Computer Vision

UNIVERSITY OF TRENTO

CV Project

Project - Sport 2

3D camera calibration

Student *Marco Bonan*

241478

Professor *Nicola Conci*

Teaching assistant *Niccolò Bisagno*

September 2024

Abstract

This project focuses on the calibration cameras surrounding the SANBÀPOLIS Sports Hall. The goals of the project are to calibrate the cameras, reconstruct their positions, and develop a graphical user interface (GUI) for simultaneous point highlighting across multiple camera views.

Contents

1 Introduction

The *SANBÀPOLIS* Sports Hall, (Trento) is surrounded by 13 UHD cameras. 3 are placed on the ceiling, facing down on the court, and the others are placed around the court. The cameras fully cover the playing area and can be used for multiple purposes. In this project only side cameras are considered. The ultimate goal of this work is to:

- perform intrinsic calibration of the cameras;
- perform extrinsic calibration of the cameras, reconstructing their position in the arena;
- develop a GUI tool to highlight points on the court surface on all camera views simultaneously.

1.1 Scope of the work

Camera calibration (both intrinsic and extrinsic) is fundamental to reconstruct the 3D geometry location and serves as a basis for more complex algorithms such as 3D reconstruction of the environment or 3D tracking, (of players, the ball) during sport matches. Those process are nowadays very common in sport events. Analytics can provide high value for spectators, referees, teams, event organizers and even sport betting companies. Last but not least, game data can be harvested for *Big Data* applications.

Marco Bonan

2 Methodology

The project is developed in python, relying mainly on the OpenCV `opencv-python` package. The code is mainly structured in a custom library called `CameraUtils.py` which contains classes, methods and other function, that are used in the Calibration scripts, for the sake of code reusability and simpler code structure. This Project [4] was used as inspiration and starting point, in particular the algorithm to find and extract the calibration sample frames is derived from the project.

More information about the code and instruction on how to use it can be found on the archive README.

2.1 SetUp

All ten cameras displaced around the field are identified with a camera ID and an approximate position according to figure 1. Moreover, each cam must be treated either as normal camera or a wide-lens camera (more on this later). These info are initialized in the configuration file in order to successfully instantiate objects for each camera. The config file contains many editable-options for the calibration scripts.

2.2 Intrinsic calibration

A set of calibration videos was provided with a chessboard show to all cameras all along their field of view. For each camera a given number of samples is extracted from the video and stored. Each camera is calibrate on its samples chessboard images using the standard `cv2.calibrateCamera()` procedure. The obtained camera matrix is then refined and saved on the camera object, along with the re-projection error.

The chessboard square-size is known to be 28mm, however, if this parameter is passed to the calibration function, I was not able to get any useful calibration, though it is not a strictly necessary information, and by using an arbitrary-unit square size of 1, calibration was successful.

2.2.1 Wide-Lens

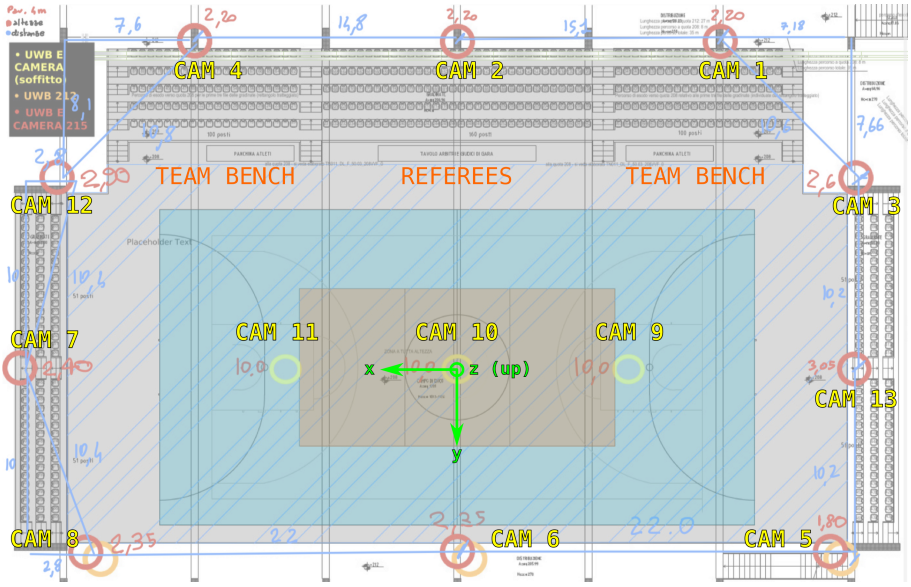
While coding the calibration scripts, I encountered issues with certain cameras that were not calibrating properly. This was particularly noticeable with cameras 2, 6, and 12. A common characteristic among these cameras is their wider field of view, so I decided to apply a fisheye camera model. OpenCV provides a dedicated module, `cv2.fisheye`, for this type of camera. By using this model, I was able to achieve satisfactory calibration and distortion correction.

To verify the success of the calibration, a comparison image (original vs undistorted) is displayed to the user at the end of the process

2.3 Extrinsic calibration

In order to perform the Extrinsic calibration of camera, each camera needs a set of known corresponding points (real world $\leftrightarrow$ image). Since no coordinates of key-points in the real world are provided, I had to rely on the court lines, which are standardized according to official rules of volleyball and basketball [1, 2]. The World reference frame origin is placed in the centre of the playing court, and its orientation is such that, looking from the main stands, the y-axis points straight, the x-axis points to the right, and the z-axis is orthogonal to the floor (WRF is shown in figure 1). I have identified sets of key point on the court surface, corresponding to corners and intersections of the lines, their coordinates are stored in the configuration file. Choosing only points on a 2D surface is not ideal, however as said earlier it was not possible to correctly known the location of points other than on the playing surface

Figure 1: Camera disposition in *SANBÀPOLIS*



[1] In order to simplify the acquisition of the pairs for each camera, I have developed a simple graphical user interface (GUI). For each camera the GUI (figure 2) reads the array of points form the config file and shows the current camera view and a court map with the current point highlighted. The user has the following options.

- Select the corresponding point on the image, the *matplotlib* utils such as the zoom can be used to better locate the points.
- Skip the current point acquisition, in case it is out of the FOV or it is obstructed.
- End the acquisition if there are enough matches, the minimum value is 6 [2], but more is better, since points on the volleyball courts are mostly collinear.
- Skip the whole camera calibration

Once the corresponding points are identified, they are saved as attributes to the camera instance. Then the `ExtCalibration()` method is called on the camera instance to compute the transformation matrices between the world and camera coordinates. The method calculates the rotation and translation vectors using the `cv2.solvePnP` function, which solves the Perspective-n-Point problem to find the position and orientation of the camera relative to the world coordinate system. It then computes the world-to-camera and camera-to-world transformation matrices and stores them as instance attributes. The latter matrix contain the estimation of the camera location in the real world, which will be discussed in section 3.

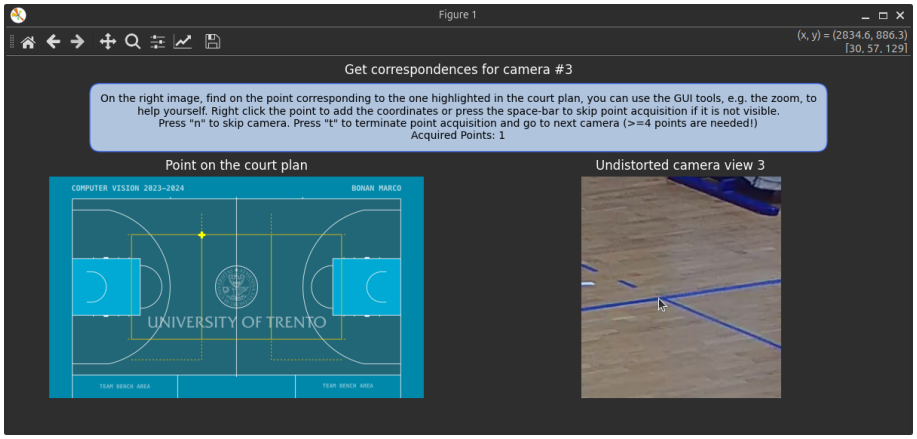
2.3.1 Localization of points in the court plane

The third goal of the project is to be able to simultaneously highlight a selected point in all views. In order to accomplish this, I have chosen to compute (for each camera) a plane to plane homograph

[1] Points on the net and/or its rods might be used but they are more difficult to identify
[2] The minimum number of pairs required by 'cv2.solvePnP' is 6

matrix which maps a point on th image plane to the court plane. The calculation of the homography matrix is also handled in the `ExtCalibration()` method using the `cv2.findHomography()` function.

Figure 2: GUI tool for correspondences selection



2.4 Points Highlighting

Points highlighting across all cameras is quite simple, yet effective, and it is based on a instance of th `PlotCameras` class. The script `PointsHighlighting` is an example of how to use the class.

The algorithm is simple, and the user interacts with a GUI (figure 3):

- All views are initialized and shown in a *matplotlib* window, roughly respecting their position in the real world, with respect to a scheme of the court.
- The user can right click on a point in one of the views and that point is mapped using the camera homography matrix to the real world coordinates, which are appended to an array.
- As new points are appended the `DrawPoints()` method maps the point the all camera views and to the court scheme. The mapped points are checked to be in the FOV of the camera, and filtered. All visible points are then drawn, sharing the same color to easier identification.
- The user can *clear* the highlighted point by pressing the "c" key

In order to achieve faster rendering, the images are scaled down, so image coordinates shown in the top right corner of the *matplotlib* window, are not corresponding to actual values.

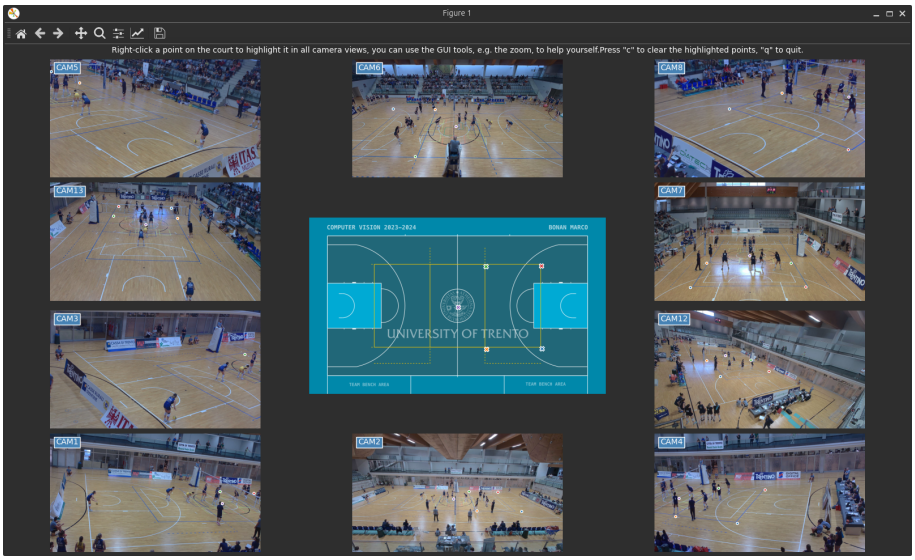
3 Results

This section presents the results from the camera calibration. It cover the calibration accuracy using metrics such as reprojection error and location error.

3.1 Intrinsic calibration

The intrinsic calibration is evaluated using the reprojection error metric. This metric is obtained by:

Figure 3: GUI tool for point highlighting



- calculating the L2 Norm (or Euclidean distance) for all points found in the calibration image and averaging this overall error by the number of point, effectively getting an average Euclidean error for each calibration image;
- the previous error is computed for all calibration images, and the results again averaged.

The end result is the final reprojection error averaged across all calibration images,this means that smaller the error (which is in pixels units), the better is the calibration. Reprojection error values are reported on table 1.

3.2 Extrinsic calibration

The estimate camera position of each camera is extracted from the first 3 elements of the 4th column of the Camera to World RF matrix, which correspond to the x,y,z coordinates in the court reference frame. Table 1 contains the estimated (x, y, z) position and the distance from (x, y, z) position retrieved from the annotations[3] in figure 1, which will be referred as *actual* position of the camera. Figure 4

3.2.1 Effect of the intrinsinc calibration on the extrinsic

In earlier results it seemed that a larger reprojection error lead to larger distance from the *actual* position. Figure 5 compares the reprojection error with the distance error for each camera, however it is hard to identify trends with this small amount of data. If we take in consideration the three largest positioning error, i.e CAM7, CAM4, CAM13 we can see that they have respectively the 6th, 8th and 4th best reprojection error.

CAM7 and CAM13 are placed on opposite ends of the court, roughly at the mid of the playing area, thus when taking correspondences, points are mostly collinear, this could be the main issue that cause a poor extrinsic calibration. Actually for these cameras more key points were added (belonging to the basketball court circular and semicircular areas) as an attempt to decrease the collinearity of points,

[3] Annotations whose accuracy is not known but should be roughly correct

however due to the symmetries of the playing lines, the easily identifiable points are mostly collinear, and such the external calibration did not improve much, remaining in error range of a couple meters of the current reported value.

A possible solution to confirm this hypothesis would be to step in the *SANBÀPOLIS* court and place markers (with adhesive tape) on the floor in random locations, and annotates their position in the WRF. An even better approach could be to bring a sort of calibration stick of adjustable height with an easily identifiable marker on the top, to get points also in the direction orthogonal to the court.

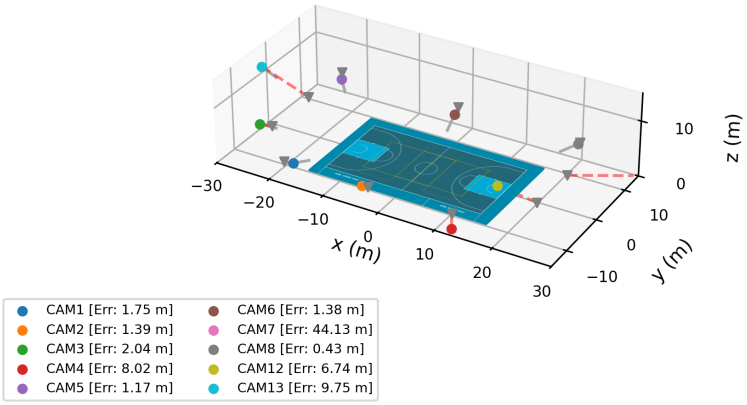
Table 1: Calibration Errors

CAM	x	y	z	est. x	est. y	est. z	distance	re-projection error
6	0	10.2	6.35	−0.1	9.15	5.46	1.38	0.068
12	24.8	−10	6.9	18.12	−10.05	7.81	6.74	0.168
2	0	−17.9	6.2	−1.28	−17.47	5.85	1.39	0.212
13	−22	0	7.05	−31.34	0.64	9.79	9.75	0.288
5	−22	10.2	6.8	−21.83	9.74	5.73	1.17	0.347
7	24.8	0	6.4	67.35	2.2	17.86	44.13	0.378
1	−15.1	−17.9	6.2	−14.23	−16.56	5.49	1.75	0.429
3	−22.3	−10.2	6.6	−23.99	−11.26	7.04	2.04	0.456
8	22	10	6.35	21.76	9.72	6.12	0.43	0.46
4	14.8	−18.1	6.2	18.51	−24.85	8.43	8.02	0.487

All values are expressed in meters, except for the re-projection error which is pixels

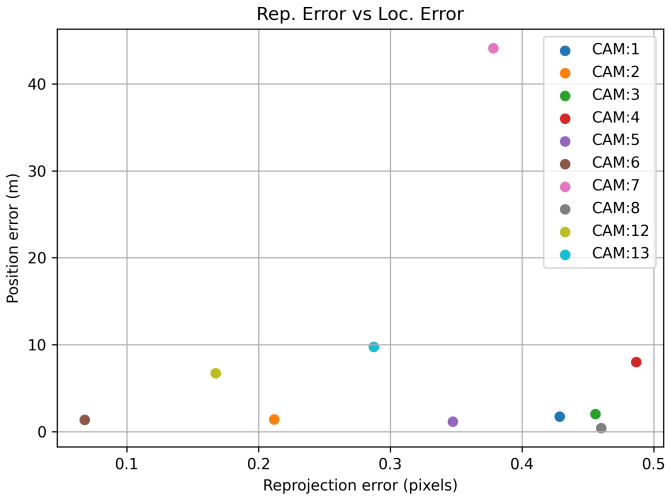
Figure 4

Camera positions with respect to the court



The plot size is limited, for this reason CAM7 is out of range an such not visible

Figure 5: Errors relationship



3.3 Points Highlighting

The points highlighting GUI is quite simple, yet it satisfy its requirements, there is not much word to spend on it, since its accuracy is strictly related to the homography matrix computation result.

This tool satisfies the project goal, providing an effective way to visually highlight points from multiple perspectives. This tool can be sued as a basis tool for visualization of player tracking, and with some changes it can be adapted to 3D tracking visualization. However, instead of relying on a simple homography matrix, a multi-stereo camera setup with proper calibration is required.

4 Conclusions

This project results can be summarized in:

- the creation of a python module `CameraUtils` for calibrating the SANBÀPOLIS side cameras, provided with documentation.
- scripts to calibrate the cameras and plot the results
- script to show the point highlighting across multiple views

The result of intrinsic calibration are quite satisfactory, the reconstruction of the camera position instead is less accurate, in addition to the already proposed attempt at improvement, the direction of future development could be integrating in this code base method for the multi stereo setup calibration, and try to use those result for the point mapping on the court.

For a visual description of the project, a video is attached to the project archive. More info can be found on the project repository [3].

References

[1] FIBA. *Official Basketball Rules 2023*. 2023. URL: https://www.fiba.basketball/documents/official-basketball-rules/current.pdf (visited on 06/20/2024).

[2] FIVB. *FIVB Volleyball Rules 2021-2024*. 2024. URL: https://www.fivb.com/wp-content/uploads/2024/03/FIVB-Volleyball_Rules_2021_2024_pe.pdf (visited on 06/20/2024).

[3] Bonan Marco. *CV-Project-Sport 2024*. 2024. URL: https://github.com/TatoPaato/CV-Project-Sport.

[4] Tommaselli E. Rizzetto A. *CV-CameraCalibration*. 2024. URL: https://github.com/Elia-Tomaselli/CV-CameraCalibration (visited on 06/20/2024).