Eric Hsiao

CSE 152A

Final Project Report

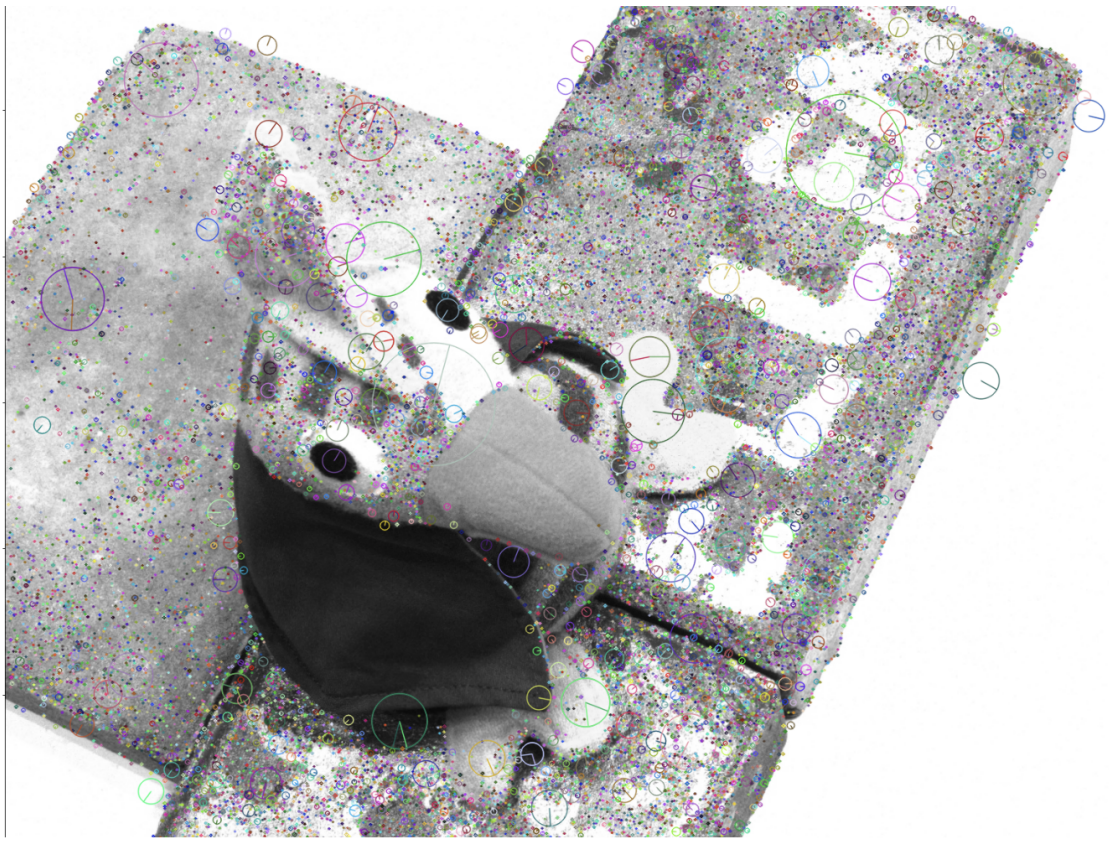
Part 1

In part 1, there wasn't much to do besides getting the helper methods set up and uploading the image files.

Part 2

In order to identify points of interest, I used OpenCV's SIFT detection algorithm. Using a SIFT method, I was able to detect keypoints and compute descriptors which were used to visualize the keypoints on all of the images.

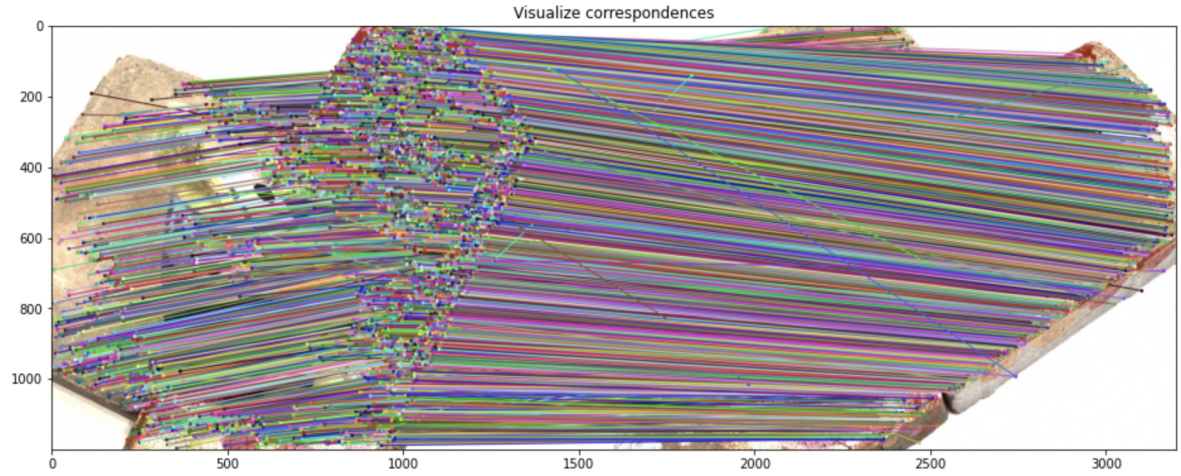
Here is an example image using the first image:



Part 3

In part 3, we needed to make the function that gets correspondences between two images. Using the knnMatch function, I was able to obtain potential matches between the keypoints of two images. In order to obtain enough correspondences without having too many outliers, I tested multiple thresholds for accepting matches.

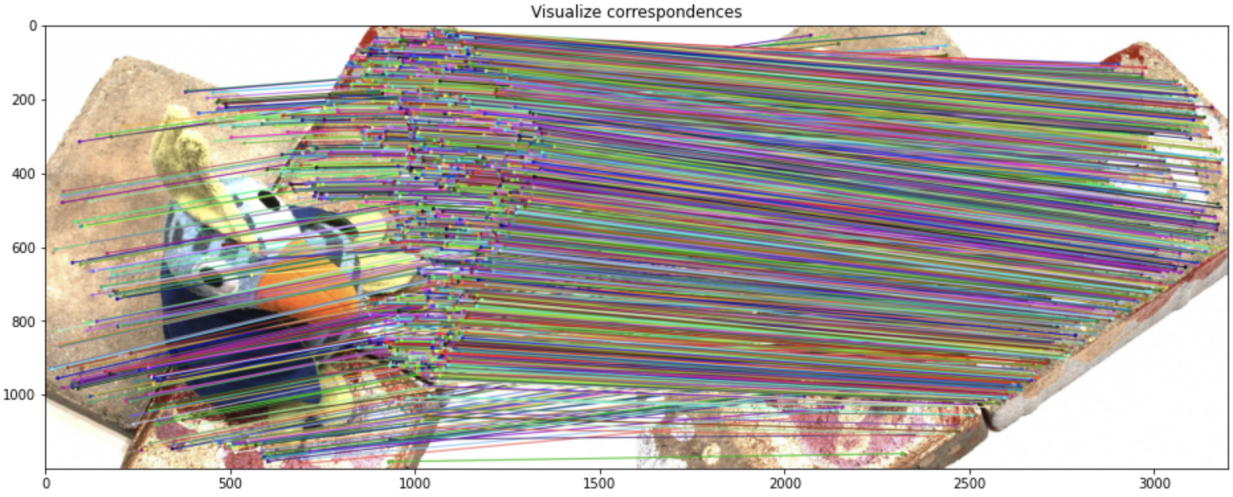
I first used the default value of 0.75:




```
Number of correspondences derived: 4802
```

- There seemed to be a good number of correspondences but there also are a lot of wrong correspondences.

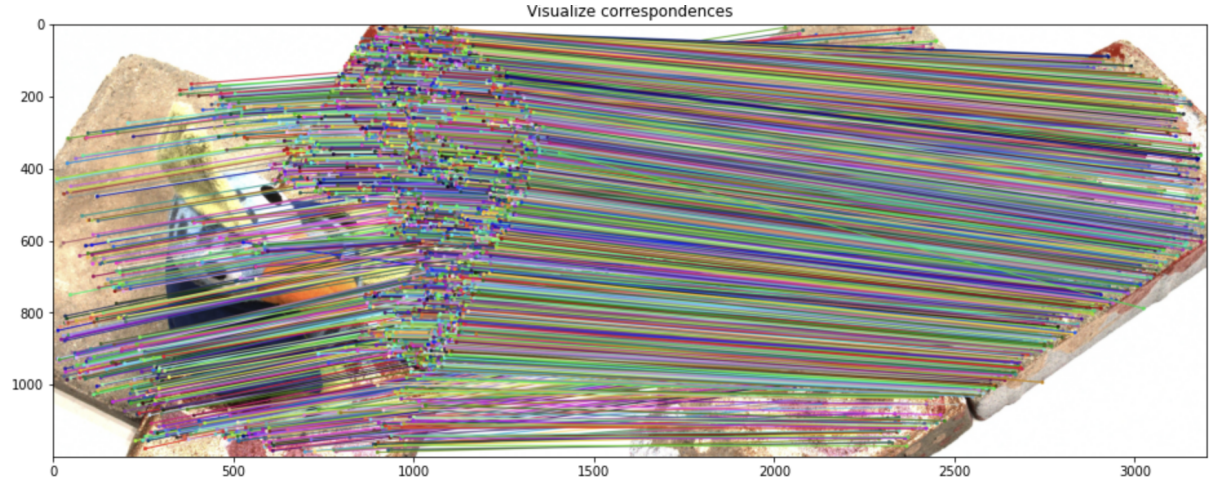
I lowered the threshold to 0.5 and got the following:




```
Number of correspondences derived: 1617
```

- The number of correspondences roughly dropped to ⅓ of what it was for using a threshold of 0.75
- There didn't seem to be any easily noticeable wrong correspondences.

Next, I raised the threshold to 0.6 since I judged that it would obtain more accurate correspondences without adding too many wrong correspondences and got the following:

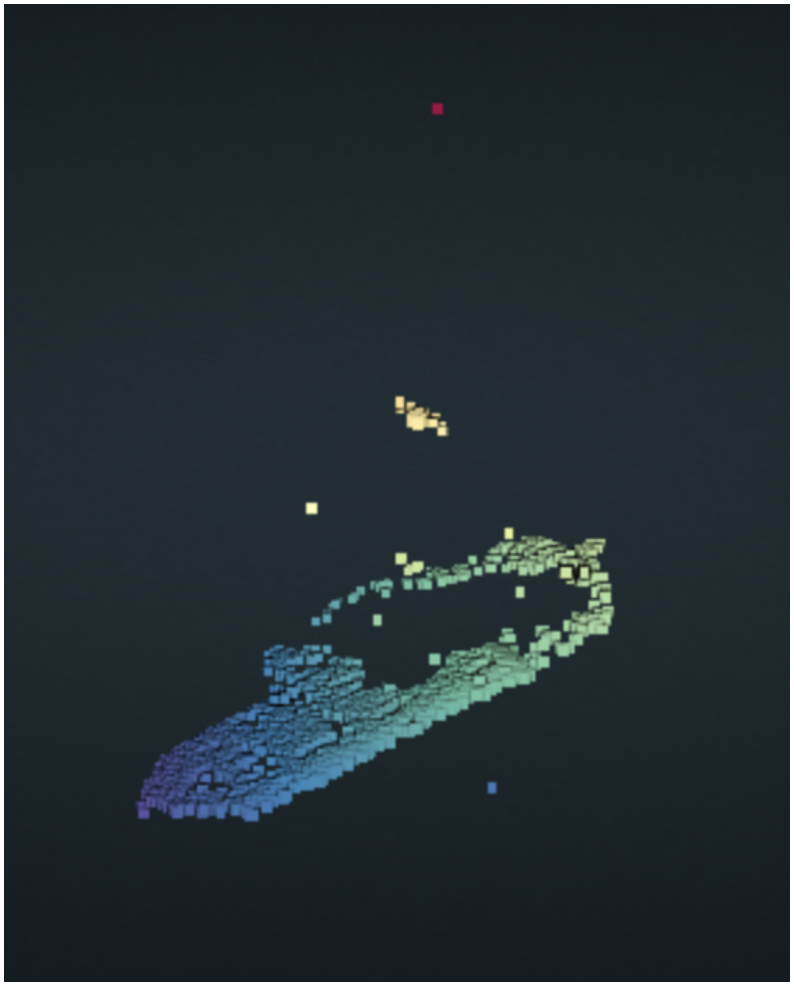



```
Number of correspondences derived: 2843
```

- Using 0.6 got a good number of correspondences without any very noticeable wrong correspondences.

I tested a few more thresholds a little larger than 0.6, but they only got a few more correspondences and introduced a couple wrong correspondences, so I determined that 0.6 was a good enough threshold.

After completing the code for 3.2, I obtained the visualization of the reconstruction for images[0] and images[1] shown below (next page).

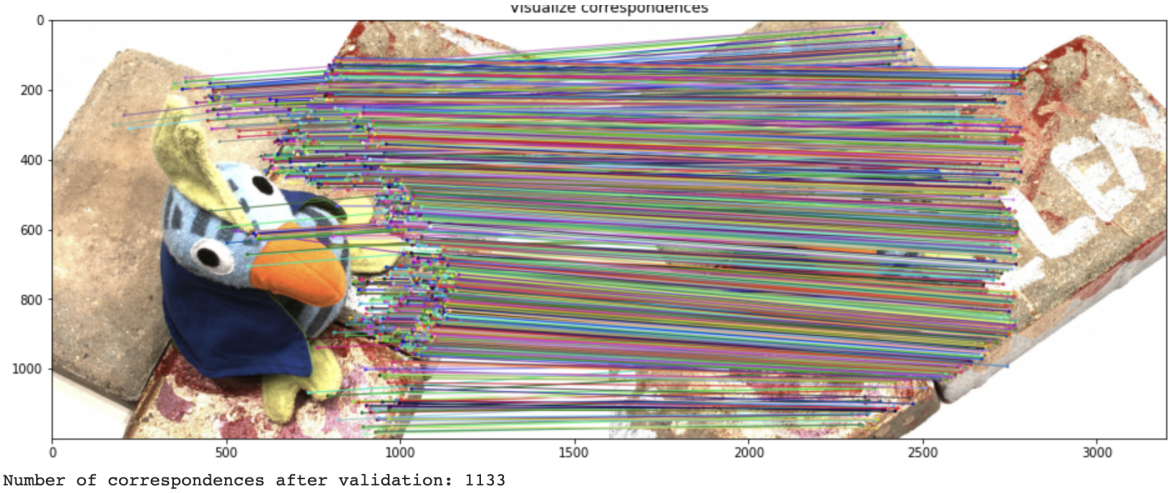


There seemed to be a few outliers, but overall it seemed like a reasonable result. I decided that I would need to move further in the project to perform the reconstruction using all of the images before I could make any more decisions.

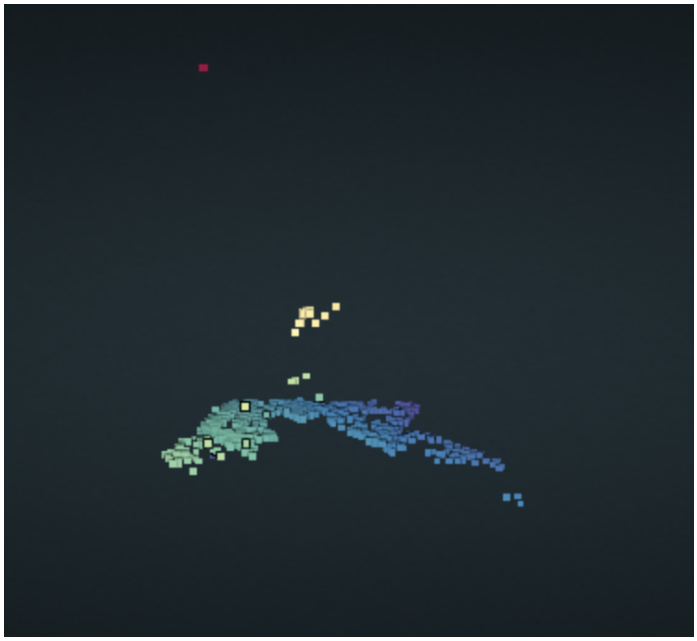
Part 4

In this part, I filled in the code and tested multiple values for the threshold. I saw that a good starting point for this value was one percent of either the height or the width which I determined to be 12 and 16 respectively through printing out the shape of one of the images.

Using 12 as the threshold, I got the image below. It seemed like about 30% of the correspondences remained after invalidating invalid ones.



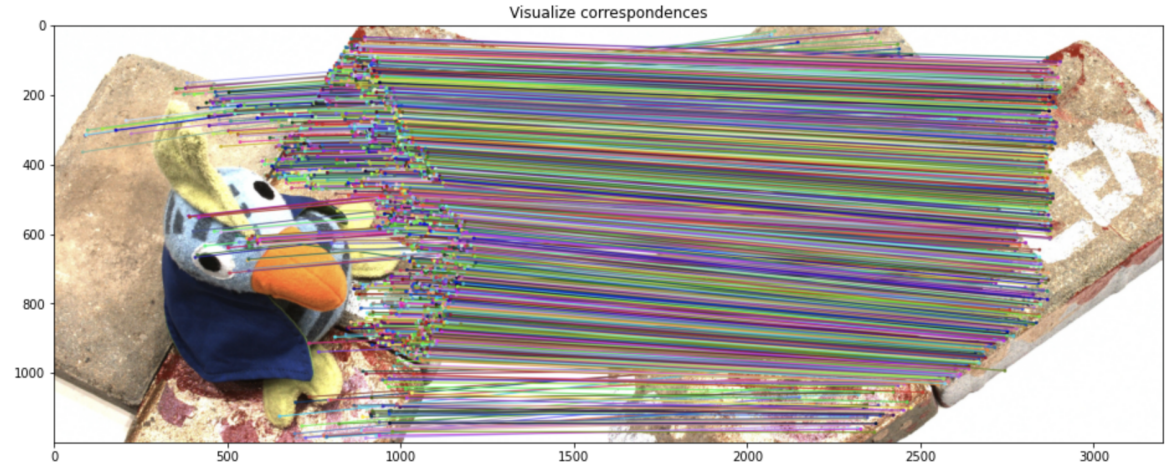
The following visualization was made.



Compared to the previous visualization made before validating the correspondences, it seemed as if most of the points mapping the bird were

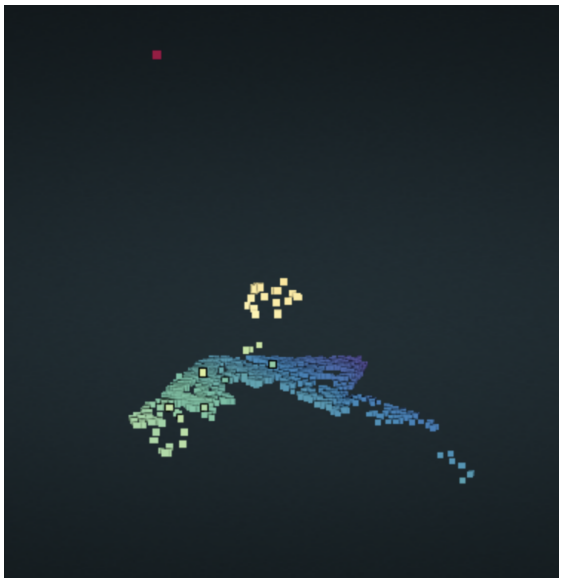
removed. Most of the remaining correspondences were for a section of the objects that the bird is standing on.

Next, I tried raising the threshold to 16 which is 1% of the width of the images.



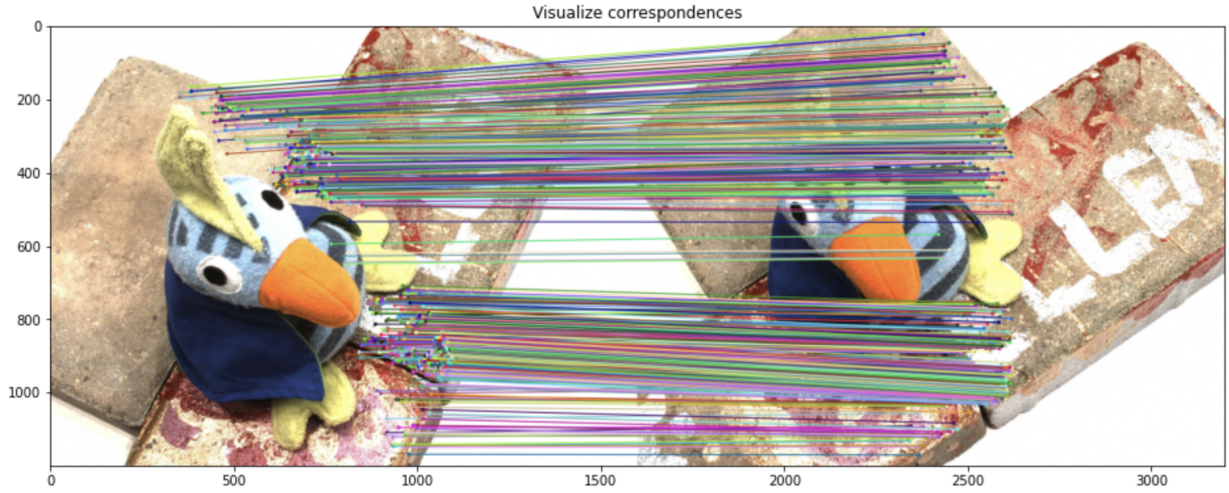

```
Number of correspondences after validation: 1656
```

There were roughly 500 more correspondences than using 12 as the threshold.



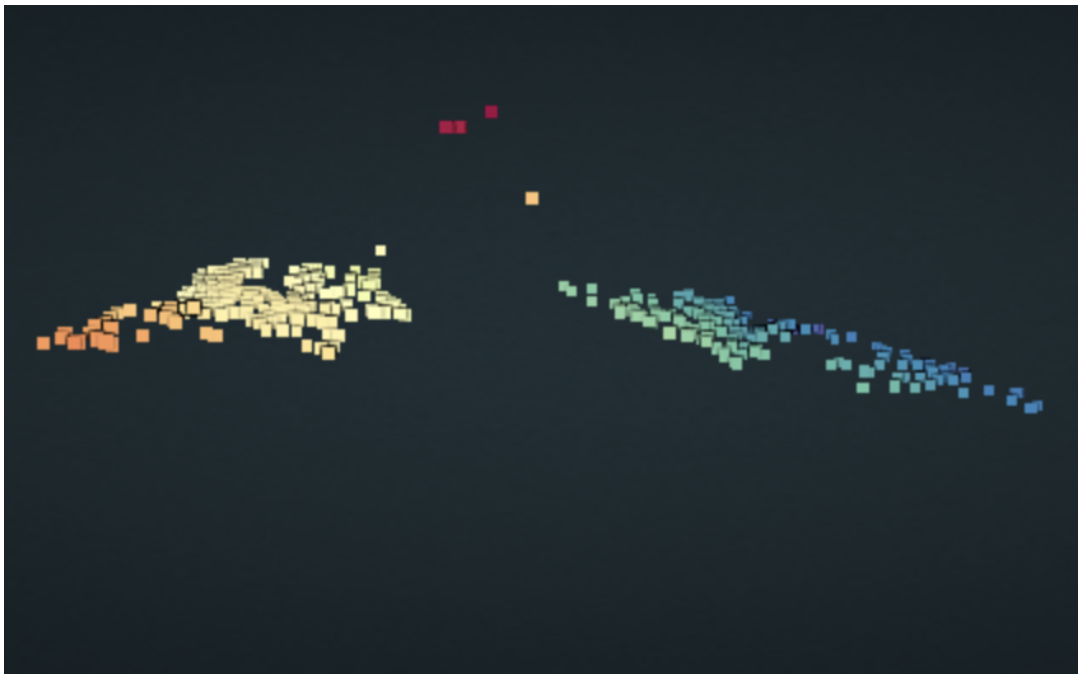
From the visualization, I saw that more points were taken from a different part of the image forming a small outline corresponding to points close to the duck.

Lastly, I tried lowering the threshold to 6 just to see if using an even lower threshold would produce an even sharper outline of the image.



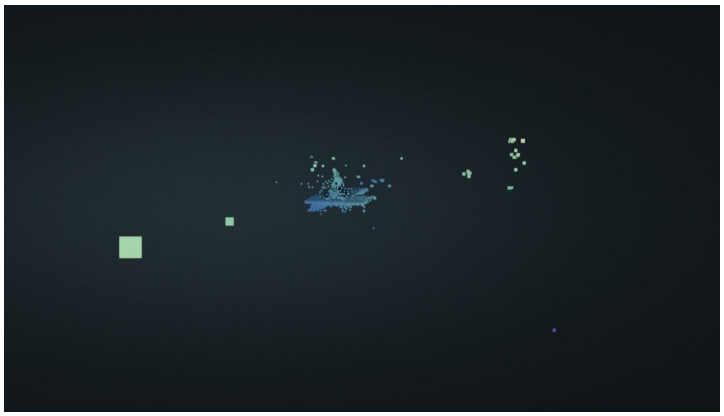
Number of correspondences after validation: 522

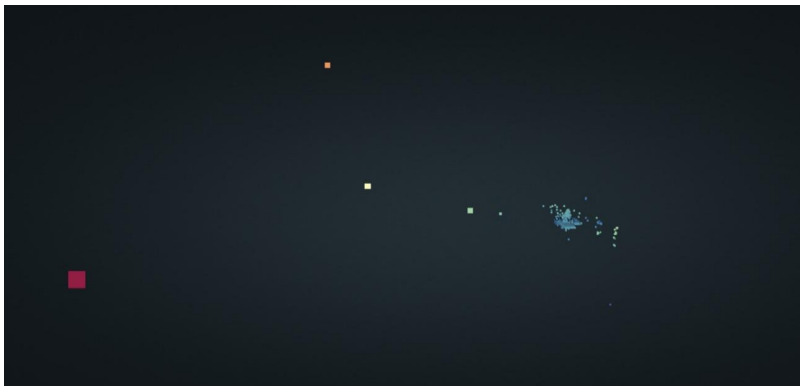
From just visualizing the correspondences, I determined that too many were removed which would make visualizing the entire object more difficult later on.



From the visualization, it is difficult to see the outline that was present using a threshold of 12 and of 16. Based on these results, I decided to temporarily just use 12 as the threshold and see future visualizations before adjusting it again.

In part 4.2, I ran the code to use all of the images and got the following visualization of the object.



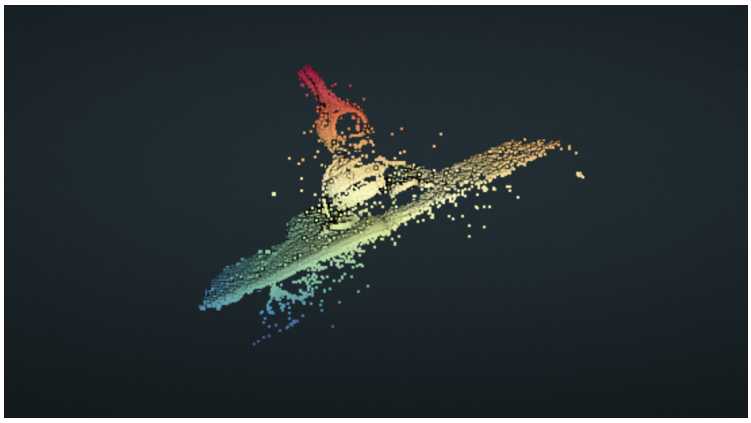


From here, it seemed like I had a decent visualization of the object, but just had some very far outliers. My goal after was to get rid of those outliers.

Part 5

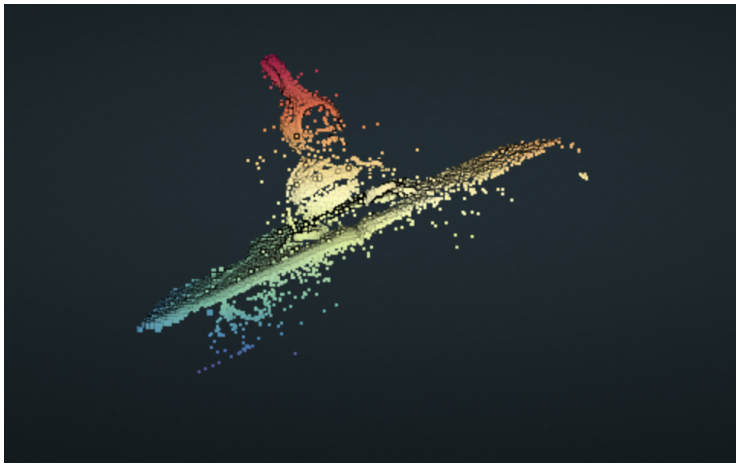
In this part, I just had to optimize the results I got, removing outliers and making the outline of the object sharp. To do this, I followed the guide and used open3d. After performing the setup code, I decided to first test the Statistical outlier removal method of removing outliers.

First, I tried using the default std_ratio of 2.0 and got the following visualization.



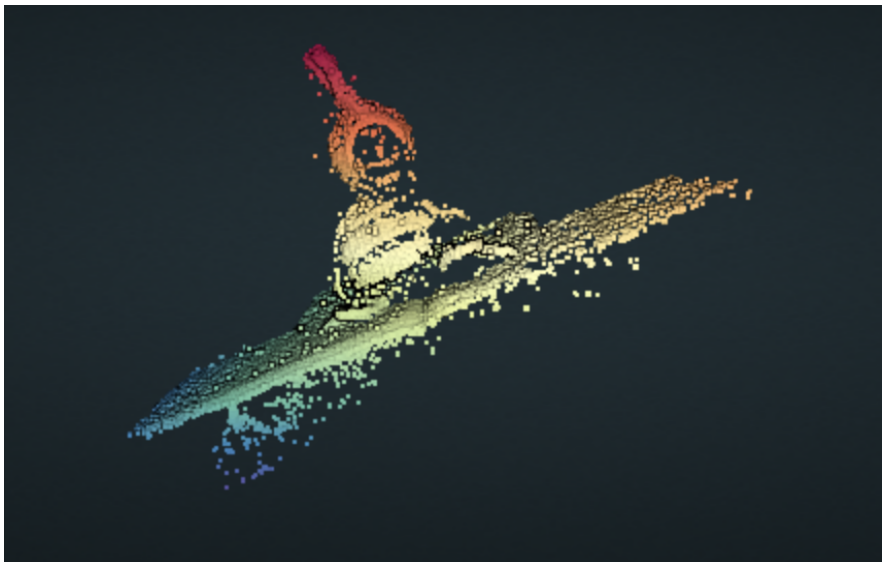
Already, the major outliers were removed and the visualization was pretty good. However, there still were a lot of outlier points that do not belong (the random points forming a small "cloud" around the outline of the duck).

Because the std_ratio of 2.0 did not get rid of all of these outliers, I reduced it to 1.5 and got the following results:



I noticed that a couple of outliers were removed, but overall the visualization and sharpness of the duck did not improve by much.

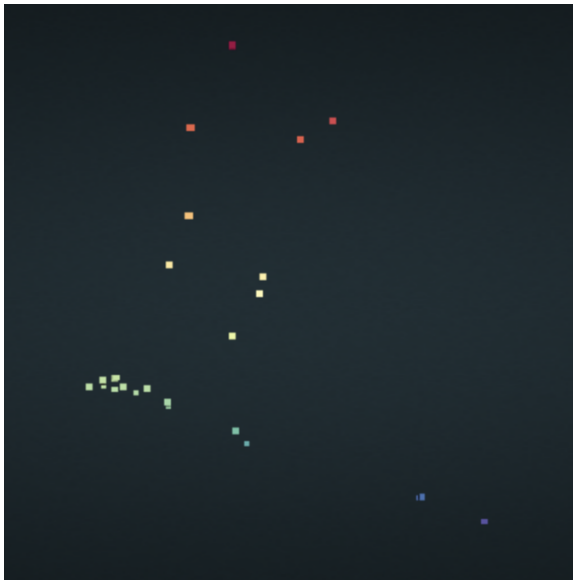
I then lowered the std_ratio to 0.5 and got the following visualization:



I was surprised by how much sharper the outline of the duck became and how many outliers were removed. The visualization was much better. Furthermore, it seemed to almost completely match the reference visualization provided. At this point, I determined that using the Statistical outlier removal method with a std_ratio of 0.5 was a very good way of optimizing the visualization.

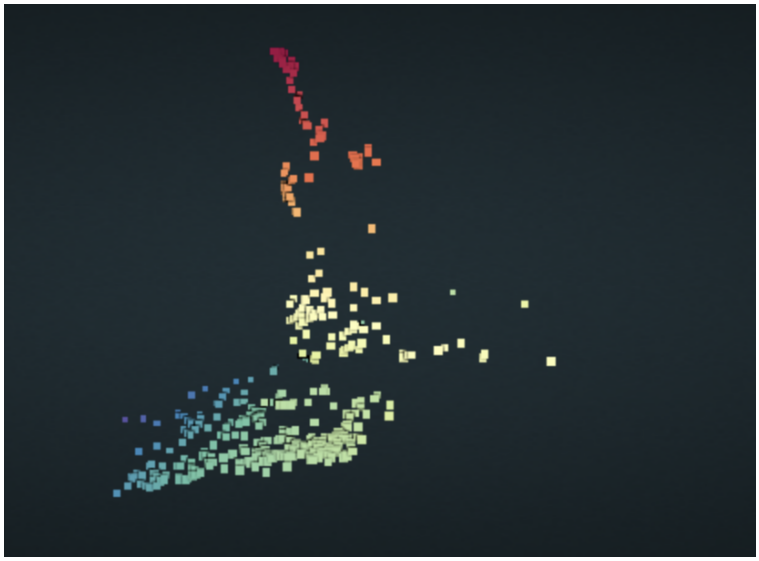
From there, I was pretty satisfied with my results but still wanted to try using the Radius outlier removal method.

When I set nb_points and the radius to their default values (16 and 0.05 respectively), nothing was displayed in the visualization, so I determined that too many points were being removed. I tried increasing the radius higher and higher until points were being displayed, but there were so little so I decided to start decreasing the nb_points. With nb_points of 4 and the same radius, I still wasn't getting an output. With nb_points of 2 and the same radius, I finally got a couple of points in the visualization.



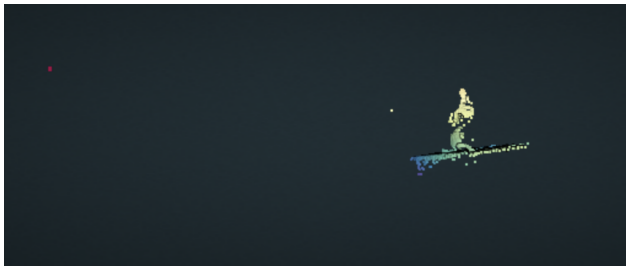
This clearly wasn't accurate, but at least I still got points.

I then tried nb_points = 1 and got the following:



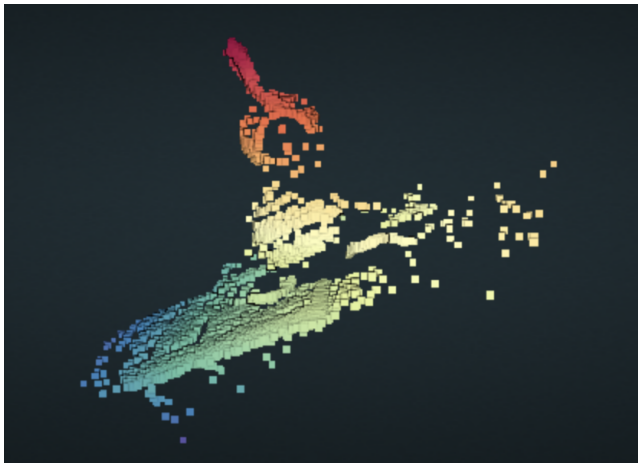
I finally got something that looked remotely similar to what I expected to get.

I wasn't getting enough points still, so I decided to increase the radius size.
Using nb_points = 1 and radius = 0.5, I got the following:



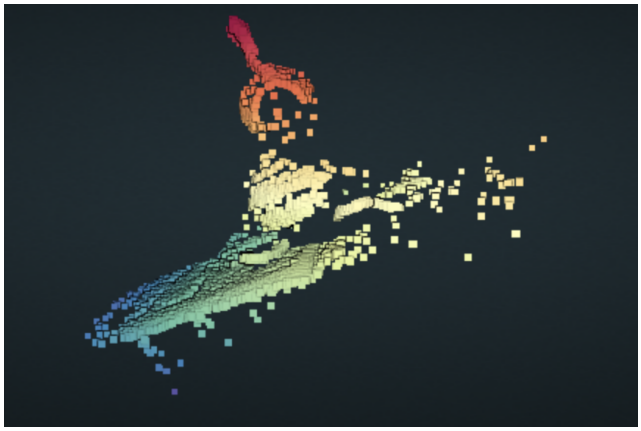
This let me know that I increased the radius size too much.

Then I tried a radius = 0.2 and got:



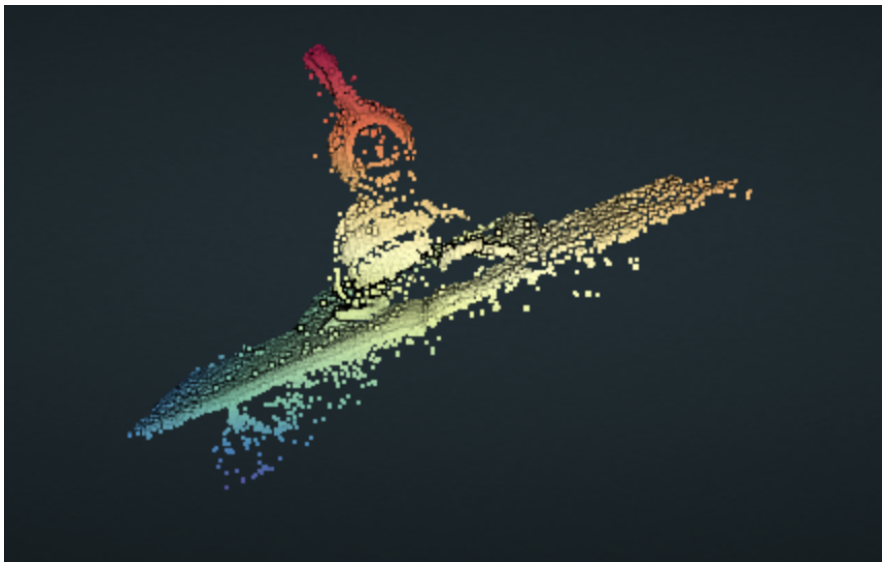
It seemed pretty good, but I was missing some points on the right side.

I kept on trying slightly larger values of radius until I got the far outliers again. Using a radius of 0.213, I got close to the most number of points without a large outlier.



However, this result is still much worse than what I got using the other method.

After optimizing with both the Statistical outlier removal method and the Radius outlier removal method, I obtained the most accurate results using the Statistical outlier removal method with nb_neighbors = 20 and std_ratio = 0.5. These parameters helped me get the following visualization.



Since I was able to get a great visualization with the previous settings in previous parts, I decided to stick with them and finalize with the current visualization I obtained.

That concludes my report of my final project.