Computer Vision - 3D camera calibration (geometry and 3D reconstruction) Project Report

Stefano Bonetto (247179), Simone Roman (247181)

University of Trento

stefano.bonetto@studenti.unitn.it, simone.roman@studenti.unitn.it

1. Introduction

This project aims to reconstruct a 3D model of a volleyball court using multiple camera views. Furthermore, we developed a tool that allows users to select a point in one camera view and see its corresponding location projected in another camera view. Additionally, we implement a 3D ball tracking algorithm that enables user to see a 3D plot of ball's trajectory during a volleyball match.
Project's pipeline (see Figure 1) starts with camera calibration, involving the estimation of intrinsic and extrinsic parameters. Intrinsic calibration is used to determine each camera's internal characteristics, such as focal length and lens distortion, while extrinsic calibration establishes the position and orientation of each camera relative to the court. Next, we implement multi-camera mapping by developing a tool that allows users to select a point in one camera view and see the corresponding point across other perspectives. This is achieved by applying homography matrix, which accurately maps points between views. Finally, we address 3D ball tracking by employing YOLOv11 for object detection, applying 3D triangulation techniques to reconstruct points in 3D space, and utilizing a particle filter to track the ball. All the code is available on our repo at github.com/Roman-Simone/computer-vision-project

2. Camera Calibration

Camera calibration is essential for transforming images from multiple camera views into a coherent 3D representation of a scene. Calibration involves determining two sets of parameters: intrinsic parameters, which describe the internal characteristics of each camera, and extrinsic parameters, which specify each camera's position and orientation relative to the real-world.
By finding both intrinsic and extrinsic parameters, we ensure that each camera view can accurately reflect real-world distances, angles, and spatial relationships.

2.1. Intrinsic parameters

To determine intrinsic camera parameters, three main state-of-the-art techniques are widely used. *Self-calibration* approaches[1] skip the need for calibration objects, instead relying on natural scene features captured from multiple angles. This method leverages image-intrinsic constraints, such as epipolar geometry, though often with reduced precision. *Deep learning-based* techniques[2] predict intrinsic parameters directly from images, trained on large datasets to quickly calibrate from real-world scenes without patterns. While flexible, this method depends on the diversity of training data and may lack the high precision of traditional methods. Lastly, *geometric-prior-based* methods[3] use known patterns to analyze the relationship between 3D pattern points and their 2D image projections, allowing for accurate calibration of focal length, principal points, and lens distortion.
Considering this last approach as the best trade-off for our use-case, we decided to opt for this method using a known calibration pattern: a chessboard grid[4].
To perform the camera calibration process, we adhered to the official OpenCV Camera Calibration[5] tutorial. This methodology proved to be both effective and straightforward, allowing us to accurately determine the necessary intrinsic parameters for each camera. A key requirement for this method is capturing a substantial number of images of a chessboard pattern from various angles and positions (see Figure 2). Although collecting multiple calibration images was time-consuming, it was crucial for minimizing reprojection errors (representing the distance between a projected point and a measured one).

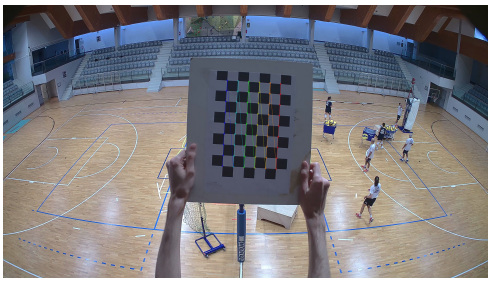

Figure 2: *A frame of the calibration video for camera n° 6*

2.2. Extrinsic parameters

Extrinsic parameters describe each camera's position and orientation in the world, or in this case, relative to the volleyball field. This information is essential for placing each camera in a shared 3D coordinate system, enabling all cameras to reference the same real-world scene.
In order to compute the extrinsic matrix, we need at least four paired points from the camera plane to the real world. To find them, we develop a tool (see Figure 3) that allows us to select the corners of the basketball and the volleyball court which are notable points with known coordinates in the real world.

Figure 3: *Tool for manually selecting the points in the views.*

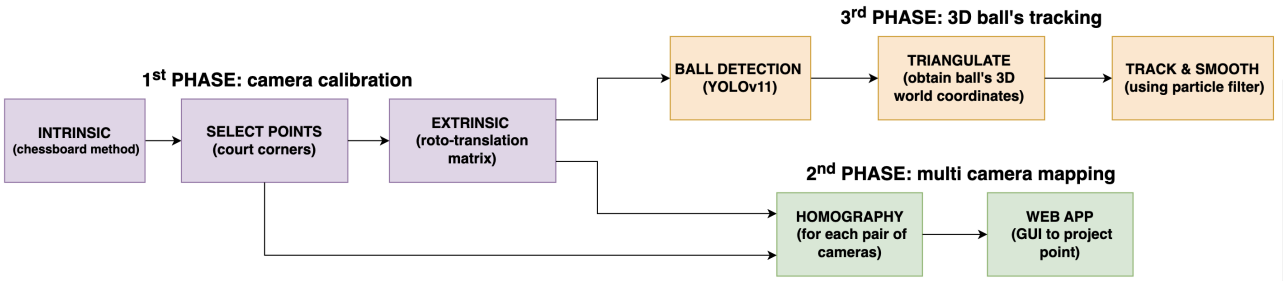

Figure 1: *Our project workflow is organized into three distinct phases. In the first phase, we estimate both intrinsic and extrinsic camera parameters. The second phase focuses on calculating the homography matrix for each pair of cameras, displaying corresponding points across different image planes through an interactive GUI. Finally, in the third phase, we perform 3D ball tracking using YOLOv11 as the detection model, combined with a particle filter to estimate the ball's trajectory.*

After collecting this reference points, we can compute the **translation vector** and the **rotation matrix** for each camera, which together define the camera's position and orientation in the real world. The extrinsic matrix maps 3D world coordinates to 2D image coordinates. In our application, we reverse this process: starting from 2D image coordinates, we project them back into 3D world coordinates by using the inverse of the extrinsic matrix. However, to fully reconstruct 3D positions, we also need depth information, which is obtained from multiple camera views.
The extrinsic parameter calibration enables a clear visualization of the camera setup within the unified 3D space (see Figure 4, ensuring that all cameras are accurately aligned and coordinated for a coherent 3D reconstruction.

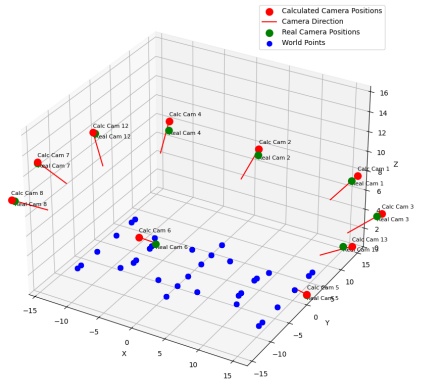

Figure 4: *3D visualization of camera positions, their directionality, and reference points in the world.*

3. Multi-Camera Mapping

The second step involve developing a tool that allows user to click on a specific point in one camera view and visualize the corresponding point across all other cameras.
In order to do that we compute the homography matrix[6] for each pair of cameras. This enables us to map points from one plane (the image captured by one camera) to another (the image captured by a different camera).
The process for computing the homography matrix involves three main steps:

1. We need at least four paired points between the two image planes. To obtain them we use the points collected in the previous phase for the extrinsic matrix calculation.
2. We process the image points using the intrinsic parameters to remove the effects of lens distortion, providing a normalized coordinate system that is essential for accurate mapping.
3. With undistorted points from both cameras, we use the `cv2.findHomography` function to compute the homography matrix.

This approach ensure precision and high accuracy in mapping points from one plane to another. However, his major drawback is that for points not belonging to the court plane (z = 0 in the world coordinate) the projected points are completely incorrect. In our case, this limitation is not critical, as our region of interest is confined to the court itself.
After doing this, we build an application that allows user to click on a point on a given camera and see the same correspondent point projected in one another camera view (See Fig 5).

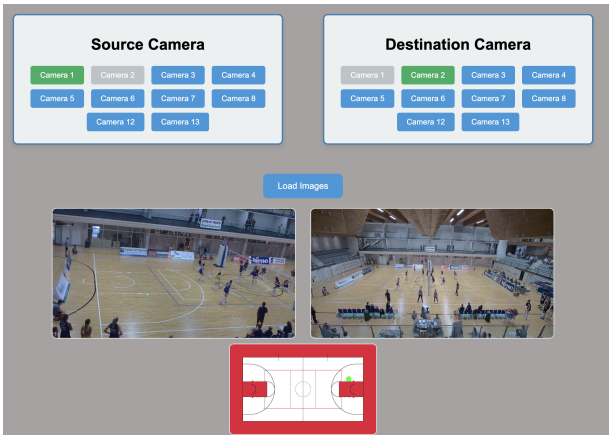

Figure 5: *GUI for point selection and projection.*

4. 3D Ball Tracking

The third part of our project consists in 3D ball tracking. The goal is to estimate ball's trajectory and drawing a 3D plot of it. In order to do that we start by taking 2D detections of the ball for each camera. After doing that we estimate the 3D world coordinates of the ball using triangulation. Lastly we've been

able to approximate the trajectory of the ball simply applying particle filter and some smoothing operations.

4.1. 2D detection

We began by reviewing state-of-the-art methods for reliably detecting our volleyball. After evaluating various options, we selected YOLOv11[7] as the best choice due to its speed and high accuracy for our task.
After choosing the model, we had to fine-tune it accordingly to our task. In order to do that we follow these steps:

1. We create our dataset extracting frames from the videos of different camera views, resulting in a collection of 1,000 frames;
2. We label the dataset using Roboflow[8];
3. To enhance dataset variety and diversity we enlarge it with applying different augmentations (flip, hue, brightness, blur, noise) obtaining a 2,400 images dataset;
4. We train the model on colab using 70% of entire dataset, while retaining 20% for validation and 10% for testing;

After training, we obtained updated weights for our YOLO model and tested them on real video footage, achieving high accuracy across most camera views. However, the model struggled with the second camera view due to its wider field of view, which makes it challenging to detect the small ball with high confidence.
To address this issue, we implemented a windowing approach (see Figure 6) by dividing each frame from this camera into a grid of 640×640 pixel cells and running inference on each window separately with the YOLO model. This method increases accuracy but requires more computational resources and time, as each frame now involves multiple inferences rather than a single pass. For this reason we decide to apply it just to this critical camera.

Figure 6: *Visual feed of our YOLO window approach.*

4.2. Triangulation Process

The triangulation process transforms 2D points detected in different camera views into a 3D coordinate system. Given two cameras and their corresponding 2D points, we extract the projection matrix P given extrinsic (rotation matrix R and translation vector t) and intrinsic parameters (camera matrix K):

$$P = K[R|t] \tag{1}$$

Given two points, $a_1 = (x_1, y_1)$ and $a_2 = (x_2, y_2)$, observed from two different camera perspectives and corresponding to the ball's position, our objective is to determine the 3D coordinates of the ball, $A = (X, Y, Z)$, such that the following projection equations are satisfied:

$$a_1 = P_1\, A \quad \text{and} \quad a_2 = P_2\, A \tag{2}$$

These calculations are performed using OpenCV's built-in function `cv2.triangulatePoints`[9].
For each frame, we reconstructed 3D points from different 2D camera views pair. These point's reconstructions, while they should correspond to the same 3D point, may vary slightly due to small errors. For this reason, we choose the point closest to the last detected point (of the previous frame). For the first frame, we use the average of the points, ignoring outliers, to get a stable starting point.

4.3. Particle Filter Tracking

To estimate the ball's trajectory, we explored various methodologies and ultimately chose to adopt a particle filter due to its ability to handle uncertainty in motion prediction and update the estimated position iteratively based on observed data. This approach is particularly advantageous in dynamic scenes where the ball's motion may be affected by noise or sudden changes in direction.
The particle filter is initialized with 1,000 particles, each representing a potential 3D position of the ball. Initially, these particles are distributed randomly around the first observed position of the ball (or an averaged initial detection if we obtain more then one reconstructed points). At each iteration, the particle filter performs a prediction step followed by an update step. First, we estimate the next state of the ball based on the motion model. Then, using the triangulated points as measurements, we correct this estimate to align with observed data. Finally, particles are resampled based on how closely they match the measurement, refining the trajectory estimation.
The ball's movement is quite unpredictable, following a nonlinear path with variable velocity and acceleration, however our method is quite flexible to adapt to these kind of scenarios.

4.4. Trajectory Smoothing

The trajectory obtained from the particle filter resembles a jagged line. To improve its visual quality and realism, we applied smoothing operations. First, we used spline interpolation, fitting a univariate spline to each coordinate axis to smooth the path. Next, we applied a moving average filter to further refine and stabilize the trajectory, enhancing both accuracy and visual appeal. Careful tuning of hyperparameters for these techniques was essential to achieve optimal smoothing of the trajectory.
In the final step, we visualize the 3D trajectory by plotting it alongside the field corners and the detected points (see Figure 7), providing a comprehensive view of the ball's path relative to its surroundings.

5. Conclusions and Future Improvements

This project successfully demonstrated a comprehensive 3D camera calibration pipeline for multi-camera setup, 3D reconstruction, and real-time ball tracking on a volleyball court. By combining intrinsic and extrinsic calibration, multi-camera mapping, and a robust ball tracking system using YOLOv11 and particle filtering, we created an effective solution for reconstructing and visualizing ball trajectories in 3D space. The results show promise for enhancing analysis in sports and similar applications. Future improvements could focus on optimizing detection accuracy in wider fields of view and expanding the system to support more complex motion patterns.
This project serves as a solid foundation for further development in computer vision applications for sports analysis.

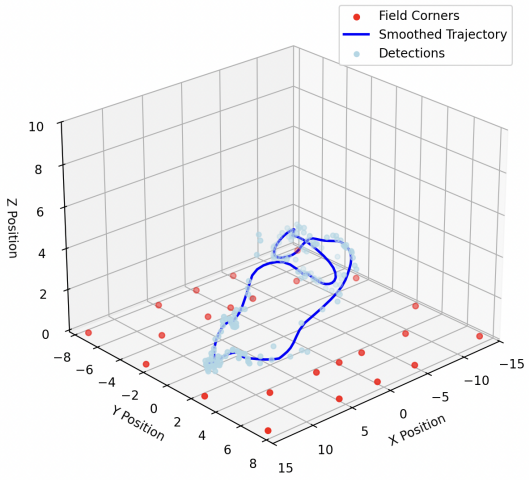

Figure 7: *3D tracking of the ball from two different point of view.*

6. References

[1] E. Hemayed, "A survey of camera self-calibration," in *Proceedings of the IEEE Conference on Advanced Video and Signal Based Surveillance, 2003.*, 2003, pp. 351–357.

[2] K. Liao, L. Nie, S. Huang, C. Lin, J. Zhang, Y. Zhao, M. Gabbouj, and D. Tao, "Deep learning for camera calibration and beyond: A survey," 2024. [Online]. Available: https://arxiv.org/abs/2303.10559

[3] R. Carroll, M. Agrawala, and A. Agarwala, "Optimizing content-preserving projections for wide-angle images," *ACM Trans. Graph.*, vol. 28, 07 2009.

[4] Z. Zhang, "A flexible new technique for camera calibration," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 11, pp. 1330–1334, 2000.

[5] OpenCV Team, "Camera calibration with opencv." [Online]. Available: https://docs.opencv.org/master/dc/dbb/tutorial_py_calibration.html

[6] ——, "Feature matching + homography to find objects." [Online]. Available: https://docs.opencv.org/3.4/d1/de0/tutorial_py_feature_homography.html

[7] G. Jocher and J. Qiu, "Ultralytics yolo11," 2024. [Online]. Available: https://github.com/ultralytics/ultralytics

[8] RoboFlow, Inc., "Roboflow: End-to-end computer vision pipeline and annotation tool," 2024, accessed: 2024-11-14. [Online]. Available: https://roboflow.com/

[9] *Triangulation Module — OpenCV Documentation*, OpenCV, 2024. [Online]. Available: https://docs.opencv.org/4.x/d0/dbd/group__triangulation.html