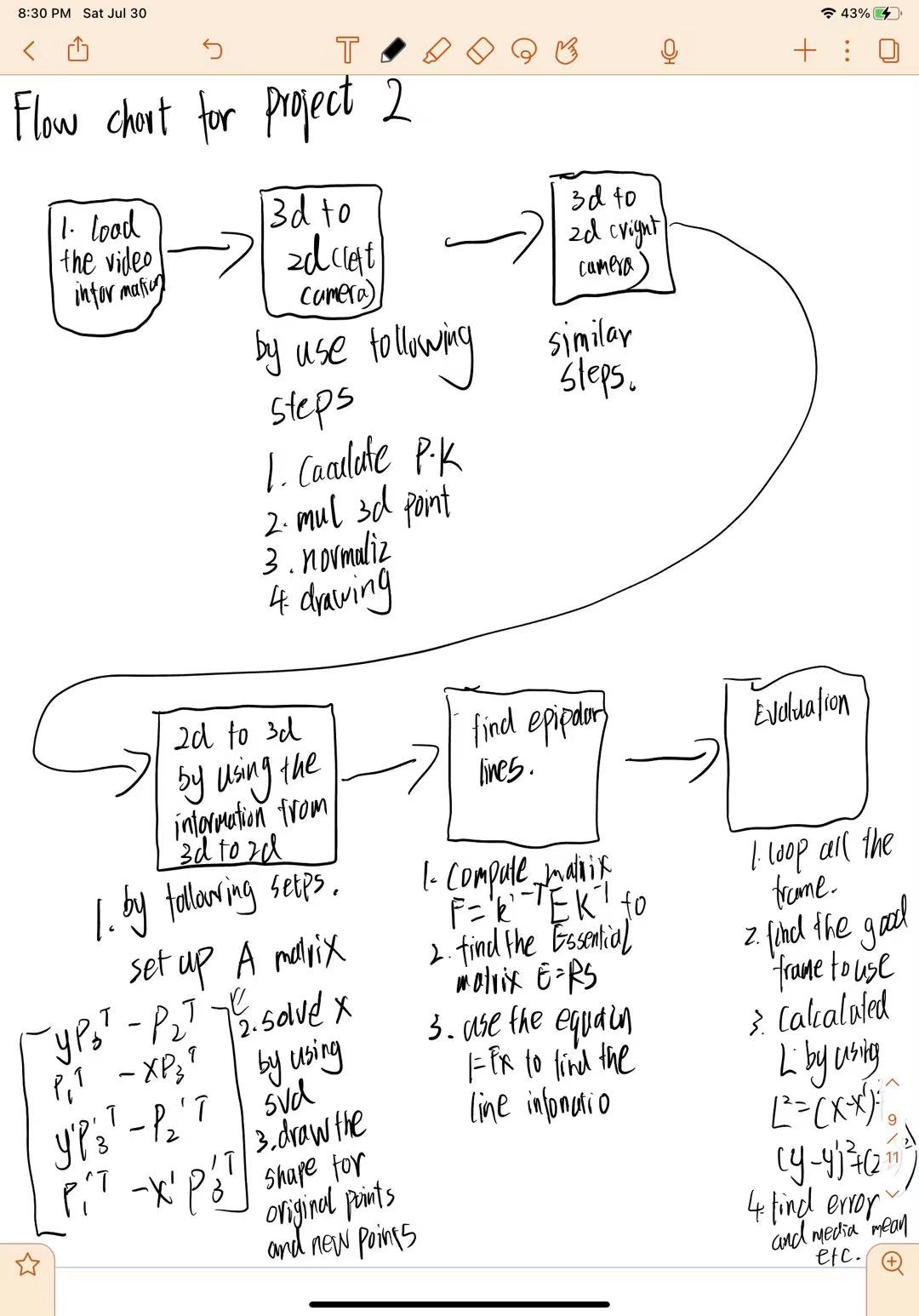
1a.

I think this project is about using computer vision knowledge to use two cameras projecting 3d to 2d and using 2d points to reconstruct 3d. Understanding what is epipolar lines, how to create them, and the lasting is an evaluation of the error and the algorithm efficiency.

1b.

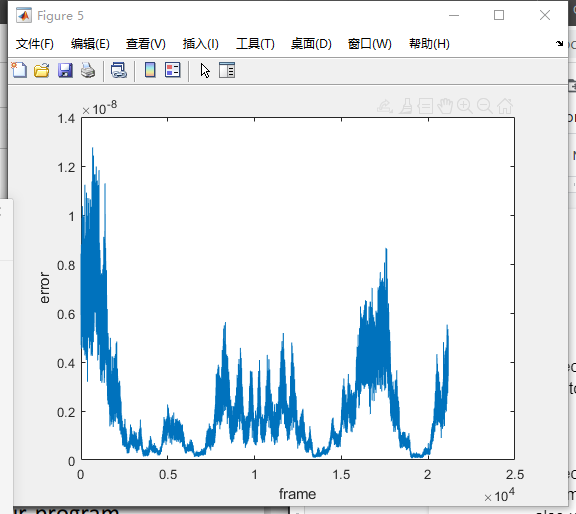
.

The internal parameters are also called intrinsic parameters, and it’s should be Kmat, and the, and external parameter also called extrinsic parameters. It's corresponding to the P = K[R|t]’s R and t which should be Rmat. The location of camera vue2 should be at [-4.45006e+03, 5.55792e+03, 1.9490e+03], and the location of camera vue4 should be at [4.42356e+03, 5.49036e+03, 1.88901e+03].

1c.

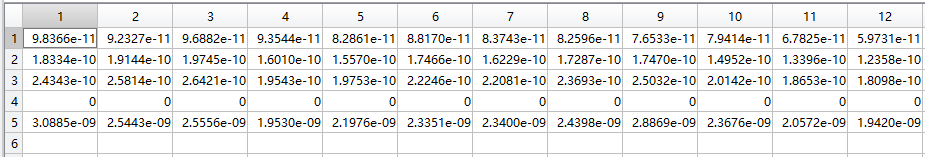
I observed when my program calculated the last part, which is the evaluation part, it becomes pretty slow. I think it’s because there are over twenty thousand frames that need to be calculated, and the 2d to 3d projection, 3d to 2d projection, and the epipolar line, these shapes are correct; however, the position they stand are not very accurate. I did try different pictures, and it works fine.

1d.

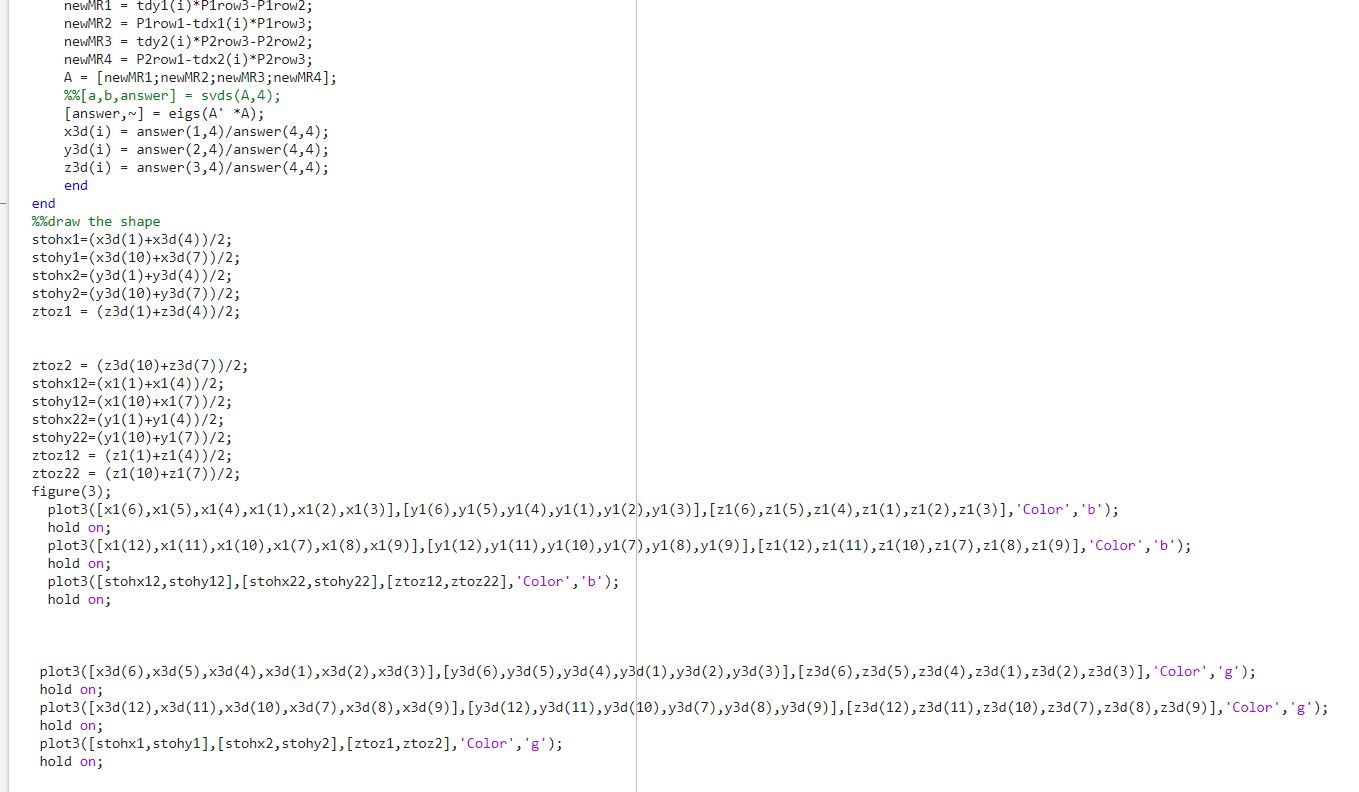
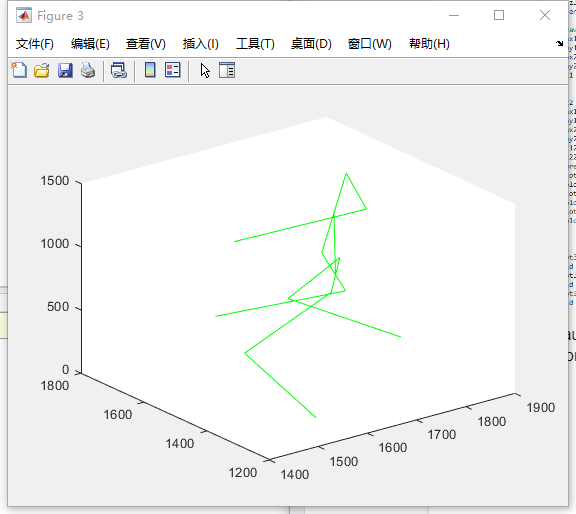


I do not have a single set of plot because I calculated all of the data at once and based on the graph we can clearly see the error are not constant, and there are several valleys between 250 to 750, and 1250 to 2000.

I also created a table to show values of min, max, standard deviation, average, and median. Base on the graph, the first row represents 12 joints’ median, the second row represents 12 joints’ mean, the third row represents standard deviation, the fourth row represents min, and the last row represents max.

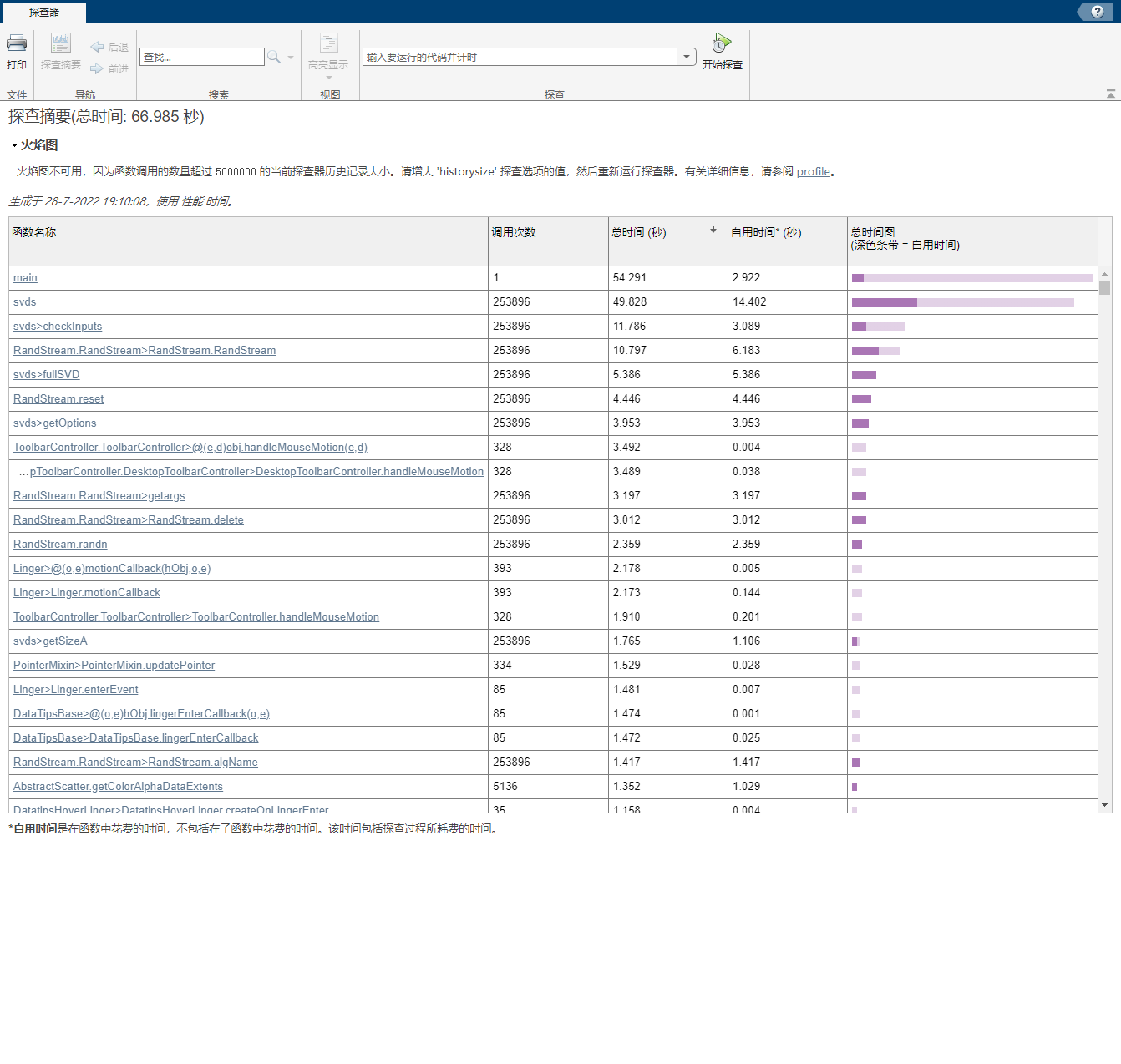


1e.



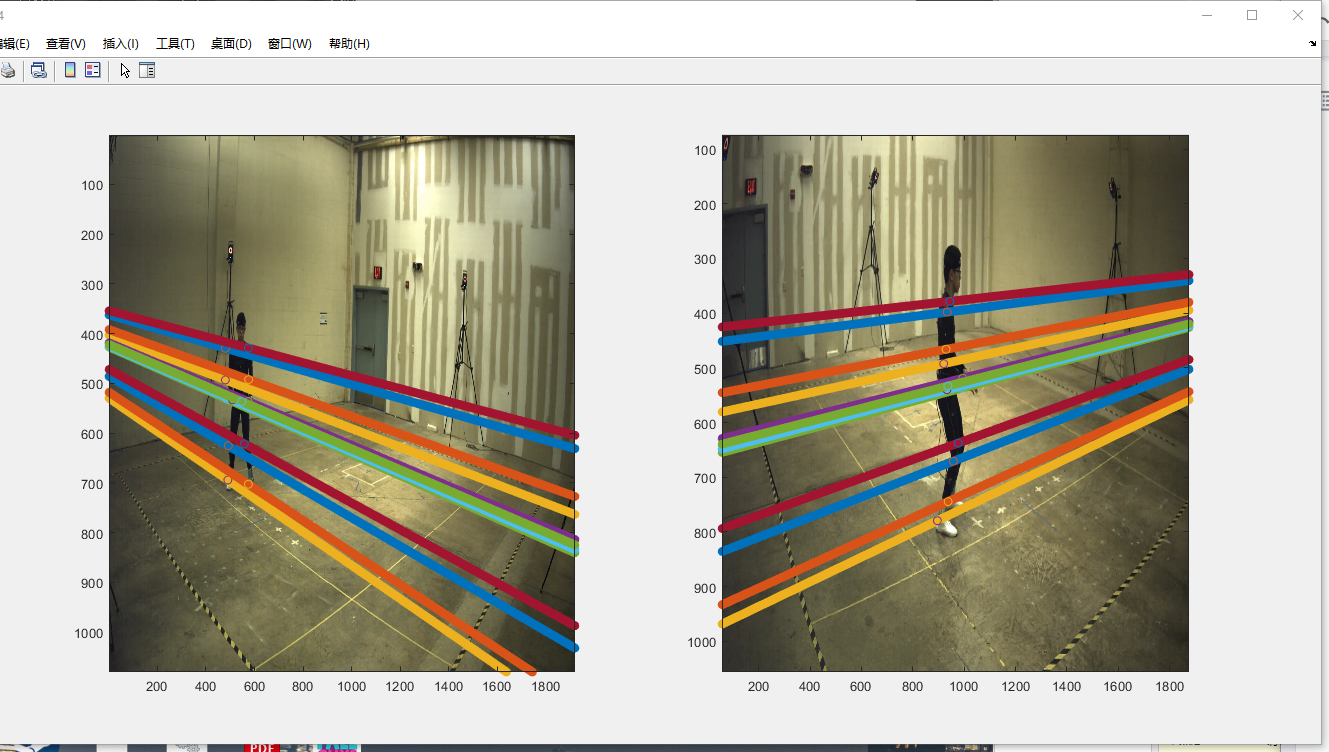
Because the error is around 10^-8; therefore, these two skeletons are so close that we only can see one of them, and this is the code for me to prove I did draw two shapes to compare.

1f.

Base 

Based on the graph above we could find out the most time we spend in this program is calculating the svd, and the reason it’s so slow it’s because we have over 20000 qualified frames to calculate; therefore, I decided to use the for loop to calculated each one, and because of time, I didn’t do any algorithm improvement.

1g.



The way I generate the epipolar lines is first to find out the matrix E, K’^-T, and K^-1, and then use the equation F=k’^-TEk^-1 to find the F, and then use l = Fx to calculate the line’s information.