placed as presented in the Figure 5.1. The \(y'\) axis is placed along the line between \(C\) and \(f\), i.e., it defines the direction at which the camera points.

The orientation of the camera is parameterized by three angles: \textbf{pan}, \textbf{tilt}, and \textbf{swing}. In short, pan is the rotation with respect to the \(z'\) axis, tilt with respect to \(x'\), and swing with respect to \(y'\). Figure 5.2(a) depicts the pan angle, which we will denote as \(\alpha\), and (b) the tilt angle \(\beta\). The figure also depicts the relation of these angles with the parameters of the 3D frame of reference of Figure 5.1. In our work, the swing angle is supposed to be 0, since soccer broadcasts force it to ensure that the image looks horizontal.

Figure 5.3 depicts the camera model we consider, with respect to the camera frame of reference \([C, (x', y', z')]\). This model is known as \textit{pinhole camera with no lens distortion}.

We define the image plane as a rectangular plane parallel to the \(x'-z'\) plane and placed at a distance \(F\) with respect to the focal point. (\(F\) is known as \textit{focal length}.) The dimensions of the image plane are the dimensions of the camera sensor that records the images. We will denote them as \(D_w\) and \(D_h\), that stand for dimension in width and height, respectively. In analog cameras, for instance, it is very common to use photographic film that is 35 mm in width (\(D_w = 35\) mm).
The intersection point between the image plane and the $y'$ axis is denoted as $f'$, since it is the projected point of $f$. Its components with respect to the camera frame of reference \([C, (x', y', z')]\) are \(f' = (0, F, 0)\).

We then define a 2D frame of reference inside the image plane. The origin \(o'\) is placed at the upper left corner of the image. The axes \((u', v')\) are parallel to the image plane and are also considered orthonormal.

5.1.3 Sampling of the image plane

The last step is to consider the sampling of the image plane in pixels. Let \(r_U\) and \(r_V\) be the image plane resolution in pixels in both dimensions (horizontal and vertical respectively). The origin will also be placed at \(o'\) and we denote the pixels coordinates as:

\[
[u, v] \quad \text{for } u = 1 \ldots r_U \quad \text{and} \quad v = 1 \ldots r_V.
\]

Then, the position for each of them is assumed to be the one of its center, i.e., the position of the pixel \([u, v]\) in the frame of reference of the image plane \([o', (u', v')]\) is:

\[
\begin{align*}
u' & = \left( v - \frac{1}{2} \right) \frac{D_w}{r_U} \\
u' & = \left( v - \frac{1}{2} \right) \frac{D_h}{r_V}
\end{align*}
\]

where \(\frac{1}{k_V} = \frac{D_h}{r_V}, \frac{1}{k_U} = \frac{D_w}{r_U}\) are the sizes of a pixel.

Forcing the pixels to be square \((k_U = k_V = 1)\), given a certain image resolution \(r_V, r_U\), and assuming that the width of the photographic film is \(D_w = 35\) mm, the vertical size of the photogram is set to:

\[
D_h = \frac{D_w}{r_U} r_V = \frac{r_V}{r_U} 35 \text{ mm}.
\]
5.2 Camera Calibration Via Vanishing Points

5.1.4 Projection equations

Once we have defined the camera model and the frames of reference, the following step is to derive, first, the projection equations that relate the coordinates of a pixel in the image to the projected point in the field plane. The other way around, we also provide the equations that relate the coordinates of a 3D-world point to its projected point in the image.

The former equations, although known and studied since long time ago, have been deduced from scratch in the Euclidean space, since we have found it enriching to fully understand all the steps involved. The later ones are provided in the projective space as will be motivated later.

In order not to disrupt the thread of this dissertation, all the equation deduction is presented in Appendix B.

5.2 Camera Calibration Via Vanishing Points

As introduced, the camera model used in this work is parameterized by six values: pan, tilt, zoom, and position in the 3D-world coordinates $C_x$, $C_y$, and $C_z$. Recalling that our objective is obtain the point of view of the scene, this can be translated into calculating these six parameters. As we will show below, the first three may be obtained from the coordinates of the vanishing points of the two sets of lines of the field plane. In order to calculate the position of the camera in the 3D world, i.e. the three last parameters, we need also to recognize at least two points of the scene.

5.2.1 Pan, tilt, and zoom from vanishing points

In [WT91] the authors present a methodology to calibrate the pinhole camera based on the computation of the vanishing point of three pairs of parallel lines. In our case, however, the scene only has two pairs of parallel lines, and therefore two vanishing points. On the plus side, as introduced previously, the cameras of soccer matches are forced not to swing.

Following, we derive a new way to obtain the pan, tilt, and zoom parameters from two vanishing points when there is no swing.

The deductions will be done in the context of projective geometry for many reasons, as motivated in [Fau01]: homogeneous coordinates usually make the algebra linear, and in particular a perspective projection is easily representable projectively. In our case in particular, on top of that, handling vanishing points is simple and intuitive in the projective space while tedious and laborious in Euclidean geometry.
5.2 Camera Calibration Via Vanishing Points

Let us give some properties of the projective space $\mathbb{P}^3$, which are used in the following derivation. First, a common representation of a point in $\mathbb{P}^3$ is $[\xi_1, \xi_2, \xi_3, \xi_4]$, not all zero, which are called homogeneous coordinates. The reason for the use of the “$[]$” symbol is to denote that the coordinates of a point are defined up to a non-zero scale factor, i.e., for any $\lambda \in \mathbb{R}$, $\lambda \neq 0$, $[\lambda \xi_1, \lambda \xi_2, \lambda \xi_3, \lambda \xi_4]$ represent the same point in $\mathbb{P}^3$.

Any point $(x, y, z) \in \mathbb{R}^3$ is usually represented in $\mathbb{P}^3$ as $[x, y, z, 1]$. The other way around, any point in $\mathbb{P}^3$ with $\xi_4 \neq 0$ corresponds to $(\xi_1, \xi_2, \xi_3, \xi_4) \in \mathbb{R}^3$. The vanishing plane of $\mathbb{P}^3$ is represented as $\xi_4 = 0$. The vanishing point of a direction $(v_1, v_2, v_3)$ in $\mathbb{R}^3$ is $[v_1, v_2, v_3, 0]$.

Note the coherence between the definition of the homogeneous coordinates up to a factor and the fact that any vector $\lambda(v_1, v_2, v_3)$ in $\mathbb{R}^3$, with $\lambda \neq 0$, defines the same direction and so has the same vanishing point. The analogous relations hold for $\mathbb{R}^2$ and $\mathbb{P}^2$ in the case of the image plane.

We are interested in the two directions of the straight lines in a soccer field, which we denote as horizontal and vertical. In our frame of reference, these directions are $(1, 0, 0)$ and $(0, 1, 0)$, respectively. The vanishing points of these directions are, therefore, $[0, 1, 0, 0]$ and $[1, 0, 0, 0]$.

Let us assume we have computed the image coordinates of the vanishing points of both vertical and horizontal lines, as presented in Section 3.7. We denote them as $\infty_v = (\infty_{v1}, \infty_{v2})$ and $\infty_h = (\infty_{h1}, \infty_{h2})$, respectively.

From Equation B.5 (see Appendix B for the equation deduction), we have that, for any point in the projective space $\mathbb{P}^3$, its projection to the image projective plane $\mathbb{P}^2$ is the following:

$$[\mu_1, \mu_2, \mu_3]^t = P [\xi_1, \xi_2, \xi_3, \xi_4]^t$$

Then, its coordinates in the image plane $\mathbb{R}^2$ are $(u, v) = \left(\frac{\mu_1}{\mu_3}, \frac{\mu_2}{\mu_3}\right)$.

In particular, let us calculate the coordinates in the image projective plane of the vertical and horizontal vanishing points:

$$[\mu_{v1}, \mu_{v2}, \mu_{v3}]^t = P [0, 1, 0, 0]^t \quad [\mu_{h1}, \mu_{h2}, \mu_{h3}]^t = P [1, 0, 0, 0]^t$$

And so the pixel coordinates of the vanishing points will be:

$$\infty_v = (\infty_{v1}, \infty_{v2}) = \left(\frac{\mu_{v1}}{\mu_{v3}}, \frac{\mu_{v2}}{\mu_{v3}}\right) \quad \infty_h = (\infty_{h1}, \infty_{h2}) = \left(\frac{\mu_{h1}}{\mu_{h3}}, \frac{\mu_{h2}}{\mu_{h3}}\right)$$

Substituting the value of $P$ obtained in Appendix B, we get:

$$\infty_{h1} = \frac{Fk_U}{\tan \alpha \cos \beta} + \frac{1}{2}r_U \quad \infty_{h2} = -Fk_V \tan \beta + \frac{1}{2}r_V$$
5.2 Camera Calibration Via Vanishing Points

\[ \infty v_1 = -Fk_U \tan \alpha \frac{1}{\cos \beta} + \frac{1}{2} r_U \]
\[ \infty v_2 = -Fk_V \tan \beta + \frac{1}{2} r_V \]

Note that this result is coherent with the fact that we imposed that the camera does not swing, since the \( v \) coordinate of both vanishing points is the same, i.e., the vanishing line is horizontal.

A little bit of algebraic manipulation leads us to the values of **pan, tilt, and zoom from the vanishing points** of vertical and horizontal lines:

\[
\alpha = \text{sign} \left( \infty h_1 - \frac{1}{2} r_U \right) \tan^{-1} \left( \sqrt{\frac{-\infty v_1 - r_U / 2}{\infty h_1 - r_U / 2}} \right)
\]
\[
\beta = \sin^{-1} \left( \frac{\infty v_2 - r_V / 2}{\infty v_1 - r_U / 2} \cdot \frac{k_U}{k_V} \cdot \sqrt{\frac{-\infty v_1 - r_U / 2}{\infty h_1 - r_U / 2}} \right)
\]
\[
F = -\frac{\infty v_2 - r_V / 2}{k_V}
\]

### 5.2.2 3D-world coordinates from recognized points

Once pan, tilt, and zoom have been obtained, we use the method presented in [WT91] to calculate the position of the camera. It mainly needs the matching between two points of the image to their real-world coordinates.

Generally, our method has more than two recognized points in the scene. In these cases, we compute the position parameters for each pair of them and compute the mean of all the obtained positions.

Figure 5.4 shows the result in three different frames. The images show the original frame with the line model overlaid, projected using the camera parameters obtained. Since the model matches with the actual lines in the images, it can be stated that the result is correct.

![Figure 5.4: Calibration results: Line model overlaid to the actual image from the computed point of view](image-url)
The techniques presented until this point are based on still images, i.e., they handle each frame of a sequence independently. Consecutive frames, however, usually have a high correlation between their content. One could think of taking advantage of this correlation in various ways.

In coding of sequences, for instance, it is usual not to code the frame content itself, but some information that allow us to obtain it from previous or following frames. As an example, some block matching algorithms divide a particular frame in squared blocks, and the most similar block is searched in previous frames for each of these. Then, instead of coding the block itself, one usually codes the displacement in pixels done to the block to reach the most similar one and the error between them. Thanks to the high correlation between frames, the amount of information to be coded is reduced considerably and therefore the compression achieved is high.

In our context, the approach is different: we want to take advantage of the temporal redundancy to improve the performance of our algorithm. If it is used as the basis of content retrieval in soccer broadcast, for instance, we want our algorithm to work as close to real time as possible. This way, match statistics may be presented with an as little as possible delay.

Thus, our approach is to take advantage of the temporal redundancy to save computational effort, by reusing some parameters computed previously that in consecutive frames do not change significantly.

We propose to divide the frames in a sequence into different types, depending on the computational effort we devote to their processing. The less computational effort we devote
to a frame, the more results we reuse from previous ones, to the potential detriment of the accuracy of the calculated parameters. Drawing a parallel with video coding, in that case the frames are divided depending on the amount of bits used to code them, having an impact on the quality of the decoded result. Specifically, we propose four different levels of frames:

- **Level 1** is the most accurate but entails the highest computational effort. It consists of computing all the parameters from the frame itself, without reusing any information.

- **Level 2** consists mainly in reusing the position of the camera from the previous frame, thus avoiding the process of recognizing the instances in the image, as we will see below.

- **Level 3** is based on a tracking process of the detected lines, provided that their position has not changed significantly.

- **Level 4** simply performs a prediction of the PTZ (See Notation in page vi for clarification) parameters based on the previous values, so it is the least accurate.

Following, we motivate and give a detailed description of all these levels of frame processing. Finally, an overview of how the algorithm decides which level to use for each frame is presented.

### 6.1 Level 1

As introduced, this level does not take advantage of any previous computation. Given a frame, it computes all the camera parameters from scratch, involving all the steps presented in this work.

Figure 6.1 depicts the whole process, from the original frame preprocessing to the storage of xyz and PTZ for their use in following frames. Let us take a look at the blocks involved,
putting special emphasis on the parameters needed for each of them to work, since this will help us understand the following levels.

First of all, our input is a frame of a sequence, which we first preprocess to extract the line mask. The following step is to calculate the Hough transform of the line mask and exploit its result to obtain two sets of line instances, still not recognized, as presented in previous chapters. This information is first used to compute the PTZ parameters (recall that this algorithm did not need to recognize the instances). The two sets of instances are then recognized to compute the xyz parameters, gathering also the PTZ parameters computed previously.

### 6.2 Level 2

Observing the last steps of the Level 1 procedure, one could think of skipping the point-recognition step, thus avoiding all the computational load it entails. The xyz parameters cannot be computed without recognized points, so we propose to reuse the xyz parameters from the previous frame, which is acceptable since the position of the camera is usually static. Recall that we are working with the output from a shot classifier, so this assumption is acceptable.

Thanks to the fact that the PTZ parameters computation does not need the instances to be recognized, the skipping of the recognition step does not prevent the algorithm to compute the PTZ parameters from the current frame, for the sake of accuracy.

Figure 6.2 depicts the scheme of this level, where the point recognition block has been removed thanks to the reusing of the xyz parameters from previous frames.

![Figure 6.2: Level 2 processing scheme, xyz parameters are reused from previous frames](image)

One could think of a slight modification of this algorithm consisting in the reusing of the xyz parameters not from the previous frames, but from the whole shot, or even from an a priori knowledge of the set of camera positions.
6.3 Level 3

The third level proposed takes a step forward in reducing the computational load of the algorithm, skipping the Hough transform and its further exploitation.

The main idea of this level is to track the lines along the frames, i.e., given the line mask of a frame and the lines from the previous one, estimate the new positions that best match the line mask.

The first approach we may think of would be to track the detected lines from the previous frames, i.e., the lines that were visible in the previous view. This has a clear drawback: we cannot handle the appearance of new lines due to the camera motion. Our proposal to solve this issue is to, given the camera parameters of the previous frame, reconstruct the line model and track the lines that are artificially generated. This way, if a line appears at some point, the reconstruction of the line model will actually have the line and so we will be able to track it in future frames.

Figure 6.3 shows the scheme proposed for this level, where we may observe that we track the modeled lines instead of the detected ones.

Regarding the last blocks of the scheme (xyz and PTZ extraction), it is important to note that, in contrast to Level 1, point and line recognition is not performed, although we compute the xyz parameters. This is possible thanks to the fact that we use the modeled line model, and so the lines and their intersection are already recognized.

Finally, let us describe how we perform the line tracking. At his Level, we have a set of modeled lines of the previous frame and the line mask of the current one. For simplicity, we track each line of the model independently. In short, the surrounding of each modeled segment is explored and the line that best fits the mask in that area is considered as the
new line.

Let us give a formal description of this algorithm. Each visible modeled segment is sampled in \( N_s \) different points \( p_i \), evenly spaced. Centered at each \( p_i \), a segment of length \( L_s \) and orthogonal to the modeled line is drawn and intersected with the line mask of the current frame. This way we obtain \( N_s \) sets of points \( \{q^j_{p_i} | j = 1 \ldots N_p \} \) at certain distances \( d^j_{p_i} \) from the modeled segment. Let \( \bar{d}_{p_i} \) be the median of the distances to the point \( p_i \) (The median is used in order for the algorithm to be robust to outliers due to noise in the mask).

Figure 6.4 depicts the process we are describing. In (a), a part of the mask from the new frame is shown, with the \( N_s = 20 \) sampled points of the original line \( p_i \) in red. Note how the segment (in blue) intersects the mask. In (b), the medians \( \bar{d}_{p_i} \) of the distances of the mask pixels to the line are plotted as red stars. Note how the last median is clearly an outlier, due to the fact that the corner of the line was hidden by a player.

![Figure 6.4: Illustration of the line tracking procedure: (a) original mask with the points from the previous line and the segments to explore the mask and (b) plot of the median of the distances and the interpolated line with (blue) and without (red) outliers](image-url)

The next step is to interpolate all the medians of the distances to obtain an estimation of the new position of the line. We want, therefore, to interpolate the set of points \( \{(i, \bar{d}_{p_i})\} \), as in Figure 6.4(b). The first approach is to interpolate all of them, but this example proofs the inaccuracy of this approach due to the outliers. In blue, we depict the interpolation of all the distances, without discarding outliers, which clearly distorts the result. In order to reject the outliers, we propose the following method.

Let \( \bar{d} \) be the median of all the \( \bar{d}_{p_i} \) values. We then discard the 20% percent most farther points do \( \bar{d} \). In other words, if we have \( N_s = 20 \) points (and so 20 distances), we discard
the 4 points that are farther of $\bar{d}$. In Figure 6.4(b), in green, we depict the result of the interpolation using this method, which is clearly more accurate thanks to the removal of outliers.

The position of the points $p_i$ is moved by the interpolated distances along the direction of the segments and the line that is formed is considered as the line of the new frame.

### 6.4 Level 4

The fourth level is the most simple of our algorithm, and it does not use the frame itself, but just the parameters obtained from the previous and following frames.

The idea is to, given a set of frames between two already-processed ones, lineally interpolate their parameters along the frames. Note that this part of the algorithm introduces a delay in the computation of the parameters, since it has to wait for a set of frames, but it can be negligible for many applications.

Figure 6.5 depicts the scheme for this level, where just the parameters of the neighbor frames are used: PTZ and xyz are interpolated.

![Figure 6.5: Level 4 processing scheme](image)

### 6.5 Level Scheduling

As we have seen, the different levels presented entail different computational load and provide different levels of accuracy. The last step to process a video sequence is to provide an algorithm of level scheduling, i.e., to decide which level processes each frame.

The way to do it clearly depends on the restrictions of each application. Whether we need the results in real time or not, or the accuracy demanded, for instance, will have a clear impact on the level we have to use. In order to give an example of how to schedule the different levels we will describe a realistic situation of soccer broadcast.

Let us imagine we know the exact position of the set of possible cameras in the field be-
forehand, which is clearly plausible. Once the first frame is processed by Level 1, and the xyz parameters of the camera are obtained, one could think of selecting the closest camera of the field and use the xyz parameters known in advance, which are known to be precise. From then on, the PTZ parameters could be updated in Level 2. At a certain interval, we can use Level 1 again to check whether we have changed of camera.

If the performance of our implementation is not fast enough to process each frame in Level 2, we can use Level 3, which is much faster but is potentially less accurate. Finally, we could process one frame out of $N$ with Level 1 or 2 and interpolate the results between each pair of them using Level 4, which would be an approximation of the accurate result.

The accuracy demanded by the application also influences the scheduling. If we need an accurate reconstruction to locate some objects of the scene, for instance, we cannot use Level 4. In contrast, if we just ask for an approximation of the point we are focused, the field where the camera points, for instance, a rough approximation of the camera parameters would be enough and therefore we could speed the process up by the use of Level 4.
Chapter 7

Experiments

The first part of this chapter analyzes the performance of the block involved in the algorithm and highlights the most relevant results at each step. The second part is devoted to giving two practical demonstrations of some parts of the algorithm.

7.1 Algorithm Testing

We tested our algorithm on a data set consisting of 100 key-frames of wide shots, selected from four real soccer sequences in different stadiums\(^1\). The interlace effect is clearly notorious in the majority of the key-frames, so it is an issue the algorithm must cope with.

The experimentation and testing of all the algorithms involved in the method presented was done in the high-level language Matlab\(^\circledR\), with a resulting code consisting of approximately 1800 lines.

Following, we describe the results obtained for each block of the algorithm. For each of them, we present the particular values of the parameters used (refer to the notation section in page vi to recall the meaning of these parameters), we highlight some relevant examples and draw some conclusions.

A result from a certain block is considered correct if it does not prevent the following block from working properly.

\(^1\)The “soccer” key-frames used in this work belong to MEDIAPRO, S.L., and are copyright protected. They have been provided by MEDIAPRO, S.L. with the only goal of research under the framework of the i3media project.
7.1 Algorithm Testing

7.1.1 Image Preprocessing

The parameters used as threshold of the grass binarization in the tests are $T_R = 4\%$ and $T_B = 3\%$. The results referring to the grass detector prove that the algorithm is **reliable, efficient, and robust**. The results are completely correct in 95 images. Actually, as introduced, grass extraction has been improved in the work of David Varas, which I advised. In his work, this block has been tested in 250000 frames of sequences from real broadcast of the last season of the Spanish soccer league. The results have been completely satisfactory.

Typical grass masks may be found in Figure 2.3(a,b). A more challenging situation due to shades is shown in Figure 2.3(c). The algorithm presented a good behavior in this situations of two colors in the grass.

Regarding the grass mask refinement, since only the size of the holes is taken into account to fill or keep them, there is a challenging situation in discerning between holes in the mask due to players in wide shots (which we want to keep) or the ones due to wide white lines in close shots as in Figure 2.3(c). The five errors found in the test were caused precisely by keeping the lines in the grass mask, avoiding the next steps to detect some of them. Figure 7.1 shows three of them.

![Figure 7.1: Grass detection wrong results, where lines are removed form the grass mask: (a,b) static shots of vertical lines and (c) closer line view](image)

In contrast to our intuition, interlace plays an important and positive role in this part of the algorithm. In common shots, due to the motion of the camera and interlace, lines are disconnected in patches and merged with the grass. This makes the algorithm to keep them in the mask, since the patches have a smaller area than the threshold we use.

When the camera is static, this effect does not take place and therefore there might be errors due to large areas of lines that are removed as players. This effect is worse in vertical lines, since they are wider. This is the case of Figure 7.1(a,b). Another possibility that may cause the algorithm to erase the lines is a closer view (among the wide views) of them, due to their width, as in Figure 7.1(c).
7.1 Algorithm Testing

7.1.2 Line Detection

The performance of the line detection and classification is also considered really positive, since the cases where a final mistake is caused by this step are limited to shots we still have not considered.

A correct result of a typical goal and central views may be observed in Figures 3.11 and 3.12 respectively, where the point clouds are correctly grouped, the horizontal and vertical sets of lines are properly grouped and, in turn, the line that does not follow this pattern are correctly ignored.

Figure 7.2 shows a sample of the errors found. In (a), two mistakes are made. First, the lines of the farther goal are missed, since they are too thin and short to be detected. Second, the white vertical posts are detected as lines and so mislead the classification algorithm. In (b), the green advertising panel with white letters misleads the detection step and creates a false positive. Finally, (c) depicts the non-detection of the central line because it was removed from the line mask in the previous step.

![Figure 7.2: Line detection wrong results, where lines are removed form the grass mask: (a) far lines are not detected and vertical posts mislead the classification algorithm, (b) advertisement panels create a false positive, and (c) bad results in grass detection erase one of the lines](image)

This errors are a clear example of the distance between a proof-of-concept algorithm and its applicability in a real-world scenario, specially the one in (a). While the idea behind the algorithm might be valid, there are many particularities to take into account to make it work in the real world, which do not fit in a Master Thesis.

7.1.3 Instances Recognition

The instance recognition algorithm is efficient and solves correctly some difficult situations such as player occlusions. Figure 4.1 depicts an example of corner occlusion but correct detection and classification. This makes us believe that the algorithm has solid basis.
7.1 Algorithm Testing

Despite this, it is the weakest link of the chain in our algorithm. Apart from the errors coming from line classification, the block itself misses a significant amount of frames mainly due to particularities of the views that have not been taken into account in the first place. It needs therefore to be improved to cope with all the variability that may be found in a real-world scenario.

On the plus side, the majority of the errors are easily solvable, by just introducing that particular situation into the casuistry. Our understanding is, again, that this refinement process is out of the scope of a research project such as this Master Thesis.

To sum up, we believe that the algorithm is a good proof of concept that shows promising results, but it has to be refined to be applied in real environments.

7.1.4 Point of View Extraction

The extraction of the point of view of the scene introduces no errors if the points and lines are recognized properly and at least three lines are visible to compute the vanishing points.

Regarding the accuracy of the calibration obtained, the more points we recognize, the better. Figure 5.4 shows the level of accuracy reached from a sufficient amount of points recognized. In the following section, the accuracy of the calibration is further analyzed to check its temporal consistency.

7.1.5 Temporal Redundancy

Regarding the results taking advantage of the temporal redundancy of the algorithm, we have performed 4 tests, each of them focused on each of the levels presented. In an actual scenario, all the levels could be combined. A sequence of 50 consecutive frames was chosen and a subset of 8 frames is depicted in each experiment.

**Experiment 1:** This first experiment shows the results of Level 1, i.e., without taking advantage of the temporal redundancy and processing each frame with the most costly level. Figure 7.3 shows the results on the 8 selected frames, were we may observe that the accuracy reached is sufficient for a wide range of applications.

**Experiment 2:** In this section we analyze the performance of Level 2, i.e., recomputing only the PTZ parameters at each step and reusing xyz. Specifically, we wanted to simulate the scenario where we know the position of the camera in advance, so we calculated the mean of the xyz positions extracted from Level 1 and we assumed it was the actual posi-
The camera is thus calibrated using this constant values of xyz and the computed PTZ at each frame.

The results with this approach, however, were not accurate enough. The reason for that is that the estimation of the PTZ parameters oscillates frame to frame. In the case of Level 1, this fact was compensated by an oscillation in the camera position. A clear example of this effect is that a too close estimated zoom can be compensated with a farther camera position.

Since in this experiment the xyz parameters were constant, the accuracy of the result oscillated with the PTZ estimation.

In order to overcome this situation, we smoothed the PTZ parameters estimation using a moving average filter of size 20. The accuracy with this approach was improved significantly.
7.1 Algorithm Testing

Figure 7.5: Level 3 calibration results on a subset of 8 frames

...cantly and the results obtained were very close to the Level 1 approximation. Figure 7.4 depict the result for this Level.

Experiment 3: The third experiment corresponds to Level 3, where lines are tracked from frame to frame. In this case, the initial frame was processed using Level 1, and the results were tracked along the sequence.

Figure 7.5 shows the results for this Level, where we may note that the accuracy reached is also really close to, or even better than, with Level 1. This fact does not mean that Level 1 is dispensable, since fast movements of the camera may cause Level 3 to get lost. It is recommendable, therefore, to use Level 1 often.

Experiment 4: Finally, the most simple approach is analyzed in this section, i.e., the interpolation of the xyz parameters between frames. Specifically, we have processed the two extreme frames of the sequence with Level 1, and we have interpolated the parameters to the frames in between.

Results are depicted in Figure 7.6. In contrast with our intuition, the accuracy reached is comparable to the one of the other levels. On top of that, we notice that, since the parameters obtained are completely smooth due to the lineal interpolation, there is no oscillation in the calibration, so the user’s experience is improved with respect to other levels.

We cannot, however, separate the frames processed with Level 1 too much, since a sudden change of direction of the camera would be ignored by this Level.
7.2 Practical Demonstrations

This Section demonstrates two possible practical uses of the algorithm presented. Both of them use partial results of middle blocks so not all the steps need to be computed to achieve these applications. Although useful in a real context, these applications are straightforward with the chain of results of our algorithm.

7.2.1 Player Markers

The first partial result we may think of exploiting is the grass mask. In this work it is used to extract the field lines and to classify the shots in the case of David Varas’ work. Another possibility might be to use the holes in it to obtain player markers. This information might be useful as a base for player tracking algorithms.

Figure 7.7 depicts three examples of this application, where the bounding box of each hole

![Figure 7.7: Player markers demonstration: the bounding boxes of the holes in the grass mask are overlaid to the original image](image)
in the image grass is overlaid to the original image. In (a), two close players are merged into one marker, which is understandable, since our algorithm does not study the content of the hole to separate two players. This issue is tackled by the tracking algorithms which model the color of the players to distinguish these cases, so the possible use holds. In (b), parts of the scoreboard and advertisement panels are considered as players. This is yet another example of the gap between a proof-of-concept algorithm and its applicability in a real-world scenario.

7.2.2 Outside Line

Going further in the processing chain, we might find a handy application to the coordinates of the vanishing point: plotting the outside line. The controversial outside rule of soccer is difficult to judge, since it would ideally need a line to be plotted on the field, in parallel to the field lines.

This idea has a simple translation in our context, when having the vanishing point of the vertical lines of the field: it consist of just plotting a line passing through the vanishing point and the player whose position is to be assessed.

Figure 7.8 depicts two examples of outside line calculated using this technique.

![Figure 7.8: Outside line plotting demonstration: The plotted line would be parallel to the field lines in the 3D world](image-url)
Chapter 8

Conclusions and Future Work

This Project has provided a new global approach to camera calibration, in pursuit of the automatic analysis of soccer sequences in the context of the i3media project. We take advantage of all the links in the chain towards the point of view extraction, simplifying each of them to achieve a better performance while obtaining enough accurate results for the specific purposes. The core of the method is a deep exploitation of the Hough domain properties, to provide a useful stream of data for scene analysis.

In particular, we achieve a robust grass extraction algorithm that successfully handles hard transitions in the field, yet being very simple and efficient. In this context, the author has been the advisor of a final degree's project towards the shot classification of soccer sequences, based on the grass layout in the image. As a result, the grass detector has been improved and tested on a large data set of 8 football matches, corresponding to 8 different football fields and leading to more than 250000 images of real-world broadcasts.

Based on a single Hough transform we detect the lines and ellipses in the field, as well as the vanishing points of the two sets of parallel lines. A theoretical background explanation is given that supports the algorithms used. Thanks to a clear line pattern in the Hough domain, we classify the detected lines into horizontal and vertical, and in turn we sort them in such a way that the following recognition step is eased. Thanks to the fact that cameras do not swing in real broadcasts, a robust way to compute the vanishing points is presented, taking further advantage of Hough domain properties that are mathematically proven.

Intersection points are detected and characterized in a novel manner that allows successful results even in case of occlusion. An intuitive representation of all these data allows a
simple and effective **point and line recognition** algorithm.

**Point of view extraction** is achieved by matching a simple pinhole camera model to the scene. Pan, tilt, and zoom parameters are extracted from the coordinates of the vanishing points, and the position parameters are obtained from the recognition of at least two points in the field. The separation of camera calibration in two steps allows us to speed the process up by skipping all the steps involved in the line and point recognition for some frames.

We take advantage of **temporal redundancy** by extrapolating some information from one frame to its temporal neighbors. Specifically, we define four different levels of analysis that can be combined depending on the application. Level 1 is the most computationally expensive since it performs all the computations for each frame. In Level 2 we reuse the xyz parameters from previous frames, since they do not change in a shot with static cameras. Note that this Level is possible thanks to the two-step calibration proposed. Level 3 is based on the tracking of the lines, in a robust way that processes each line independently thus simplifying the algorithm. The potential of this Level is that not only do we propagate the position of the lines but we also reuse their recognition. Finally, Level 4 allows us to process only a frame out of some and to interpolate the results between them.

In order to test the potential of our proposal, a set of **experiments** are presented. Each link of the chain shows promising features but has to be refined to cope with all the casuistry that one may find in a real-world scenario. Some configurations to handle video sequences are presented and it is shown that the accuracy obtained is sufficient for many of the objectives demanded in the context of our Project.

All in all, this project presents a proof of concept that validates a new approach to camera point of view extraction in soccer scenes that simplifies each step of the algorithm towards an improvement in efficiency.

The futures lines of work are mainly focused on the refinement of the algorithm to make it work properly in front of all the particularities that a real-world environment presents. We would also like to make more use in the final algorithm of the theoretical properties presented about the ellipses in the Hough domain. In parallel with this, and in the context of the i3media project, effort will be devoted to implement the algorithm efficiently to achieve a performance close to real time.
Appendix A

Grass Mask Refinement Using Morphological Filtering

As described in Section 2.1, the draft mask obtained by the color thresholding must be refined. This appendix describes a former approach we followed, based only on morphological filtering. Although the final algorithm did not include this solution, we consider it interesting enough to be included in this dissertation.

Let $m[u, v]$ be the first grass mask draft, depicted in Figure A.1(a). First of all, we clean the undesired little dots that appear on the field lines. They may be seen as local minima of the image (a few 0 pixels among 1s), so we erase them by applying a closing filter. Since we noticed that these minima are always two pixel-width at most, the structuring element we consider is a three pixel line $se_1 = [1, 1, 1]$. The first refinement is then:

$$m[u, v] = \text{close}(m[u, v], se_1)$$

The result of this filtering is shown in Figure A.1(b). Although the global shape of the mask has almost not changed, we have successfully erased the small undesired pixels that, although not noticeable, would have caused problems in the further steps, as we will motivate later.

Next, we want to mask the undesired parts of the spectators. Since they are local maxima, the process we apply is an opening filter. This time, the areas we want to remove are bigger than in the previous step. In addition, we want to remove them completely, because they are a clear source of noise. Our first attempt is, therefore, to use a square structuring element of size 21 (after a trial-and-error process to manually adjust its size). Formally,
being $se_2$ a structuring element consisting of a $21 \times 21$ square with its origin at the center, the second morphological filtering process, which refines the current mask, is described as:

$$m[u, v] = \text{open}(m[u, v], se_2)$$

The mask after this opening is depicted in Figure A.1(c). As we may observe, we have correctly removed all the unwanted remainders of the spectators area. The players mask, however, has been considerably reduced.

The last step is, therefore, to reconstruct the mask of the players, in order to erase them as precisely as possible, by means of an erosion filter. This step justifies the former deletion of the small pixel dots, since after an erosion, those small dots would have increased noticeably. The selection of the structuring element in this case is more complicated. Our first approach was to use the same as in the previous opening process. By trying different kind of views, however, we noticed that this configuration was not correct. The problem appeared in the view where a few area of spectators (less than 20 pixels) appeared in the upper area of the image. By eroding the mask, this area of spectators was uncovered. Although it was a minor problem in terms of area, this mistake could easily mislead the further steps of the algorithm.

In order to overcome this situation, we cut the half upper part of the structuring element. In other words, we used a rectangle of $11 \times 21$ with the origin at the top of it, i.e.:

$$se_3 = \begin{bmatrix}
1 & \cdots & 1 & \cdots & 1 \\
\vdots & \ddots & \vdots & \ddots & \vdots \\
1 & \cdots & 1 & \cdots & 1
\end{bmatrix}$$

This way, the effect of the filtering in the global shape is very similar to the one we wanted, and we avoid the removal of the small pieces of spectators at the top of the image.

Formally:

$$m[u, v] = \text{erode}(m[u, v], se_3)$$

The mask after this erosion is shown in Figure A.1(d), which is the final output of the simplification step.
Figure A.1: Morphological refinement of the grass mask: (a) draft mask, (b) mask after closing by $se_1$, (c) mask after opening by $se_2$, and (d) mask after eroding by $se_3$. 
Appendix B

Pinhole Camera Projection Equations

Recalling all the theoretical frames of reference defined in Chapter 5, this appendix is devoted to deriving the equations that relate the components in the different frames of references. Using these relations we first will compute the equations from the image plane to the field plane in the 3D world, and after that we will derive the inverse equations, from a 3D-world point to the image plane.

The former relation will be deduced in $\mathbb{R}^3$, since it allows us better understanding the process. The later one, however, for simplicity and convenience, will be presented in $\mathbb{P}^3$.

B.1 From image plane to the field plane

In this first section we are interested in the expression of the projection that relates the components of a point $p'$ in the pixel image frame of reference $[o', [u, v]]$, which we denote as $[u_p', v_p']$, with the components of its projection $p$ to the field plane in the 3D world, in the frame of reference $[O, (x, y, z)]$, i.e., $(x_p, y_p, 0)$. Recall Figures 5.1 and 5.3 for clarification.

Formally, in our application, we want to obtain a function $\pi$ as follows:

$$\pi : \mathbb{R}^2 \rightarrow \mathbb{R}^2$$

$$[u_p', v_p'] \rightarrow \pi([u_p', v_p']) = (x_p, y_p)$$

where the $z$ component is assumed to be 0, since we consider only points on the field plane. Note that, if this assumption was not made, the inverse projection from the image plane to the 3D world would not be defined, since each point in the image plane is the projection of a whole line in the 3D world.
B.1 From image plane to the field plane

Following, we derive the expression of $\pi$ step by step.

Recalling the expressions of Equation 5.1, we have that the coordinates of the pixels $[u_p', v_p']$ in the frame of reference $[o', (u', v')]$ are:

$$u_p' = \left( u_p' - \frac{1}{2} \right) \frac{D_w}{r_U}$$
$$v_p' = \left( v_p' - \frac{1}{2} \right) \frac{D_h}{r_V}$$

Next, the equations that relate the image plane coordinates with those of the camera frame of reference are, as shown in Figure 5.3:

$$x_p' = -\frac{r_U}{2} + u_p' = -\frac{D_w}{2} + \left( u_p' - \frac{1}{2} \right) \frac{D_w}{r_U}$$
$$y_p' = F$$
$$z_p' = \frac{r_V}{2} - v_p' = \frac{D_h}{2} - \left( v_p' - \frac{1}{2} \right) \frac{D_h}{r_V}$$ (B.1)

The next step is to relate the coordinates in the camera field of reference with those in the field frame of reference. As introduced in Section 5.1, the change of variable consists of two rotations: pan and tilt, and a translation. Note that this model of rotation is consistent with the motion model of a camera on a tripod. Let us describe these changes of variables.

First, and recalling the relations in Figure 5.2(a), the pan angle $\alpha$ can be obtained as:

$$h = \sqrt{(y_f + C_y)^2 + (x_f - C_x)^2}$$
$$\sin \alpha = \frac{x_f - C_x}{h} = \frac{x_f - C_x}{\sqrt{(y_f + C_y)^2 + (x_f - C_x)^2}}$$
$$\cos \alpha = \frac{y_f + C_y}{h} = \frac{y_f + C_y}{\sqrt{(y_f + C_y)^2 + (x_f - C_x)^2}}$$

Then, the change of variable is represented by the following matrix:

$$A = \begin{bmatrix}
\cos \alpha & \sin \alpha & 0 \\
\frac{y_f+C_y}{h} & \frac{x_f-C_x}{h} & 0 \\
\frac{y_f+C_y}{h} & \frac{x_f-C_x}{h} & 0 \\
0 & 0 & 1
\end{bmatrix}$$
Second, as depicted in Figure 5.2(b), the tilt angle $\beta$ can be obtained as:

\[
\begin{align*}
\sin \beta &= \frac{C_z}{d} = \frac{C_z}{\sqrt{C_z^2 + (y_f + C_y)^2 + (x_f - C_x)^2}} \\
\cos \beta &= \frac{h}{d} = \frac{\sqrt{(y_f + C_y)^2 + (x_f - C_x)^2}}{\sqrt{C_z^2 + (y_f + C_y)^2 + (x_f - C_x)^2}}
\end{align*}
\]

And the matrix representing the rotation is:

\[
B = \begin{bmatrix}
1 & 0 & 0 \\
0 & \cos \beta & \sin \beta \\
0 & -\sin \beta & \cos \beta
\end{bmatrix} = \begin{bmatrix}
1 & 0 & 0 \\
0 & \frac{h}{d} & \frac{C_z}{d} \\
0 & -\frac{C_z}{d} & \frac{h}{d}
\end{bmatrix}
\]

The last change of variable is the translation between $O$ and $C$.

Therefore, the global change of variable gathering the three steps presented is as follows:

\[
\begin{bmatrix}
x_p' \\
y_p' \\
z_p'
\end{bmatrix} = A \cdot B \cdot \begin{bmatrix}
x_p' \\
y_p' \\
z_p'
\end{bmatrix} + \begin{bmatrix}
C_x \\
C_y \\
C_z
\end{bmatrix}
\]

(B.2)

Once we have the expression of $p'$ in terms of the frame of reference of the field, let us derive the expression of the projection from $p'$ to $p$. The line of projection is defined between $C$ and $p'$ (recall Figure 5.3), whose equation (in the field frame of reference) is:

\[(C_x, -C_y, C_z) + \lambda(x_{p'} - C_x, y_{p'} + C_y, z_{p'} - C_z)\]

And now the projected point $f$ is obtained by imposing $z = 0$ in the line, i.e.:

\[C_z + \lambda(z_{p'} - C_z) = 0 \Rightarrow \lambda = -\frac{C_z}{z_{p'} - C_z}\]

Note that we assume that the camera is not pointed horizontally ($z_{p'} - C_z \neq 0$), since in our framework the camera is not on the field plane.

The expressions for the other two components are obtained by substituting the value of $\lambda$ at the equation of the line:

\[
\begin{align*}
x_p &= C_x - \frac{C_z}{z_{p'} - C_z}(x_{p'} - C_x) \\
y_p &= -C_y - \frac{C_z}{z_{p'} - C_z}(y_{p'} + C_y)
\end{align*}
\]

(B.3)
The following scheme depicts the global expression of $\pi$, where each step is linked to the equation used:

$$
\begin{pmatrix}
u_p' \\
v_p'
\end{pmatrix} \overset{(B.1)}{\rightarrow} \begin{pmatrix}x'_p \\
y'_p \\
z'_p
\end{pmatrix} \overset{(B.2)}{\rightarrow} \begin{pmatrix}x_p' \\
y'_p \\
z'_p
\end{pmatrix} \overset{(B.3)}{\rightarrow} \begin{pmatrix}x_p \\
y_p
\end{pmatrix}
$$

## B.2 From 3D world to image plane

This section is devoted to deriving the projection from a 3D-world point to the image plane. In contrast to the projection of the previous section, in this case the expressions are defined for any point in the 3D world (strictly speaking, except the focal point itself).

In this case, the expressions will be presented in matrix form in the projective space, as motivated in Section 5.2. Refer to the same section to recall the basis of the projective space. Formally, we want to obtain the matrix $P \in \mathbb{R}^{3 \times 4}$ such that for any point $(x, y, z)$ in the 3D-world coordinates gives us the projected point $(u, v)$ in the image plane as follows:

$$
[\mu_1, \mu_2, \mu_3]^t = P [x, y, z, 1]^t \quad (u, v) = \begin{pmatrix} \mu_1 \\ \mu_2 \\ \mu_3 \\ \mu_3 \end{pmatrix} \quad (B.4)
$$

Inverting the deduction in the previous section and skipping the derivation, being $k_U = r_U / D_w$ and $k_V = r_V / D_h$, we have that:

$$
P = CT \begin{bmatrix} R' & -R't \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

where:

$$
C = \begin{bmatrix} F k_U & 0 & r_U / 2 \\ 0 & -F k_V & r_V / 2 \\ 0 & 0 & 1 \end{bmatrix} \quad T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \quad t = \begin{bmatrix} C_x \\ -C_y \\ C_z \end{bmatrix}
$$

$$
R = AB = \begin{bmatrix} \cos \alpha & \sin \alpha \cos \beta & \sin \alpha \sin \beta \\ -\sin \alpha & \cos \alpha \cos \beta & \cos \alpha \sin \beta \\ 0 & -\sin \beta & \cos \beta \end{bmatrix}
$$

Note that the relation in Equation B.4 can be generalized to any point of the projective space:

$$
[\mu_1, \mu_2, \mu_3]^t = P [\xi_1, \xi_2, \xi_3, \xi_4]^t \quad (B.5)
$$
Bibliography


