AER 1515: Perception for Robotics

Homework 2

By: Alexis Bruneau: 1008704270

Presented to Dr.Steven Waslander

November 9th, 2022

Content

Table of Figures

Section 1 Feature Point Detection and Correspondences

Question 1.1 Feature Detection

Using a detector and a descriptor of your choice, detect 1000 keypoints and compute their descriptors in every image pair in both the training set and the test set. Present the left image with keypoints for each image pair in the test set. Briefly discuss any observations you make as to the appearance of unreliable keypoints (ones that may lead to difficulty in performing matching later in the assignment).

A function called question1_1 was written for this question. This function detects 1000 keypoints and their descriptors for each image pairs. The following images illustrates the left images in the test set and their keypoints. It is to note that two enumerators were used from DrawMatchesFlags to facilitate making observations. The default one and DRAW_Rich_KEYPOINTS were used [1]. The orb detector was chosen for this question.

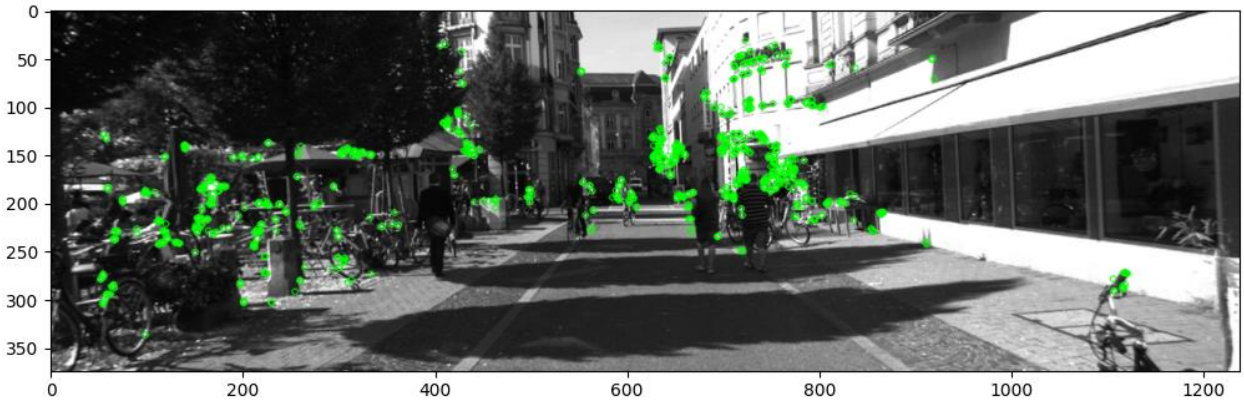

Figure 1 Left Image 1 Test Set - 1000 Default Keypoints

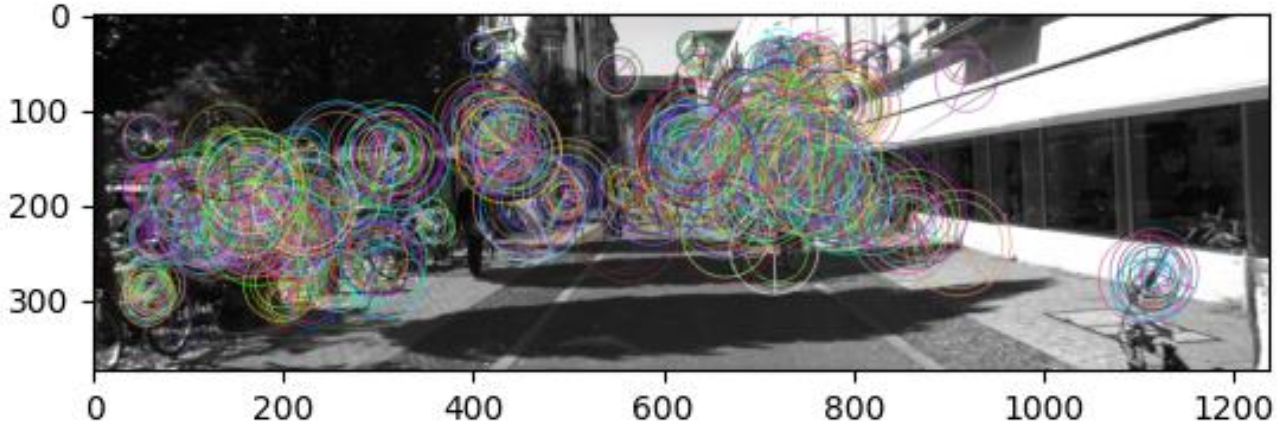

Figure 2 Left Image 1 Test Set - 1000 Rich Keypoints

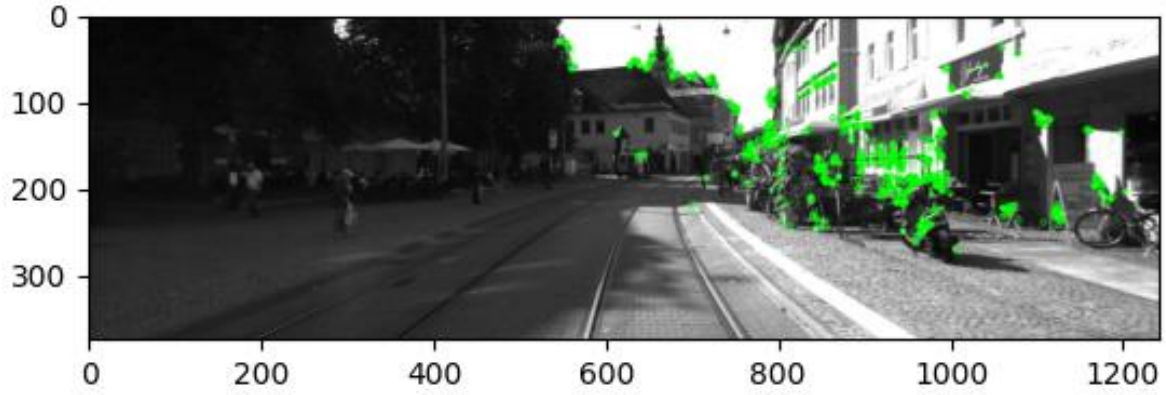

Figure 3 Left Image 2 Test Set – 1000 Default Keypoints

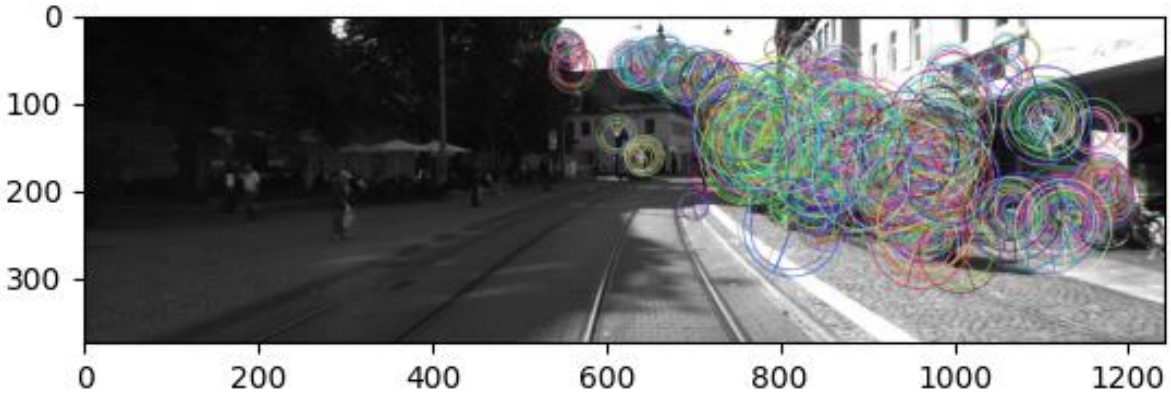

Figure 4 Left Image 2 Test Set - 1000 Rich Keypoints

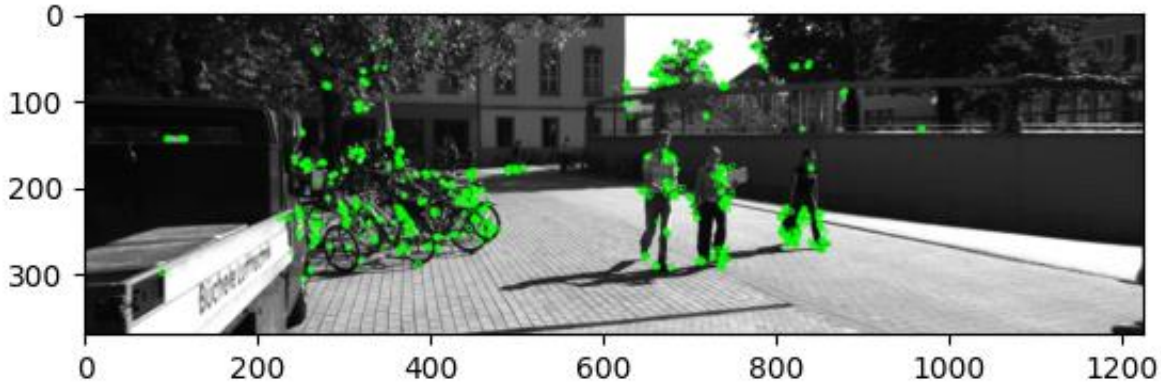

Figure 5 Left Image 3 Test Set - 1000 Default keypoints

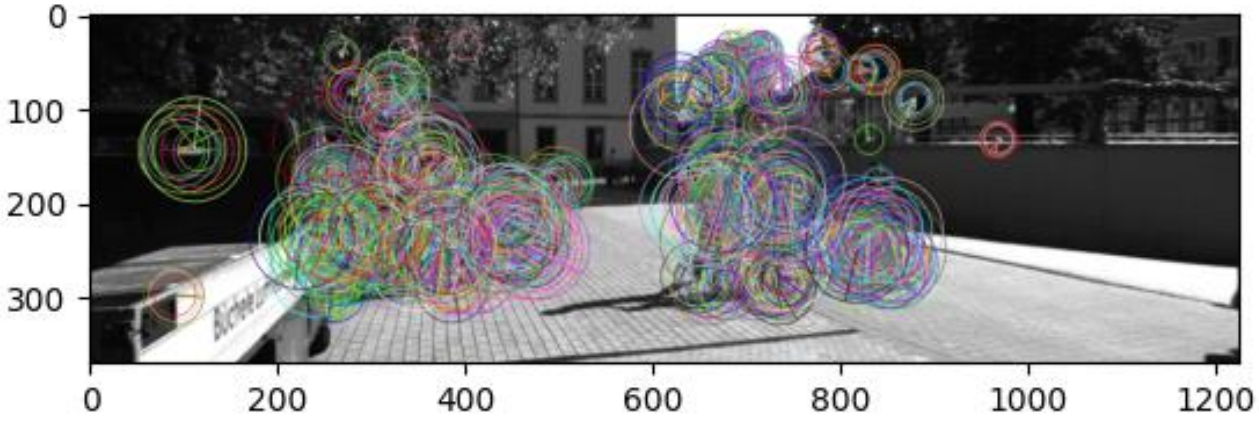

Figure 6 Left Image 3 Test Set - 1000 Rich Keypoints

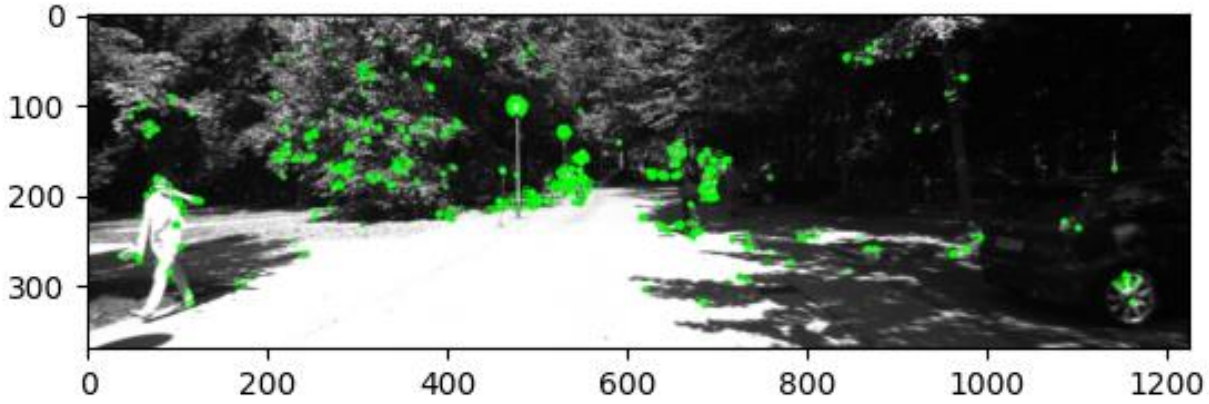

Figure 7 Left Image 4 Test Set – 1000 Default keypoints

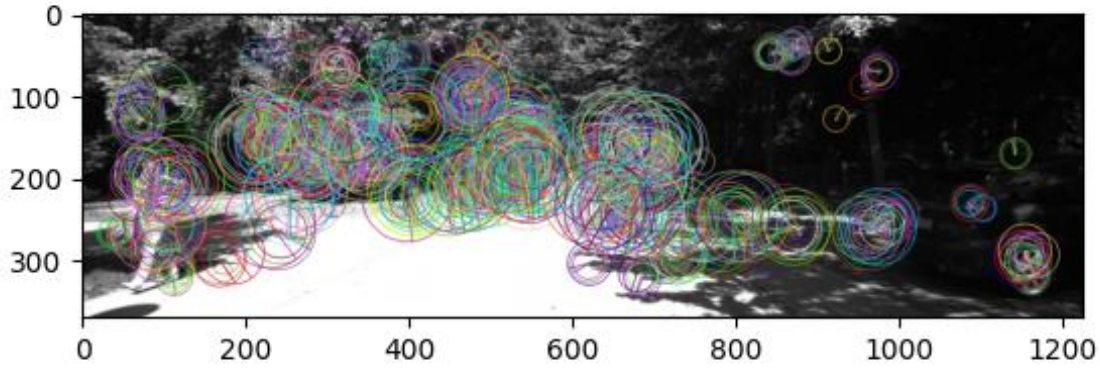

Figure 8 Left Image 4 Test Set - 1000 Rich Keypoints

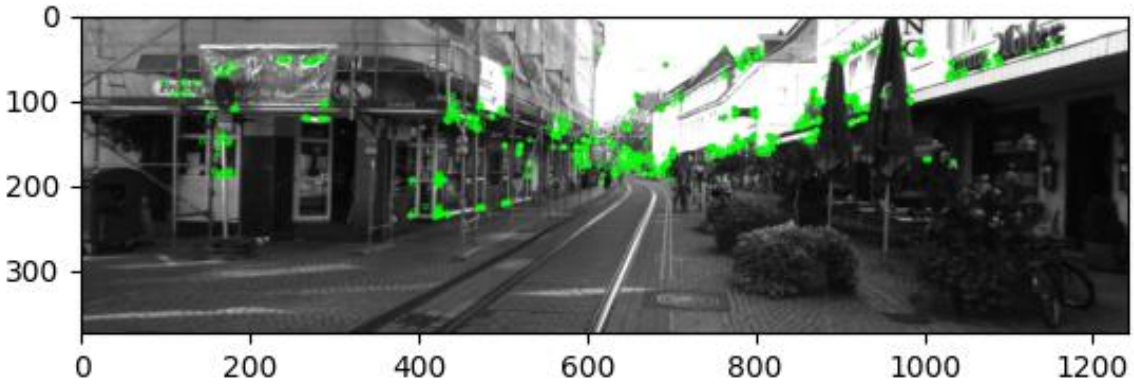

Figure 9 Left Image 5 Test Set - 1000 Default keypoints

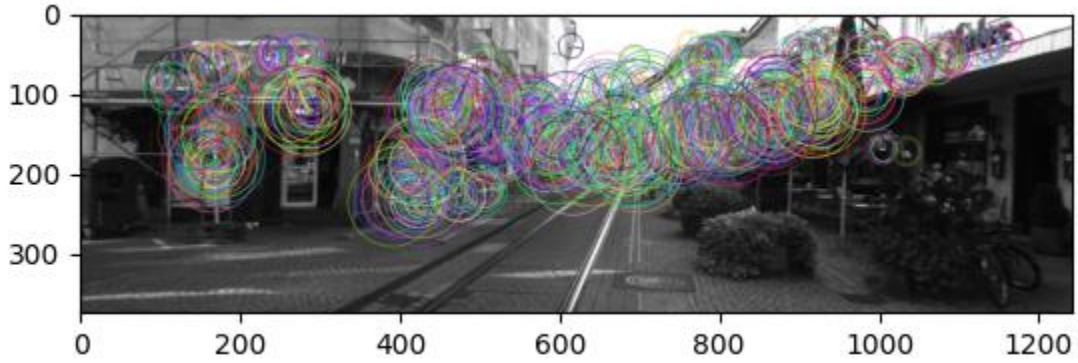

Figure 10 Left Image 5 Test Set - 1000 Rich Keypoints

Keypoints detection consists of locating key object parts. Those keypoints helps underlying object in a feature-rich manner. The detector is the method to compute image information and decide whether an area is an image feature or not. There are various types of feature detectors like edge, corner, and points. To detect these features, feature point detectors like SIFT, SURF, ORB, FAST, BRIEF [2]. Usually, the interest point we want to detect match the following requirements [3]:

- Rich image content (brightness variation, color variation) withing the local window
- Well-defined representation (signature) for matching/comparing with other points
- Well-defined position in the image
- Should be invariant to image rotation and scaling

The following observations were made by looking at the figures presented above. Theirs not a lot of keypoints that appears in areas that are darker. This is probably because it does not meet the first bullet point in the usual criteria for interest point. For example, most keypoints in Figure 3 and Figure 4 are located on the right side, and for both images, the left side is in the shade. This pattern can be observed for each figure presented. Also, for some pictures, the density of keypoints (keypoints / area. For example, in Figure 6, most keypoints are located on the two bicycle and the people. This might lead for harder matching in further questions.

Question 1.2 Feature Matching

Use the keypoints and their descriptors to find their correspondences between image pairs. Describe how you matched these features and present your feature matches for the images in the test set like the sample below. Given that the image pairs are rectified, your feature matches should follow an epipolar constraint (i.e., matches lie on a horizontal line). To evaluate your results, calculate disparity based on keypoint matches, then find the depth for the detected keypoints using camera calibration information. The ground truth depth map for the training set is given for you to evaluate your estimated depths. Note that the ground truth depth map is generated using LiDAR-based depth completion, so it does not cover the entire image. Pixels without a corresponding depth are labelled with a depth of 0. The ground truth depth map is with respect to the left camera frame. Note: the starter code includes functions to load the calibration for the stereo cameras. These functions 1 return a class object with multiple camera calibration matrices, but you will only need to use the calibration information for the left camera (p2) and the right camera (p3).

In this question, features were matched between the left and right images. The process to match features between pair of images is the following. The first step was to get 1000 keypoints for each image. In this case, ORB detector was used.

Once the keypoints and descriptor are saved for the left and right image, the steps to match each keypoint starts. The technique of brute-force matching was used. This technique consists of comparing two sets of keypoint descriptors and generates a result that is a list of matches [4]. Each keypoint descriptor in the first set, gets compared to every keypoint descriptor in the second set (exhaustion of all possible combinations). The result of each comparison is a distance value, and the smallest distance is considered the best match.

There are different ways to calculate the distance between both sets. In this case, since ORB was used, the hamming distance (recommend method for ORB). This method is used instead of Euclidean distance, norm_l1 is the following. The ORB descriptors are vectors of binary values. Hence, applying Euclidian distance between binary distance would give either 1 or 0. Instead, Norm_Hamming is performing Boolean operation on the vectors to compare the descriptors efficiently [5].

For this question, since we know that the image pair is rectified, an epipolar constraint was placed. To apply the constraint, a mask was placed so when comparing the best possible matches between keypoints, it was done on the same horizontal line. It can be seen from the figures underneath that the matches between keypoints are all horizontal, which is the expected results.

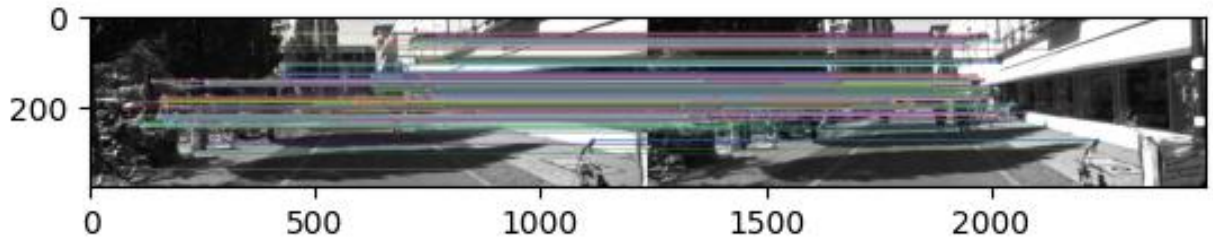

Figure 11 Feature Matching (Epipolar Constraint)- Test Image 1

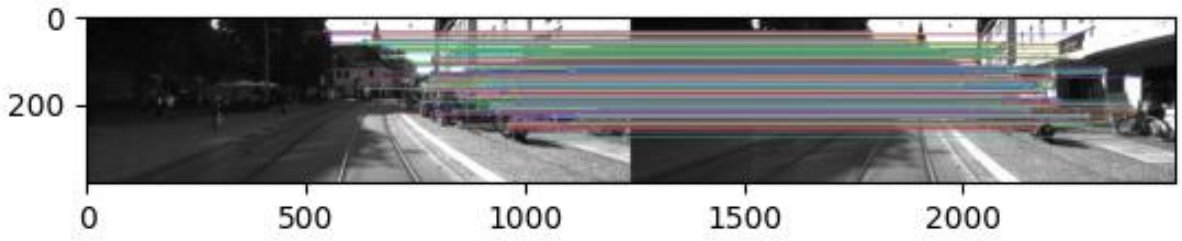

Figure 12 Feature Matching (Epipolar Constraint) - Test Image 2

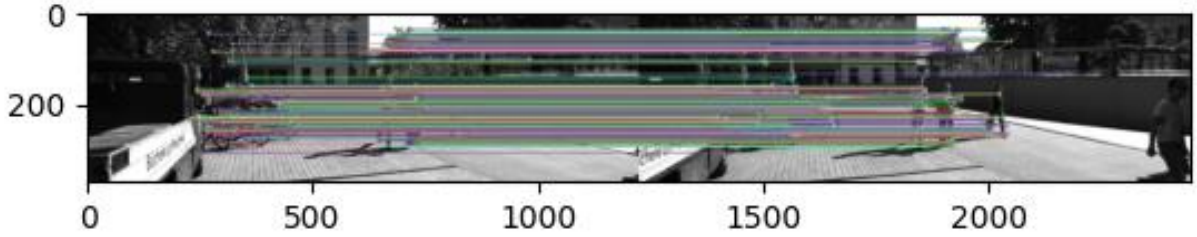

Figure 13 Feature Matching (Epipolar Constraint) - Test Image 3

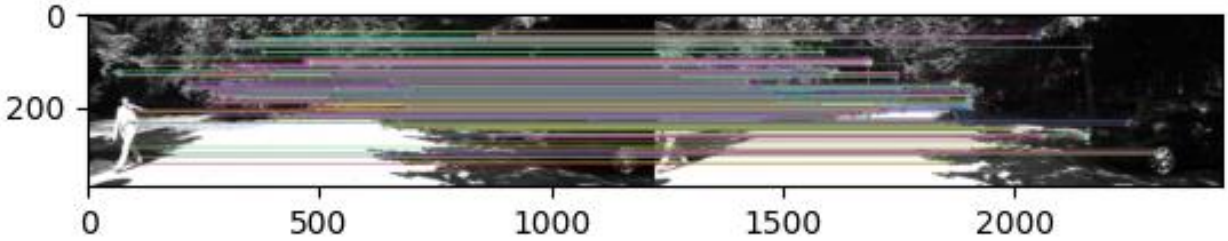

Figure 14 Feature Matching (Epipolar Constraint) - Test Image 4

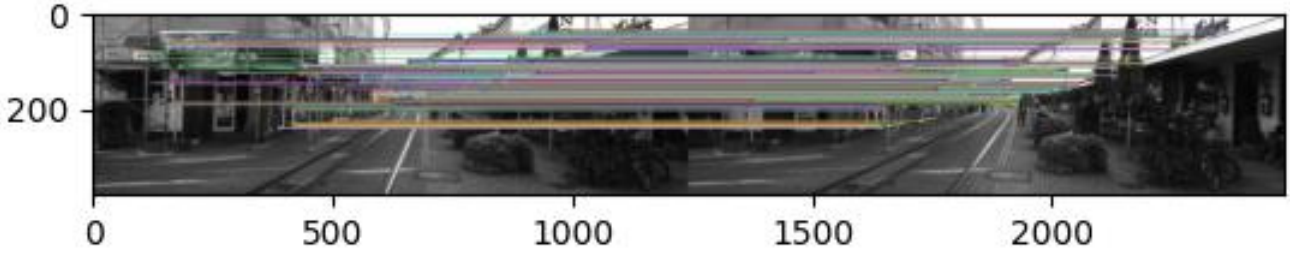

Figure 15 Feature Matching (Epipolar Constraint) - Test Image 5

1.3 Outlier explanation

For question 1.3, the epipolar constraint was removed, but different algorithms were used to remove outliers. It is also to note that Sift was used instead of ORB. More documentation for outliers' algorithms using SIFT were found, explaining the choice of switch. The following algorithms were used in this order for outlier rejection: Lowe's ratio test, Ransac algorithm, and maximum allowed depth distance.

The Lowe's ratio test consists of the following steps. First, it computes the distance between feature f_i in the first image (left image in our case), and all the features f_j in image two (right image for us). Afterwards, it finds the feature that had the minimum distance between the left and right image and stores it f_c. Then, it proceeds to find the second closest distance to feature f_i and stores it as f_s. Once calculated, the distance between f_c and f_s is calculated and stored as the distance ratio. Finally, the matches that respect the following equation are kept.

$$distance\ ratio < distance\ ratio\ threshold$$

Usually, a distance ratio around 0.5 is chosen. This would mean that the best match is two times better than the second-best match [6].

After the Lowe's ratio test, the Ransac algorithm was used. The algorithm can be explained as follows; it selects randomly a minimum number of points required to determine the model parameters. Using this number, it solves for the model. Then, it determines how many points fits within a predefined tolerance ϵ. It then calculates the number of inliers over the total number in the set. If that fraction exceeds a threshold, it re-estimates the model parameters using all the identified inliers and terminate. Otherwise, the steps are repeated for a maximum of N time [7].

Finally, after those two algorithms were applied, the depth for each match was calculated. If the depth exceeded 80 meters, those points were removed since long-range depth predictions become very inaccurate as mentioned in the assignment.

Combining each of those algorithms removed a big portion of the matches. On average, around 150 matches were left per images. From the figure underneath, most of the outliers were removed from Lowe's ratio test which removed on average 830 matches. Afterwards, the maximum depth removed the second the greatest number of matches averaging a removal of around 16 matches and finally, the Ransacks method removed an average of around 11 matches. It is important to remember the order at which those algorithms were perform which is the major reason as to why Lowe's ratio test removed so much more outliers.

```
After filtering, we went from 1000 to 123 matches. Lowes removed 848 matches, Ransac removed 4 matches, and the maximum depth removed 25
After filtering, we went from 1000 to 173 matches. Lowes removed 775 matches, Ransac removed 20 matches, and the maximum depth removed 32
After filtering, we went from 1001 to 129 matches. Lowes removed 865 matches, Ransac removed 7 matches, and the maximum depth removed 0
After filtering, we went from 1000 to 228 matches. Lowes removed 771 matches, Ransac removed 1 matches, and the maximum depth removed 0
After filtering, we went from 1000 to 97 matches. Lowes removed 860 matches, Ransac removed 21 matches, and the maximum depth removed 22
```

Figure 16 Outlier removal results

The figures underneath illustrate the drawn matches after the removal of outliers.

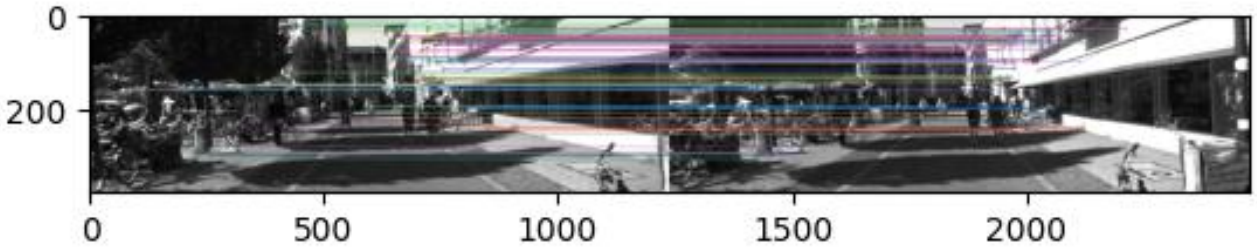

Figure 17 Feature Matching (Removed Outliers) - Test Image 1

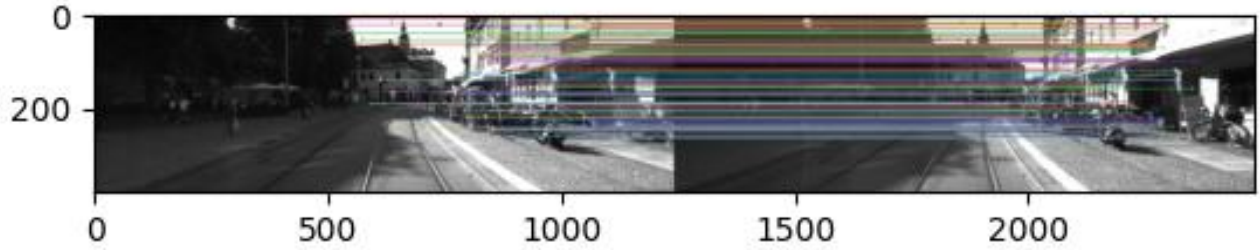

Figure 18 Feature Matching (Removed Outliers) - Test Image 2

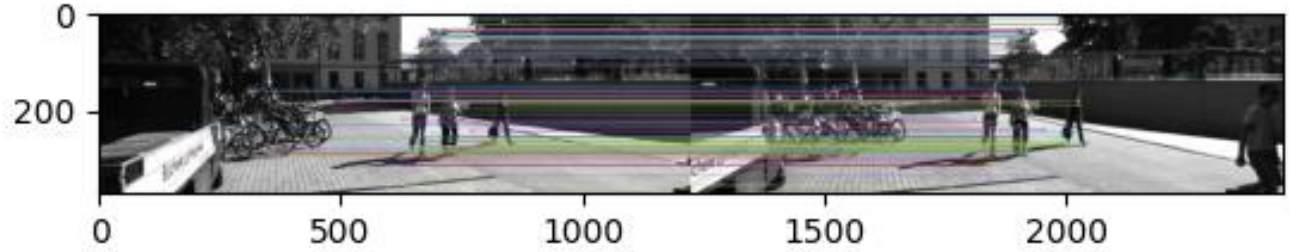

Figure 19 Feature Matching (Removed Outliers) - Test Image 3

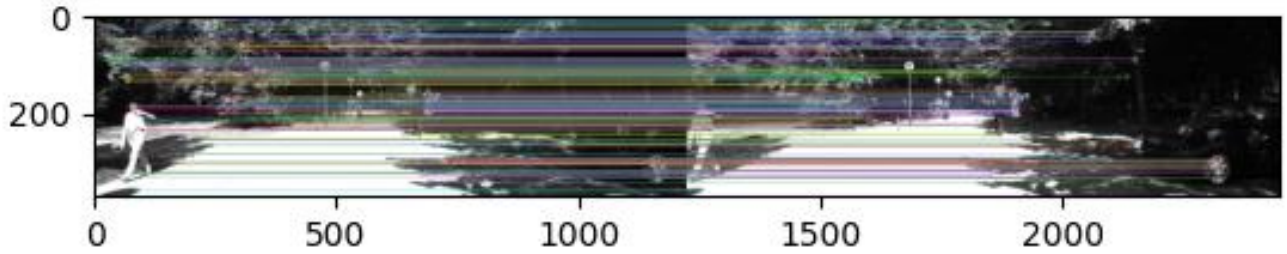

Figure 20 Feature Matching (Removed Outliers) - Test Image 4

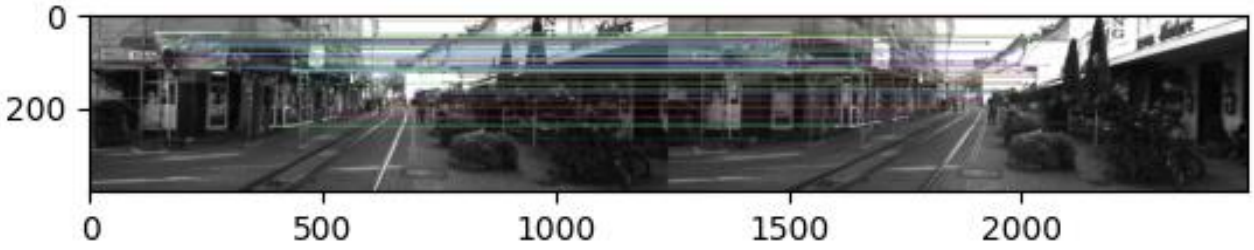

Figure 21 Feature Matching (Removed Outliers) - Test Image 5

It is to note that when the depth of the ground truth was calculated, cv.IMREAD_GRAYSCALE and cv.IMREAD_ANYDEPTH was used to calculate the depth. The only difference with the second option was that the depth needed to be divided by 255. The figure underneath illustrates the result between both methods. The first method was kept calculating the depth of the ground truth image.

```
The depth when using ANYDEPTH and dividing by 255 is : 0.0
The depth using Grayscale technique is: 0
The depth when using ANYDEPTH and dividing by 255 is : 0.0
The depth using Grayscale technique is: 7
The depth when using ANYDEPTH and dividing by 255 is : 7.047058823529412
The depth using Grayscale technique is: 16
The depth when using ANYDEPTH and dividing by 255 is : 16.29019607843137
The depth using Grayscale technique is: 0
The depth when using ANYDEPTH and dividing by 255 is : 0.0
The depth using Grayscale technique is: 0
The depth when using ANYDEPTH and dividing by 255 is : 0.0
The depth using Grayscale technique is: 0
The depth when using ANYDEPTH and dividing by 255 is : 0.0
The depth using Grayscale technique is: 15
The depth when using ANYDEPTH and dividing by 255 is : 15.709803921568627
```

Figure 22 GRAYSCALE vs ANYDEPTH depth comparison

The RMSE was calculated for question 1.2 when only the epipolar constraint was applied. The average RMSE across the 10 training images was 2120. The RMSE for question 1.3 when using the outlier remover algorithm (Lowe's ratio test, Ransac, max distance) had a much better accuracy. The RMSE was 232, around 10 time smaller. It was expected that the RMSE in 1.3 would be lower since we removed the outliers.

Question 2 3D Point Cloud Registration

2.1 Nearest Neighbour Search

For question 2.1, the brute-force search method was used to find the best matches between the source point cloud P to target point Q). The algorithm consists of looking at one row and keeping it fix (source point cloud P) and comparing all the possible target point cloud Q. For each possibility, it calculates the Euclidean distance between the points and the algorithm returns the match with the minimum distance.

This algorithm repeats this process for each source points. One the corresponding points are found; the mean Euclidean distance of all matches is calculated. The mean Euclidean distance was calculated after each ICP iteration for the bunny, dragon, and armadillo dataset. The results are presented underneath.

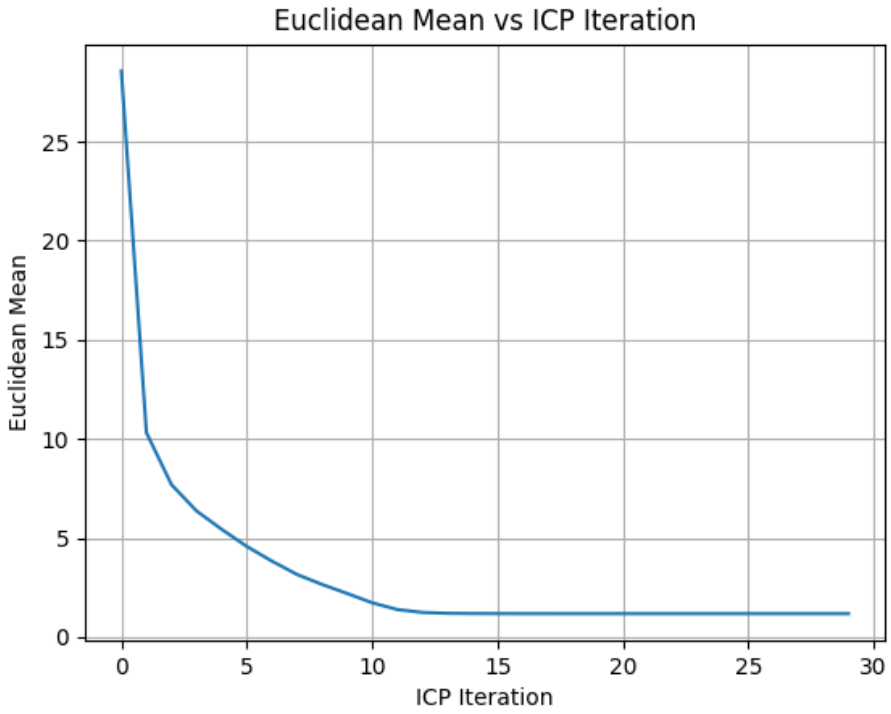

Figure 23 Mean Euclidean Distance vs ICP Iteration (Armadillo Dataset)

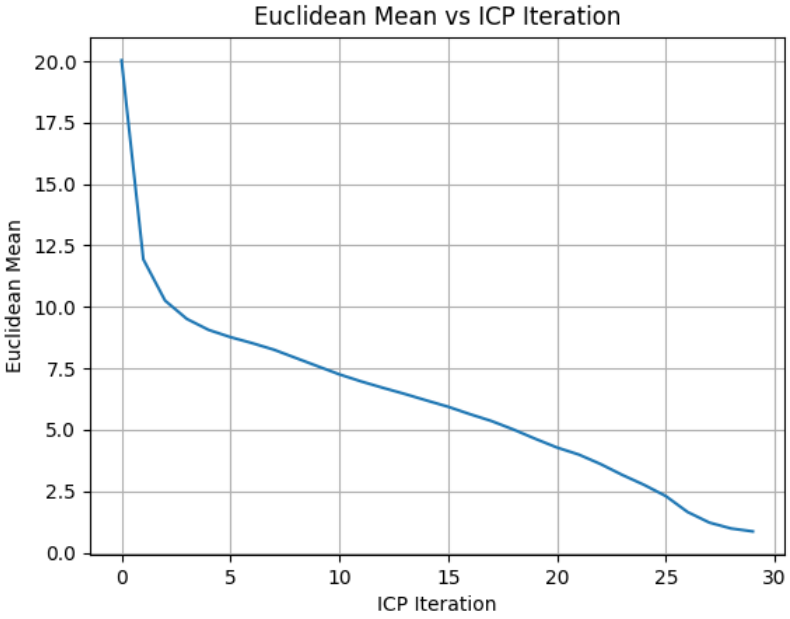

Figure 24 Mean Euclidean Distance vs ICP Iteration (Dragon)

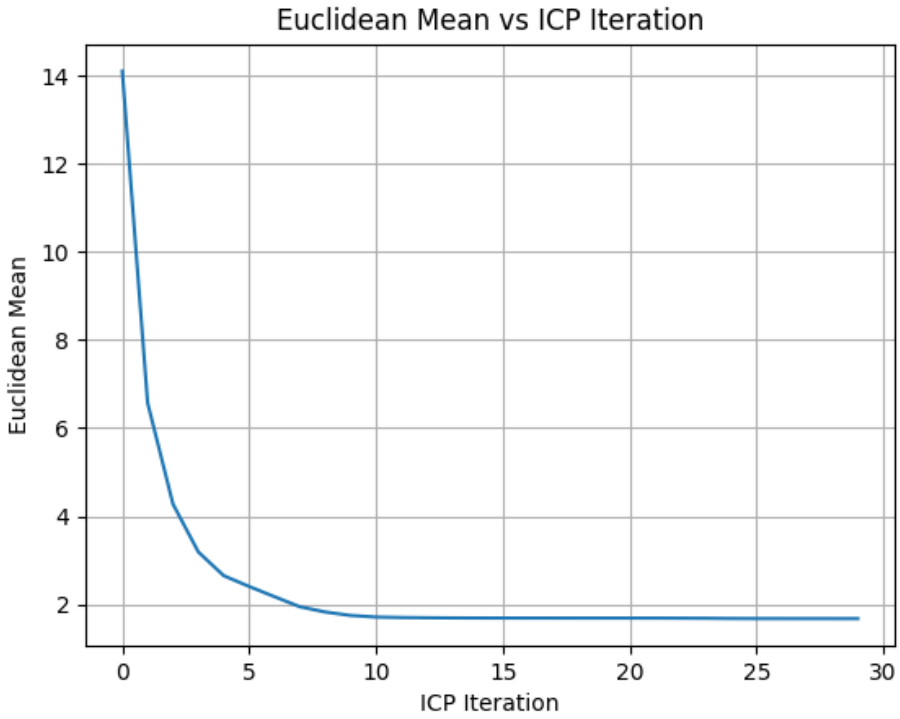

Figure 25 Mean Euclidean Distance vs ICP Iteration (Bunny)

From the figures illustrated above, we can see that the Mean Euclidean Distance decreases as there are more ICP Iteration. The Mean Euclidean Distance decreases for each ICP iteration for the following reasons. In our case, the main logic behind the ICP algorithm is to calculate the best matches between the source and the target. Then using the best matches found, the source points are transformed accordingly to the 6D pose found. This makes the new source points closer (more accurate) to the target points. For each ICP Iteration, this process repeats the steps that were mentioned. This explains the decrease in the Mean Euclidean Distance as the ICP Iteration increase. An interesting observation is that the Armadillo and Bunny Mean Euclidean Distance converged much faster than the Dragon dataset. This can indicate that there is more disparity between the source and target points even when they are aligned.

2.2 Nearest Neighbour Search

For question 2.2, the 6D pose that aligns the source point cloud to the target cloud was found using SVD algorithm. The following graphs illustrates the translation X, Y, and Z after each ICP iterations for the bunny, dragon, and armadillo dataset.

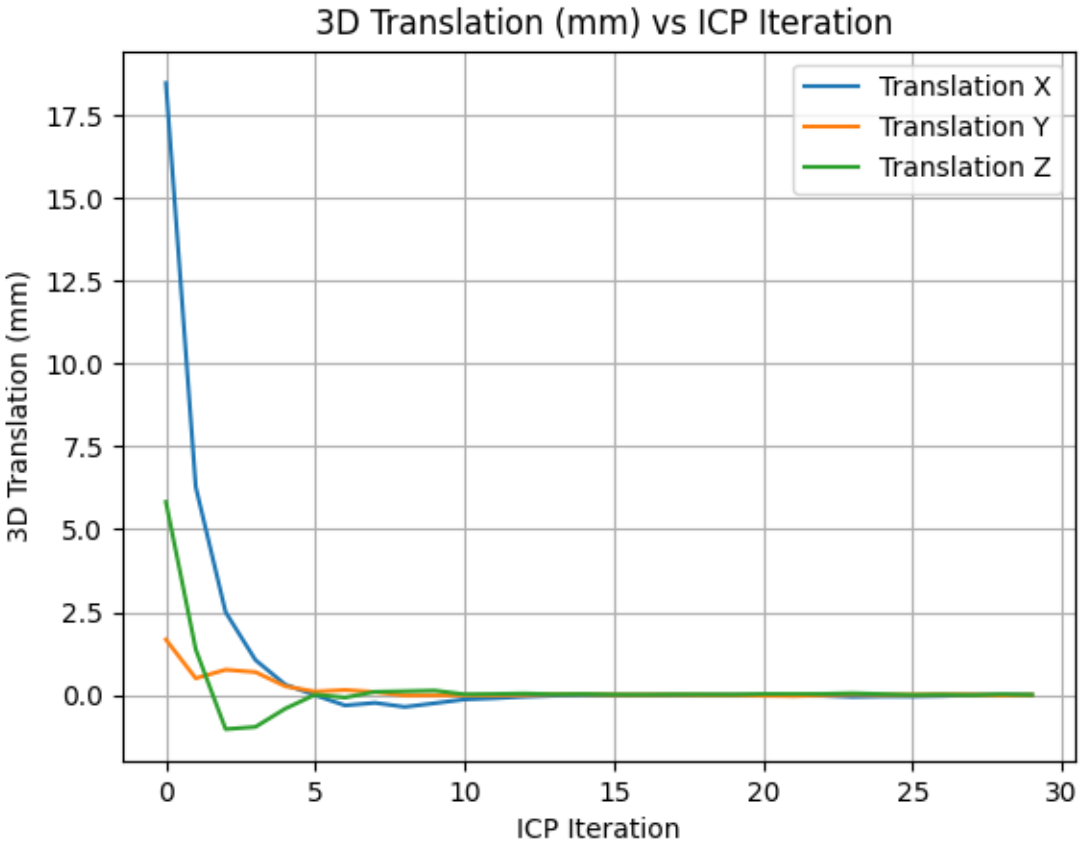

Figure 26 3D Translation (mm) vs ICP Iteration (Bunny Dataset)

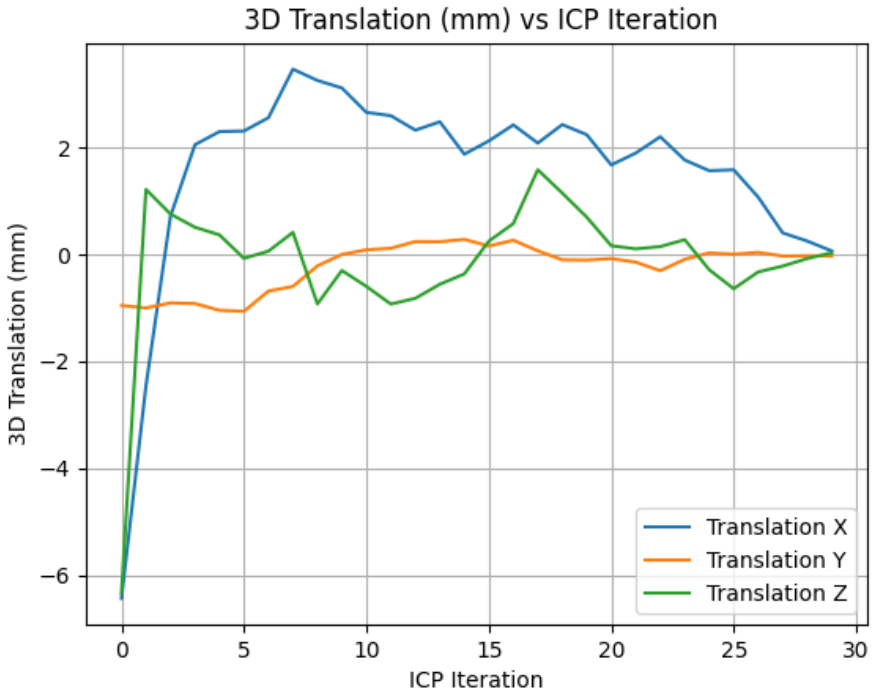

Figure 27 3D Translation (mm) vs ICP Iteration (Dragon Dataset)

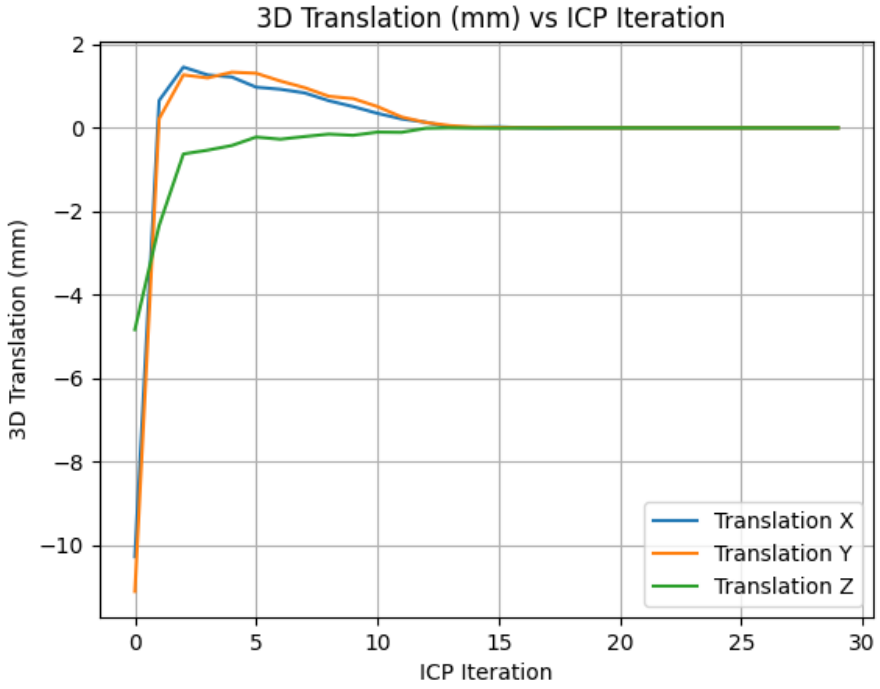

Figure 28 3D Translation (mm) vs ICP Iteration (Armadillo Dataset)

The technique to calculate the translation for each ICP is the same as the one discussed in question 2.1. The bunny and armadillo translations were able to converge to 0 rapidly (5 to 10 iterations). However, we see that the translation for the dragon does not converges as quickly. It does decrease as the iterations increases, but it took around 30 iterations. This can indicate disparity between its source points and target points. From our results, we can conclude that if we have matching source and target points, as the iterations increase, the pose matrix will converge towards an identity matrix (no required transformation). This is the expected behaviour since for each iteration, the source points are getting closer to the target points. Hence, the required transformation should become less and less.

2.3 Iterative Closet Point Registration [15 pts]

Given the completed functions nearest_search and estimate_pose, you need to implement the function icp that can align the source point cloud P to the target point cloud Q. Here we show an ICP registration result on the sample point cloud data. To validate your implementation, you need to visualize your registration results (before and after registration) on all training and test data (3 pairs of the point clouds). Moreover, we provide the ground truth 6D pose for the training point cloud pairs, and you need to compute the pose error (3D translation and 3D rotation error) between your estimated pose and the ground truth pose. To summarize, your need to show the following items for this question:

- *The visualization of the registrations on all the point cloud data (3 pairs).*
- *The 6D pose error on the training data set (object "bunny" and "dragon").*
- *The 6D pose output on the test data set (object "armadillo").*

The following are the visualization of the registrations on all the point cloud data (before and after 30 icp registrations). To have a fair comparison before and after the point cloud registration, the same axis were compared.

Bunny:

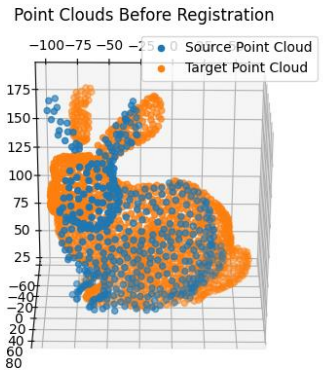

Figure 29 Visualization before ICP (Front Bunny)

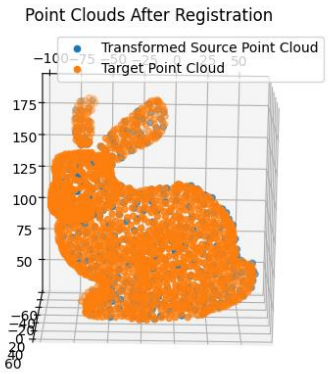

Figure 30 Visualization after ICP (Front Bunny)

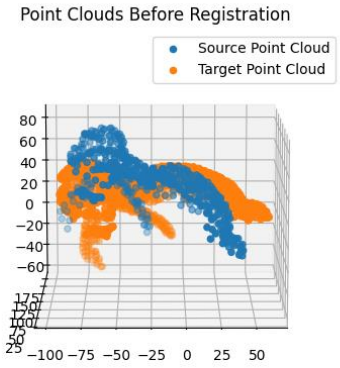

Figure 31 Visualization before ICP (Side Bunny)

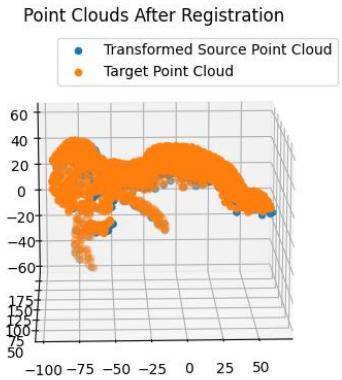

Figure 32 Visualization after ICP (Side Bunny)

Dragon:

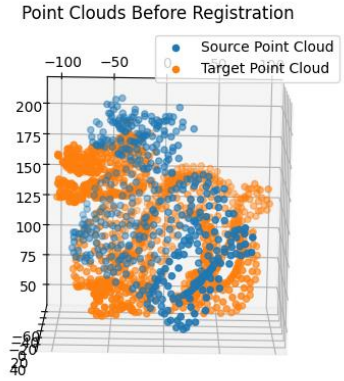

Figure 33 Visualization before ICP (Front Dragon)

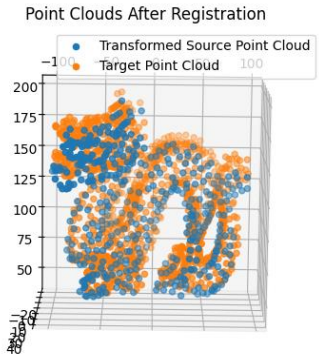

Figure 34 Visualization after ICP (Front Dragon)

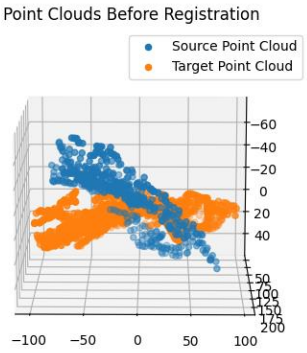

Figure 35 Visualization before ICP (Side Dragon)

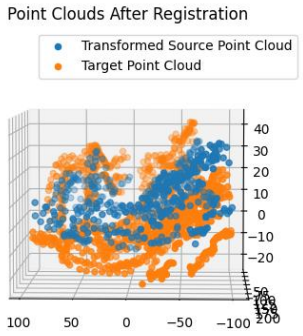

Figure 36 Visualization after ICP (Side Dragon)

Armadillo:

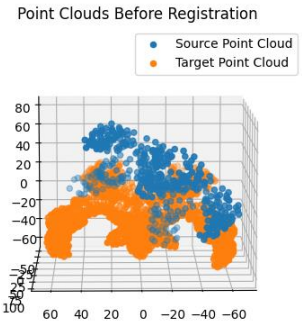

Figure 37 Visualization before ICP (Front Armadillo)

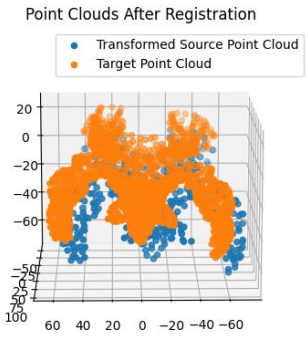

Figure 38 Visualization after ICP (Front Armadillo)

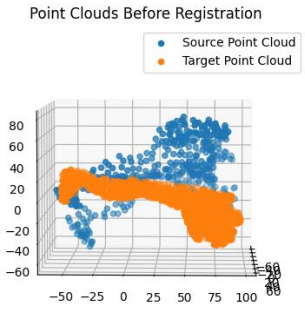

Figure 39 Visualization before ICP (Side Armadillo)

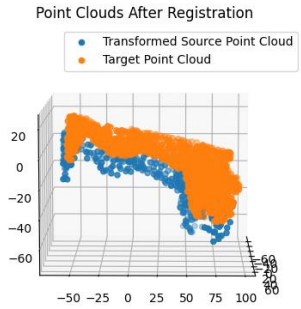

Figure 40 Visualization after ICP (Side Armadillo)

From our observations, we can see that point cloud are almost perfectly aligned for the bunny, and are well aligned for the armadillo, but there is a slight misalignment, and the dragon points are aligned alright, but appears to be the worst alignment among the tree tests.

The error for the 6D pose of the bunny and the armadillo was calculated using the relative error between each terms of the calculated pose and the ground truth pose. The following are the results obtained.

For the bunny:

The given ground truth was the following:

$$gt - bunny = \begin{bmatrix} 0.874 & -0.113 & -0.473 & 24.757 \\ 0.11 & 0.993 & -0.034 & 4.564 \\ 0.474 & -0.022 & 0.88 & 10.865 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The calculated pose was the following

$$calculated\ pose = \begin{bmatrix} 0.874 & 10.667 & -0.469 & 26.513 \\ 0.113 & 0.992 & -0.054 & 4.916 \\ 0.472 & -0.006 & 0.882 & 7.531 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Using the relative error between each matrix element, the following was obtained:

$$relative\ error\ \% = \begin{bmatrix} 0.061 & 10.667 & 0.867 & 7.092 \\ 2.668 & 0.125 & 58.505 & 7.714 \\ 0.374 & 73.492 & 0.132 & 30.684 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

For the dragon:

The given ground truth was the following:

$$gt - dragon = \begin{bmatrix} 0.71 & -0.318 & 0.629 & 46.364 \\ 0.319 & 0.941 & 0.115 & 3.316 \\ -0.628 & 0.119 & 0.769 & -6.464 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The calculated pose was the following

$$calculated\ pose = \begin{bmatrix} 0.734 & -0.271 & 0.623 & 38.026 \\ 0.434 & 0.892 & -0.123 & 1.938 \\ -0.522 & 0.361 & 0.773 & -31.785 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Using the relative error between each matrix element, the following was obtained:

$$relative\ error\ \% = \begin{bmatrix} 3.473 & 14.734 & 1.016 & 17.982 \\ 35.935 & 5.131 & 206.99 & 41.562 \\ 16.899 & 202.973 & 0.511 & 391.71 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

From the error pose, we see that the calculated pose for the bunny was pretty good. The biggest error in that pose estimation was the z transition (30.684%). The calculated pose calculated a z translation smaller than the ground truth (7.531 vs 10.865). Again, for the dragon, we see that the error pose is reasonable. However, the accuracy of the dragon is far smaller than the bunny. Some errors were much higher such as the z translation (error or 391.71%). The calculated z translation was -31.785 vs -6.464. From the visualization, that was expected since the bunny point cloud registration was much more accurate compared to the dragon.

Finally, the 6D pose output on the test set was the following:

$$\begin{bmatrix} 0.794 & -0.131 & 0.594 & -6.143 \\ -0.328 & 0.730 & 0.600 & -8.31 \\ -0.512 & -0.671 & 0.536 & -18.92 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Reference

[1] “Drawing Function of Keypoints and Matches.” *OpenCV*, https://docs.opencv.org/4.x/d4/d5d/group__features2d__draw.html.

[2] Sivamani, K., et al. “Real Time Feature Point Detectors for Determining Ego Motions.” *Procedia Computer Science*, Elsevier, 27 Feb. 2020, https://www.sciencedirect.com/science/article/pii/S1877050920300776#:~:text=There%20are%20various%20types%20of,size%20and%20high%20processing%20speed.

[3] YouTube, YouTube, 2 Mar. 2021, https://www.youtube.com/watch?v=wcqbiHonfbo&t=20s&ab_channel=FirstPrinciplesofComputerVision. Accessed 4 Nov. 2022.

[4] Howse, Joseph, and Joe Minichino. “Learning Opencv 4 Computer Vision with Python 3.” *O'Reilly Online Learning*, Packt Publishing, https://www.oreilly.com/library/view/learning-opencv-4/9781789531619/b838910f-5ceb-4181-a3dc-8d57aa21e6f0.xhtml#:~:text=A%20brute%2Dforce%20matcher%20is,is%20involved%20in%20the%20algorithm.

[5] Colin747Colin747 4, et al. “Orb/BFMatcher - Why norm_hamming Distance?” *Stack Overflow*, 1 July 1964, https://stackoverflow.com/questions/43614497/orb-bfmatcher-why-norm-hamming-distance.

[6] elenaelena 84322 gold badges99 silver badges1818 bronze badges, et al. “How Does the Lowe's Ratio Test Work?” *Stack Overflow*, 1 Sept. 1965, https://stackoverflow.com/questions/51197091/how-does-the-lowes-ratio-test-work.

[7] *Overview of the RANSAC Algorithm - Electrical Engineering and Computer ...* http://www.cse.yorku.ca/~kosta/CompVis_Notes/ransac.pdf.