16-833: Robot Localization and Mapping, Fall 2020
Homework #4 - 3D Dense Reconstruction Using ICP and Point-based Fusion

<u>Due: Monday November 30, 11:59pm, 2020</u>

Your homework should be submitted as a **typeset PDF file** along with a folder including **code only (no data)**. During your implementation, only write your code under the sections marked with `TODO`, `TEST`, or `EVAL`, and do not change any other line. The PDF and code must be submitted on **Gradescope**. If you have questions, post them on Piazza or come to office hours. Please do not post solutions or codes on Piazza. This homework must be done **individually**, and plagiarism will be taken seriously. You are free to discuss and troubleshoot with others, but the code and writeup must be your own. Note that you should list the name and Andrew ID of each student you have discussed with on the first page of your PDF file.

0. Preparation for Using Vectorization in Matlab (Highly recommended!)

Because of the computational complexity of this homework, we strongly suggest that you implement the main functions using vectorization techniques in Matlab. A general introduction can be found at:

<u>https://www.mathworks.com/help/matlab/matlab_prog/vectorization.html</u>.

With vectorization, you can speed up the runtime of your Matlab program from hours to seconds. Notice that you will not get full points if your system cannot generate correct outputs in the ***required time*** in each implementation section (accuracy and efficiency are both important!).

1. Overview

In this homework you will implement a basic 3D dense mapping system (see Matlab code file `ICP_FUSION.m`) that reconstructs a 3D dense model of a static indoor environment with given RGB-D image sequences. The system follows the tracking and mapping framework proposed in [1], which is commonly used in many state-of-the-art visual tracking and mapping systems. In this framework, mapping relies on the pose estimation from tracking to update the global model, and tracking uses the refined global model from mapping as reference. The visual tracking part of the system, which is also known as localization or visual odometry (VO), will be solved using point-to-plane iterative closest point (ICP) [2]. The mapping part will be done using point-based fusion [1]. However, unlike in [1], we do not consider the estimation of dynamic objects in this homework since the input data are from a static environment. Also, for simplicity and efficiency, you are required to implement both ICP and point-based fusion with some modifications from the original algorithms, which will be described in the later sections.

Camera parameters are stored in the file `cam_param.mat`. The input RGB-D sequence [3] is preprocessed into a sequence of colored pointcloud with normals and stored in the file `seq.mat`. Notice that there are no holes (every pixel has a valid measurement) in the original depth sequence, and therefore the pointcloud and the normals do not have holes either. The system will load the two files the first time it runs, which takes about 30 seconds. To save your time, do not clear workspace or close/reopen Matlab too often when working on this homework because it will have to reload the files each time.

2. Iterative Closest Point (ICP) (40 points)

Please read Section 3.5 in [2] and Section 5 in [1] before working on the following problems.

(a) What is the difference between the point-to-point ICP and the point-to-plane ICP algorithm in terms of the of the energy functions that they minimize? (5 points)

(b) In Section 3.5 in [2], Eq. 20 shows an important step of approximating a rigid body transformation as a linear function. This later enables them to solve the original nonlinear pose estimation problem in Eq. 16 incrementally in a closed form manner as seen in Eqs. 22-26.

Please write down the steps needed to derive Eq. 20, that is,

$$\begin{aligned}
\widetilde{\mathrm{T}}^z_{g,k}\dot{\mathbf{V}}_k(\mathbf{u}) &= \widetilde{\mathrm{R}}^z\widetilde{\mathbf{V}}^g_k(\mathbf{u}) + \widetilde{\mathbf{t}}^z \\
&\vdots \\
&= \mathbf{G}(\mathbf{u})\mathbf{x} + \widetilde{\mathbf{V}}^g_k(\mathbf{u})
\end{aligned} \tag{1}$$

Make use of the small angle assumption in Eq. 18 and the definition of the state vector $\mathbf{x}$ in Eq. 19 in your derivation. (10 points)

(c) Fill in all the parts marked with `TODO` in the following two files to implement a point-to-plane ICP algorithm:

- `getRigidTransform.m`

- `toSkewSym.m`

In `getRigidTransform.m`, you'll be solving the nonlinear pose estimation problem incrementally as a linear system of equations. The `TODO` portions essentially ask you to construct the A matrix and b vector and solve the corresponding normal equations. (Ref: Eqs. 24-26 in section 3.5 in [2])

Note that the point-to-plane ICP algorithm that we implement in `getRigidTransform.m` is slightly different from the one applied in [2], as it only iterates on one downsampled level instead of several pyramid levels. Also, instead of using the fast projective data association algorithm (requires high sensor frame rate, e.g. 30 fps), it finds the closest points within a local patch of size 3×3 around the corresponding downsampled points (because our input sequence has relatively low frame rate about 7.5 fps). The patch is labelled as:

1	2	3
4	5	6
7	8	9

where the center grid represents the corresponding point and the numbers in the grid are the indices of the cell `new_neighbors` in `findLocalClosest.m`. During your implementation, you can take `findLocalClosest()` function as a black box that gives you correct associations for each point. (15 points)

(d) Set `is_debug_icp = 1` in `ICP_FUSION.m` to test your ICP algorithm with the debug setting. The debug setting takes the current frame to be the next reference frame, and applies a built-in pointcloud merging function based on the pose estimation result from your ICP function to generate the global model. The system should output a 3D global point cloud model of an indoor scene in less than ***30 seconds*** if your implementation is correct and appropriate in terms of vectorization. Attach at least three clear figures of your results from different views in the PDF file and describe what you see in a few lines. (10 points)

NOTE: It is okay if the registration is not perfect (but results will still look reasonable) since the modified ICP only iterates on one downsampled level with few iterations.

3. Point-based Fusion (40 points)

Please read Section 4 in [1] before working on the following problems.

(a) Other than the ability to handle dynamic objects easily, what are some advantages of point-based fusion methods (like [1]) over volumetric fusion methods with regular grids (like [2]) in general? (5 points)

(b) Both the pointcloud and the normal of the current frame have to be transformed to the pose estimated by ICP before being fused with the global map. If a 3D point $\mathbf{p}$ with normal $\mathbf{n}$ is transformed to $\mathbf{p}' = R\mathbf{p} + \mathbf{t}$, where R and $\mathbf{t}$ are the rotation and translation respectively, what is the corresponding transformed normal $\mathbf{n}'$ given the same transformation of R and $\mathbf{t}$? (5 points)

(c) Take a look at `pointBasedFusion.m` which shows the framework of the point-based map fusion function. This function applies point-based fusion to get a single global map `fusion_map` which is simply an unstructured set of points $\bar{P}_k$ each with associated 3D position, normal, RGB color value, scalar confidence counter and time counter stored as a struct indexed by $k \in \mathbb{N}$.

To complete the `pointBasedFusion.m` function, fill in all the parts marked with `TODO` in the following four files:

- `projMapToFrame.m`

- `isUsableInputPoints.m`

- `avgProjMapWithInputData.m`

- `updateFusionMapWithProjMap.m`

Note that the point-based fusion that we implement in `pointBasedFusion.m` is slightly different from the original point-based fusion described in Section 4 of [1]. Firstly, we project the global model into the frame with original resolution instead of a supersampling frame. Secondly, the radius of each point is not calculated or used in the fusion process at all. Lastly, unstable points are not removed from the global model since our sequence is short and the environment is static. (20 points)

(d) Now set `is_debug_icp = 0` in `ICP_FUSION.m` to test your point-based fusion algorithm together with your ICP algorithm. The system should output two figures of the 3D global point cloud model (one with color and the other one with lighting) in less than ***30 seconds*** if your implementation is correct and appropriate in terms of vectorization. Again attach at least three clear figures of each of them from different views in the PDF file and describe what you see in a few lines. (10 points)

(If the output map contains obvious misalignments or distortion, there must be some problems in your implementation. Try to set `is_debug_fusion = 1` in `ICP_FUSION.m` to test the fusion part only independent of the ICP part you implemented in Q2. The `is_debug_fusion = 1` flag instead applies a built-in ICP function to align every input point cloud to the global model.)

4. Results and Discussions (20 points)

(a) Does using the fusion model as the next reference frame result in better ICP registration than using current frame as the next reference frame? Why or why not? (5 points)

(b) Set both `is_debug_icp` = `0` and `is_debug_fusion` = `0`, run the system, and take a look at the `summary.txt` file generated by the system. The `inliers` and `RMSE` show the number and root mean square error (RMSE) of all the matched point pairs in the last iteration of each frame-to-model registration respectively. How can you tell the performance of pointcloud registration given those values? Also, the `compression ratio` is calculated as $\frac{\#points\ in\ the\ output\ fusion\ map}{\#points\ in\ the\ entire\ input\ sequence}$. How can you tell the performance of point-based fusion given that value? (5 points)

(c) Keep the same settings in (b) and additionally set `is_eval` = `1` in `ICP_FUSION.m` to visualize the normal vectors, confidence counts, and latest update time stamps on the final fusion map. A false-colored map is shown in `Figure 2`, where different colors represents different normal directions of each point (The blue arrows show the directions of some randomly downsampled normal vectors). In both `Figure 3` and `Figure 4`, lighter colors represent larger values of confidence counts (in log space due to scaling issue) and time stamps respectively. Attach each of the three output figures in the PDF file and explain what you observe from the figures (at least one line for each figure). (10 points)

5. Code Submission Rules

Upload all of your Matlab code in the `ICP_fusion/` folder. Please paste the contents of the `summary.txt` file into your PDF writeup. Do not upload any `.mat` file, Matlab diary, image, or other types of data. **You may lose up to 20 points for not following the submission rules!**

References

[1] Keller, Maik, et al. "Real-time 3D reconstruction in dynamic scenes using point-based fusion." *International Conference on 3D vision (3DV)*, 2013. Link: *http://ieeexplore.ieee.org/document/6599048/*.

[2] Newcombe, Richard A., et al. "KinectFusion: Real-time dense surface mapping and tracking." *10th IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, 2011. Link: *http://ieeexplore.ieee.org/document/6162880/*.

[3] Handa, Ankur, et al. "A benchmark for RGB-D visual odometry, 3D reconstruction and SLAM." *IEEE International Conference on Robotics and Automation (ICRA)*, 2014. Website: *https://www.doc.ic.ac.uk/~ahanda/VaFRIC/iclnuim.html*.