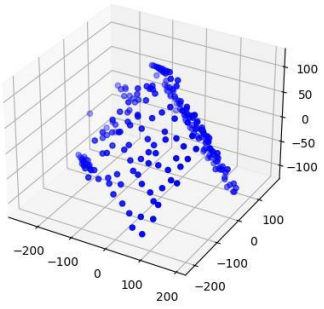
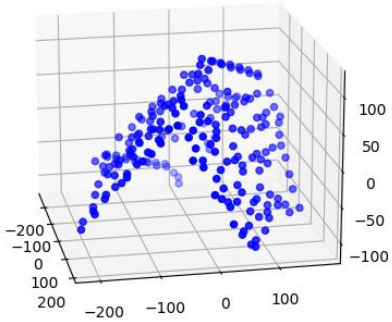
CS543/ECE549 Assignment 5

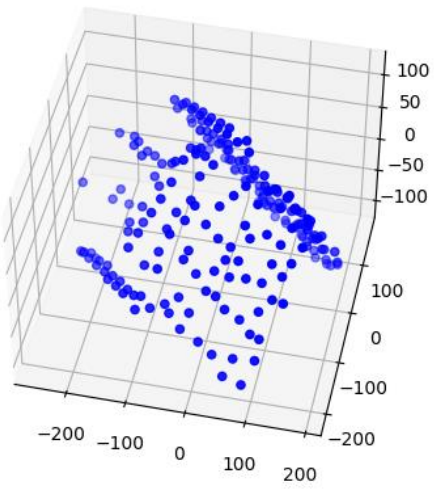
Name: Derek Yang
NetId: mcy3

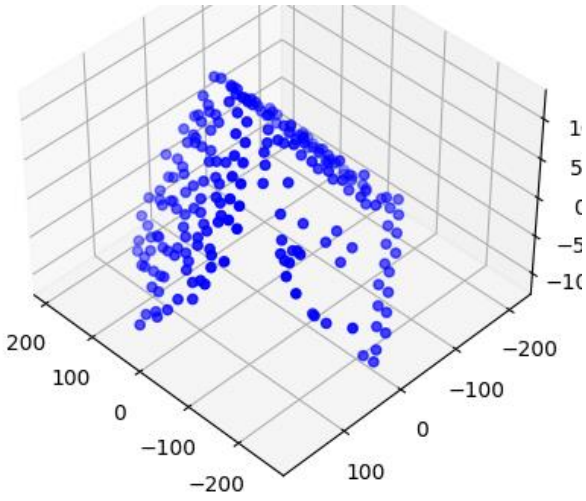
Part 1: Affine factorization

A: Display the 3D structure (you may want to include snapshots from several viewpoints to show the structure clearly). Report the Q matrix you found to eliminate the affine ambiguity. Discuss whether or not the reconstruction has an ambiguity.









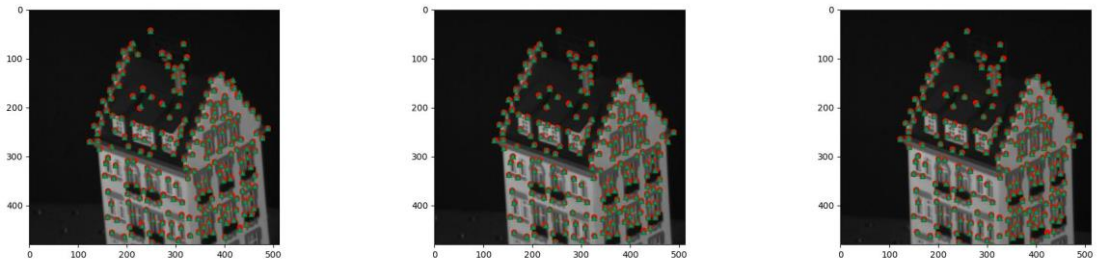
Q matrix

```
C:\Users\derek\Documents\GitHub\myCode\CS5
[[ 0.079173    0.          0.        ]
 [ 0.00396129  0.08433389  0.        ]
 [-0.00028779  0.00145269  0.04777143]]
```

As shown in the 3D model, there is clearly still ambiguity in our estimation.

B: Display three frames with both the observed feature points and the estimated projected 3D points overlayed.

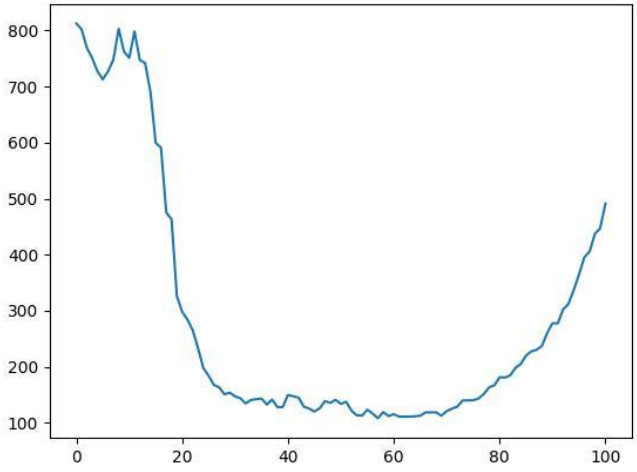
Observed points are in red dots and estimated points are in green triangles



C: Report your total residual (sum of squared Euclidean distances, in pixels, between the observed and the reprojected features) over all the frames, and plot the per-frame residual as a function of the frame number.

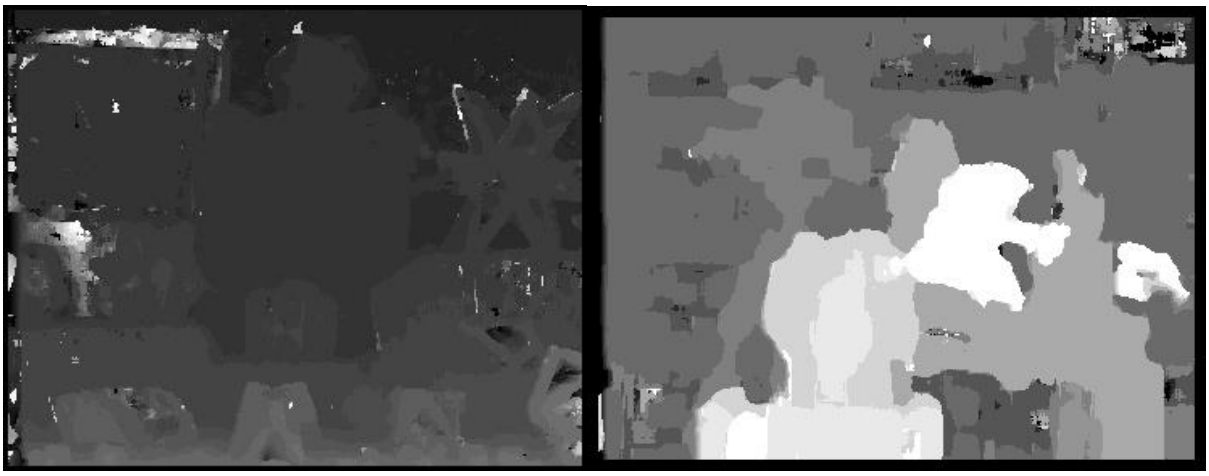
The total residual using sum of squared Euclidean distances is 28583.414640984298

Below is the graph of residual per frame with the residual on the y axis
and number of frames on x axis



Part 2: Binocular stereo

A: Display best output disparity maps for both pairs.

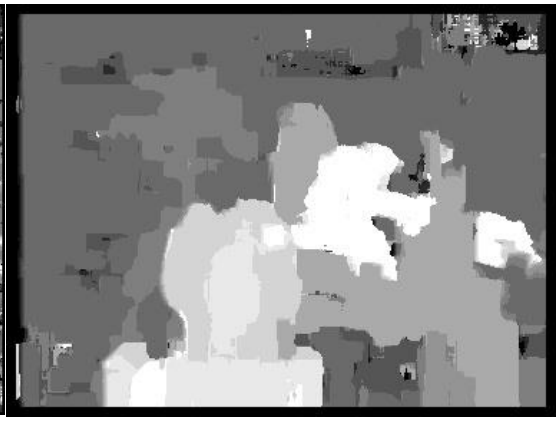


B: Study of implementation parameters:

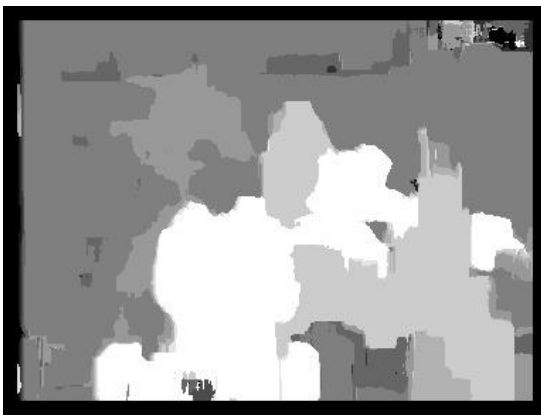
1. **Search window size:** show disparity maps for several window sizes and discuss which window size works the best (or what are the tradeoffs between using different window sizes). How does the running time depend on window size?

4*4 window

14*14 window



20*20 window



40*40 window

Execution time didn't vary too much for 4*4 and 12*12 with 14 seconds using the SSD method, while 20*20 window took 16 seconds and 40*40 window took 21 seconds. So increasing window size will ultimately increase the execution time since there is more calculation needs to be done in each pixel.

Increasing window size will result in a blurrier image with less and less detail but will be more tolerable with image noise. On the other hand, smaller window size will result in more detail, but the noise will be more significant.

I end up preferring with a window size around 14 because it leaves us with enough details without too many noises.

2. **Disparity range:** what is the range of the scanline in the second image that should be traversed in order to find a match for a given location in the first image? Examine the stereo pair to determine what is the maximum disparity value that makes sense, where to start the search on the scanline, and which direction to search in. Report which settings you ended up using.

```
half = int(windowSize/2)
if method == "SSD":
    minSSD = 99999999
    temp = 0
    lf = i-offset if i-offset > half else half
    rt = i+offset if i+offset < right.shape[1]-half else right.shape[1]-half
    for k in range(lf,rt):
        temp = np.sum((window-right[j-half:j+half,k-half:k+half])**2)
        if temp < minSSD:
            minSSD = temp
            disparity = i - k
```

I search on the scanline from the original coordinates of the left image with the offset going both sides of that pixel on the right image to insure I can find the correct window on the right image. (So basically [-offset, offset])

I did some testing to find out the correct offset in this case and end up with a offset of 12 which gave me a reasonable result without taking too much time (16 seconds).

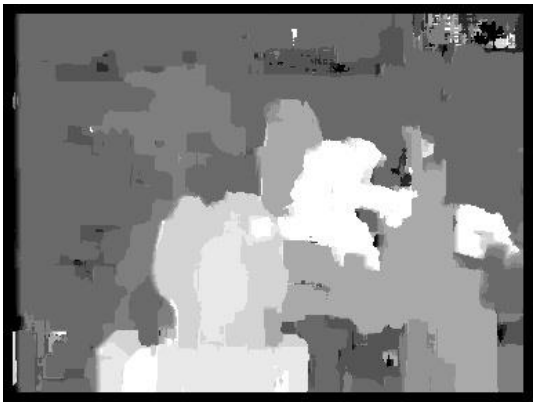
3. **Matching function:** try sum of squared differences (SSD), sum of absolute differences (SAD), and normalized correlation. Show the output disparity maps for each. Discuss whether there is any difference between using these functions, both in terms of quality of the results and in terms of running time.

SSD:

```
83      for i,j,window in sliding_window(img1,step,windowSize):
84          disparities[j,i] = abs(match(window,img2,i,j,"SSD",
```

PROBLEMS TERMINAL OUTPUT GITLENS JUPYTER DEBUG CONSOLE COMMENTS

```
C:\Users\derek\Documents\GitHub\myCode\CS543_CV\Lab5_Affine_factorization_binocular_stereo>python Yang_Derek_a5_p2.py
Processing.. 99% complete
 Time spent:  16.25936460494995
```



SAD:

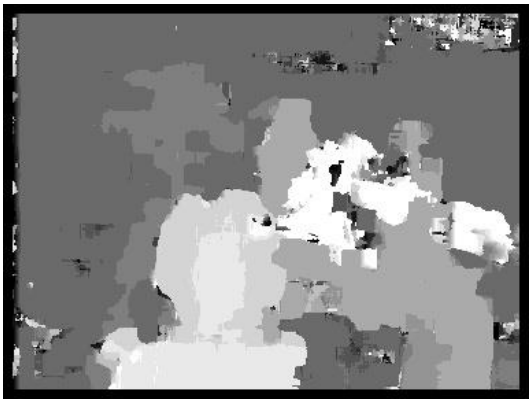
```
for i,j,window in sliding_window(img1,step,windowSize)
    disparities[j,i] = abs(match(window,img2,i,j,"SAD"
```

```
PROBLEMS  TERMINAL  OUTPUT  GITLENS  JUPYTER  DEBUG CONSOLE  COMMENTS

C:\Users\derek\Documents\GitHub\myCode\CS543_CV\Lab5_Affine_factorization_binocular_stereo>python Yang_Derek_a5_p2.py
Processing.. 99% complete
 Time spent:  16.100694179534912
```

NCC:

```
    disparities[j,i] = abs(match(window,img2,i,j,"NCC"
```

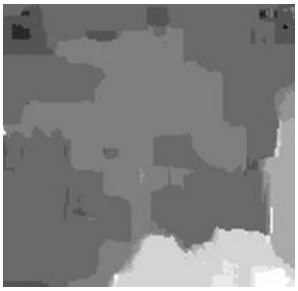
```
PROBLEMS  TERMINAL  OUTPUT  GITLENS  JUPYTER  DEBUG CONSOLE  COMMENTS

C:\Users\derek\Documents\GitHub\myCode\CS543_CV\Lab5_Affine_factorization_binocular_stereo>python Yang_Derek_a5_p2.py
Processing.. 99% complete
 Time spent:  55.72595167160034
```



SSD and SAD has a similar run time of 16 seconds while NCC has a significant longer time of 55 seconds due to the amount of computation needed.

SAD seems to be more tolerable to noise as it has the clearest edge of all the different methods. NCC gives out the most detail than any of the methods, but that also means the result will contain more noises than the other two method.

C: Discuss the shortcomings of your algorithm. Where do the estimated disparity maps look good, and where do they look bad? What would be required to produce better results? Also discuss the running time of your approach and what might be needed to make stereo run faster.



The camera on the back looks good with good details and clean outlines.

There is still significant noise over the image which doesn't look too good.

The neck of the lamp got lost because it was too thin. Which I don't think I can avoid.

In order for a better result, I believe we need more preprocessing like a blur to lower the noise of the background. Or even blurring after the disparity map is created.

Time wise, I down sampled the larger size image to make it faster. The most time spent would be the numpy methods that I called which usually takes a lot of times to executed. A hand carved version of those methods would certainly be faster than the numpy methods.

```
np.sum((window_m/np.linalg.norm(window_m))*(right_m/np.linalg.norm(right_m)))
```

```
np.sum(np.abs(window-right[j-half:j+half,k-half:k+half]))
```