Project Report

Rune Sleeuwaert, Mateo Van Geyt, Robbe De Helt & Yorben Joosen

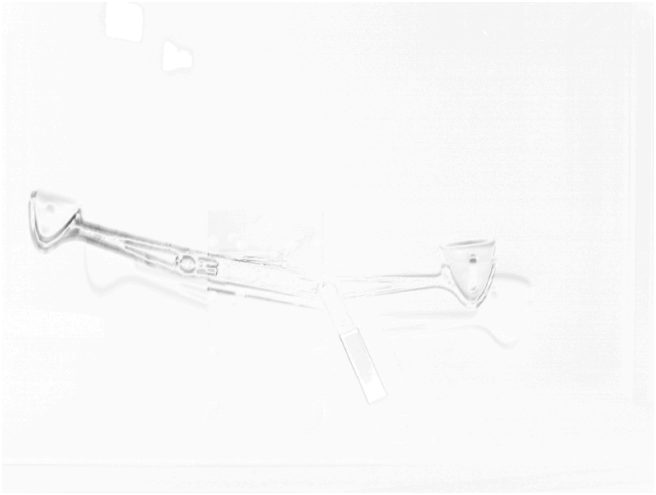
For this project we had three main objectives:

- Measuring the deformations of each mechanical part, including joints (bearings).
- Detecting vibration frequencies in the system.
- Validating the virtual controller trajectory using camera measured trajectories.

We were immediately sceptical about the applicability of image recognition techniques for measuring differences and errors this small, because the quality of the videos aren't very good. We did however try a few techniques.

Frame difference:

Initially we tried simplifying the video by creating a kind of movement mask. This is done by subtracting each frame from the next, in areas where there was no movement the result will be 0. In other regions the subtraction will result in a non 0 outcome and the movement will be visible.



In the end this didn't really simplify the problem and we didn't really use it.

Position & angle

One of the objectives was to validate the virtual controller. To be able to do this we first needed a reliable way to track a certain part of the bar. We made the code to track the object in the "object_tracking_basic.py" python file. We have done quit a lot of research to decide wich motion tracking technique we wanted to use.

1) Boost Tracker
    - Verouderd en traag
    - Bestaat nog voor legacy redenen

2) MIL Tracker
    - Beter dan boost
    - Does a bad job of reporting failure
3) KCF Tracker
    - Kernelized Correlation Filters
    - Véél beter dan de vorige twee
    - Does not handle occlusion well <- Wat betekend dit?
    -> occlusion = Occlusion means that there is something you want to see, but can't due to some property of your sensor setup => If you are developing a system which tracks objects (people, cars, ...) then occlusion occurs if an object you are tracking is hidden (occluded) by another object.
4) CSRT Tracker
    - Discriminative Correlation Filter
    - Meestal een beetje meer accuraat dan KCF maar trager
5) MedianFlow Tracker
    - Goed met reporting failures
    - Faalt bij too large of a gap in motion zoals te snel of veranderen van uitzicht
6) TLD
    - Niet goed?
7) MOSSE Tracker
    - Héél snel, maar niet zo accuraat als bv. CSRT of KCF
8) GOTURN Tracker
    - Het enigste [[Deep Learning]] [[Algorithm]] in [[OpenCV]]
    - Heeft model files nodig
    - Niet aan te raden?

In the end, we concluded that the KCF, motion tracking technique was the best suited for our problem. It has pretty good accuracy and isn't too computational expensive.

You select an area you want to track, and it returns a list for the values of its coordinates for every frame. Now that we have the data of the object, we can analyse this. We do this in the "trajectory_analysis.py" python file. This file has a displayPath method that will use the data to plot the path of the tracked object. Another analysis method is the display_angle, which shows the position (angle) of the object. This method makes use of the calculate_angle method, which uses pythagoras to calculate the angle of the object.

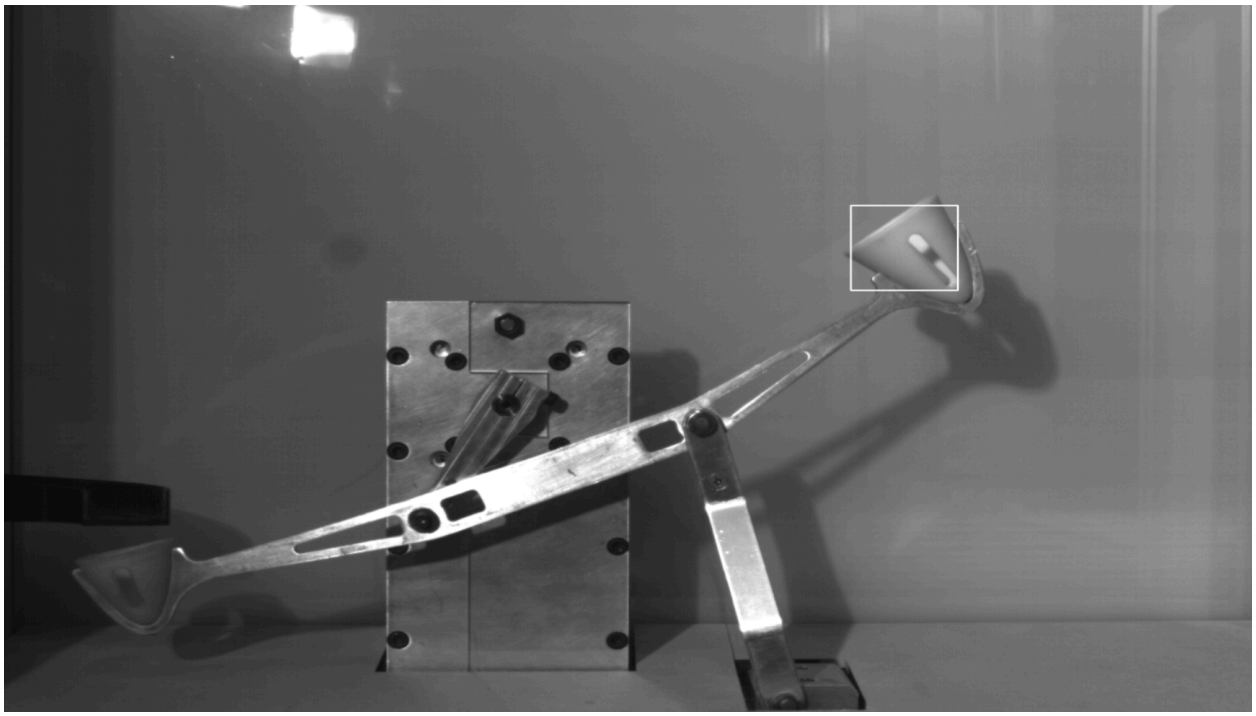
```
zero_aligned_list = cls.align_with_zero(input_array)
middle_x, middle_y = cls.calc_middle(zero_aligned_list)
return list(map(lambda rect: math.atan2(rect[0] - middle_x, rect[1] - middle_y), zero_aligned_list))
```

To do this, we first need to align every "box" (the coordinates (x,y), width and height of the object) to zero, so we can can calculate the coordinate of the middle point of the box. Now that we have this data, we can calculate the angle. But now we get the angle in radian so we need to convert it to degree, which is pretty simple. Then we can plot this data using the matplotlib library.
Now that we have the angle data, we can calculate the angle speed using the following formula.

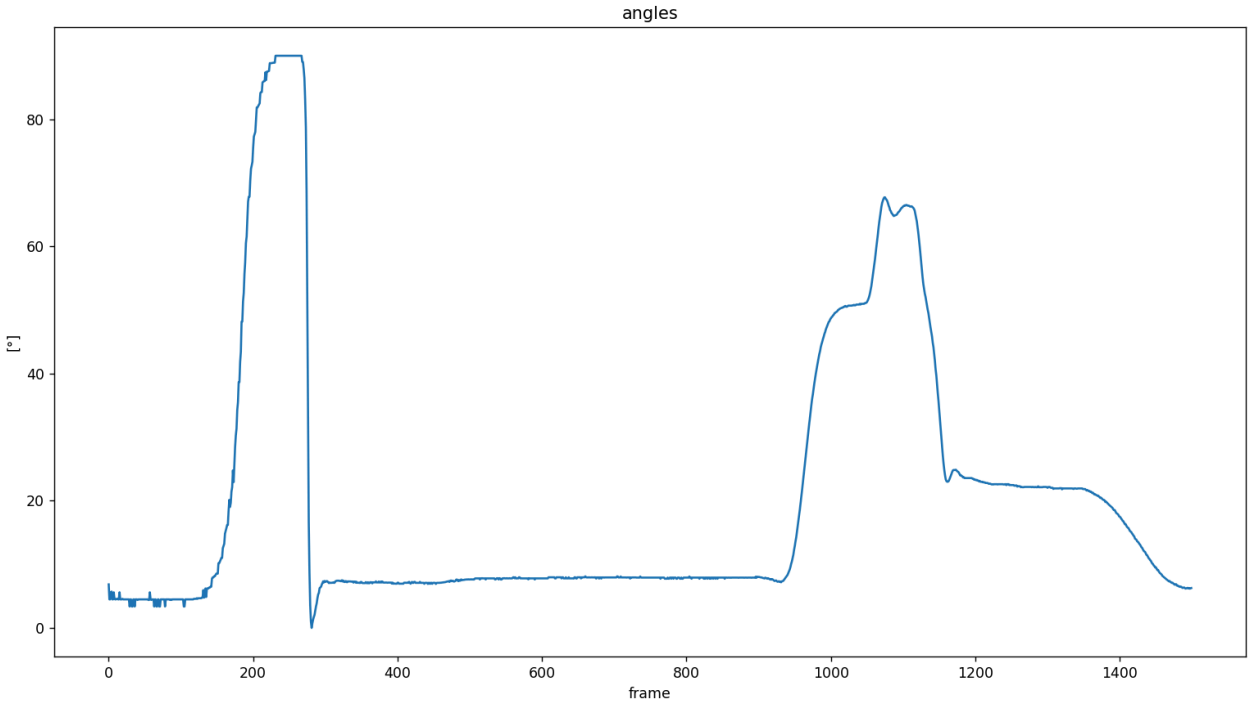
```
return list(map(lambda pair: abs(pair[1] - pair[0]), pairwise(input_angles)))
```

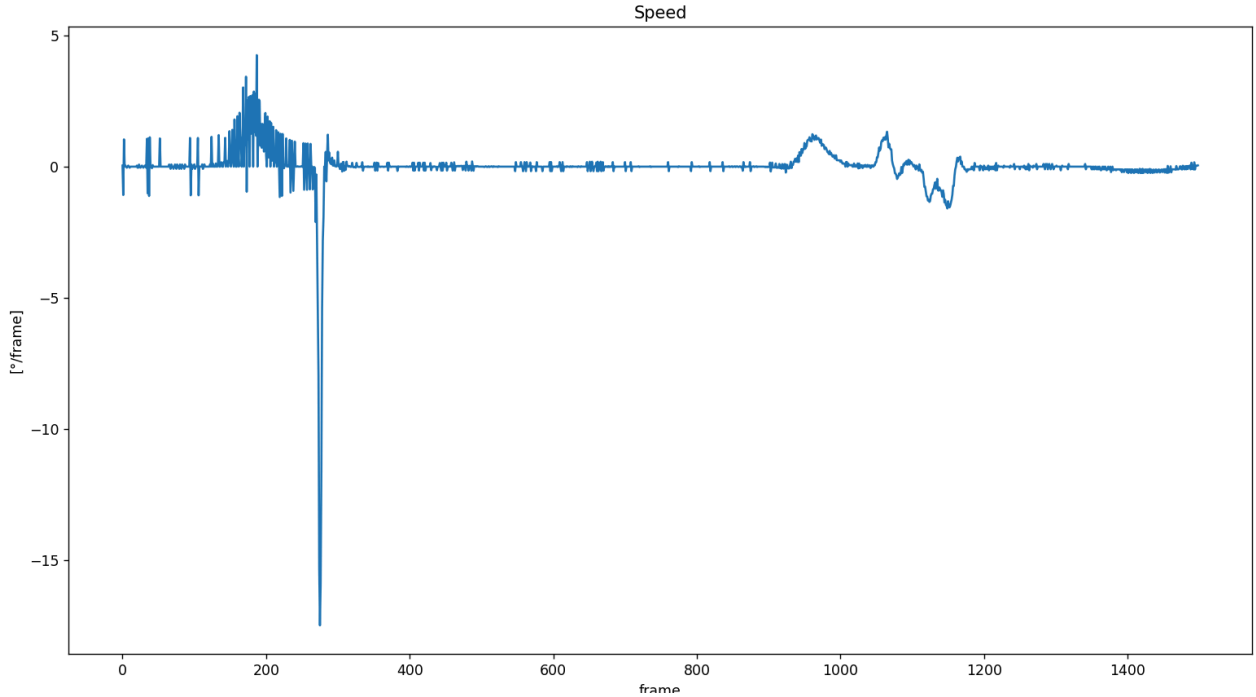
Which calculates the difference in angle using the current and next angle. We can also plot this. Lastly, we can also calculate the vibrations. If we have the angle, you can see if there are vibrations. Based on the information given, we concluded that the vibrations they wanted to see were those which occurred when the motor was almost not moving. So we can check everytime the motor speed is very low (approximately zero) and calculate the difference of the x- and y-pixels. This will give us the x- and y-vibrations.

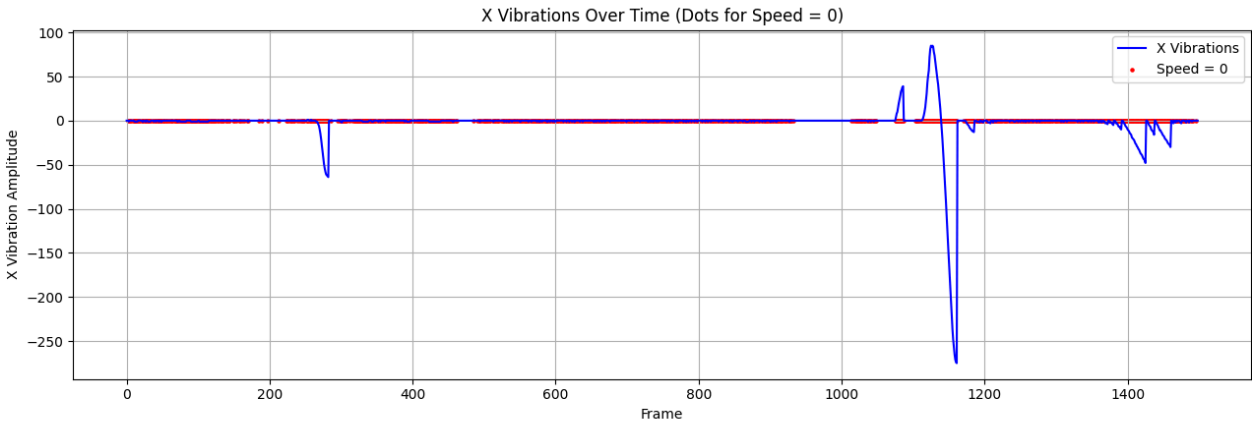
Sadly, it is very difficult to select something on the video because it can't be resized. We have a resize() function, but if we do this it breaks the tracking. So we can't use it, but this makes it so the video is very big. But once you select something and press enter, the video is resized to a normal size.

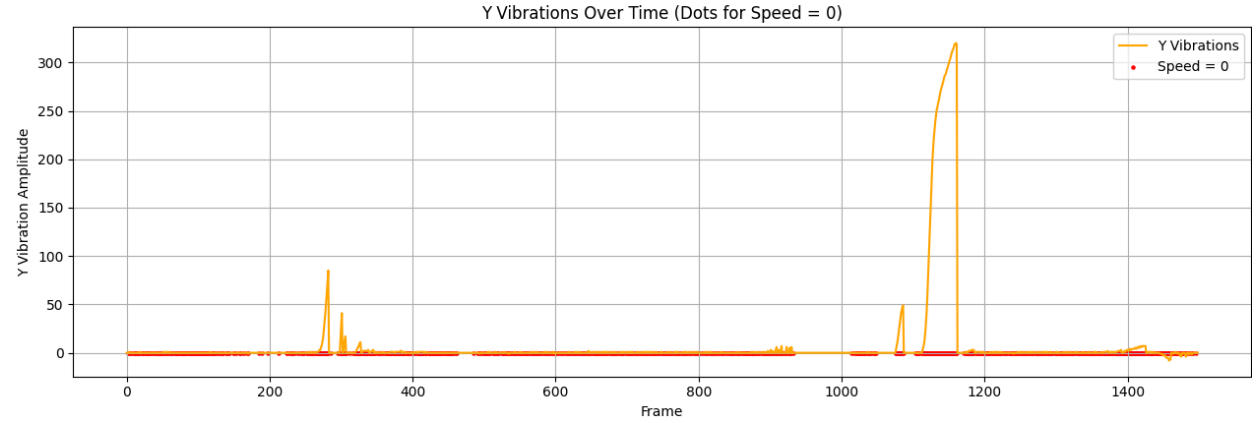


So select a part on the bar, and let the video play out. Once you do this you get this:


angles
[°]
0
20
40
60
80
0
200
400
600
800
1000
1200
1400
frame


Speed
[°/frame]
5
0
−5
−10
−15
0
200
400
600
800
1000
1200
1400
frame

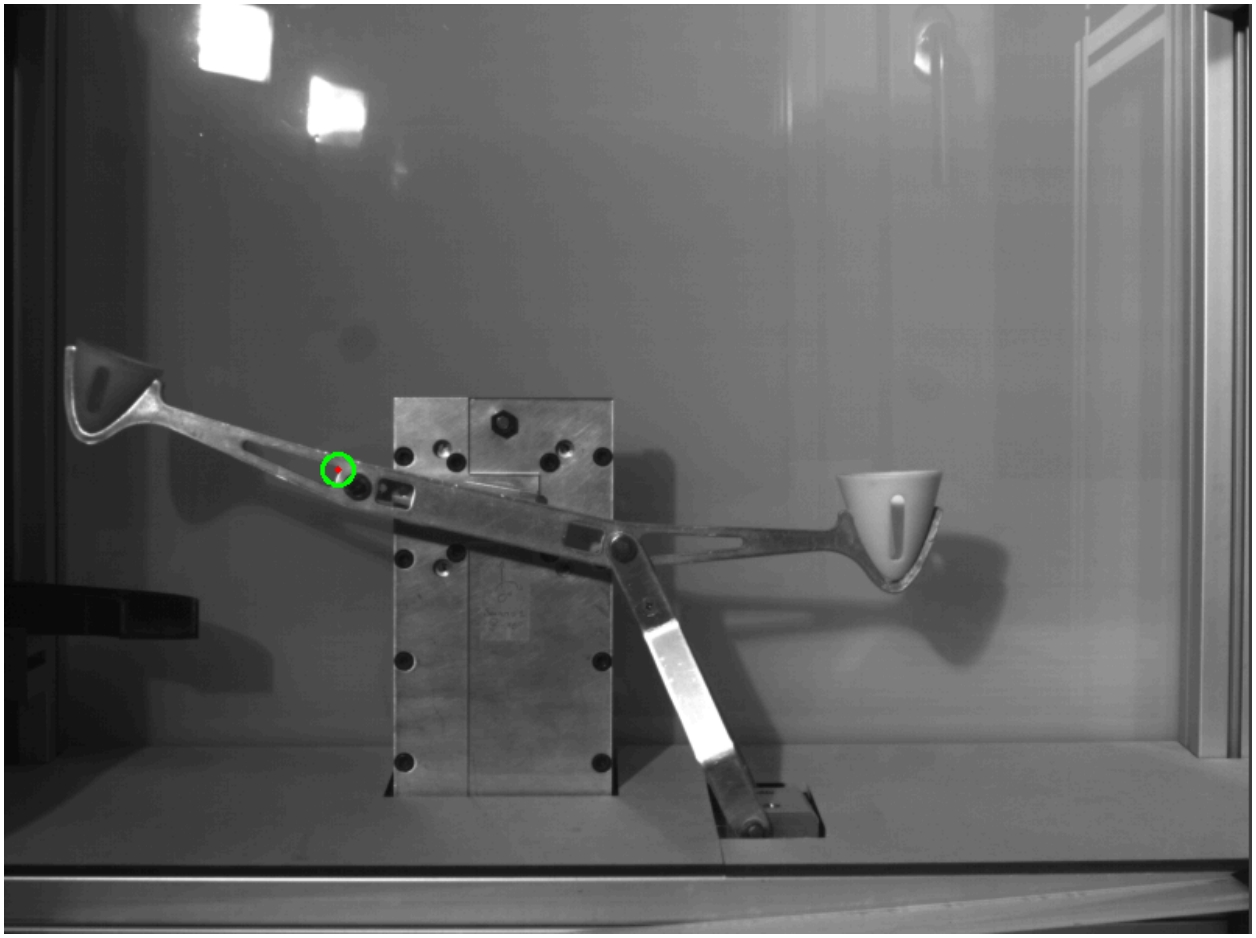




As you can see, it doesn't totally match the given graphs. But if you transpone the data or use different reference points (selected area to track and point to which it is calculated) I'm pretty sure the data will be useful. In the document given to us, it never mentioned what exactly was used for the selected area and what was used for the reference point. I have sent the EM student, a mail to ask about the used points. But the answer wasn't very clear, so we have done it like this. But the EM student knows what points he wants to use, so he can easily adjust the code a little bit especially because we are using python which is a very easy language.

Circle Tracking:

Another way we tried measuring the movement of the Juggler was by tracking the bearings, because of their simple shape (circle) we thought this would simplify the problem.
In reality, it turned out to make it more difficult. Our program kept identifying circles where there were none. Eventually we got it to look for circles of the wanted size. But still, during tracking the circle that was recognized tended to drift off, completely ruining the accuracy of our measurements. This led us to the decision not to pursue this route further



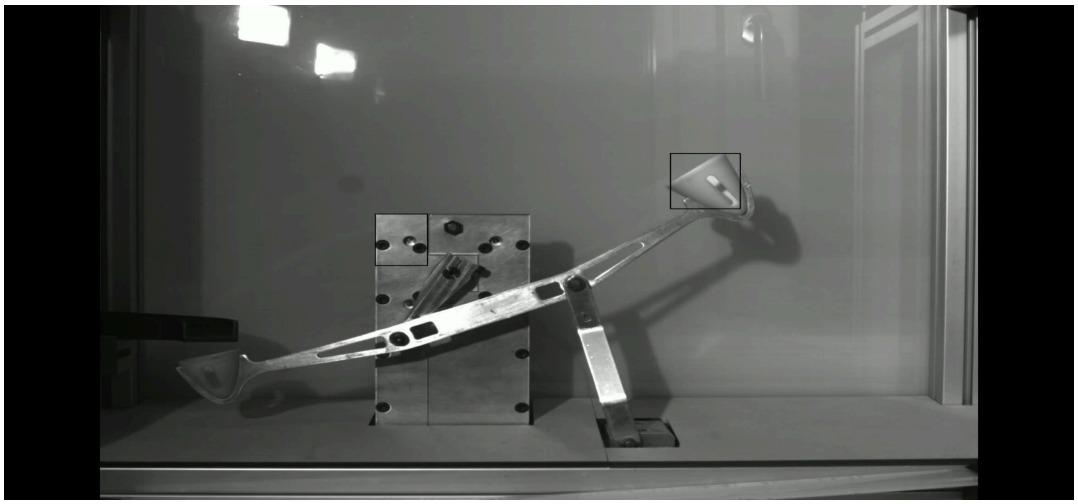
Code:

The CircleTracker class allows a user to click on a circle in a video to track its movement. It uses OpenCV's cv2.HoughCircles to detect circles near the click and selects the closest one. The selected circle's region is cropped as a template for tracking using cv2.matchTemplate. During video playback, the class updates the circle's position based on the template's match in each frame, displaying its location in real-time.

Hu moments

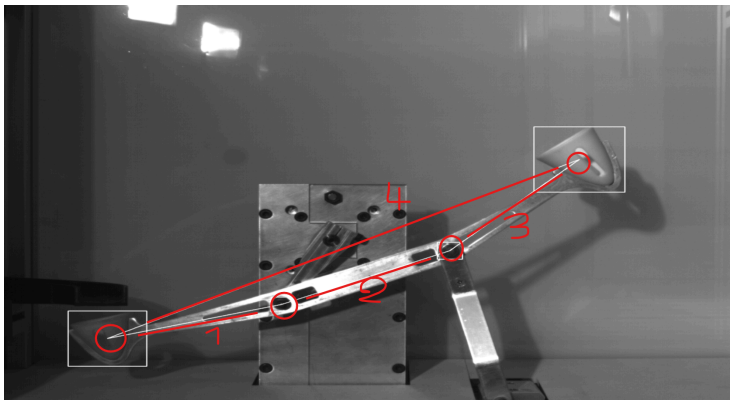
This is also a method we tried to track the movement of the juggler. The nice thing about it is that it's invariant to translation, scale, and rotation.
We tried tracking the left and right cup, but it only seemed to work for the right cup and even there it wasn’t very accurate. The reason that the left cup didn’t work may be due to it being in a dark place and thus there not being a lot of contrast in the reference image.

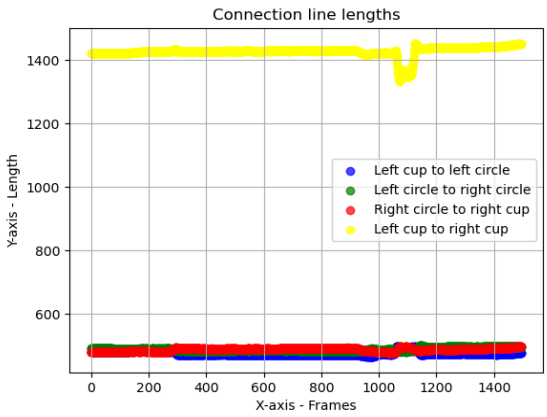
Some ways the accuracy could potentially be improved is by comparing the areas of the reference image and the contours of each frame but this kinda defeats the purpose of being invariant to scale. Another method may be to use the cv2.matchShapes function that uses hu moments instead of calculating these ourselves.



Rigid structure deformation

We track the movement of all the points notated on the image below. If, during the moment, any of the distances between the points change, the rigid structure undergoes deformation.





To better analyze any changes in distances, we graph the distance between all the points. As displayed, there might be a deformation at frame number 1100. More thorough analysis is required to validate.