CP 2

Anastasia Sharangia

Problem:

Problem 2.1

- Determine the velocity, mass, and drag coefficient of a ball by analyzing its motion in a video from point A to point B.
- Develop suitable tests.
- Use ball motion ODEs and numerical methods.

ODE used:

Ball Motion

$$\frac{dx}{dt} = v_x$$
$$\frac{dy}{dt} = v_y$$
$$\frac{dv_x}{dt} = -\frac{k}{m} v_x \sqrt{v_x^2 + v_y^2}$$
$$\frac{dv_y}{dt} = -g - \frac{k}{m} v_y \sqrt{v_x^2 + v_y^2}$$

$$x(0) = x_0, \quad y(0) = y_0, \quad v_x(0) = v_{x0}, \quad v_y(0) = v_{y0}$$

- v_x: The horizontal component of the ball's velocity.
- v_y: The vertical component of the ball's velocity.
- g: The acceleration due to gravity $\approx 9.81\ m/s^2$ on Earth's surface.
- k: The drag coefficient, incorporates air resistance in the model.
- m: The mass of the ball.

Solution:

Transform_to_px:

Because our density and gravity are in meters and our other parameters from the video are in pixels i created a method that takes them as parameters, scale and frame_rate and returns the scaled version to pixels.

Calculate_mass:

This function returns the mass of an object which is a sphere, it takes radius and density as parameters, calculates volume and then multiplies it on density and returns the mass.

Then we go into the functuon that opens the video, do background subtraction using cv2's built in function, find out our g and density in pixels by passing g=9.81 and density = 0.92 (natural rubber, since it is a ball i assumed it is a natural rubber), then scale it to 100px = 1m, you can change that parameter if you want 😉, and fps of the video(i find out it using the built in cv2 function), Then i initialize some arrays to have the values after the video closes, and now we open the video, and apply mask, threshold and other stuff, to do edge detection and find clusters, after we find the clusters centroid we find out the ffarthest point from the clusters centroid and that represents our radius for this ball, which then gives us the ability to calculate the mass, since we have the radius and density.

Now we get to calculating the velocity, we start with initializing vx and vy with zeroes, and then calculate it using this logic:

We calculate the distance between the two frames centroids, which is the displacement, and we find the time in which this distance was covered using 1/fps, so velocity = displacement/time, and since we have both parameters we have found velocity for x and y and to find the final velocity we sum up the sqeuares of vx and vy and then take out the square root from it.

Now since we have velocity and mass, we use the ball moving ODE to find drag coefficient, we find kx and ky and then calculate the average of this two, if the drag coefficient is negative it means that the mistake has been made because the code might read it wrongly because of 1 pixel difference between centroids of two frames, so if the k is positive only then do we add it to the array.

Video Closes:

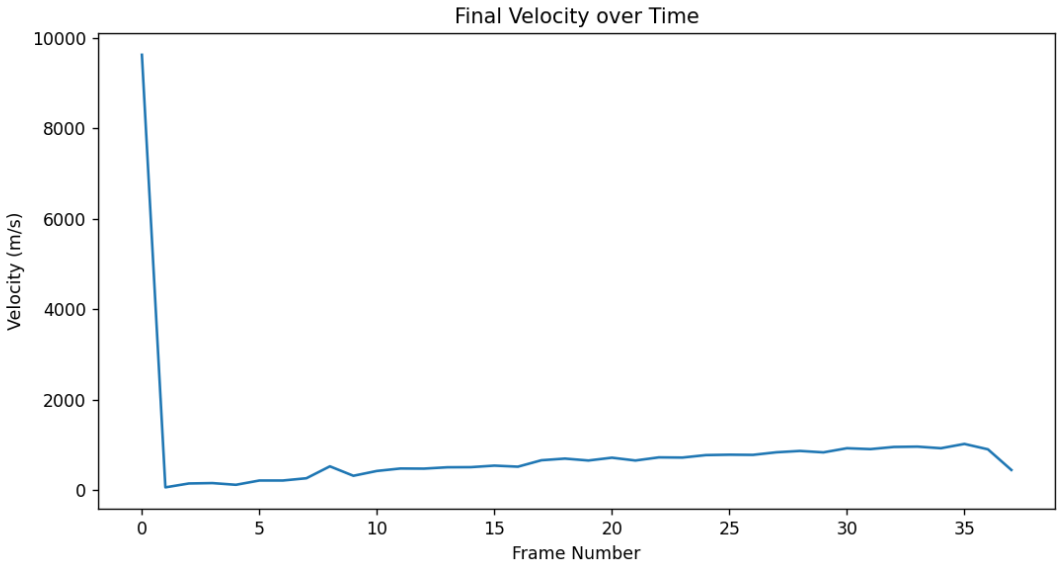
We calculate the average of radiuses that have been added to the array when the video was open (since radius is found using clustering, it is a bit inaccurate so we find the average of it for it to be precise and a constant) and we calculate the Final mass using this radius, and we find the average of drag coefficient since it is also a constant, lastly we plot the velocities using matplotlib to see how it changes over time.

THE END (magari var ra, dzaan kaia)

Input video:


XMDEMO

Output:


Final Velocity over Time
Velocity (m/s)
Frame Number
10000
8000
6000
4000
2000
0
0
5
10
15
20
25
30
35

```
vx: -3090.0 px/s, vy: -9120.0 px/s, v_final: 9629.25230742242 px/s
vx: -60.0 px/s, vy: 30.0 px/s, v_final: 67.08203932499369 px/s
vx: -30.0 px/s, vy: 150.0 px/s, v_final: 152.97058540778355 px/s
vx: -60.0 px/s, vy: 150.0 px/s, v_final: 161.55494421403512 px/s
vx: 30.0 px/s, vy: 120.0 px/s, v_final: 123.69316876852982 px/s
vx: 60.0 px/s, vy: 210.0 px/s, v_final: 218.40329667841556 px/s
vx: 60.0 px/s, vy: 210.0 px/s, v_final: 218.40329667841556 px/s
vx: 240.0 px/s, vy: 120.0 px/s, v_final: 268.32815729997475 px/s
vx: -150.0 px/s, vy: 510.0 px/s, v_final: 531.6013544000805 px/s
vx: 120.0 px/s, vy: 300.0 px/s, v_final: 323.10988842807024 px/s
vx: 90.0 px/s, vy: 420.0 px/s, v_final: 429.53463189829057 px/s
vx: 60.0 px/s, vy: 480.0 px/s, v_final: 483.735464897913 px/s
vx: -30.0 px/s, vy: 480.0 px/s, v_final: 480.9365862564419 px/s
vx: 30.0 px/s, vy: 510.0 px/s, v_final: 510.88159097779203 px/s
vx: 60.0 px/s, vy: 510.0 px/s, v_final: 513.5172830587107 px/s
vx: 90.0 px/s, vy: 540.0 px/s, v_final: 547.4486277268397 px/s
vx: 120.0 px/s, vy: 510.0 px/s, v_final: 523.9274758971894 px/s
vx: -90.0 px/s, vy: 660.0 px/s, v_final: 666.1080993352356 px/s
vx: 450.0 px/s, vy: 540.0 px/s, v_final: 702.9224708315988 px/s
vx: -30.0 px/s, vy: 660.0 px/s, v_final: 660.6814663663572 px/s
vx: 60.0 px/s, vy: 720.0 px/s, v_final: 722.4956747275377 px/s
vx: 30.0 px/s, vy: 660.0 px/s, v_final: 660.6814663663572 px/s
vx: 120.0 px/s, vy: 720.0 px/s, v_final: 729.9315036357864 px/s
vx: 90.0 px/s, vy: 720.0 px/s, v_final: 725.6031973468695 px/s
vx: 0.0 px/s, vy: 780.0 px/s, v_final: 780.0 px/s
vx: 120.0 px/s, vy: 780.0 px/s, v_final: 789.1767862779543 px/s
vx: 90.0 px/s, vy: 780.0 px/s, v_final: 785.175139698144 px/s
vx: 60.0 px/s, vy: 840.0 px/s, v_final: 842.140130857092 px/s
vx: 60.0 px/s, vy: 870.0 px/s, v_final: 872.066511224918 px/s
vx: 30.0 px/s, vy: 840.0 px/s, v_final: 840.535543567314 px/s
vx: 60.0 px/s, vy: 930.0 px/s, v_final: 931.933474020544 px/s
vx: 150.0 px/s, vy: 900.0 px/s, v_final: 912.4143795447329 px/s
vx: 30.0 px/s, vy: 960.0 px/s, v_final: 960.4686356149273 px/s
vx: 120.0 px/s, vy: 960.0 px/s, v_final: 967.470929795826 px/s
vx: 60.0 px/s, vy: 930.0 px/s, v_final: 931.933474020544 px/s
vx: 120.0 px/s, vy: 1020.0 px/s, v_final: 1027.0345661174215 px/s
vx: 120.0 px/s, vy: 900.0 px/s, v_final: 907.9647570252934 px/s
vx: 30.0 px/s, vy: 450.0 px/s, v_final: 450.99889135118724 px/s


 Mass: 0.29412346610559675kg, Drag coefficient: 0.003388070299500138
```

Code:

```python
import numpy as np
import cv2
import matplotlib.pyplot as plt


def transform_to_px(g, density, scale, frame_rate):
    scaled_g = g * scale / frame_rate ** 2
    scaled_density = density / scale ** 3
    return scaled_g, scaled_density
```

```python
def calculate_mass(radius, density):
    volume = (4 / 3) * np.pi * (radius ** 3)
    mass = density * volume
    return mass

def detect_single_moving_object(video_path):
    cap = cv2.VideoCapture(video_path)
    fps = cap.get(cv2.CAP_PROP_FPS)

    # I will say that hypothetically 100px is 1m
    # g = 9.81 Gravitational acceleration (m/s^2)
    # density = 0.92 Natural rubber density (kg/m^3)
    g, density = transform_to_px(9.81, 0.92, 100, fps)

    back_sub = cv2.createBackgroundSubtractorMOG2(history=500, varThreshold=50, detectShadows=False)

    prev_centroid = None
    prev_vx = None
    prev_vy = None
    velocities = []
    drag_coefficients = []
    radiuses = []

    while cap.isOpened():
        ret, frame = cap.read()
        if not ret or frame is None:
            break

        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        fg_mask = back_sub.apply(gray)
        _, thresh = cv2.threshold(fg_mask, 50, 255, cv2.THRESH_BINARY)
        kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5, 5))
        cleaned = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, kernel)
        cleaned = cv2.morphologyEx(cleaned, cv2.MORPH_CLOSE, kernel)
        contours, _ = cv2.findContours(cleaned, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

        if contours:
            largest_contour = max(contours, key=cv2.contourArea)
            (x, y), radius = cv2.minEnclosingCircle(largest_contour)
            radiuses.append(radius)

            if radius > 5:
                center = (int(x), int(y))
                cv2.circle(frame, center, int(radius), (0, 0, 255), 2)

                mass = calculate_mass(radius, density)

                vx, vy = 0, 0

                if prev_centroid is not None:
                    dx = center[1] - prev_centroid[1]
                    dy = center[0] - prev_centroid[0]

                    dt = 1/fps

                    vx = dx / dt
                    vy = dy / dt
```

```
                    v_final = np.sqrt(vx**2 + vy**2)
                    velocities.append(v_final)

                    if vx != 0 or vy != 0:
                        if prev_vx is not None and prev_vy is not None:
                            dvx_dt = (vx - prev_vx) / dt
                            dvy_dt = (vy - prev_vy) / dt

                            if vx != 0 and vy != 0:
                                k_vx = (-mass * dvx_dt) / (vx * np.sqrt(vx**2 + vy**2))
                                k_vy = -mass * (dvy_dt + g) / (vy * np.sqrt(vx**2 + vy**2))
                            else:
                                k_vx = 0
                                k_vy = 0

                            k = (k_vx + k_vy) / 2 if (vx != 0 and vy != 0) else 0

                            if k > 0: drag_coefficients.append(k)

                            print(f"vx: {vx} px/s, vy: {vy} px/s, v_final: {v_final} px/s")

                prev_centroid = center
                prev_vx = vx
                prev_vy = vy

        cv2.imshow('Single Moving Object Detection', frame)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    cap.release()
    cv2.destroyAllWindows()

    avg_radius = np.mean(radiuses)
    mass = calculate_mass(avg_radius, density)

    avg_drag = np.mean(drag_coefficients)

    print(f"\n \n Mass: {mass}kg, Drag coefficient: {avg_drag} \n \n")

    plt.figure(figsize=(10, 5))
    plt.plot(velocities)
    plt.title("Final Velocity over Time")
    plt.xlabel("Frame Number")
    plt.ylabel("Velocity (m/s)")
    plt.show()

video_path = './content/cp2_anastasia_sharangia.mp4'
detect_single_moving_object(video_path)
```

Limitations:

When the method works:

When the inputs are correct, close to the truth, the answers will be close to the truth as well, the video is not overly complicated meaning that the background is not too complicated, so that it clusters correctly, also there should be only one moving object, camera must be still. Video should be 2D.

When the method fails:

It is dependent on the inputs of gravity, density and scale, since giving different inputs from this gives us a completely different answer, overly complicated background gives us incorrect clusters and multiple moving objects also does not help us, also when the camera follows the object and is not still it also fails. Video that is not 2D.

Fail video:

Conclusion:

This project combines video analysis and physics to determine velocity, mass, and drag coefficient of a sphere. By scaling physical parameters to pixel units, estimating mass via clustering, and calculating velocity from centroid displacement, it achieves reliable results. While minor errors from clustering and pixel precision exist, this code finds mass, velocity and drag coefficient.