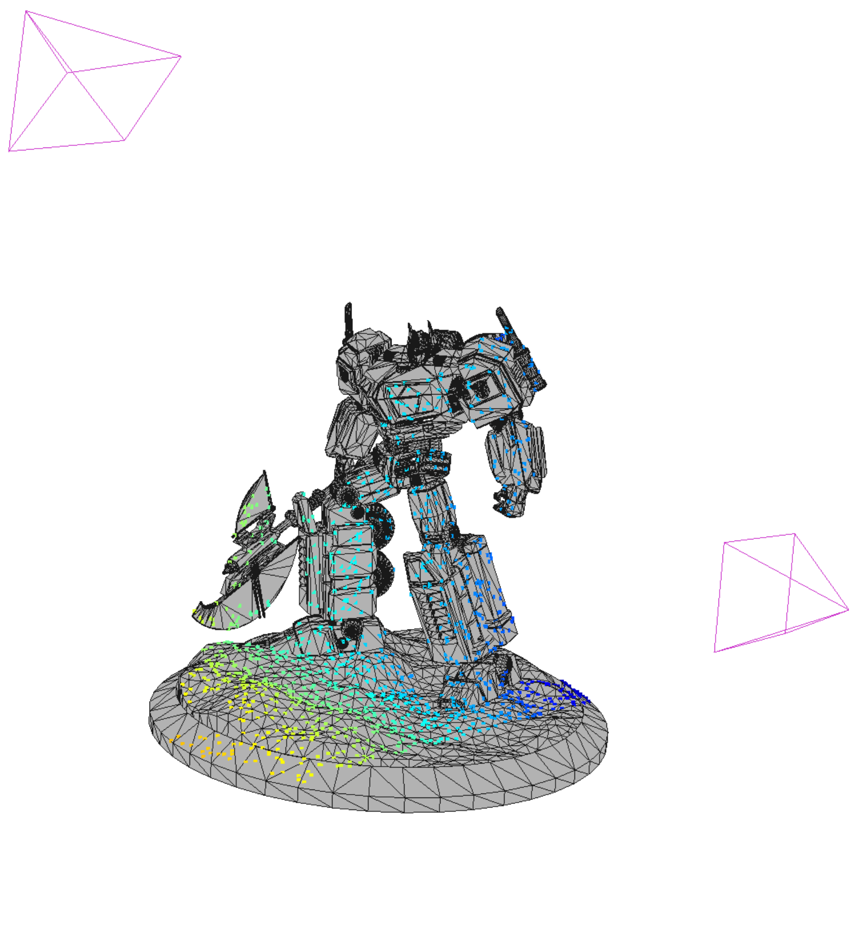
Computer Vision HW3

Jiaming Hu

Feb 19th, 2022

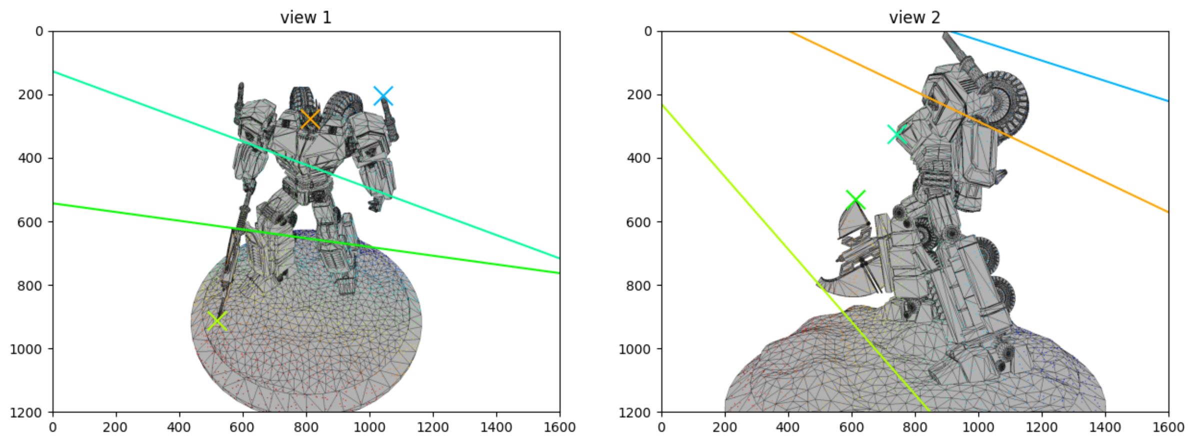
Problem 2



Problem 3

Included in code file

Problem 4



Problem 5,6

Included in code file

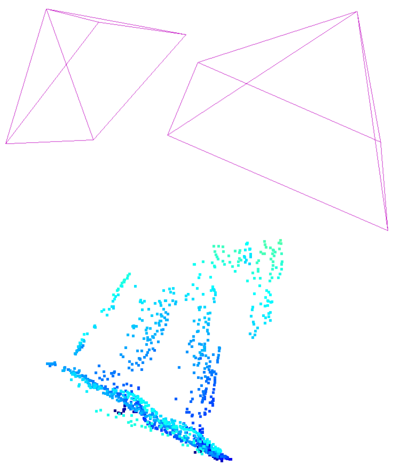
Problem 7

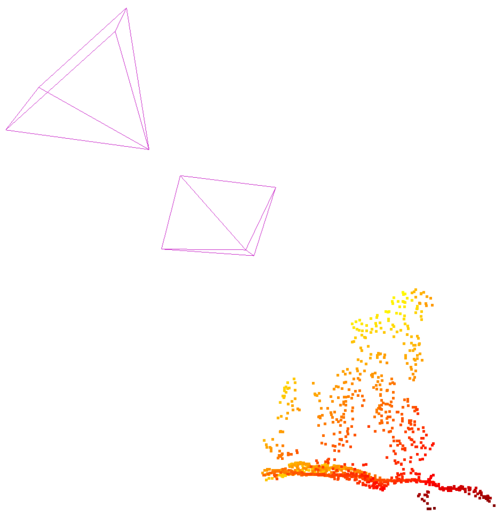
The reason of not normalizing using equilibration is because of the homogeneous property of our input points. It is clear that we use homogeneous coordinates for our input points, in this case are two 2d points. So each of those points has their last entry being 1, so we would have a $1{*}f_9$ as the last addition term of each equation, which f_9 is the last entry of the fundamental matrix F. If we use equilibration to normalized the equations, what happens is that it changes the 1 into different numbers, and are generally different for each equation, so each row of the matrix A would have their last entries be different, which would make our estimation of F inaccurate.

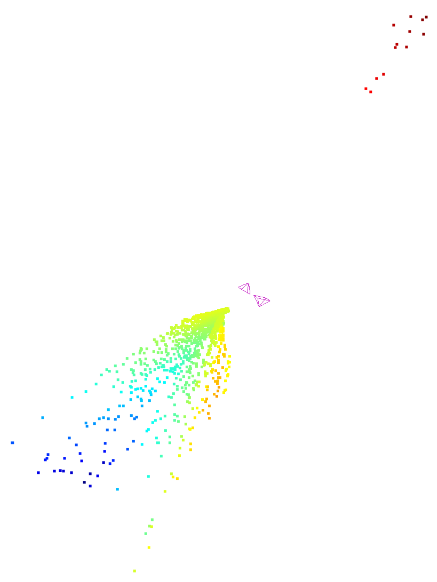
Problem 8,9

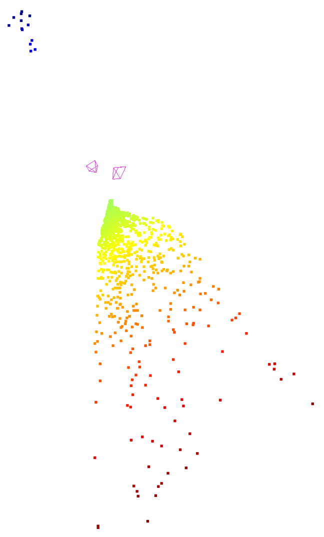
Included in code file

Problem 10



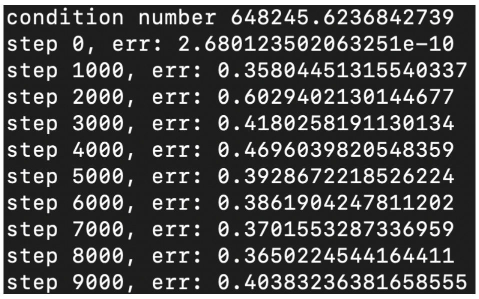






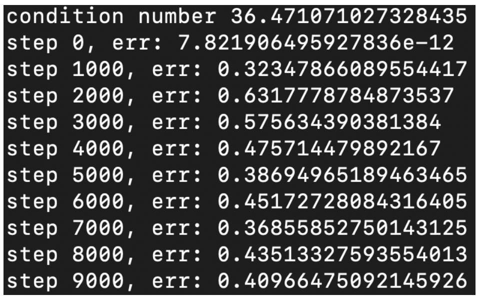
Problem 12

8-point algorithm (no noise):



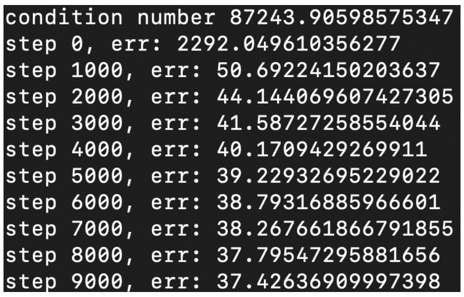
```
condition number 648245.6236842739
step 0, err: 2.680123502063251e-10
step 1000, err: 0.3580445131554033 7
step 2000, err: 0.6029402130144677
step 3000, err: 0.4180258191130134
step 4000, err: 0.4696039820548359
step 5000, err: 0.3928672218526224
step 6000, err: 0.3861904247811202
step 7000, err: 0.3701553287336959
step 8000, err: 0.3650224544164411
step 9000, err: 0.40383236381658555
```

normalized 8-point algorithm (no noise):



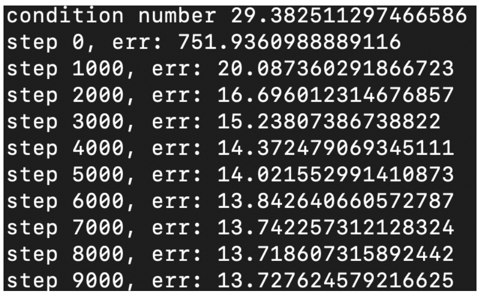
```
condition number 36.47107102732843 5
step 0, err: 7.821906495927836e-12
step 1000, err: 0.3234786608955441 7
step 2000, err: 0.6317778784873537
step 3000, err: 0.575634390381384
step 4000, err: 0.475714479892167
step 5000, err: 0.38694965189463465
step 6000, err: 0.45172728084316405
step 7000, err: 0.36855852750143125
step 8000, err: 0.43513327593554013
step 9000, err: 0.40966475092145926
```

8-point algorithm (with noise):



```
condition number 87243.90598575347
step 0, err: 2292.049610356277
step 1000, err: 50.69224150203637
step 2000, err: 44.144069607427305
step 3000, err: 41.5872725855404 4
step 4000, err: 40.1709429269911
step 5000, err: 39.22932695229022
step 6000, err: 38.79316885966601
step 7000, err: 38.267661866791855
step 8000, err: 37.79547295881656
step 9000, err: 37.42636909997398
```

normalized 8-point algorithm (with noise)



```
condition number 29.382511297466586
step 0, err: 751.9360988889116
step 1000, err: 20.087360291866723
step 2000, err: 16.696012314676857
step 3000, err: 15.23807386738822
step 4000, err: 14.372479069345111
step 5000, err: 14.021552991410873
step 6000, err: 13.842640660572787
step 7000, err: 13.742257312128324
step 8000, err: 13.718607315892442
step 9000, err: 13.727624579216625
```

We can see that when there is no noise, bundle adjustment is actually doing the opposite thing, which in some sense it is ”creating” some error then minimize the error it created.

This is expected because adjusts the positions of points as the program does not know that we started with the correct answer.

When there is noise, however, bundle adjustment works perfectly as we can see it minimized the projection error substantially.

Problem 13

I have attached a new version of code in .tar file. The nature of 10 camera SfM requires different function than what stereo SfM uses.

Besides trivial modifications, such as register new cameras using all poses and adding more x_is to the function, the main difference is that I created a function called "triangulate multiple camera", which uses numerical optimization to calculate the coordinates of 3d points when there are multiple cameras viewing the same point. I ran into 2 points creating problems when solving for SVD, because they have their last coordinates of smallest eigenvector being close to 0, as we need to divide the the coordinates by its last entry to bring it to homogeneous form. But that should be fine because we have a total of about 6000 points, it is normal that some of them are problematic. In this case, 2 out of 6000 points does not hurt our algorithm and result.

I have not yet figure out how to modify function "Align B to A" since I assume it requires using all p_is, so my answer is only true up to a scale. It gives good illustration nonetheless.

Here are some of my final visualizations of the statue: