AER 1515 - Perception for Robotics

Assignment 2

Submission guidelines: This assignment is due on **Friday, November 4, 2022** at 11:59 pm. Include the following items in a `.zip` file and upload to Quercus:

- PDF of assignment solution write-up
- `README` detailing how to run the code and any dependencies the code has
- Question 1
  - output file `P3_result.txt`
  - documented Python that generated your images and `P3_result.txt`
- Question 2
  - documented Python that generated your plots and the point cloud registration results

1 Feature Point Detection and Correspondences

Feature point detection and matching is an important task in many robotics applications, such as image stitching, 3D reconstruction, visual odometry and object tracking. This assignment will cover feature point detection in images, matching features between stereo camera pairs, and filtering out incorrect correspondences using an outlier rejection algorithm. The rectified stereo image pairs are from the KITTI dataset. You are provided with 10 image pairs along with the ground truth depth map to validate your algorithm (the training set). Your code will be tested on the 5 provided image pairs without ground truth depth maps (the test set).

- **Setting up your development environment.** The starter code is provided in the file **starter_code_feature.py**. It is developed for **Python 3.5+**. You will need the following Python packages for this assignment: **OpenCV** for working with image features, **NumPy** for working with arrays, and **Matplotlib** for displaying images. For students who are not familiar with Python and OpenCV, there are good tutorials here (make sure the version number in the dropdown menu on the top-left matches your OpenCV version).

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).

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

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**).

Figure 1: Feature points matching

1.3 Outlier Rejection

In the previous section, we were able to apply an epipolar constraint because the images are rectified. This may not always be applicable. Perform outlier rejection on your original feature correspondences from your matcher. Do not apply the epipolar constraint and use any algorithm of your choice (covered in class or from a paper). Tune your algorithm on the training set to achieve optimal results, and then test your algorithm on the test set. In your write-up, present the feature correspondences after outlier filtering (on the test set) and describe the algorithm you used, including any necessary equations. Briefly discuss the performance of your outlier rejection algorithm compared to using the epipolar constraint. Also, please present your feature matches on the test set in a *P3_result.txt* file with the same format as the *example.txt*. The code to generate this file is provided. Your results will be evaluated based on the correct number of feature matches and true positive rates, the RMSE of your depth estimates on the test set, and the explanation of your algorithm. Algorithms should be able to return an average of 100 matches per image on the test set. Additionally, you should filter any matches with depths over 80 m as long-range depth predictions become very inaccurate. **Make sure your result file is in the same format as the example file. The format is provided in the starter code.**

2 3D Point Cloud Registration

3D point cloud registration is an important problem in robotic perception, such as 3D reconstruction and autonomous driving. As shown in Figure 2, the goal is to find a rigid transform from one point cloud to another such that they align together. In this question, you will be developing a classical method for 3D registration: the iterative Closest Point (ICP) algorithm. This question will cover the required operations in the ICP algorithm: nearest neighbour search, rigid transformation estimation, and iterative point cloud alignment.

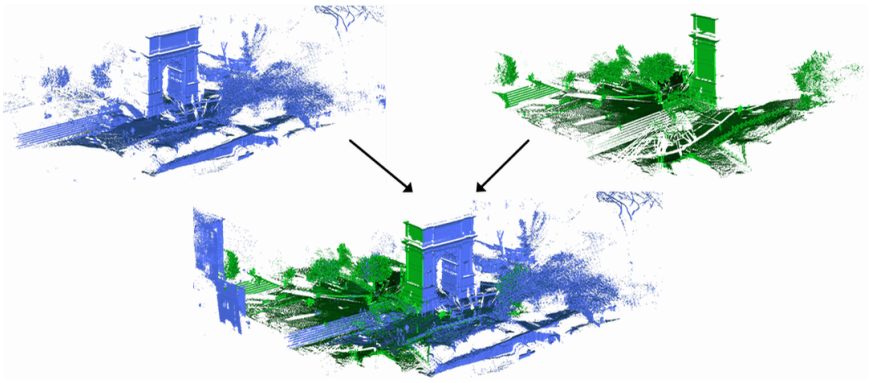

Figure 2: The Point cloud registration from ETH Terrestrial Laser Scanner (TLS) data.

Given the source point cloud and the target point cloud, you will need to estimate a 6D pose (4x4 transformation matrix) to align the source point cloud to the target point cloud, as shown in Figure 5. We assume we already have a good initial guess to start with. The 3D point cloud data are from the Stanford 3D Scanning Repository, the unit of the point cloud data is millimeter (mm). You are provided with two point cloud pairs along with the ground truth 6D poses to validate your implementation (the training set). Your code will be tested on the one provided point cloud pair without ground truth 6D poses (the test set).

- **Setting up your development environment.** The starter code is provided in the file **starter_code_registration.py**. The starter code provided is for **Python 3.5+**. You will need the following Python packages for this assignment: **Pandas** for loading point cloud data, **NumPy** for working with arrays, and **Matplotlib** for visualizing point clouds.

2.1 Nearest Neighbour Search

Given the input pair of the point clouds (source point cloud $\mathbf{P}$ and target point cloud $\mathbf{Q}$), implement the function **nearest_search** in the starter code that can build the correspondence between source and target point cloud for ICP algorithm. Specifically, for each 3D point $\mathbf{p}$ in the source point cloud $\mathbf{P}$, find the corresponding point $\mathbf{q_{nn}}$ in the target point cloud $\mathbf{Q}$ that has the closest euclidean distance. Also, you need to compute the average closest euclidean distance for all the correspondence. For the implementation, you can use the brute-force search and do not have to implement the KD-tree algorithm.

In this implementation, you build the correspondence $\{\mathbf{p} - \mathbf{q_{nn}}\}$ between source point cloud $\mathbf{P}$ and target point cloud $\mathbf{Q}$ for only one iteration. For a complete ICP algorithm, we need to call the function **nearest_search** in the ICP iterations. For each iteration, the ICP loss (the average closest euclidean distance) may not has the same value. Here we show a plot of the ICP loss with only 5 iterations. After completing Question 2.3, you need to show the plots of the ICP loss for all training and test data (3 pairs of the point cloud data) with 30 iterations. Please also describe how the ICP loss changes during the ICP iterations and explain why.

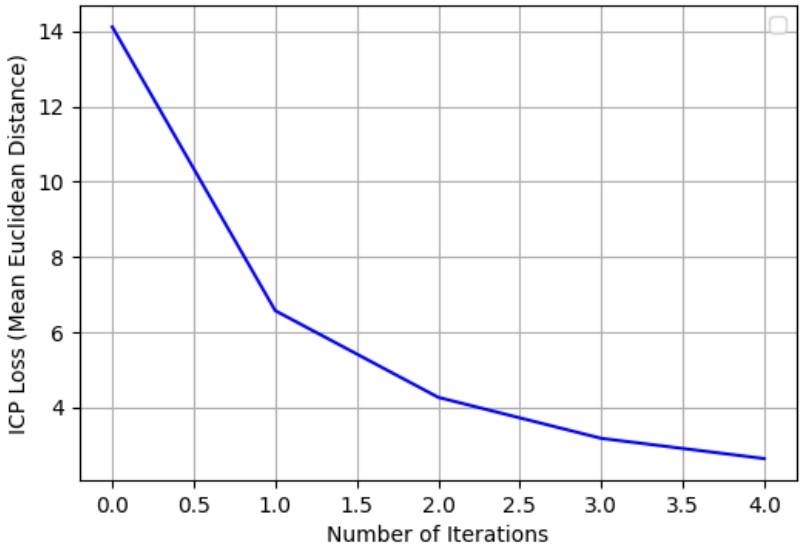

Figure 3: ICP loss with 5 ICP iterations on the example point cloud (object "bunny").

2.2 Pose Estimation from the Correspondence

Given the correspondences $\{\mathbf{p} - \mathbf{q_{nn}}\}$ between the source point cloud $\mathbf{P}$ and target point cloud $\mathbf{Q}$ (from Question 2.1), implement the function **estimate_pose** in the starter code for estimating the 6D pose (4x4 transformation matrix) that aligns the source point cloud $\mathbf{P}$ to the target point cloud $\mathbf{Q}$. This can be done by the singular value decomposition (SVD)-based algorithm or nonlinear optimization-based approach.

In this implementation, you compute the 6D pose based on the correspondences $\{\mathbf{p} - \mathbf{q_{nn}}\}$ for only one iteration. For a complete ICP algorithm, we need to call the function **estimate_pose** in the ICP iterations. For each iteration, the estimated 6D pose may not be the same. Here we show a plot of the estimated 3D translation (on X, Y and Z axes) with only 5 iterations. After completing Question 2.3, you need to show the plots of the estimated 3D translation for all training and test data (3 pairs of the point cloud data) with 30 iterations. Please also describe how the estimated 6D pose changes during the ICP iterations and explain why.

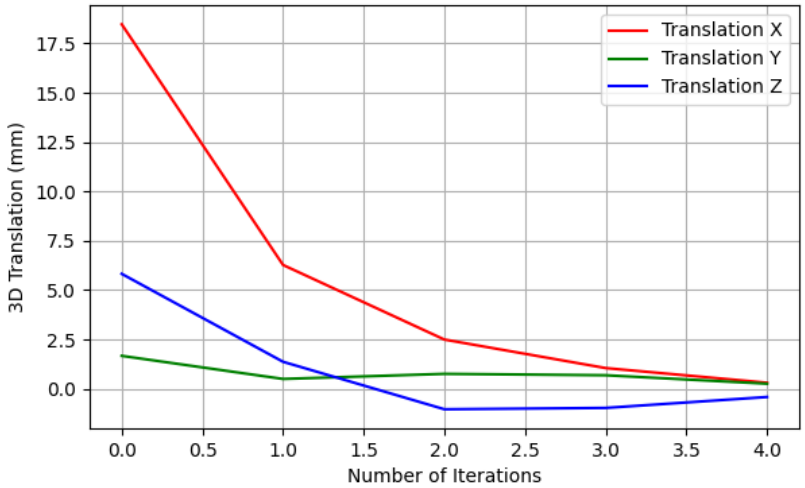

Figure 4: 3D translation results with 5 ICP iterations on the example point cloud (object "bunny").

2.3 Iterative Closet Point Registration

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").

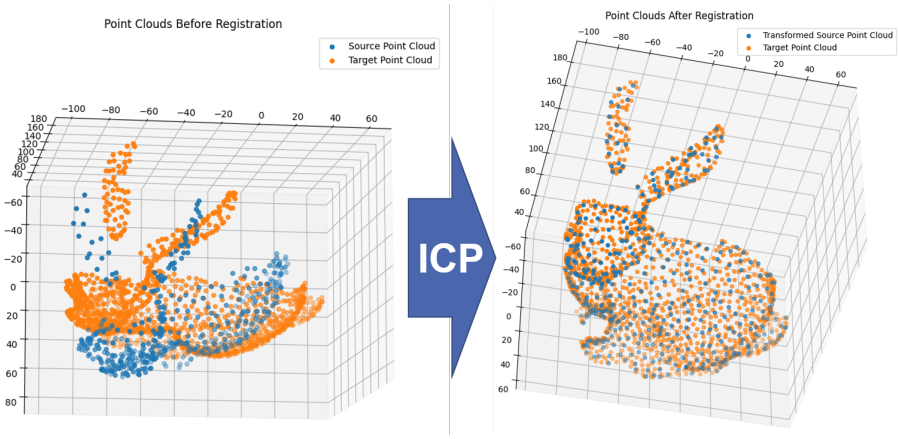

Figure 5: ICP registration result on the example point cloud (object "bunny").

Marking guidelines:

- Code Clarity [10 pts]
- Question 1.1: Feature Detection [15 pts]
  - Images with features overlaid [10 pts]
  - Explanation of feature detector and unreliable keypoints [5 pts]
- Question 1.2: Feature Matching [15 pts]
  - Images with non-outlier matches overlaid [10 pts]
  - Explanation of the feature matcher used [5 pts]
- Question 1.3: Outlier Rejection [15 pts]
  - Images with non-outlier matches overlaid [5 pts]
  - Results of the outlier rejection algorithm [5 pts]
  - Explanation of the outlier rejection algorithm [5 pts]
- Question 2.1: Nearest Neighbour Search [15 pts]
  - ICP loss plot [10 pts]
  - Description and explanation of the ICP loss [5 pts]
- Question 2.2: Pose Estimation from the Correspondence [15 pts]
  - The plot of the estimated 3D translation [10 pts]
  - Description and explanation of the estimated 6D pose [5 pts]
- Question 2.3: Iterative Closet Point Registration [15 pts]
  - The visualization of the registration results [10 pts]
  - The 6D pose error on the training set [2 pts]
  - The 6D pose output on the test set [3 pts]