Automatic Camera Calibration for Image Sequences of a Football Match

Flávio Szenberg[1], Paulo Cezar Pinto Carvalho[2], Marcelo Gattass[1]

[1]TeCGraf - Computer Science Department, PUC-Rio
Rua Marquês de São Vicente, 255, 22453-900, Rio de Janeiro, RJ, Brazil
{szenberg, gattass}@tecgraf.puc-rio.br

[2]IMPA - Institute of Pure and Applied Mathematics
Estrada Dona Castorina, 110, 22460-320, Rio de Janeiro, RJ, Brazil
pcezar@visgraf.impa.br

Abstract

In the broadcast of sports events one can commonly see adds or logos that are not actually there – instead, they are inserted into the image, with the appropriate perspective representation, by means of specialized computer graphics hardware. Such techniques involve camera calibration and the tracking of objects in the scene. This article introduces an automatic camera calibration algorithm for a smooth sequence of images of a football (soccer) match taken in the penalty area near one of the goals. The algorithm takes special steps for the first scene in the sequence and then uses coherence to efficiently update camera parameters for the remaining images. The algorithm is capable of treating in real-time a sequence of images obtained from a TV broadcast, without requiring any specialized hardware.

Keywords: camera calibration, automatic camera calibration, computer vision, tracking, object recognition, image processing.

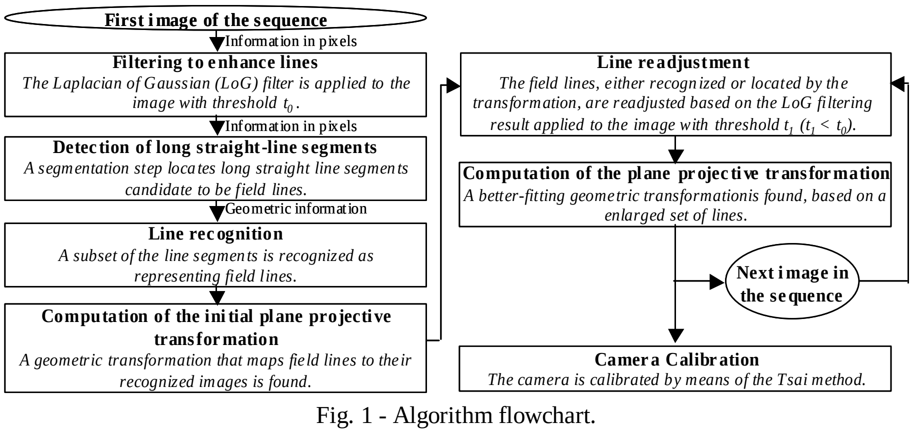
1 Introduction

In the broadcast of sports events one can commonly see adds or logos that are not actually there – instead, they are inserted into the image, with the appropriate perspective representation, by means of specialized computer graphics hardware. Such techniques involve camera calibration and the tracking of objects in the scene.
In the present work, we describe an algorithm that, for a given broadcast image, is capable of calibrating the camera responsible for visualizing it and of tracking the field lines in the following scenes in the sequence. With this algorithm, the programs require minimal user intervention, thus performing what we call *automatic camera calibration*. Furthermore, we seek to develop efficient algorithms that can be used in widely available PC computers.
There are several works on camera calibration, such as [1], [2], [3], and the most

classical one, [4], which introduced the well-known Tsai method for camera calibration. All methods presented in these works require the user to specify reference points – that is, points in the image for which one knows the true position in the real world. In this work, our purpose is to obtain such data automatically.
The method for automatically obtaining reference points is based on object recognition. Some works related to this method include [5], [6], [7] and [8]. [9] discusses some limitations of model-based recognition.
In [10], a camera self-calibration method for video sequences is presented. Some points of interest in the images are tracked and, by using the Kruppa equations, the camera is calibrated. In the present work, we are interested in recognizing and tracking a model, based on the field lines, and only then calibrating the camera.
The scenes we are interested in tracking are those of a football (soccer) match, focused on the penalty area near one of the goals. There is where the events of the match that call more attention take place. In this article we restrict our scope to scene sequences that do not present cuts. That is, the camera's movement is smooth and the image sequence follows a coherent pattern.
The primary objective of our work was to develop an algorithm for a real-time system that works with television broadcast images. A major requirement for the algorithm is its ability to process at least 30 images per second.

2 General View of the Algorithm

The proposed algorithm is based on a sequence of steps, illustrated in the flowchart shown in Fig. 1. The first four steps are performed only for the first image in the scene. They include a segmentation step that supports the extraction of the field lines and the determination of a preliminary plane projective transformation. For the subsequent images of the sequence, the algorithm performs an adjustment of the camera parameters based on the lines obtained in the previous images and on the preliminary transformation.


First image of the sequence
Information in pixels
Filtering to enhance lines
The Laplacian of Gaussian (LoG) filter is applied to the image with threshold t_0.
Information in pixels
Detection of long straight-line segments
A segmentation step locates long straight line segments candidate to be field lines.
Geometric information
Line recognition
A subset of the line segments is recognized as representing field lines.
Computation of the initial plane projective transformation
A geometric transformation that maps field lines to their recognized images is found.
Line readjustment
The field lines, either recognized or located by the transformation, are readjusted based on the LoG filtering result applied to the image with threshold t_1 ($t_1 < t_0$).
Computation of the plane projective transformation
A better-fitting geometric transformationis found, based on a enlarged set of lines.
Next image in the sequence
Camera Calibration
The camera is calibrated by means of the Tsai method.
Fig. 1 - Algorithm flowchart.

3 Filtering

This step of the algorithm is responsible for improving the images obtained from the

television broadcast. Our primary objective here is to enhance the image to support the extraction of straight-line segments corresponding to the markings on the field. To perform this step, we use classical image-processing tools [11]. In the present article, image processing is always done in grayscale. To transform color images into grayscale images, the image luminance is computed.
In order to extract pixels from the image that are candidate to lie on a straight-line segment, we apply the Laplacian filter. Because of noise, problems may arise if this filter is directly applied to the image. To minimize them, the Gaussian filter must first be applied to the image. Thus, the image is filtered by a composition of the above filters, usually called the *Laplacian of Gaussian* (*LoG*) filter.
The pixels in the image resulting from the *LoG* filtering with values greater than a specified threshold (black pixels) are candidate to be on one of the line segments. The remaining pixels (white pixels) are discarded.
Figs. 3, 4, and 5 show an example of this filtering step for the image given in Fig. 2. We will use this image to explain all the steps of the proposed algorithm. In these figures we can notice the improvement due to applying a Gaussian filter prior to the Laplacian filtering. The lines are much more noticeable in Fig. 5 than in Fig. 4.

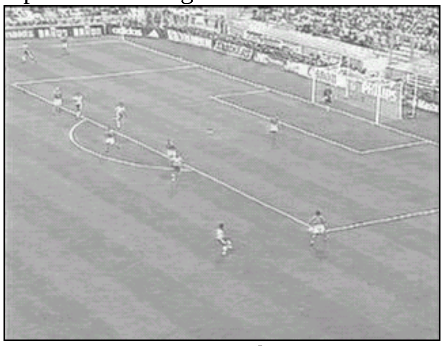


Fig. 2 - Original image.

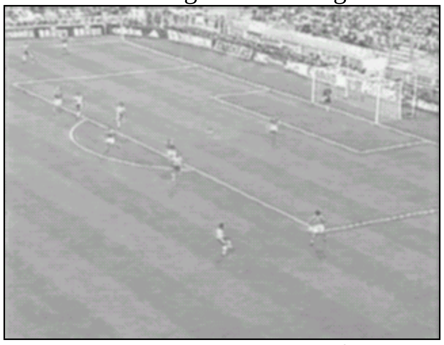


Fig. 3 - Image with Gaussian filter.

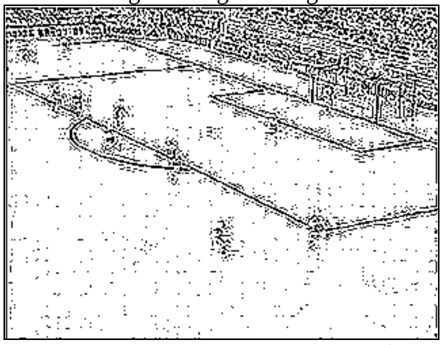


Fig. 4 - Original image with Laplacian filter.

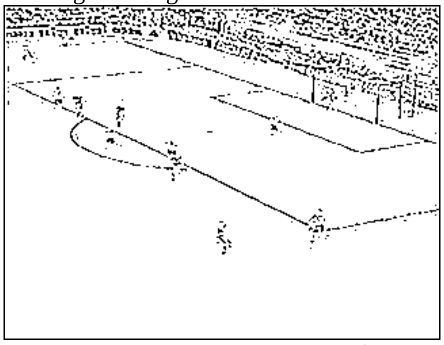


Fig. 5 - Original image with *LoG* filter.

4 Detecting Long Segments

Once the filtering step has roughly determined the pixels corresponding to the line segments, the next step consists in locating such segments. In our specific case, we are interested in long straight-line segments.
Up to now in the algorithm, we only have pixel information. The pixels that have

passed through the *LoG* filtering are black; those that have not are white. The desired result, however, must be a geometric structure – more explicitly, parameters that define the straight-line segments.

The procedure proposed for detecting long segments is divided in two steps:

1. Eliminating pixels that do not lie on line segments:

In this step, the image is divided into cells by a regular grid as shown in Fig. 6. For each of these cells, we compute the covariance matrix

$$\begin{bmatrix} \sum_{i=0}^{n}(x_i - x')^2 & \sum_{i=0}^{n}(x_i - x')(y_i - y') \\ \sum_{i=0}^{n}(x_i - x')(y_i - y') & \sum_{i=0}^{n}(y_i - y')^2 \end{bmatrix}$$

where n is the number of black pixels in the cell, x_i and y_i are the coordinates of each black pixel, and x' and y' are the corresponding averages. By computing the eigenvalues λ_1 and λ_2 of this matrix, we can determine to what extent the black pixels of each cell are positioned as a straight line. If one of such eigenvalues is null or if the ratio between the largest and the smallest eigenvalue is greater than a specified value, then the cell is selected and the eigenvector relative to the largest eigenvalue will provide the predominant direction for black pixel orientation. Otherwise, the cell is discarded. The result can be seen in Fig. 7.

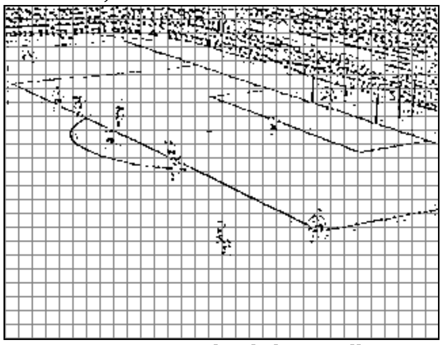


Fig. 6 - Image divided into cells.

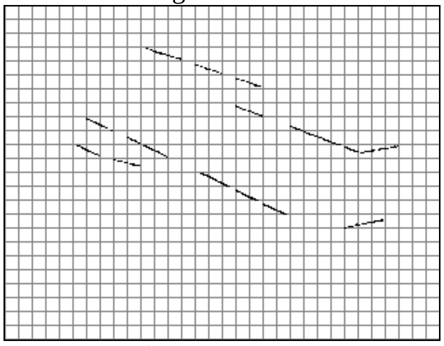


Fig. 7 - Pixels forming straight-line segments.

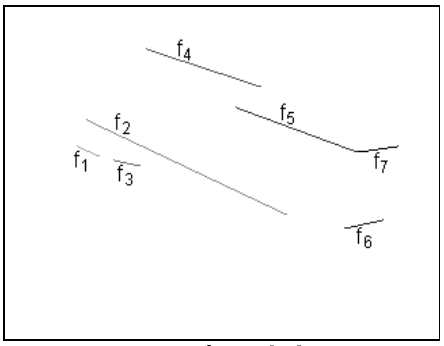


Fig. 8 - Extraction of straight-line segments.

2. Determining line segments:

The described cells are traversed in such a way that columns are processed from left to right and the cells in each column are processed bottom-up. Each cell is given a label, which is a nonnegative integer. If there is no predominant direction in a cell, it is labeled 0. Otherwise the three neighboring cells to the left and the cell below the given cell are checked to verify whether they have a predominant direction similar to the one of the current cell. If any of them does, then the current cell receives its label; otherwise, a new label is used for the current cell. After this numbering scheme is performed, cells with the same label are grouped and, afterwards, groups that correspond to segments that lie on the same line are merged. At the end of the process, each group provides a line segment. The result is illustrated in Fig. 8.

5 Recognizing Field Lines

The information we have now is a collection of line segments, some of which correspond to field lines. Our current goal, then, is to recognize, among those segments, the ones that are indeed projections of a field line, and to identify that line.

For this purpose we use a model-based recognition method [5]. In our case, the model is the set of lines of a football field. This method is based on interpretations, that is, it interprets both data sets – the real model and the input data – and checks whether they are equivalent, according to certain restrictions

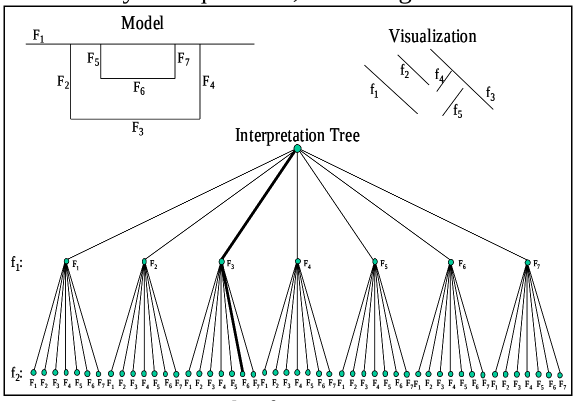


Fig. 9 - Example of an interpretation tree.

We use a tree structure, called *interpretation tree*, in which each leaf represents one possible solution. Fig. 9 illustrates the two first levels of an interpretation tree.

The line segments obtained in the previous step (f_1 through f_5) must be matched to the model given by the field lines F_1 through F_7. Each node of the tree represents a correspondence between a found segment f_x and a model segment F_v. The validity of a node is determined by a set of restrictions, some of which are given below.

1. Two lines cannot have the same representation.
2. Two lines are parallel (or almost parallel, due to the projective transformation) in the visualization only if their representatives in the model are also parallel.
3. All lines must be in the same half-plane determined by the line representing F_1.
4. All lines must be in the same half-plane determined by the line representing F_2, except for the one representing F_1.

An example of an application of these restrictions is the invalidation of the node shown in bold in Fig. 9; in this node, f_1 represents F_1 and f_2 represent F_2. The node is invalid because f_1 and f_2 are parallel, while F_1 and F_2 are orthogonal (thus contradicting restriction 2).

We do not require that all f_x lines correspond to some F_v model line. This leads to the existence of multiple feasible solutions. Therefore, a tie-breaking criterion must be applied: the chosen solution is the one with the largest sum of the lengths of the lines with representations. This, for instance, automatically discards the trivial

solution (where none of the lines has a representation).
For the situation in Fig.9 we have: $f_1 : \emptyset$, $f_2 : F_3$, $f_3 : \emptyset$, $f_4 : F_1$, $f_5 : F_6$, $f_6 : F_4$, $f_7 : F_7$, where $f_x : F_y$ means that line f_x represents line F_y and $f_x : \emptyset$ indicates that f_x represents none of the model lines.

6 Computing the Planar Projective Transformation

In the previous section we discussed how to recognize visible field lines. However, some of the lines may not be clear in the image, and the algorithm may fail to detect them. In this section we shall discuss an algorithm for locating such lines.
We begin by using the pairs $f_x : F_y$ obtained in the previous step to compute a planar projective transformation that projects field lines onto the lines representing them. For that purpose, we find the intersection points of the recognized pairs of lines. This generates a set of points for which the position both in the field and in the image is known. We also compute, in the image, the vanishing points relative to the *ox* and *oy* directions of the model. Then, we use least squares to find a projective transformation – represented by a 3×3 matrix in homogeneous coordinates – that best maps field points to image points.
This initial transformation may not be very good due to errors in the location of the detected lines and because we may have a small number of points, since there may be field lines for which we have not found a representative. The transformation error is illustrated by Fig. 10, in which we can notice a considerable difference between a sideline of the penalty area and its reconstruction.

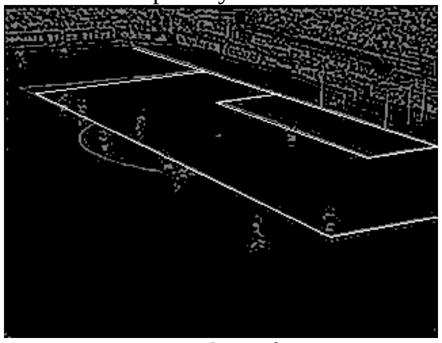


Fig. 10 - Initial transformation.

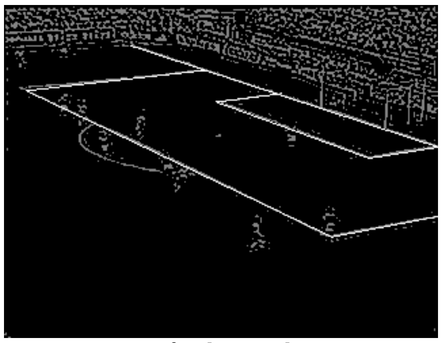


Fig. 11 - After line readjustment.

To improve this solution, we shall use the computed transformation to find the missing field lines. A field line may not have been correctly located by the previous step, especially if the line is faded in the image and most of its pixels are discarded by the *LoG* filter (for instance, the above mentioned sideline of the penalty area was not located in Fig. 8). However, once an approximate position for that line is known, these pixels may be retrieved by a *LoG* filter with a smaller threshold, but subject to be near the predicted position of one of the field lines.
The idea is to partition the pixels of the image resulting from the *LoG* filtering according to the nearest projected field line (computed by the initial projective transformation), discarding those pixels whose distance to all lines is larger than a certain value.
From each group of pixels, we use least squares to compute straight lines that best fit

those pixels. Such lines will replace those that originated the groups. We call this step of the method *line readjustment.*

With this new set of lines, we compute a new planar projective transformation. This new transformation is expected to have better quality than the previous one, because the input points used to obtain the transformation are restricted to small regions in the vicinity of each line. Therefore, the noise introduced by the effect of other elements of the image is reduced.

With this new set of lines, we compute a new planar projective transformation, in the same way as done for the initial transformation. The result of this readjustment is illustrated in Fig. 11. These images are in negative to better highlight the extracted field lines, represented in white. The gray points in the images are the output of the *LoG* filter, and they are in the image only to provide an idea of the precision of the transformation. We can notice that the sideline of the penalty area in the upper part of the images is better located in Fig. 11 than in Fig. 10.

Some problems may arise in this readjustment due to the players. If there is a large number of them over a line – for instance, when there is a wall –, this line may not be well adjusted.

7 Camera Calibration

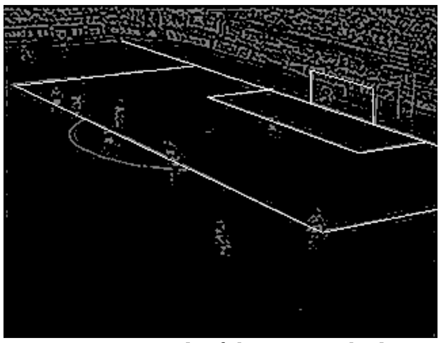


Fig. 12 - Result of the Tsai method.

The projective transformation computed in the previous section is planar, therefore it does not consider points outside the plane of the field, such as the top of the goal posts. In order to obtain a full three-dimensional reconstruction of the field, we must calibrate the camera used to capture the image, that is, we must find its intrinsic and extrinsic parameters. Tsai's algorithm [4] is employed for such purpose. A regular grid is generated inside the penalty area of the model, and pairs of the form (pc_i, pi_i), where pc_i are points of the model and pi_i are their images according to the last computed planar transformation, are passed to the calibration procedure. With this information the position, orientation and zoom factor of the camera can be recovered.

The result of the Tsai method is illustrated in Fig. 12. One can notice that it is now possible to appropriately reconstruct objects outside the field, such as the goal posts.

8 Working with a Sequence of Images

When there is a sequence of images (with no cuts), we are interested in calibrating the camera with the least possible computational effort, in order to obtain real-time processing. Since we are dealing with television images, this means that, for each image, we have 1/30 of a second to calibrate the camera.

For the first image, we apply the camera calibration process described above. In order to optimize the proposed algorithm from the second image on, we shall take advantage of the previous image. We can use its final plane projective transformation as an initial transformation for the current image, and go directly to the line readjusting method.

9 Results

To test the proposed algorithm we have analyzed two sequences of images. The first one is a synthetic sequence obtained from an OpenGL visualization of a football field aims at verifying if model. The second sequence was obtained by capturing real images from a TV broadcast using a video capture board. Each sequence has 27 frames with a resolution of 320x240. Figs. 13 and 14 show the first and the last images in each sequence, showing the reconstructed elements of the field.
In the first sequence of images the results from the algorithm reproduce the original model. The second sequence aims at verifying the behavior of the algorithm when working with real images. These should be harder for the algorithm, since they contain extraneous elements – such as the audience and marketing plates around the field – and feature curved field lines, due to lens distortion.

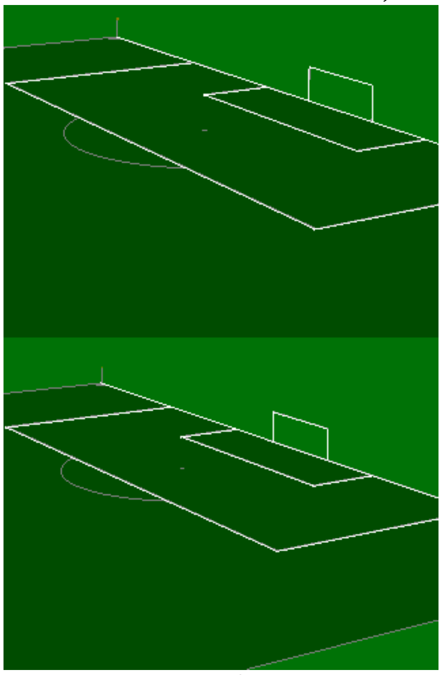


Fig. 13 - Artificial data.

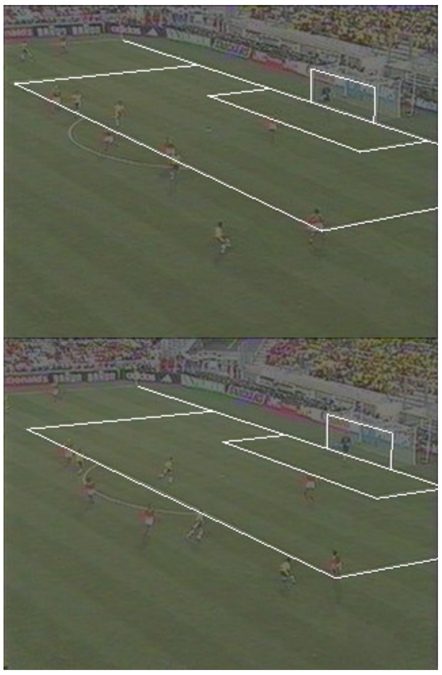


Fig. 14 - Real data.

In the artificial sequence of images, visual errors cannot be noticed. In the real sequence, there are some small errors in the superposition of the reconstructed lines (white) over the image lines.
The numerical errors for the first sequence are shown in Tables 1 and 2. These tables present the correct projected coordinates and those resulting from our algorithm (reconstructed coordinates). The comparison shows that the error, for each point, is never greater than 2 pixels, with the typical average of about 0.5 pixel. These small errors result mainly from the discrete nature of the low-resolution image.

The tests were conducted in a Pentium III 600 MHz. The processing time was 380 milliseconds for the first sequence and 350 milliseconds for the second one. This difference results from the fact that the algorithm detected 10 line segments in the first image of the synthetic sequence and only 7 in the TV sequence, as we can see in Fig. 8 (the side lines and goal posts were not detected in the first TV image). This increase in the number of detected segments influences the processing time of the recognition step, increasing the depth of the interpretation tree. Both processing times are well below the time limit for real-time processing. If the desired frame rate is 30 fps, up to 900 milliseconds could be used for processing 27 frames.

10 Conclusions

The algorithm presented here has generated good results even when applied to noisy images extracted from TV. Our goal to obtain an efficient algorithm that could be used in widely available computers was reached. In the hardware platform where the tests were performed the processing time was well below the time needed for real-time processing. The extra time could be used, for example, to draw ads and logos on the field.

11 Future Works

Although the method presented here is capable of performing camera calibration in real time for an image sequence, the sequence resulting from the insertion of new elements in the scene suffers from some degree of jittery, due to fluctuations in the computed camera position and orientation. We intend to investigate processes for smoothing the sequence of cameras by applying Kalman filtering [12] or other related techniques.
Another interesting work is to develop techniques to track other objects moving on the field, such as the ball and the players. We also plan to investigate an efficient algorithm to draw objects on the field behind the players, thus giving the impression that the objects drawn are at grass level and the players seem to walk over them.

12 Acknowledgments

This work was developed in TeCGraf/PUC-Rio and Visgraf/IMPA, and was partially funded by CNPq. TeCGraf is a Laboratory mainly funded by PETROBRAS/CENPES. We are also grateful to Ralph Costa and Dibio Borges for their valuable suggestions.

References

1. Carvalho PCP, Szenberg F, Gattass M. Image-based Modeling Using a Two-step Camera Calibration Method. In: Proceedings of International Symposium on Computer Graphics, Image Processing and Vision, SIBGRAPI’98, Rio de Janeiro, 1998, pp 388-395
2. Faugeras O. Three-Dimensional Computer Vision: a Geometric ViewPoint. MIT

Press, 1993
3. Jain R, Kasturi R, Schunck BG. Machine Vision. McGraw-Hill, 1995
4. Tsai R. An Efficient and Accurate Camera Calibration Technique for 3D Machine Vision. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, Miami Beach, FL, 1986, pp 364-374
5. Grimson WEL. Object Recognition by Computer: The Role of Geometric Constraints. Massachusetts Institute of Technology, MIT Press, 1990
6. Munkelt O, Zierl C. Fast 3-D Object Recognition using Feature Based Aspect-Trees. Technishe Universität München, Institut für Informatik, Germany
7. Nagao K, Grimson WEL. Object Recognition by Alignment using Invariant Projections of Planar Surfaces. MIT, Artificial Intelligence Laboratory, A.I. Memo no. 1463, February, 1994
8. Pla F. Matching Features Points in Image Sequences through a Region-Based Method. In: Computer Vision and Image Understanding, vol. 66, no. 3, June, 1997, pp 271-285
9. Schweitzer H. Computational Limitations of Model Based Recognition. In: http://www.utdallas.edu/~haim/publications/html/modelnpc.html
10. Zeller C, Faugeras O. Camera Self-Calibration from Video Sequences: the Kruppa Equations Revisited, INRIA Sophia Antipolis, Programme 4, Rapport de Recherche no. 2793, Frévrier, 1996
11. Gonzalez RC, Woods RE. Digital Image Processing. Addison-Wesley Publishing Company, 1992
12. Welch G, Bishop G. An Introduction to the Kalman Filter. In: http://www.cs.unc.edu/~welch/media/ps/kalman.ps

Field's Points			**Correct Coordinates**		**Reconstructed Coordinates**		**Error** (Euclidean Distance)
x	y	z	u	v	u	v	
105.0	68.00	0.00	81.707	216.584	81.731	215.972	0.612
88.5	13.84	0.00	230.117	78.133	228.747	77.525	1.499
88.5	54.16	0.00	1.236	183.463	0.424	183.197	0.854
99.5	24.84	0.00	259.039	134.206	258.566	133.815	0.614
99.5	43.16	0.00	146.690	174.826	146.067	174.484	0.711
105.0	30.34	0.00	269.817	155.102	269.629	154.697	0.446
105.0	30.34	2.44	270.921	181.066	270.215	180.863	0.735
105.0	37.66	2.44	224.101	194.645	223.291	194.407	0.845
105.0	37.66	0.00	223.405	170.271	223.082	169.876	0.510
						Average Error	**0.696**

Tab. 1 - Comparison between the correct and reconstructed coordinates for the first scene.

Field's Points			**Correct Coordinates**		**Reconstructed Coordinates**		**Error** (Euclidean Distance)
x	y	z	u	v	u	v	
105.0	68.00	0.00	97.167	205.940	96.791	205.585	0.517
88.5	13.84	0.00	243.883	66.434	243.549	66.022	0.530
88.5	54.16	0.00	16.101	173.174	15.655	172.623	0.709
99.5	24.84	0.00	273.344	124.029	273.125	123.715	0.382
99.5	43.16	0.00	160.672	164.798	160.366	164.421	0.486
105.0	30.34	0.00	284.160	145.173	283.992	144.914	0.309
105.0	30.34	2.44	285.241	171.290	284.886	171.090	0.407
105.0	37.66	2.44	238.127	184.768	237.744	184.538	0.447
105.0	37.66	0.00	237.462	160.349	237.252	160.063	0.355
						Average Error	**0.452**

Tab. 2- Comparison between the correct and reconstructed coordinates for the last scene.