CTIM
CENTRO DE TECNOLOGÍAS
DE LA IMAGEN

CTIM Technical Report

ISSN 2254-2353

Augmented Reality in Sport Scenarios Using Cameras Mounted on a Tripod

Miguel Alemán Flores, Luis Álvarez León, Pedro Henríquez Castellano and Agustín Trujillo Pino

No. 2

Las Palmas de Gran Canaria
13 September 2012

Augmented Reality in Sport Scenarios Using Cameras Mounted on a Tripod

Miguel Alemán-Flores Luis Alvarez Pedro Henriquez
Agustín Trujillo

26/07/12

Abstract

In this paper we address the problem of inserting virtual content in a video sequence. The method we propose uses just image information. We perform primitive tracking, camera calibration, real and virtual camera synchronisation and finally rendering to insert the virtual content in the real video sequence. To simplify the calibration step we assume that cameras are mounted on a tripod (which is a common situation in practise). The primitive tracking procedure, which uses lines and circles as primitives, is performed by means of a CART (Classification and Regression Tree). Finally, the virtual and real camera synchronisation and rendering is performed using functions of OpenGL (Open Graphic Library). We have applied the method proposed to sport event scenarios, specifically, soccer matches. In order to illustrate its performance, it has been applied to real HD (High Definition) video sequences. The quality of the proposed method is validated by inserting virtual elements in such HD video sequence.

Keywords: Augmented Reality, Graphic Insertion, Camera Calibration

1 Introduction

The broadcasting of sport events increasingly introduces the processing of video sequences for the insertion of virtual objects. On the one hand, these objects can be for a better understanding of the scene such as country flags in swimming, yellow down line in football, offside line in soccer and puck path in hockey. At the other hand, them can be for adding advertisements in different places during the event without disturbing the viewers. Most methods of virtual objects insertion in a video consist of two main stages, which are camera calibration and virtual content insertion. Camera calibration consists of several stages, which usually include initialisation, calibration estimation, primitive tracking and calibration refining. Moreover, virtual content insertion is divided in two steps, camera synchronisation and rendering. In this paper, we analyse this problem in real application scenarios, where we deal with some additional problems, such as the small number of visible primitives which are usually visible and are needed for camera calibration, or the large size of HD videos. The main assumptions we make are, firstly, that the video sequences have been acquired using a camera mounted on a tripod (which is a common situation in practise) and fixed in location it can freely rotate and change their intrinsic parameters by zooming. Secondly, that there exists a certain contrast between all primitives of interest for calibration (lines and circles) and the background (grass). Usually, in sport event scenarios, primitives are well contrasted with respect to the background (the green of the grass in the case of soccer matches).

In our implementation, there are several stages as is shown in Figure 1. Initialisation stage consists of two different process, camera calibration initialisation and virtual objects configuration. Camera calibration initialisation is divided in three steps, which are only carried out on the first frame: load of previously calculated information (geometrical parameters of the tripod and training classes for the decision tree), primitive detection, and camera calibration for the first frame. Primitive detection is performed by means of a morphological method described in [7], whereas the camera calibration technique is explained in this paper. In virtual object configuration we have to define the objects appearance and their positions in the soccer stadium. For the following frames, we directly start at the calibration estimation and we consider the information obtained from the previous two frames. We assume that the changes between consecutive frames are not too large and, furthermore, as we are dealing with cameras placed on a tripod, the movements of the camera are restricted. Afterwards, we continue the process with the primitive tracking stage, which searches for the primitives in the image using a decision tree and the projection of the reference primitives. The latter are the lines and circles in a model of a soccer field with actual dimensions, which are projected using the homography estimated from the previous two frames. Finally, from the primitives detected and the geometry of the tripod, we refine the calibration for the current frame. When we have the camera calibrated in a frame, we can synchronise the real camera with a virtual camera and project the virtual objects onto the real image using OpenGL [2]. We use OpenGL because it provides functions to manage easily virtual camera and virtual objects. For example changing the viewpoint or the virtual camera position, adding textures and blending to the virtual objects, etc.

This paper is structured as follows: In section 2, we summarise the state of the art. Section 3 explains the geometry and calibration process for cameras mounted on a tripod. In section 4, we introduce a method to perform primitive tracking using a decision tree. In section 5, we describe the camera synchronisation. Section 6 is the explanation of the rendering. In section 7, we show some experiments and results. Finally, in section 8, we present our main conclusions.

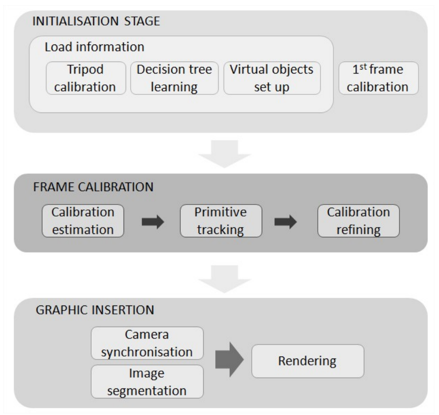

Figure 1: Stages in our implementation for inserting virtual content in a video sequence.

2 Related works

Different approaches have been presented to tackle the problem of insert virtual content or advertisements in sport event videos. To achieve incorporate virtual graphics in a video, we have to calibrate the camera to get the camera parameters and be able to synchronise the real camera with the virtual camera. To perform a video camera calibration, the most common strategy consists of certain tasks which are applied on each video frame: feature extraction (e.g. primitives and background), camera calibration estimation (based on the previous frames), primitive tracking, improvement of camera calibration estimation. See for instance [3, 4, 5, 6, 9]. In our case, we deal with cameras mounted on a tripod, and therefore, we have to take into account the changes in the camera model, as explained in [8, 15]. With the calibration done, we can start the graphic insertion, which can be divided in two steps: camera synchronisation and rendering. Different techniques and libraries are used to perform these stages. For example in [10, 11] they project the virtual content with the projection matrix obtained in the camera calibration step, and paint the projected content pixel by pixel in the real image. To give a more realistic appearance mixing the virtual and real image, a blending technique is used in [13]. On the other hand, graphic libraries are used to improve the procedure efficiency as in [12]. A common issue in all the related works is the image segmentation for detect pixels which can be replaced, e.g. grass, and which not, e.g. players.

3 Geometry and calibration of cameras mounted on a tripod

A tripod is defined by a centre of rotation $\bar{X}_0 = (X_0, Y_0, Z_0)^T$ and two unitary rotation axes $\bar{e}^0 = \left(\bar{e}^0_x, \bar{e}^0_y, \bar{e}^0_z\right)^T$, $\bar{e}^1 = \left(\bar{e}^1_x, \bar{e}^1_y, \bar{e}^1_z\right)^T$. We call $R\left(\bar{e}^k, \theta_k\right)$ the matrix to rotate by an angle of θ_k about axis $\bar{e}^k$. In order to rotate a 3D point $\bar{X}$ about axis $\bar{e}^k$ using the centre of rotation $\bar{X}_0$, the transformation turns into the following equation:

$$\bar{X}\left(\theta_k\right) = \bar{X}_0 + R\left(\bar{e}^k, \theta_k\right)\left(\bar{X} - \bar{X}_0\right) \tag{1}$$

The general motion of a tripod is the composition for two rotations of the above type. We assume that the centre of rotation $\bar{X}_0$ is the same for both axes, which is equivalent to assume that the two axes about which the tripod rotates intersect at a point. This is a common situation and the points are then transformed according to the general equation for the motion of a tripod:

$$\bar{X}\left(\theta_0, \theta_1\right) = \bar{X}_0 + R\left(\bar{e}^0, \theta_0\right) R\left(\bar{e}^1, \theta_1\right)\left(\bar{X} - \bar{X}_0\right) \tag{2}$$

From now on, we use the following notation :

$$R\left(\theta_0, \theta_1\right) \equiv R\left(\bar{e}^0, \theta_0\right) R\left(\bar{e}^1, \theta_1\right) \tag{3}$$

$$\bar{t}\left(\theta_0, \theta_1\right) = \bar{X}_0 - R\left(\theta_0, \theta_1\right)\bar{X}_0 \tag{4}$$

Therefore, Equation 2 can be written in the form:

$$\bar{X}\left(\theta_0, \theta_1\right) = R\left(\theta_0, \theta_1\right)\bar{X} + \bar{t}\left(\theta_0, \theta_1\right) \tag{5}$$

The general equation for the projection of a 3D point $\bar{X} = (X, Y, Z)^T$ onto the image plane is as follows:

$$s\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = A\left(f_0\right) R_0 \left\lfloor Id, -\bar{c}^0 \right\rfloor \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{6}$$

where

$$A\left(f_0\right) = \begin{pmatrix} f_0 & 0 & x_c \\ 0 & rf_0 & y_c \\ 0 & 0 & 1 \end{pmatrix} \tag{7}$$

$$R_0 = \begin{pmatrix} r^0_{00} & r^0_{01} & r^0_{02} \\ r^0_{10} & r^0_{11} & r^0_{12} \\ r^0_{20} & r^0_{21} & r^0_{22} \end{pmatrix} \tag{8}$$

$$\left[Id, -\bar{c}^0\right] = \begin{pmatrix} 1 & 0 & 0 & -\bar{c}_x^0 \\ 0 & 1 & 0 & -\bar{c}_y^0 \\ 0 & 0 & 1 & -\bar{c}_z^0 \end{pmatrix} \tag{9}$$

In Equation 6, we assume that the possible lens distortion has previously been corrected. The matrix $P_0 \equiv A(f_0) R_0 \left\lfloor Id, -\bar{c}^0 \right\rfloor$ is called projection matrix. For each frame, the values of $(f_n, \theta_0^n, \theta_1^n)$ determine the projection matrix as follows:

$$P(f_n, \theta_0^n, \theta_1^n) \equiv A(f_0) R_0 \left[Id, -\bar{c}^0\right] \begin{pmatrix} R(\theta_0^n, \theta_1^n) & \bar{t}(\theta_0^n, \theta_1^n) \\ 0 & 1 \end{pmatrix} \tag{10}$$

Therefore, considering the following expression:

$$P_n(f_n, \theta_0^n, \theta_1^n) \equiv A(f_n) R_0 R(\theta_0^n, \theta_1^n) \left\lfloor Id, R^T(\theta_0^n, \theta_1^n)\left(\bar{t}(\theta_0^n, \theta_1^n) - \bar{c}^0\right)\right\rfloor \tag{11}$$

we can deduce that the rotation and focus of the camera after the motion are:

$$R_n \equiv R_0 R(\theta_0^n, \theta_1^n) \tag{12}$$

$$\bar{c}^n = -R^T(\theta_0^n, \theta_1^n)\left(\bar{t}(\theta_0^n, \theta_1^n) - \bar{c}^0\right) \tag{13}$$

We have to take into account that any view acquired with the tripod can be considered as a reference to move it, and when we change the initial reference camera, we are also modifying the rotation axes of the tripod.

In practise, in order to estimate the geometry of the tripod, we previously calibrate some isolated frames from the video sequence using standard calibration techniques, and then we estimate the geometry of the tripod using a standard bundle adjustment technique.

4 Primitive tracking by means of a decision tree

At the primitive tracking stage, we use a calibration estimation and a CART decision tree. For the calibration estimation at frame n, we use the parameters (f, θ_0, θ_1) from frames $n-1$ and $n-2$, as explained above. A CART decision tree, as those described in [1], is used to detect the white primitives. To build the decision tree is necessary a learning stage based on a training set with information about different classes. For each video sequence, we read a classification data set, which contains information about two classes, primitives and background. Usually, in our soccer field scenarios, primitives are white and the background is green. In the data set, we have RGB values obtained from a manual segmentation of the first frame of the sequence. For the rest of the frames we perform the primitive tracking that is completely explained in [14]. After the tracking stage, we proceed to the improvement of the calibration estimation.

5 Camera synchronisation

We use OpenGL to create a 3D virtual world which will be mixed with the real world image. To be able to insert objects in the real image with the same perspective, we need to synchronise the virtual camera with the real camera. That is mean, we have to place the virtual camera at the same position of real camera and with the same rotation and zoom. The synchronisation is done by calculating virtual camera parameters from real camera parameters. The parameters which define a real camera are rotation, translation and clip plane, as we can see in Figure 2. The clip plane is defined by the focus, the centre and the intrinsic parameters of the camera. To perform the camera synchronisation, we have to configure the virtual camera with the real camera parameters. OpenGL has functions that implements this process, but needs some inputs which we have to calculate. These inputs are: camera centre, projection centre and a vector indicating the camera vertical axis direction (VUP). Moreover, we need to define the viewing volume. The viewing volume determines how a 3D object is projected onto a 2D image. For a perspective projection, the viewing volume is a frustum. Determining the frustum in OpenGL needs distances from projection centre to clipping planes (left,right,top,bottom) and distances from the camera to the near and far depth clipping planes, as shown in Figure 3.

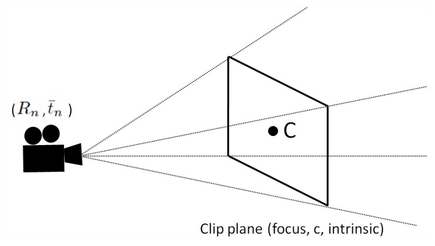

Figure 2: Real camera. Where R_n is rotation, $\bar{t}_n$ is translation and C is image centre, the centre components are x_c and y_c from intrinsic parameters shown in expression 7

We use the euclidean camera calibration performed in the previous stage to obtain all the requested parameters. Firstly, we get the inverse projection matrix from the euclidean camera, being the projection matrix as is showed in 11. The inverse projection matrix is obtained as follows:

$$P^{-1} \equiv R_0^T A^{-1} (f_0) \left\lfloor Id, \bar{c}^0 \right\rfloor \tag{14}$$

Now, we can calculate the principal point multiplying this matrix by the image centre, $C_v = P^{-1}C$, which belongs to the intrinsic parameters in

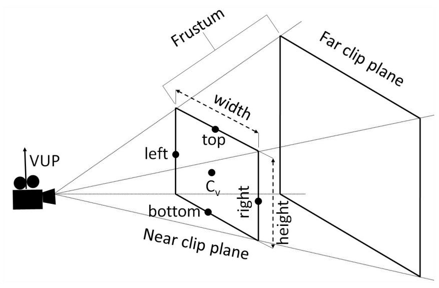

Figure 3: Virtual camera. Where VUP is the vector that indicates the camera vertical axis. width and height are the real image width and height. C_v is the principal point. The points top,left,right and bottom are the known points to define the clipping planes.

the euclidean camera. Secondly, we have to define the frustum clipping planes as is shown in Figure 3. To calculate them, we obtain the distances from principal point to the sides of the near clip plane. We know that the dimensions of the near clip plane are the real images dimensions. Firstly, we obtain four points, one for each clipping plane. These points are top,right,left and bottom, as we can see in Figure 3. They are defined using the near clipping plane dimensions and coordinates of the image centre $C = (x_c, y_c)$, where x_c and y_c are extracted from the intrinsic parameters, expression 7. Then, they are multiplied by the inverse projection matrix:

$$top = P^{-1} \left(x_c, height - 1, 1, 1\right)^T \tag{15}$$

$$bottom = P^{-1} \left(x_c, 0, 1, 1\right)^T \tag{16}$$

$$right = P^{-1} \left(width - 1, y_c, 1, 1\right)^T \tag{17}$$

$$left = P^{-1} \left(0, y_c, 1, 1\right)^T \tag{18}$$

Now we have to calculate the distances between the principal point and the points previously calculated to pass them to OpenGL as parameters to define the frustum.

Finally, we obtain the VUP projecting the vector from principal point to up side of the near clip plane:

$$VUP = P^{-1} \left(x_c, 0, 1, 1\right)^T - (2C - C_v) \tag{19}$$

As a result of the synchronisation, the virtual camera is able to obtain images with the same perspective of the real camera as we can see in Figure 4, which shows a virtual camera synchronised with a real camera.

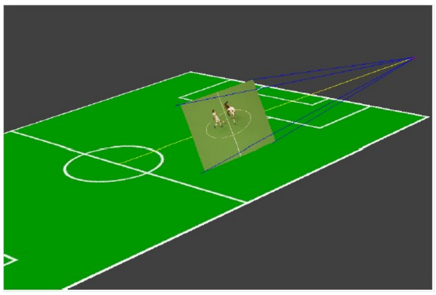

Figure 4: Virtual camera synchronised. The real image is in the near clip frame, where the virtual objects are rendered.

6 Rendering

In the initialisation stage, we have configured in the virtual world different polygons placed on the grass and 3D objects using OpenGL. To perform a correct rendering, we have taken into account the lens distortion model. Applying an inverse distortion model to polygon points we distort the polygons which will fit better with real distorted image. When the cameras have been synchronised, we can render the resulting frame adding the virtual content to the real image. For example, if in Figure 4 we add objects in the central circle, with OpenGL we can project them onto the real image. But first, a segmentation of the real image must be done. This segmentation allows differentiate the grass from the players and the lines. That is useful for the virtual ads which are painted on the grass. We only replace the pixels belonging the grass avoiding the occlusion of the players or lines. The segmentation is done converting image to HSV space. We calculate the H histogram and get the maximum value. It will be the green of the grass because in our scenario, it is the dominant colour. But it is normal finding different tones of green in the grass, to deal with that, we use thresholds to select an interval of H values that belongs to the grass. The mask obtained is stored in the alpha channel of the original image, this channel is used by OpenGL to know which pixel is transparent. Then we paint the original image as background of our virtual world, and OpenGL replaces transparent pixels with pixels from the virtual world.

7 Numerical experiments and results

We have tested our method on different video sequences using both, scale models and real scenes from soccer matches. The sequences acquired using the scale model consist of 1440 x 809 frames. Real soccer sequences are 1920 x 1080 high definition video sequences. Before applying the initialisation stage to the sequence, we have to obtain certain information, such as the learning data set for the decision tree and the tripod calibration. The learning data set is manually obtained by segmenting the first frame of the sequence, with only two different classes: primitives (white lines and circles) and grass. An example of manual segmentation is shown in Figure 5. Tripod geometry calibration is calculated with some frames extracted at different instants of the sequence. We can see the results for these frames in Figure 6. With this information, we perform the initialisation stage and calibrate the whole sequence. The calibration we have obtained is used for the insertion of graphics into the video. The results are shown in Figures 7, 8, 9, 10 and in videos provided as supplementary material (the videos are also in http://www.ctim.es/demo104/). Since we deal with cameras mounted on a tripod, the restrictions of the geometry of the tripod strongly simplify the problem of camera calibration and allow recovering an accurate frame calibration in cases where standard calibration methods fail.

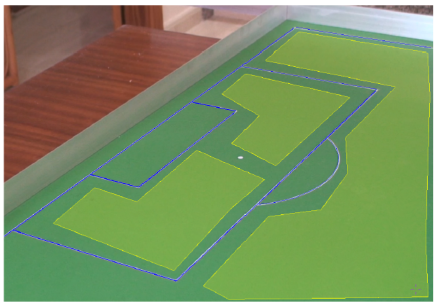

Figure 5: Two different classes are used in the manual segmentation: white primitives and grass. Grass is segmented using polygons, whereas segments are used for primitives.

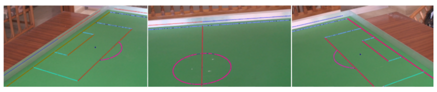

Figure 6: Tripod geometry calibration for three reference frames. To validate the results, soccer field primitives are projected onto the real images using different colours

8 Conclusions

In this paper we study the augmented reality in sport scenarios using cameras mounted on a tripod, in these scenarios there are usually a small number of visible primitives which can be considered to perform the calibration. To solve this problem, we firstly assume that the camera is mounted on a tripod (which is a common situation in practise) and we study the geometry of the tripod from a mathematical point of view. This assumption strongly simplifies the calibration problem and allows recovering the frame calibration in situations where general calibration techniques fail. Secondly, we use a simple method for primitive tracking based on a CART (Classification and Regression Tree). This method is used in the calibration procedure and takes into account colour information. Besides, for camera synchronisation we made a correspondence between real camera and virtual camera, calculating virtual camera parameters from real camera parameters. Finally, we render using OpenGL because it offers easy management of virtual camera and optimised graphic processing at graphic card.

We present some experiments using HD videos of sport events (soccer matches) in both, scale models and real scenarios. In order to validate our approach, we insert some graphics into the video sequences. The numerical results we present are precise and very promising.

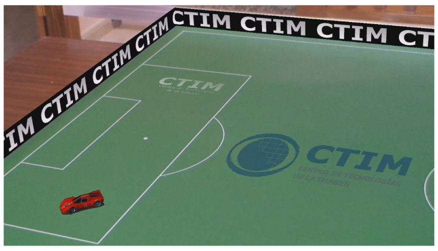

Figure 7: Graphic insertion with camera calibration using tripod geometry and primitive tracking. Extracted from the videos provided as supplementary material (http://www.ctim.es/demo104/).

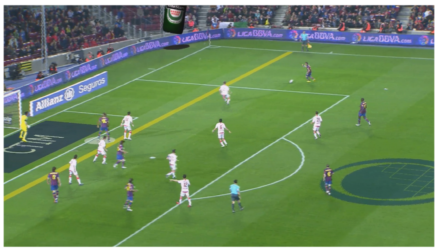

Figure 8: Graphic insertion with camera calibration using tripod geometry and primitive tracking. Extracted from the videos provided as supplementary material (http://www.ctim.es/demo104/).

Acknowledgement

This research has partially been supported by the MICINN project reference MTM2010-17615 (Ministerio de Ciencia e Innovación. Spain). We acknowledge MEDIAPRODUCCION S.L. for providing us with the real HD video

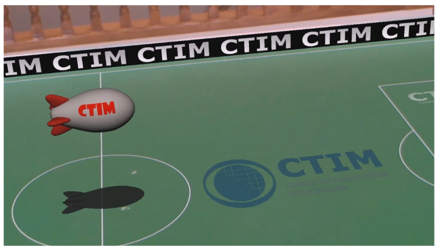

Figure 9: Graphic insertion with camera calibration using tripod geometry and primitive tracking. Extracted from the videos provided as supplementary material (http://www.ctim.es/demo104/).

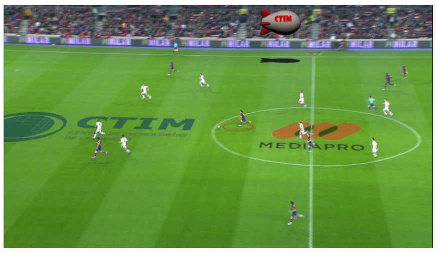

Figure 10: Graphic insertion with camera calibration using tripod geometry and primitive tracking. Extracted from the videos provided as supplementary material (http://www.ctim.es/demo104/).

we use in the numerical experiments.

References

[1] L. Breiman, JH. Friedman, RA. Olshen, CJ. Stone: Classification and Regression Trees. Belmont, CA: Wadsworth, 1984.

[2] D.Shreiner, M. Woo, J. Neider, T. Davis: OpenGL Programming Guide. 2007.

[3] JB. Hayet, J. Piater: On-Line Rectification of Sport Sequences with Moving Cameras. In: MICAI 2007: Advances in Artificial Intelligence, volume 4827, pages 736-746, 2007.

[4] H. Kim, KS. Hong: Robust Image Mosaicing of Soccer Videos using Self-Calibration and Line Tracking. In: Pattern analysis and applications, volume 4, pages 9-19, 2001.

[5] D. Farin, S. Krabbe, PHN. de With, W. Effelsberg: Robust camera calibration for sport videos using court models. In: Storage and Retrieval Methods and Applications for Multimedia, volume 5307, pages 80-91, 2004.

[6] D. Farin, J.G. Han, PHN. de With: Fast camera calibration for the analysis of sport sequences. In: IEEE International Conference on Multimedia and Expo (ICME), volume 1-2, pages 482-485, 2005.

[7] M. Aleman-Flores, L. Alvarez, P. Henriquez, L. Mazorra: Morphological Thick Line Center Detection. In: 7th International Conference on Image Analysis and Recognition, volume 6111, pages 71-80, 2010.

[8] E. Hayman, D. Murray: The effects of translational misalignment when self-calibration rotating and zooming cameras. In: IEEE Transactions on Pattern Analysis and Machine Intelligence, volume 25, pages 1015-1020, 2003.

[9] Q. Li, Y. Luo: Automatic camera calibration for images of soccer match. In: Proceedings of World Academy of Science, Engineering and Technology, volume 1, pages 170-173, 2005.

[10] J Han, D. Farin, P.H.N. de With: A mixed-reality system for broadcasting sports video to mobile devices. In: IEEE Multimedia, volume 18, pages 72-84, 2011.

[11] S. Li, B. Lu: Automatic camera calibration technique and its application in virtual advertisement insertion system. In: 2nd IEEE Conference on Industrial Electronics and Applications. ICIEA 2007, volume 1, pages 288-292, 2007.

[12] K. Wan, X. Yan: Advertising insertion in sports webcasts. In: IEEE Multimedia, volume 14, pages 78-82, 2007.

[13] C. Chang, K. Hsieh, M. Chiang, J. Wu: Virtual spotlighted advertising for tennis videos. In: Journal of visual communication and image representation, volume 21, pages 595-612, 2010.

[14] L. Alvarez, P. Henriquez, J. Sanchez: CART application to image primitives tracking. In: Conferencia de la Asociacion Espaola para la Inteligencia Artificial (CAEPIA), 2011.

[15] J. Knight, A. Zisserman, I. Reid: Linear auto-calibration for ground plane motion. In: 2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, volume 1, pages 503-510, 2003.