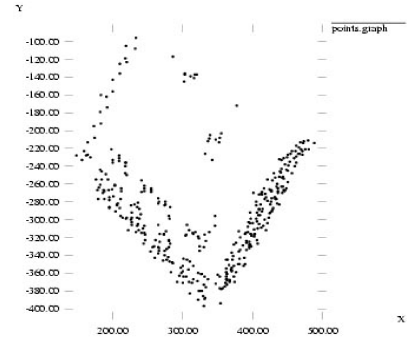
CS543/ECE549 Assignment 5

Name: fanmin shi
NetId: fshi5

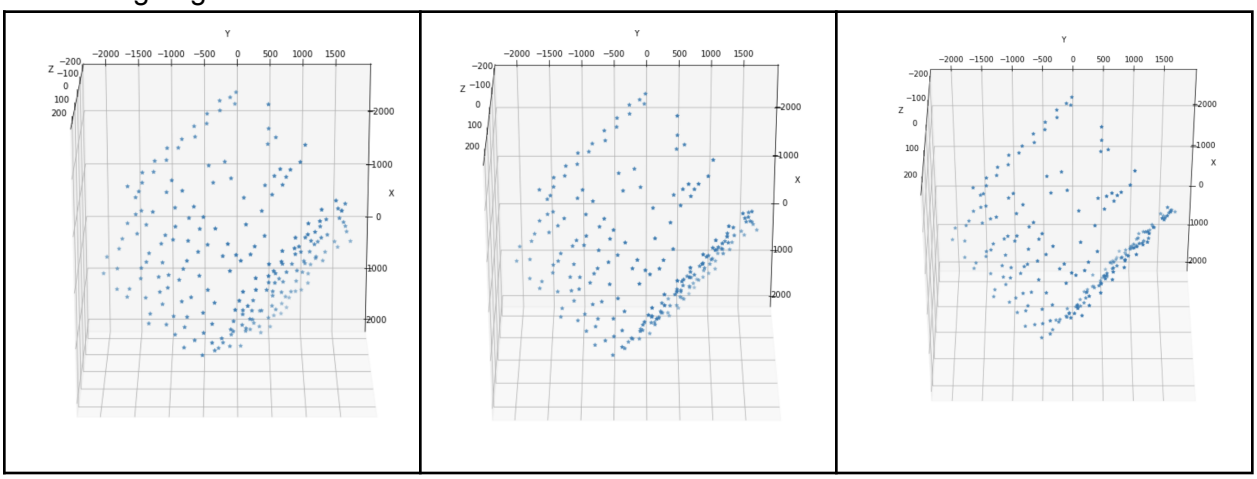
Part 1: Affine factorization

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.

Ground truth from the slide.



The view angle is tilted 10 degrees from the Z axis starting at 110 degree downward from picture 1-3 to make it look the ground truth. I use `ax.view_init(elev=130, azim=0)` to adjust the viewing angle of the Z axis.



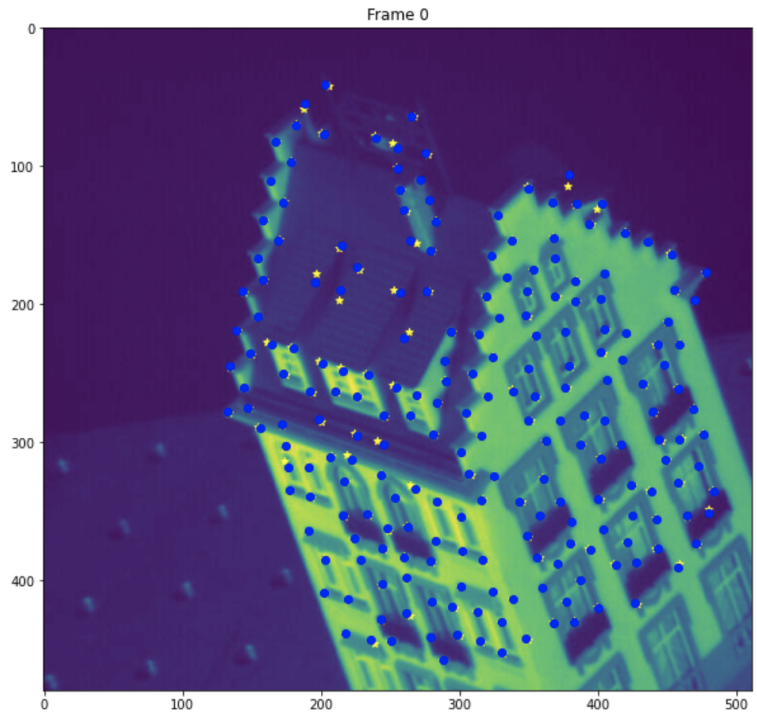
Q matrix is the following

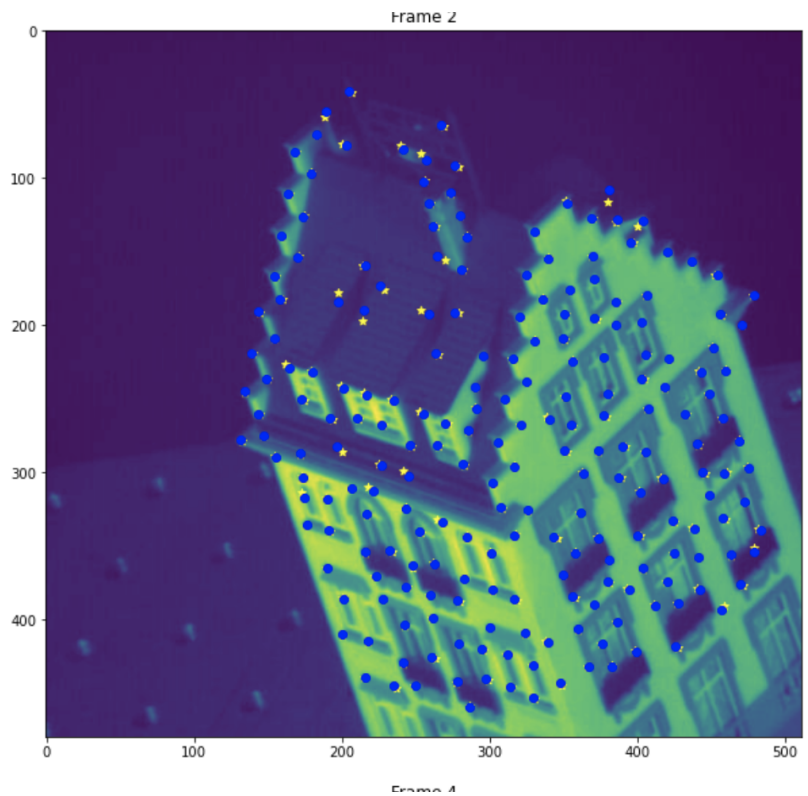
```
[[ 7.94778741e-03  0.00000000e+00  0.00000000e+00]
 [ 4.20523075e-19  8.53955215e-03  0.00000000e+00]
 [-1.23516354e-18  2.29743571e-18  2.53757864e-02]]
```

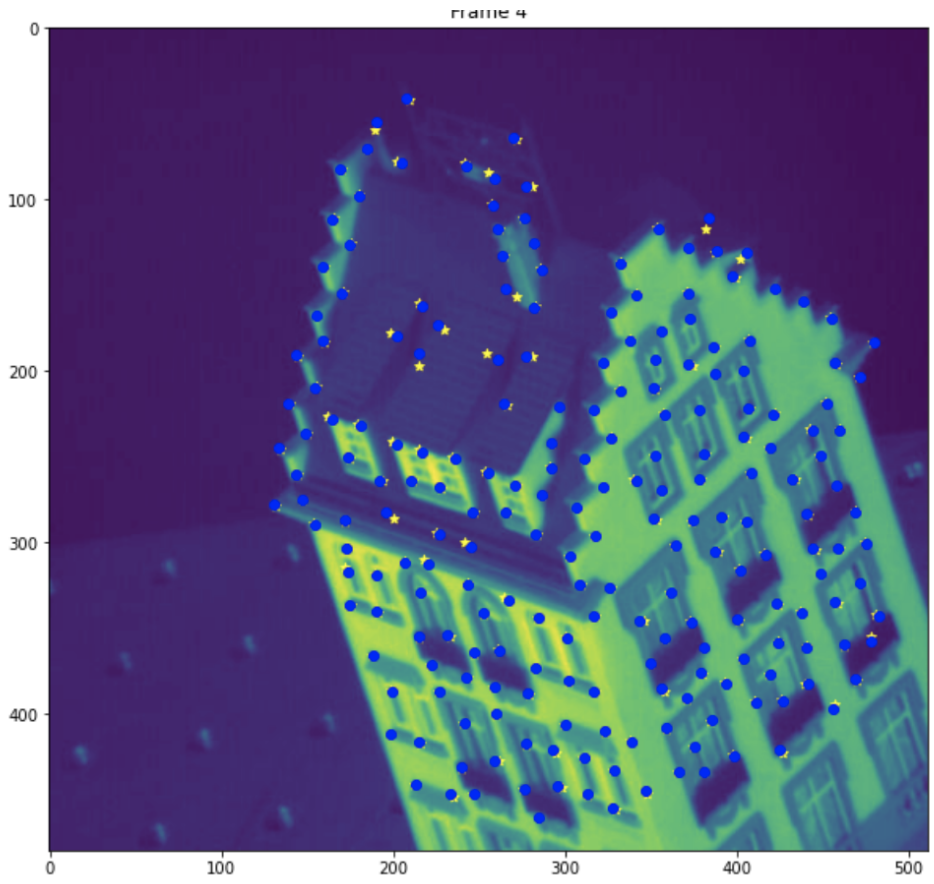
Based on the points cloud reconstruction, I would say that the reconstruction has an ambiguity. I found that the Z value is between (-200, 200) and X and Y has values between (-2000, 2000). As you can see, the Z value is not on the same scale as X and Y. If you plot with the correct axis without scaling by pyplot, you can see that that depth is quite flat.

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

The blue dots are the ground truth and the Yellow dots are the projected one.

Frame 0	

Frame 2	 Frame 2 0 100 200 300 400 0 100 200 300 400 500

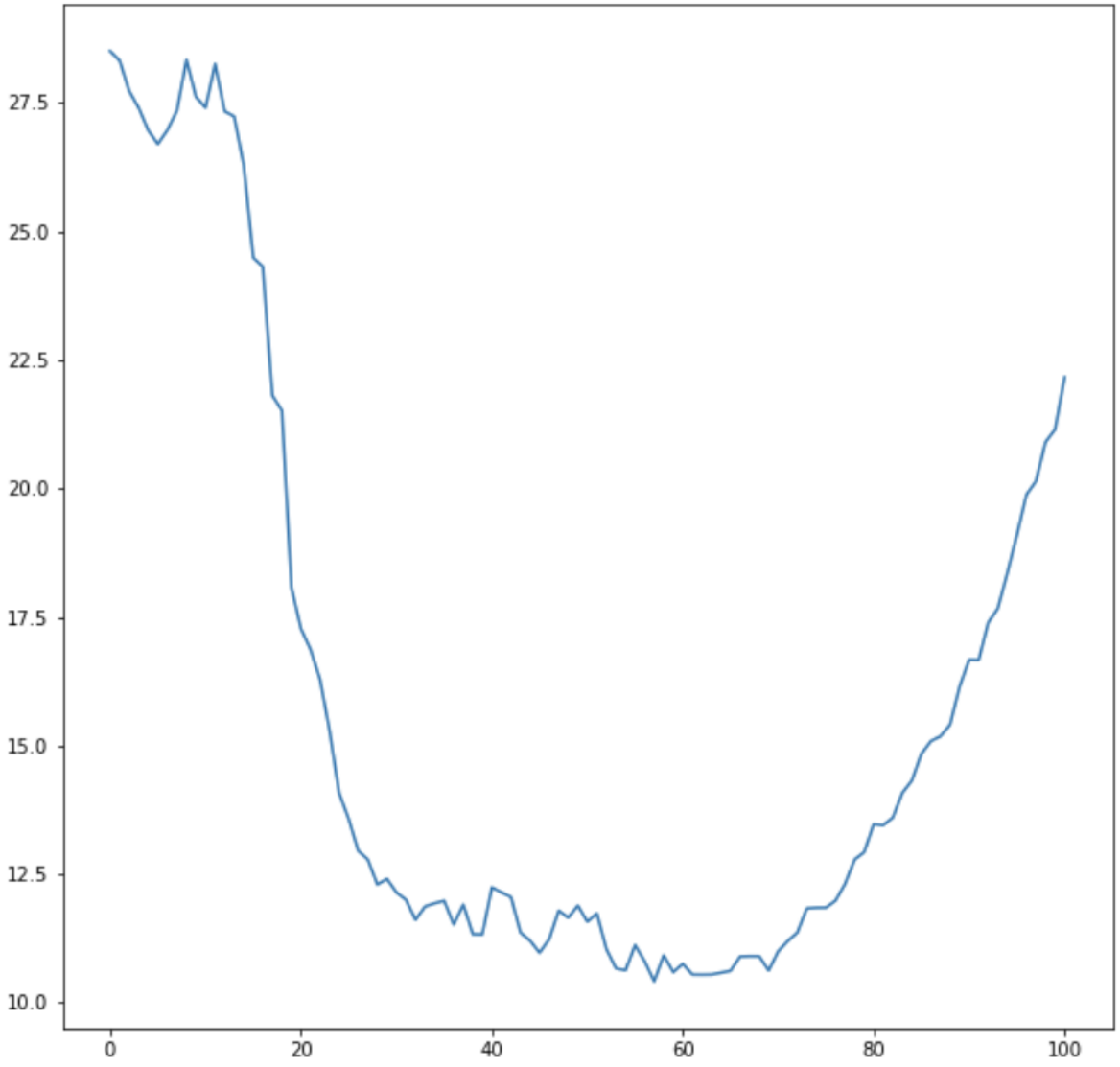
Frame 4	

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.

To calculate total residual, I first used `np.linalg.norm()` the observer and projected pts from each frame. Then I sum over residuals from each from over all frames to obtain the answer below.

Total residual: 1591.6427326100998

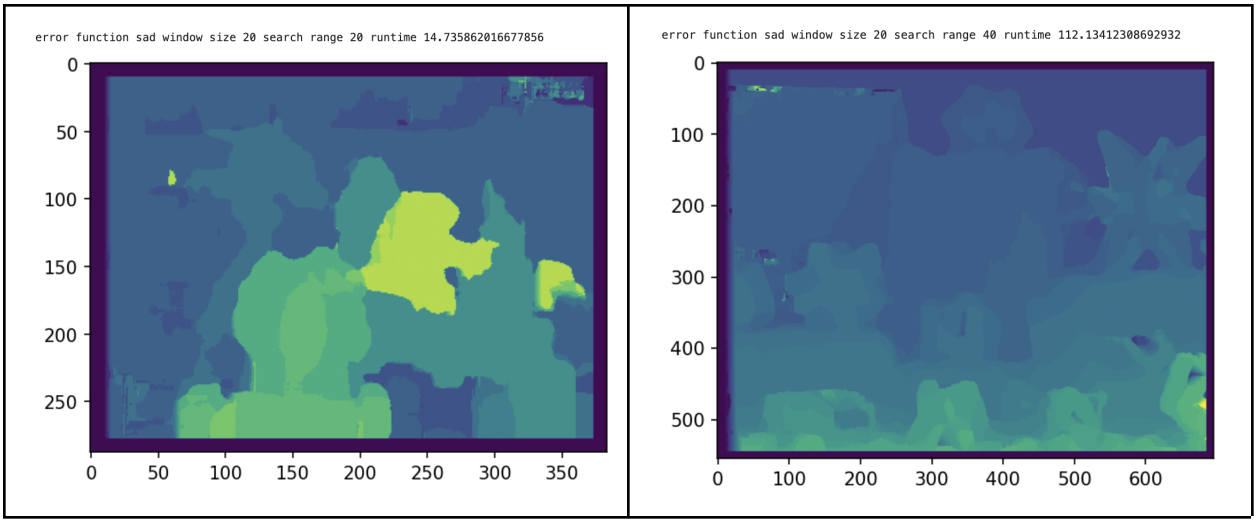
Frame Vs Residual plots.



Part 2: Binocular stereo

A: Display best output disparity maps for both pairs.

Tsukar	Moebius



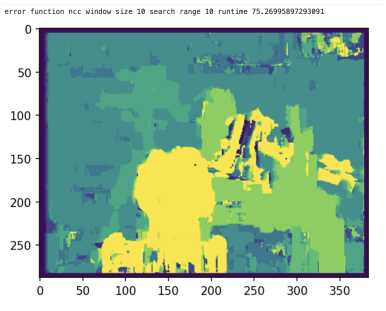
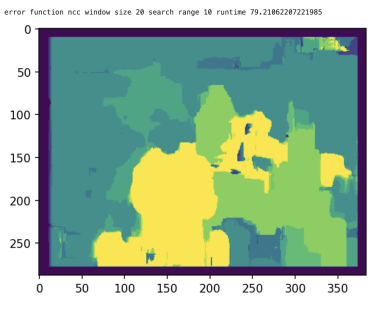
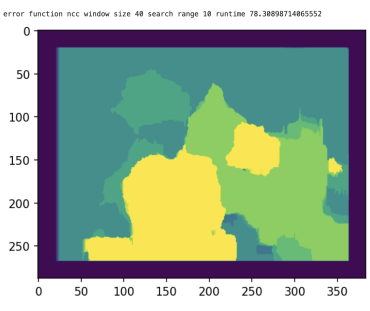
For tsubar image, I found that the SAD with windows=20 range=20 produces the best output. I found the edge is quite smooth and depths between objects quite distinct. You can see that the lamp is closest and very bright.

For Moebius, I found window size =20 also produce the best result. The vertical and horizontal edges are quite smooth and depth color is quite uniform; the objects in front are brighter than the ones at the back.

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?

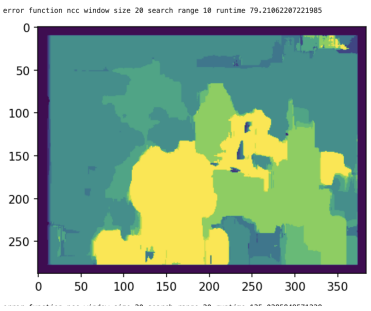
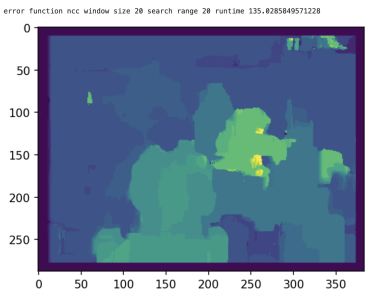
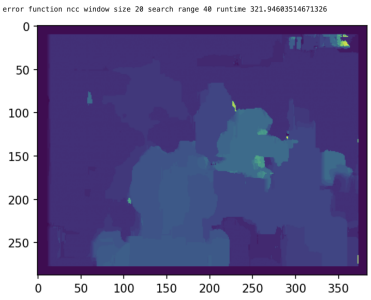
I will use the outputs from NCC and image tsukuba to compare the window sizes.

Window size = 10	Window size = 20	Window size = 40
		

Base on the disparity map, when windows size is 20, it sits in between too much details vs too less details. It seems to me 20 is a good window size. The trade off when choosing the windows is whether you want more detail (along with more noises) or less detail (less noise). If you want more detail, you can see lots of objects along with their depth. But at the same time, you will get random noises. If you don't want to have noises, then you can choose a larger window size. However, you will see less objects and their corresponding depths. The runtime slightly increased from 75 -> 79 -> 78 seconds. I would say window size is not a dominant factor of run time increase.

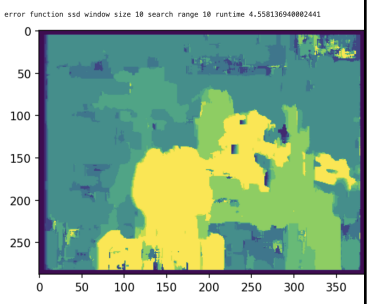
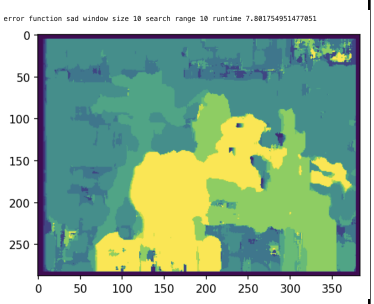
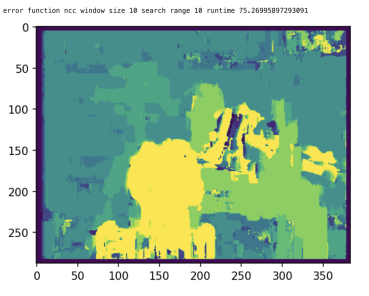
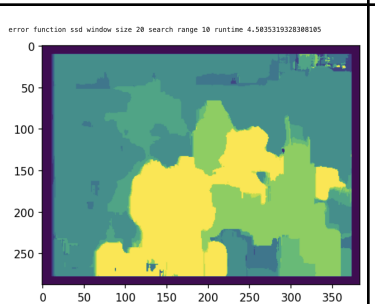
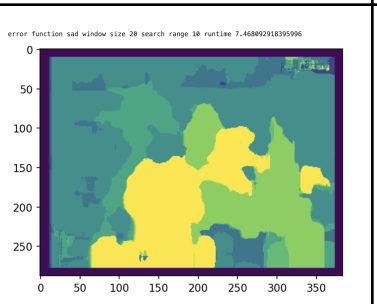
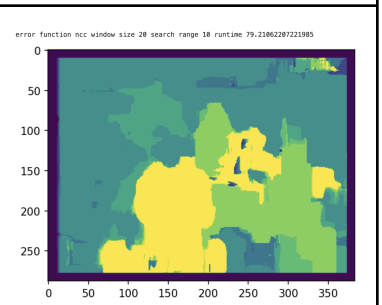
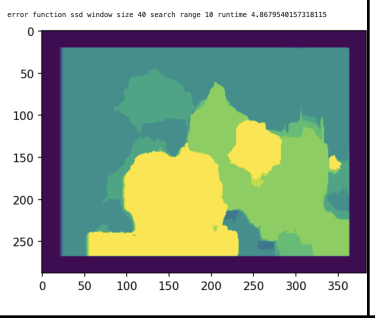
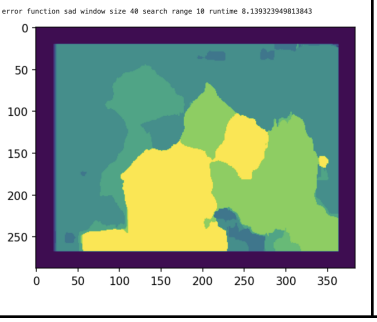
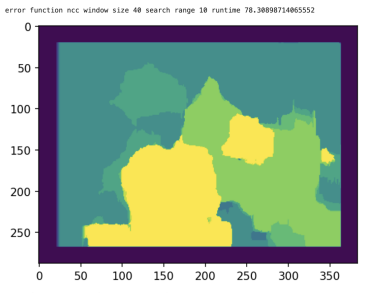
2. **Disparity range:** what is the range of the scanline in the second image that should be traversed in order to find a match for a given location in the first image? Examine the stereo pair to determine what is the maximum disparity value that makes sense, where to start the search on the scanline,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

I will use the outputs from NCC and image tsukuba to compare the range.

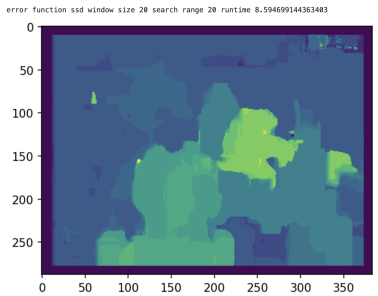
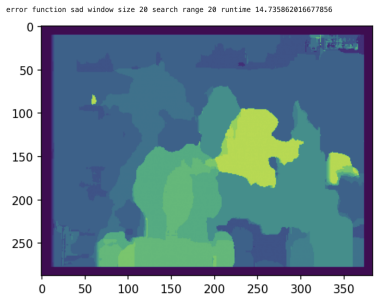
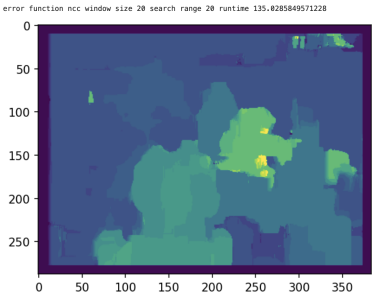
Range=10	range=20	range-40
		

The way I set up my algorithm is that I always scan from left to right. In order to do that, I will make sure that the right image is on the left side of the left image. So scanning from left to right will find the match. By doing that, it simplifies my algorithm a bit so that the scanning does have to go either direction. By using search range=20, I noticed that the lamp depth map is more uniform than that of range=10. The big green blob disappears. By comparing range=20 to range =40, the intensity somehow decreases probably due to py plot function. However, the overall depth and shape looks the same. If we look at the runtime of each range, it is 79 -> 135 -> 321 seconds. So the search range definitely impacts the runtime. I would say range = 10 is sufficient to a good enough depth map while the runtime is much faster.

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	SAD	NCC
windows= 10			
windows= 20			
windows= 40			

Based on my outputs, I don’t see a significant difference between each approach. The quality looks the same variety and different window sizes.

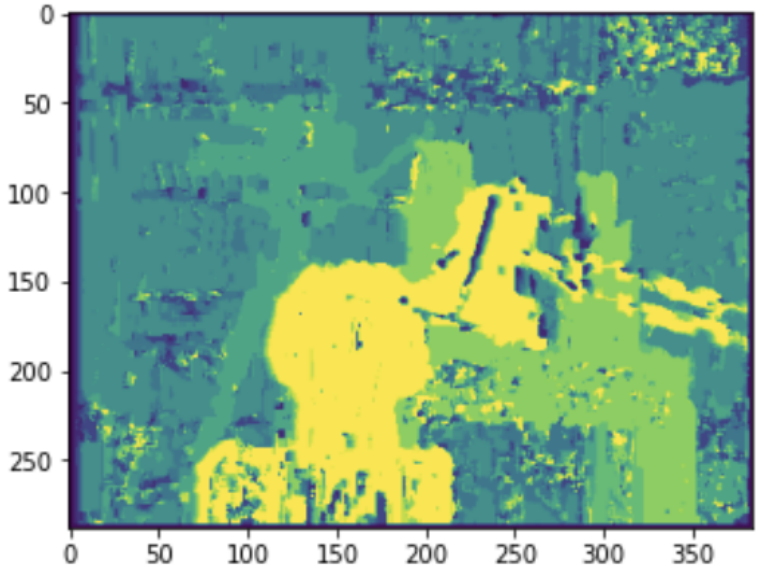
SSD range=20	SAD range=20	NCC range=20
		

By comparing the results using range=20, the results look very similar to me. However, it seems like SAD is producing a more uniform depth map for the lamp than other twos. However, the runtime is drastically different. It seems like window size doesn't change the runtime for SAD and SSD as well. They both run 4 seconds and 8 seconds regardless of the window size increase. When comparing change of search range, when doubling the range from 10 -> 20, SAD and SSD increase runtime to 8 and 14 seconds respectively, However, comparing run time from SAD, SSD, and NCC. They are 8, 14, 135 seconds. So NCC runs much slower yet not producing a higher quality result. I would prefer to use SAD for this because it produces similar quality but at the minimum runtime comparing to the others.

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.

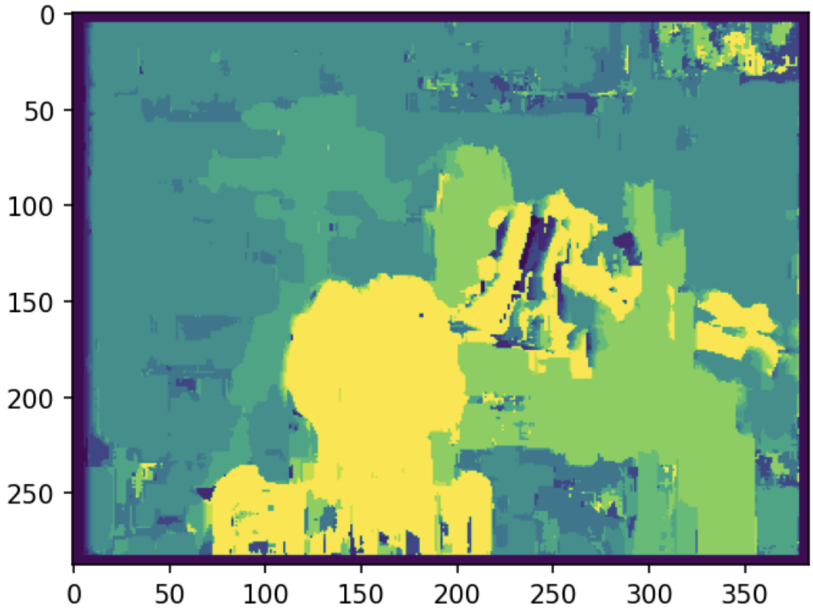
For tsukuba images, the disparity starts to look bad when the window size is 5 and search range is 10. This creates a lot of noise like this.

error f ssd size 5 search range 10



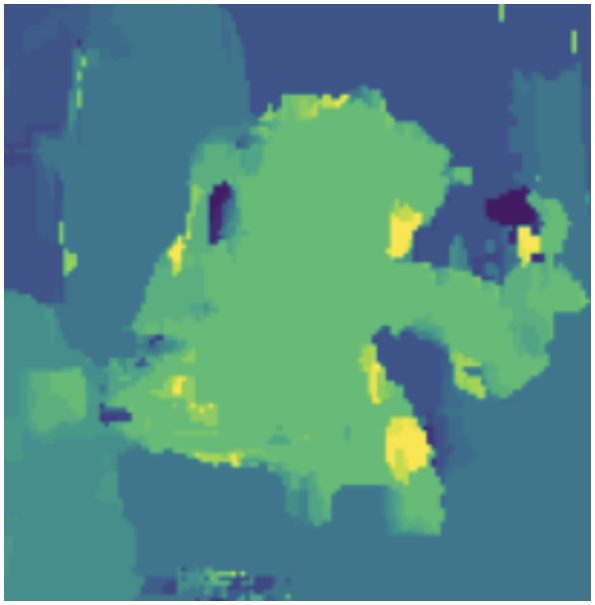
error f ssd size 5 search range 20

However, the image starts to look good when the window size is increased to 10.



Still you can see noises in the lamp as shown below.

When I keep the window size but increase the search range to 20, the noise on the lamp is gone. This makes sense because the longer search can find a better match.

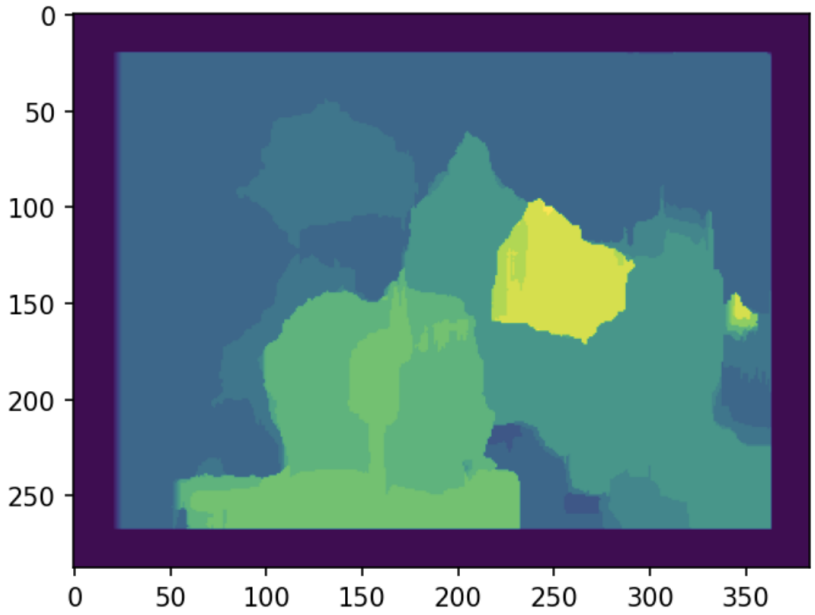


With the windows size = 10, I see unsmooth edge as shown below

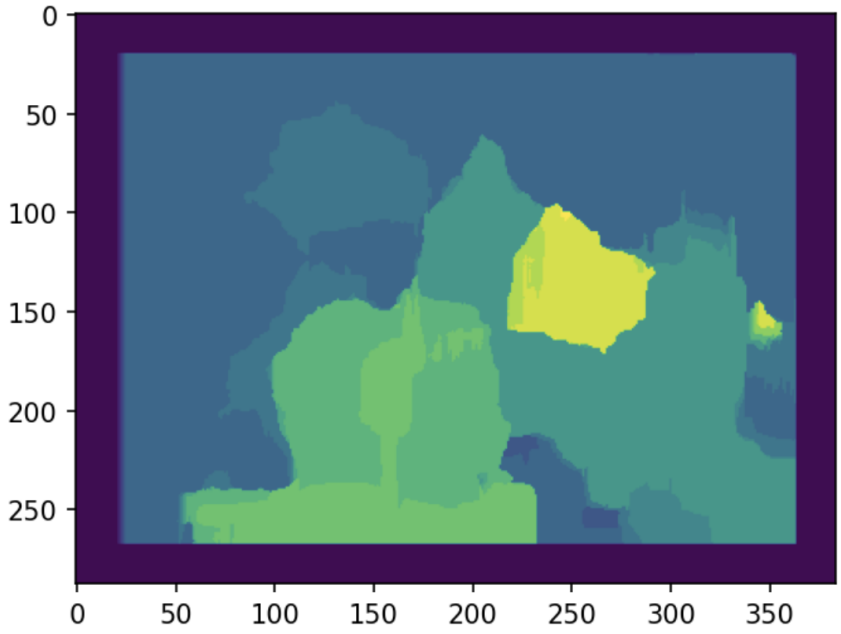
However when windows size = 20, the edge is more smooth as shown below.

I also noticed that the depth maps look quite similar when the window size and search range is above a threshold. For example, windows = 40 and range = 40 looks the same as windows size = 40 and range = 20. However, the runtime is twice as long. So picking the right threshold will decrease the runtime but not the quality.

error function ncc window size 40 search range 20 runtime 158.86934208869934



error function ncc window size 40 search range 40 runtime 303.3272030353546



In conclusion, if the window size and search range is small, it tends to produce lots of noise because it is unable to find the right match. However, if the window size and search range increases too much, the result will look similar but the runtime is too large.

The runtime for tsukuba is slow when using NCC. It took about 321.94603514671326

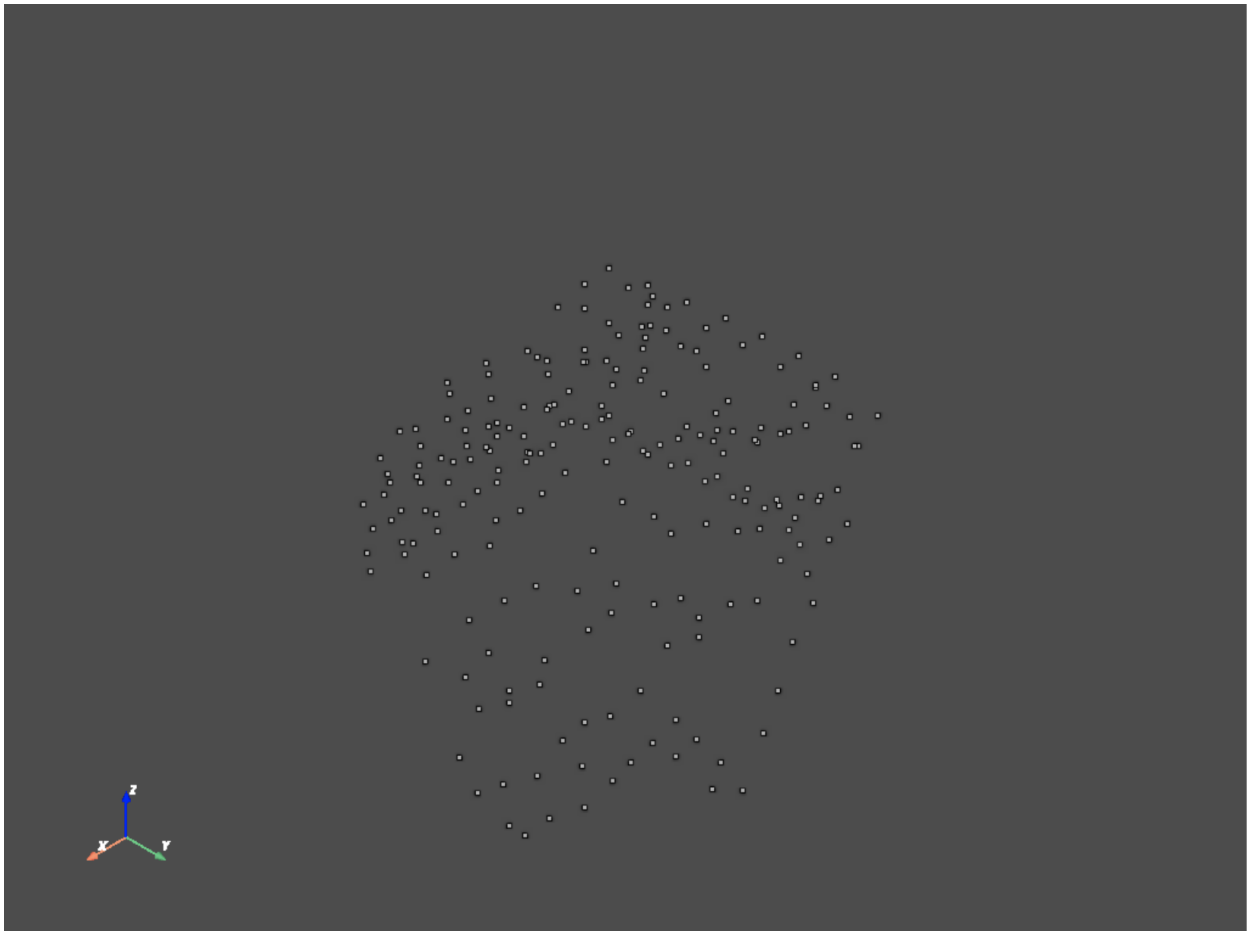
Seconds for window size 20 search range 40. This is quite computationally inefficient on larger images. To improve on the speed for NCC, I would downsample the image and calculate the depth map and then up sample the depth map to the original size. The speed increase will be (scaling factor)^2. So if I scale down the image by 2 which means (height/2, width/2), then the speed increase will be 4 because I have 4 times less pixels to iterate.

Part 3: Extra Credit

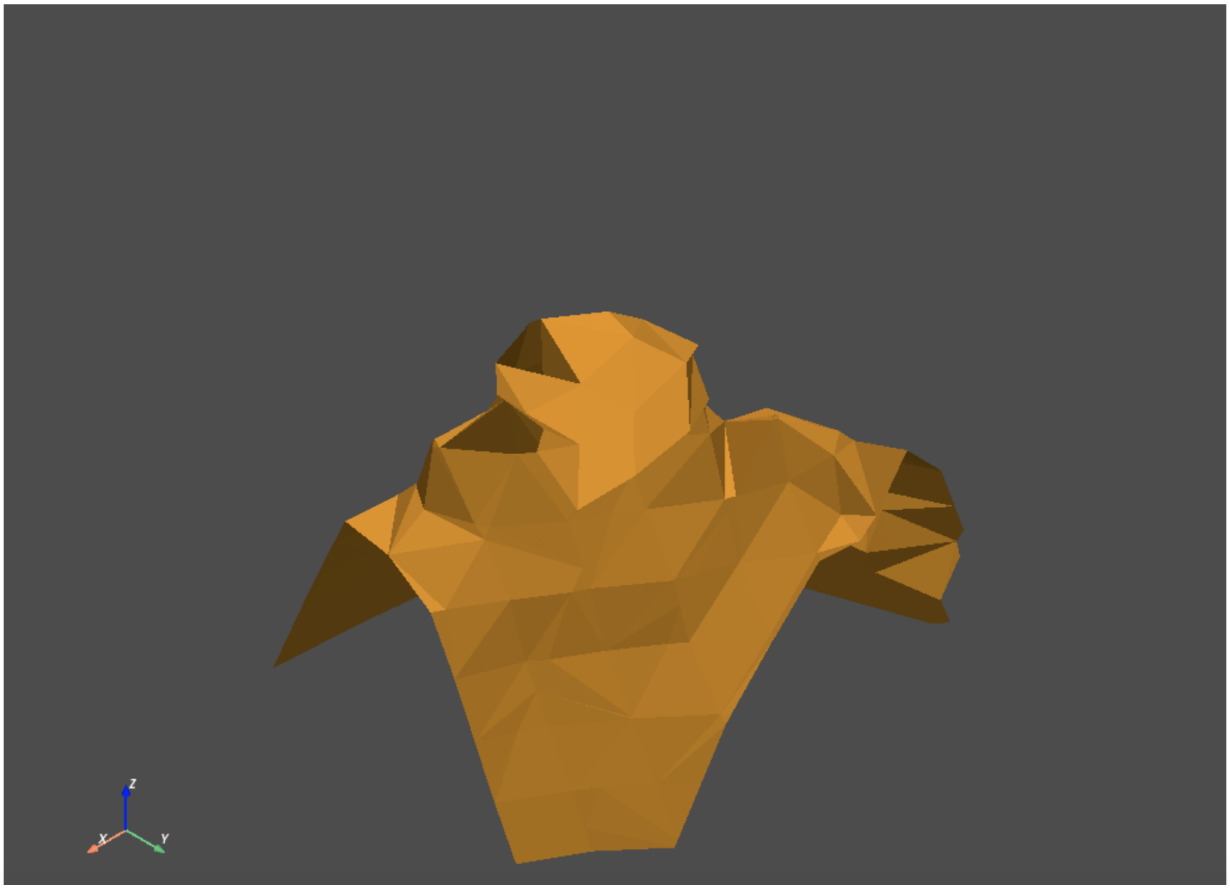
Part 1: Create a textured 3D model of the reconstructed points.

I first generated 3D points by computing S = Q^-1*S. I noticed that the Z magnitude is a lot less than the X and Y. So when plotting Z value looks small. So I normalized Z with respect to X and Y by a scaling factor computed from max of X / max of X. e.g `S[2] = S[2]*np.max(S[0])/np.max(S[2])`
I visualized the 3D points by converting the S points into point clouds `point_cloud = pv.PolyData(S.T)` using pyvista as pv.

The point cloud looks like the following



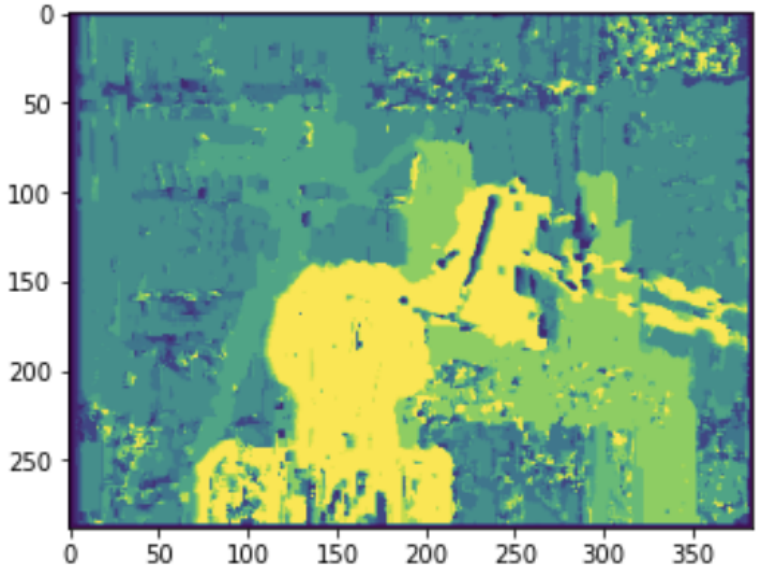
Then I create a mesh using the pyvista function `mesh = point_cloud.reconstruct_surface()`
The result is the following.



Part 2: Point Clouds Computation

I created the points cloud for the following depth image where the error function is SSD and window size=5 and search range =10. Even though this image is a bit noise but it does find depth in a granular level due to small window size.

```
error f ssd size 5 search range 10
```




```
error f ssd size 5 search range 20
```

The reference code is from this blog post [Estimate Point Clouds From Depth Images in Python | by Chayma Zatout | Better Programming]

The idea is simply to compute correct X, Y, Z points based on

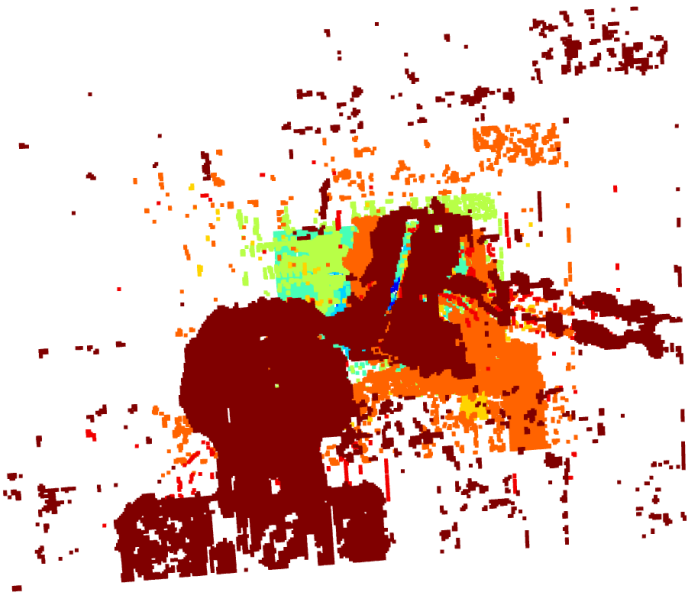
$$\begin{cases} z = depth(i, j) \\ x = \dfrac{(j - c_x) \times z}{f_x} \\ y = \dfrac{(i - c_y) \times z}{f_y} \end{cases}$$

Where cx and cy correspond to camera centers and fx and fy correspond to focal length. The code is below

```
def compute_points_cloud(d):
    # normalize depth value to 255
    d /= np.max(d) * 255
    # use the camera parameters from the blog posts.
    # Depth camera parameters:
    # Depth camera parameters:
    FX_DEPTH = 5.8262448167737955e+02
    FY_DEPTH = 5.8269103270988637e+02
    CX_DEPTH = 3.1304475870804731e+02
    CY_DEPTH = 2.3844389626620386e+02

    pcd = []
    height, width = d.shape
    for i in range(height):
        for j in range(width):
            if d[i][j] == 0:
                continue
            z = d[i][j]
            y = (j - CX_DEPTH) * z / FX_DEPTH
```

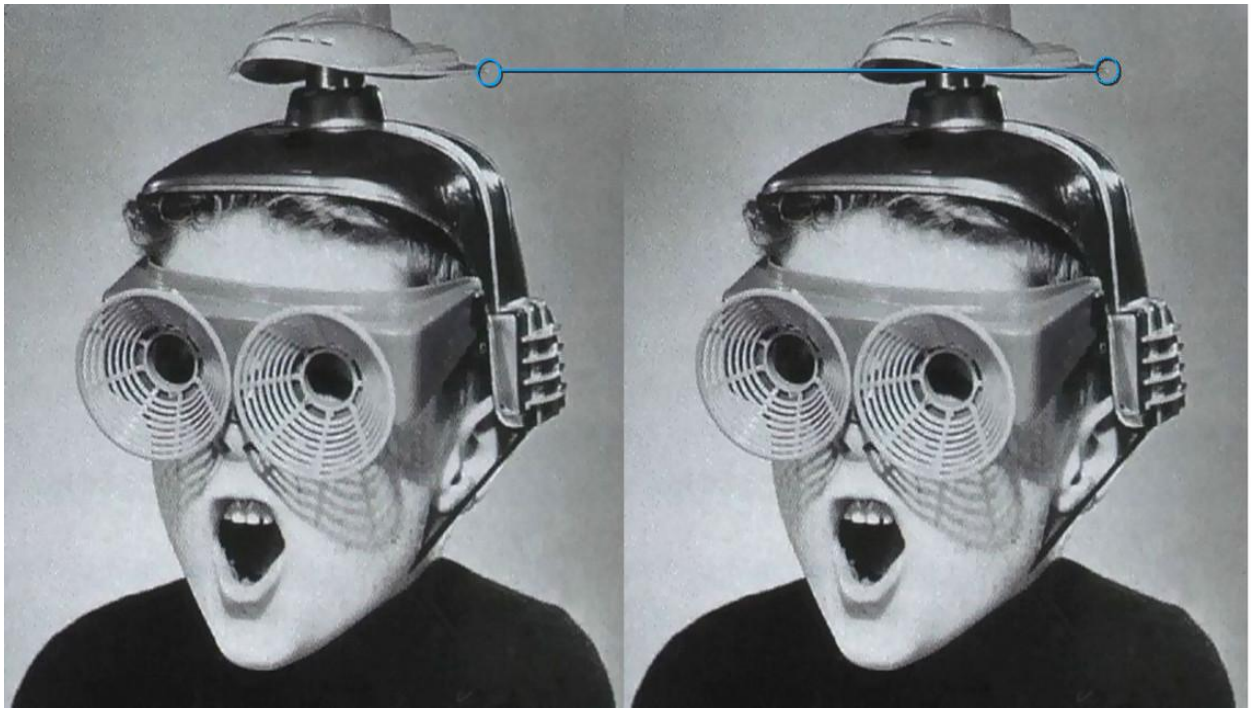
```
            x = (i - CY_DEPTH) * z / FY_DEPTH
            pcd.append([x, y, z])

    pcd_o3d = o3d.geometry.PointCloud()  # create point cloud object
    pcd_o3d.points = o3d.utility.Vector3dVector(pcd)  # set pcd_np as the
point cloud points
    # Visualize:
    o3d.visualization.draw_geometries([pcd_o3d])
```

The following is the generated 3d points. For some reason, the depth map is quite discretized. So in 3D points cloud, it looks like a stack of 2D images where the depth is defined by the depth map.

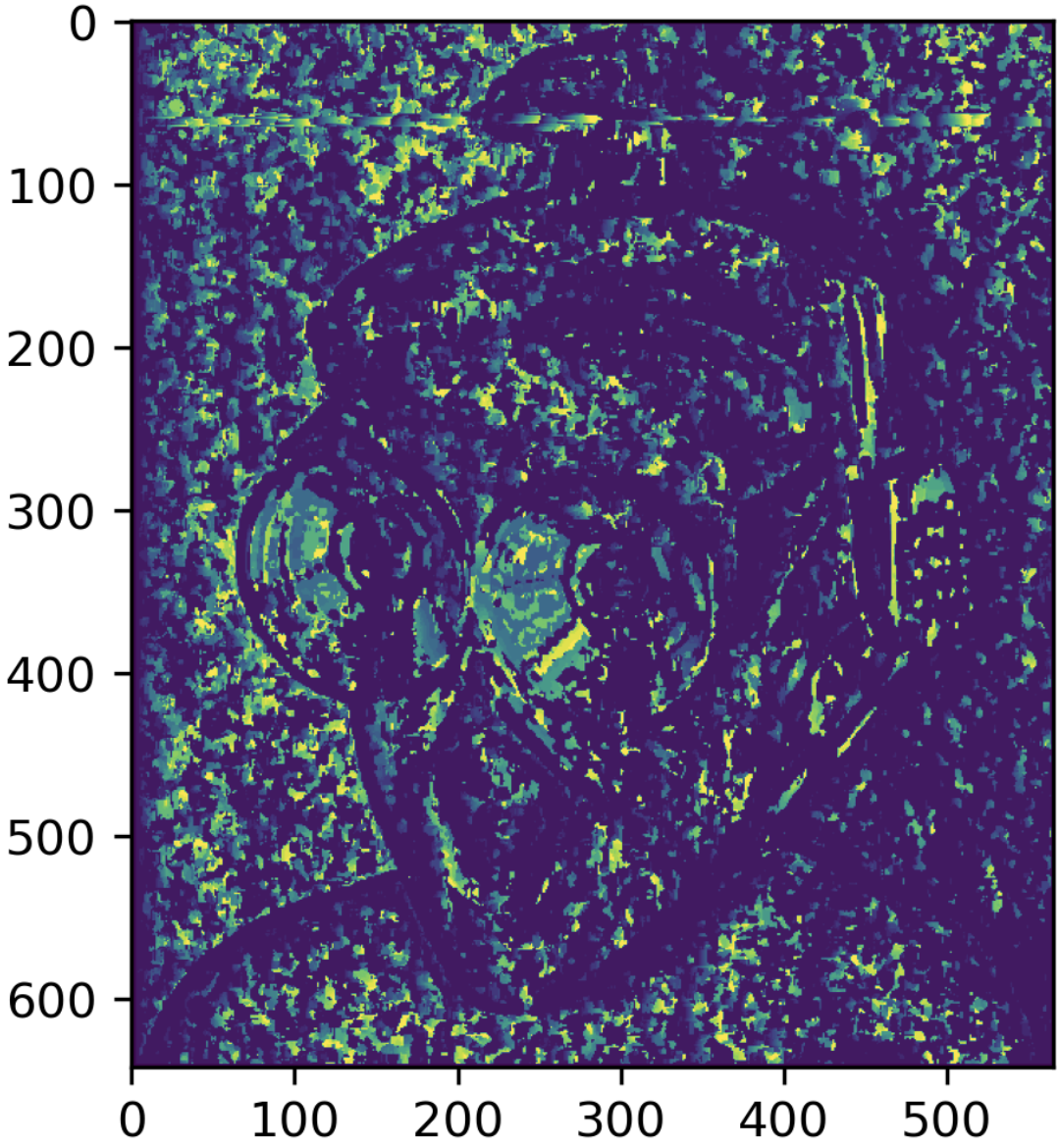


Part 2: additional rectified stereo pairs

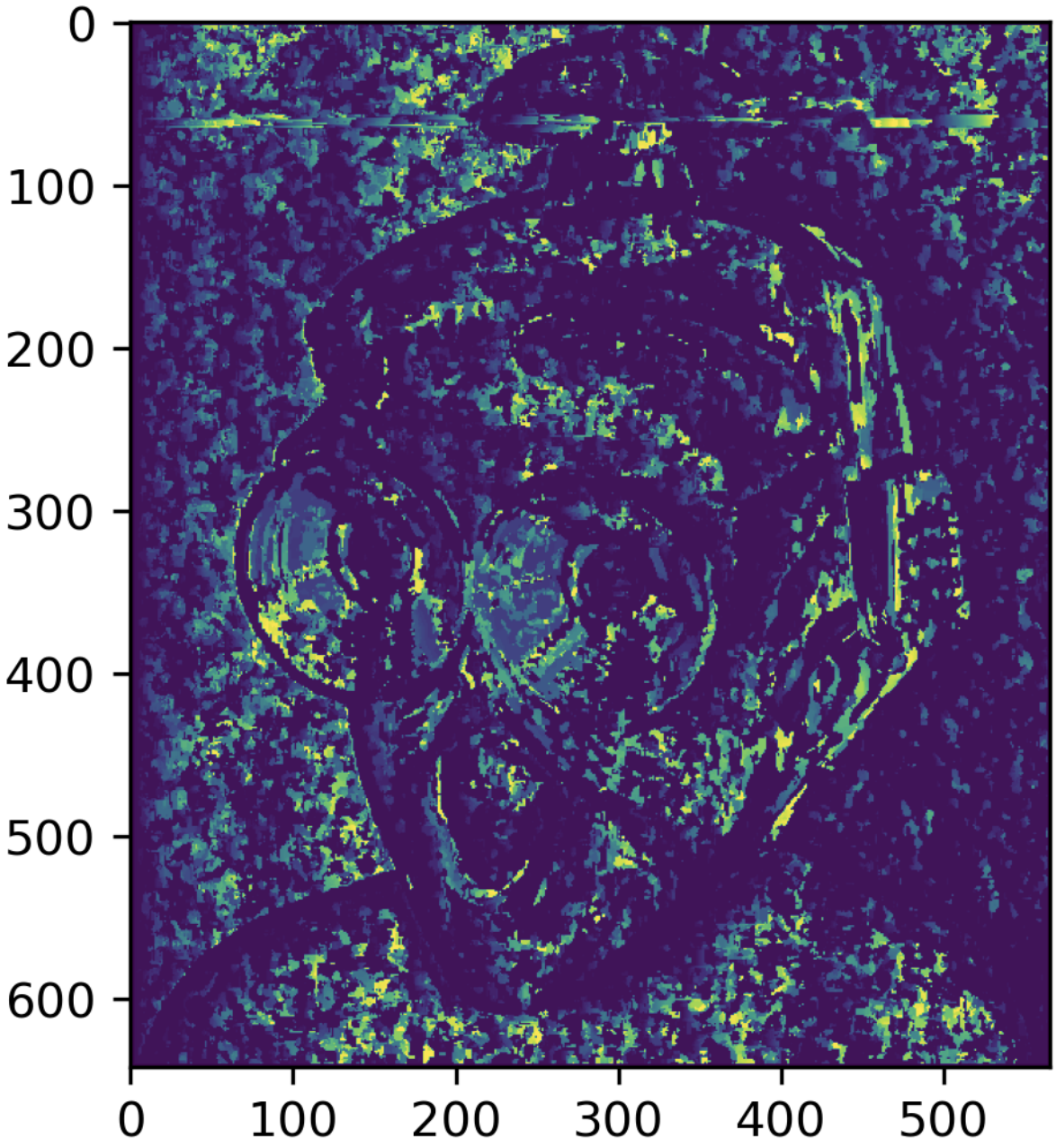
Found the following rectified stereo pairs from this link [Disparity Evaluation using stereo rectified images](#).



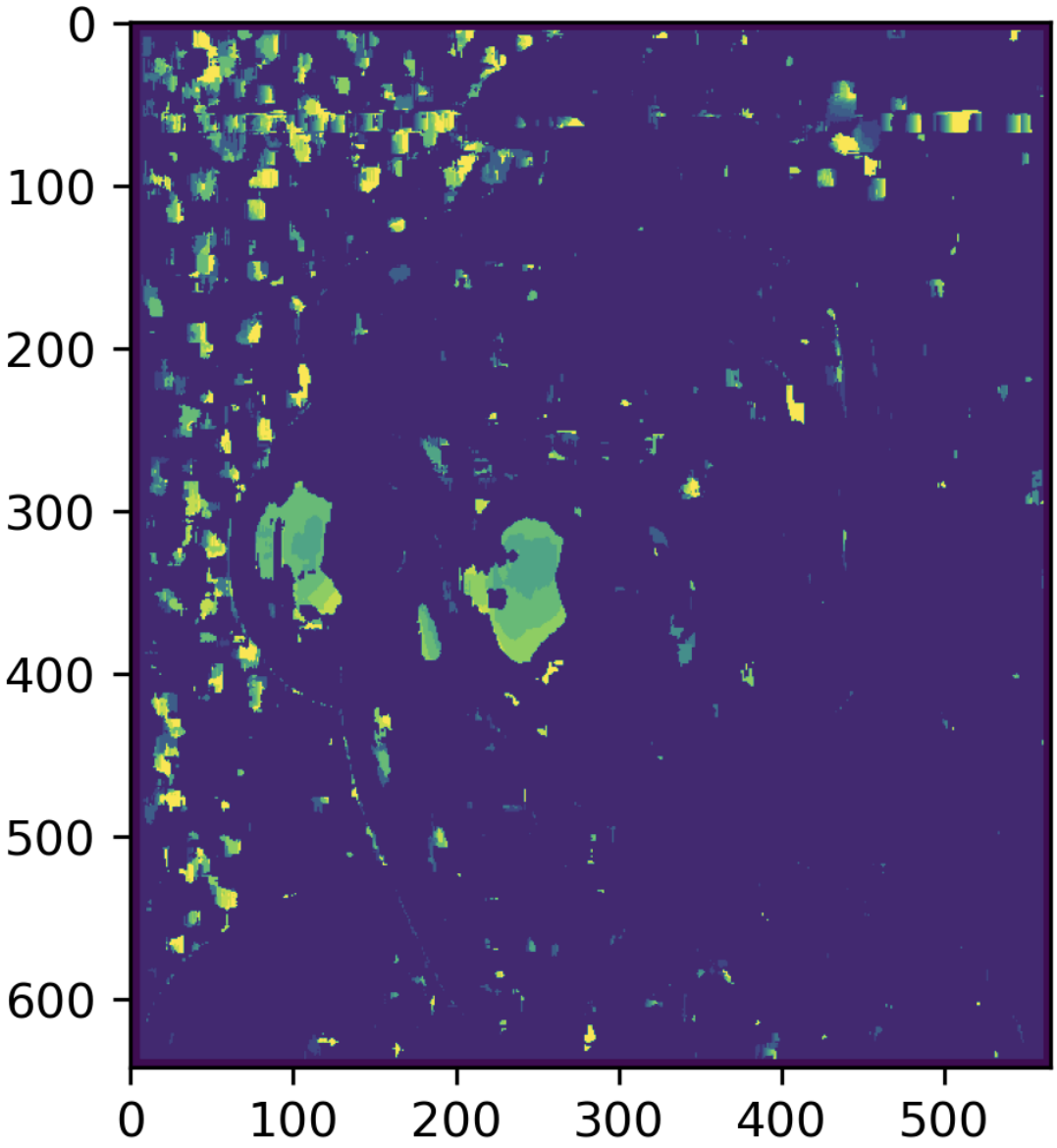
The above pairs produced quite artistic depth maps.



error f ssd size 5 search range 40



ror_f_ssd_size_5_search_range_80



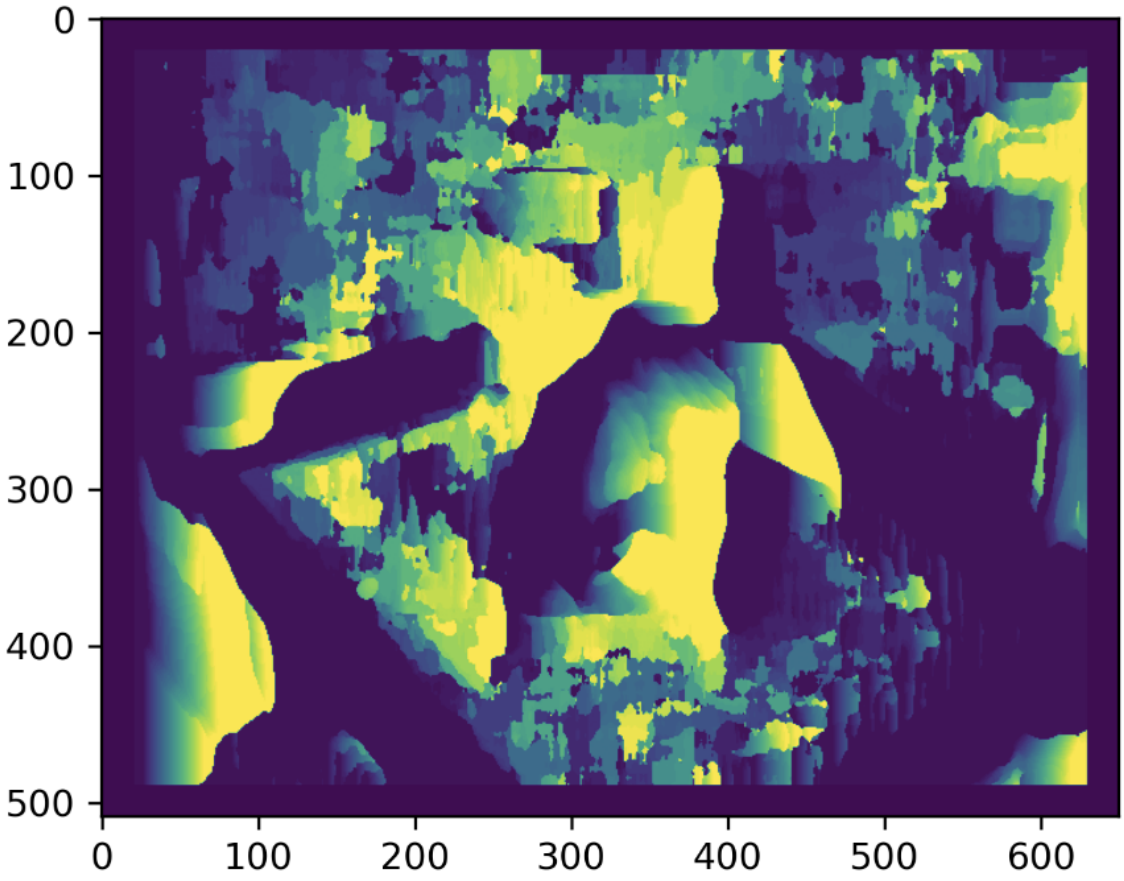
or f ssd size 10 search range 20

I tried another pair of stereo pairs as show before


right

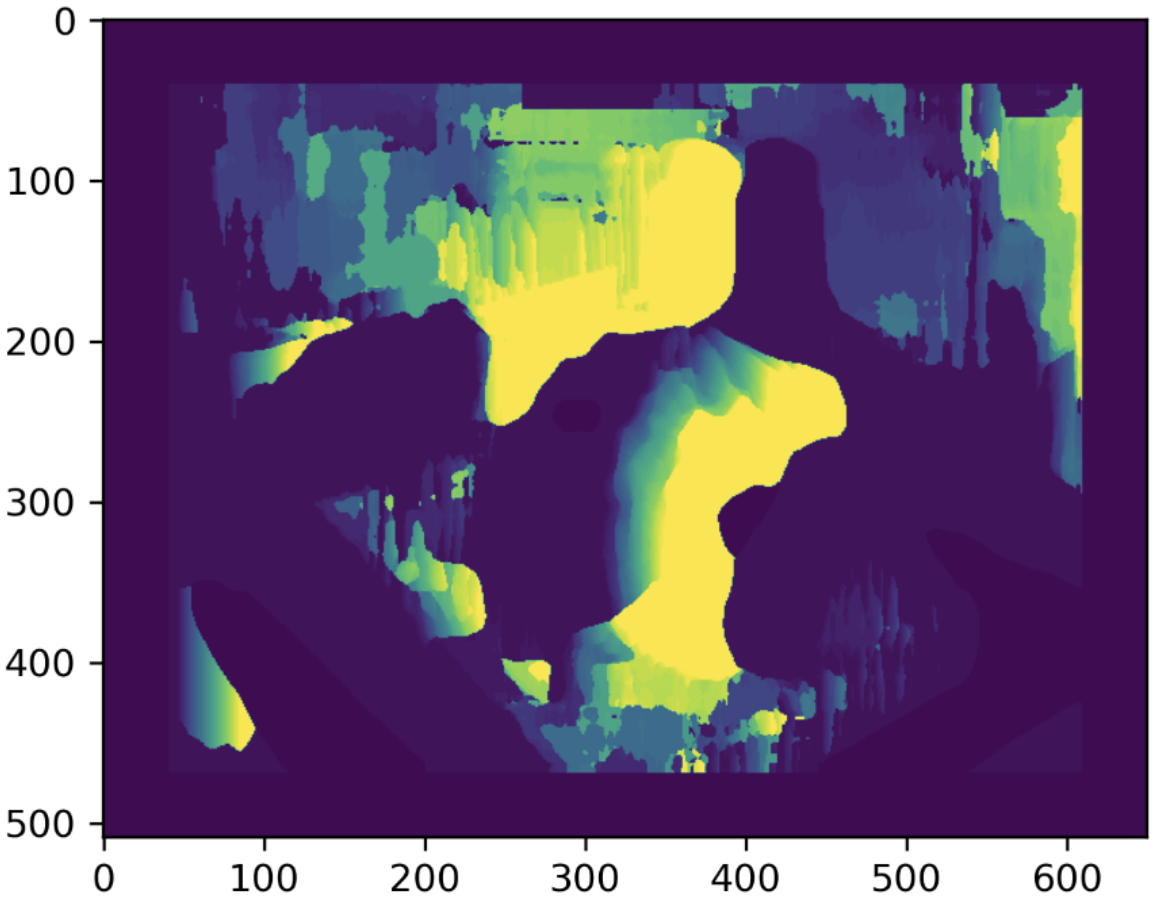
The result is quite hard to comprehend. I suspect the noise in the picture is producing weird artifacts in the final depth map.

When window size is 40, it produces the following result.



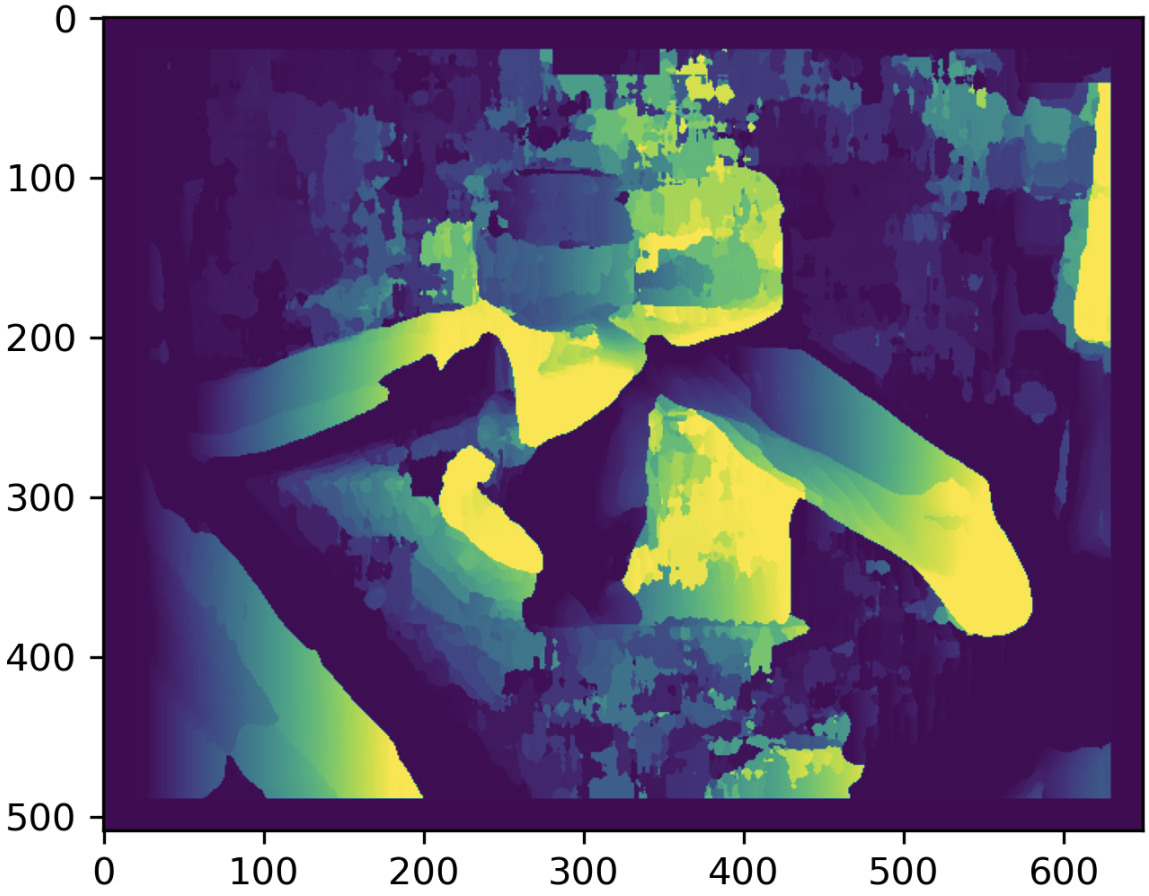
error f ssd size 40 search range 80

When window size is 80, it produces the following.



error f ssd size 80 search range 80

When window size = 160, the shape starts to become more apparent. Where you can see the cap clearly.



I suspect the random noises existing in the stereo images are creating lots of noise in the depth map. One approach might be to filter out the depth value below or above a certain value.