Bachelor Semester Project

Automatic Multiple Camera Calibration

CVLAB - EPFL

Arthur Giroux
arthur.giroux@epfl.ch

June 11, 2013

Abstract

The goal of the project is to provide a simple and automatic way of calibrating multiple cameras that will be use in a bigger project for multiple people tracking done in the CVLAB.
Calibrating a camera means that we have a way to know the position in space of this camera and all that it sees.
The calibration is a very important part of the multiple people tracking project because you can't do anything without it and when you have a fully user-based tool it can take hours to do the calibration for several cameras.

All the tools that I made are available at `https://github.com/arthurgiroux/depth-camera-automatic-calibration`

Contents

Chapter 1

Introduction

At first the goal of the project was to do a web interface where people could upload their data and they will get a calibration but we realized that the problem of the calibration itself was challenging and adding more constraints on top of it will be too much.
So we decided to focus on making tools to calibrate the cameras that could be easily imported in the current pipeline.
The goal was to track the marker in space and compute the calibration with just these data.

Chapter 2

Tracking a marker

2.1 Theory

2.1.1 Purpose

In order to estimate the position in space of a point we need to view it from at least two different cameras. Thus we need a simple way to have a set of known points that are visible from multiple cameras.
To achieve this we will use a calibration wand that is composed of two balls (a red one and a green one) separated by a known distance (in our case 1 meter). Then we will wave the wand in the space in front of the cameras so that it's seen from multiple cameras and that it covers a lot of space.

2.1.2 Color segmentation

The balls are easily distinguishable by their color, to track them efficiently we first need to select only the red and green objects (cf. 2.2).
To do this we go from RGB to HSV. HSV is a representation of RGB where the color of a pixel is determined by 3 components : the **Hue**, the **Saturation** and the **Value**.

This representation allow us to easily do a segmentation of our images depending on the color. Indeed we know that our ball is red but it will be brighter or darker depending on the lights and the orientation. We easily determine a range for the HSV where the Hue and Saturation can vary a lot but the Value doesn't change much.
We do two different range for the HSV value, one for the "brighter" and one for the "darker" part of each marker and we assemble the two.

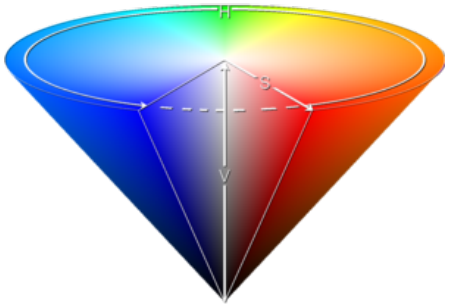
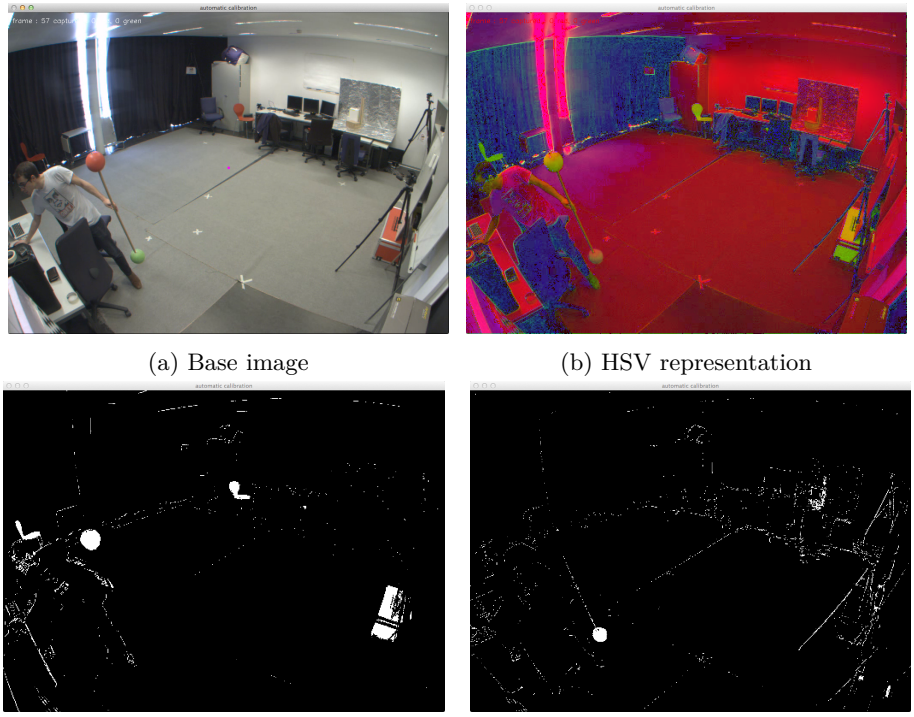


Figure 2.1: HSV Representation



(a) Base image

(b) HSV representation

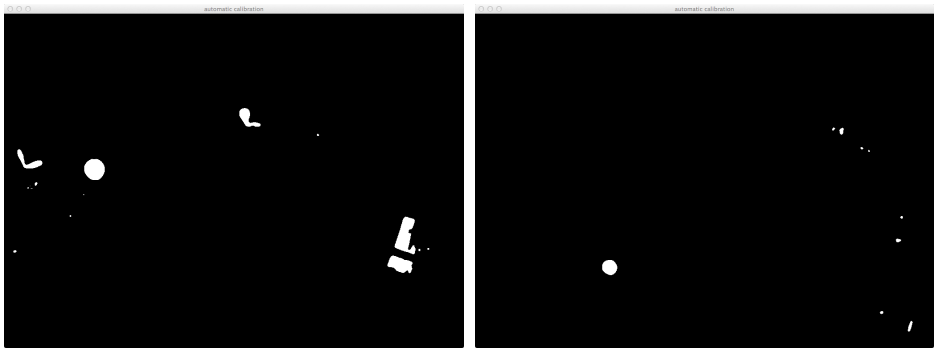
(c) Red segmentation

(d) Green segmentation

Figure 2.2: HSV segmentation, this is the result of only keeping green / red objects in our images

We can see that we have our green and red marker visible but now we need to get rid of all the imperfections in order to begin the tracking.
For this we will use the combination of two transformation: Opening and Closing.
The Opening transformation is useful for removing small objects and the Closing transformation is useful for removing small holes.
We also apply a gaussian blur to smooth the image and remove any remaining holes.

The effects of the transformations are shown in Fig. 2.3.



(a) Red (b) Green

Figure 2.3: Filtered segmentation, this is a clean result of only big green / red objects in our images

2.1.3 Circle detection

Now what we need to do is to detect circles in these images, for this we will use the Hough Circle Transform (cf. Fig. 2.4). It will give use a set of detected circles with the center and the radius.

Figure 2.4: Circle detection using Hough transformation

We can see that the two balls are detected but we also have a lot of noise. How can we select the right circles among them ? To solve this issue we will add some constraints specifics to our problem:

2.1.3.1 Spatial coherency

Let's assume that we have successfully tracked the marker at time T. then we know that at time $T + 1$ the marker should be very close to this last known position.

In fact we can fairly assume that the new circle corresponding to the marker will be the circle closest to the last known. So now we only have to check the closest circle to the previous one.

But what happens if right after time T the marker is not visible ? Then it will take the closest circle (that surely will be some noise) and take this noise as our marker. We don't want that ! That's why we will use temporal coherency.

2.1.3.2 Temporal coherency

We know that the marker travel at a certain average speed. The marker can't suddenly appear in (1200, 1200) if it was in (5, 5) one second ago !

We have the time T of the last known marker position. Let's say that we now are at time T_2.
We take the new detected circle as our new marker position if and only if the distance between it and the last position is less than

$$AVG\_SPEED * (T_2 - T)$$

So we will always take the new position of the marker as the closest new detected position and we allow it to be further away depending on the time elapsed since the last recorded marker position.

2.2 Implementation

2.2.1 usage

You can launch the tool as follows:

./tracking <path to video file> <output path>

2.2.2 Playback controls

The playback controls window (Fig. 2.5) allows you to navigate into the video stream. You can also press the spacebar to pause the video.

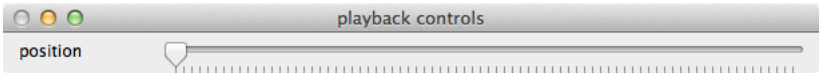


Figure 2.5: Playback controls window

2.2.3 Parameters window

The parameters window (Fig. 2.6) allows you to change the parameters of the Hough Circle Transformation from OpenCV.

Center threshold – It is the accumulator threshold for the circle centers at the detection stage. The smaller it is, the more false circles may be detected. Circles, corresponding to the larger accumulator values, will be returned first.
Max radius – Maximum circle radius.
Canny threshold – It is the higher threshold of the two passed to the Canny() edge detector (the lower one is twice smaller).
Min distance – Minimum distance between the centers of the detected circles. If the parameter is too small, multiple neighbor circles may be falsely detected in addition to a true one. If it is too large, some circles may be missed.
Accumulator resolution – Inverse ratio of the accumulator resolution to the image resolution. For example, if parameter=1 , the accumulator has the same resolution as the input image. If parameter=2 , the accumulator has half as big width and height.
Max radius – Maximum circle radius.

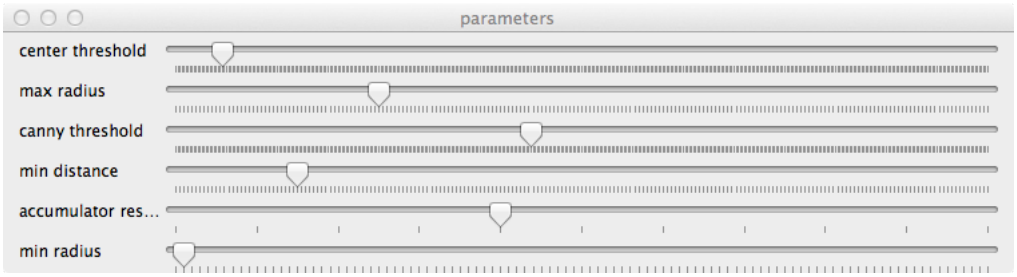


Figure 2.6: Parameters window

2.2.4 Main window

This is where the tracking operates (Fig. 2.7).
The last known red point will appear teal and the last green will appear pink.

When the tool record a point the circle will be blue, otherwise it's red.

First of all you have the following controls at you disposition to navigate into the different views:

1 – Main image.
2 – HSV image.
3 – Red AND green segmentation.
4 – Red segmentation.
5 – Green segmentation.

Then you have the following controls to do the actions:

R – Start/Stop the recording.
W – Write the tracking to the files filename_red.txt and filename_green.txt in the output path.
S – Show/Hide the path of the recorded points.
D – Delete **all** the recorded point.
J – Delete the last recorded **red** point.
K – Delete the last recorded **green** point.

You can also refine the tracking by setting a fake last known point for the **red** (**left** click) and **green** (**right** click) anywhere on the image.

Tracking steps example:

- First of all left/right click roughly where the markers are to refine the position.
- Then adjust the parameters if the markers are not nicely detected.
- Once you are satisfied you can launch the recording with "r" and see the points being recorded with "s".
- If some points are recorded but shouldn't you can pause, use "j" or "k" to delete the points, skip the part and unpause.
- Once you are satisfied with the tracking you can press "w" and the tracking will be written to the files using the following format: <frame n#> <x> <y> <radius>

Figure 2.7: Tracking example

Chapter 3

Calibration

Now that we have the data using the tracking, we want to retrieve the calibration matrix for each camera.
First of all let's modelize the problem.

3.1 Pinhole Camera Model

The pinhole camera model (Fig. 3.1) represents the behavior of an ideal pinhole camera, it states that in the absence of noise, a camera maps world points to image points as follows :

$$\lambda \mathbf{u} = \mathbf{P}\mathbf{x} \tag{3.1}$$

$$\mathbf{u} = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \quad \mathbf{x} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Where $\boldsymbol{u}$ is the 2D coordinates, $\boldsymbol{x}$ is the 3D coordinates, λ is a scaling factor and $\boldsymbol{P}$ is the Projection matrix.

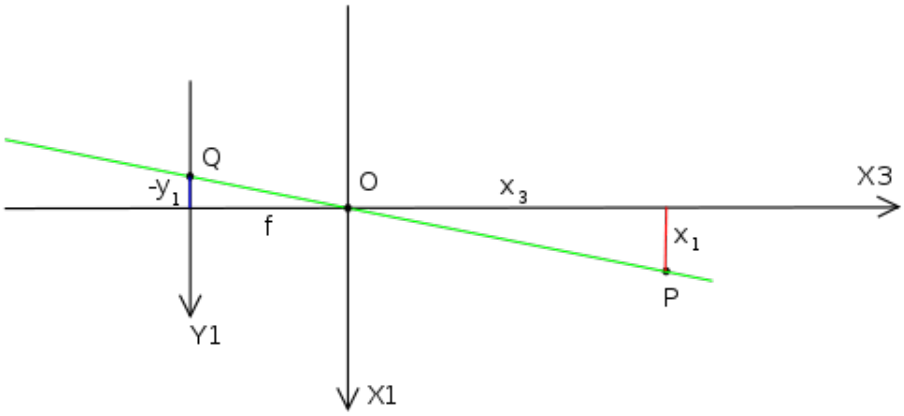


Figure 3.1: Pinhole camera model

Now that we have a model to represent our problem, we need to analyze all the parameters that takes part in it.

3.2 Analyzing the parameters

3.2.1 Projection Matrix

In 3.1, $\mathbf{P}$ represents the projection matrix and it's what we want to solve for each of our cameras.
It can be decomposed as follows:

$$\mathbf{P} = \mathbf{K}\,[\mathbf{R} \mid \mathbf{t}] \tag{3.2}$$

$\mathbf{K}$ intrinsic parameters matrix, it represents all the parameters internal to the camera and independent of the view.
$[\mathbf{R} \mid \mathbf{t}]$ extrinsic parameters matrix, denote the transformation from 3d-world coordinates to 3d-camera coordinates
$\mathbf{R}$ rotation matrix
$\mathbf{t}$ translation matrix

3.2.2 Intrinsic parameters

We assume that we have this matrix for all the cameras. It can be easily obtain using a grid calibration tool that I provide in my repository.

$$\mathbf{K} = \begin{bmatrix} f & 0 & u_0 \\ 0 & \alpha f & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

f : effective focal length
α : aspect ratio
(u_0, v_0) : center of projection

Now that we have an overview of the model and its parameters, we can express the problem and find a way to resolve it.

3.3 Expressing the problem

Using the equation 3.1 we can easily express the 2D points from the 3D points and the projection matrix:

$$u = \frac{\mathbf{p}_1\mathbf{x}}{\mathbf{p}_3\mathbf{x}} \quad v = \frac{\mathbf{p}_2\mathbf{x}}{\mathbf{p}_3\mathbf{x}}$$

Where $\mathbf{p}_i$ denotes the i^{th} row of $\mathbf{P}$, (u, v) is the 2D point coordinates and $\mathbf{x}$ the

3D point coordinates.

Therefore we can express the problem as a minimization problem. Indeed, we want to find the projection matrix for each camera such that it minimizes the error between the known set of 2D points and the reconstructed points.

We can express the problem as the minimization of this equation where N is the number of cameras and M is the number of known 2D points:

$$E_{geom} = \sum_{i=1}^{N} \sum_{j=1}^{M} \left| u_{ij} - \frac{\mathbf{p_{i1}} \mathbf{x}_j}{\mathbf{p_{i3}} \mathbf{x}_j} \right|^2 + \left| v_{ij} - \frac{\mathbf{p_{i2}} \mathbf{x}_j}{\mathbf{p_{i3}} \mathbf{x}_j} \right|^2$$

Now that we have a way of expressing the problem, we can find an initial solution.

3.4 Initial solution

To find the initial solution we will mainly use the fact that we have a set of 2D points visible from at least 2 views.

3.4.1 Fundamental Matrix

Given points from two differents views, a fundamental matrix $\mathbf{F}$ is a projection of the points from one view to the other (Fig. 3.2). We can find this matrix easily given at least 8 corresponding points.

Let's recall that we have the intrinsic parameters matrix $\mathbf{K}$ for all our cameras.
Besides we have the following relation between the Fundamental matrix $\mathbf{F}$, the Essential matrix $\mathbf{E}$ and the intrinsic matrix of the two cameras $\mathbf{K}_1$ $\mathbf{K}_2$

$$\mathbf{E} = {\mathbf{K}_2}^\top \mathbf{F} \mathbf{K}_1$$

And $\mathbf{E}$ can be decomposed

$$\mathbf{E} = \mathbf{R} \left[\mathbf{t} \right]_\times$$

Where $\mathbf{R}$ denotes the rotation and $\mathbf{t}$ the translation to go from one camera coordinates to the other.

We need to find these $\mathbf{R}$ and $\mathbf{t}$ to deduce our projection matrix $\mathbf{P}$.

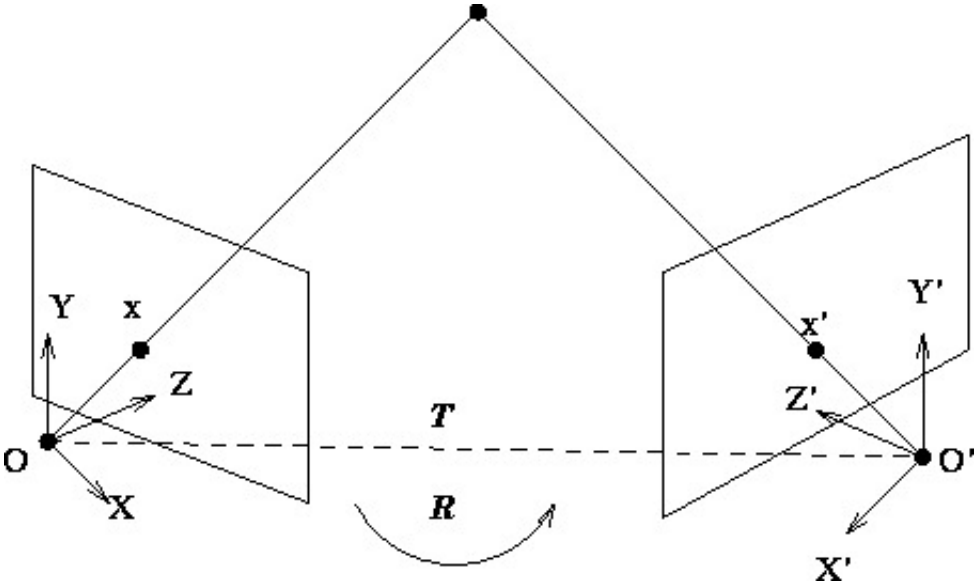


Figure 3.2: Fundamental matrix representation

3.4.2 Decomposition of E

We can find $\mathbf{R}$ and $\mathbf{t}$ using an SVD decomposition of $\mathbf{E}$.
Indeed, an SVD of $\mathbf{E}$ gives

$$\mathbf{E} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathbf{T}}$$

Where $\mathbf{U}$ and $\mathbf{V}$ are orthogonal 3×3 matrices and $\mathbf{\Sigma}$ is a 3×3 diagonal matrix with

$$\mathbf{\Sigma} = \begin{pmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

The diagonal entries of $\mathbf{\Sigma}$ are the singular values of $\boldsymbol{E}$.

Now we define

$$\boldsymbol{W}^{-\mathbf{1}} = \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$\boldsymbol{Z} = \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

And we have

$$\boldsymbol{R} = \boldsymbol{U}\boldsymbol{W}^{-1}\boldsymbol{V}^{\boldsymbol{T}}$$

and

$$[\boldsymbol{t}]_{\times} = \boldsymbol{V}\boldsymbol{Z}\boldsymbol{V}^{\boldsymbol{T}}$$

There are four solutions for $(\boldsymbol{R}\,\boldsymbol{t})$ because we can replace $\boldsymbol{W}^{-1}$ with $\boldsymbol{W}$ and $\boldsymbol{t}$ with $-\boldsymbol{t}$ but among these solutions only one is possible in a practical point of view.

Indeed, only one will give a 3D point which lies in front of both cameras, the others will lie behind at least one camera.

To find the right solution we use a perspective transformation on the points for each camera and we check that $z > 0$ for both transformation.

We have found $\boldsymbol{R}$ and $\boldsymbol{t}$ but $\boldsymbol{t}$ has been found up to a scaling factor s. To find this scaling factor we need to use the fact that we know the physical distance between the two markers.

Thus we can reproject both markers in 3D, get the distance between the two and divide this distance by our physical known distance to find the factor s.

So we can find the fundamental matrices for successive pairs of camera $(1,2), (2,3), ..., (N-1, N)$ and we can extract the relative $(\boldsymbol{R}\,\boldsymbol{t})$ for each one.

3.4.3 Defining a global coordinate system

Now that we have computed the relative rotation and translation

$$(\boldsymbol{R_{1,2}}\ \boldsymbol{t_{1,2}}), (\boldsymbol{R_{2,3}}\ \boldsymbol{t_{2,3}}), ..., (\boldsymbol{R_{N-1,N}}\ \boldsymbol{t_{N-1,N}})$$

We can get our projection matrix.

For this we will define a global coordinate system which take the first camera as the reference:

$$\begin{array}{rclrcl} \boldsymbol{R_1} & = & \boldsymbol{I} & \boldsymbol{t_1} & = & \boldsymbol{0} \\ \boldsymbol{R_{n+1}} & = & \boldsymbol{R_n R_{n,n+1}} & \boldsymbol{t_{n+1}} & = & \boldsymbol{R_n t_{n,n+1}} + \boldsymbol{t_n} \end{array}$$

We now have an initial solution for our projection matrix. But as our problem is a minimization one, it can be improves to find the best solution from an initial solution.

3.5 Refining the solution

For refining our initial solution to find the best one that fits our constraint we will use a bundle adjustment method.
The bundle adjustment uses some initial approximation and so some optimization until it converges.
These are the steps of the bundle adjustment

Let $\boldsymbol{X}$ be the initial estimate of parameters

Find a increment $\boldsymbol{\Delta X}$ such that

$$|f(\boldsymbol{X} + \boldsymbol{\Delta X}) - \boldsymbol{Z}| < |f(\boldsymbol{X}) - \boldsymbol{Z}|$$

Replace $\boldsymbol{X} \leftarrow \boldsymbol{X} + \boldsymbol{\Delta X}$
Repeat until it converges.

The difficulty in this method is to find a good $\boldsymbol{\Delta X}$ such that it allows the solution to converges rapidly.
To compute this $\boldsymbol{\Delta X}$ we will use the Levenberg-Marquadt method.

It computes $\boldsymbol{\Delta X}$ as the solution of

$$\left(\boldsymbol{J}^\top \boldsymbol{J} + \lambda \boldsymbol{I}\right) \boldsymbol{\Delta X} = \boldsymbol{J}^\top \left(f(\boldsymbol{X}) - \boldsymbol{Z}\right)$$

in our case we can initialize λ at 0.001
If $\boldsymbol{\Delta X}$ gives a better result, we update $\boldsymbol{X}$ and we set $\lambda \leftarrow 10\lambda$

if not we set $\lambda \leftarrow \frac{\lambda}{10}$ and recomputes $\boldsymbol{\Delta X}$

3.6 Superposition of our calibration with the real world

Now that we have found our optimal projection matrices, we might want to superpose our calibration on top of our real world model.

Indeed for now the origin of the calibration is the first camera but if for example your setup is made to track a volleyball match, then you want your origin to be on the floor and at one of the corner of the court.

To do this the easier way is to put an object easily distinguishable (with at least 3 points, a grid calibration for example) where you want your origin and in a way that it's axis-aligned.

Now what we will do is to view this object from the first camera (the current origin), get the 2D location of each point on this object and solve the Perspective-N-Point (PnP) problem.
It will give us a rotation $\boldsymbol{R}$ and a translation $\boldsymbol{t}$ matrix to go from the object coordinates to camera coordinates.

Now the only thing to do is multiply all the rotations in our projection matrices with $\boldsymbol{R}^\top$ and add to all our translation matrices $-\boldsymbol{t}$.

That way our new origin for our coordinates system will be at the origin of this object.

Chapter 4

Conclusion

I've successfully implemented tools to get a semi-automatic calibration for multiple cameras (Fig. 4.1).
The tools are not perfect but the implementation is simple and can be integrated easily in an existing pipeline.

Because this was not a simple task and I didn't have much time we didn't do self-calibration for the intrinsic parameters as it was planned and we added a step where you use a calibration grid to get these values.

I didn't go into a fully automatic path because it required to implement much more things in the tracking part like a more robust circle detection (that OpenCV doesn't provide), the use of background subtraction and more constraints on the markers.

For the calibration part it was very challenging for me because I didn't have any background in computer vision and the calibration field is very wide. I spent a lot of time reading papers that were all saying different things and I had to separate the right from wrong.
Besides, as multiple camera setup is not very common yet, I was stuck sometimes for hours on a little technical detail that I couldn't find anywhere.

I had a lot of fun playing with OpenCV, it's a great API and I had to really dive into it in this project.

In the end I've learned a lot about multiple view geometry, camera calibration and computer vision in general. I really enjoyed doing this project because it was something concrete and useful and I was fully involved in it even for filming the sport games or setting up the cameras.

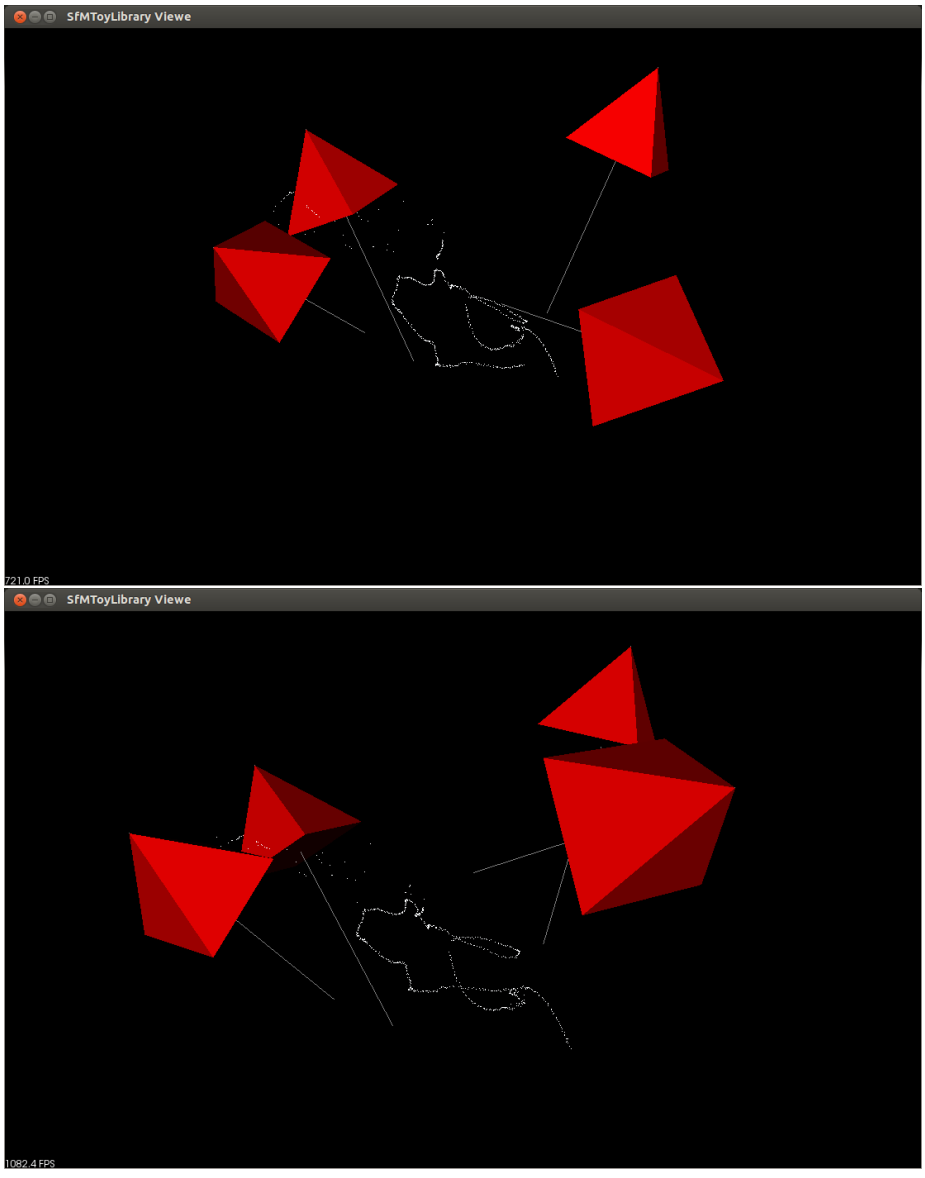


Figure 4.1: Calibration example, showing the reconstructed path of the red marker and the cameras position

Bibliography

[1] Baggio Daniel. *Mastering OpenCV with Practical Computer Vision Projects.* PACKT publishing, 2012.

[2] Wedge Daniel. The fundamental matrix song. http://danielwedge.com/fmatrix/.

[3] Xu Gang and Zhang Zhengyou. *Epipolar geometry in Stereo, Motion and Object Recognition.* Kluwer Academic Publishers, 1996.

[4] Mitchelson Joel. *Multiple Camera Studio Methods for Visual Capture of Human Motion.* PhD thesis, University of Surry, 2003.

[5] Lourakis Manolis and Deriche Rachid. Camera self-calibration using the singular value decomposition of the fundamental matrix. Technical report, INRIA, 1999.

[6] OpenCV. Opencv documentation. http://opencv.willowgarage.com/documentation/cpp/.

[7] Hartley Richard and Zisserman Andrew. *Multiple View Geometry in Computer Vision Second Edition.* Cambridge University Press, march 2004.

[8] Svoboda Tomas. A software for complete calibration of multicamera systems, 2005.

[9] Zhang Z. A flexible new technique for camera calibration. Technical report, IEEE Transactions on Pattern Analysis and Machine Intelligence, 2000.