POLITECNICO
MILANO 1863

Ping Pong Ball 3D Trajectory

Image Analysis and Computer Vision

Forough Rajabi(10807901)

Academic year: 2022-23

Contents

Abstract

This report outlines the process of detecting and tracking a ping pong ball using multiple camera views. The project utilized background subtraction and morphological operations to isolate the ball in video frames, followed by corner detection techniques to approximate the ball's position. The goal was to create a robust system that tracks the ball in 2D and reconstructs its 3D trajectory. Video footage from multiple smartphones was processed using MATLAB's Computer Vision Toolbox, employing techniques such as foreground masking, Gaussian mixture modeling, and triangulation for ball localization. The results demonstrated that background subtraction, followed by corner feature detection, is an effective approach for ball tracking, although some challenges, such as noise and erroneous corner detection, remain. Further refinement using reprojection error thresholding and spatial constraints enabled the extraction of the ball's 3D trajectory, enhancing the accuracy of the tracking system.

1. Introduction

Tracking the position and motion of a fast-moving object like a ping pong ball is a challenging task, particularly in environments where the background and lighting conditions vary. This project explores the use of background subtraction and feature detection methods to address the problem of accurately detecting and tracking a ping pong ball across multiple video frames from different camera perspectives. The primary goal of this project is to identify the ball's location in 2D video footage, and subsequently, calculate its 3D trajectory using triangulation techniques.

To achieve this, multiple steps were taken, including camera calibration, background subtraction, morphological processing, and the application of corner detection algorithms. Videos from multiple smartphones were synchronized and processed in MATLAB, with the use of the ForegroundDetector and detectMinEigenFeatures functions from the Computer Vision Toolbox. Finally, triangulation was applied to estimate the 3D coordinates of the ball based on matching points from different camera views. This report outlines the implementation, results, and challenges faced in the development of the ball detection and tracking system.

2. Overview of Approach

This project is using background subtraction by generating a foreground mask (namely, a binary image containing the pixels belonging to moving objects in the scene). Then morphological operations are applied to clean up the mask, and corner detection is used to find potential ball positions. The strongest corners are selected to approximate the ball's location, and the position is overlaid on the original frame with a circle.

3. Implementation

3.1. Tools and Technologies:

The videos have been taken by 4 personal smartphones which only three of them were used for the processing step since they gave better results. The devices are as follows:

- Samsung A59 which is located in the top right corner with 1.5m height (wrt the defined origin in code scripts)
- Samsung A70 which is located in the bottom right corner with 1.4m height
- Iphon_x which is located on the right side of the table with 1.25m height.

All the videos have been processed in Matlab by useful computer vision toolbox functions such as, ForegroundDetector, detectMinEigenFeatures and, triangulateMultiview.

3.2. Data Acquisition:

The primary data for this ball tracking project consists of simultaneous video recordings from three cameras positioned around a ping pong table, each capturing the game from different angles. These videos are recorded in (720*1080) resolution at a rate of 30 frames per second. Additionally, four calibration images featuring a checkerboard pattern were taken from each camera in various orientations.

3.3. Preprocessing:

First of all, the videos were cut into 4 seconds duration to make it easy for the processing and all of them synced manually with Clipchapm. Then Camera parameters calculated by calibration and localization:

Calibration**:**

Camera calibration is the process of estimating camera parameters by using images that contain a calibration pattern. The parameters include camera intrinsics(focal length, principal point, skew, …), distortion coefficients, and camera extrinsics. Use these camera parameters to remove lens distortion effects from an image, measure planar objects, reconstruct 3-D scenes from multiple cameras, and perform other computer vision applications.

As discussed in the course, a simple calibration approach can be achieved using Zhang's method. By capturing multiple images of the same planar surface with a known geometry and the same camera, homographies can be computed between the points on the plane and their corresponding points in the images. These homographies are then applied to the circular points of the scene (denoted as I = [1,i,0] and J = [1,−i,0] to find the circular points in each image. From these points, a conic is fitted to obtain the Image of the Absolute Conic (IAC), and the camera's intrinsic parameters can be derived using Cholesky factorization.

Matlab's Camera Calibration Toolbox, which implements Zhang's method, to achieve more reliable results. For each camera, 4 images were captured of a black-and-white checkerboard pattern, and used the command detectCheckerboardPoints(img) to automatically detect the checkerboard points in several frames. After detecting the points, the function estimateCameraParameters(imagePoints, worldPoints) was applied to perform the calibration and retrieve the camera intrinsics.

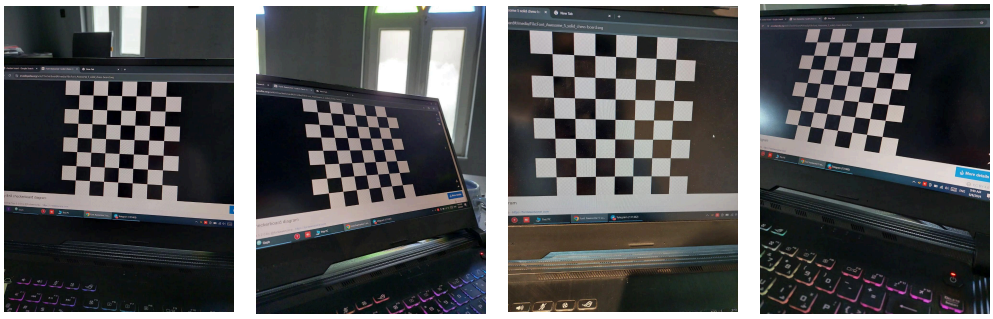


Figure1: checkerboard images taken by camera A59

3.3.1. Localization:

After calibrating the cameras, the next step is to determine their extrinsic parameters, the positions and orientations relative to the world reference frame. This is done by manually selecting a planar surface in an image from each camera and then fitting a homography to transform the surface into a new reference frame that I define.

A reference frame has been chosen with its origin at one of the corners of the ping pong table. The x-axis is directed along the short side of the table, the y-axis runs along the long side, and the z-axis points upward. I used the standard dimensions of a ping pong table.

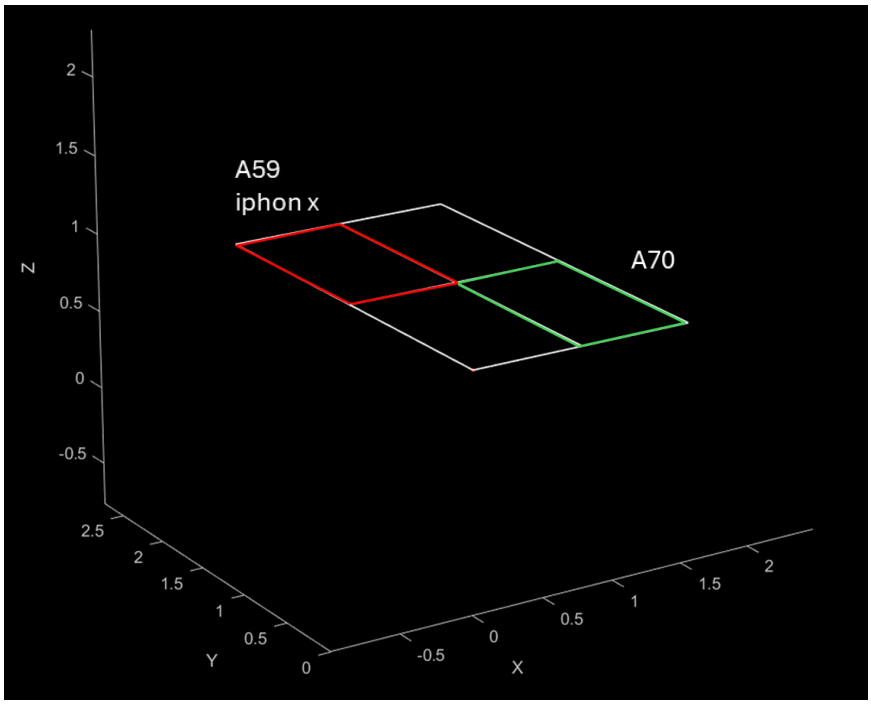


Figure2: Ping-pong table with standard size in the defined reference frame and planar surface in an image from each camera (red and green).

MATLAB's Camera Calibration Toolbox includes the extrinsics function, which computes the camera's rotation matrix and translation vector based on the selected points in the image, the known world points, and the intrinsic camera parameters. Also the camera's position and orientation vectors were calculated for easier plotting and visualization (**Figure11**).

```
%% gets 4 points (p1,...,p4) which are the matching points in the world refernce frame
hold on;
[x, y] = getpts();
hold off;

[rotationMatrix,translationVector] = extrinsics([x y], [P1; P2; P3; P4], parameters);

orientation = rotationMatrix';
location = -translationVector * orientation;
```

3.4. Processing and Results

3.4.1. Ball Detection:

There are several approaches to get the location of the balls in the video frames, the first approach I have tried was detecting the ball by filtering the color of the ball. However, due to the speed of the ball and the background color, it was not possible to detect the ball correctly.

Figure3: It is not possible to detect the ball by filtering the color

The second method which worked better is background subtraction, a straightforward method for background subtraction involves first estimating the background at time t. Then, subtract the estimated background from the current input frame. Finally, apply a threshold to the absolute difference obtained from the subtraction to generate the foreground mask. The first attempt was to apply Mean and Median Filtering that we use the average or the median of the previous N frames. The formula for the mean filtering is as follows:

$$Background\,(x, y, t) = \frac{1}{n} \sum_{i=1}^{n} Image(x, y, t - i)$$

Therefore , the foreground mask computed as follows:

Mean: $|Image\ (x, y, t)\ -\ \frac{1}{n}\ \sum_{i=1}^{n} Image(x, y, t\ -\ i)|\ >\ Threshold$

Median: $|Image\ (x, y, t)\ -\ median\{Image(x, y, t\ -\ i)\}|\ >\ Threshold$

$where\ i\ \in\ (1,\ \ldots,\ n)$

I used both for filtering however I faced many random noises.

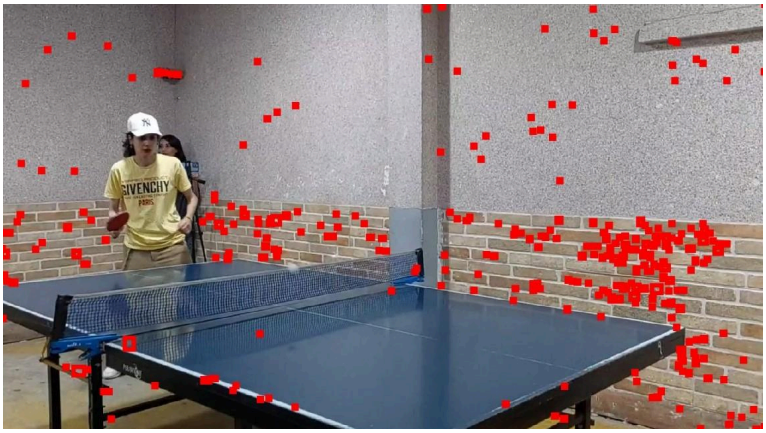


Figure4: using median filtering for background subtraction was unsuccessful for ball detection.

Another approach for the background subtraction which was successful for detecting the ball, is using the ForegroundDetector function in MATLAB's computer vision toolbox which computes and returns a foreground mask using the Gaussian mixture model (GMM). A Gaussian Mixture is a function that is composed of several Gaussians, each identified by k $\in$ {1,…, K}, where K is the number of clusters of our dataset. The main idea of this algorithm is that each background pixel is modeled with a mixture of Gaussians (K = 3 to 5) and its parameter is updated over time. If the pixel comes from one of those Gaussians, define it as background.

```
% Background subtractor
foregroundDetector = vision.ForegroundDetector('NumGaussians', 50, ...
                                               'LearningRate', 0.05);
```

It takes grayscale or truecolor (RGB) image as input (in my case is truecolor image which is frames of the video). The parameter ('NumGaussians', 50) specifies that the detector uses 50 Gaussian distributions to model the background, allowing it to handle complex backgrounds with more variability and the parameter ('LearningRate', 0.05) controls how quickly the model adapts to changes in the scene. A lower value makes the model learn more slowly, which is useful for gradual changes in the background. And at the end, returned as a binary mask.

Figure5: Applying ForegroundDetector. The white color in the image represents the ball while moving.

3.4.2. Ball Tracking:

After detecting the ball, it is possible to track the ball in the video frames and get the position of the ball in each frame. The method I have used is finding the corners in the video by using detectMinEigenFeatures function of MATLAB's computer vision toolbox. This function returns a cornerPoints object points that contains information about corner features detected in the 2-D grayscale or binary input using the minimum eigenvalue algorithm developed by Shi and Tomasi. It can help to identify the white color in the processed frames. Then I have selected the strongest (most prominent) corners from the detected ones which are representing the ball positions.However, before starting to find the corners, it is necessary to apply morphological 'opening' operation (erosion followed by dilation) to the binary mask of the frame using a structuring element ('se'). This operation helps to remove noise and small objects, improving the detection of the ball in the binary mask.

```
se = strel('square', 5); % Structuring element for morphological
%                                                        operations
% Morphological operation (Opening)
frame = imopen(mask, se);

% Parameters for corner detection
maxCorners = 2;
qualityLevel = 0.6;
minDistance = 25;
blockSize = 9;

% Convert frame to double and detect corners
frame_double = double(frame);
corners = detectMinEigenFeatures(frame_double, ...
                                 'MinQuality', qualityLevel, ...
                                 'FilterSize', blockSize);

% Filter the detected corners based on distance and quality
if ~isempty(corners) && corners.Count >= maxCorners
    strongestCorners = selectStrongest(corners, maxCorners);
    pos = strongestCorners.Location;
    x = pos(1, 1);
    y = pos(1, 2);
    % Store the location of the ball
    ballLocations = [ballLocations; x, y];
    oframe = insertShape(oframe, 'Circle', [x, y, 8], 'Color', ...
                                    [250, 0, 0], 'LineWidth', 2);

end
```

The frame is converted to a ‘double’ precision format to be used by the corner detection algorithm. The detectMinEigenFeatures function detects corners in the frame using the minimum eigenvalue method. It identifies strong corners (which are useful for tracking) based on the ‘MinQuality’ (corner quality threshold) and ‘FilterSize’ (block size for filtering). Tried to choose the best value for each parameter by applying some tests.
It is notable that although the ball has been detected and tracked, there are still some points detected wrongly, for example, some parts of the players are also being detected as strong corners. The solution that I have decided to apply is filtering the points by assigning a boundary for x, y coordinates and a threshold for the reprojecting error that I have explained in detail in section 3.4.3.

Figure6: Before applying the morphological operation

Figure7: After applying the morphological operation

Figure8: Selecting the strongest corner and adding circles to the original structures.

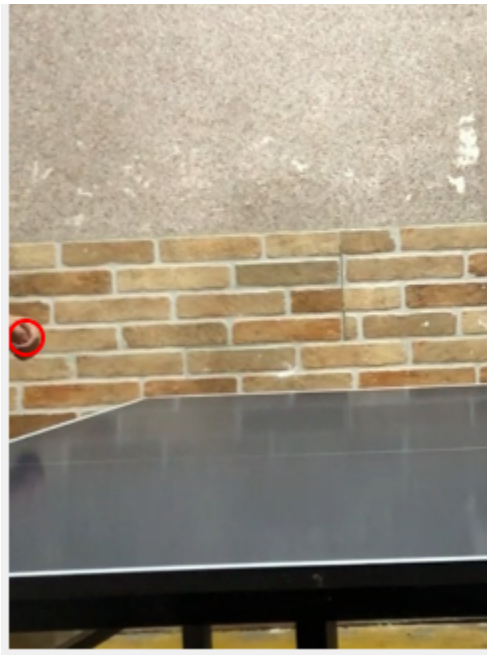


Figure9: wrong detection of the ball position.

3.4.3. Ball 3D trajectory:

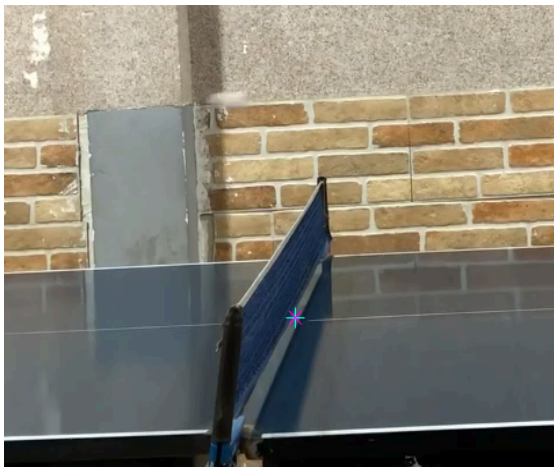
Ball trajectory in 3D space is possible by triangulation. The purpose of triangulation is to determine the position of points that are matched across multiple cameras in the world reference frame. In MATLAB, triangulation is carried out using the function triangulateMultiview. This function takes in a pointTrack object, which stores matching points from different camera views, a cameraPoses object that contains the camera positions and orientations, and the camera intrinsics. The function returns the triangulated point and the reprojection error.

```
%%Triangulating Point

track = pointTrack([1; 2; 3], [pts_a59; pts_a70; pts_iphonx]);
poses = table;
poses.ViewId = uint32([1; 2; 3]);
poses.Orientation = {params_a59.orientation;params_a70.orientation; ...
                                          params_iphonx.orientation};
poses.Location = {params_a59.location;params_a70.location; ...
                                       params_iphonx.location};
intrinsics = [params_a59.intrinsics,params_a70.intrinsics, ...
                                     params_iphonx.intrinsics];
[wp,re] = triangulateMultiview(track, poses, intrinsics);
```

(a)Point selected in a59 video frame

(b)Point selected in a70 video frame



(c) Point selected in iphon-x video frame

Figure10: (a),(b),(c) Matching points from different camera views

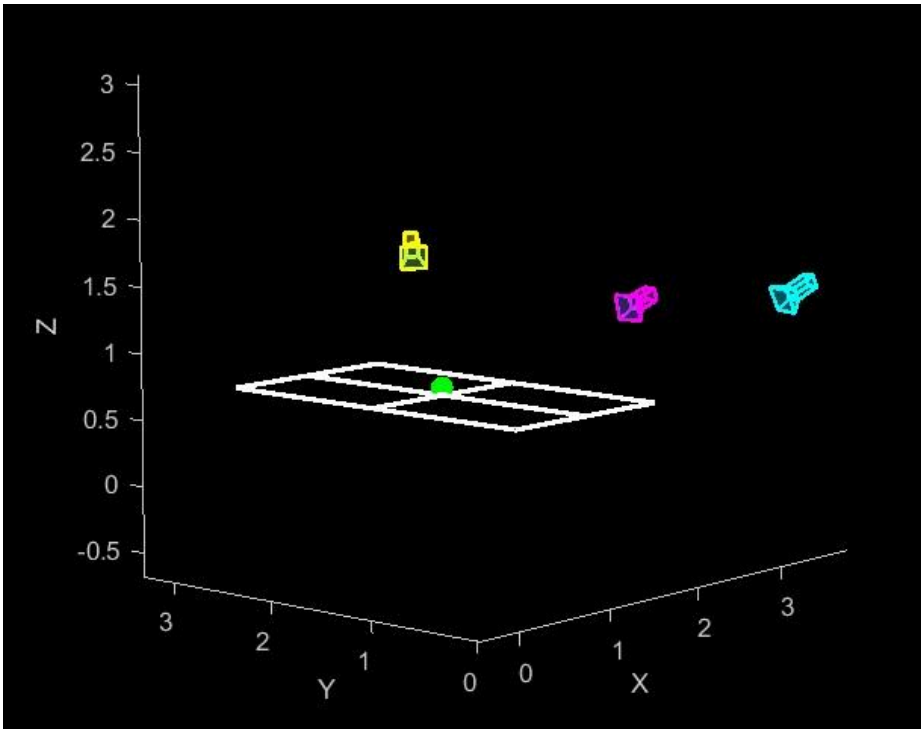


Figure11: The green spot is the triangulated point with reprojection error about 17.

Despite applying the method discussed in section 3.4.2, each frame may return several possible ball candidates. Therefore, it is necessary to iterate through all the candidates for each camera and eliminate the invalid ones. To filter out invalid points, the reprojection error is thresholded, and the x and y coordinates of the points are constrained to lie within the boundaries of the table.

```
wp_x_bound = [0, 1.525];
wp_y_bound = [0, 2.74];
re_thresh = 45;

frameNum = 1;
while frameNum <= 82 %with respect to the size of the min{BallLoc1,BallLoc2
                                                                % ,BallLoc3}


    BallLoc1 = BallLoc_a59.ballLocations(frameNum, :);
    BallLoc2 = BallLoc_a70.ballLocations(frameNum,:);
    BallLoc3 = BallLoc_iphonx.ballLocations(frameNum,:);

    % Create pointTrack objects for triangulation
   for b1 = 1:size(BallLoc1, 1)
        for b2 = size(BallLoc2, 1)
            for b3 = size(BallLoc3, 1)
                p1 = BallLoc1(b1, :);
                p2 = BallLoc2(b2, :);
                p3 = BallLoc3(b3, :);

                track = pointTrack([1; 2; 3], [p1; p2; p3]);
                [wp, re] = triangulateMultiview(track, poses, intrinsics);

                % Plot the triangulated points
               if re < re_thresh && wp(1) >= wp_x_bound(1) && ...
                                    wp(1) <= wp_x_bound(2) && ...
                                    wp(2) >= wp_y_bound(1) && ...
                                    wp(2) <= wp_y_bound(2)

                    plot3(wp(1), wp(2), wp(3),'.', 'MarkerSize', 10, 'Color'
                                                                    , 'red');
                    hold on;
                end
```

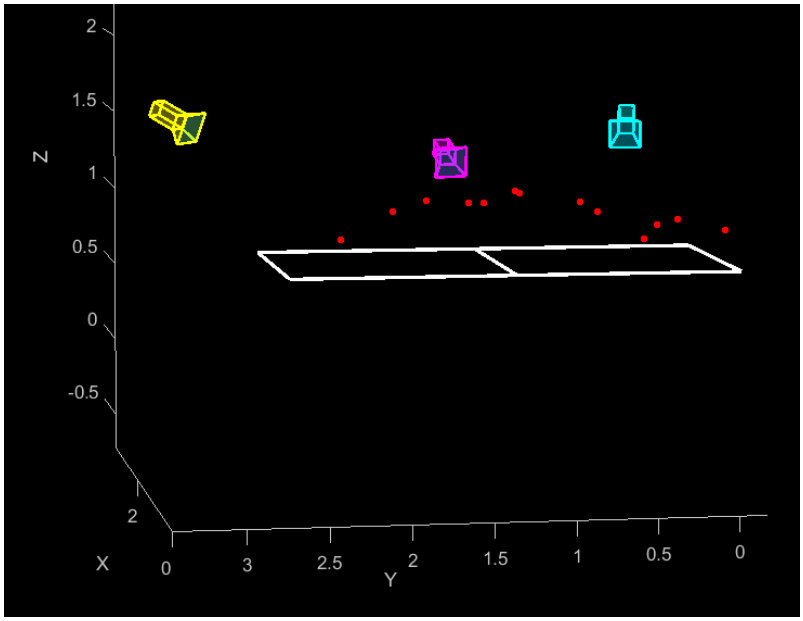


Figure12: Ball 3D trajectory by applying the filters

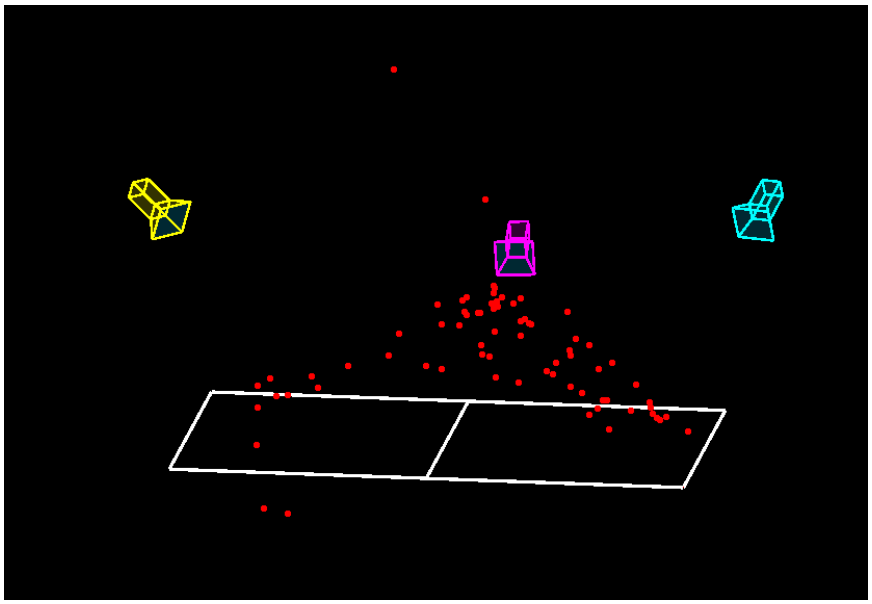


Figure13: Ball 3D trajectory without filter

4. Conclusion

This project aimed to implement a system to detect and track a ping pong ball in 2D and reconstruct its 3D trajectory using video footage from multiple camera views. Background subtraction and corner detection provided an effective approach for isolating the ball, while triangulation yielded 3D coordinates. Despite the overall success, some challenges were encountered, such as noise from lighting variations, false detections, and difficulty in handling occlusions.

5. Bibliography

[1] Course 'Image analysis and computer vision' , Prof. Vincenzo Caglioti and Prof. Giacomo Boracchi
[2] MATLAB Documentation
[3] https://towardsdatascience.com/gaussian-mixture-models-explained-6986aaf5a95
[4] http://personal.ee.surrey.ac.uk/Personal/R.Bowden/publications/avbs01/avbs01.pdf