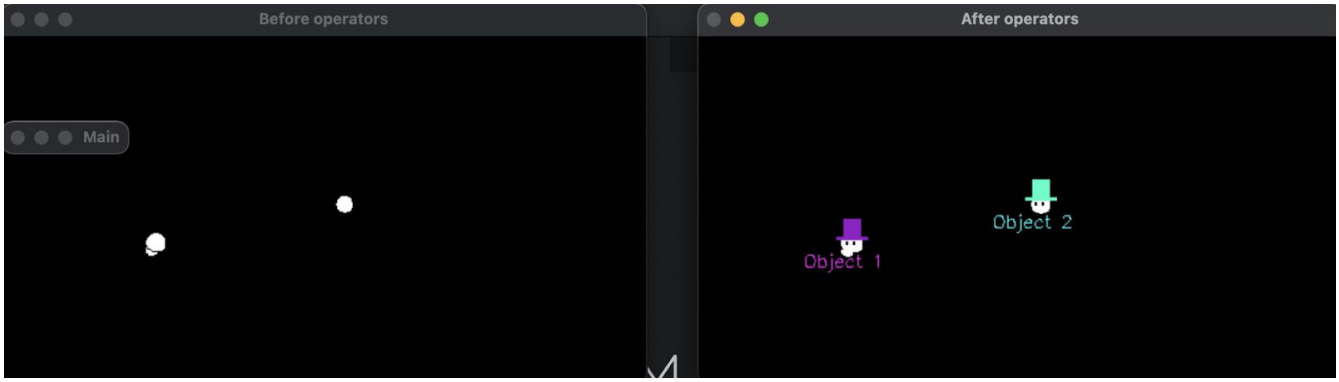
APPROACH

I used some aspects of the provided sample code regions.py and capture.py to help locate the centroid and read and output the video. The program reads in a file and takes the average of all the pixel values in all the frames. It then uses this average as a reference to carry out background subtraction for each frame in the video. At first I used a list to store all the video frames and then loop over the list from which I can subtract the reference image. However, this method would eat up a lot of memory space really quickly, so another implementation was put in place where I would just loop over the video twice. I wanted to show the effect of using the morphological operators simultaneously, but when I used cv2.imshow() twice, it just alternated between the two really quickly. Hence I had to create two windows to show the before and after applying morphological operators to the frames.

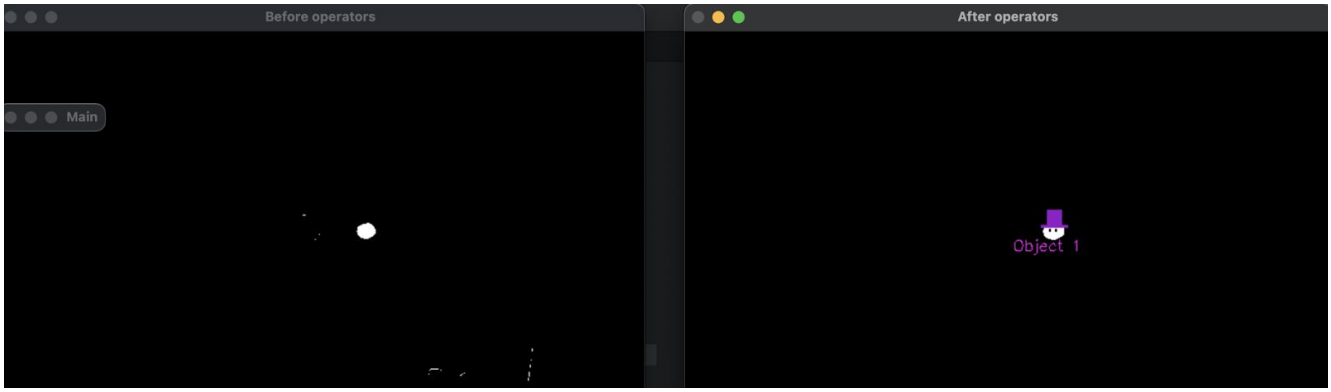
To track each object, I found the location of each centroid and compared it to the centroids of the previous frame. For example, if a centroid in the current frame is closer to object 1 than to object 2 in the previous frame, then that centroid is labelled as object 1. I then created a list of all the x and y coordinates of both objects and at the end, plotted it on the same graph for comparison.

As a fun experiment, I decided to put tiny top hats on each of the objects with eyes as well. In order to do this, I had to convert the thresholded image back to an RGB image so the colors would show. Then I had to take each centroid and set some set of pixels some distance away from the centroid to add hats and eyes. I then decided to label these objects as they were moving, so I added some text labels that would follow these objects. This was implemented through adding the text at some point below the centroid so it would not obstruct the object itself.

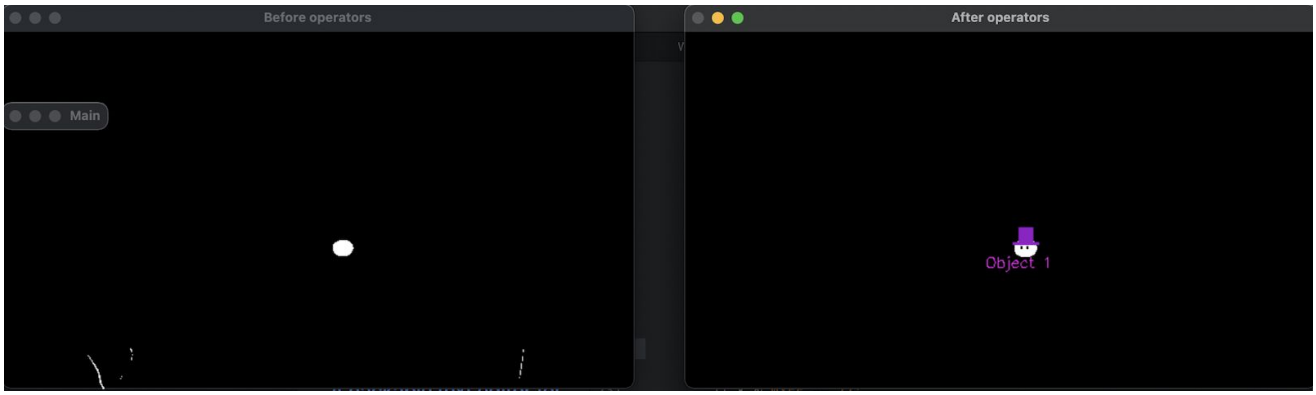
BEFORE AND AFTER MORPHOLOGICAL OPERATORS



This is a screenshot of before and after applying morphological operators for the video double.mp4. There wasn't much noise in this video, but the issue here was that the balls kept being broken into more than 1 piece, which is a problem in tracking these objects, because the code would see the image on the left as 3 different objects instead of 2.



This is a screenshot of before and after applying morphological operators for the video sphere.mp4. This video had some more noise compared to double.mp4 as evident in the left image where there are some white speckles. This was completely removed after applying erosion.

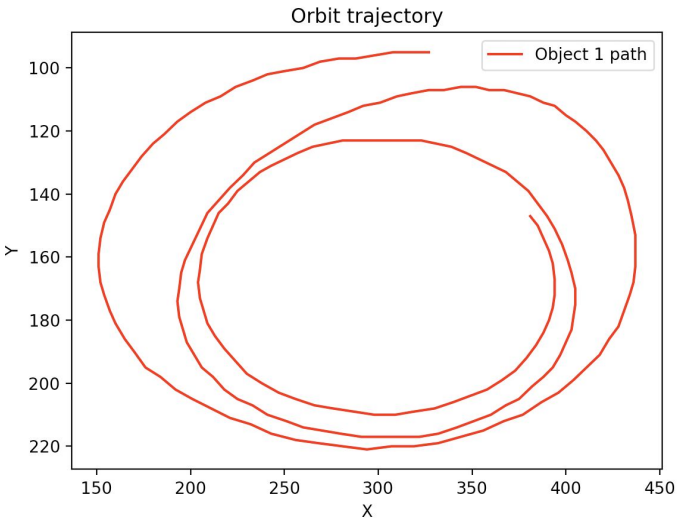


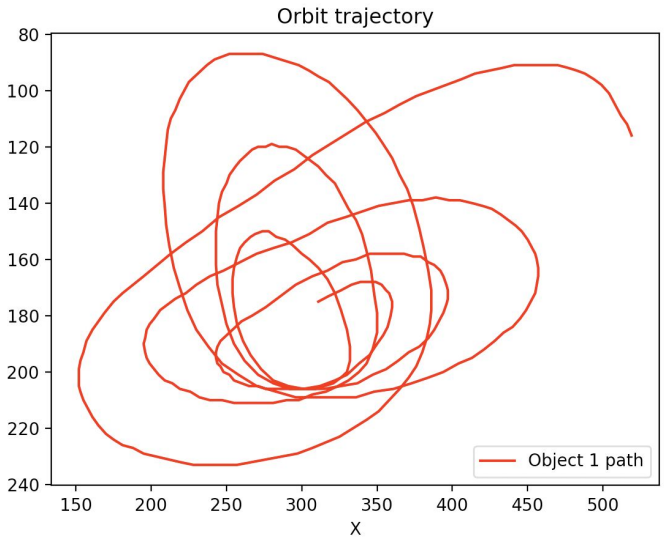
This screenshot is of the ellipse.mp4 video. Because the ball was moving a lot more in this video, the fabric was also bending a lot more. This video had more noise than the previous two.

PLOTS

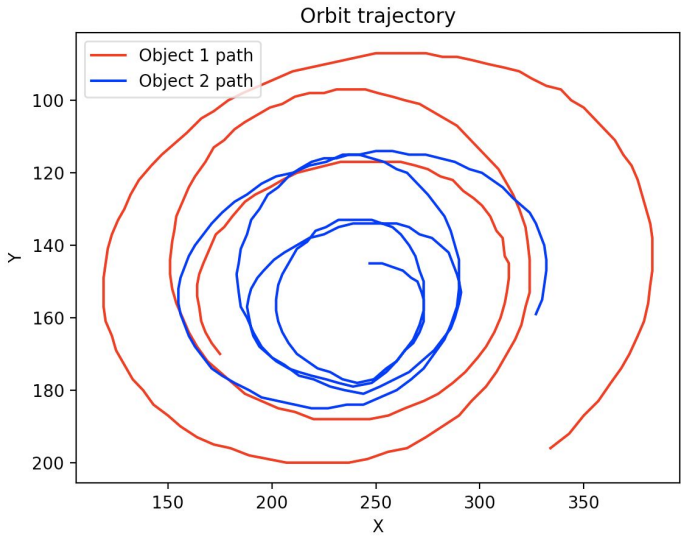
As we would expect, since these videos are simulating gravity, throwing it in around the edge would cause a circular orbit as shown on the left.

If we were to then throw it roughly closer to the center of the system, then it would mimic a more elliptical orbit as shown on the right.





If we take two objects in circular orbit, then it could run a collision course as shown in the plot below. If the timing were changed, the intersections of the blue and red lines would be where the two objects would collide.



ASSUMPTIONS AND CONSIDERATIONS

The code assumes that there are either one or two objects of interest in the data. If there were three objects of interest in the video, then it would just track and label two of the objects while completely ignoring the third object. The last object would still be thresholded from the background given that it does not stay in one place for a long time, but it would not be labelled and tracked. This would work on objects of different colors since it's implemented as temporal averaging instead of thresholding on a specific color. In very bright conditions however, this would not work as reflection from other objects could interfere with the tracking and distinguishing objects of interest from the background. For example, I tried to input a video of an outdoors swimming competition, but the water was glistening on and off, so the pixel values there changed even more than the swimmers themselves.

One interesting challenge that came up while doing this was the idea of object permanence. I could not use videos that were from a flatter angle of view where the ball would disappear completely because it orbited behind the central ball. In this case, the centroid tracking function would return an error since there are no objects in the video.

VIDEO CREDITS

Double.mp4 was taken from https://www.instructables.com/Spacetime-Table/. This website was an instructional website on how to create your own gravity well demo. The other two were taken from https://www.youtube.com/watch?v=cHySqQtb-rk&t=137s, which were just demonstrations of different types of gravitational interactions in space.