CS 498: Assignment 3: Iterative Closest Point and Odometry

Due by 11:59pm Friday, Mar 25th 2022

March 4, 2022

Submission

In this assignment, you will code your own point cloud alignment and depth odometry algorithm to estimate camera poses and build 3D maps of the environment through raw depth observation. The starter code consists of 2 python files you will modify along with a folder of some images and metadata. Please put together a single PDF with your answers and figures for each problem, and submit it to Gradescope (Course Code: JBXJVZ). If your code produces figures, they should go in this pdf, along with a description of what you did to solve the problem. We recommend you add your answers to the latex template files we provided. For code submission, make sure you use the provided ".py" files with your modification and the dumped ".npz" file. The graders will check both your PDF submission and your code submission if needed.

Iterative Closed Point

In this part, your task is to align two point clouds. The algorithm we will leverage is called the iterative closed point. There are several components.

Question 1 (Unprojection)[1 pts]: Here, you will be modifying the function "rgbd2pts" in "icp.py". Given the input RGB-D image and the intrinsic matrix, your task is to unproject the RGB-depth image observations back to 3D as a form of a colored point cloud. We provide an open3d visualization for the point cloud and you should report the resulting figure in the document. Meanwhile, please avoid solutions using for-loop or directly calling off-the-shelf unprojection function.

Question 1 Answer: The purpose of unprojection is to take our RGB + Depth image along with the intrinsic matrix to then generate our original 3d image. Therefore for visualization we have to map our 2d coordinate with its RGB value on the flat image to the 3d coordinate. Here are my results for the Unprojection:

Image 0 :

Image 40 :

Question 2 (Rigid Transform Fitting) [3 pts]: Here, you will be modifying the function "fit_rigid" in "icp.py". For this question, your task is to implement the rigid transformation fitting algorithm, assuming

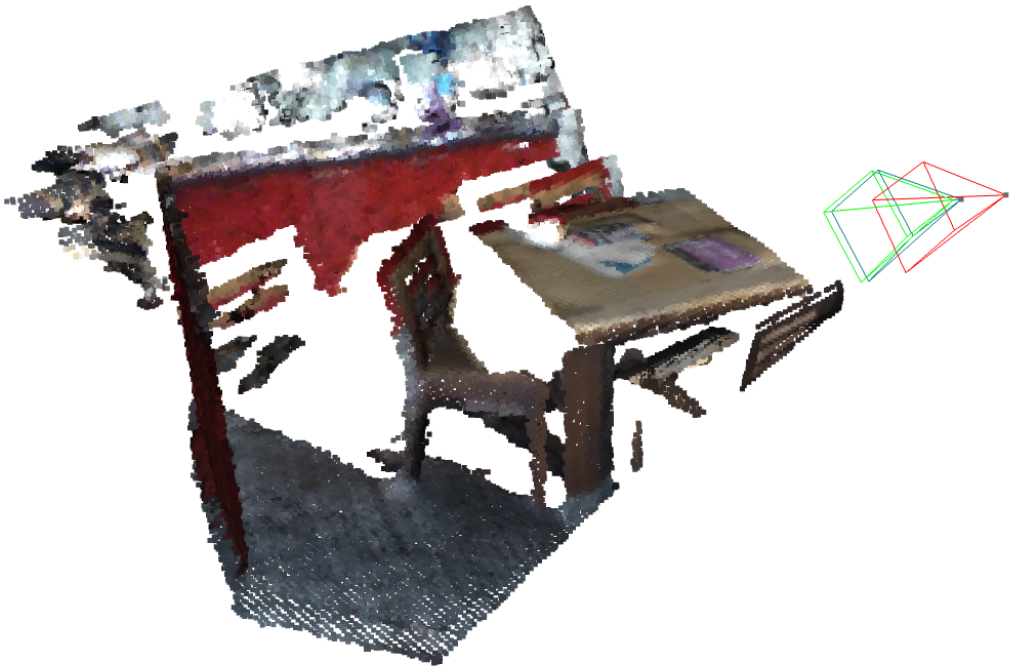
you are given N pairs of corresponding 3d points. You should refer to our lecture slide on depth sensing, and additional references could be found here: link1, link2, link3. You will use the point-to-point distance in this part. In addition, please provide an answer and necessary mathematical justification on what will happen if the input corresponding point pair contains errors and whether you could identify such situation.

Question 2 Answer: The purpose of Rigid Transform is given two point clouds with corresponding pairs of points, we can use SVD to calculate our Rotation and Translation matrix. When doing Point-to-Point correspondences, our goal is to minimize the distance between our source and target points. The issue though with our approach is that there may be errors in creating these correspondences. In ICP which we do later, we will be using KDTree to find the nearest neighbor, which is the closest point in the target point cloud to each point in our source. Although this is effective, this may not be the correct correspondence, therefore we cant perfectly optimize our rotation and translation matrix. The problem we are solving is essentially least squares and by not having correct correspondences we will fall into a local linear minimum rather than global. This is the reason for development of algorithms such as GO-ICP, where we attempt to initialize our first transformation as close as possible to the global minimum.

We can easily tell if we have errors by the overlapped point cloud. If we see poor overlap in certain areas of the point cloud, that is evidence enough that our rigid transformation may not be the most accurate across all points.

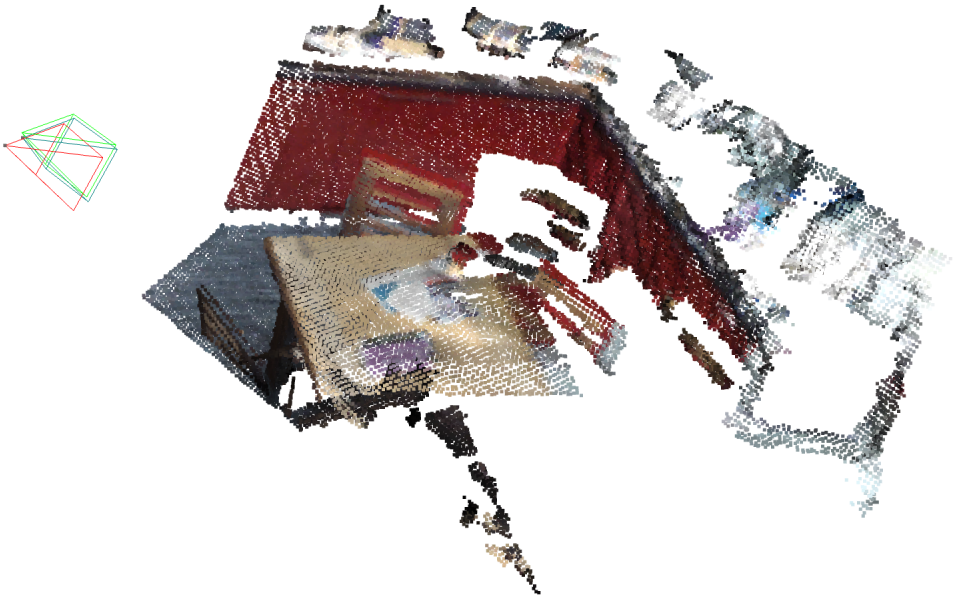
Question 3 (Point-to-Point ICP Loop) [3 pts]: Here you will be modifying the function "icp" in "icp.py". Your task is to complete the full ICP algorithm. You are provided a starter code with hints in the comments. Please complete it and try to align the provided two point clouds. We provide a visualization for the aligned results which includes your estimated camera poses compared to the GT camera poses. Please provide an image of the final visualization in this pdf along with a description of the algorithm.

Question 3 Answer: The purpose of the ICP algorithm is to take two point clouds, where we do not know what the correspondences are between them, and find the optimal rotation and translation transformations that would map our source point cloud on our target. The way we do this is, we first find the closest point in our target point cloud to each point in our source. This is not a perfect correspondence but rather an estimation that we hope will improve as we get our point clouds closer together. We then calculate our rotation and translation matrix to create our T, transformation matrix. We the apply this transformation matrix to our source point cloud hoping it is closer to the target. We will continue to repeat this process until we have no significant increase in our inlier ratio. The results of our Point to Point ICP (with a threshold of 0.04 for inlier points) is convergence to about 97 percent inlier ratio after 14 iterations. Here is the output of the merge of our two point clouds.



Question 4 (Point-to-Plane ICP) [2 pt]: Here you will be modifying "icp.py". Please extend your point-to-point ICP to allow it to take point-to-plane distance. Please run the alignment on your testing cases again and visualize the alignment results. Please justify when point-to-plane might be preferred over point-to-point distances (provide mathematical justification if necessary).

Question 4 Answer: Point to plane is another method of calculating the best rotation and translation matrix from source to target. The difference is, instead of trying to minimize our euclidean distance between predicted point correspondences, we will be trying to minimize the distance of the normal vector from points of our source to the surface (or plane) of the target. By doing so the number of iterations to get similar performance to point-to-point is greatly reduced. Also, with the assumption that our angle of rotation is small, we can approximate the parameters of our rotation and translation matrix with least squared regression. The intuitive benefit of point to plane is that areas of the source and target point clouds that are parallel to each other can slide along each other allowing for quicker convergence. Point to plane metric is therefore always more preferable than point to point due to the performance gains in using it, but also when a point cloud has clearly defined surfaces to calculate our normal information. The results of our Point to Plane ICP (with a threshold of 0.04 for inlier points) is convergence to about 96 percent inlier ratio after 4 iterations.



Question 5 (Translation and Rotation Error) [1 pt]: Now, we would like to evaluate how good our estimated poses are. Unlike other prediction tasks where the output is categorical data or euclidean vectors, pose estimation lies in SE(3) space. And we might want to evaluate rotation and translation components separately. Please check this reference and implement the translation and rotation error defined in Eq. 5-6. (link). Please report your translation and rotation errors for the two ICP algorithms, along with the estimated and gt pose.

Question 5 Answer: We calculate here our rotation/translation error for point to point and point to plane.

- Point to Plane Rotation/Translation Error: 0.043/0.022
- Point to Point Rotation/Translation Error: 0.014/0.019

Ground Truth Pose

$$\begin{bmatrix} 0.99833244 & -0.03859617 & 0.04303848 & 0.05789683 \\ 0.0402418 & 0.99846981 & -0.03803178 & 0.0360833 \\ -0.04150506 & 0.03970066 & 0.9983549 & -0.06788991 \\ 0. & 0. & 0. & 1. \end{bmatrix}$$

Estimated Pose

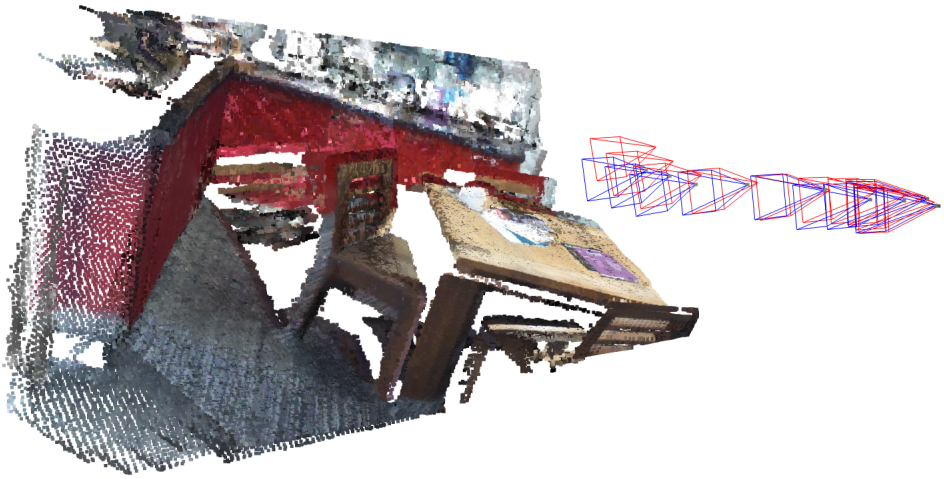
$$\begin{bmatrix} 0.99870453 & -0.03847881 & 0.03329619 & 0.07230002 \\ 0.03943812 & 0.99881117 & -0.02865096 & 0.02815352 \\ -0.03215415 & 0.02992698 & 0.99903478 & -0.07770738 \\ 0. & 0. & 0. & 1. \end{bmatrix}$$

Odometry

Now we will expand to estimate the trajectory of camera poses from an RGBD image sequence.

Question 6 (Odometry) [2 pts]: Here you will be modifying "odometry.py". Your task is to estimate the camera pose in an incremental fashion, which means that we will estimate camera pose $\mathbf{T}_t$ assuming the previous step's camera pose $\mathbf{T}_{t-1}$ is given. The key is to leverage ICP and estimate the relative transformation between the two and apply the difference through pose composition to get the current step's estimation. We will assume that frame 0's camera coordinate is the world frame origin. We also provide helper functions to visualize the aligned point cloud, read the GT camera poses and visualize the camera trajectory. Please complete the provided code and report the point cloud alignment and camera trajectory in your report.

Question 6 Answer: This odometry was done between frame 0 and frame 100 in 10 frame increments.



Question 7 (Relative Trajectory Error) [2 pts]: Odometry cares about the accuracy of the relative poses. Due to the global pose drift, estimating the absolute pose error as we did for ICP might not be suitable for evaluating odometry. Thus, people use average relative pose error to measure the pose estimation accuracy for odometry/slam algorithms. Based on your implemented rotation and translation error estimation algorithm, please implement the following average relative pose error: (link eq. 2 and eq. 3). Please visualize the estimated trajectory and ground-truth trajectory and report the error.

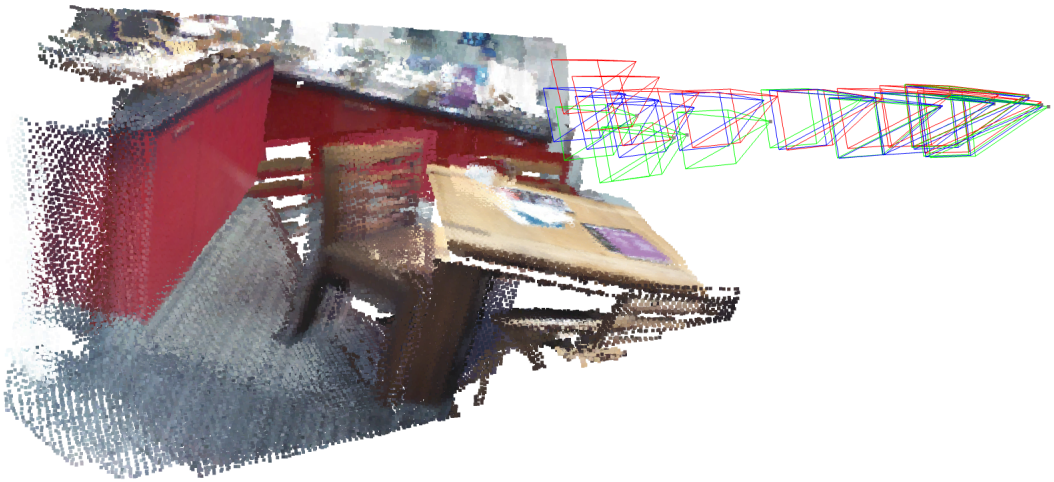
Question 7 Answer:

- Odometry Rotation/Translation Error: 0.009/0.016

Question 8 (Pose Graph Optimization) [2 pts]: So far, we have been only leveraging the relative transformation between adjacent frames. Absolute poses in global space are computed by accumulating the relative transformations. Do you think it is the best idea? What if there is one pair with a catastrophic alignment error? Given the odometry pose estimation of frame 0 and frame 40, do you anticipate that they will be consistent with directly running ICP estimation between their corresponding point cloud? In this question, you will leverage a tool called pose graph optimization to help with the challenge. The general

idea is simple: each frame's pose is a node in this graph, and any frames could be connected through an edge. On each edge, we define a cost measuring the agreement between the relative pose estimate between the two nodes and their pose estimation. By minimizing the energy, we are promoting global consistency across all the frames. More guidance is provided in the code.

Question 8 Answer: The greater the movement between two images the more difficult it becomes to calculate the rotation and translation in between. Have a series of pictures to iteratively calculate our rotation and translation matrix gives smoother results. We will use Open3d Pose graph optimization to get the ideal transformation between poses. Here is the output of my results. Each node of our pose graph will be the pose, and each edge will be the transformation calculated by ICP between the starting node and ending node.



We see that our graph optimization perfectly aligned our point clouds as well as got our green camera to world positions to track (although not perfectly) the predictions from odometry as well as ground truth.