POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Tennis Ball Detection and Tracking

Image Analysis and Computer Vision Project

Christian Grasso (10652464)

Filippo Lazzati (10629918)

Academic Year: 2021-22

Contents

List of Figures

1 | Introduction

People love doing sport, as well as watching it. During the last years and thanks to the advancement in technology, new ways of watching, analyzing and gathering statistics from a sport event arose. However, the use of technology and in particular of *computer vision* techniques entered actively in the sport process; in fact, cameras are not used anymore only by fans to watch a match or by commentators to provide aggregates on the match (e.g., the distance travelled by each football player during the match), but they are part of the match. Some examples of this are the **Goal-line technology** adopted in football to check whether a ball entered or not the football goal or the system adopted in athletics to verify who is the runner that gets past the finish line first.
In this context, **Hawk-Eye** [21] borned, initially for television purposes, but then in 2006 it has been adopted. Hawk-Eye is a computer vision system used in numerous sports such as cricket, tennis, Gaelic football, badminton, hurling, rugby union, association football and volleyball, to visually track the trajectory of the ball and display a profile of its statistically most likely path as a moving image.[1] In other words, it is a system of cameras that automatically detects and tracks a ball (and its 3D position) during the match, with the aim of preventing human errors in line-calling (the act of deciding whether the ball landed inside or outside the court baseline). Up to the 2021, such system was only a form of support for the judges (it was consulted a limited number of times during a match), but then the pandemic allowed it to become the only judge in the tennis court.

[1] `https://en.wikipedia.org/wiki/Hawk-Eye#Tennis`

2 | Problem formulation

The goal of this project is to realize a computer vision system that is able to automatically detect and track a tennis ball from videos recorded from multiple cameras, and to analyze the trajectory. The objectives can be summed up in the following list:

- detect a tennis ball in a video;
- triangulate the tennis ball from multiple views;
- track a tennis ball during the video;
- compute the ball velocity;
- understand whether the ball landed inside or outside the court.

3 | Report overview

The organization of this report is the following:

1. chapter 4 shows an overview of the current techniques adopted in literature and which of them have inspired our work;
2. chapter 5 shows briefly the ideas and techniques behind our implementation;
3. chapter 6 describes in detail the techniques adopted as well as the `Matlab` code;
4. chapter 7 presents an example of the pipeline of execution;
5. chapter 8 presents and provides some comments about the results we have obtained;
6. chapter 9 sums up everything done in the project;
7. chapter 10 presents some ideas for possible future works.

4 | Related work

In literature we can find many works about tennis (and sport in general), which are focused mainly on tennis ball detection and tracking. However, it should be noticed that papers with better results are commercial documents, and as such they provide just a very high-level description of the system, without entering the details of the implementation. A brief overview of the main works in literature is presented below.

In [1] the authors propose a solution for processing recorded stereo smartphone videos exploiting multiple-view geometry and state estimation filtering. It is interesting the use of morphological operators to find the position of the tennis ball. In [2] a more precise and complex algorithm to determine the ball position is presented, leveraging the **Support Vector Machine** algorithm to identify the ball from the features; however, the most interesting aspect presented in this thesis is the particle filter implementation adopted for tracking the ball. Paper [3] proposes a graph-theoretic formulation for tracking the tennis ball in monocular sequences. Paper [4], even though it is not peer-reviewed, proposes a nice algorithm for court line extraction exploiting the **top-hat transform**; we exploit some ideas present in this paper to help us improve our results. Paper [5] shows how the **particle filters** can be used for coping with environment changes. Paper [6] is one of the most important papers about foreground detection; it presents a method based on a mixture of Gaussians to cope with lighting changes and others in the environment and thus improving the foreground detection. This is also the algorithm implemented by *Matlab* in

`vision.ForegroundDetector`

along with the ideas proposed in [7].

The papers presented so far are just some of the papers we read to devise our own way to solve this problem. However, the paper that has influenced us the most is [8], in which a new low-cost computer vision system is proposed.

5 | Our solution approach

Our approach to solve this task spans from the hardware infrastructure to record the videos to the software implementation. We believe that a list is well-suited present our approach:

1. we have decided to use three smartphones placed around a tennis court to record the videos to use to implement and test our code;
2. before exploiting the recordings, an important step is to calibrate the cameras (without the intrinsics parameters, camera localization and triangulation tasks cannot be performed); so we recorded some other videos of a checkerboard through the smartphones to use the `Matlab` *Calibration Toolbox* to calibrate the cameras (and remove distortions, if any);
3. to localize the cameras around the court, we have selected manually some points of the court and then exploited the `Matlab` *Calibration Toolbox* (we have decided to use the `Matlab` implementation and to avoid the manual implementation used in the homework because `Matlab` code is optimized and gives better results, which are essential for the following steps);
4. next, to synchronize the recordings of the various cameras, we have decided to trust the audio of videos: we compute the *cross-correlation* of the various audio signals (taken from the videos) and extract the time lag (shift) that gives the highest correlation; however, this value has been just used as starting point for a manual synchronization, since the sound speed ($343m/s$) cannot be ruled out in this context (we record videos at 60fps, so every $1/60 \approx 17ms$ a new frame is taken, but in $17ms$ the sound of the ball might not have reached all the cameras, since it takes $20m/343m/s \approx 60ms$ to travel $20m$);
5. the detection of the ball inside a frame is the most difficult task; in fact, as explained for instance in [2], a color mask cannot be applied since the yellow of the ball is lost and the ball (small object moving quickly) takes the color of the background; so we decided to combine *background subtraction* (made using a median filter on

the first N frames of the videos) and *frame differencing* (with the past $4th$, $6th$ and $8th$ frames) to obtain a decent mask, that is then additionally cleaned through some morphological operators (in particular area opening, image closure and image dilation);

6. it should be remarked that the tracking of the ball has been performed on the 3D position of the ball, and not on the position in each video separately[1]; so, to perform *triangulation*, we compute all the possible combinations of all the possible candidates in all the possible videos (example: with 3 candidates for video, since we have 3 videos, the number of candidates becomes $3^3 = 27$; clearly, this slows down the computation) by looping and calling the `Matlab` command **triangulateMultiview**, which, as written in the documentation, is based on the method described in [20], so basically the idea is to find the camera matrices and then minimize a suitable loss function (since, because of noise, naive triangulation by intersecting back-projected rays will fail) or by exploiting an optimal MLE;

7. to track the position of the ball in 3D, we have decided to implement a *Kalman filter* with unknown correspondences, where "unknown correspondence means that in performing the Kalman filter update step, the exact measurement corresponding to the state that the filter is trying to estimate is unknown"[2]. The physics model adopted is the usual uniformly accelerated motion model (so notice that we have decided to neglect the drag and lift coefficients as described in [12]);

8. to solve the problem of *data association* (how to assign the unknown correspondences), we use 3 types of "data cleaning operations":

   (a) first, we delete all the candidates outside the court or too high;

   (b) then, we apply the **RANSAC** algorithm to the last N 3D positions (in the last N time instants) of the tennis ball (we fit a plane in the 3D space), and we keep (at current time instant) only the candidate closest to such plane;

   (c) we then keep or discard such candidate according to whether it lies inside a parallelepiped (not a sphere, because computing the Euclidean distance is more expensive) centered in the Kalman prediction (where the ball should be) and with sides proportional to the variance of the prediction (so if the the prediction

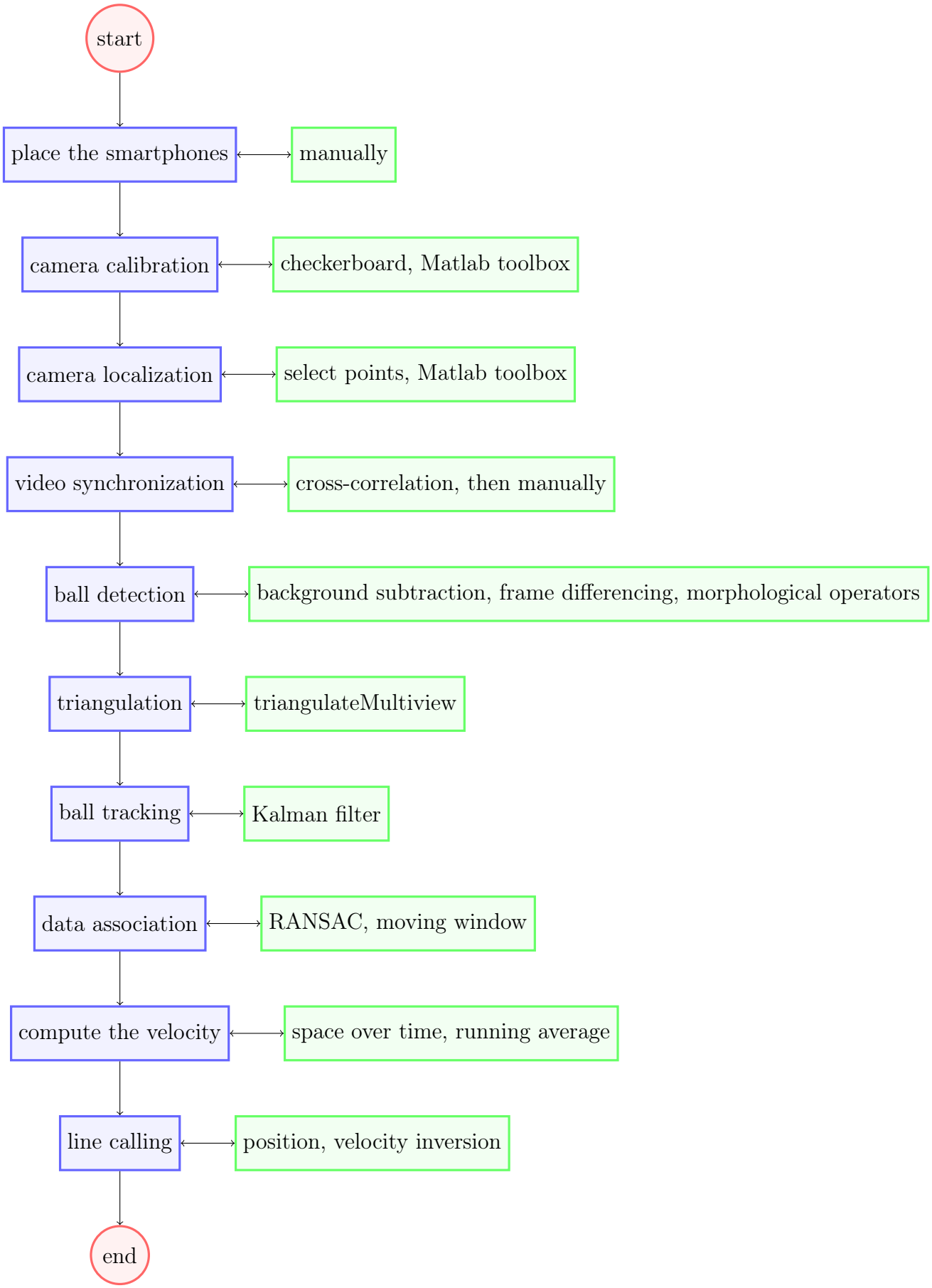
[1] The rationale behind this decision is that, as previously mentioned, with low-quality cameras the detection of the ball is the most difficult task, and it provides also many outliers; to get rid of them, data association techniques are used, but they might not be enough; instead, triangulate the ball with the other two guesses and keeping only positions under a certain threshold of the reprojection error, can reduce a lot the number of outliers.

[2] see paper [1], paragraph H.

is more uncertain, such window is bigger).

9. to compute the velocity, we use a running average among the velocity of the ball in the past time instants (the idea of the running average is to avoid to keep in memory all the past velocity measurements) computed simply by $(x_i - x_{i-1})/\Delta t$, where Δt is the time between two frames;
10. finally, to check whether the ball landed inside or outside, we can check when the z coordinate is below a certain threshold or the v_z (velocity along vertical axis) inverts its sign.

To sum up the approach, one can refer to the following scheme:


start
place the smartphones
manually
camera calibration
checkerboard, Matlab toolbox
camera localization
select points, Matlab toolbox
video synchronization
cross-correlation, then manually
ball detection
background subtraction, frame differencing, morphological operators
triangulation
triangulateMultiview
ball tracking
Kalman filter
data association
RANSAC, moving window
compute the velocity
space over time, running average
line calling
position, velocity inversion
end

6 | Implementation

In this chapter, the steps listed in 5 are explained and seen in detail.

6.1. Hardware infrastructure and video recording

Some videos have been recorded through 3 personal smartphones to have some data on which work. The material has been recorded at Tennis Club Romano of Romano di Lombardia on court 2. The videos show two players playing on a clay court in indoor conditions.
The devices used for recording the videos are:

- *Samsung* Galaxy S20 FE, which has been placed atop a pole for keeping the volleyball net at more than $2.5m$;
- *Samsung* Galaxy S20 +, which has been placed on top of the judge chair, at more than $2m$ of height;
- *Samsung* Note 8, which has been placed atop a pole for keeping the volleyball net at more than $2m$ (lower than the *Samsung* Galaxy S20 FE).

The configuration of the smartphones can be seen in 6.1:

(a) S20FE (b) S20+

(c) NOTE8

Figure 6.1: The support for all the devices.

Instead, a panoramic of the scene can be seen in figure 6.2:

Figure 6.2: A panoramic of the scene.

All the videos have been recorded at a frame rate of $60fps$; moreover, note8 and s20fe[1] have a quality of $1080x1920$ pixels while the s20plus has $1920x1080$ pixels.

6.2. Camera calibration

The goal of *camera calibration* is to retrieve the **intrinsics** of the camera, the parameters that depend only on the camera and not on its position with respect to the world reference frame (that is, the focal length, the principal point position and the skew parameter).

As seen during the course, a simple camera calibration device can be realized exploiting the idea of Zhang in [10]: if we take multiple photos of the same plane through the same camera, and we know the true planar scene, then we can fit homographies between points of the plane scene and points of each of the photos. We can then apply such homographies to the circular points of the true scene, that is $\mathbf{I} = [1, i, 0]$ and $\mathbf{J} = [1, -i, 0]$, to obtain the circular points of the various images. Finally, we can fit a conic to all these new circular points to obtain the Image of the Absolute Conic, from which we can retrieve the intrinsics by Cholesky factorisation.

Since this phase is rather critic[2] (triangulation is based on this), we have decided to use the `Matlab` Camera calibration toolbox implementation of the Zhang's method to retrieve better results. For each camera, we have recorded a video of a black and white chessboard, and then through the command:

```
detectCheckerboardPoints(img)
```

[1] From now on, we will refer to the three smartphones as note8, s20fe and s20plus.
[2] As we have seen by doing the homework, due to noise, it is easy to obtain numerical errors.

we have automatically detected (the algorithm adopted by `Matlab` is the one in [11]) the checkerboard's points in multiple frames drawn from the video. Then we have called the function:

`estimateCameraParameters(imagePoints,worldPoints)`

to apply the Zhang's method and retrieve the `intrinsics` of both the cameras.

6.3. Camera localization

After calibrating the cameras, we need to compute the **extrinsics** of the cameras (positions and orientations relative to the world reference frame). This can be accomplished by manually selecting a planar surface in an image from each camera, and then fitting an homography that transforms the surface to a new reference frame defined by us.

The frame of reference we have chosen has its origin in one of the corners of the tennis court, with the x axis positive towards the short side of the court, the y axis positive towards the long side of the court and z pointing upwards, as shown in 6.3. In the same figure, the surfaces used for localization are also highlighted; we have used the standard size of a tennis court to obtain the coordinates of the world points.

The MATLAB Camera Calibration toolbox already includes the `extrinsics` function which returns the camera rotation matrix and translation vector from the selected points in the image, the world points and the camera parameters. We also calculate the location and orientation vectors for easier plotting.

```
[x, y] = getpts();
% P1..4 are the matching points in the world reference frame
[rotationMatrix, translationVector] = extrinsics([x y], ...
    [P1; P2; P3; P4], cameraParams);
orientation = rotationMatrix';
location = -translationVector * orientation;
```

The results of the localization, shown in 6.3, are fairly accurate and reflect the real position of the cameras.

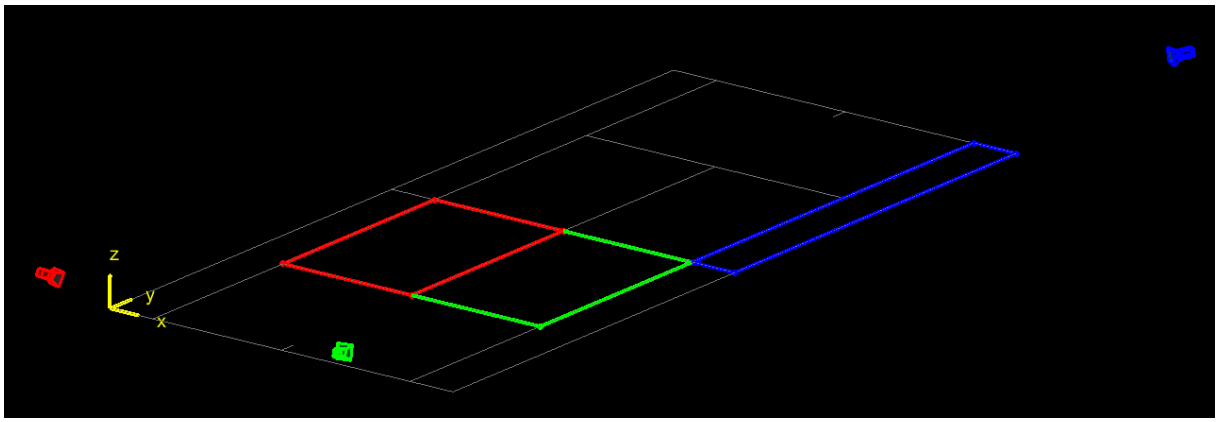


Figure 6.3: Camera localization results, with the frame of reference and the planes used for the localization of each camera

6.4. Court segmentation

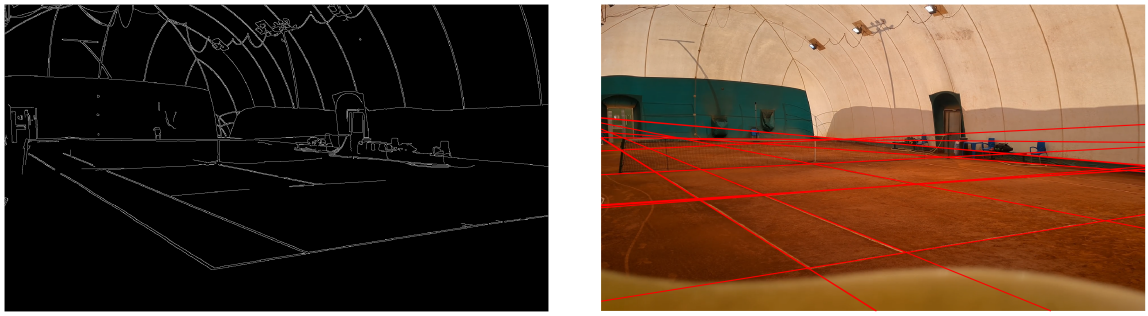
In order to automate the camera localization step, and to try obtaining more accurate results for the camera extrinsics, we have tried to implement edge detection for the tennis court; however, as you can see since this step is missing in chapter 5, we have not been able to provide a useful court segmentation code[3]

Starting with a still image of the field with no players[4], **Canny edge detector** algorithm is applied using the `edge` MATLAB function (6.4a). We can use the results to detect straight lines by applying the **Hough transform**. A Hough transform of a datum (in this case a point on a detected edge) is a set of models compatible with the datum in the *model space*. For a point in the cartesian plane, the HT will be a sinusoid; collinear points will map to (almost) concurrent HTs. Finally, we detect the intersections between the detected edges to find the corners of a planar surface to be used for localization.

```
[H,T,R] = hough(edges);
% Keep the top 8 peaks and limit to top 30% results
P = houghpeaks(H, 8, 'threshold', .3*max(H(:)));
hlines = houghlines(edges, T, R, P, 'FillGap', 20, 'MinLength', 100);
```

[3]All the papers read about this task employed videos in which the colour of the court inside the baselines is different from the colour of the court outside the baseline; clearly, being our court a clay court, this approach is not feasible.

[4]In our case this is actually computed using a median filter, as explained in 6.6.1.



(a) Canny edge detection results (b) Detected court edges

Figure 6.4: Court segmentation results

In the end, we have decided not to use the results of court segmentation for localization, as they did not improve precision over the manually selected points and it is still needed to manually enter the corresponding points in the world reference frame.

6.5. Video Synchronization

One of the main problems we faced is synchronizing the three videos. Since the cameras are not connected, starting the recording at the same time instant is not a trivial task. In order to do this, we tried to produce a sound (clapping) at equal distance among the cameras, and then synchronizing the videos using this sound; however, this approach did not yield frame-perfect results, which is required to achieve triangulation. Therefore, we opted for a "semi-automatic" approach: we first compute the cross correlation[5] between the audio signals of two videos, then we take the lag for which there occurs the maximum correlation, and finally transform such time lag into number of frames lag; the second step is to manually synchronize the videos starting from this time lag. The code for synchronizing, for instance, S20FE and Note8 is:

```
%% synchronize s20fe and note8
% read audio and video of s20fe
[y_s20fe, sample_rate_s20fe] = audioread('videos/s20fe.mp4');
info_audio_s20fe = audioinfo('videos/s20fe.mp4');
v_s20fe = VideoReader('videos/s20fe.mp4');
frames_s20fe = read(v_s20fe,[1 Inf]);
info_video_s20fe = info(vision.VideoFileReader('videos/s20fe.mp4'));
```

[5]Cross-correlation measures the similarity between a vector x and shifted (lagged) copies of a vector y as a function of the lag. See `https://ch.mathworks.com/help/matlab/ref/xcorr.html`.

```
% read audio and video of note8
[y_note8, sample_rate_note8] = audioread('videos/note8.mp4');
info_audio_note8 = audioinfo('videos/note8.mp4');
v_note8 = VideoReader('videos/note8.mp4');
frames_note8 = read(v_note8,[1 Inf]); % 1280x720x3x292
info_video_note8 = info(vision.VideoFileReader('videos/note8.mp4'));
```

Then take the sample rate and the frame rate (will be used to convert time lag into "frame lag"):

```
sample_rate = sample_rate_s20fe; % same as note8
frame_rate = info_video_s20fe.VideoFrameRate; % same as note8
```

A plot of the two signals is here reported (on the x axis there are the seconds):

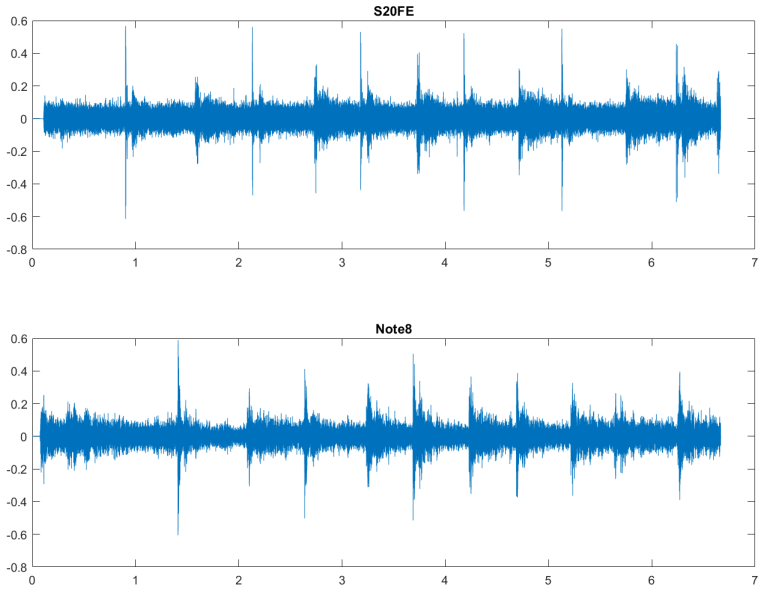


Figure 6.5: The audio trace of the videos recorded by the s20fe and the note8. The alternating amplitudes in the intensity peaks are due to the different distances of the cameras from the player hitting the ball.

Next, we can compute the cross-correlation between the two signals and take the time lag for which the correlation is maximized:

```
signal_s20fe = y_s20fe(:, 1);
signal_note8 = y_note8(:, 1);
[c,lags] = xcorr(signal_s20fe,signal_note8);
```

```
% take the maximum
[max_lag, index_max_lag] = max(abs(c));
```

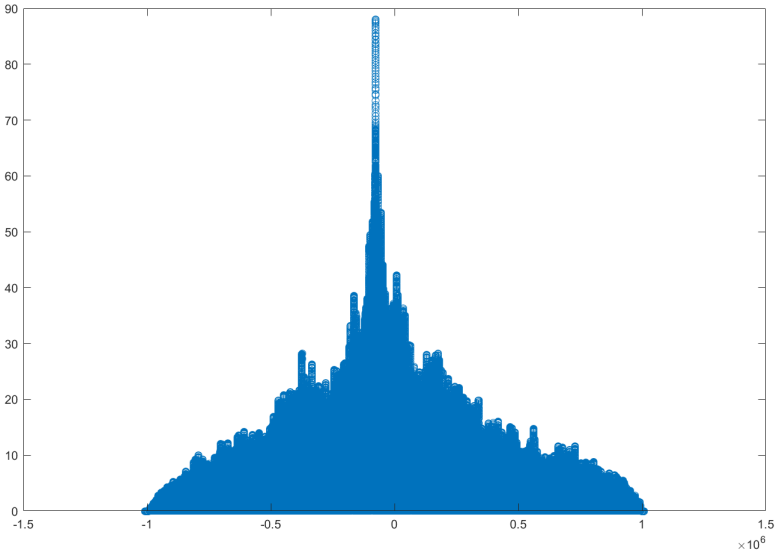


Figure 6.6: A plot of the xcorr of the two audio signals; on the x axis there are the lags.

And finally convert the time lag into a "frame lag":

```
shift_between_signals = lags(index_max_lag);
shift_in_seconds = shift_between_signals / sample_rate;
shift_in_frames = round(frame_rate * shift_in_seconds);
```

Finally, starting from `shift_in_frames`, we have manually synchronized the videos.

6.6. Ball candidate detection

In literature, we have found a huge number of image processing techniques to perform the detection of the tennis ball in a video. However, many of them are not useful for our project because of the low quality of the cameras.
For example, the idea of applying a color filter (a yellow filter) to the image to retrieve only the objects with the same color of the ball does not work, because the ball is moving at high velocity and undergoes deformation; this makes its color to be mixed with that of the background, making this approach not suitable. You can notice this in figure 6.7:

Figure 6.7: The ball cannot be detected by its color because it is a small object that moves fast.

As previously mentioned, our ball candidate detection algorithm exploits background subtraction, frame differencing and morphological operators.

6.6.1. Background subtraction

The idea of background subtraction is very simple: to keep in memory a model (image) of the background (static objects), and every time we need to detect a foreground object, simply compute the image difference between the current image and the background (element-wise difference). There are many different approaches to do this, like computing an average filter with the first N frames of the video or exploiting a mixture of Gaussian models to model every single pixel of the image during time (this goes under the name of adaptive background mixture models [6]). We have decided to adopt a **median filter** of the first N frames to model the background. This because the results we obtain are more customizable than those obtained using the `vision.ForegroundDetector` object of `Matlab`, which implements the idea of [6]. This method is clearly better than adopting a **median filter**, since, as Grimson and Stauffer explain, using adaptive mixture of Gaussians (i.e. a linear combination of multivariate Gaussians distributions) for every pixel that change over time, allows to cope with the problem of changes in lighting and shadow. However, given the limited application of our project (it has to work properly just on the videos we recorded, which are of a tennis court under a cover and therefore no changes in illumination occur), a **median filter** works fine.

6.6.2. Frame differencing

The **median filter** for background subtraction is then combined with **frame differencing** to filter out foreground objects (like the players) that move during time but not fast enough (like the tennis ball). We apply **frame differencing** in a window with the past $4th$, $6th$ and $8th$ frames to improve the detection. We have also tried to *deinterlace* the videos before performing background subtraction and **frame differencing**, as suggested in [2], but we have not found huge improvements, thus we have removed it.

6.6.3. Morphological operators

Even after **background subtraction** and **frame differencing**, a lot of wrong ball candidates are detected, because of noise or because of certain parts of the player that move fast. Thus, the application of morphological operators can rule out many wrong candidates, despite some always remain. According to the definition provided by Gonzalez and Woods in [22], "mathematical morphology" can be seen "as a tool for extracting image components that are useful in the representation and description of region shape, such as boundaries, skeletons, and the convex hull". At a low-level, morphological operators are simple pixel-wise operations on binary images, like, for instance, logic between a foreground set and a structuring element, repeated for every piece of image; however, this simplicity brings at great high-level results, like eroding or dilating a foreground object, or opening or closing it, or even matching some basic patterns (e.g. hit-miss transform, that we tried to use to match the ball size, but then removed it).

6.6.4. Ball candidate detection implementation

The code of our implementation is shown and explained below (notice that our implementation works with grayscale images, so that it is less computationally expensive). It should be remarked that our implementation works on binary masks.
First of all, we compute a model of the background by computing the **median** of 200 frames of the video:

```
frames1 = read(v1, [200 400]);
background1_bw = rgb2gray(median(frames1, 4));
```

Then we compute the background subtraction mask by taking the absolute difference between frame and background, applying the *image close* morphological operator to "close" the ball pixels in a single spot, then applying a threshold for removing the noise and applying the complement of morphological `bwareaopen` to remove blobs greater than a

certain size (basically, to remove the players):

```
% Background subtraction
bw_bg_sub = imclose(imabsdiff(frame_bw, background_bw), strel('disk', 5));
mask_bg_sub = bw_bg_sub > mask_thresh;
mask_bg_sub = mask_bg_sub & ~bwareaopen(mask_bg_sub, 200);
```

The next step is to apply frame differencing, and this is done by combining computing three differences: the current frame with the one of 4 frames ago, the current with that of 6 frames ago and the current with that of 8 frames ago. Then, similarly to background subtraction, `imclose`, a threshold and then the complement of `bwareaopen` are applied to clean the mask, and then the mask is given in AND to the others:

```
% Frame differencing
mask_frame_diff = ones(size(frame_bw, 1), size(frame_bw, 2));
for df = [8, 6, 4]
    % the past frame is stored in a circular buffer
    frame_idx = mod(index0 - df, size(frame_buffer, 1)) + 1;
    other_frame_bw = frame_buffer{frame_idx, video_idx};

    bw_frame_diff = imclose(imabsdiff(frame_bw, other_frame_bw), ...
        strel('disk', 3));
    mask_other_frame_diff = bw_frame_diff > mask_thresh;
    mask_other_frame_diff = mask_other_frame_diff & ...
        ~bwareaopen(mask_other_frame_diff, 200);
    mask_frame_diff = mask_frame_diff & mask_other_frame_diff;
end
```

The final mask is obtained by putting in AND the background subtraction mask with the frame difference mask, and then applying `imdilate` to enlarge the candidates:

```
    % AND masks and return region properties
    mask_and = mask_bg_sub & mask_frame_diff;
    mask_final = imdilate(mask_and, strel('disk', 2));

    props = regionprops(mask_final, 'BoundingBox', 'Centroid');
    bbx = vertcat(props.BoundingBox);
    cc = vertcat(props.Centroid);
```

In 6.8 the four different masks computed for detecting the ball are shown:

Figure 6.8: From left to right (the current frame and 4 masks): the current frame with all the detections enlighted in red; the mask obtained with background subtraction (mask_bg_sub); the mask obtained by the AND of the frame differencing masks (mask_frame_diff); the mask obtained by the AND of the frame difference mask and that of background subtraction (mask_and); the final mask obtained by dilating the previous one (mask_final).

6.7. Triangulation

The goal of triangulation is to find the location of points matched across the cameras in the world reference frame.

As mentioned, even after applying the techniques outlined in 6.6, multiple candidates will be returned for each frame. Thus, we need to iterate over all the candidates for each camera, and filter out the invalid ones.

Triangulation is performed using the MATLAB function `triangulateMultiview`, which implements the algorithm described in [20]. The function takes as input a `pointTrack` object (an appropriate data structure to hold matching points from multiple views), a `cameraPoses` object built using the camera locations and orientation, and the camera intrinsics. The returned values are the triangulated point and the reprojection error.

The reprojection error `re` is thresholded to exclude points which are probably invalid, along with the `x` and `y` coordinates of the point which are bounded to be inside of the court and the area visible by all cameras.

```
re_thresh = 25;
wp_bounds_x = [-5, 15.97]; % Court width +/- 5m
wp_bounds_y = [0, 23.58]; % Court length
```

```
for c1 = 1:size(centroids1, 1)
  for c2 = 1:size(centroids2, 1)
    for c3 = 1:size(centroids3, 1)
      p1 = centroids1(c1, :);
      p2 = centroids2(c2, :);
      p3 = centroids3(c3, :);

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

      % The "in" fucntion checks if the value is in the specified range
      if re < re_thresh && in(wp(1), wp_bounds_x) && in(wp(2), wp_bounds_y)
        triangulated_pts_count = triangulated_pts_count + 1;
        points_buffer{point_buffer_idx}(triangulated_pts_count, :) = wp;
      end
    end
  end
end
```

6.8. Data association

According to [19], "Data Association (DA) is a process of associating uncertain measurements (ball position in image obtained after detection) to known tracks. It requires detected ball candidates as input"; moreover "in presence of noisy detections, the problem of false positives and false negatives makes ball tracking using DA, a computationally complex process in complex environments".
Our idea to extract at most 1 candidate from the set of candidates, which draws inspiration from [9], is based on **RANSAC**[6]: we fit a (vertical) plane to the set of 3D points made up of the ball candidates in the current frame and the last 10 frames, and we only keep the inliers, so that candidates far away from the plane (and thus from the parabolic trajectory) are discarded:

```
% Fit plane to points using RANSAC and keep inliers
[plane, inlierIdx] = pcfitplane(pointCloud(all_pts(:, 1:3)), ...
    0.05, [1 0 0], 40);
inliers = all_pts(inlierIdx, :);
```

[6] "Random sample consensus (RANSAC) is an iterative method to estimate parameters of a mathematical model from a set of observed data that contains outliers, when outliers are to be accorded no influence on the values of the estimates." see https://en.wikipedia.org/wiki/Random_sample_consensus

we then keep only the candidate closest to the fitted plane:

```
min_dist_to_pred = Inf;
selected_point = nan(3, 1);

% Find the point in the current frame with the lowest distance
% from the fitted plane
for p = 1:size(inliers, 1)
    if inliers(p, 4) == point_buffer_idx
        % Point is from current frame
        dist_to_pred = norm(inliers(p, 1:3) - mu_k(1:3).');
        if dist_to_pred < min_dist_to_pred
            selected_point = inliers(p, 1:3);
            min_dist_to_pred = dist_to_pred;
        end
    end
end
```

How RANSAC is used is shown in figure 6.9:

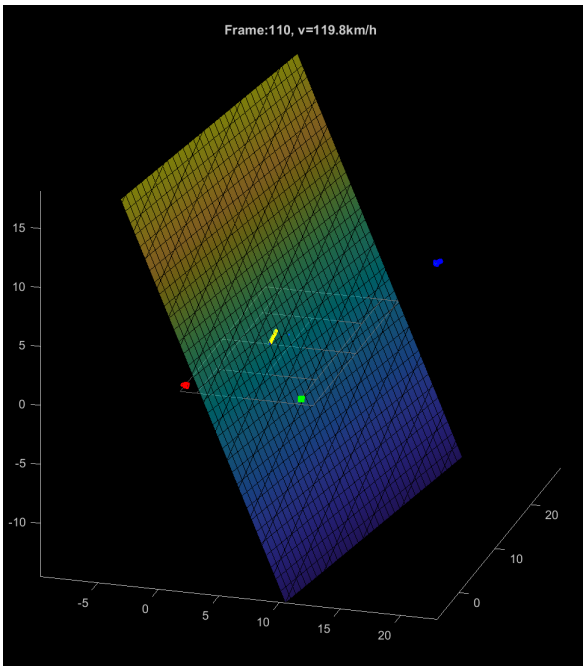


Figure 6.9: The plane is fitted using the past positions of the ball shown in yellow in the plot.

Finally, to avoid using wrong predictions when there are few candidates, we check whether the candidate found by RANSAC lies or not inside a parallelepiped centered in the predicted new state. Such "3D window" has its sides directly proportional to the current variance, so that when the filter is less certain about the prediction in the previous frames, the window will be enlarged and it will trust the candidate even if it is far away from the prediction:

```
window_k = 15;
sigma_pos = sigma_k(1:3, 1:3);
window = window_k * diag(sigma_pos);

if (in_rect_centered(selected_point, [mu_k(1:3); window]))
    % Use point for Kalman filter update
else
    % Use Kalman prediction only
end
```

where the function `in_rect_centered` simply checks if a point lies in a parallelepiped.

6.9. Tennis ball tracking

Tracking the ball is essential to cope with all the cases in which the detection task is not able to provide well-suited candidates, and also to rule out unlikely candidates. Moreover, it allows to insert some knowledge to the problem, that is the uniformly accelerated motion model for the ball[7]. We initially tried a **particle filter**, but then moved to a **Kalman filter**. It works with unknown correspondences, so to solve the data association problem we devised various heuristics and techniques presented in 6.8.
The model is: $x_{t+1} = Ax_t + Bu_t + \xi_t$, where ξ is the model noise and the other matrices are (we have relied on the model of [1]):

$$A = \begin{bmatrix} 1 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta t \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, B = \begin{bmatrix} 0 \\ 1/2(\Delta t)^2 \\ 0 \\ 0 \\ \Delta t \\ 0 \end{bmatrix}, u_t = g.$$

[7] Actually, as previously mentioned, a more accurate model exploiting the drag and lift coefficients (see [12]) might be used, but for the sake of simplicity we decided to adopt this simple linear model.

Our implementation is presented and explained in the following.
First of all, we initialize all the constants and variables that will be used during the algorithm. Then, when a time step is elapsed, the prediction for the new state is computed:

```
% Run Kalman prediction step
[mu_k1, sigma_k1] = kalman_predict(A_k, mu_k, sigma_k, Bk_uk, Q);
mu_k = mu_k1; sigma_k = sigma_k1;
```

where the function `kalman_predict` simply applies the system dynamics:

```
function [mu_k1, sigma_k1] = kalman_predict(A_k, mu_k, sigma_k, Bk_uk, Q)
    % Kalman filter prediction step
    mu_k1 = A_k * mu_k + Bk_uk;
    if mu_k1(3) < 0
        mu_k1(3) = -mu_k1(3);
        mu_k1(6) = -mu_k1(6);
    end
    sigma_k1 = A_k * sigma_k * A_k.' + Q;
end
```

In case that the ball candidates detection algorithm is able to find a candidate that the data association code will accept as candidate, then the filter is updated using such measurement; otherwise, the filter will consider the prediction as the new state:

```
[mu_k1, sigma_k1] = kalman_update(selected_point, mu_k, sigma_k, C_k, R);
mu_k = mu_k1; sigma_k = sigma_k1;
```

where the function `kalman_update` simply applies the Kalman update equations:

```
function [mu_k1, sigma_k1] = kalman_update(y_k, mu_k, sigma_k, C_k, R)
    % Kalman filter update step
    K = sigma_k * C_k.' / (C_k * sigma_k * C_k.' + R);
    mu_k1 = mu_k + K * (y_k.' - C_k * mu_k);
    sigma_k1 = sigma_k - K * C_k * sigma_k;
end
```

6.10. Compute the velocity

To compute the velocity, we might simply use the value in the state vector of the Kalman filter; however, such value is slow to converge to the real value, therefore we have decided to adopt a different approach.
Since the videos are recorded with a frame rate of $60fps$, then between two frames $1/60 \approx 17ms$ pass; moreover, thanks to the filter, we always know the position of the ball. Therefore, we can simply compute its velocity through the usual definition: $v = \Delta s/\Delta t$. However, also this value might be very wrong, because of errors in the knowledge of the state (3D position), so we decide to compute an average with all the past velocities (such average is reset under certain conditions) to obtain more trustworthy results. To waste less memory, we decide to compute the average in an online way, i.e. by noticing that:
$\hat{\mu}_k = \frac{1}{k}\sum_{i=1}^{k} x_i = \frac{1}{k}(x_k + \sum_{i=1}^{k-1} x_i) = \frac{1}{k}(x_k + (k-1)\hat{\mu}_{k-1}) = \hat{\mu}_{k-1} + \frac{1}{k}(x_k - \hat{\mu}_{k-1})$.
The code is:

```
% Calculate velocity between this frame and the previous frame
v_current = norm(last_wp - mu_k(1:3))/dt * 3.6;

if ~ isnan(last_wp)
    % Update velocity rolling average
    v_samples = v_samples + 1;
    v = v + (v_current - v)/v_samples;
end
```

6.11. Line calling

The line calling functionality is that of determining whether the ball has bounced inside or outside the court. To do so, we can simply compare the position of the ball with that of the court lines when its position on the z axis is close to zero (or when its velocity along the z axis inverts its sign becoming positive). However, no deformation model is taken into account, so the estimate will be very rough.

7 | Pipeline of execution - Example

In this section, we present a clean example of the execution of our method, starting from the original frame, to the final detection of the ball.

1. the original frame of the s20fe camera is:

Figure 7.1: Original frame.

2. the second step is to convert the frame in grayscale since, as aforementioned, our method does not use colours for the detection:

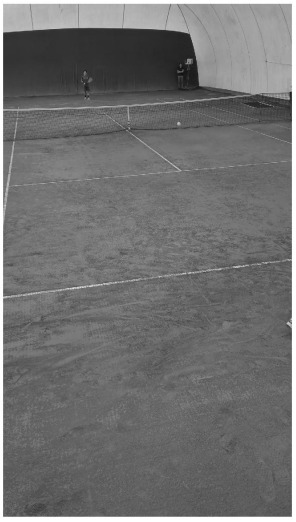


Figure 7.2: Frame in black and white.

3. the detection of the ball passes through various steps (see 6.6.4); first of all we compute the difference (using the `imabsdiff` function) between the grayscale frame and the background (background subtraction):

Figure 7.3: Background subtraction with imabsdiff.

4. to the result of background subtraction, we apply the *morphological closure* to "close" the ball pixels in a single spot, and we do so by using a 5 pixel size structuring element:

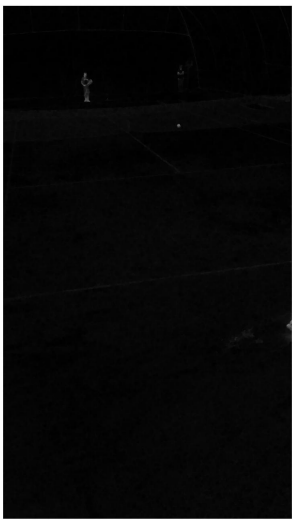


Figure 7.4: Background subtraction with morphological closure.

5. we threshold the just obtained image to get a binary image (the threshold is chosen empirically):

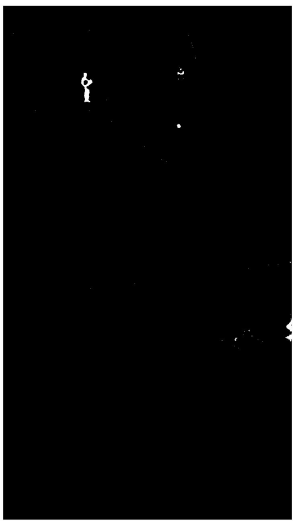


Figure 7.5: Background subtraction bynarized.

6. we apply `bwareaopen` to remove connected components greater than a certain size (200 pixels); basically we aim to remove the player:

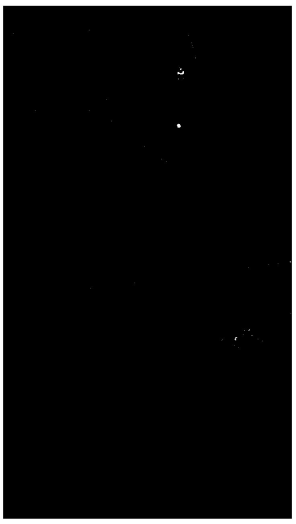


Figure 7.6: Background subtraction cleaned with bwareopen.

7. we now take again the grayscale frame of step 2 and compute the frame difference with the previous three frames at time t-4, t-6 and t-8, applying `imclose` with a 3 pixel structuring element and `bwareaopen` and then ANDing the results (see 6.6.4 for details):

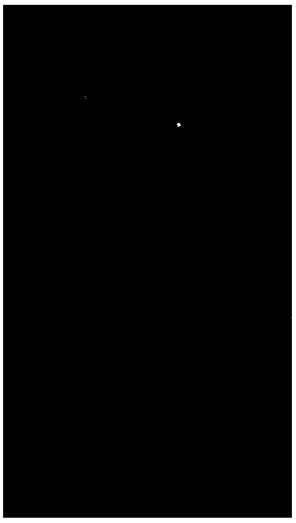


Figure 7.7: Frame differencing.

8. we can finally compute the *logic AND* between the frame difference mask and the background subtraction one:

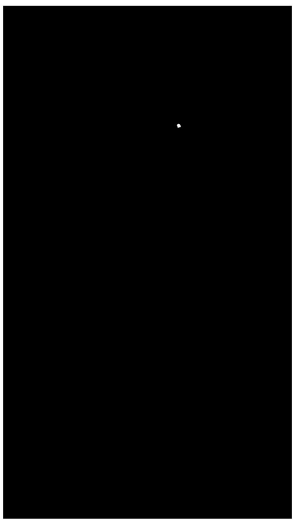


Figure 7.8: Logical and of background subtraction and frame differencing.

9. the final mask is obtained by applying the *dilation* morphological operator to "enlarge" the result; we have used a disk-shape 2-pixel size structuring element:

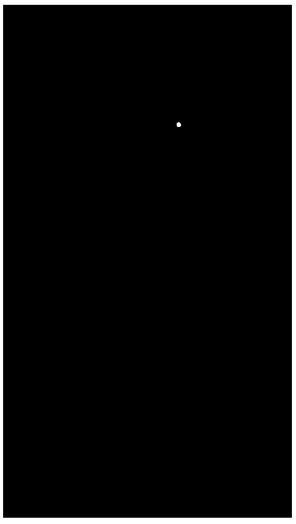


Figure 7.9: The final mask.

10. coming back to the original frame, we can draw a bounding box around the detection:

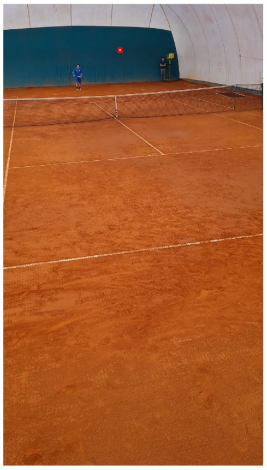


Figure 7.10: Ball detection in the original frame.

11. the next step is to apply `triangulateMultiview` to triangulate such candidate/s with the one/s of the other cameras to obtain the 3D position of the ball; a plot of the result of such operation is:

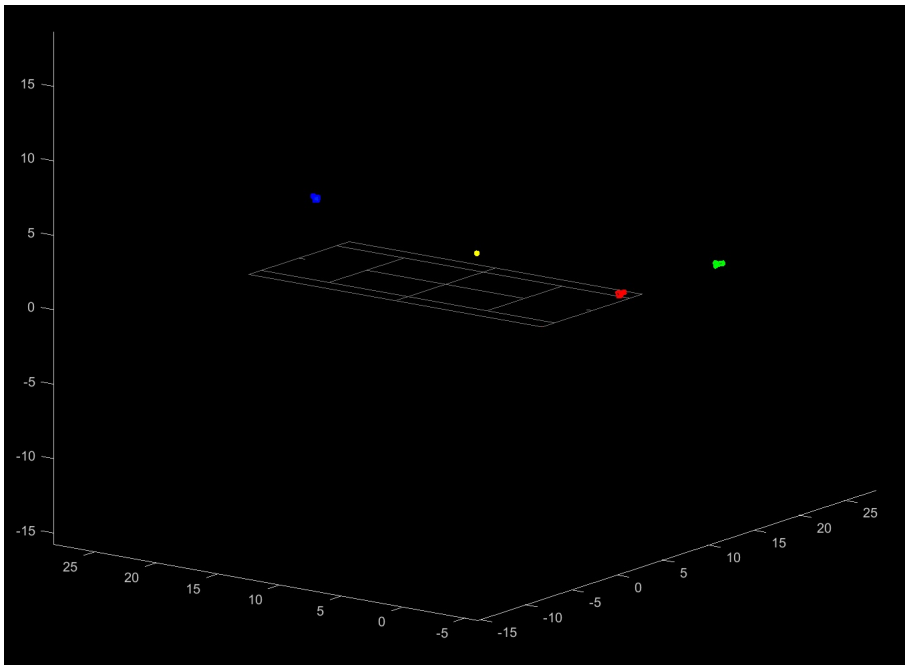


Figure 7.11: Result of triangulation.

12. finally, we decide (data association step) whether or not the result/s of triangulation is/are valid, and we combine the results with the previous guess on the ball position

by updating the *Kalman filter*:

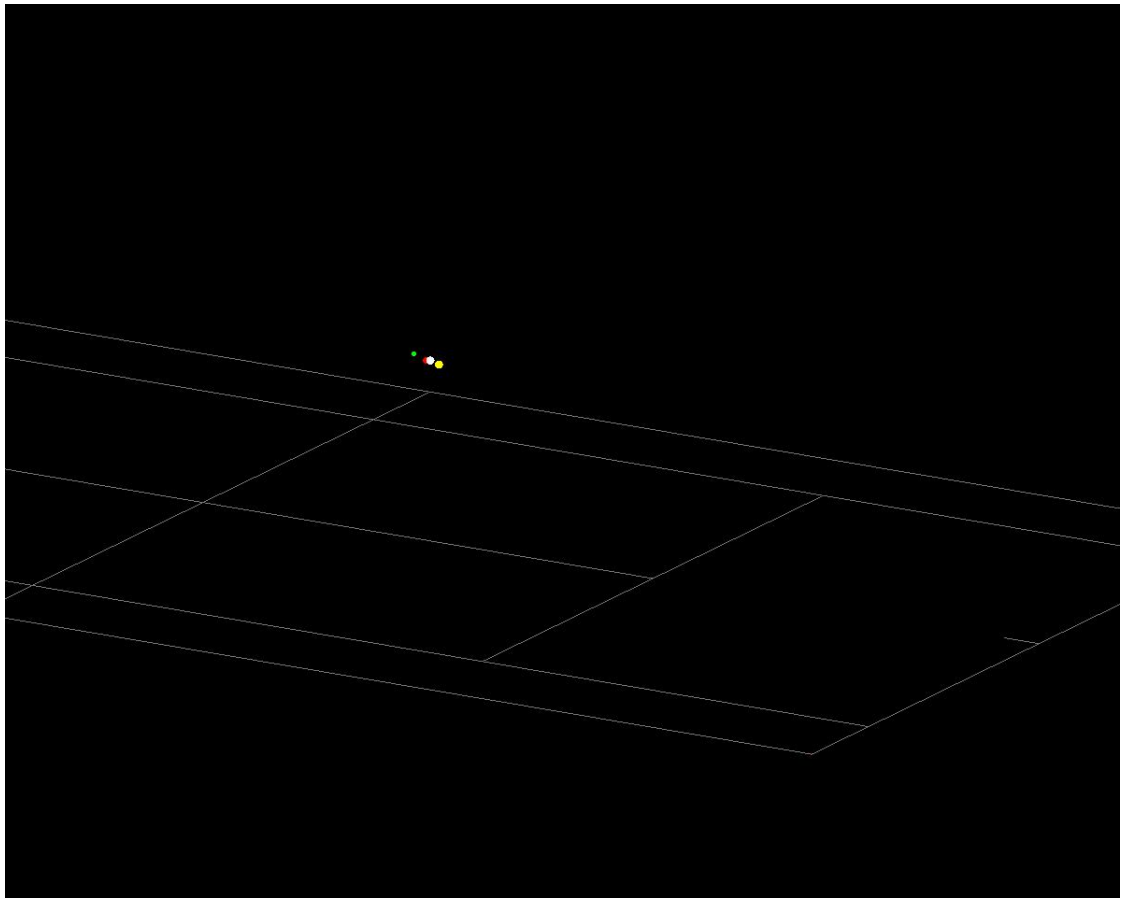


Figure 7.12: The update of the Kalman filter. Green = previous frame point; red = Kalman prediction before update; white = triangulated point; yellow = result after Kalman update.

You can find attached to this report a video showing the 3D trajectory of the ball during time.

8 | Results

We succeeded in detecting and tracking the tennis ball in the videos recorded. Sometimes the position provided by our system is corrupted a lot by noise and deviates from the proper position, but our system is fast to recover.
We present our results by using some images in this section. First of all, you should notice that our system can find the position as well as the velocity and plot them in 3D; the resulting trajectory is parabolic:

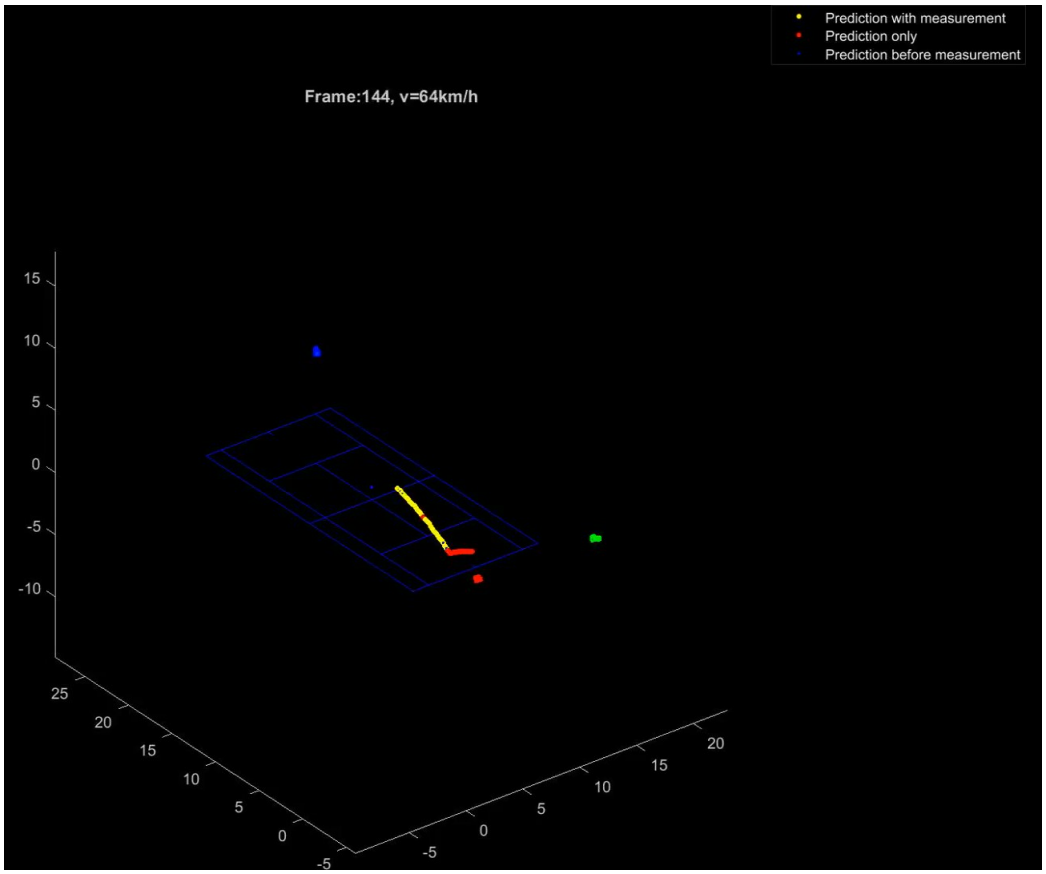


Figure 8.1: A plot in 3D of the trajectory of the ball along with the velocity.

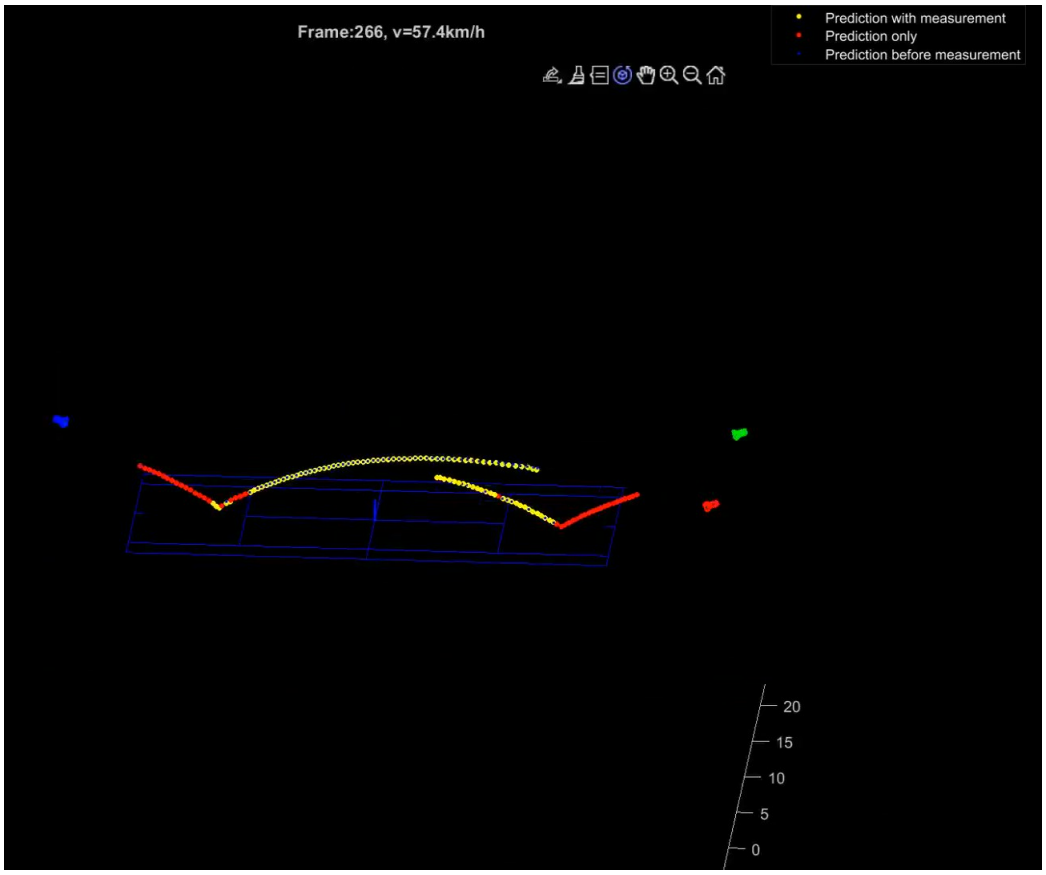


Figure 8.2: A lateral view of the trajectory of the ball that highlights the shape of the curves.

In the next two images, you can see the reprojected points in one of the cameras; there is a clear distinction on how good is the detection when it is with measurement (in yellow) or it is only the prediction of Kalman (in red):

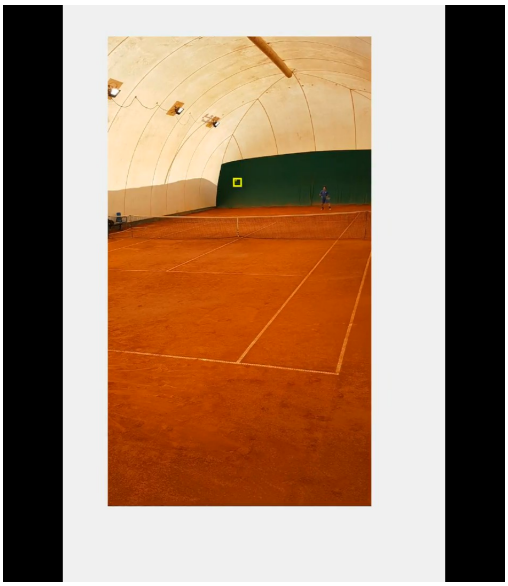


Figure 8.3: The reprojected position in the note8 when there is a measurement.

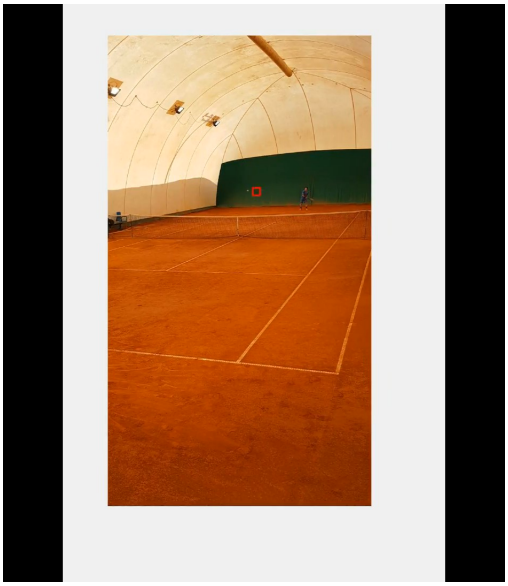


Figure 8.4: The reprojected position in the note8 when there is no trustworthy measurement.

9 | Conclusion

To sum up, the goal of this project was to automatically detect and track the tennis ball from multiple video recordings, and to analyze its motion. It has been achieved through the implementation of the code explained in 6, which is based on ideas found on various papers (cited in the bibliography) and on a specific technique for data association which is of our own (something similar but far more complex is presented in [10]), that is the use of *RANSAC* for fitting a plane in the past positions of the tennis ball to detect the outliers. The rationale behind it is that fitting a plane is far less computationally expensive than trying to understand which point is the most likely next point of a parabolae, and moreover, that the probability that outliers lie next to such plane is low. We have to remark that our code would work rather fine even without this additional feature.
One negative aspect of our implementation is that it does not work in real-time. As a matter of fact, the workload of detecting candidates for all the three cameras and then merging the results is a lot for a single computer (and for the `Matlab` language). This is why we think that a more efficient implementation (using another language) and distributing the load (maybe also parallelizing the tasks) might speed up the process a lot.

10 | Ideas for future works

Some ideas on something that might be added to improve our project:

- adding a deformation model for the ball for improving the line calling functionality;
- detecting and tracking the players;
- performing action recognition of the strokes of the players;
- triangulate using only some cameras (in our implementation the ball must be visible in all the cameras at the same time for the triangulation to work).

Bibliography

[1] Tennis Ball Tracking: 3D Trajectory Estimation using Smartphone Videos, Megan Fazio, Kyle Fisher, and Tori Fujinami

[2] Tennis Ball Tracking for Automatic Annotation of Broadcast Tennis Video, Fei Yan

[3] Layered Data Association Using Graph-Theoretic Formulation with Application to Tennis Ball Tracking in Monocular Sequences, Fei Yan, William Christmas, and Josef Kittler, Member, IEEE

[4] Line Extraction and Player Tracking in Tennis Videos, Volkan Erol

[5] Persistent Particle Filters For Background Subtraction, Yair Movshovitz-Attias

[6] Adaptive background mixture models for real-time tracking, Chris Stauffer, W.E.L Grimson

[7] Kaewtrakulpong, P. and R. Bowden. An Improved Adaptive Background Mixture Model for Realtime Tracking with Shadow Detection

[8] A Low-Cost Computer Vision System for Real-Time Tennis Analysis, Stefano Messelodi, Carla Maria Modena, Michael Sgro

[9] Robust Data Association, Vincent Lepetit, Ali Shahrokn, Pascal Fua

[10] Zhang, Z. “A Flexible New Technique for Camera Calibration.”

[11] Geiger, A., F. Moosmann, O. Car, and B. Schuster. "Automatic Camera and Range Sensor Calibration using a Single Shot,"

[12] "Measurements of drag and lift on tennis balls in flight", Rod Cross, Crawford Lindsey

[13] "A mixed-state CONDENSATION tracker with automatic model-switching", Michael Isard, Andrew Blake

[14] "Physics Based 3D Ball Tracking for Tennis Videos", Antoine Poliakov, Denis Marraud, Livier Reithler, and Christophe Chatain

[15] "Introduction to Data Association", slides from CSE598C Fall 2012 Bob Collins, CSE, PSU

[16] "Players Tracking and Ball Detection for an Automatic Tennis Video Annotation", Kosit Teachabarikiti, Thanarat H. Chalidabhongse, Arit Thammano

[17] "Automatic annotation of tennis games: An integration of audio, vision, and learning", Fei Yan, Josef Kittler, David Windridge, William Christmas, Krystian Mikolajczyk, Stephen Cox, Qiang Huang

[18] "A technology platform for automatic high-level tennis game analysis", Vito Renò, Nicola Mosca, Massimiliano Nitti, Tiziana D'Orazio, Cataldo Guaragnella, Donato Campagnoli, Andrea Prati, Ettore Stella

[19] "Ball tracking in sports: a survey", Paresh R. Kamble, Avinash G. Keskar, Kishor M. Bhurchandi

[20] Hartley, Richard, and Andrew Zisserman. Multiple View Geometry in Computer Vision. 2nd ed. Cambridge, UK; New York; Cambridge University Press, 2003

[21] Baodong, Y.: Hawkeye technology using tennis match. Computer Modelling and New Technologies 18(12C), 400 402 (2014)

[22] "Digital Image Processing", Rafael C. Gonzalez, Richard E. Woods