Project 1 – Report

Omkar. A. Chittar

119193556

Q1. In the given video, a red ball is thrown against a wall. Assuming that the trajectory of the ball follows the equation of a parabola:
1. Detect and plot the pixel coordinates of the center point of the ball in the video.
2. Use Standard Least Squares to fit a curve to the extracted coordinates. For the estimated parabola you must,

a. Print the equation of the curve.
b. Plot the data with your best fit curve.

3. Assuming that the origin of the video is at the top-left of the frame as shown below, compute the x-coordinate of the ball's landing spot in pixels, if the y-coordinate of the landing spot is defined as 300 pixels greater than its first detected location.

Solution 1.1 :

We first import the required packages and libraries:

- **cv2 and numpy for image processing**
- **cmath for complex math calculations**
- **matplotlib for visualization purposes**

Approach: Detecting and plotting the coordinates of the center point of the ball in the video:

2. The red channel filtering is done using the **inRange()** function from OpenCV library that generates a binary image consisting of just red pixels.
3. The function **centroid(image)** is defined which calculates the centroid of the ball in a given frame taken as an image. The function takes the image as input, and uses **np.nonzero()** to get the indices of the non-zero pixels in the image, which are then used to calculate the x and y coordinates of the centroid using the formula for centroid. The function returns the coordinates of the centroid.

```
def centroid(image):
  y_values, x_values = np.nonzero(image)

  X = np.sum(x_values)/x_values.shape[0]
  Y = np.sum(y_values)/y_values.shape[0]

  return (int(X), int(Y))
```

4. The ball is detected in the first frame of the video using the **cv2.inRange()** and **cv2.bitwise_and()** functions, and the result is saved in **filtered** and **output** respectively.
5. A loop is created that reads all the frames of the video one-by-one and detects the ball in each frame using the same **cv2.inRange()** and **cv2.bitwise_and()** functions. The x and y coordinates of the centroid of the ball in each frame are saved in the **coordinates** list.
6. The x and y coordinates of all the detected ball in frames are stored in separate lists **x** and **y** and then plotted, giving us the following results.

Results:

Fig 1. Choosing a frame from the video stream

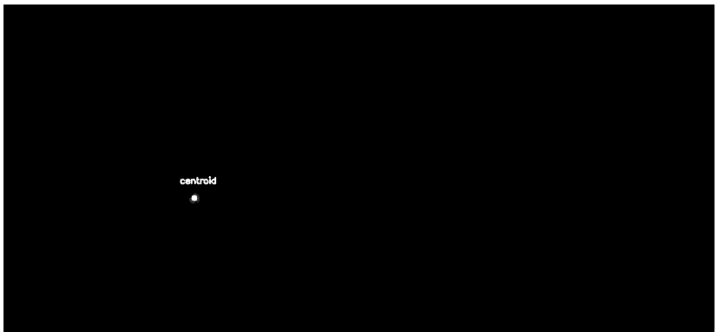

Fig 2. Plotting the centroid of the ball in the selected frame

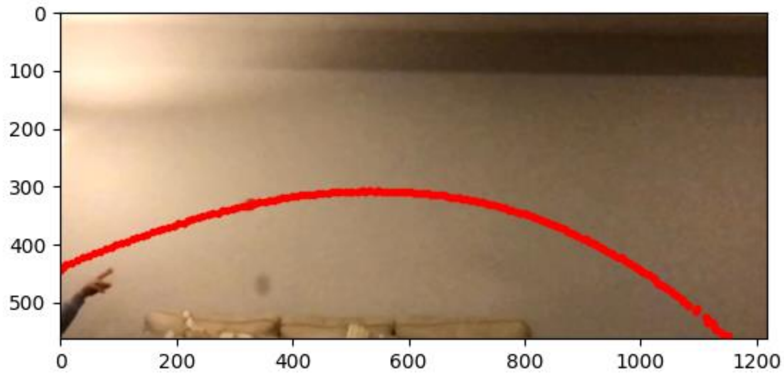

Fig 3. Plotting the centroid of the ball in all the frames

Problems Encountered:

- Noise in the frames due to the hand which also has a lot of red-component in it caused issues while filtering the red channel.
- This was mitigated by introducing components of blue and green in the channeling mask and tuning the values till appropriately fair results were achieved.

Solution 1.2:

Approach: a. Printing the equation of the curve.

1. We define a function named **curve_fit()** which is used to fit a polynomial to a given set of x and y data points using least-squares regression.
2. The function takes as input the x and y data arrays, as well as an optional degree argument that specifies the degree of the polynomial curve to fit (default is 2).
3. We first construct a design matrix X by taking powers of x_data up to the specified degree such that we get the following Matrix.

$$X = \begin{matrix} x1^2 & x1 & 1 \\ x2^2 & x2 & 1 \\ x3^2 & x3 & 1 \\ . & . & . \\ . & . & . \\ . & . & . \\ xn^2 & xn & 1 \end{matrix}$$

4. We then solve the least-squares problem to obtain the coefficients of the polynomial curve.

```
# Solve the least-squares problem
XT_X = np.dot(X.T, X)
XT_Y = np.dot(X.T, y_data)
coeffs = np.linalg.solve(XT_X, XT_Y)
```

5. The function returns the x-values, y-values, and coefficients of the fitted curve. The coefficients are printed out as the equation of the fitted curve.

Results:

```
Equation of the fitted curve is:
y = 458.620351 + (-0.606833)x + (0.000595)x^2
```

Approach: b. Plotting the data with the best fit curve.

1. For plotting the curve we make use of the data obtained from the **curve_fit()** function. The function takes the x and y data and the degree of the polynomial as inputs, and returns the coefficients of the polynomial curve.
2. The data points and the fitted curve are plotted using **plt.plot()** which makes use of the coefficients and the x and y data.
3. The x and y data are converted to numpy arrays, and the **curve_fit()** function is called with the x and y data and a degree of 2 as inputs to fit a quadratic curve to the data.

Results:

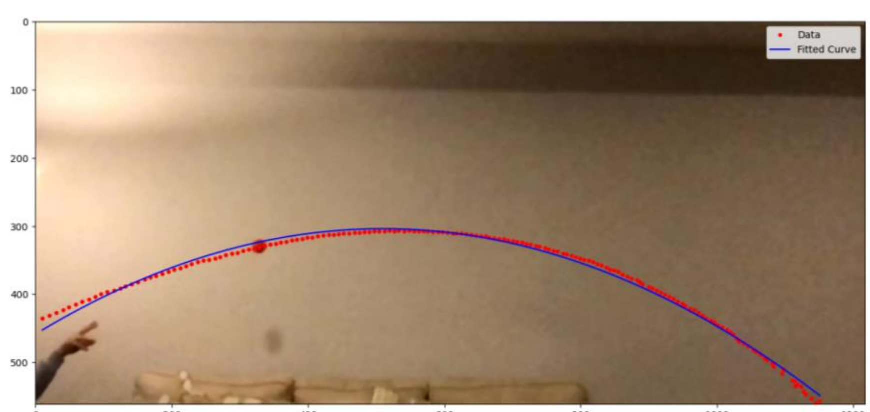

Fig 4. Estimating the Trajectory of the ball

Problems Encountered:

- When just the fitted curve is plotted alone, the origin is assumed to be at the bottom left making the plot to be inverted.
- This can be rectified by adding an image to the plot, as the origin of the image is at the top left.

Solution 1.3:

Approach: For calculating the x-coordinate of the landing spot of the ball,

Calculating Y-coordinate at x=0.

The equation of the fitted curve(parabola) is of the form

y = ax^2 + bx + c.

from the above equation of the parabola we have,

at x1 = 0,

y1 = c = 456.456721

At the landing spot, which is 300 below the initial point

y2 = 300 + y1 = 756.456721

Substituting these values of y2 and c in the equation of the parabola to solve for x2 (landing spot), we get,

ax2^2 + bx2 + c = y2

and solve for x2 using the quadratic formula, we get two roots

```
The roots are
(-367.46284239858704+0j)
(1383.7441983307904+0j)
```

The ball is thrown rightwards away from the origin, which indicates x-positive

Therefore, we select the positive value of the root.

Results:

```
X co-ordinate of the landing spot =  (1383.7441983307904+0j)
```

Problems Encountered:

The problems faced for this part coincided with 1.2. Everything else was pretty straight forward.