POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Tracking Volley Ball

IACV PROJECT REPORT

Author: **Lynda Attouche**

Student ID: 10911633
Supervisor: Prof. Vicenzo Caglioti
Academic Year: 2022-2023

Contents

Contents **iii**

1 Introduction **1**

2 Problem Formulation **3**

3 State of the art **5**

4 Implemented solution **7**
4.1 Data collection 8
4.2 Frames processing: stabilization 8
4.3 Ball Detection 10
4.3.1 Background substraction: Mixture of Gaussian 10
4.3.2 Frame segmentation 10
4.3.3 Ball object filtering 11
4.4 Ball tracking 13
4.4.1 2D trajectory 13
4.4.2 3D trajectory 14

5 Results **15**

6 Conclusion **19**

Bibliography **21**

A Appendix **23**

1 | Introduction

The immersion of AI and in particular computer vision in everyday life affects several sectors and one in particular: sports. Thanks to various advanced algorithms and methods this technology can analyze and extract insights from large amounts of data. What's more, the analyses and insights gained are very difficult to formulate and obtain in the traditional ways that we used to do (via manual observations and through sports experts).

One of the main practices is the tracking of objects (balls, baseballs, bats, etc.) and/or players and thus extracting their movement and behavior in real-time. This enables us to get highlights in an optimal way such as player performances, technical aspects, and different game strategies. It also allows highlighting highlights and statistics of a match, for example, points scored, fouls committed, etc.

Research studies have arisen and are expanding in the area of computer vision in sports, showing the potential and importance of integrating this technology. Examples include the analysis of football team performance (Liu et al., 2019) and the analysis of basketball tactics during a game (Chen et al., 2017).

Having a career as a volleyball player and volleyball being a sport that fascinates me since I was a kid, I wanted to combine the world of computer vision, which I am discovering and which interests me more and more, and volleyball in this project. A specific question has been handled and will be presented in this report. The latter will be structured to introduce the problem and review some existing work. It will also outline the method and process developed to address our problem step by step. It will be concluded with the final results and some openings for extending the project.

2 | Problem Formulation

As mentioned previously I chose to focus on volleyball as a sport. And the goal is to develop a system (tool) based on computer vision to detect and track the ball from videos taken by a camera (single view).

This can be summarized in 3 main sub-objectives:

1. Ball detection
2. Ball tracking (2D trajectory)
3. 3D reconstruction of the trajectory

3 | State of the art

Several works and research concerning detection and tracking in sports and particularly volleyball can be found in the literature. Different approaches have been developed to best address these two issues and whether it is for a single object (e.g. the ball) or several (ball, players, referee,...), single view, or multiview. They also differ in the techniques developed. Some studies are based on image processing, deep learning, machine learning, etc. The articles found are quite explicit at the theoretical level but more implicit at the implementation level and during this project they were most useful.

To illustrate the discourse above, in what follows some solutions to the detection and tracking problems from different papers will be introduced.

Firstly, in [1], authors treat Multiple object tracking using kalman filter and optical flow. They follow an approach based on preprocessing including techniques such as overflow, Kalman filter, passing through homographic transformations, but also weighting procedures of the movements as weights for colors. The use of optical flow technique was to measure the movement of pixels in sequential frames, and the Kalman filter to predict the movement of players in real-time. The system was able to accurately track player movement in real time, providing valuable information for performance analysis and game strategizing..

Another study has been studied and which concern basketball[2]. This study presents a sports tracking system using the mean shift algorithm. After using image processing techniques to extract features from the volleyball sequences as in the previous study, the authors applied the mean shift algorithm to track the ball movement in real time. This algorithm is based on the fact that the tracking box will move until surrounds the object of interest and this over the frames.

The algorithm is quite interesting in practice and powerful but when you have a noisy background the preprocessing part should be really successful and usually, we also work on the preprocessing of the frame colors.

The last article [3] is quite recent and contains a very comprehensive and rich review. In a very explicit way, the authors of this article explain the summaries of the new methods of machine learning (Support Vector Machine, K-Nearest Neighbors, Naive Bayes), deep learning (all that is Convolution Neural Networks,Recurrent Neural Networks but also transformers).

The advantages of these new methods can be easily distinguished. The power of transformers and deep learning in computer vision is growing rapidly. With all models, however, it is essential to have a sufficiently large dataset to train them. Otherwise, the performance will be less good and the error rate will be much higher.

The papers presented were those that had the greatest impact on the proposed solution, notably the first and part of the second.

4 | Implemented solution

The implemented solution, which will be explained in this report, can be divided as follows:

1. Data collection: it includes the videos on which the system will perform the detection and which it will perform.
2. Stabilization of the video if needed concerns the speed and small instabilities that improve the quality.
3. Processing of the video frames: this step consists in preparing the frames so that the detection algorithm can give the best rendering. The better the processing is done and adapted to the detection system, the more accurate the detection will be.
4. Detection of the moving ball using the Gaussian Mixture Model(GMM) and some traditional methods (pre-defined filters) and criteria.
5. 2D trajectory generation using Kalman filter and analysis.
6. 3D trajectory reconstruction/Extraction.

The process can be summarised in the following pipeline:

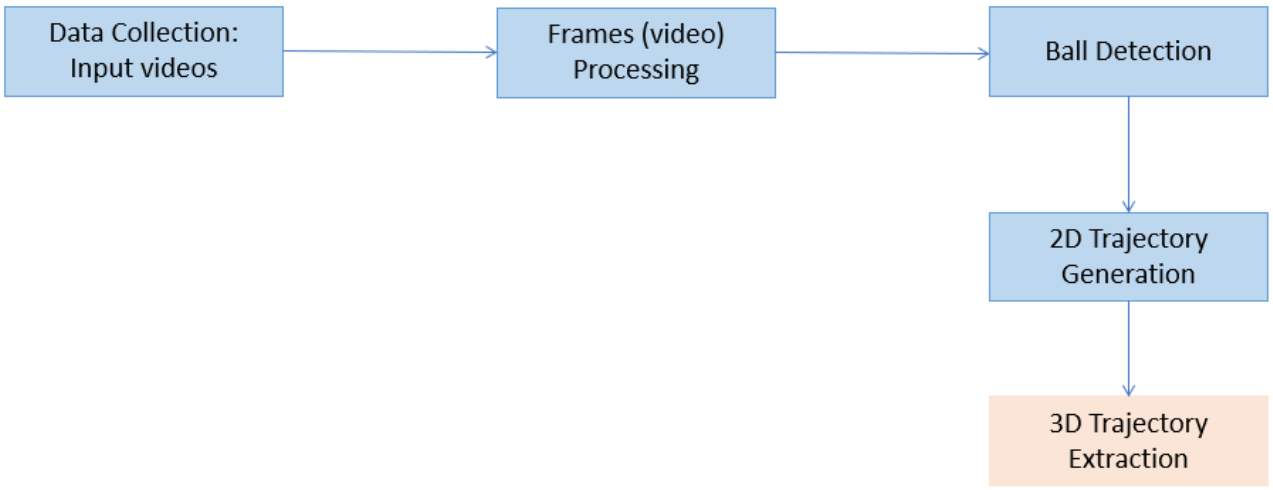

Figure 4.1: Pipeline of solution for Volley ball tracking

In the following sections, each of the above steps will be detailed in terms of theory and implementation. The latter was done using Python as a programming language and OpenCV for the Computer Vision functions and methods.

4.1. Data collection

The first thing that has been done in this project is data collection. In particular, the search for and gathering of volleyball videos (in this case of games). A Russian championship volleyball video website is available for computer vision tasks. The videos used in this project were collected from this site. The goal was to have various videos with different ball movements. The only drawback is that when using internet data, it is unlikely to give accurate information that can be used for camera calibration. This issue had an impact on the 3D reconstruction of the trajectory, however, the work was conducted with this in mind. In 4.2e a frame from a selected video is presented.

Figure 4.2: Frame of volleyball game video

4.2. Frames processing: stabilization

To achieve accurate and reliable results for ball detection and tracking, it is important to ensure that the video is stabilized in real-time. When the ball in motion is recorded, there may be parasitic movements, such as oscillations, shakes, and swinging movements. These movements can affect the quality of the video and make it more difficult to track the ball. Therefore, stabilization of the video is essential to ensure that the data obtained is accurate and reliable.

The stabilization that has been applied to the videos in this project is based on transformations of the motion homography. It is used as a description of how the ball moves relative to the camera.

A homographic matrix can be represented as a 3x3 matrix, such as:

$$\mathbf{H} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}$$

Where each element h_{ij} is a real number.

A point in a plane can be represented by a column vector $[x, y, 1]^T$. The homographic transformation of such a point can be obtained by multiplying the homographic matrix by the vector:

$$\begin{bmatrix} x' & y' & w' \end{bmatrix} = \mathbf{H} \begin{bmatrix} x & y & 1 \end{bmatrix}$$

Where x' and y' are the coordinates of the transformed point and w' is a normalisation coefficient. To get the coordinates of the normalised point, we need to divide the coordinates by w':

$$\begin{bmatrix} \frac{x'}{w'} & \frac{y'}{w'} & 1 \end{bmatrix} = \begin{bmatrix} \frac{h_{11}x+h_{12}y+h_{13}}{h_{31}x+h_{32}y+h_{33}} & \frac{h_{21}x+h_{22}y+h_{23}}{h_{31}x+h_{32}y+h_{33}} & 1 \end{bmatrix}$$

This is how a homographic matrix can be used to describe a projective transformation between two planes.

The code for this operation is as follows:

```
# Define the motion model
warp_mode = cv2.MOTION_HOMOGRAPHY

# Compute the enhanced correlation coefficient transform between the previous
    and current grayscale frames

(cc, warp_matrix) = cv2.findTransformECC(prev_gray, gray, warp_matrix,
    warp_mode, termination_criteria)

# Use the warpPerspective function to stabilize the frame
stabilized = cv2.warpPerspective(frame, warp_matrix, (width, height),
    flags=cv2.INTER_CUBIC + cv2.WARP_INVERSE_MAP)
```

The function *cv2.findTransformECC* in Python uses a similar approach that we have seen previously to calculate the homographic matrix between two images. It uses a code error correction, namely, ECC, algorithm to find the best match between the key points in the two images and then uses these matches to calculate the homographic matrix that minimizes the error between the images (previous frame and current frame). Next, the *cv2.warpPerspective* function is used to apply the homographic matrix to the frame image to produce a stabilized one.

4.3. Ball Detection

This step can be divided into sub-steps:

4.3.1. Background substraction: Mixture of Gaussian

The first step is the subtraction of the background with a method based on gaussian multi-layers observation. It allows object detection by subtracting the background in a video, it acts as a first filter.

This kind of algorithm also has the property of adapting to the variation of lighting in the image which makes the detection robust.

This method has been implemented via the following code:

```
# Create a background subtractor using the MOG2 method
fgbg =cv2.createBackgroundSubtractorMOG2()
# Convert the stabilized frame to grayscale
frame = cv2.cvtColor(stabilized , cv2.COLOR_BGR2GRAY)

# Apply background subtraction to the grayscale frame
fgmask = fgbg.apply(frame)
```

The algorithm uses a statistical model to represent the background. It is based on a weighted average of the background pixels, so that the most frequently observed pixels are weighted more heavily. And the pixels that deviate from the obtained average represent pixels of moving objects (ball, player, referee,..etc). The model is updated in real time using motion detection algorithms.

All pixels in the image are compared to their state in the previous model to detect motion. Pixels that show significant motion are updated in the model, while pixels that do not show significant motion are weighted more heavily in the average.

4.3.2. Frame segmentation

In order to better detect the objects in the image (frame video), techniques aimed at the segmentation of the image have been applied. These techniques include noise elimination methods (to eliminate isolated pixels in the image that may just disturb the detection of the edges).

Then, we want to keep only the most important contours (i.e. objects) in the image. The chosen method uses a combination of thresholding, gradient and non-maximum suppression to detect the most important edges in the image.

And finally, a last filter has been applied following a binary mesh. This step was added because despite the first two methods some points stood out even though they represented the ceiling, the floor, etc. The method allows to keep only the pixels with a colour higher than 128.

At the end of these steps we obtain an image containing only the detected edges and no "disturbing" pixels. And this image will be used to detect the contours of the frame image.

Those steps can be summarized in the following code lines:

```
# Perform morphological opening on the resulting mask
fgmask = cv2.morphologyEx(fgmask, cv2.MORPH_OPEN, kernel)

# Apply Canny edge detection on the foreground mask
e = cv2.Canny(fgmask, 50, 190, 3)

# Threshold the image to keep only edges (value 0 to 127 will be set to 0
    and values 128 to 255 will be set to 255)
_, fgmask = cv2.threshold(e, 127, 255, 0)
```

It is important to note that the parameters used have been fine-tuned to obtain the best results. They may vary depending on the video used.

4.3.3. Ball object filtering

Now, certainly we have removed the background and only the foreground objects remain. But this is not sufficient. Indeed, we have objects and no only one object remains. That is, in our current frame image after the previous steps we have the ball, the players and any objects that can move. As our goal is to detect only the ball, it is necessary to use other filters to distinguish it. i.e. distinguish a blob (binary object) representing a ball from any other object.

To do this, the geometrical properties of the ball have been put forward and exploited. They can be listed as follows:

- **The circularity filter:**

  We know that a ball is spherical/circular in shape. In the video, it happens to appear as such and therefore we need to build a filter that can detect the ball according to its circularity (shape). To do this, it is simple, the circularity is given by:

  $$circularity = 1 - \frac{4\pi A}{p^2}$$

  where, A is the area of the object and p its perimeter. Now if the value of the circularity of a detected object is between 0.3 and 0.8, the object is a candidate to be a balloon.

- **Eccentricity filter:**

  A ball can appear as an ellipse in the video because of its motion and the caption. This means that if we restrict ourselves to circularity, the ball will not be detected when it is elliptical. So adding this new filter is mandatory. The ellipse shape is defined by a value called eccentricity. The eccentricity of an ellipse is a parameter

that measures the flatness of the ellipse. More precisely, it is the ratio of the distance between the center of the ellipse and the furthest point on the ellipse and the semi-major axis of the ellipse. The formula to calculate it is as follows:

$$eccentricity = \frac{c}{a}$$

where a is the semi major axis and c is the distance from the center of the ellipse to either focus.

The value varies from 0 for a circle (where the focal lengths are equal and superimpose on the centre) to 1 for a parabola (where the focal lengths are infinite). And in our case we keep only the ones under 0.7.

- **Size filter:**

  The last filter is the size of the object. Once the two previous filters have been applied, there may be objects remaining (they have passed the filters) that do not represent the ball (hands for instance). It is therefore necessary to play with the size of the ball and to eliminate any object inferior to a certain value, a value which is finetuned several times to obtain the best one.

This step of filtering detection is implemented and can be checked in the Appendix.

At the end of the steps mentioned in this section, the ball has been detected and the tracking can be conducted. However, there is a point that should be taken into account, in some frames, other objects (rarely) are detected instead of the ball. In spite of the implemented methods, some of them are passed (like some players' faces). But again, this is not the case in overall.

4.4. Ball tracking

4.4.1. 2D trajectory

Our work is based on 2 dimensions, the x direction of the ball and the y direction. So to track it and generate the trajectory, we have to consider the movements along x and along y.

The intuition, which in fact is correct, is as follows:

- **According to the x-direction:** The ball moves horizontally in a straight line from left to right (whether it is moving in the same camp or towards the opposite camp).
- **according to the y-direction:** The ball must either follow a parabolic trajectory (when receiving, passing, serving, etc.) or a straight trajectory (when smashing, or smashed serve).

We can observe in the figure 5.4 the ball detected through the videos according to the number of frames. It confirms the above mentioned behavior.

In order to detect in a continuous way the straight lines or parabolas that form the blobs representing the ball during the video we use a method called: **Kalman filter**, described in [4]. It is a method used to estimate the position and velocity of a moving object.

It is an incremental method that uses sequential measurements to estimate the states of a system. Thus, when a ball is not detected in an image, the Kalman filter can be used to estimate its position in the next frame using the previous position of the ball and an estimate of its velocity. This allows to preserve a continuous estimation of the balloon position, even when it is not visible in the frame image. By applying the Kalman filter we can have this continuous trajectory of the ball.

The Kalman filter is based on a kind of modelling of our system to estimate the states of the ball. The ball can be represented by its position and its speed. To do this it goes through two stages:

1. **Prediction step**: it estimates the states at a given time using the previous states and an estimate of the acceleration (the acceleration being the ratio between the change in speed of a mobile and the time needed to make this change in speed).
2. **Optimisation step:** to correct the estimates. This second step is not performed if the estimates (predictions) are verified.

This method has explained and guided for implementation using this webside The results of this operation can be illustrated in 5.5

4.4.2. 3D trajectory

Now that we have the 2D trajectory of the ball, the goal is to still reconstruct the 3D trajectory. In order to answer this question there are two types of methods:

1. **From the actual position of the ball:**

   This implies having more information about the camera (which has to be calibrated) and its position in reality. In this case, having the 2D trajectory, it will be enough to backproject the image of the straight line or parabola representing the ball's trajectory. This implies the calibration of the camera, introduction of new concepts and notions (horizon, vanishing point, backprojection,...etc). In the context of this project, as we do not have the necessary information for the calibration of the camera nor its location in the real world, it is impossible to reconstruct the exact trajectory of the ball in the real world.

2. **Using physic properties:**

   During a volleyball rally, the ball forms a trajectory that is either parabolic from a reception, pass or some serve or a straight trajectory from a spike or serve.

   We will use this information to reconstruct the z-coordinate of a point and thus generate the trajectory in 3D.

   Given the intial position of the 3D ball X_0, Y_0, Z_0 with velocity (V_x, V_y, V_z and the gravitational force g, the 3D of ball coordinate at iteration t is obtained as follows:

   $$X_t = X_0 + tV_x$$

   $$Y_t = Y_0 + tV_y$$

   $$Z_t = Z_0 + tV_z + \frac{1}{2}gt^2$$

   For each point in 2D, the corresponding point in 3D is calculated using the equations above and then the 3D trajectory can be generated.

   Another option will be to use a parabolic trajectory using the coordinates of a points, the trajectory will be then given by:

   $$T : ax^2 + bx + c$$

   where,a,b,c are the coefficients of the parabolic.

   This last method has been implemented using *curve_fit*, which is a python method that allows fitting the data point to a chosen function, in our case a parabolic curve and thus by minimizing the sum of the squares of the differences between the y coordinate and the value calculated using the parabolic function.

   In the case of a smash for instance, where the trajectory is a straight line, the fitting is following the equation:

   $$T : ax + b$$

   where,a and b are the coefficients of the line.

5 | Results

In this section the results obtained in each step will be presented, a frame of a serving sequence has been taken.

Let's start with the processing part, the images below represent the steps in order

Figure 5.1: 1. Background substracting using Mixture of Gaussian

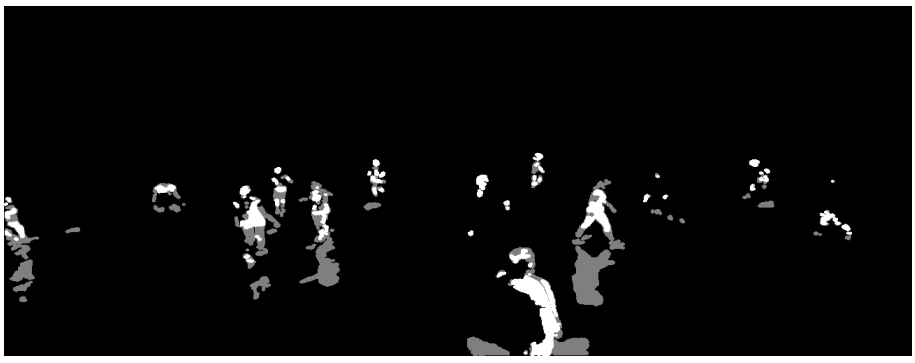

Figure 5.2: 2. Edge detection

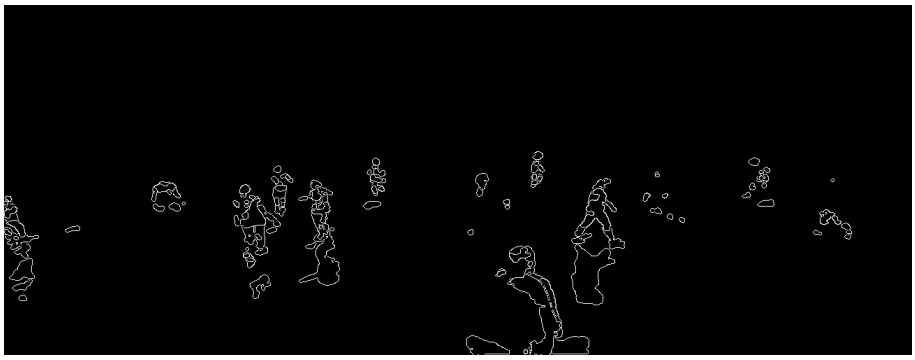

Figure 5.3: 3. Thresholding

Then comes the ball detection, below are the results obtained illustrating the ball candidates detected during the filtering following x-direction and y-direction:

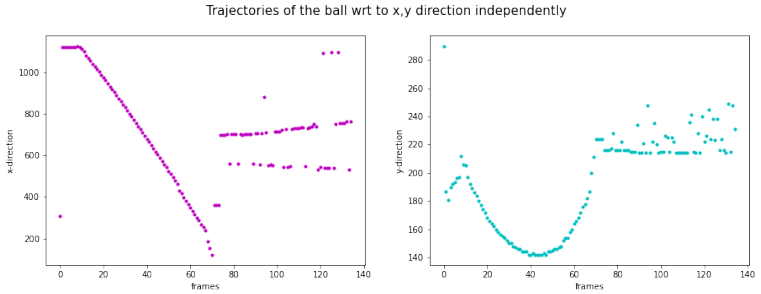

Figure 5.4: Trajectories of the ball wrt to x,y direction

As seen before, Kalman filter has been used to have a kind of continuity by estimation, the results of this operation is presented in the following figure

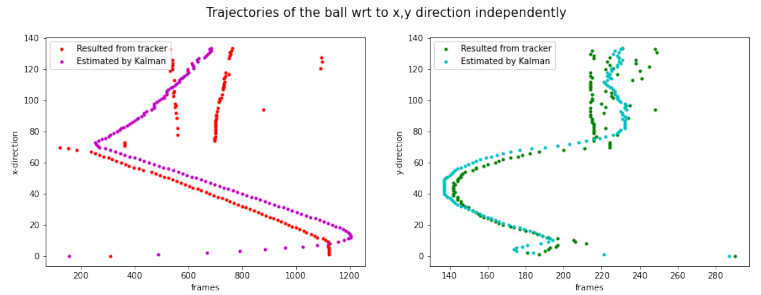

Figure 5.5: Trajectories of the ball wrt to x,y direction - Kalman filter result

After that comes the part where the 2D trajectory has been generated and it is illustrated below (the y xis has been inverted to have the curve in the right direction)

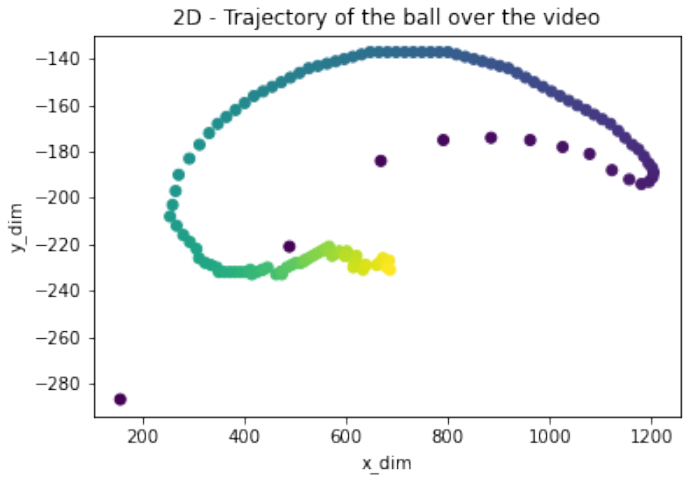

Figure 5.6: 2D Trajectory of the ball

And finally, below is the final result of the 3D ball trajectory

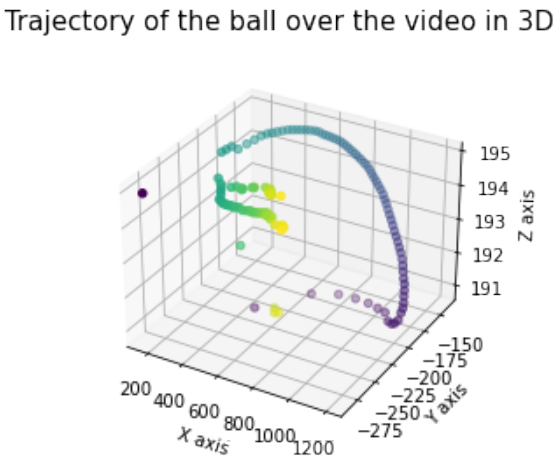

Figure 5.7: 3D Trajectory of the ball

6 | Conclusion

In conclusion, in this project we developed a solution to track the movement of a volleyball using image processing and pattern recognition algorithms. We collected data using video and used it to plot the trajectory of the ball in space.

The proposed solution has limitations, however, including inaccuracy in detection and tracking. A future work will be to test "revolutionary methods" from AI such as Convolutional Neuron Networks and transformers which have proven their efficiency. I find it interesting to explore this vision of things. But, also, another option is to change the data type. Instead of using internet data, one can record a volleyball game with its own device and use the real world reconstruction of the trajectory.

Finally, this project also allowed me to apply the image processing and computer vision methods I learned during the course and also to combine this with my passion, volleyball.

Bibliography

[1] Shantaiya, S. Verma, Kesari Mehta, Kamal. (2015). Multiple object tracking using kalman filter and optical flow. European Journal of Advances in Engineering and Technology. 2. 34-39.

[2] Margarat, G. Simi and S. Sivasubramanian. “Moving Basket Ball Detection and Tracking System by ,different Approaches.” (2019).

[3] D. He, L. Li and L. An, "Notice of Violation of IEEE Publication Principles: Study on Sports Volleyball Tracking Technology Based on Image Processing and 3D Space Matching," in IEEE Access, vol. 8, pp. 94258-94267, 2020, doi: 10.1109/ACCESS.2020.2990941.

[4] Q. Li, R. Li, K. Ji and W. Dai, "Kalman Filter and Its Application," 2015 8th International Conference on Intelligent Networks and Intelligent Systems (ICINIS), Tianjin, China, 2015, pp. 74-77, doi: 10.1109/ICINIS.2015.35.

A | Appendix

Kalman filter

```
import numpy as np
import matplotlib.pyplot as plt

class KalmanFilter(object):
    def __init__(self, dt, a_x,a_y, x_measurement_std, y_measurement_std,
        acceleration_std):
        """
        :param dt: time step
        :param a_x: acceleration in x-direction
        :param a_y: acceleration in y-direction
        :param x_measurement_std: standard deviation of the measurement (in
            x-direction)
        :param y_measurement_std: standard deviation of the measurement (in
            y-direction)
        :param acceleration_std: stadard deviation of the acceleration
        """

        # Time step
        self.dt = dt

        # Acceleration state (defined using acceleration wrt x-direction and
            y-direction)
        self.acc= np.matrix([[a_x],[a_y]])

        # Intial State
        self.x = np.matrix([[0], [0], [0], [0]])

        # Transition Matrix A
        self.A = np.matrix([[1, 0, self.dt, 0],[0, 1, 0, self.dt],[0, 0, 1,
            0],[0, 0, 0, 1]])


        # Control Input Matrix B
        self.B = np.matrix([[(self.dt**2)/2, 0],
                            [0,(self.dt**2)/2],
                            [self.dt,0],
                            [0,self.dt]])
```

```
        # Mapping State to Measurement
        self.H = np.matrix([[1, 0, 0, 0],
                            [0, 1, 0, 0]])

        # Process Covariance
        self.Q = np.matrix([[(self.dt**4)/4, 0, (self.dt**3)/2, 0],
                            [0, (self.dt**4)/4, 0, (self.dt**3)/2],
                            [(self.dt**3)/2, 0, self.dt**2, 0],
                            [0, (self.dt**3)/2, 0, self.dt**2]]) *
                               acceleration_std**2

        # Measurement Covariance
        self.R = np.matrix([[x_measurement_std**2,0],
                            [0, y_measurement_std**2]])

        # Covariance Matrix, initialized to identity
        self.P = np.eye(self.A.shape[1])

    def predict(self):

        # State update
        self.x = np.dot(self.A, self.x) + np.dot(self.B, self.acc)

        # Error covariance
        self.P = np.dot(np.dot(self.A, self.P), self.A.T) + self.Q
        return self.x[0:2]

    def update(self, z):

        # Updates
        S = np.dot(self.H, np.dot(self.P, self.H.T)) + self.R

        # Kalman Gain
        K = np.dot(np.dot(self.P, self.H.T), np.linalg.inv(S))

        self.x = np.round(self.x + np.dot(K, (z - np.dot(self.H, self.x))))

        I = np.eye(self.H.shape[1])

        # Covariance matrix
        self.P = (I - (K * self.H)) * self.P


        return self.x[0:2]
```

Python code

```
import sys
import cv2
import math
import re
import numpy as np
import matplotlib.pyplot as plt
import os
from scipy.optimize import curve_fit
from KalmanFilter import KalmanFilter #from python script
dir_videos = r'C:\Users\Lynda\Documents\Volley Ball Tracking/Videos/'
dir_files = r'C:\Users\Lynda\Documents\Volley Ball Tracking/Data_points/'

video = dir_videos +"para.mp4"
Ball Detection & Tracking (2D)
# Open the video file using OpenCV's VideoCapture function
cap = cv2.VideoCapture(video)

# Get the first frame as reference
ret, prev = cap.read()

# Resize the first frame to 1300x500 (not mondatory only for laptop 13") and
    convert it to grayscale
prev_gray = cv2.resize(cv2.cvtColor(prev, cv2.COLOR_BGR2GRAY),(1300,500))

# Create a structuring element in the shape of an ellipse with a size of (5, 5)
kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(5,5))

# Create a background subtractor using the MOG2 method
fgbg =cv2.createBackgroundSubtractorMOG2()

# Create a list to store ball
ball = []

# Define the motion model
warp_mode = cv2.MOTION_HOMOGRAPHY

# Set the termination criteria for the optical flow algorithm
termination_criteria = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10,
    1e-3)

# Initialize the connected components (cc) and warp matrix (warp_matrix) to
    None
cc, warp_matrix = None,None

# Initialize a tuple 'tmp' with values (0,0)
tmp = (0,0)
```

```
# Read the video frame by frame
while 1:
    # Capture the current frame
    ret, frame = cap.read()

    # If the video has ended, break the loop
    if not ret:
        break

    # Resize the frame to 1300x500 pixels
    f = cv2.resize(frame,(1300,500))

    # Also resize the original frame to 1300x500 pixels
    frame = cv2.resize(frame,(1300,500))


    # 1. Frame Processing
    # Convert the current frame to grayscale
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Perform histogram equalization on the grayscale frame
    frame_gray = cv2.equalizeHist(gray)

    # Compute the enhanced correlation coefficient transform between the
        previous and current grayscale frames
    (cc, warp_matrix) = cv2.findTransformECC(prev_gray, gray, warp_matrix,
        warp_mode, termination_criteria)

    # Get the height and width of the frame
    height, width = frame.shape[:2]

    # Use the warpPerspective function to stabilize the frame
    stabilized = cv2.warpPerspective(frame, warp_matrix, (width, height),
        flags=cv2.INTER_CUBIC + cv2.WARP_INVERSE_MAP)

    # Convert the stabilized frame to grayscale
    frame = cv2.cvtColor(stabilized , cv2.COLOR_BGR2GRAY)

    # Apply background subtraction to the grayscale frame
    fgmask = fgbg.apply(frame)

    # Perform morphological opening on the resulting mask
    fgmask = cv2.morphologyEx(fgmask, cv2.MORPH_OPEN, kernel)

    # Apply Canny edge detection on the foreground mask
    e = cv2.Canny(fgmask, 50, 190, 3)

    # Threshold the image to keep only edges (value 0 to 127 will be set to 0
        and values 128 to 255 will be set to 255)
```

```
_, fgmask = cv2.threshold(e, 127, 255, 0)

#cv2.imshow('mask',fgmask)



# 2. Ball detection using filtering techniques

# Find contours in the binary image "fgmask"
contours, _ = cv2.findContours(fgmask, cv2.RETR_CCOMP,
    cv2.CHAIN_APPROX_SIMPLE)


# Define a variable that will contain a possible candidate ball
candidate_accepted = None

# Update previous frame
prev_gray = frame_gray


# Loop over contours
for contour in contours:

    # Extract the bounding rectangle coordinates (x, y) of the top-left
       corner and the width (w) and height (h) of the rectangle
       surrounding a contour
    (x, y, w, h) = cv2.boundingRect(contour)

    # Compute the area of the contour
    area = cv2.contourArea(contour)

    # Compute perimeter of the contour
    perimeter = cv2.arcLength(contour, True)

    # Define variable for circularity
    circularity = 0

    # i. Circular shape filter: detect circular blobs

    # Check if the perimeter is not null
    if perimeter !=0:

        # Calculate circularity of the contour
        circularity = 4 *math.pi* area/ perimeter**2

        # Check if the contour has a circularity between 0.3 and 0.8.
           Circularity is a measure of how close the contour shape is to a
           circle
        if circularity >=0.3 and circularity <=0.8:
```

```
                candidate_accepted = contour

        # ii. Elliptical shape filter: detect elliptical blobs

        # Check whether the width of the blob is not null
        elif w!=0:

            # Calculate eccentricity of the contour
            eccentricity = h/w

            # Check if the shape of the contour has a circularity value between
                0 and 0.7, to get elliptical ball
            if eccentricity>0 and eccentricity <=0.7:
                candidate_accepted = contour

        # iii. Size filter: allows removing small blobs

        # Check if the contour area is within the specified range (20 to 60)
        if  area<60 and area>20:
            candidate_accepted = None

    if candidate_accepted is not None:

        # Get the minimum enclosing circle for the candidate object
        (x, y), radius = cv2.minEnclosingCircle(candidate_accepted)

        # Keep track of the ball in current frame
        tmp = (x,y)

        # Add the coordinate of the detected ball to the list
        ball.append(np.array([[x], [y]]))

        # Draw a circle with center at (x,y) and radius 20 in the image f using
            BGR color (155,200,95) and line width 2
        cv2.circle(f, (int(x),int(y)), 20, (155,200,95), 2)

    # Display the current frame
    cv2.imshow('frame',f)

    # Control loop, if 'q' pressed, breaks it
    if cv2.waitKey(10) == ord("q"):
        break


# Release the video capture
cap.release()

# Destroy all open windows.
cv2.destroyAllWindows()
```

```
print("Number of ball locations found is: ", len(ball))
cap = cv2.VideoCapture(video)
print("Number of frame of the video is: ",
    int(cap.get(cv2.CAP_PROP_FRAME_COUNT)))
2D Trajectory
# Define an array of rounded x-coordinates of the ball, obtained from the ball
    list previously found
x_dim = np.array([round(ball[i][0][0]) for i in range(len(ball))])

# Define an array of rounded y-coordinates of the ball, obtained from the ball
    list previously found
y_dim = np.array([round(ball[i][1][0]) for i in range(len(ball))])

# Define an array of the frame number where the ball has been detected
frames_det = np.arange(len(ball))

# Create a figure and axis object using subplots
fig,ax = plt.subplots(1,2,figsize=(15,5))

# Plot the data points from fr_dim and x_dim on the first axis (index 0)
ax[0].plot(frames_det, x_dim, '.', color='m')

# Plot the data points from fr_dim and y_dim on the second axis (index 1)
ax[1].plot(frames_det, y_dim, '.', color='c')

# Set the x-label for the first axis to "frames"
ax[0].set_xlabel('frames')

# Set the y-label for the first axis to "x-direction"
ax[0].set_ylabel('x-direction')

# Set the x-label for the second axis to "frames"
ax[1].set_xlabel('frames')

# Set the y-label for the second axis to "y-direction"
ax[1].set_ylabel('y-direction')

# Add a title to the figure using suptitle
fig.suptitle('Trajectories of the ball wrt to x,y direction independently',
    fontsize=15)
#Apply Kalman to estimate the location of the object (to keep tracking of the
    ball when it disappears)

# Initialize the KalmanFilter object with parameters
KF = KalmanFilter(0.1, 1, 1, 1, 0.1,0.1)

# Create an empty list to store the updates
updates = []
```

```
# Loop through each blob in the list of blobs
for i in range(len(ball)):

    # Get the 2D point for the blob in this frame
    measurement = ball[i]

    # Predict step
    (x, y) = KF.predict()

    # Update step
    (x1, y1) = KF.update(measurement)

    # Convert the resulting x1 and y1 from numpy arrays to floats and append
        to updates
    updates.append([np.array(x1)[0][0],np.array(y1)[0][0]])
# Create arrays for the predicted x and y positions, and the frames of the
    candidates blobs
tracked_positions = np.array(updates)

# Create an array of the predicted x positions
x_estim = np.array([round(tracked_positions[i][0]) for i in
    range(len(tracked_positions))])

# Create an array of the predicted y positions
y_estim = np.array([round(tracked_positions[i][1]) for i in
    range(len(tracked_positions))])

# Create an array of the frames
frames = np.arange(len(tracked_positions))

# Create a figure with 1 row and 2 columns of subplots
fig, ax = plt.subplots(1, 2, figsize=(15,5))

# Plot the predicted x-direction and y-direction trajectory of the candidates
    blobs
ax[0].plot( x_dim,frames, '.', color='r', label = 'Resulted from tracker')
ax[0].plot( x_estim, frames,'.', color='m',label = "Estimated by Kalman")

ax[1].plot( y_dim, frames,'.', color='g', label = 'Resulted from tracker')
ax[1].plot(y_estim, frames, '.', color='c', label = "Estimated by Kalman")

# Add labels for the x and y axis for both subplots
ax[0].set_xlabel('frames')
ax[0].set_ylabel('x-direction')
ax[1].set_xlabel('frames')
ax[1].set_ylabel('y-direction')

# Add legends
ax[0].legend(loc="upper left")
```

```
ax[1].legend(loc="upper left")

# Add a title to the figure
fig.suptitle('Trajectories of the ball wrt to x,y direction independently',
    fontsize=15)
plt.scatter(x_estim, y_estim, c=range(len(x_estim)))
plt.title("2D - Trajectory of the ball over the video")
plt.xlabel("x_dim")
plt.ylabel("y_dim")
# Stack the x and y dimensions of found and estimated into separate arrays
found = np.stack([x_dim, y_dim], axis=1)
estimated = np.stack([x_estim, y_estim], axis=1)

# Save the found and estimated arrays as tab-delimited text files
np.savetxt(dir_files+"parabolicV_balls_detected.txt", found, delimiter="\t",
    header="x\y", comments='')
np.savetxt(dir_files+"parabolicV_balls_estimated.txt", estimated,
    delimiter="\t", header="x\y", comments='')
Trajectory reconstruction 3D
# Define the parabolic equation
def parabolic_curve(x, a, b, c):
    return a * x**2 + b * x + c

# Fit the parabolic curve to the data
popt, pcov = curve_fit(parabolic_curve, x_dim, y_dim)

# Calculate the estimated 3D trajectory
z_dim = parabolic_curve(x_dim, *popt)
# Define the linear equation
def linear_curve(x, a, b):
    return a * x + b

# Fit the linear curve to the data
popt, pcov = curve_fit(linear_curve, x_dim, y_dim)

# Calculate the estimated 3D trajectory
#z_dim = linear_curve(x_dim, *popt)
# Create a 3D plot using the projection argument
fig,ax = plt.figure(), plt.axes(projection='3d')
fig.suptitle("Trajectory of the ball over the video in 3D", fontsize=15)

# plot the 3D scatter plot
ax.scatter(x_estim, -y_estim, z_dim, c = range(len(x_dim)))

# set x, y, and z labels
ax.set_xlabel('X axis')
ax.set_ylabel('Y axis')
ax.set_zlabel('Z axis')
```

```
plt.show()
```