CS 498: Assignment 3: Iterative Closest Point and Odometry

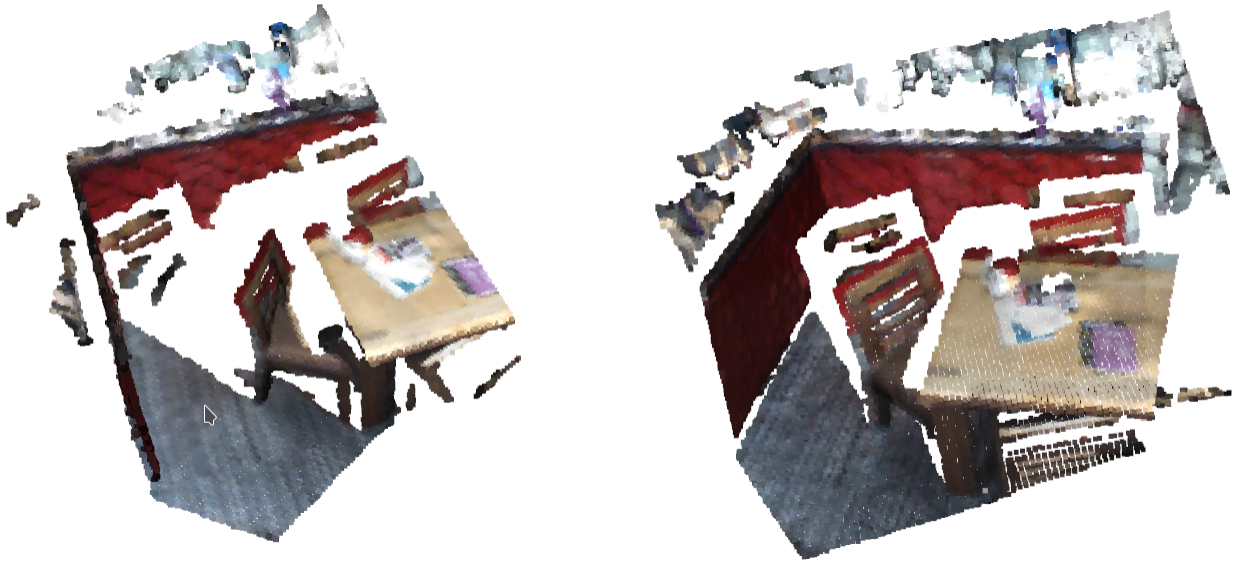
Chenhui Zhang

March 26, 2022

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.



(a) Source Point Cloud (b) Target Point Cloud

Figure 1: Point Cloud from Depth Image

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.

Answer Errors in the point correspondence due to sensor uncertainty, moving object, e.t.c. could result in outliers in the observation. Recall that our optimization objective for point-to-point ICP is

$$\min_{\mathbf{T}} \|\mathbf{T}\mathbf{p}_i - \mathbf{q}_i\|_2^2.$$

Due to the quadratic term in the optimization objectives of the vanilla point-to-point and point-to-plane ICP, the error term to optimize is very susceptible to outliers. Even a couple of outliers could drive the error term very high, shifting the optimization procedure to a less desired direction and resulting in slower convergence or the total failure of registration.

To account for the noise in point correspondence due to measurement uncertainty, we can weight the correspondences based on sensing uncertainty by using weighted mean and cross-covariance matrices. This is especially important when the measurement noise varies according to the distance, especially with stereo cameras. In addition, we can also use robust kernels to decrease the influence of outliers. We can use Huber (Smooth L1) kernel instead of quadratic kernel to down-weigh potential outliers. Besides, there are also some adaptive methods (Chebrolu et al., 2021) to adjust the kernel in an online fashion to account for moving objects. Furthermore, we can utilize moving object segmentation (Chen et al., 2021) to mask out moving objects, which could be outliers in the point correspondence. At last, it's also possible to reject the pairs that are not locally consistent by utilizing neighborhood information (Dorai et al., 1998).

Reference: ICP & Point Cloud Registration - Part 2: Unknown Data Association (Cyrill Stachniss, 2021)

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.

- Initialize $\mathbf{R} = \mathbf{I}_3$ and $\mathbf{t} = \mathbf{0}$, i.e. $\mathbf{T} = \mathbf{I}_3$
- Iterate until `inlier_ratio` ≤ 0.1:
  - Project source points $\mathbf{p}'_i = \mathbf{R}\mathbf{p}_i + \mathbf{t}$.
  - Establish correspondence between $\mathbf{p}'_i$'s and $\mathbf{q}_i$'s by constructing and querying KDTree.
  - Get relative update $\Delta\mathbf{T}$ from `fit_rigid` and update $\mathbf{T} := (\Delta\mathbf{T})\mathbf{T}$.
  - Update `inlier_ratio` based on the pair-wise distance from querying KDTree.

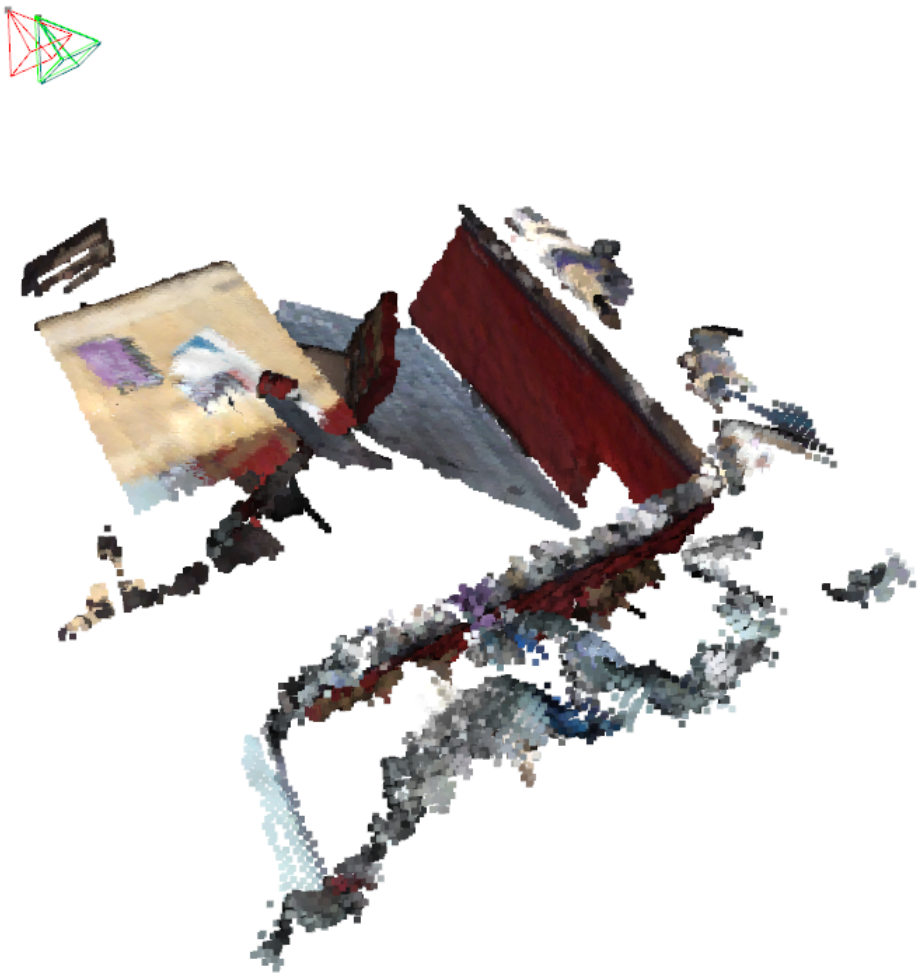


Figure 2: Aligned Point Cloud and Camera Pose from Point-to-Point ICP

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).

Answer For point-to-plane ICP, I modified `fit_rigid` to optimize the objective function of point-to-plane ICP. I used the linear approximation for sine and cosine functions under the assumption of small angle to linearize the optimization objective and solve it with least square described by Low (2004). Point-to-plane ICP takes the normal information of the surface into account when formulating its optimization objective. It is shown to have better convergence properties than point-to-point ICP (Rusinkiewicz and Levoy, 2001). In the presence of noise and outliers, point-to-point ICP could generates large numbers of incorrect correspondence when the point clouds are relatively far from each other or does not have many overlaps, which will slowing the rate of convergence. On the other hand, point-to-plane is less sensitive to the presence of noise. Furthermore, if we have a very sparse point cloud obtained by LiDAR, point-to-point ICP might not even be matching the real geometry surfaces and could simply matching the scanning pattern instead. Therefore, we might prefer point-to-plane distance over point-to-point distance when the point clouds **does not have very large overlapping regions** or when **the point cloud is very sparse**.

Figure 3: Aligned Point Cloud and Camera Pose from Point-to-Plane ICP

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.

- Point-to-Point ICP

```
Ground truth pose:
[[ 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.  ]]
Estimated pose:
[[ 0.99870445 -0.03847965 0.03329758 0.07229601]
[ 0.03943908 0.99881107 -0.02865307 0.02815681]
[-0.03215543 0.02992917 0.99903467 -0.07770555]
[ 0.  0.  0.  1.  ]]
Rotation/Translation Error (0.014068664222977584, 0.01914451156025902)
```

- Point-to-Plane ICP

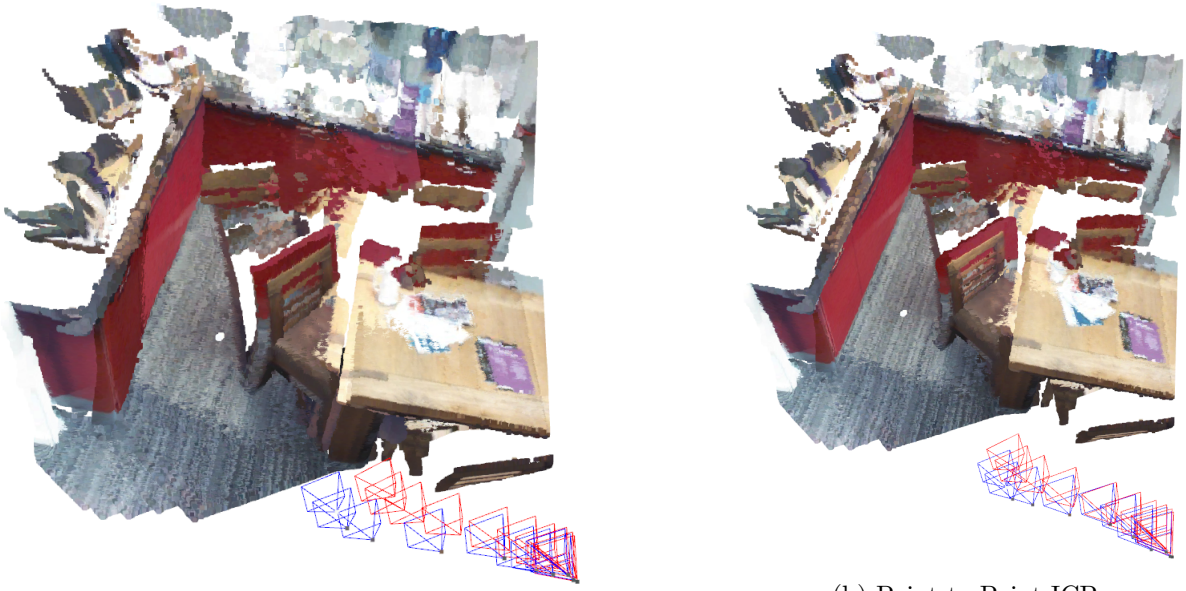
```
Ground truth pose:
[[ 0.99833244 -0.03859617 0.04303848 0.05789683]
[ 0.0402418 0.99846981 -0.03803178 0.0360833 ]
```

```
[-0.04150506 0.03970066 0.9983549 -0.06788991]
[ 0.  0.  0.  1.  ]]
Estimated pose
[[ 0.99675494 -0.04240015 0.05250252 0.03237006]
[ 0.04452167 0.9980827 -0.03159298 0.02568704]
[-0.05068381 0.03422677 0.99730098 -0.0626827 ]
[ 0.  0.  0.  1.  ]]
Rotation/Translation Error (0.0486711722165241, 0.028050196683065673)
```

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.



(a) Point-to-Plane ICP

(b) Point-to-Point ICP

Figure 4: Odometry Estimation by Incremental Update

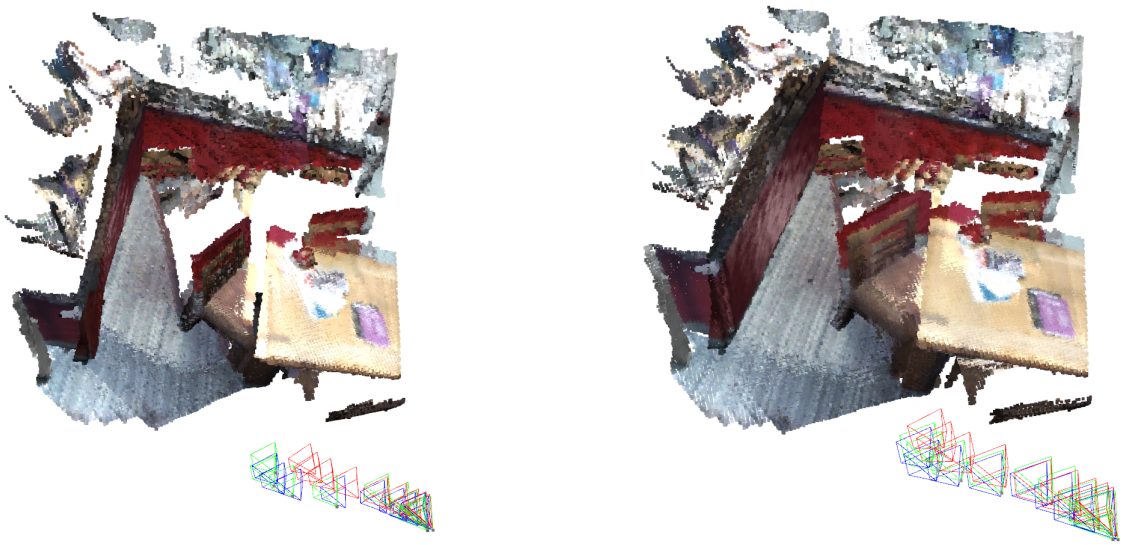
Question 7 (Relative Trajectory Error) [2 pts]: Odometry cares about the accuracy of the relative poses. Due to the global pose drift, estimating the absolute pose error as we did for ICP might not be suitable for evaluating odometry. Thus, people use average relative pose error to measure the pose estimation 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.

- Point-to-Plane ICP
  `Rotation/Translation Error 0.028555210323697563 0.04830801808931714`
- Point-to-Point ICP
  `Rotation/Translation Error 0.03303689194980612 0.038588892577576195`

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.

Answer I don't think it's a good idea to estimate the absolute pose by accumulating relative transformations. Although the estimation for each relative transformation could be locally optimal, we don't have any guarantee on the global optimality. Small errors in local steps could accumulate over time and one catastrophic failure could potentially destroy the final result. Given frame 0 and frame 40, I don't expect their odometry pose estimation is consistent with directly running ICP estimation between their corresponding point cloud because estimating poses in this incremental fashion will accumulate error overtime.



(a) Point-to-Plane ICP

(b) Point-to-Point ICP

Figure 5: Odometry Estimation by Pose Graph Optimization

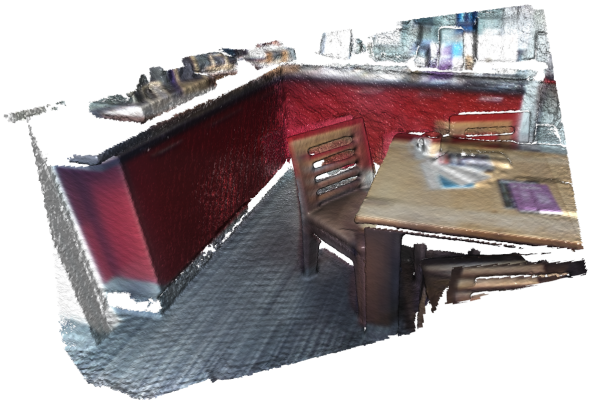
It seems that Pose Graph estimation (Green) with point-to-point ICP yields slightly better result in this sceanrio.

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.

To perform volumetric fusion, I followed the following resources:
`https://github.com/andyzeng/tsdf-fusion-python`
`https://github.com/chengkunli96/KinectFusion/blob/main/src/kinect_fusion/fusion.py`
`http://www.open3d.org/docs/latest/tutorial/t_reconstruction_system/customized_integration.html`
I first initialize a `VoxelBlockGrid` object from `Open3D` for 3D representation. Then, for each frame, I

- select the frustum block and unroll voxel coordinates;
- transform unrolled voxel coordinates to the camera coordinate system of the current frame;
- remove the points that go out of the frame after projection;
- identify the voxel indices that need updating;
- load the color and depth information at the coordinates projected from the voxel coordinates;
- calculate and truncate the new (T)SDF based on the incoming frame;
- update weights, TSDF, and color with the rules for Volume Integration covered in class;
- synchronize updates back to `VoxelBlockGrid`



(a) Volume Integration (Custom)

(b) Volume Integration (Open3D)

Figure 6: TSDF Volumetric Fusion

References

N. Chebrolu, T. Läbe, O. Vysotska, J. Behley, and C. Stachniss. Adaptive robust kernels for non-linear least squares problems. *IEEE Robotics and Automation Letters*, 6(2):2240–2247, 2021.

X. Chen, S. Li, B. Mersch, L. Wiesmann, J. Gall, J. Behley, and C. Stachniss. Moving object segmentation in 3d lidar data: A learning-based approach exploiting sequential data. *IEEE Robotics and Automation Letters*, 6(4):6529–6536, 2021.

C. Dorai, G. Wang, A. K. Jain, and C. Mercer. Registration and integration of multiple object views for 3d model construction. *IEEE Transactions on pattern analysis and machine intelligence*, 20(1):83–89, 1998.

K.-L. Low. Linear least-squares optimization for point-to-plane icp surface registration. *Chapel Hill, University of North Carolina*, 4(10):1–3, 2004.

S. Rusinkiewicz and M. Levoy. Efficient variants of the icp algorithm. In *Proceedings third international conference on 3-D digital imaging and modeling*, pages 145–152. IEEE, 2001.