** ENPM673 Perception Project 1**

Sets up the environment: The code mounts your Google Drive, navigates to a specific project folder, and creates an output folder if it doesn't already exist.

Prepares for image processing: This code establishes the groundwork for you to insert your image processing and analysis code within the indicated area.

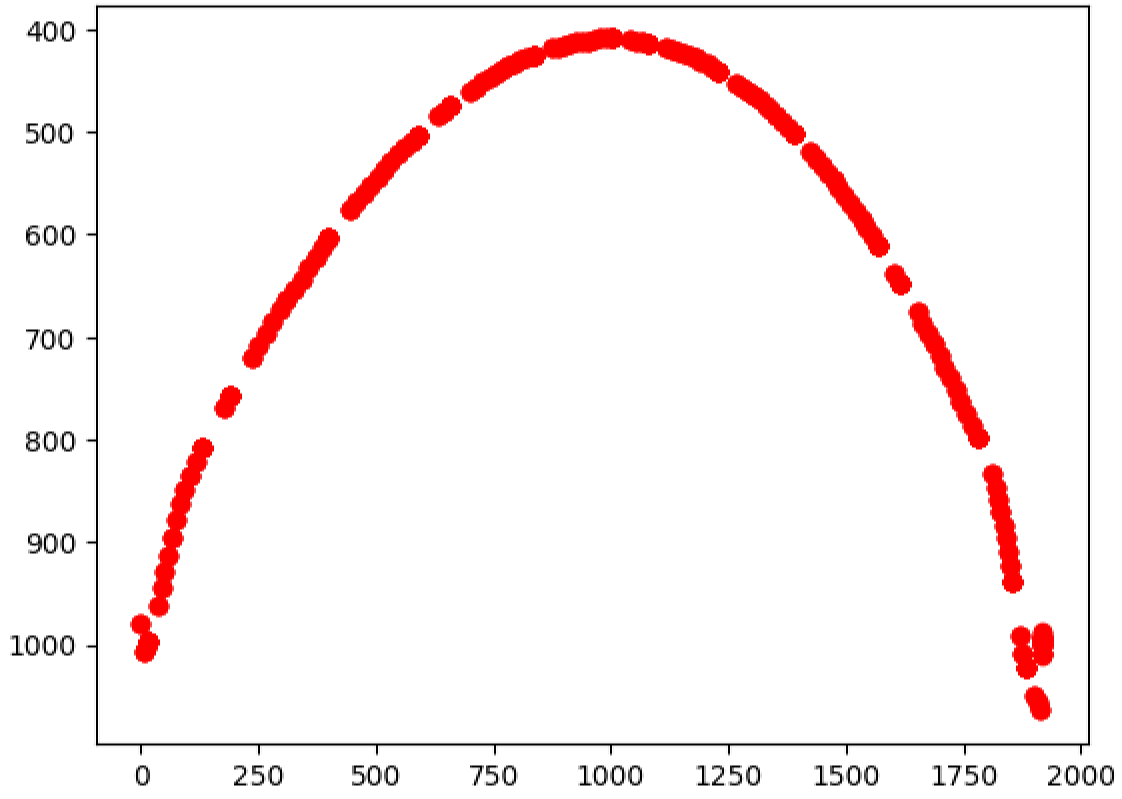
```
import os
from google.colab import drive
from google.colab.patches import cv2_imshow
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt

# Mount your Google Drive for file access
drive.mount('/content/drive/')

# Define the path to your project folder
path_to_folder = "ENPM673"
%cd /content/drive/My\ Drive/{path_to_folder}

# Specify the location for storing output frames
output_folder_path = os.path.join('/content/drive/My Drive/ENPM673/', 'frames_store')

# Create the output folder if it doesn't already exist
if not os.path.exists(output_folder_path):
    os.makedirs(output_folder_path)
```

```
Drive already mounted at /content/drive/; to attempt to forcibly remount, call drive.mou
/content/drive/My Drive/ENPM673
```

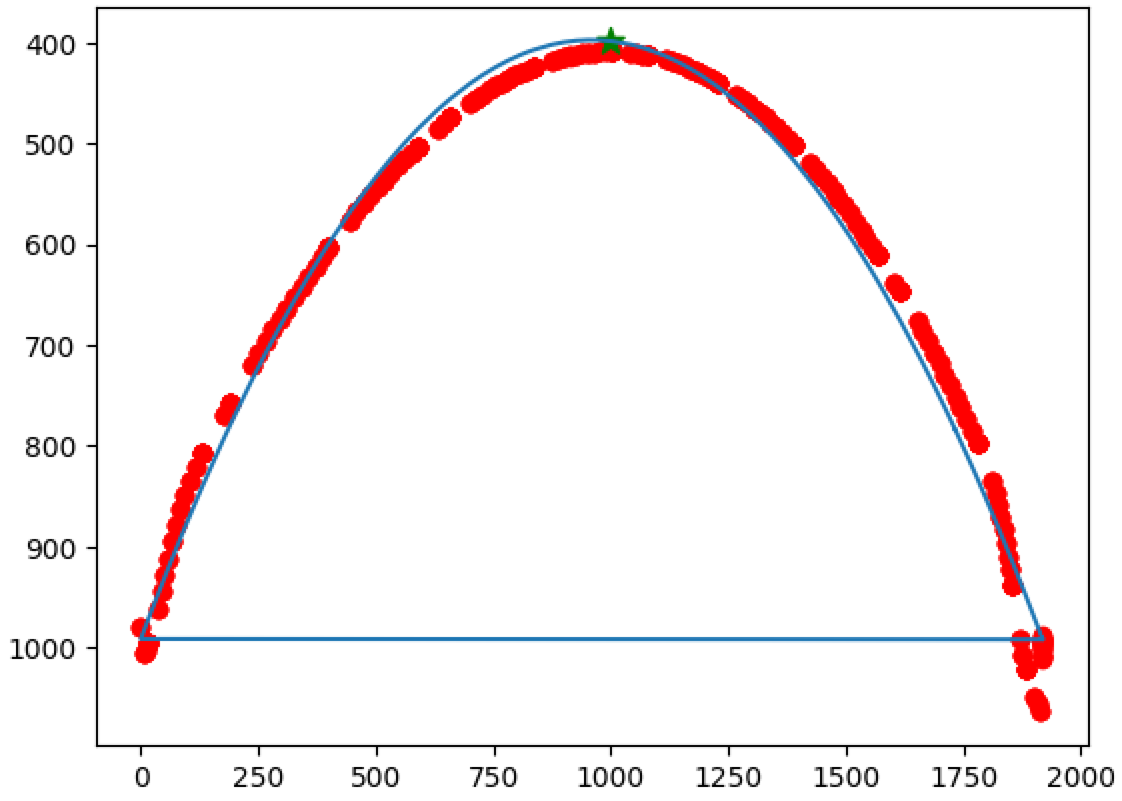
1. Object Detection and tracking: The code opens a video file named 'object_tracking.mp4' using OpenCV. In each frame, it finds pixels with low intensity (RGB values below 30), likely targeting a dark object against a brighter background. If a dark region is found, the code calculates its center (centroid). The centroid's x and y coordinates are stored in the lists x_centers and y_centers.
2. Trajectory Analysis and Curve Fitting The tracked centroids are plotted, visualizing the object's path over time. The code uses least squares regression to fit a parabolic curve (y = ax² + bx + c) to the trajectory data. Based on the fitted curve, it predicts a future position of the object by calculating y when x = 1000. The fitted curve and the extrapolated point are added to the plot.

```
# --- Object Tracking Setup ---

# Store coordinates of the tracked object's center
x_centers = []
y_centers = []

# Lists for potential further calculations (purpose currently unclear)
x_values = []
y_values = []

# Open the video file
cap = cv.VideoCapture('object_tracking.mp4')

# --- Main Processing Loop ---

while True:
    # Read a frame from the video
    ret, frame = cap.read()
    if not ret:  # Check if a frame was successfully read
        break

    # Isolate dark pixels (assuming we're tracking a dark object)
    low_intensity_condition = np.all(frame < 30, axis=2)
    low_intensity_indices = np.where(low_intensity_condition)

    # Separate the x and y coordinates of the dark pixels
    y_coords, x_coords = low_intensity_indices[0], low_intensity_indices[1]

    # Approximate area of the dark region (optional)
    area = len(x_coords)

    # Calculate and store the object's centroid
    if area > 0:  # Proceed only if a dark region is found
        center_x = int(np.mean(x_coords))
        center_y = int(np.mean(y_coords))
        x_centers.append(center_x)
        y_centers.append(center_y)

        # (Optional) Mark the centroid on the frame for visualization
        # cv.circle(frame, (center_x, center_y), radius=3, color=(0, 255, 0), thickness=-1)

    # ... (Add code to save frames or display the frame if needed)

    # Exit the loop if 'q' is pressed
    if cv.waitKey(10) & 0xFF == ord('q'):
        break

# Release resources when finished with the video
cap.release()
cv.destroyAllWindows()
```

```
# --- Trajectory Analysis ---

# Plot the object's trajectory
plt.scatter(x_centers, y_centers, c='r')
plt.gca().invert_yaxis()  # Adjust axes if needed
plt.show()

# --- Curve Fitting ---

# Prepare data for parabolic curve fitting
x_squares = np.array([x**2 for x in x_centers])
x_centers = np.array(x_centers)
y_centers = np.array(y_centers)
length = len(x_centers)

# Fit a parabola using least squares regression
X = np.column_stack((x_squares, x_centers, np.ones(length)))
inverse = np.linalg.inv(X.T @ X)
B = inverse @ X.T @ y_centers
a, b, c = B

# Extrapolate a point on the curve
x = 1000
y = a * x**2 + b * x + c
print("The y coordinate when x is 1000 is:", y)

# Visualize the fitted curve and extrapolated point
plt.scatter(x_centers, y_centers, c='r')
plt.plot(x_centers, a*x_squares + b*x_centers + c)
plt.scatter(x, y, color='green', marker='*', s=100)
plt.gca().invert_yaxis()
plt.show()

# Print the curve equation
equation = f'y = {a}*x**2 + {b}*x + {c}'
print("Equation of the curve is: ")
print(equation)

# A loop to store the x and y values to fit a parabola on a frame
for x_val in range(1900):
    x_values.append(x_val)
    y = a*x_val**2+b*x_val+c
    y_values.append(y)
```



```
The y coordinate when x is 1000 is: 398.42526288905026
```

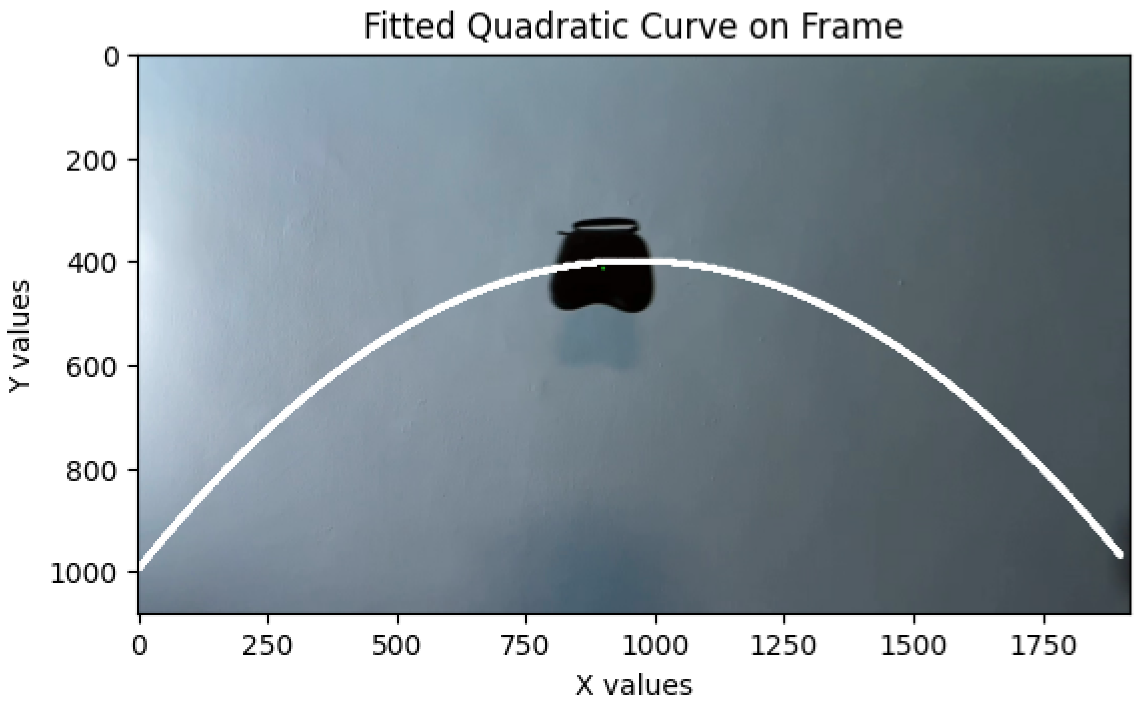


```
Equation of the curve is:
y = 0.0006456610616844081*x**2 + -1.238779479210716*x + 991.5436804153583
```

Plotting a parabola Based on a randomly selected frame: The code loads a previously saved image frame (frame_450.jpg) and displays it.

It overlays a scatter plot of points (stored in x_values and y_values),representing a calculated trajectory or analysis results, on top of the image.

```
# Load a saved image frame
frame = cv.imread('/content/drive/MyDrive/ENPM673/frames_store/frame_450.jpg')
# Display the image frame
plt.imshow(frame)
# Plot calculated points (likely from previous analysis) as white stars
plt.scatter(x_values, y_values, color='white', marker='*', s = 2)  # Plot the average positio
plt.xlabel('X values')
plt.ylabel('Y values')
plt.title('Fitted Quadratic Curve on Frame')
plt.show()
```


Fitted Quadratic Curve on Frame
Y values
X values