Ping-Pong Project: Orientation Tracking of Racket

The Goal

In the previous document, I explained how I was able to track the ping-pong racket, and also concluded that I'm able to get an "angle", which was the angle between the major axis of the racket and the X axis.

In the past two weeks I have been working on a way to be able to get the Euler angles (which consist of roll, pitch, and yaw) for the ping-pong racket in an efficient manner.

The Result

I experimented with two approaches for obtaining the Euler angles of the racket. Both of these approaches consist of using three constant points on the racket, and only differ in how we find these points.

The first approach will be the inferior compared to the second approach, for reasons which I will explain.

The constant points which will be used for tracking the racket blob consist of one endpoint on the major axis, one endpoint on the minor axis, and finally, the center of the blob.

The reason we pick these three points is due to being able to properly capture all of the Euler angles for the racket.

Note that in each frame we have two stereo images each containing these three points. For each of the three corresponding points between the two images, we use the Triangulate function in order to find the coordinations of the three points in 3D space, we then find the Euler Angles by first finding the transformation matrix between these three points, and then converting that transformation matrix into the Euler angles.

Approach 1 – Finding constant points automatically:

After finding the racket blob by using a mask, we can also easily find the centroid in each of the stereo images (in turn finding the 3D coordinations of the center easily) However, in order to find an endpoint for each

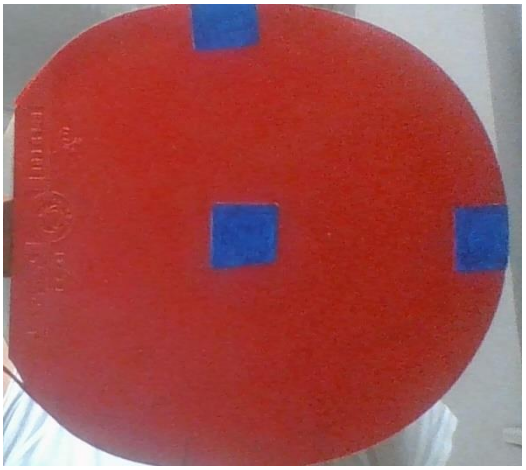
one of the axis', we must compute rather complex formulas:

```
point1_image1 = [ blobMeasurements.Centroid(1)+blobMeasurements.MajorAxisLength*cosd(blobMeasurements.Orientation)/2,
                  blobMeasurements.Centroid(2)+blobMeasurements.MajorAxisLength*sind(blobMeasurements.Orientation)/2]

point1_image2 = [ blobMeasurements.Centroid(1)+blobMeasurements.MajorAxisLength*cosd(blobMeasurements.Orientation)/2,
                  blobMeasurements.Centroid(2)+blobMeasurements.MajorAxisLength*sind(blobMeasurements.Orientation)/2]
----
point2_image1 = [ blobMeasurements.Centroid(1)-blobMeasurements.MinorAxisLength*sind(blobMeasurements.Orientation)/2,
                  blobMeasurements.Centroid(2)+blobMeasurements.MinorAxisLength*cosd(blobMeasurements.Orientation)/2]

point2_image2 = [ blobMeasurements.Centroid(1)-blobMeasurements.MinorAxisLength*sind(blobMeasurements.Orientation)/2,
                  blobMeasurements.Centroid(2)+blobMeasurements.MinorAxisLength*cosd(blobMeasurements.Orientation)/2]
```

Since we have to do these computations on each frame, the amount of processing time this needs is very large, and I noticed that my program ran very slower than before implementing this part of the code.
I had to change my approach in order to get better results…

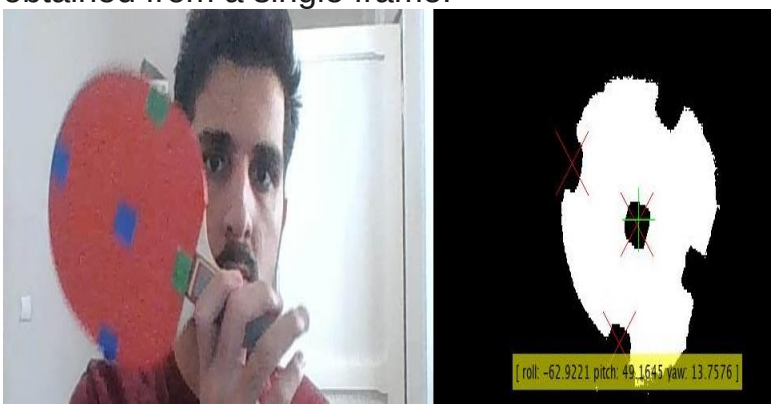
Approach 2 – Finding constant points by using markers:

By attaching colored stickers on to the three points we are looking to track, finding the coordinations of the points is much easier, since all we need to do is a simple color thresholding, and then track the three blobs.



As suspected, by tracking the Euler angles using these colored stickers, the computational time needed to find the 3D coordinations is drastically decreased, and is almost identical to before we implemented the function.

As an example, here is the result obtained from a single frame:



Here I'm only using the blue stickers to track the racket. I was trying out different colors and noticed I was getting good results from this color.

Conclusion

By using markers on the ping pong racket, we are able to find the coordinations of the constant points in 3D space. After having the three points, the computations needed to find the Euler angles of the racket plane is as follows:

```
v1=P2-P1;
v2=P3-P1;

X = normalize((P1+P2)/2 -P3);
Z = normalize(cross(v1,v2));
Y = cross(Z,X);

rotation_matrix = [X' Y' Z'];

orientation=rotm2eul( rotation_matrix );
```