Camera Parameters Extraction Using Real-Time Video Analysis In Football Broadcasting

Miguel Ramirez Pereira Duarte
miguel.r.duarte@ist.utl.pt
Instituto Superior Técnico, Lisboa, Portugal

***Abstract*—This work aimed at developing a software prototype capable of extracting the camera's parameters using real-time video analysis in football broadcasting. The developed algorithm is divided in two parts, the initialization and the tracking. The initialization is used to find a homography that best projects the field model onto the image when there is no prior information about the camera's parameters. The tracking part starts when the camera's parameters for the previous frame are known. Since the camera's position is fixed, an initial process was created that receives several images captured by a specific camera and computes its position, which reduces the number of variable parameters making the tracking process faster and more reliable. The different parts of the algorithm were then tested, concluding that in most cases the parameters were correctly extracted in real-time.**

Keywords: Camera's parameters, Augmented reality, Sports analysis, Real-time, Video analysis

***Index Terms*—Camera's parameters, Augmented reality, Sports analysis, Real-time, Video analysis**

I. Introduction

OVER the last few years, the use of augmented reality (AR) graphics in sports broadcasts has been increasing. In football matches, the most common types of these AR graphics are the visualization of the offside line, the ball's velocity in a shot to goal, the circle with radius of 9,15 meters around the ball in a free-kick and the respective distance to goal. One example of AR in a football broadcast can be seen in Fig. 1.

Fig. 1. Broadcast augmented reality example. Image provided by wTVision.

To achieve the illusion that these graphics are on the real world scene that is being filmed, the camera's parameters must be extracted in real time, which are then sent to a graphics engine that is responsible for rendering the necessary graphics. Here, a method will be presented that is able to perform the first task, i.e, to extract the camera's parameters in real-time, analysing the video feed of a football match broadcast.

Camera calibration is a computer vision field that is already extensively studied. The camera pose determination problem, for cameras with constant focal length, was firstly formulated by Fischler and Bolles [4] in 1981. Many different methods were developed to solve this problem, normally by recognizing features of some known object.

In [5], [8] some other algorithms were developed specifically for football matches, that used a more complete PTZ[1] camera models. The methods presented in those articles were later improved by D. Farin in [3], [2]. These last articles present a more generic calibration algorithm that can be used in any sport, as long as the court has a sufficient number of straight lines. Its structure is divided in two parts: the initialization and the tracking. The first is used when no information about the camera's parameters is known, and the second is used in the opposite situation. The layout of the court lines must be known in order to create a model that must be integrated in the algorithm.

G.A. Thomas described in [7] a more complete method than the one presented by D. Farin, since it already uses the information from the centre circle and fixes the camera's position for all images. The camera's position is computed before any other process by analysing several images and solving an optimization problem. This allows for a reduction of the number of variables in the tracking phase and it also increases the robustness of the algorithm. The results obtained in [7] show that by having a fixed position, a smother camera's rotation is achieved.

II. Background

The pinhole camera model was the model chosen for the development of this work, because it is very simple and it has all the features needed. An illustration of the camera frame in this mathematical model can be seen in the Fig. 2.

In Fig. 2 the camera's coordinate system $\boldsymbol{C}$ is represented by the vectors $\{\boldsymbol{x_C}, \boldsymbol{y_C}, \boldsymbol{z_C}\}$, from which the origin $\boldsymbol{O} = (0,0,0)^T$ represents the camera's centre, the pinhole. By f is represented the focal length (also known as focal distance), which is the distance between the camera centre and the focal plane. The point $\boldsymbol{x}$ is the projection of point $\boldsymbol{X}$ in the image plane.

[1]PTZ (Pan-Tilt-Zoom) cameras are rotating and zooming cameras.

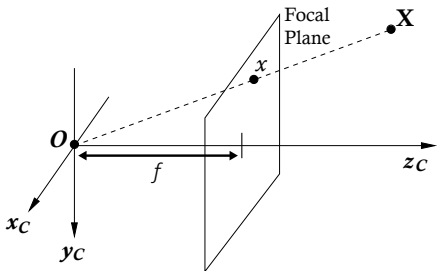


Fig. 2. Pinhole Camera Frame $\boldsymbol{C}$.

Images coordinates are measured in pixels, normally with the origin in the left upper corner. The focal plane in the pinhole camera model is embedded in $\mathbb{R}^3$, so a mapping that translates the points in the image plane into pixels must be found. This transformation is represented by an upper triangular 3×3 matrix $\boldsymbol{K}$, which contains the camera's intrinsic parameters.

For the development of this work, since in football broadcasts only high-end cameras are used, the skew intrinsic parameter can be considered zero. The intrinsic matrix that will be used in this work is

$$\boldsymbol{K} = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix}, \tag{1}$$

where f is the focal length and (c_x, c_y) represents the camera's optical centre, in pixels. Assuming that the optical centre is the image centre, the camera's intrinsic matrix will have only one unknown, the focal length, which is responsible for re-scaling the image coordinates into pixels.

III. Implementation

The main structure of the method that was developed during this work is very similar to the ones presented in [3], [2], [7]. The algorithm is divided in two parts: the initialization and the tracking.

A. Initialization

The initialization is used when there is no prior information about the camera's parameters. The goal of this stage is to find the homography that best projects the field model onto the image. A simplified flowchart of the method that was implemented is represented in Fig. 3.

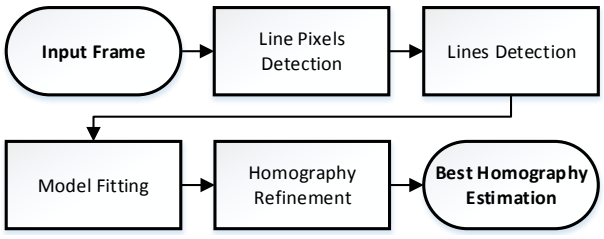


Fig. 3. Simplified initialization flowchart.

1) Line Pixels Detection

The line pixels detection presented here was based on the method in [3]. The images must be converted to the YC_BC_R colour space to perform this part of the algorithm. The luminance value of a pixel located in (x, y) will be represented by $l(x, y)$. The line pixels detector outputs a binary image $\boldsymbol{L}$, which has $\boldsymbol{L}(x, y) = 1$ if the (x, y) pixel is considered to belong to a line and $\boldsymbol{L}(x, y) = 0$ otherwise.

Since the field lines are always white, a very simple way to detect the line pixels is by detecting the white pixels in the image. A robust line pixels detection can not be performed using a simple white pixel detector, it must have other additional constraints in order to minimize the number of false detections. These false detections occur given the fact that not all white pixels belong to field lines. There are other white elements present in the images that can belong, for instance, to the players outfit, the stadium advertisements or to the audience in the stands.

The field lines have intrinsic characteristics that are shown in Fig. 4. Assuming that a line is not wider than δ (in pixels), new constraints are imposed that improve the performance of the line pixels detection. This new line pixels detector can be represented by

$$\begin{aligned} &\boldsymbol{L}(x, y) = \\ &= \begin{cases} 1, & \text{if } f_1(x,y) \quad \wedge \quad f_2(x,y) \quad \wedge \quad f_3(x,y) \\ 1, & \text{if } f_1(x,y) \quad \wedge \quad f_4(x,y) \quad \wedge \quad f_5(x,y) \\ 0, & \text{otherwise,} \end{cases} \end{aligned} \tag{2}$$

where

$$\begin{aligned} f_1(x,y) &= l(x,y) \geq \sigma_l \\ f_2(x,y) &= l(x,y) - l(x-\delta, y) \geq \sigma_d \\ f_3(x,y) &= l(x,y) - l(x+\delta, y) \geq \sigma_d \\ f_4(x,y) &= l(x,y) - l(x, y-\delta) \geq \sigma_d \\ f_5(x,y) &= l(x,y) - l(x, y+\delta) \geq \sigma_d. \end{aligned} \tag{3}$$

The threshold σ_l is the minimum luminance for a pixel to be considered white and σ_d is the minimum luminance difference between a bright pixel (possibly a line pixel) and a darker pixel (a grass pixel), for a pixel to be considered to be part of a line. The first line of Eq.(2) is true if a given pixel in (x, y) is considered to be white and has darker pixels at a horizontal distance of δ pixels, i.e. detects pixels that possibly belong to a vertical line. The second line of Eq.(2) performs the analogous test but for detecting horizontal line pixels.

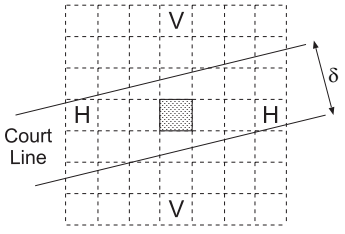


Fig. 4. Example of a horizontal line configuration (adapted from [3]).

Despite the extra restrictions, a lot of false detection occurred in the stadium stands, because of the high level of texture presented in them. To solve this, an extra filter was

developed that removed all the detected pixels outside the field. This mask will be denoted as field area mask.

The field area mask is computed by first creating a histogram of the hue values. The global maximum H_{MAX} within the green interval, between 0 and 120, is then computed. If there are local maxima within the green interval that have a substantial bin value, at least 20% of the bin value of H_{MAX}, and which the hue value does not vary more than 30 from H_{MAX}, they also must be considered to compute the threshold limits of the hue filter. The *i-th* local maxima that fulfil these conditions will be denoted as H^i_{max}. This ensures that even if a field has more than one shade of green (very common in day light games) the field area mask is still correctly computed. The threshold values of the hue filter are computed as follows:

$$H_{low} = \min_i \left(H^i_{max}\right) - \sigma_H, \tag{4}$$

$$H_{high} = \max_i \left(H^i_{max}\right) + \sigma_H. \tag{5}$$

The value σ_H represents the margin of the hue filter. The value of σ_H was set to $\sigma_H = 10$. The filter output is a binary image, in which the value in (x, y) is computed according to

$$\begin{cases} 1, & \text{if } H_{low} \leq h(x,y) \leq H_{high} \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

A closing morphological operation with an elliptic structuring element and a median filter are then applied to the obtained mask. Afterwords the largest blob is found, which corresponds to the final field area mask. These last operations make sure that all the stands pixels have value 0, whereas all the field area pixels have value 1. With the field area mask computed, the final candidate line pixels can be obtained by applying the mask to the previously detected pixels.

2) Line Detection

After the computation of candidate line pixels is concluded, the field lines must be detected. To accomplish this, a standard Hough transform [1] is used. After that, the lines must be classified as vertical or horizontal and sorted by left to right or top to bottom, depending on their orientation.

In this work, only the images from the three main cameras in a football broadcast will be taken into consideration. These are the main camera and the left and right 16,5-meters cameras. These cameras are all located above the field plane, generally in the first level stand. The main camera is roughly aligned with the halfway field line and the 16,5-meters cameras, the ones typically used in an offside replay, are roughly aligned with the penalty area's lines from both sides (also called the 16,5-meters lines). The three main cameras layout can be visualized in Fig. 5, as well as the chosen model frame.

The Hough transform is a great method to detect lines given a set of line points. It is relatively unaffected by line gaps and by outliers, which makes it robust. The parameter space for the lines is (ρ, θ), where ρ is the shortest distance from the origin to line and θ is the angle between the horizontal axis and the line normal. This parametrization allows the use of lines with infinite and zero slope, *i.e.* vertical and horizontal lines, which is very important in this case.

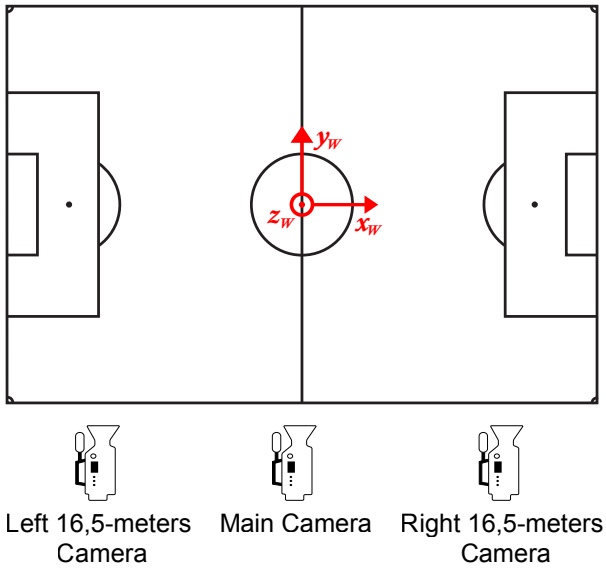


Fig. 5. General cameras' locations in a football broadcast.

To avoid making ambiguous matches in the model fitting step, it is necessary to classify the lines as vertical or horizontal and sort them from left to right or top to bottom, depending on their position and orientation. The classification of the lines is achieved following the data provided in Table I, which was gathered through the analysis of several images from different cameras, pointed at various areas of the field.

TABLE I
VERTICAL OR HORIZONTAL LINE CLASSIFICATION FOR DIFFERENT CAMERAS, BASED ON THEIR θ VALUES.

	Vertical $\theta[^\circ]$	**Horizontal** $\theta[^\circ]$
Left cam.	$]-85, 70[$	$[-90, -85] \cup [70, 89]$
Main cam.	$]-73, 73[$	$[-90, -73] \cup [73, 89]$
Right cam.	$]-70, 85[$	$[-90, -70] \cup [85, 89]$

After classifying the lines, they must be sorted. Just like presented in [3], to sort the vertical lines from left to right, the distance between each line and the point in the middle of the left border is measured and the lines are sorted accordingly. For the horizontal lines a similar procedure takes place, but measuring the distance to the middle of the top border instead.

Very often, the Hough transform detects multiple lines that in fact represent just one. These lines must be eliminated to avoid making the same matches more than once, which would increase a lot the computational time of the model fitting step.

To detect duplicate lines, a search must be performed to look for two lines between which the ρ difference is less than $\Delta\rho_{min}$ and the θ difference is less than $\Delta\theta_{min}$. The values chosen for $\Delta\rho_{min}$ and $\Delta\theta_{min}$ were $\Delta\rho_{min} = 10$ pixels and $\Delta\theta_{min} = 8^\circ$, based on empirical tests. When duplicate lines are found one must be eliminated. To choose which line must be eliminated, a line scoring function was created. This function returns the number of candidate line pixels that each line covers. Based on this criterion, the line with the smallest score is eliminated.

As will be explained in Section III-A3, it is necessary to find at least two vertical and two horizontal field lines in order to perform the model fitting phase. In most cases the number of horizontal and vertical lines detected is higher than the minimum. Having too many detected lines will significantly increase the computational time of the model fitting phase. To avoid having too many lines, an extra function was developed that receives the detected vertical and horizontal lines and outputs the n_d vertical and horizontal lines that are more likely to be a part of the field. The criterion to choose which lines must be kept is the same as in the duplicates removal step, i.e., the n_d lines that cover more candidates line pixels are selected.

The smallest the n_d, the faster the model fitting will be, but the more probable it is that a correct line is ignored and a false detection is used instead, resulting in a bad homography estimation. The higher its value is, the more likely the model fitting phase is to return a good homography but the computational time will also increase.

3) Model Fitting

The goal of this phase is to find the homography that best transforms the field model space into the image plane. This is achieved by scoring several homographies obtained from the match of four points of the image with four points of the field model. The chosen points are produced by the intersection of two vertical lines with two horizontal lines. All possible combinations, without repetitions, of line pairs are tested and the homography with the best score is stored.

There are some matches that can be excluded just by analysing the ratio between the horizontal and vertical lines of the matched points or the geometry of the projected field. These tests are much less time-consuming than the homography computation function or the homography scoring function. As some of the matches will result in bad field projections, this will avoid scoring a lot of homographies.

The test performed before the homography computation is the following: Field aspect ratio (#1) - the ratio between the length of the vertical lines and the horizontal lines in the field model must be approximately the same as the ratio between the length of the matched vertical and horizontal lines from the image. If the matched points passed the first test, then the homography is computed and the following tests are applied: Field too small (#2) - the projected field length or width can not be under 50% of the image length or width, respectively; Top line bigger than the bottom line (#3) - the top line of the projected field can not be bigger in length than the bottom line of the projected field.

If the homography for a given match passed all the rejection tests, it must be evaluated. The evaluation function scores how well does the homography projects the field model into the image. Similarly to what D. Farin described in [3], each pixel of the detected lines that is covered by the projected field contributes with +1 to the score and each pixel of the projected line that does not cover a candidate line pixel contributes with -0.2. This ensures that a bad field projection, despite covering more candidate line pixels than a good projection, will score less than a good projection since it will probably have some projected lines that will not cover any line pixels.

4) Homography Refinement

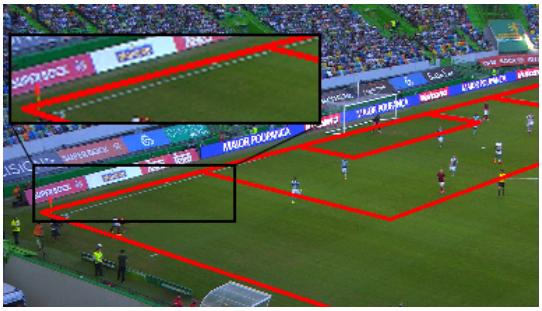
The homography achieved at the end of the model fitting step can sometimes have a slight projection error. To reduce this error it is necessary to find the best possible homography for a given image. To accomplish this task, a last step was added that refines the homography, so that the distance between the line pixels and the projected lines is minimized.

The process used for the homography refinement was based on the method used in [3], [7], which will be explained in more detail in Section III-B4. The only difference between the method applied here and the method described in Section III-B4 is that instead of finding the parameters from which the best homography is achieved, here we look directly for the homography that minimizes a cost function.

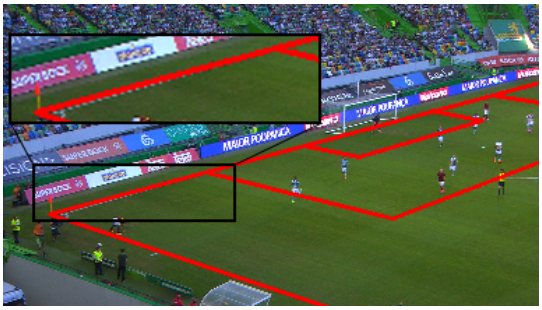
Let us denote by $\boldsymbol{p_i} = (x, y, 1)_i^T$ the detected line pixels and their corresponding closest model line by $\boldsymbol{l_i} = (n_x, n_y - d)_i^T$. The operator that normalizes vectors in homogeneous coordinates such that $\mathcal{L} : (x\ y\ w) \rightarrow (x/w\ y/w\ 1)$ will be denoted by $\mathcal{L}\{\cdot\}$. The cost function for the homography refinement process is similar to the one presented in [3], [7], and can be expressed as

$$E_T = \sum_{i}^{\#points} \left[\boldsymbol{l_i}^T \mathcal{L}\{\boldsymbol{H}^{-1} \boldsymbol{p_i}\} \right]^2. \quad (7)$$

E_T denotes the total squared error of the projected field using the homography $\boldsymbol{H}$. Besides computing the E_T, at the end of the refinement, the average distance of each line pixel to its closest projected field line, in meters, is found. This metric was denoted by E_μ, which will be a useful indicator of the quality of the results. In Fig. 6 an example of a projected field before and after the homography refinement is shown.



(a) Field projection (in red) before homography refinement.



(b) Field projection (in red) after homography refinement.

Fig. 6. Example of the homography refinement improvement.

B. Tracking

The tracking phase starts after a good set of camera parameters for a previous frame have been found. A simplified flowchart of the method that was implemented is represented in Fig. 7.

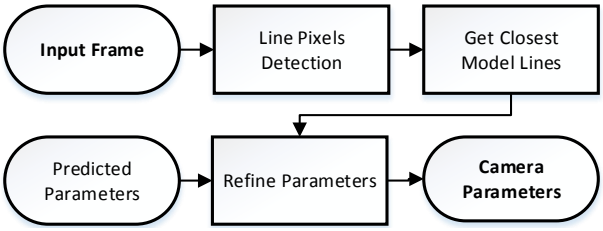


Fig. 7. Simplified tracking flowchart.

1) Camera Parameters Extraction

Obtaining the parameters from the homography is a much more complicated procedure than the inverse process. In the cases where the camera's intrinsic parameters are known (the camera is calibrated), there are linear methods to extract the camera's pose. In the case being studied, the camera's intrinsics parameters are not known and the focal length is not fixed, which increases the difficulty of the task a great deal. Since no detailed procedure to solve this problem was found, a new one was developed.

Let us denote all the camera's variable parameters by $\boldsymbol{\Psi} = (f, \gamma, \beta, \alpha, X_{cam}, Y_{cam}, Z_{cam})$ and the operation that outputs the homography for a given set of parameters $\boldsymbol{\Psi}$ by $\mathcal{H}(\boldsymbol{\Psi})$. In Fig. 8 a representation of the world frame $\boldsymbol{W}$, the camera frame $\boldsymbol{C}$ and the camera parameters can be visualized.

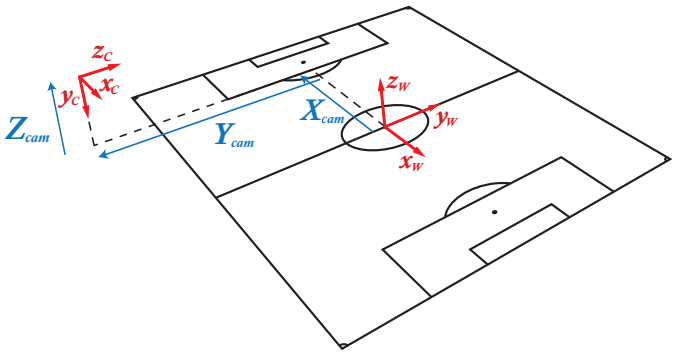


Fig. 8. Representation of the world frame and the camera frame.

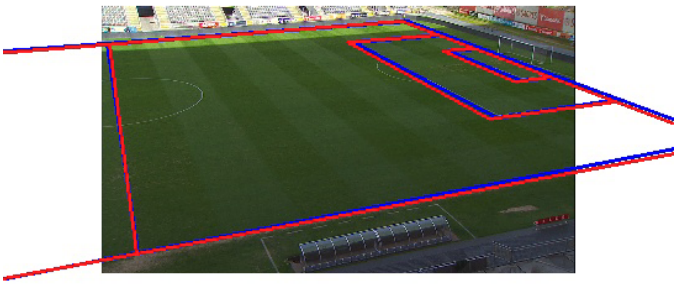
The method can be divided in two steps: the first step outputs a rough estimate of the parameters and the second one refines them. Both steps are minimizations of non-linear functions, computed using the Levenberg-Marquardt minimization algorithm [6].

The goal of the first step is to find the set of parameters $\boldsymbol{\Psi}^*$ that minimizes the entries error between the homography $\boldsymbol{H}$ and the homography obtained by the parameters $\mathcal{H}(\boldsymbol{\Psi})$. This cost function can be represented as

$$E_h = \sum_{i=1}^{3}\sum_{j=1}^{3}\left[h_{ij} - h_{ij}(\boldsymbol{\Psi})\right]^2, \tag{8}$$

where h_{ij} and $h_{ij}(\boldsymbol{\Psi})$ represent the entry in the i-th row and j-th column of $\boldsymbol{H}$ and $\mathcal{H}(\boldsymbol{\Psi})$, respectively. By E_h is represented the total squared error between the homographies entries. The parameters estimate obtained by this method still have some error, as seen in Fig. 9(a). They can be further refined on the second step, using again a non-linear function minimization, with the method described in Section III-B4, using the parameters obtained in the first step as the initial guess. In Fig. 9 the projections of the field using $\mathcal{H}(\boldsymbol{\Psi})$ after the first and second steps are shown.

(a) Frame with the projected field using $\boldsymbol{H}$ (in blue) and using $\mathcal{H}(\boldsymbol{\Psi})$ after the first step (in red).



(b) Frame with the projected field using $\boldsymbol{H}$ (in blue) and using $\mathcal{H}(\boldsymbol{\Psi})$ after the second step (in red).

Fig. 9. Projections of the field using $\mathcal{H}(\boldsymbol{\Psi})$ after the first and second steps of the parameters extraction method.

2) Expected Parameters

The solver described in Section III-B4 requires an initial guess for the parameters that must as close as possible to $\boldsymbol{\Psi}^*$ to achieve the best results. By analysing the parameters differences between the last two frames, the parameters for the current frame t can be predicted by doing

$$\hat{\boldsymbol{\Psi}}_{\boldsymbol{t}} = \boldsymbol{\Psi}_{\boldsymbol{t-1}} + \alpha\Delta\boldsymbol{\Psi}_{\boldsymbol{t-1}}, \tag{9}$$

where

$$\Delta\boldsymbol{\Psi}_{\boldsymbol{t-1}} = \boldsymbol{\Psi}_{\boldsymbol{t-1}} - \boldsymbol{\Psi}_{\boldsymbol{t-2}}, \tag{10}$$

and

$$\alpha \in [0, 1]. \tag{11}$$

When the frame being analysed is the one immediately after the initialization, it is not possible to apply this method since there is no frame $t-2$. In this case the initial guess that will be used in the solver will simply be $\boldsymbol{\Psi}_{\boldsymbol{t-1}}$.

3) Line Pixels Detection

As in the initialization stage, in the tracking phase there is also the need to find the candidate line pixels. The process used to get the line pixels is the same as the one described in Section III-A1. To filter any unwanted pixels, instead of using the field area mask, a similar method to what was used in for evaluate a homography in Section III-A3 was used.

By projecting the field model onto an image using the predicted parameters, a mask can be created that only allows to pass pixels that are covered by the field lines. This type of mask was already denoted as a field lines mask. Using different line widths ω_H, the mask can be more or less strict.

4) Non-linear Function Minimization

To refine the predicted camera parameters it is necessary to have a cost function. This cost function must depend on the camera parameters $\mathbf{\Psi}$, and its minimization should lead to a set of camera parameters that are very close to the real ones. Let us denote by $\mathcal{H}^{-1}(\mathbf{\Psi})$ the homography inverse obtained with the parameters $\mathbf{\Psi}$. The adopted cost function is

$$E_T = \sum_{i}^{\#points} \left[\boldsymbol{l_i}^T \mathcal{L}\{\mathcal{H}^{-1}(\mathbf{\Psi})\boldsymbol{p_i}\} \right]^2 . \quad (12)$$

The cost function returns the value E_T which is the total squared error of the projected field using the parameters $\mathbf{\Psi}$. Changing the parameters $\mathbf{\Psi}$ will result in a different homography, thus the projection error will also change. Here, as in Section III-A4, after refining the parameters, the error E_μ is computed, which is the average distance from each line pixel to its closest line in meters.

Since the function to minimize is non-linear, the algorithm chosen to solve this problem was once again the Levenberg-Marquardt minimization algorithm, which will allow to find the optimal parameters $\mathbf{\Psi}^*$. The solver will receive as inputs the initial guess of the parameters, the detected line pixels and the respective closest model lines.

To find the closest model line for each detected line pixel $\boldsymbol{p_i} = (x, y, 1)_i^T$, the first step is to transform them into the model space $\boldsymbol{p_i\prime} = (x, y, z)_i^T$. Then, the distances between each point $p_i\prime$ and all the model field lines are computed. The line that has the smallest distance to each point is stored along with the point information.

If the distance to the closest of a given point is smaller than d_{min}, the point will be discarded, since is too far from any model line to be part of a line.

In football broadcasts it is very common to have the cameras pointed at the centre field area, which does not have many visible lines, making the tracking less reliable. Using the information provided by the centre circle, the tracking becomes more robust, and less prone to errors. The circle line pixels are detected in the same way as the rest of the line pixels. By checking if the distance to the centre circle is smaller than the distance to any other line, a centre circle pixel can be detected. As the origin of the model space is the origin of the centre circle, which has 9,15 meters of radius, the distance of a pixel projected into the model space $p\prime$ to the circle is simply found by doing (see Fig. 10):

$$| \, 9,15 - || \, p\prime \, || \, | \, . \quad (13)$$

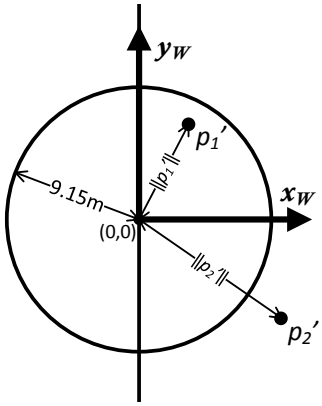


Fig. 10. Representation of the centre circle and two projected points $p_1\prime$ and $p_2\prime$.

To use this circle pixels in the parameters refinement step, in Eq.(12), when a point is closer to the circle, instead of computing the distance between the point and its closest line, Eq.(13) is used.

5) Fixed Camera Position

The majority of the functions used in this step were already developed for other parts of the work, as can be observed in the process's flowchart presented in Fig. 11. The input of the process are several images, preferable six or more, taken with the camera pointed at different field locations with different focal lengths. It is important to have images from both sides of the field to minimize the error.

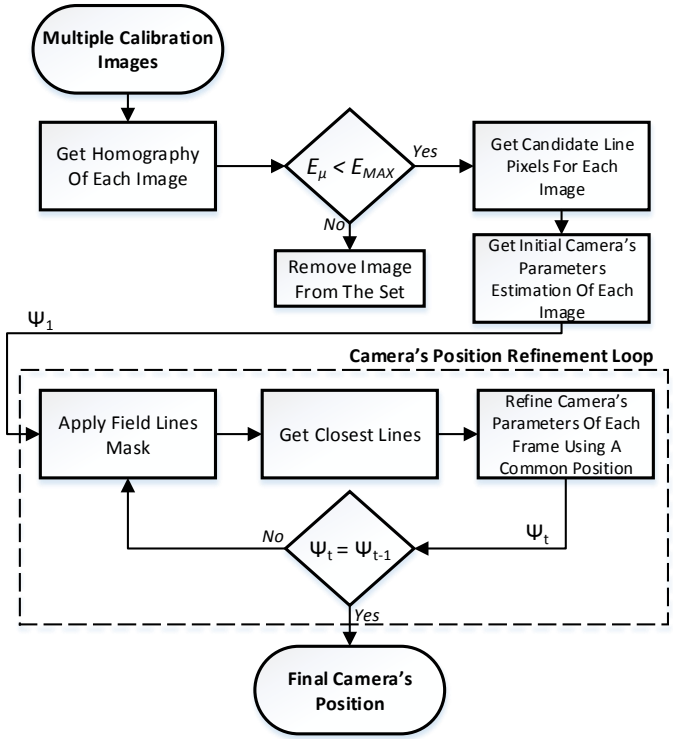


Fig. 11. Camera's position quest flowchart.

The first step of the process is to find the homography that best projects the model to each image, using the initialization method. After the homographies for each image are computed,

it is necessary to find the parameters that correspond to every homography. The final step is to refine the parameters iteratively, with a common position for all images. This is achieved by minimizing the function

$$E_T = \sum_{j}^{\#images} \sum_{i}^{\#points} \left[\boldsymbol{l_i^j}^T \mathcal{L}\{\mathcal{H}^{-1}(\boldsymbol{\Psi^j})\boldsymbol{p_i^j}\} \right]^2, \qquad (14)$$

where p_i^j is the i-th point of the j-th image. The $\boldsymbol{\Psi^j}$ are the j-th image parameters but with a common camera position with all the other images. The initial guess of $\boldsymbol{\Psi^j}$ are the parameters obtained in the previous step, were all the images' parameters were computed individually, with the exception of the camera's position. The initial guess for the common camera's position is the average of all the camera's positions from each image. This process will be iteratively performed until the parameters of all images are the same before and after the refinement.

IV. Final Architecture

Having explained all the different parts required for the field tracking, a more detailed flowchart of the whole algorithm can be seen in Fig. 12.

To visualize if the field is being correctly tracked, in each frame the field is projected using the obtained parameters. In order for the system to be able to automatically detect if the field was lost, i.e., if the tracking is no longer extracting the correct parameters from the analysis of the frames, the error E_μ is compared to a maximum error allowed E_{MAX}. If $E_\mu > E_{MAX}$, the parameters are considered incorrect, thus the initialization process must be used. As the initialization can also return incorrect homographies, the same methodology was applied here to make sure that only correct homographies could go to the tracking phase.

V. Results

To minimize the computational time of every task that has to process an image, the input frames will be scaled down to a predefined size. The minimum frame size from which the lines could still be easily detected it was found to be 384x216.

The developed algorithm was implemented in C++ using the open source computer vision library OpenCV 2.4.10 for most of the image processing tasks. For the non-linear solver, the free version of the ALGLIB library 3.9.0 was used.

The tests were computed on a ASUS G750JX laptop with an Intel(R) Core(TM) i7-4700HQ @2.40GHz CPU and 16GB of DDR3 RAM.

A. Initialization

The quality of the initialization depends on the values of the different parameters for each task. The values that demonstrated to achieve the best results were: $\sigma_l = 70$, $\sigma_d = 14$ and $\delta = 2$ for the detection of line pixels; $\sigma_H = 10$, a 3 pixels window size for the median filter and the elliptical structuring element size of 4x4 pixels for the field area mask; a threshold of 35, a minimum line length of 30 pixels and the maximum line gap of 20 pixels for the Hough transform (using a OpenCV function).

The time spent in each task of the initialization phase was computed using a group of 80 images, from different broadcasts and camera's locations. The gathered data can be observed in Table II.

TABLE II
AVERAGE TIME SPENT IN EACH TASK OF THE INITIALIZATION PHASE OF 80 DIFFERENT IMAGES.

	Average time
line pixels detection	1,97 ms
Field area mask	2,85 ms
Lines detection	1,56 ms
Lines classification	1,77 ms
Extra lines removal	0,47 ms
Field side	2,03 μs
Image's lines intersection	8,77 μs
Rejection test #1	0,94 μs
Homography computation	21,40 μs
Rejection tests #2 and #3	3,55 μ
Homography score	0,53 ms
Homography refinement	12,80 ms

Although the homography computation and score tasks are not very time consuming in comparison with the first four tasks presented on Table II, because they are computed multiple times, they are the ones that influence the most the final computation time. Therefore, in order to achieve better computational times, the number of times that these functions are used must be minimized. The reduction of the parameter n_d and the use of rejection tests are the methods that helped reaching that goal.

By changing the parameter n_d, the minimum time is achieved when setting $n_d = 2$. This would reduce the processing time but also decrease the robustness of the algorithm. Table III presents the average processing times of the initialization using four different values for the n_d parameter. To evaluate the impact on the robustness of the algorithm when using different values of n_d, 80 different images of different fields and camera's locations were initialized, for each n_d value. All the images that were used have the minimum of two vertical and two horizontal lines visible.

TABLE III
AVERAGE PROCESSING TIME AND ROBUSTNESS OF THE INITIALIZATION, USING DIFFERENT VALUES OF n_d.

	Average time	**Correct homog.**
$n_d = 2$	36,44 ms	59
$n_d = 3$	109,37 ms	64
$n_d = 4$	315,23 ms	72
$n_d = 5$	409,14 ms	73

The data in Table III shows that in fact, the higher the

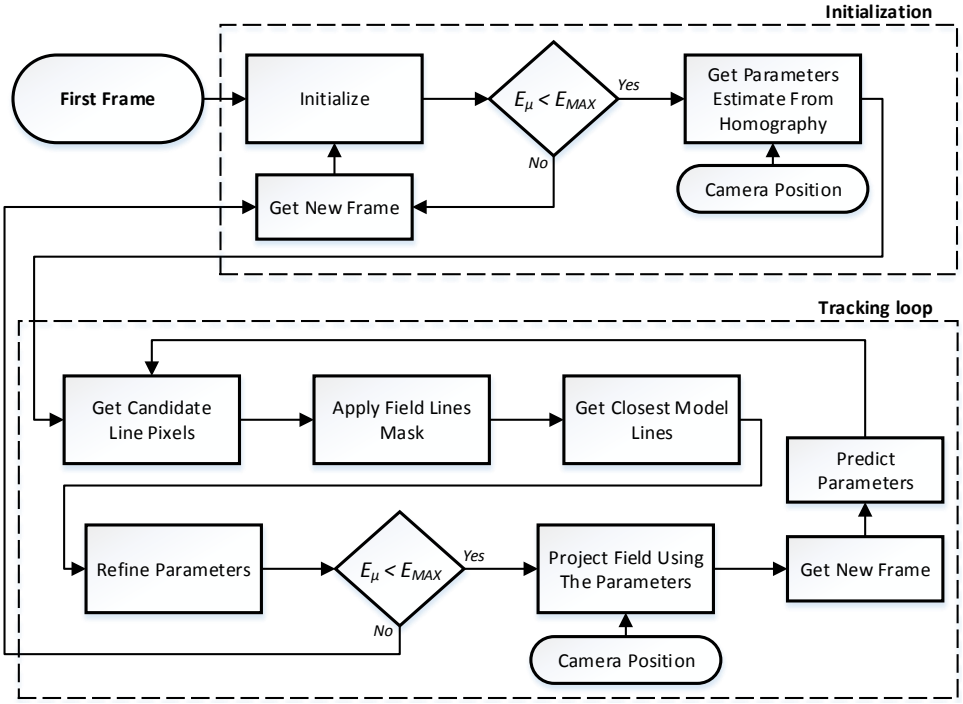


Fig. 12. Final algorithm architecture.

n_d, the longer it will take to process each frame, but the probability of finding a correct homography will also increase. This demonstrates that if the processing time is very limited, n_d should have the minimum value, while if the processing time is not a limitation, a higher value of n_d is recommended.

There are several reasons for the algorithm to not compute the correct homography even with n_d = 5. Three of these reasons are: some lines were not detected because they are too small when compared to other lines; the lines were misclassified (camera pointed at an extreme side of the field); line not detected because its far from the camera and fades when reducing the frame resolution.

Regarding the rejections tests, it was found that on average test #1 rejects 26% of the matches, and jointly applying the three tests 72% of the matches are rejected. These data were also collected from the homography computation for 80 different images, considering the four values of n_d presented before.

These results are quite satisfactory since the use of these simple tests, that are very quick to do (see the times in Table II), allow for a considerable reduction of the total initialization time. The first test is applied before the homography computation and the others are applied before the homography score, resulting in having less 26% of homographies computations and 72% less homographies scores. Since every homography score takes an average of 0,53 ms to compute, the use of these tests is a great method to reduce the total time of initialization.

Overall, the initialization worked quite well when there are two vertical and two horizontal lines visible. The main functionality that is still missing is to be able to initialize using the information provided by the centre circle, which must be developed in future work.

B. Camera's Position Calculation

All the results presented in this section and in Section V-C were obtained using ω_H = 6 pixels, d_{min} = 0.8 meters and E_{MAX} = 0,4 meters. The value chosen for the parameter ω_H is justified by the fact that most of the times the maximum observed difference between the field lines projection, using the predicted parameters, is not higher than six pixels. This way, the number of false detections that are removed is maximized without compromising the real line pixels. The value of d_{min} is small enough so that most of the false detections can be ignored, and high enough to make sure that the correct line pixels are still used in the parameters refinement.

Since this computation will be performed before the tracking starts, there are no limitations regarding the processing time. Therefore, in the extra lines removal phase, it was decided to use $n_d = 5$.

The best results are achieved when using 6 or 8 frames to find the camera's position and the number of images from each side of the field is the same. Fig. 13 shows an example of the camera's position evolution in the refinement loop. In order to evaluate the fluctuations present when searching for the camera's position, five different sets of images were used to calibrate the same camera, which are represented by different colours in the graphs.

The maximum number of iterations of the position refinement was set to 50, to avoid situations where the camera position just keeps oscillating between two similar values perpetually. If the number of images used is under 10, the

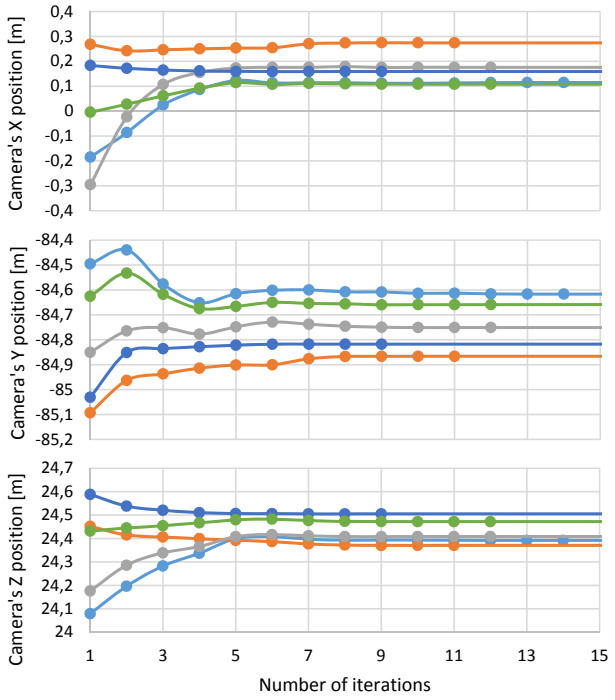


Fig. 13. Camera's position evolution during the position refinement of a main camera.

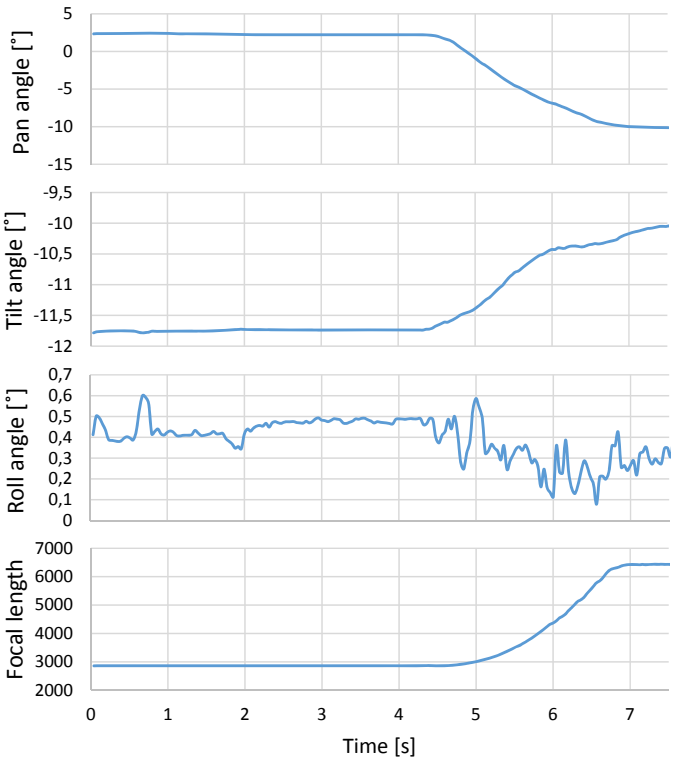


Fig. 14. Camera parameters for a small video sequence of the left camera.

processing time should not be higher than 90 s. Normally it will be around 20 s.

Even though there are some differences in the achieved position for the different sets of images, the tracking results were very similar, tracking the field correctly. The Z component is the one with less fluctuations, with all the different computed values being in an interval of less than 15 cm. The X and Y components of the camera's position both have a fluctuation of around 30 cm.

These are positive results because the fluctuation is small in comparison to the magnitude of the camera's position and especially when taking into account the small image resolution that was used. Also, considering that the camera's position is fixed was just an approximation that was done to improve the robustness and efficiency of the algorithm. Due to the fact that the camera's optical centre does not coincide with the camera's rotation axis, the camera's position will slightly change when panning or tilting. For this reason, it is impossible to develop a system that would output exactly the same position, regardless of the calibration images.

C. Tracking

From the analysis of video frames, the focal length and the pan, tilt and roll angles of the camera were extracted. By analysing the projected field in each frame it was possible to conclude that the parameters were being extracted correctly in most cases. The smoothness of the parameters that is observed in the example presented in Fig. 14, with exception of the roll angle, is another great indicator that the tracking algorithm is performing correctly.

In theory, the roll angle should always be zero. The fact that the camera mount might not be perfectly aligned with the field plane can produce a small rotation around the direction of view (the roll axis), that will also slightly change when panning and tilting the camera. Note that the oscillation observed in the graphs is in the order of decimal of degrees, which in practice has very little to no effect in the end result.

When it is necessary to initialize during the tracking, the initialization is done using n_d = 2, for the reasons previously mentioned. As seen in Table III, with n_d = 2 the initialization takes on average 36,44 ms to be processed, which is equivalent to less than one frame in a 25 fps video feed, or less that two frames for a video with 50 fps.

In Table IV the average processing times of the field tracking tasks are shown. These data were gathered from the analysis of more than 5000 frames. For each frame, an average of 1022 pixels, with a standard deviation of 187 pixels, where used in the refining parameters step.

By comparing the processing times between the initialization and the tracking phase, regarding the operations necessary to achieve the final candidate line pixels, it can be seen that using the field lines mask instead of the field area mask allows for a reduction of 46% in the processing time.

The task that takes the longest time to perform in each frame analysis is the field projection. Even though this task is considered to be part of the tracking algorithm here, in a real system it will be done by a graphics rendering, which receives the camera's parameters provided by the tracking system and has a 3D model of the object that will be projected. Here, the field projection was used to visualise in real-time the results obtained by the frame analysis.

The frame extraction step is also taken into consideration here as a part of the tracking algorithm, which in this case

TABLE IV
AVERAGE PROCESSING TIMES FOR THE DIFFERENT TASKS IN THE FIELD TRACKING STAGE. GATHERED FROM THE ANALYSIS OF 5000 FRAMES.

	Average time
Extract frame	4,33 ms
Resize frame	838,31 μs
line pixels detection	2,32 ms
Get closest model's line	195,32 μs
Refine parameters	3,71 ms
Project field	8,97 ms
Total time	20,36 ms
Total time without extras	7,06 ms

is the time that was necessary to get a new frame from a compressed video file. In a real situation, each frame will be provided directly in a memory location, so this task will not exist.

On average, each frame will be able to be processed in real-time for a 25 fps video feed (40 ms per frame) but it will fall just short of the 50 fps mark (20 ms per frame). If the time spent to project the field and in the frame extraction steps is removed, for the reasons mentioned before, the processing time for each frame drops approximately 65%, to 7,06 ms. This result is very positive, meaning that when applying this method to a real system it will be able to process each frame in real-time, for frame rates up to 120 fps.

VI. CONCLUSIONS

The initialization part was able to successfully find a correct homography, when no information about the camera's parameters are known and when at least two vertical and two horizontal field lines are correctly detected and classified. By changing the parameter n_d, the initialization can become faster but less robust or slower but with higher probability of success, which proved to be very useful.

The tracking method is more robust and faster than the initialization, because there is no need to detect and classify the lines and evaluate different possible homographies. The 7 ms that are necessary, on average, to process each frame during the field tracking allows for this algorithm to be used in real-time for video feeds with frame rates up to 120 fps, which is very positive.

A good indicator of the quality of the extracted parameters is the average distance between a line pixel and the respective closest line, E_μ, which is measured in meters. This error can be used to automatically know when the field has been lost and the parameters are no longer correct, in order to start the initialization process again.

A. Future Work

The main functionality that is missing is being able to initialize when the camera is pointed at the middle field area. To accomplish this, the centre circle must be used. Another problem that must be solved in the initialization phase is that when the cameras are pointed to an extreme side of the field, the lines can be misclassified, leading to bad homographies being computed. The line pixels detection must also be improved, to minimize the number of false detections given by the player's clothing.

A solution that avoids detecting and sorting the lines, and also uses the centre circle information, involves using directly the raw data provided by the Hough transform accumulator matrix. The Hough transform accumulator matrix obtained for a given image would be compared with several sets of preprocessed synthetic accumulator matrices, obtained by simulating the field projection with different possible camera's parameters. The parameters used to generate the preprocessed accumulator matrix that has more similarities with the accumulator obtained for the line pixels, would be the parameters returned from the initialization. A similar method to the one described here was implemented in [7].

In the field tracking phase when the camera is pointed to a area when no field lines are visible, the tracking algorithm is not able to work. These situations must be detected and, when necessary, a backup method is used instead. A feature tracking is one possibility for this backup method. Detecting the same features in consecutive frames would allow to extrapolate the rotation and the difference in the focal length between them.

Most of the developed functions involve the processing of multiple pixels, which is highly parallelized. Computing these tasks in the GPU should lead to better processing times, allowing to process the frames at a higher resolution. This would be specially helpful in situations where the lines are distant from the camera, increasing the probability of their detection.

ACKNOWLEDGEMENTS

The author would like to thank to professor Margarida Silveira from IST and to wTVision's CTO Alex Fraser for all the help and support during the development of this work.

REFERENCES

[1] R. O. Duda and P. E. Hart. Use of the Hough transformation to detect lines and curves in pictures. *Communications of the ACM*, 15(1):11–15, 1972.

[2] D. Farin, J. Han, and P. H. de With. Fast camera calibration for the analysis of sport sequences. In *Multimedia and Expo, 2005. ICME 2005. IEEE International Conference on*, pages 4–pp. IEEE, 2005.

[3] D. Farin, S. Krabbe, W. Effelsberg, et al. Robust camera calibration for sport videos using court models. In *Electronic Imaging 2004*, pages 80–91. International Society for Optics and Photonics, 2003.

[4] M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[5] H. Kim and K. S. Hong. Robust image mosaicing of soccer videos using self-calibration and line tracking. *Pattern Analysis & Applications*, 4(1):9–19, 2001.

[6] J. J. Moré. The Levenberg-Marquardt algorithm: implementation and theory. In *Numerical analysis*, pages 105–116. Springer, 1978.

[7] G. Thomas. Real-time camera pose estimation for augmenting sports scenes. In *IET European Conference on Visual Media Production*, pages 10–19. IET, 2006.

[8] A. Yamada, Y. Shirai, and J. Miura. Tracking players and a ball in video image sequence and estimating camera parameters for 3D interpretation of soccer games. In *Pattern Recognition, 2002. Proceedings. 16th International Conference on*, volume 1, pages 303–306. IEEE, 2002.