CS 498: Assignment 3: Iterative Closest Point and Odometry

Mukai Yu

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 results are:

(a) Point cloud visualization fit to original image

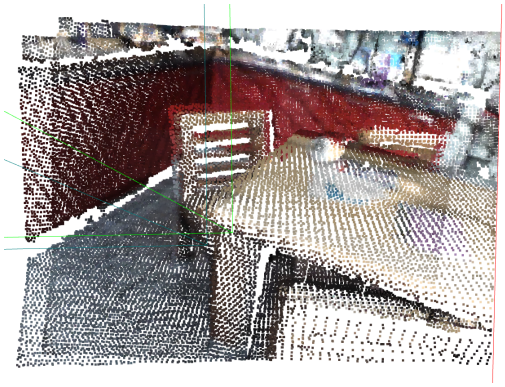
(b) Point cloud visualization in perspective view

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.

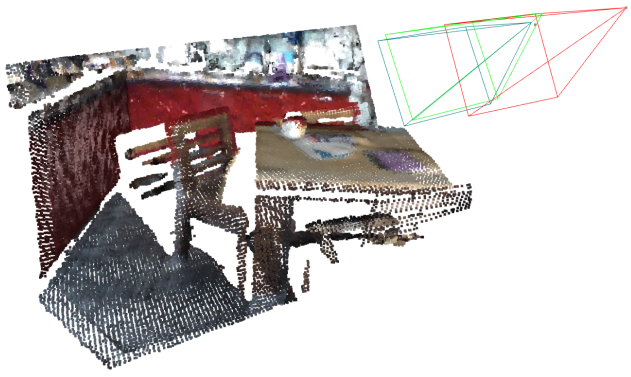
Question 2 Answer: *fit_rigid* will find the best rigid transformation that minimizes the distance between transformed source point cloud and its correspondence target point cloud even it contains error, because L2 distance is the optimization function. it can be detected if a source point gets assigned a different target point when finding closest point in the next iteration.

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 results are:



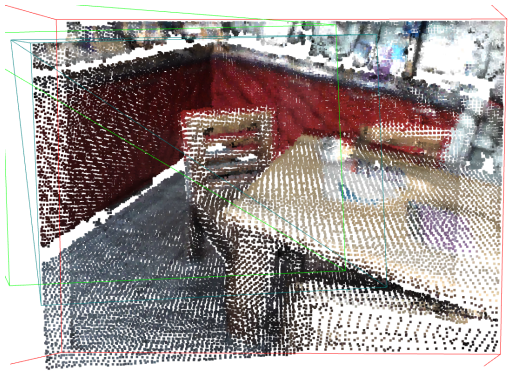
(a) Point-to-point ICP result in image view



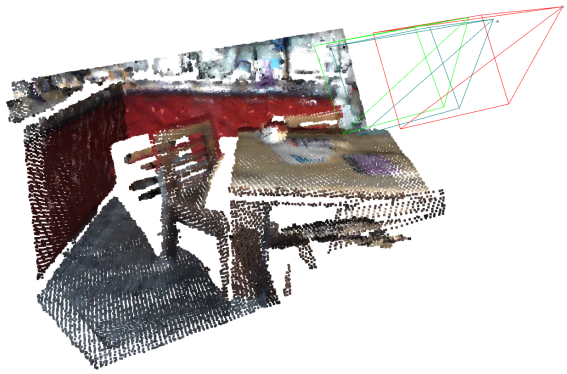
(b) Point-to-point ICP in perspective view

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: The results are:



(a) Point-to-plane ICP result in image view



(b) Point-to-plane ICP in perspective view

Point-to-plane ICP is preferred to point-to-point ICP when sampling of depth sensor is non-uniform, which can result in non-uniform sampling of a surface, in which holes in different location can be presented. In this case, point-to-point ICP cannot fit a good rigid transformation for the 2 point clouds because correspondence is assigned using euclidean distance, but point-to-plane can since it utilizes the underlying plane represented by the target points, thus it doesn't need perfect alignment in terms of point.

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:

	Translation Error	Rotation Error
Point-to-point	0.019144511560244624	0.014068664223088069
Point-to-plane	0.03168397288185179	0.07636310017130842

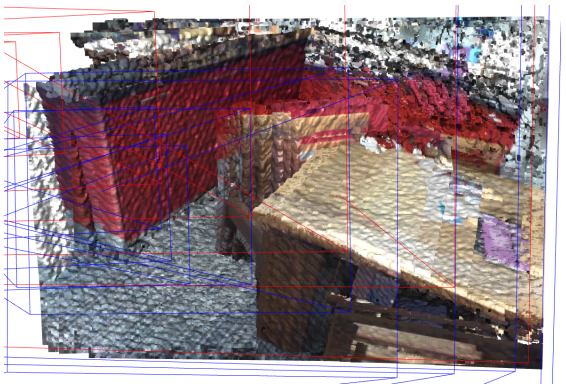
Odometry

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

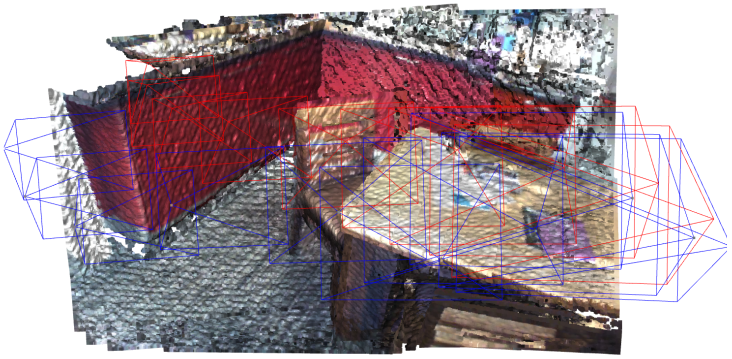
camera trajectory. Please complete the provided code and report the point cloud alignment and camera trajectory in your report.

Question 6 Answer: Cumulative error is quite big, The results are:



(a) Point-to-point odometry in image view

(b) Point-to-point odometry in top view



(c) Point-to-plane odometry in image view

(d) Point-to-plane odometry in top view

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:

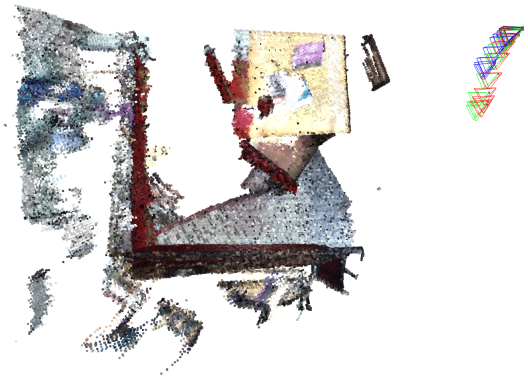
	Average Translation Error	Average Rotation Error
Point-to-point	0.031948811946544024	0.020897173834840212
Point-to-plane	0.023698607613728093	0.015091105241871176

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.

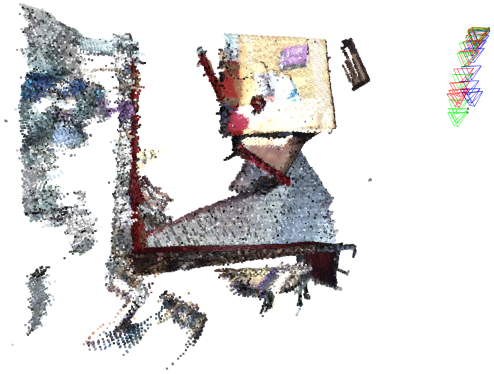
Question 8 Answer: Cumulative error cannot be eliminated, The results are:

(a) Pose graph optimization point-to-point odometry in image view



(b) Pose graph optimization point-to-point odometry in top view

(c) Pose graph optimization point-to-plane odometry in image view



(d) Pose graph optimization point-to-plane odometry in top view

Mapping (Bonus 3pt)

We believe you have a great camera trajectory estimation now. This bonus question will leverage the well-aligned camera trajectory and build a 3D map of the room. The general idea is to leverage volumetric fusion, described in KinectFusion paper and covered in our lecture. The key is to incrementally update the signed distance values for each voxel in the volume if a new depth frame comes. Try to fuse the color and depth into the volume and use a marching cube to get the extracted mesh. Leveraging existing volumetric fusion functions will not count. That being said, you could use them as a sanity check. You are allowed to use the GT camera poses. Try your best to get visually appealing results. We will choose the top solution and announce them in the class.