Problem 1:

1. Read the video and extract individual frames using OpenCV.

Mounting the google drive

```
#Mount your google drive
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mo
```

Read the video file from the google drive

```
#Import opencv2
import cv2

#Reading the video link path for object_tracking.mp4
object_video_link = '/content/drive/MyDrive/ENPM673/project1/object_tracking.mp4'
```

Check if the video is present and read the individual frames from video

All frames will be stored in frames list defined

```
#Function to read all the frames
def read_frames(video_path):
  #Read video using videoCapture in cv2
  object_video = cv2.VideoCapture(video_path)

  #defining empty list to store all the frames
  frames  = []

  #Iterate util video is open
  while object_video.isOpened():

      # Capturing  frames
      ret, frame = object_video.read()

      # Check if frames are read correctly by reading ret
      if not ret:
          break
      else:
          frames.append(frame)
  return frames

#frames list has all the individual frames
frames = read_frames(object_video_link)
```

2.Loop over each frame to extract the pixels of moving object (Hint: Use color)

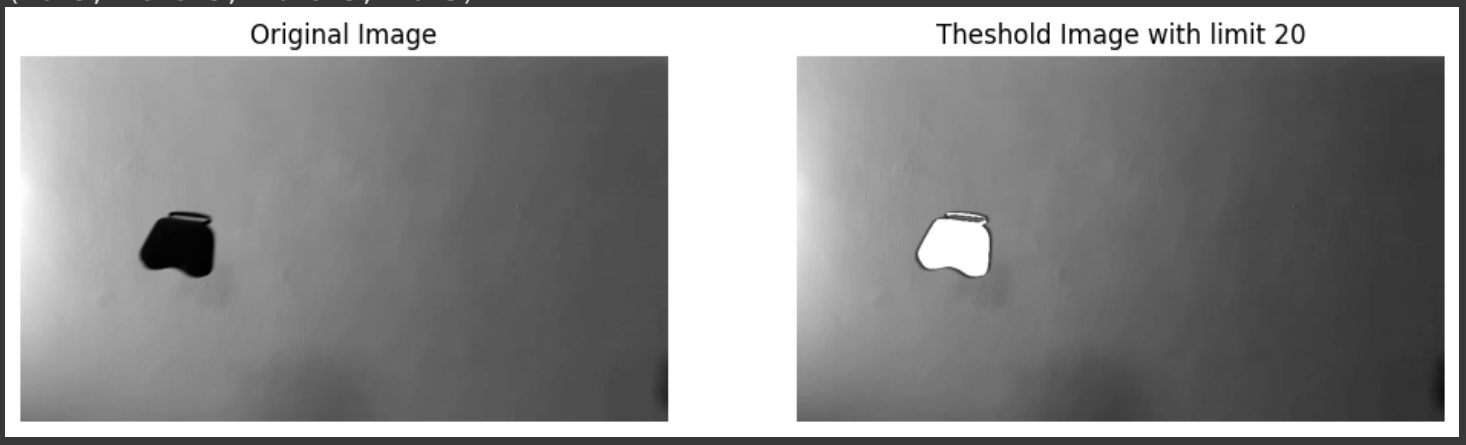
Convert RGB to Grayscale frames for pre processing

```
#defining frames grayscae list
frames_gray = []

#Iterate over the frames and convert to grayscale
for i, frame in enumerate(frames):
  frames_gray.append(cv2.cvtColor(frame, cv2.COLOR_RGB2GRAY))
```

Display any random frame to check the threshold limit for masking

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

# Create 2 plot one for original and another to check thresholing limits
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))

#Random frame
random_frame = 350

#showing random frame
ax1.imshow(frames_gray[random_frame], cmap='gray')
ax1.set_title('Original Image')
ax1.axis('off')

# Perform thresholding using np.where
thresholded_image = np.where(frames_gray[random_frame] <= 20, 255, frames_gray[random_fram

#showing random frame for thresholding purpose
ax2.imshow(thresholded_image, cmap='gray')
ax2.set_title('Theshold Image with limit 20')
ax2.axis('off')
```

```
(-0.5, 1919.5, 1079.5, -0.5)
```



Since the object is black and background is white, creating a binary mask for black moving object and storing black pixel coords to a list

```
import numpy as np
```

```
import numpy as np

#defing object coords
moving_object_coord = []

#Function to give moving object coordinates
def read_pixel_values(frame):
  #Creating a binary mask for black object (intensity<=20)
  mask = frame <= 20

  #Multiply by 255 to create binary image(0-black-->background, 255 white-->object)
  mask = mask*255

 #np.where gives the coords of x and y where the pixel == 255 (white==object)
  object_coords = np.argwhere(mask == 255)

  return object_coords

#Iterate over the frames gray
for i, frame in enumerate(frames_gray):
  tmp = read_pixel_values(frame)

  #Apending numpy array of coord to moving_object_coord
  moving_object_coord.append(tmp)

#Coodinates x,y of object
print(moving_object_coord)
```

```
[array([], shape=(0, 2), dtype=int64), array([], shape=(0, 2), dtype=int64), array([],
       [ 993, 1919],
       [ 994, 1919],
       [ 995, 1919]]), array([[ 992, 1919],
       [ 993, 1919],
       [ 994, 1919],
       [ 995, 1919]]), array([[ 992, 1919],
       [ 993, 1919],
       [ 994, 1919],
       [ 995, 1919]]), array([[ 993, 1919],
       [ 994, 1919],
       [ 995, 1919]]), array([], shape=(0, 2), dtype=int64), array([], shape=(0, 2), d
       [979,   0],
       [980,   0],
       [981,   0],
       [982,   0],
       [983,   0]]), array([[979,   0],
       [980,   0],
       [981,   0],
       [982,   0]]), array([[979,   0],
       [980,   0],
       [981,   0],
       [982,   0]]), array([[ 956,    7],
       [ 956,    8],
       [ 957,    3],
       ...,
       [1059,    0],
       [1060,    0],
       [1061,    0]]), array([[ 956,    7],
       [ 956,    8],
       [ 957,    3],
       ...,
       [1059,    0],
```

```
           [1060,    0],
           [1061,    0]]), array([[ 956,    7],
           [ 956,    8],
           [ 957,    3],
           ...,
           [1059,    0],
           [1060,    0],
           [1061,    0]]), array([[ 956,    7],
           [ 956,    8],
           [ 957,    3],
           ...,
           [1059,    0],
           [1060,    0],
           [1061,    0]]), array([[ 940,   22],
           [ 940,   23],
           [ 940,   24],
           ...,
           [1079,    1],
           [1079,    2],
           [1079,    3]]), array([[ 940,   22],
           [ 940,   23],
           [ 940,   24],
           ...,
           [1079,    2],
           [1079,    3],
```

3.Calculate the centroid of the object in every frame (doesn't have to be very precise).

Reading moving_object_coords and calculating centroid

centroid_x = sum(moving_pixel_coords_x)/len(moving_pixel_coords_x)

centroid_y = sum(moving_pixel_coords_y)/len(moving_pixel_coords_y)

```
import matplotlib.pyplot as plt

#Centroid list
centroid_lst = []

#Function for calculating centroid
def calculate_centroid(black_coord):
  #Calculating centroid ---np.mean axis==0 --> column
  centroid = np.mean(black_coord, axis=0)

  return centroid

#Iterating black object coordinates
for black_coord in moving_object_coord:
  #To remove frames without black coordinates
  if(black_coord.shape[0]!= 0):
    #Function call to find centroid
    centroid = calculate_centroid(black_coord)
    #Appending centroids to a list
    centroid_lst.append(centroid)

print(centroid_lst)

x_coords = [centroid[0] for centroid in centroid_lst]
y_coords = [centroid[1] for centroid in centroid_lst]
```

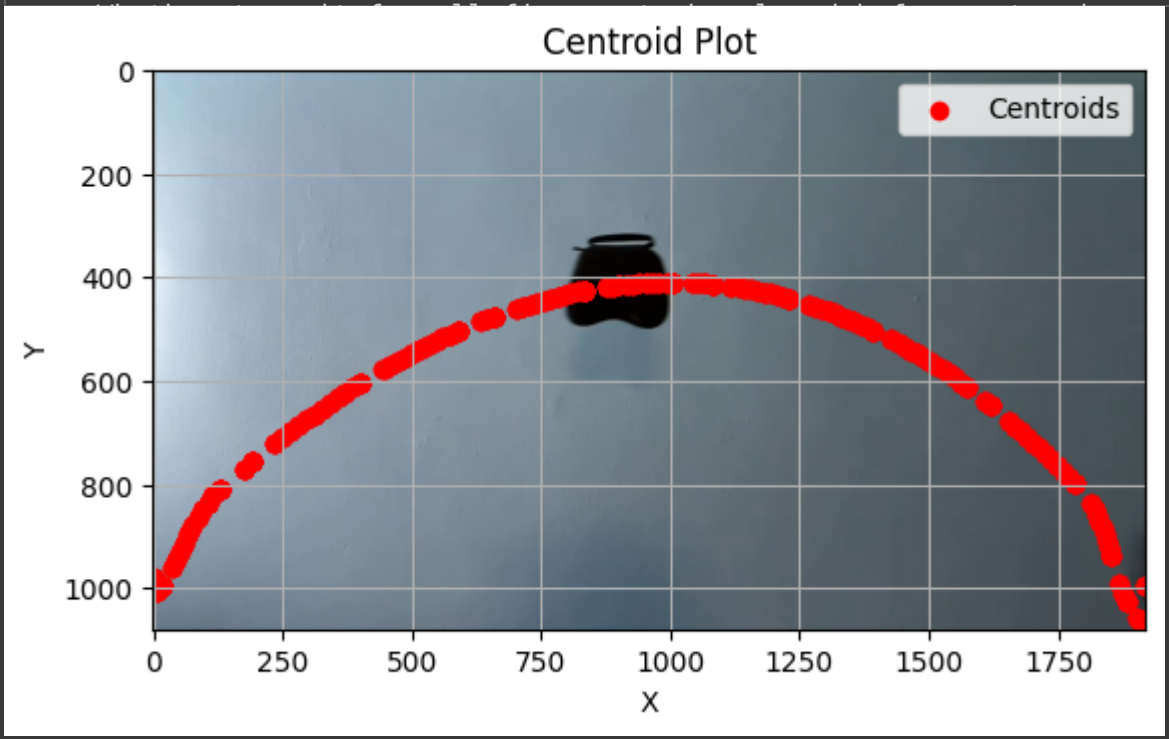
```
# Plot the centroid coordinates

#Take a random frame
plt.imshow(frames[450])
#Here the x and y coords are interchanged since np.where returns x,y assuming x as y in an
plt.scatter(y_coords, x_coords, color='red', label='Centroids')
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Centroid Plot')
plt.legend()
plt.grid(True)
plt.show
```

```
[array([ 993.5, 1919. ]), array([ 993.5, 1919. ]), array([ 993.5, 1919. ]), array([ 99
```

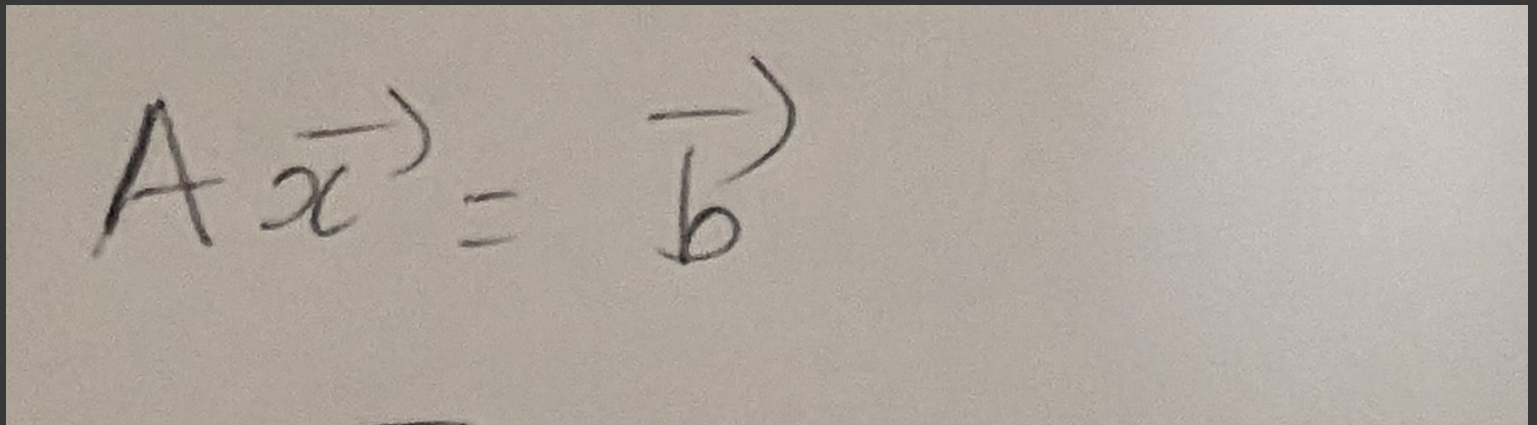
matplotlib.pyplot.show
def show(*args, **kwargs)

/usr/local/lib/python3.10/dist-packages/matplotlib/pyplot.py
Display all open figures.

Parameters
---
block : bool, optional



4. Assume TOP LEFT corner of the frame as 0,0 and accordingly use 'Standard Least Square' to fit a curve (parabola) through the found centroids in part 3.



$$A = \begin{bmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ \vdots & \vdots & \vdots \\ x_n^2 & x_n & 1 \end{bmatrix}$$

$$\vec{x} = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

$$\vec{b} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

A parabola is defined as y=ax2+bx+c

we can write system of equations as Ax = b

For n centroid points we have n number of parabolic equations.Therefore we can write Ax = b as the equation displayed above

Extracting A matrix from the centroid points

col1 represents column1 in A matrix (x^2)

col2 represents column2 in A matrix (x)

col3 represents column in A matrix(ALL ones)

```
#Since np.where interchanged the coords here x_axis = y from np.where
x_axis = np.array(y_coords)

#FInding individual cols of A
col1 = x_axis**2
col2 = x_axis
col3 = np.ones(x_axis.shape[0])

#Matrix A by concat all colmns
A = np.c_[col1, col2, col3]
print(A)
```

```
[[3.68256100e+06 1.91900000e+03 1.00000000e+00]
 [3.68256100e+06 1.91900000e+03 1.00000000e+00]
 [3.68256100e+06 1.91900000e+03 1.00000000e+00]
 ...
 [3.66548557e+06 1.91454579e+03 1.00000000e+00]
 [3.66552451e+06 1.91455596e+03 1.00000000e+00]
 [3.66552451e+06 1.91455596e+03 1.00000000e+00]]
```

Extracting matrix b from centroid points

```
#MAtrix b
b = np.array(x_coords).reshape(-1, 1)
print(b)
```

```
[[ 993.5       ]
 [ 993.5       ]
 [ 993.5       ]
 [ 994.        ]
 [ 980.5       ]
 [ 980.5       ]
 [ 980.5       ]
 [1006.18080357]
 [1006.18080357]
 [1006.21899441]
```

```
[1006.21899441]
[1003.77593521]
[1003.83877159]
[1003.83877159]
[ 997.07024451]
[ 997.07835821]
[ 997.08583869]
[ 997.03418272]
[ 997.03418272]
[ 997.03418272]
[ 997.03418272]
[ 997.03418272]
[ 997.02609776]
[ 997.02609776]
[ 961.44044214]
[ 961.61962334]
[ 961.61376317]
[ 945.83661292]
[ 945.95099784]
[ 945.95099784]
[ 930.67273831]
[ 930.67785754]
[ 930.67785754]
[ 913.1669658 ]
[ 913.12426638]
[ 913.12426638]
[ 896.29701099]
[ 896.26943187]
[ 896.26943187]
[ 896.25142782]
[ 879.00841006]
[ 879.01435235]
[ 878.99573484]
[ 863.68584038]
[ 863.67050888]
[ 863.67054594]
[ 848.86842836]
[ 848.81823546]
[ 848.82656852]
[ 835.5536896 ]
[ 835.63241462]
[ 835.64185365]
[ 835.64185365]
[ 821.98154745]
[ 822.02253829]
[ 822.02552331]
[ 808.93959497]
[ 809.07457703]
```

Solving for coefficients using

beta = (AT.A)−1.(AT.b)

```
# Solve for the coefficients
beta = (np.linalg.inv((A.T) @ A)) @ A.T @ b
beta

# Extract coefficients
a, b, c = beta

# Print the coefficients
print("Coefficients a = {:.4f}, b = {:.4f}, c = {:.4f}".format(a[0], b[0], c[0]))
```

```
# Equation of the fitted parabola
print("Curve fitted Parabolic equation: y = {:.4f}x^2 + {:.4f}x + {:.4f}".format(a[0], b[0
```

```
Coefficients a = 0.0006, b = -1.2391, c = 992.8739
Curve fitted Parabolic equation: y = 0.0006x^2 + -1.2391x + 992.8739
```

5.Given that x axis value is 1000, find the y axis value for calculated equaton in part 4

```
#Given x=1000
x = 1000
#Parabolic equation
y = a[0]*(x**2) + b[0]*x + c[0]
print("When x=1000 , y is:",y)
```

```
When x=1000 , y is: 398.8683933967901
```

6.Capture any one frame from the video (which shows the object) and plot the obtained equaton.

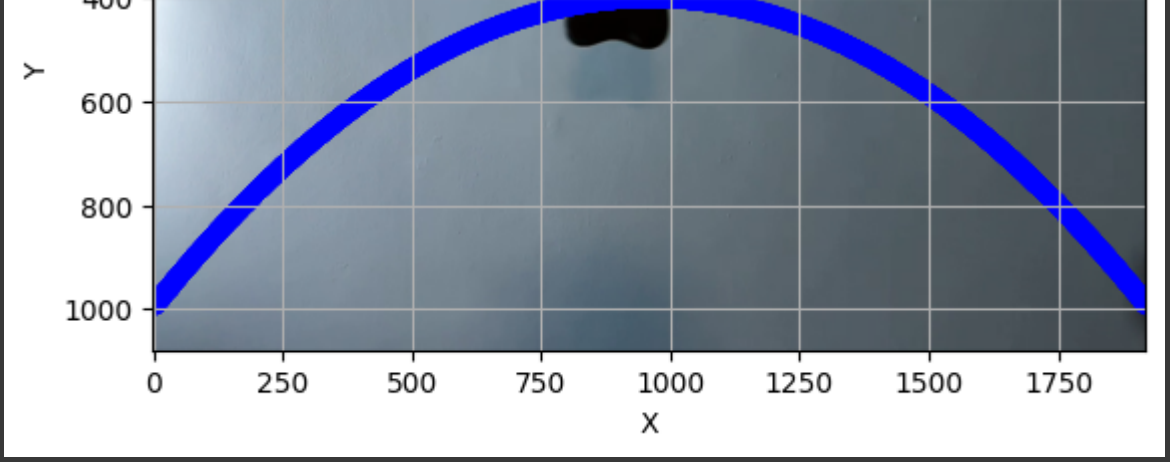
```
# Generate x values with interval of 1
x_values = np.linspace(0, frames[0].shape[1]-1, frames[0].shape[1]-1)  # Adjust the range


print(x_values)

# Compute corresponding y values using the parabolic equation
y_values = a[0] * x_values**2 + b[0]* x_values + c[0]

#Taking random frame
plt.imshow(frames[450])
#Scatter plot parabola
plt.scatter(x_values, y_values, color='blue')
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Fitted Parabola')
plt.legend()
plt.grid(True)
plt.show
```

```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note that ar
[0.00000000e+00 1.00052138e+00 2.00104275e+00 ... 1.91699896e+03
 1.91799948e+03 1.91900000e+03]
```

matplotlib.pyplot.show
def show(*args, **kwargs)

/usr/local/lib/python3.10/dist-packages/matplotlib/pyplot.py
Display all open figures.

Parameters
---
block : bool, optional


400
600
800
1000
Y
0
250
500
750
1000
1250
1500
1750
X