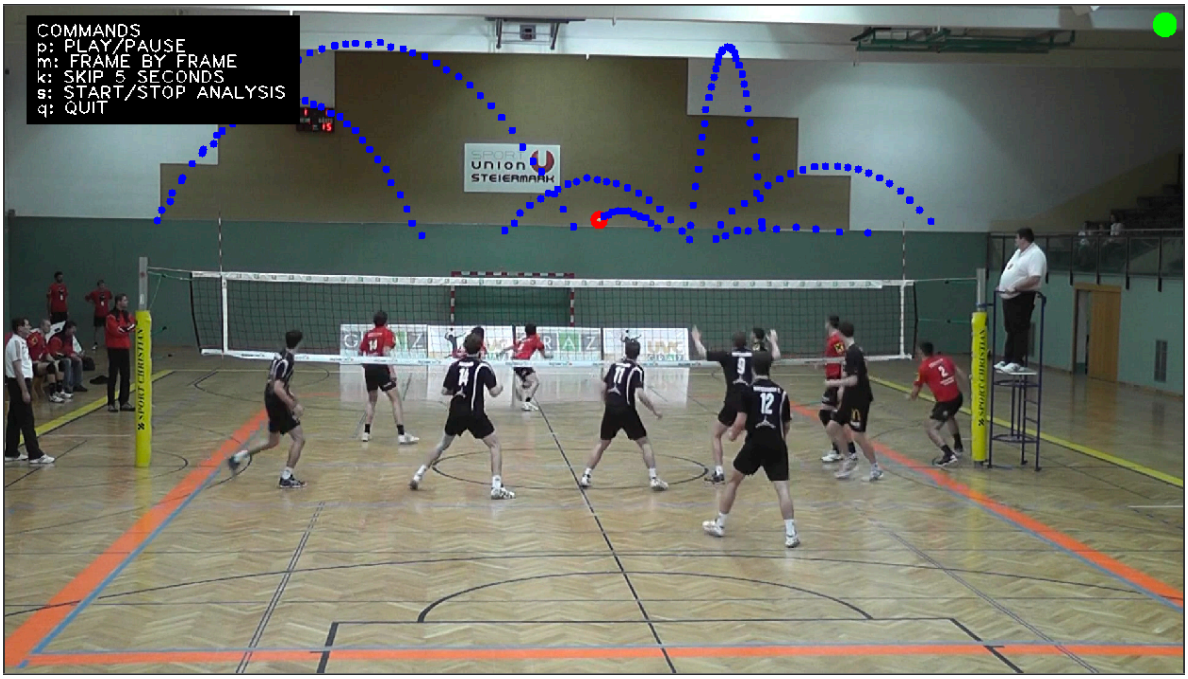
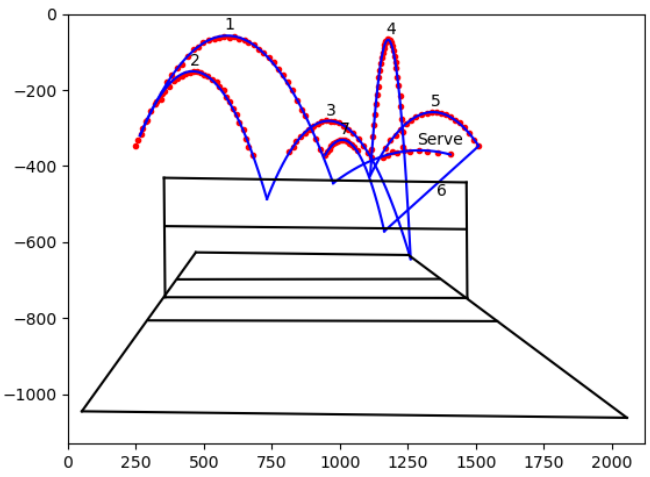
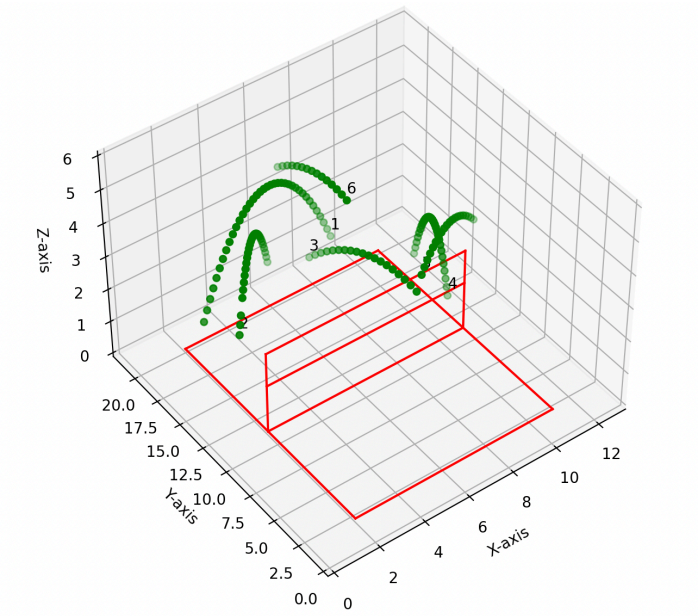
Visual Analysis of Sport Events

Video analysis and ball tracking of a volleyball action - Report

Image Analysis & Computer Vision – Polimi – 2022/23
Matteo D'Oria (10611067) – Lorenzo Castiglia (10578631)
Supervisor: Prof. V. Caglioti







Index

Introduction

Nowadays, the use of automatized methods for analyzing sport events, both for technical and tactical purpose, is fundamental and constantly growing.
Coaches and players exploit many programs to improve their team and skills or to study the weaknesses of the opponent. A computer-based approach, as in almost every case, allows for a more rapid and complete analysis.
In the past, video analysis of the event was new and therefore expensive for many disciplines and it could be applied only by people and teams with a budget to do so (like big football teams, or high-ranking tennis players), in these times the application of programs for sport analysis is wider and more democratic.

In this work, we want to provide an algorithm for ball-tracking and 3D trajectory estimation of a single-camera volleyball action. The results can be deepened for some tactical/technical analysis, like set classification, serve and attack direction, dig quality and so on.
Tracking the ball during a volleyball action is not an easy task, because of the presence of moving players on the court, which can hide the ball to the camera while it is below the net. So, an alternative approach to the constant research of the ball in the action is crucial to allow a good tracking.

The work is divided in 3 phases:
1) ball detection in the frame
2) 2D trajectory extraction
3) 3D trajectory (approximation) elaboration

It is worth noting that in this work we do not make use of Deep Learning techniques; instead, we try to perform tracking using only Computer Vision methods, applied to each frame of the video.
In the end, we will present some possible developments to improve the tracking and the trajectory estimation.

Related work & State-Of-The-Art

The field of sport video analysis and in particular of the ball tracking is constantly growing. The State-of-the-art is obviously much affected by the recent development of Deep Learning methods, which outperform any other forms of ball detection.
An example of this kind of works is "*DeepBall: Deep Neural-Network Ball Detector*" by Komorowki et al., a paper of the 2019 which describes the use a deep network detector for ball in long shot videos, like a football match.
Staying in the Deep Learning scope, neural-network based object detector can be adapted to perform ball detection. For example, YOLO original paper "*You Only Look Once: Unified, Real-Time Object Detection*" by Redmon et al in 2015 has been

exploited by thousand of other papers for its development and implementation in countless situations and works.

Alternatively, leaving the Deep Learning field, there are previous works based only on Compute Vision and geometry techniques performing sport video analysis.
For what concerns volleyball, we based our work on "*Ball tracking and 3D trajectory approximation with application to tactics analysis from single-camera volleyball sequences*" by Chen, Tsai, Lee, Yu of 2011: each frame is analyzed, firstly, from the audio point of view to detect the starting whistle by the referee and then the video is used to perform camera calibration, detect the ball and extract the 2D trajectory. At the end, using both the camera calibration matrix and the extracted 2D trajectory, a 3D approximated trajectory is computed and it can be used for any desired analysis.

Solution approach
First of all, in this project, we do not make use of Deep Learning techniques (no trained or pre-trained models), but we only exploit Computer Vision methods.
We work on the frame trying to extract the most "likely" ball, generating some trajectories and combining them to obtain the whole ball movement during the action.
The work is performed on a single fixed camera video, capturing the volleyball match from behind one of the 2 baselines.
The solution is divided in 3 phases:
1) ball detection in the frame
2) 2D trajectory extraction
3) 3D trajectory (approximation) elaboration

which will be presented in the following sections.

1. Ball detection
Given each frame of the action video, we need to find the ball.
Obviously, this task is not straightforward: there are other moving objects, like the players, but also people in the audience or the referee, which could be mistaken for the ball.
Since we are not taking advantage of any Deep Learning model, which can classify and distinguish a ball from the rest, we need to do some simplifications.
As the ball follows a physical trajectory, it is possible to limit our search of the ball candidates only in the cleanest area of the frame, which is the zone above the net. In this ROI (Region Of Interest), there are no interference by players, which are the most common and hard to solve. So, the ball detection is performed only in this area, allowing to reduce the candidates and the work to get rid of some of them, checking their shape or size.
This solution does not interfere with the trajectory extraction step: although we don't always have the ball position in the video, we can estimate where the ball might be

using physics and geometry; the ball, being a physical object in the real world, follows a parabolic trajectory and so, we just need some points (at least 5) to extract the complete parabola, using a geometric method.
This will be better explained in the following section.

Let's analyze the detection more deeply.

First of all, in the ROI, the candidates are extracted using the *createBackgroundSubtractorMOG2* method, which allows to subtract from each frame the static objects, creating a mask of the moving ones. This has 3 parameters to be tuned:
- history: length of the frame history to be taken into account (set to 50)
- varThreshold: threshold on distance between the pixel and the model to decide whether a pixel is well described by the background model (set to 50)
- detectShadows: makes the method detect shadows and mark them (set to False)

Once the mask of the frame is created, the contours are obtained using the *findContours* method and the following parameters:
- RetrievalModes: RETR_EXTERNAL (only the extreme outer contours)
- ContourApproximationModes: CHAIN_APPROX_SIMPLE (only the end points)

Once we have the candidates, there are 2 situations: the ball was never detected (new action) or vice versa.
In the first case, we need to select the most likely ball, by exploiting its circular shape and dimension. This is a crucial phase, because a mistake can lead to a completely wrong trajectory, given that each frame takes advantage of the previous ball positions. But, since the search is done in the ROI above the net, the possibility that something can be mistaken for the ball is very low.
The check is performed in this way:
a) each contour is assigned a score, which is the difference between the area of its minimum enclosing circle and its surface
b) a sorted list of the contours is returned ranked by the score
c) the "best" contour (the one most similar to a ball) is returned.

Once we have detected a ball, we keep its position and shape (x,y and the ray) inside a tracker object.
In the second case, it is possible to use the ball history to track the ball: so, among the candidates (which are not so many in the ROI), the ball is chosen among the possible as the one nearest to the estimated position (given the previous ones). So, the ball is detected with no error.
There are 2 cases in which the ball is lost:
- when it is out-of-frame: in this case the detection is performed near the point of the last detection, enlarging the area of research

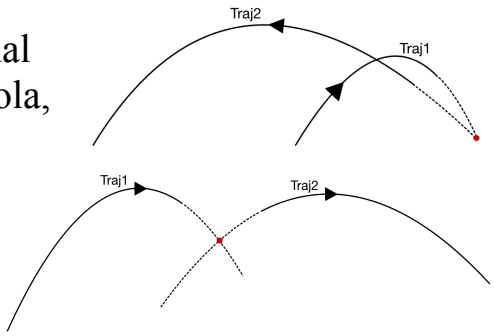
- when it is below the net: in this case, the ball is not detected and the algorithm waits until it is again visible in the ROI; the complete trajectory is extracted in a different (geometric) way

When the ball is lost for more than 40 frames (if the history of the ball is longer than 40 frames, otherwise the length of the history is used), the action is considered concluded.

2. 2D trajectory extraction

Once we have a history of the ball position, we can perform the extraction of the trajectory in 2 dimensions.
In this phase, the key part is understanding when the new position is related to another parabola. This can be done by following the trajectory of the ball and detecting a change in the direction which is not the one we expected.
If the ball is simply far from the last position and it is going up, then it is quite easy to detect a new parabola.
When the ball is near the last position (for example, for a set), then we need to work on the change of the direction of the ball, from downward to upward. To reduce the risk of not getting an higher-position ball, the ROI is slightly reduced, moving the lower limit by some pixel towards the up, when the ball is lost.

Solved this phase, we have a set of points, divided in different trajectories.
For each set, a parabola is extracted interpolating the points, using the *numpy polyfit* method, which uses the least squares method to fit a desired-order polynomial to a set of points.
These parabolas have no start or end, so the next and final step is to find the starting and final points of each parabola, by intersecting them and selecting the correct one, since they have 2 possible solutions. The correct intersection is obtained by exploiting the directions of the 2 subsequent trajectories and, if they are the same, also the first y coordinate of the second trajectory.



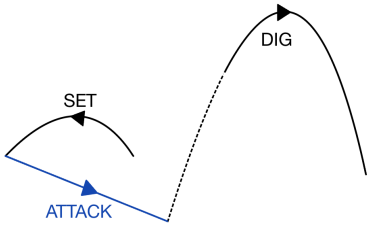
Finally, having for each parabola the equation, the starting and final point, it is possible to draw the complete trajectory of the ball.

2.1 Attack handler

For what concerns the attacks, we need to separate 2 kinds: the attack defended by the opposite team which leads to another action and the final attack.

For the first type, we decided to exploit once again geometry: since we have the two parabolae of the set and the dig, we can connect the 2 trajectories taking into account

their directions. This is simply done by extending the second bump trajectory just below the net and connect the new end-point with the end-point of the set trajectory.



For a final attack, instead, we do not have any other trajectory to take advantage of and moreover if we try to find the ball during its fast movement towards the ground, we'd encounter a lot of trouble due to the noise in the area below the net of the players.
This could be treated by using Deep Learning techniques, like training a model to distinguish the ball from the players' bodies and then draw the attack trajectory.
Otherwise, any other Computer Vision methods would lead to a very overfitted method on each particular case which cannot be generalized, specially with a not high-resolution camera.
Therefore, we decided to leave this for future implementation and not totally approximate the final attack.

3. 3D trajectory

In a volleyball action the ball has a trajectory that can be divided into multiple sub-trajectories. In this project we perform an approximation by considering those sub-trajectories as parabolic curves, so that we can use a simplified set of physical equations to describe the ball movement.
Using these equations we aim to build a system in order to find the 3D starting point of a sub-trajectory and the ball's initial velocity. From there we can reconstruct the entire trajectory using the same physical equations.
To perform this step we need some more data, the computation is showed in the following section (3.1).

3.1 Additional data and computation required

3.1.1 Timestamps

Given a volleyball action, we have so far obtained a list of sub-trajectories, each of which is a list of ball positions in the 2D frame. Knowing the specific frame number of the video in which the ball was detected and the fps, we can obtain the timestamp associated to each point of the trajectory. In our videos the fps are 25.

$$timestamp_i = \frac{frame\_number_i}{fps}$$

So for instance if a ball is detected in frame number 50, we know the timestamp is 2 seconds since the video started. The timestamp is calculated for every point.

3.1.2 Projection matrix

A camera is a mapping from the 3D real world to the 2D image space.
The real world point $(X, Y, Z)^T$ in 3D space is represented as a homogeneous 4-vector $W = (X, Y, Z, 1)^T$, the image point $(x, y)^T$ in 2D space is represented as a homogeneous 3-vector $u = (x, y, 1)^T$ and P represents the 3×4 homogeneous camera projection matrix. The mapping from the 3D real world to the 2D image is written as

$$\vec{u} = P \cdot \vec{W} \Rightarrow \vec{u} \times P\vec{W} = \vec{0} \Rightarrow \vec{u} \times \begin{bmatrix} P_1^T \\ P_2^T \\ P_3^T \end{bmatrix} \vec{W} = \vec{0} \Rightarrow \vec{u} \times \begin{bmatrix} P_1^T W \\ P_2^T W \\ P_3^T W \end{bmatrix} = \vec{0}$$

where P_i^T is the i-th row of matrix P. Solving the cross product we obtain

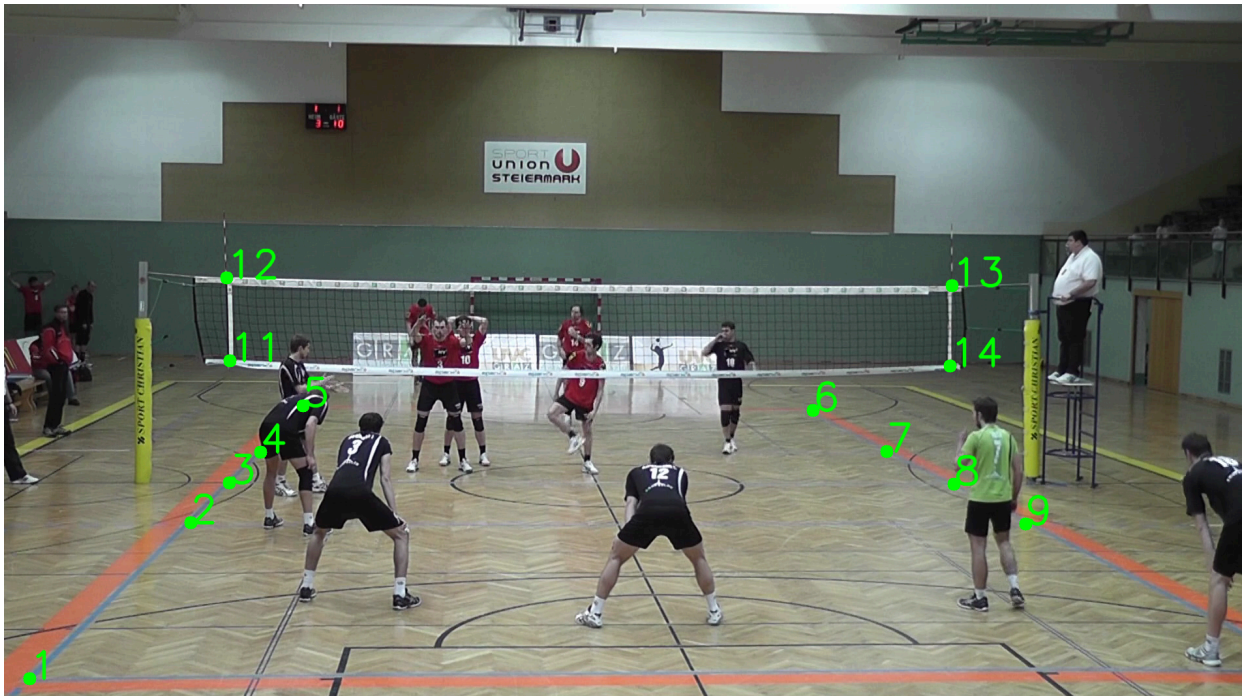
$$\begin{vmatrix} \hat{i} & \hat{j} & \hat{k} \\ x & y & 1 \\ P_1^T W & P_2^T W & P_3^T W \end{vmatrix} = \vec{0} \Rightarrow \begin{cases} -P_2^T W + yP_3^T W = 0 \\ P_1^T W - xP_3^T W = 0 \\ -yP_1^T W + xP_2^T W = 0 \end{cases} \Rightarrow$$

$$\begin{bmatrix} 0 & -W^T & yW^T \\ W^T & 0 & -xW^T \\ -yW^T & xW^T & 0 \end{bmatrix} \begin{pmatrix} P_1 \\ P_2 \\ P_3 \end{pmatrix} = 0 \Rightarrow \begin{matrix} linear \\ dep. \end{matrix} \Rightarrow \begin{bmatrix} 0 & -W^T & yW^T \\ W^T & 0 & -xW^T \end{bmatrix} \begin{pmatrix} P_1 \\ P_2 \\ P_3 \end{pmatrix} = 0$$

For each correspondence between u and W we obtain a relationship of the type shown above. The matrix P has 12 entries, and (ignoring scale) 11 degrees of freedom, so it is necessary to have at least 11 equations to solve for P. The system obtained is

$$A \cdot \underline{P} = \vec{0} \Rightarrow \begin{bmatrix} A^1 \\ A^2 \\ \vdots \\ A^n \end{bmatrix} \begin{pmatrix} P^1 \\ P^2 \\ P^3 \end{pmatrix} = \vec{0}$$

Where $A^{2n \times 12}$ is the matrix containing the points correspondences and $\underline{P}^{12 \times 1}$ the projection matrix reshaped as a vector.
We chose 14 non co-planar point correspondences to create the system. Since point n.10 is out of frame, we obtain it using projective geometry.
We wrote the 3D point coordinates manually considering the official court dimensions.



With the 14 non-coplanar point correspondences obtained above, we can solve for $\underline{P}$ using the direct linear transform (DLT) algorithm.
The solution is obtained from $argmin_{\underline{P}}\|A\underline{P}\|_2$

$$\begin{cases} argmin_{\underline{P}}\|A\underline{P}\|_2 \\ [U, D, V^T] = SVD(A) \end{cases} = argmin_{\underline{P}}\|UDV^T\underline{P}\|_2 = argmin_{\underline{P}}\|DV^T\underline{P}\|_2$$

Since U is orthogonal it doesn't affect the solution.
To avoid the null solution we constrain its norm to 1.

$$\begin{cases} argmin_{\underline{P}}\|DV^T\underline{P}\|_2 \\ \underline{y} = V^T\underline{P} \\ \|\underline{y}\|_2 = 1 \end{cases} \Rightarrow \begin{cases} argmin_{\underline{y}}\|D\underline{y}\|_2 \\ \|\underline{y}\|_2 = 1 \end{cases} \Rightarrow \underline{y} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix} \Rightarrow \underline{P} = V_{12}$$

P is obtained by reshaping $\underline{P}$, which is equal to the last column of $V^{12\times 12}$.

3.2 Trajectory estimation

As stated previously, the ball trajectory can be divided into multiple sub-trajectories. In this project we perform an approximation by considering those sub-trajectories as parabolic curves, described by the following physical equations.

$$\begin{aligned} X_t &= X_0 + V_X \cdot t \\ Y_t &= Y_0 + V_Y \cdot t \\ Z_t &= Z_0 + V_Z \cdot t + g \cdot t^2/2 \end{aligned}$$

$P_0 = (X_0, Y_0, Z_0)$ is the starting point of the sub-trajectory, $V = (V_X, V_Y, V_Z)$ the initial velocity of the ball in P_0 and (X_t, Y_t, Z_t) the ball coordinate at time t.
Using these equations we aim to build a system with P_0 and V as unknowns.

Using $u_t = (x_t, y_t, 1)^T$ and
$W_t = (X_t, Y_t, Z_t, 1)^T = (X_0 + V_X \cdot t, Y_t = Y_0 + V_Y \cdot t, Z_t = Z_0 + V_Z \cdot t + g \cdot t^2/2)^T$
We can write

$$\begin{bmatrix} 0 & -W_t^T & yW_t^T \\ W_t^T & 0 & -xW_t^T \end{bmatrix} \begin{pmatrix} P^1 \\ P^2 \\ P^3 \end{pmatrix} = 0$$

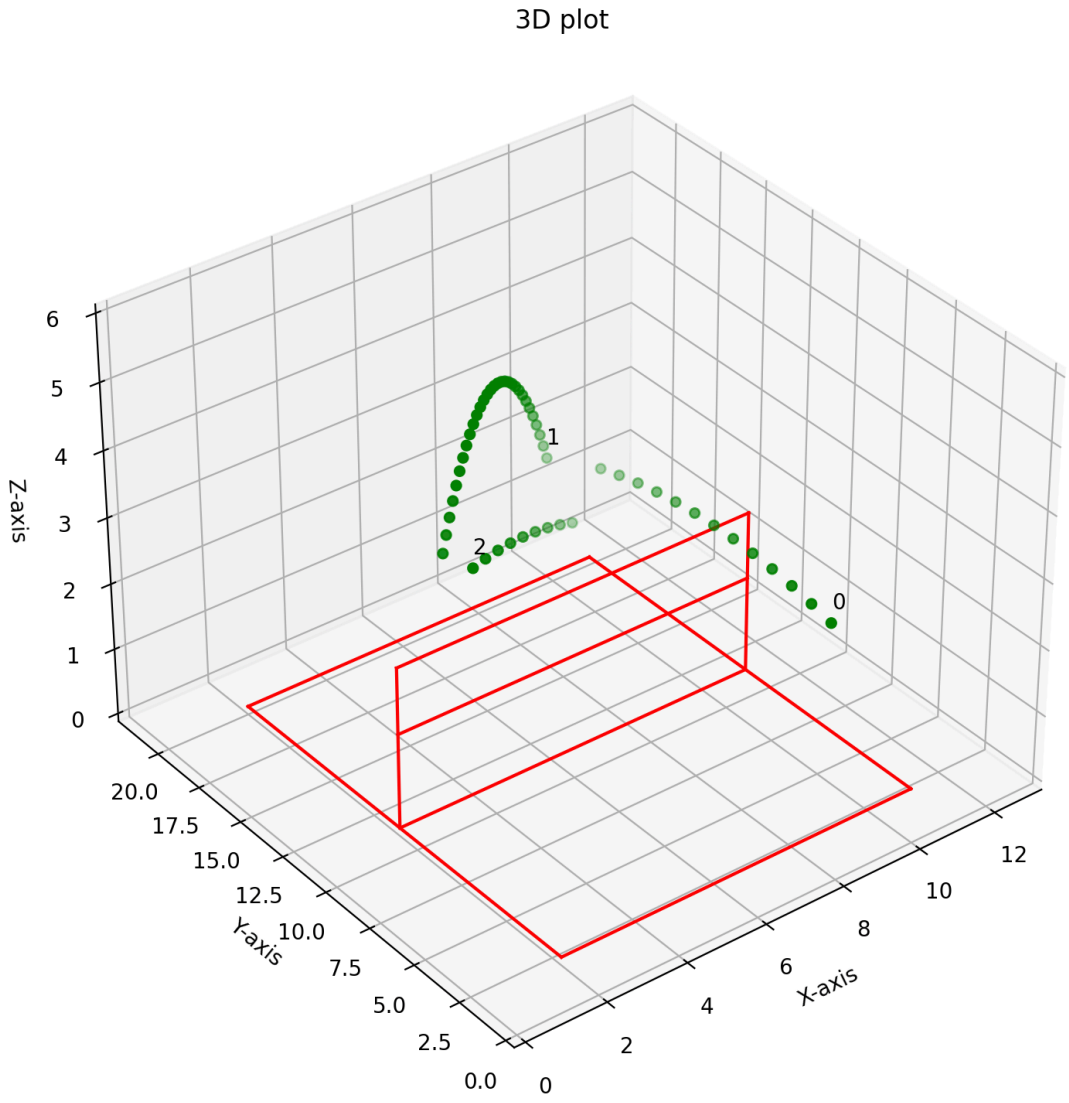
$$\begin{cases} X_0(p_{11}-x_t p_{31}) + V_X(p_{11}t - x_t p_{31}t) + Y_0(p_{12}-x_t p_{32}) + V_Y(p_{12}t - x_t p_{32}t) + Z_0(p_{13}-x_t p_{33}) + V_Z(p_{13}t - x_t p_{33}t) + gt^2/2p_{13} - gt^2/2p_{33}x_t + p_{14} - x_t p_{34} = 0 \\ X_0(y_t p_{31}-p_{21}) + V_X(y_t p_{31}t - p_{21}t) + Y_0(y_t p_{32}-p_{22}) + V_Y(y_t p_{32}t - p_{22}t) + Z_0(y_t p_{33}-p_{23}) + V_Z(y_t p_{33}t - p_{23}t) - gt^2/2p_{33} + gt^2/2p_{33}y_t - p_{24} + y_t p_{34} = 0 \end{cases}$$

$$\begin{bmatrix} p_{11}-x_1p_{31} & p_{11}t_1 - x_1p_{31}t_1 & p_{12}-x_1p_{32} & p_{12}t_1 - x_1p_{32}t_1 & p_{13}-x_1p_{33} & p_{13}t_1 - x_1p_{33}t_1 \\ p_{21}-y_1p_{31} & p_{21}t_1 - y_1p_{31}t_1 & p_{22}-y_1p_{32} & p_{22}t_1 - y_1p_{32}t_1 & p_{23}-y_1p_{33} & p_{23}t_1 - y_1p_{33}t_1 \\ \vdots & & \vdots & & \vdots & \\ p_{11}-x_Np_{31} & p_{11}t_N - x_Np_{31}t_N & p_{12}-x_Np_{32} & p_{12}t_N - x_Np_{32}t_N & p_{13}-x_Np_{33} & p_{13}t_N - x_Np_{33}t_N \\ p_{21}-y_Np_{31} & p_{21}t_N - y_Np_{31}t_N & p_{22}-y_Np_{32} & p_{22}t_N - y_Np_{32}t_N & p_{23}-y_Np_{33} & p_{23}t_N - y_Np_{33}t_N \end{bmatrix} \begin{bmatrix} X_0 \\ V_X \\ Y_0 \\ V_Y \\ Z_0 \\ V_Z \end{bmatrix} = \begin{bmatrix} x_1(p_{33}\ gt_1^2/2 + p_{34}) - (p_{13}\ gt_1^2/2 + p_{14}) \\ y_1(p_{33}\ gt_1^2/2 + p_{34}) - (p_{23}\ gt_1^2/2 + p_{24}) \\ \vdots \\ x_N(p_{33}\ gt_N^2/2 + p_{34}) - (p_{13}\ gt_N^2/2 + p_{14}) \\ y_N(p_{33}\ gt_N^2/2 + p_{34}) - (p_{23}\ gt_N^2/2 + p_{24}) \end{bmatrix}$$

The least squares solution gives the vector $(X_0, V_X, Y_0, V_Y, Z_0, V_Z)^T$. The system is constructed and solved for each sub trajectory.

3.2.1 Visualisation

At this point we have the starting point and initial velocity of each sub-trajectory, by combining them with the parabolic physical equations we can derive a list of 3D points that belong to the trajectory and visualise them in a 3D plot. An example follows.



3.3 Limitations

Our 3D reconstruction method has some limitations. In particular, it can only estimate trajectories in which ball candidates are present. This happens because in addition to the image coordinates, the method requires a timestamp for each point. An interpolated or estimated 2D trajectory doesn't have the timestamp information, unless it is manually extracted each time or guessed. Therefore an attack cannot be reconstructed.

Another side effect is that we cannot reconstruct the entire ball trajectory, but only the sections inside the ROI, above the net. This creates discontinuities between sub-trajectories, sometimes leading to errors. A way to mitigate this last problem is to find the meeting point of the two sub-trajectories, which is often outside the ROI, and use that information to improve the solution. The coordinates can be estimated from the interpolated parabolae. However, as stated in this paragraph, we would also need to estimate, basically guess, the timestamp of that occurrence.

Finally, in our experiments, we observed a good performance in every type of shot except for the serve. In most cases the depth of the ball is wrong, particularly when the trajectory is straight to the camera. We believe more information and data is needed to improve the solution. For sure a big issue is that a fast moving ball provides very few candidates, making our system underdetermined and therefore imprecise. A mitigation might be using higher frame rate videos (our are 25fps), so that we capture more data points.
For this reason, we decided to not show the serve, only defence and set ("ricezione" and "palleggio") in the following examples.

Implementation

The project was implemented in Python, only using utility libraries. In detail *opencv* to handle video files, *matplotlib* for visualisation purposes and *numpy* for arrays and the implementation of tools such as SVD. To run the code, these are the only packages required to be installed. Alternatively, a conda environment can be created using the *environment.yml* file and the command *"conda env create -f environment.yml"*.

Running the code

From CLI type the command *"python main.py"*

The command *"python main.py -h"* shows the available arguments to provide, in particular the flag -v for the video id. For instance *"python main.py -v 1"*.

Folder structure

In the root folder all the python source files are present, including the main function and all the classes and methods we implemented and the *environment.yml* file.

Inside /Sources there are:

- two text files containing the points' coordinates used for the projection matrix
- two reference images for the ball and the court
- the folder /Actions where the videos are present, filename is "0209"+video_id+".mp4"

Extra

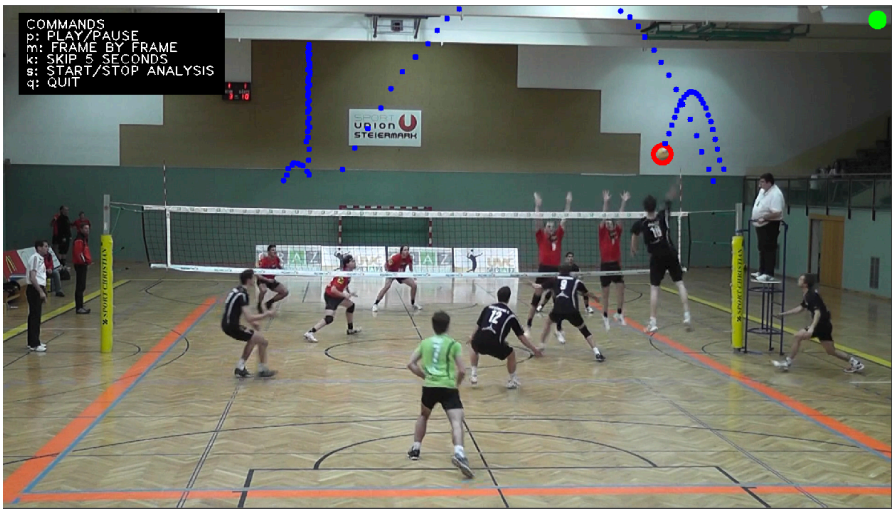
The system is also provided of a video controller that allows the user to:

- start/stop the video analysis (cmd: S)
- pause/play the video (cmd: P)
- play frame-by-frame (cmd: M)
- skip 5 seconds (cmd: K)

Experiments & results

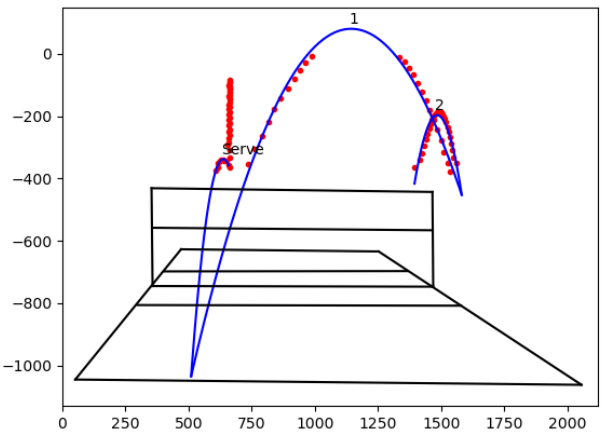
To test the video analysis system, we have used a volleyball match recorded by a camera posed behind one of the 2 baselines and from that we have extracted 7 actions. For each action, we will discuss what we obtain and what can be improved.

Action 1

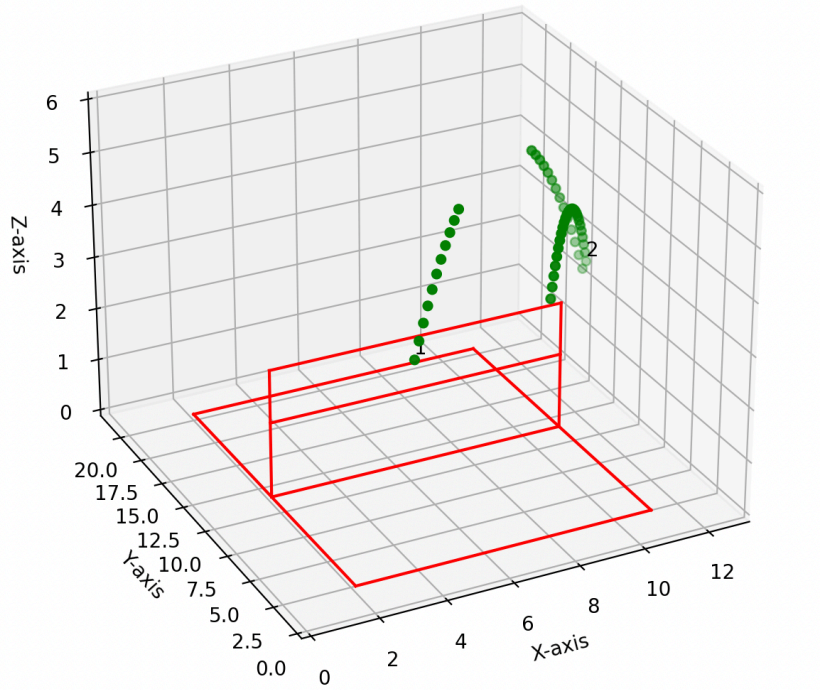


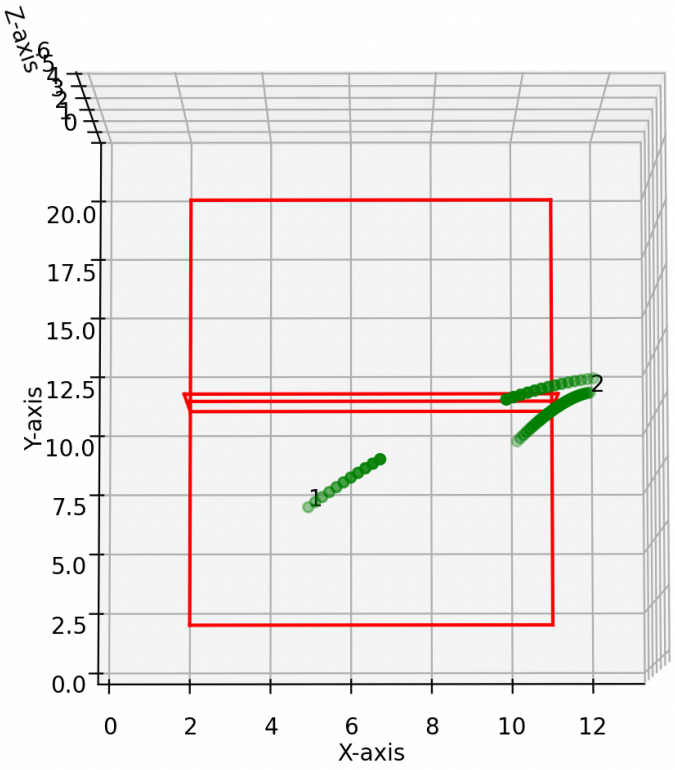
This is a simple action, with a sideline service, a dig, a set and the attack.
As it can be seen, the ball is always well detected above the net and also in case of out-of-frame.

The 2D trajectory is extracted almost correctly, with a little bit of imperfection on the trajectory #1, maybe due to the out-of-frame issue.
In this case, the final attack is not detected as explained before and the last ball position detected slightly move to left the set trajectory (#2). It can be noticed that also the throwing of the ball in air for the service (ball toss) is detected, but it is not part of the trajectory since it begins with the movement towards the other court side.

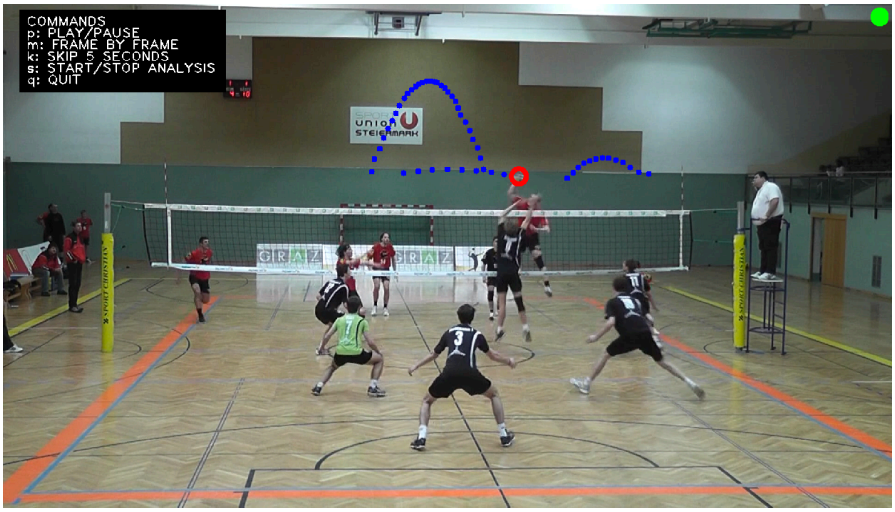


Regarding the 3D reconstruction, we obtained a very precise result for trajectories #1 and #2.
The interruption of #1 is due to the fact we only reconstruct points of ball candidates instead of the whole interpolated parabola. The reason is explained in more detail in section **3.3 Limitations**.
Since the intersection point of these two sub-trajectories is very close to the ROI, we don't observe a discontinuity between the two.



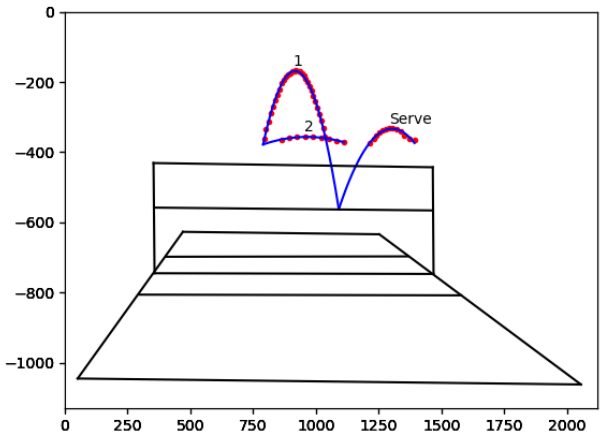


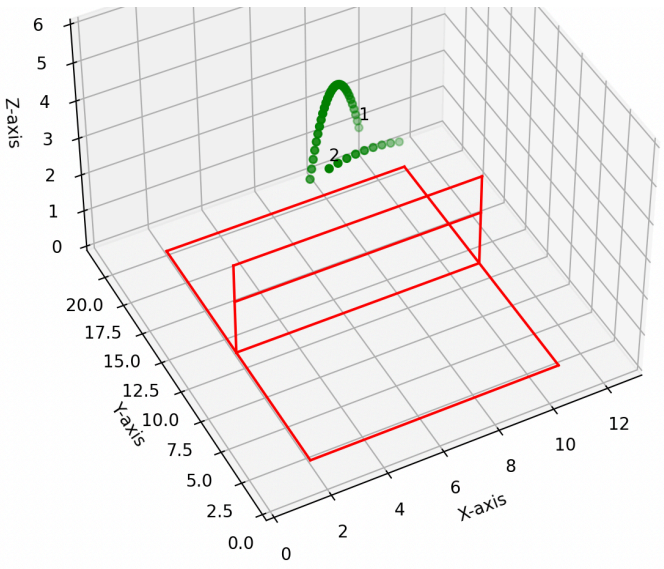
Action 2



Again a simple action, this time the ball is moved by the team beyond the net with respect to the camera.

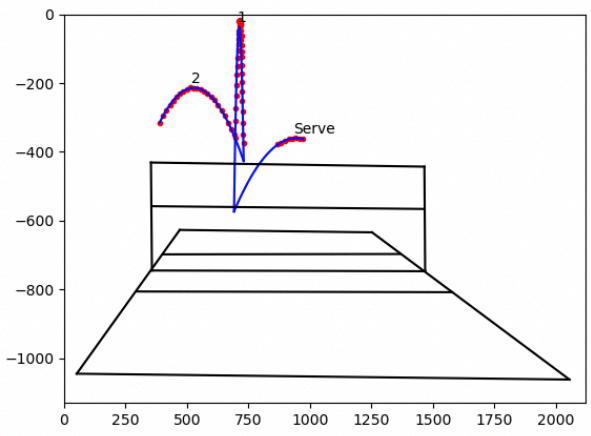
In this case, the 2D trajectory is perfectly extracted, as the blue line follows the red dots of the ball positions for whole action.
There's no ball toss since the player immediately hit the ball.
The 3D reconstruction is consistent with what we would expect.





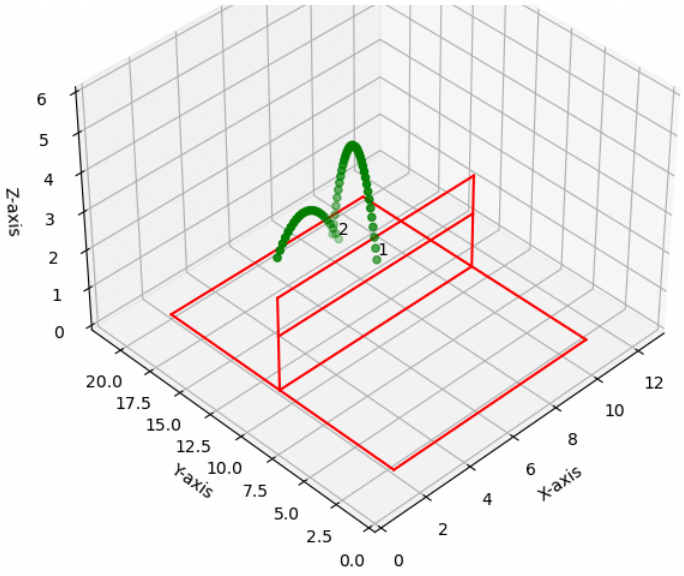
Action 3

This action has a fast and short set to the middle blocker (the player at the center of the net) which attacks immediately.



However, the set is correctly extracted in its whole trajectory.
In this case, the attack completely disappears behind the players and it is almost impossible to detect the ball and extract the attack line.

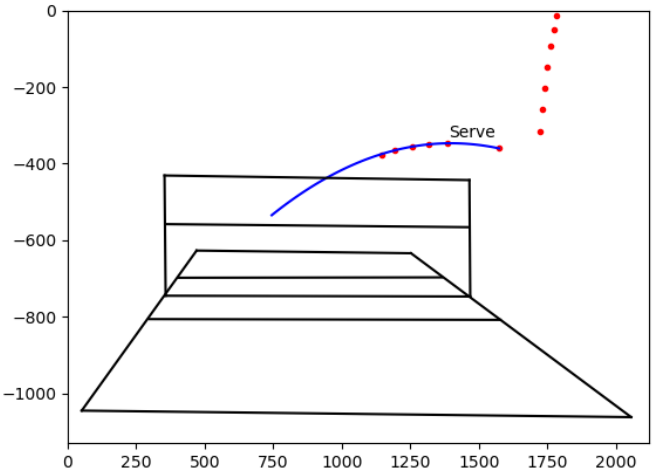
The 3D reconstruction of trajectories #1 and #2 is precise and consistent with expectations.



Action 4

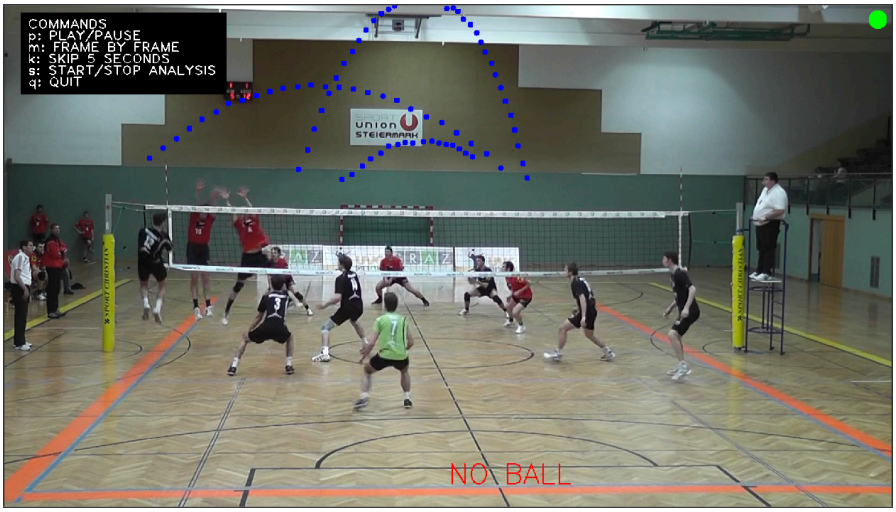
This is an “only-service” action, so either an ace or a serve error.

The trajectory is correctly extracted until the ball goes below (and beyond in this case) the net, which, as well as the players, hides it from the camera.
We’ve decided to extend the line as much as it makes sense to give a more complete trajectory of the serve.
It can be also noticed the descending part of the ball toss.



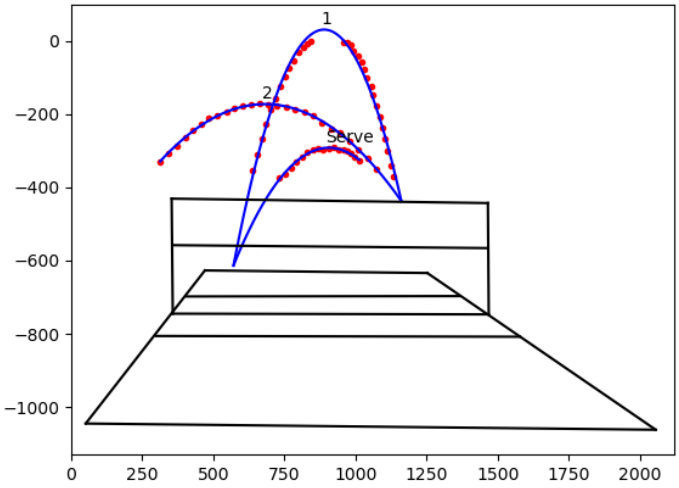
We skip the 3D reconstruction since the fast moving ball provides too few candidates to solve the system of equations, giving a grossly approximated result.

Action 5

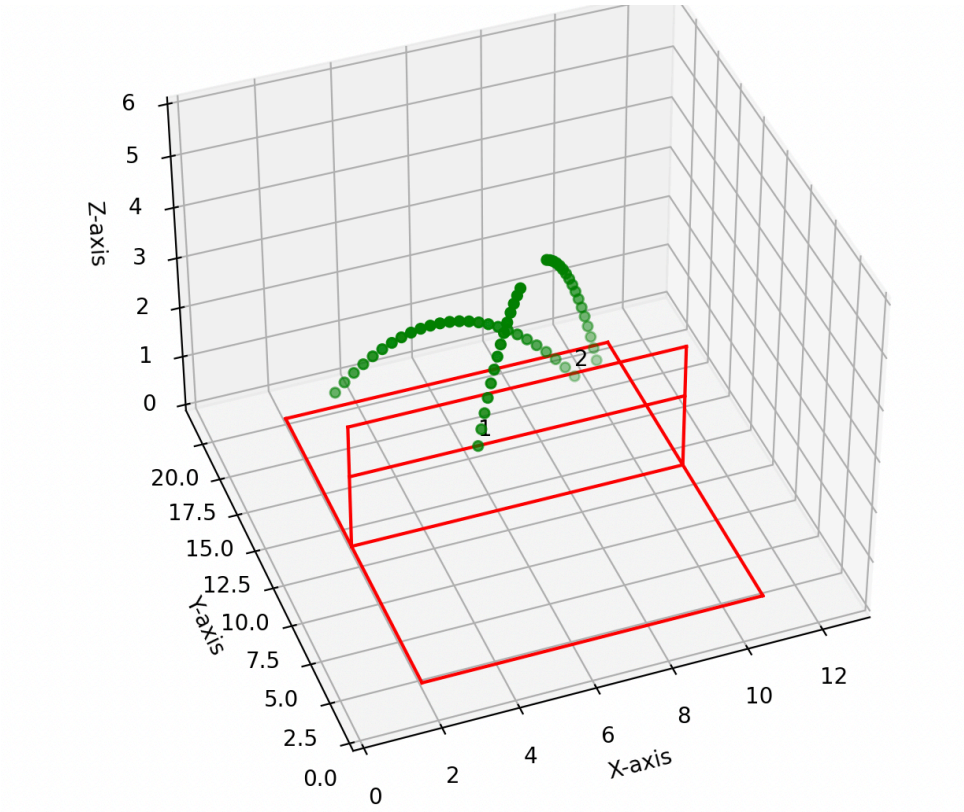


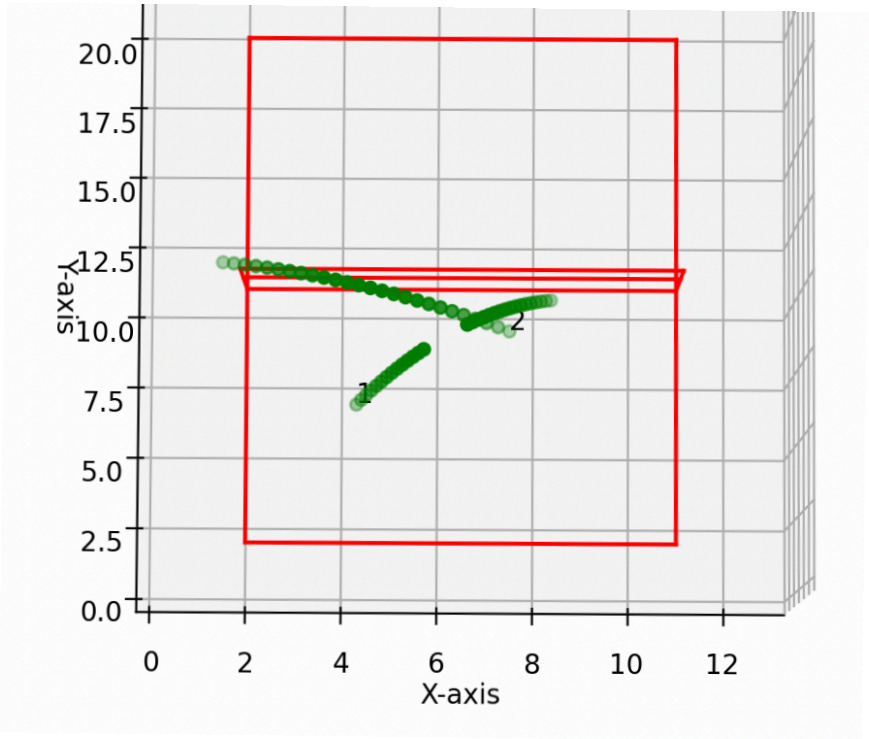
This action has another out-of-frame situation and it can be noticed a little difficulty to detect a correct ball on the serve, when it pass in front of the white poster.

However, the 2D trajectory is perfectly extracted.
Again we can observe some inaccuracies due to the short out-of-frame transition of the ball and the first detection of the ball coming back in the frame.



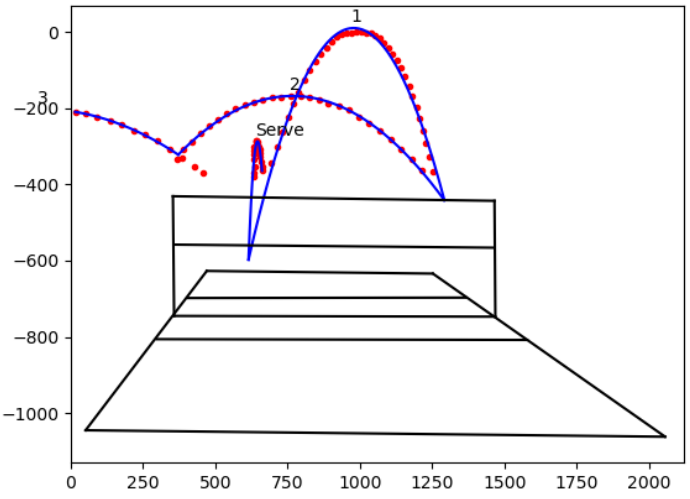
The 3D reconstruction appears to be very reasonable, except for the small inconsistency of the intersection point of the two parabolae. The two sub-trajectories appear disjointed for this reason.
A detailed explanation of the causes of this discontinuity can be found in section **3.3**.





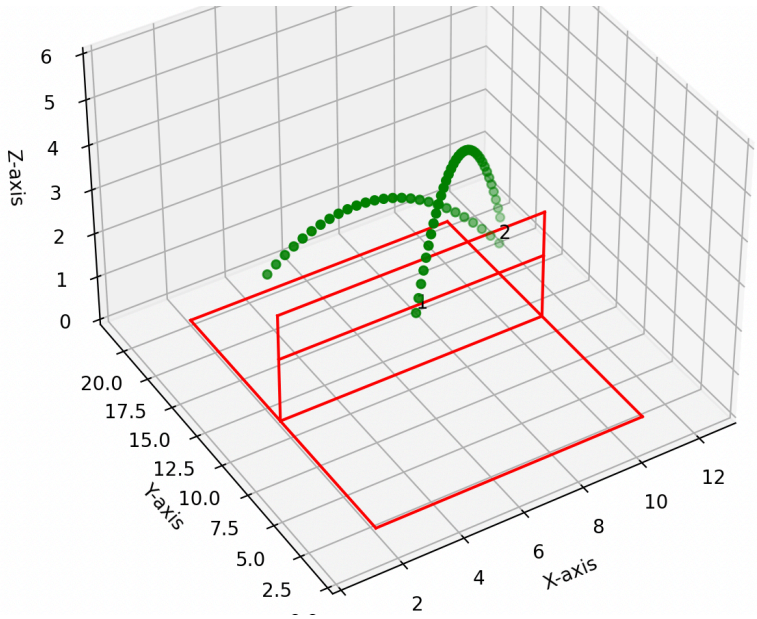
Action 6

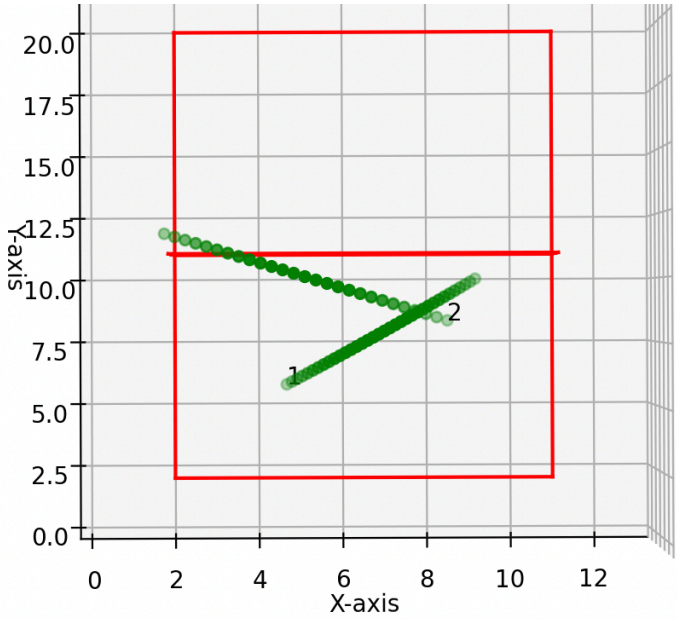
In this case, we see something different: the attack is a block-out, so it hit the block and it goes out of the court.



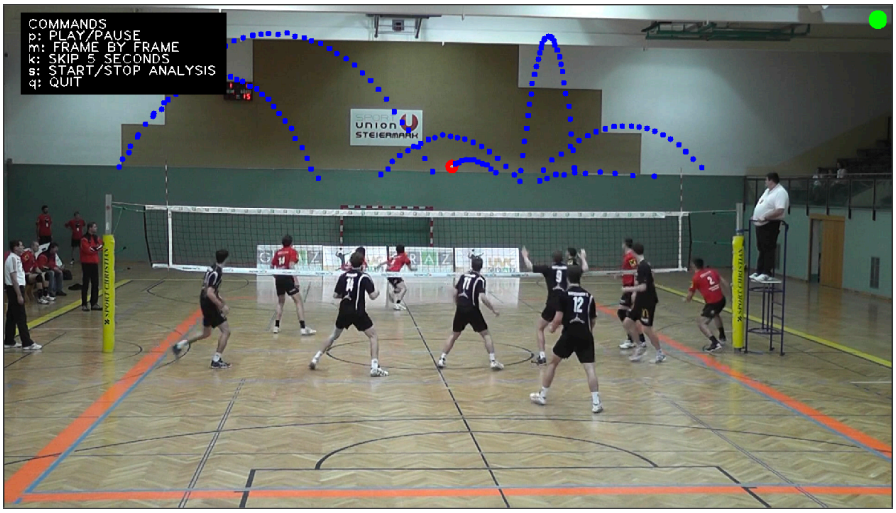
The 2D trajectory is almost perfect, until it reaches the final attack: even if it detect the 2 ball position before the block, the 2 parabolas' intersection is used as mid point and the blue line does not follow the trajectory attack-block-out but it goes attack-out, skipping the block part, which can still be noted with the 2 red dots.

The 3D reconstruction is reasonable, although we can observe again that the two trajectories are slightly disjointed. Number 2 goes just beyond the net before the attack, however it is difficult to confirm if that actually happens in the video.

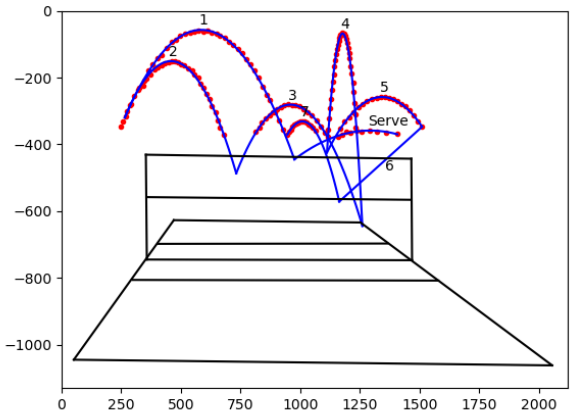




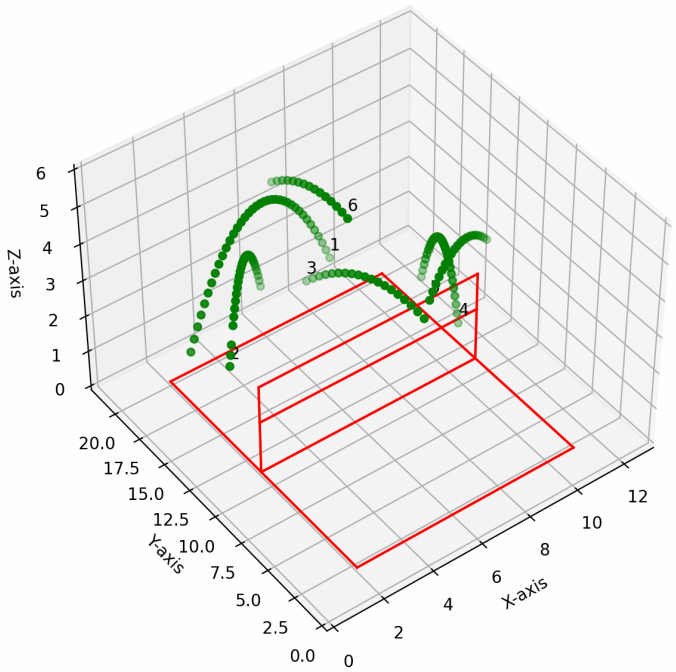
Action 7



This action is the first with 2 attack phases, one for each team. Using the numbers, it is possible to follow the complete trajectory of the ball again, otherwise it would have been really confusing.
Furthermore, we have the case of an attack (#6) defended with a following trajectory (#7): the attack is represented as a line from the last point detected of the set and an extension of the defense trajectory. In this case, the fact that players and the net hide the ball is not an issue, since we exploit geometry and physics of the ball trajectory.



In 3D, between #1 and #2 there's a discontinuity due to and error in the z-axis, which seems the sole inaccuracy of this graph. Between #2 and #3 there's empty space due to the shot being particularly low, but the sub-trajectories line up nicely. This time there was no attack so we could reconstruct the trajectory #3.
Same thing for #4 and #5 which appear to be precise. The attack is not present due to being an interpolated line (as we previously discussed), while trajectory #6 is the bounce after the attack which happens to be in the ROI and is estimated properly.



Conclusion

The project showcased the potential of computer vision techniques in sports analysis. The ability to automatically extract the 3D ball position from video footage could allow for real-time analysis and enhanced training strategies. This can significantly benefit players, trainers, and researchers in the volleyball domain and can extend to other sports as well.

In this project we successfully analysed the ball trajectory in volleyball videos and reconstructed its 2D and 3D coordinates using exclusively computer vision methods. The ball tracking using a Region of Interest (ROI) has proven to be accurate in every case we tested, allowing us to reconstruct the 2D trajectory reliably
The 3D coordinates obtained are in most cases realistic, comparing them to our expectations. It is however difficult to prove their accuracy since we only have one camera angle at the back of the court and we can only infer the approximate depth of the ball.

We pointed out the limitations of the chosen approach in 3D reconstruction, especially in the serve trajectory. However, improvements are necessary to overcome the problems such as occlusion, rapid ball movement, video quality and frame rate. Future research could explore alternative approaches, such as deep learning techniques, to address these challenges and further advance the field of sports analysis.