CS498 Intelligent Robotics, Spring 2020
Homework 4: Point Clouds, ICP, and Mapping

Due date: 4/2/2020

Instructions: This assignment is to be completed *individually*. You are allowed to discuss the problems with your classmates, but all work must be your own. You are not permitted to copy answers, take written notes, or look up online solutions.

All coding will be done on the Klamp't Jupyter system. Setup is as follows:

1. Login and open up a terminal under the `Run` tab.
2. If this is your first time, run `git clone https://github.com/krishauser/cs498ir_s2020.git; cs498ir_s2020/setup.sh`. If this is not your first time, run `cs498ir_s2020/update.sh` to get the updated homework code. Go ahead and close the terminal tab, and return to the file browser. The homework code is in `cs498ir_s2020/HW4`.
3. When you run the notebooks, check the Kernel menu to make sure you are using the `cs498ir-virtualenv` kernel.
4. Make sure to periodically backup your code to your local machine. To do this, select the notebook in the file browser and choose `Download`. (We don't expect to have the server die, but Engineering IT won't make any guarantees!)

You may implement your calculations manually using Klampt, Numpy, or the elementary operations found in the Python math module, according to your personal preference.

To submit, upload your written answers to Gradescope by April 2 before 9:45am, and upload your programming answers in the form of Jupyter notebook files (*.ipynb) onto the class http://learn.illinois.edu site.

Overview

This homework assignment is structured a bit differently than prior assignments. You should *start with programming* and move on to the written questions. Also, all the programming will occur in a single notebook.

In the HW4 folder, you will find a folder "rgbd_dataset" which contains a small subset of the TUM RGB-D SLAM dataset (https://vision.in.tum.de/data/datasets/rgbd-dataset). Browse the RGB and depth images, showing a short sequence of an RGB-D sensor moving around a potted houseplant. We will be constructing a small map of this environment in this assignment.

The programming assignment builds on top of components A and B. If you get stuck on one component, and wish to move on, you may **request a complete implementation** of that component. This sacrifices your points for that particular question, but it allows you to move forward.

Note: In this assignment, it is quite likely that you will start some long-running processes. As you are debugging, you should start with short sequences and then gradually work your way up to the full sequence once you are confident in your code. Moreover, to kill a long-running process, use the "Kernel > Interrupt" function. Please be conscious of the finite amount of memory available on the system, and avoid clobbering the server with massive models, long-running processes, and infinite loops. When you are done, don't just close your browser window, but *manually interrupt the kernel* if it is running (dark dot in the upper right of your screen) before you log out of the system.

Programming Problem A: Point Cloud Transformations

The first few cells of Lab 4 load the dataset and plot a Z-cropped point cloud in the camera's local frame (positive Z is forward). A point cloud is a Numpy array of size N x D, and to learn how to work with Numpy arrays it will be helpful for you to look through the quick start guide here: https://docs.scipy.org/doc/numpy/user/quickstart.html.

1. In the first part of the problem, you will implement `transform_pc`, which performs a rigid transform on a simple XYZ point cloud (i.e., an array of size N x 3). This is done inplace, which means you will modify the Numpy array given to you.

   For now, you don't have to worry about colors or normals, so just consider each row in the array as a 3D point.

   Running the cell after the definition of `transform_pc`, you should see the houseplant in a different orientation, corresponding to its ground truth camera pose. (The plant should be pointed "away" from you; this will look a little weird due to the way that point clouds are rendered in Matplotlib)

2. In the next cell, you will see a colored houseplant in the camera's local frame. This is an N x 9 array, whose columns are X, Y, Z, R, G, B, Nx, Ny, Nz. The color of each point is an (R,G,B) tuple, and the estimated normal direction at each point in the camera frame is (Nx,Ny,Nz). You should now modify `transform_pc` so that the R,G,B channels are untouched, while the normal information is rotated.

   In this cell, uncomment the line starting with `transform_pc` to test whether you code is properly transforming the normal directions.

Programming Problem B: Scan merging

In the cell marked for Question B, you will be asked to implement routines to merge point clouds together.

1. Complete the naïve `merge_scans` function, which simply concatenates all transformed point clouds into a unified point cloud.
2. Complete the smarter `merge_scans_grid` function, which keeps track of an occupancy grid of which voxels contains points. For each new point you consider adding to the representation, calculate the voxel to which it belongs. If no other points belong to that voxel, add it to the output point cloud. If any other points belong to that voxel, throw it out.

   Hint: a Python dictionary indexed by tuples `(int(x/res),int(y/res),int(z/res))` is a good representation of an occupancy grid of resolution `res`.
3. Test the methods by running the following cell, playing around with `N_frames`, the maximum value in `zlim`, and whether `merge_scans` or `merge_scans_grid` is called. For `merge_scans`, you will want to keep `N_frames` relatively small.

Programming Problem C. ICP

Lastly, we will build upon `transform_pc` and `merge_scans_grid` to implement an ICP algorithm. In the next cell, there are several utility subroutines, as well as a skeleton of a function `icp_unstructured`. The skeleton is mostly finished, except that you need to implement the key ICP strategy for finding matches and rejecting outliers.

1. The ICP skeleton sets up a K-D tree nearest neighbor data structure for the target point cloud before the main ICP iteration. Subsample a set of 250 points from the source point cloud, and use the `kdtree_position` object as indicated in the comments to determine the distances and points in the target point cloud that are closest to each point in the subsample. Create the `matches` object – list of pairs (a,b), where a is an index in the source point cloud and b is a matched index in the target point cloud – so that the subsequent fitting step works without error.
2. Try running the last 2 cells, observing the estimated trajectory produced by ICP and the overall reconstruction. If your algorithm is running properly, there will be some (not necessarily monotonic) reduction in the RMSE. Observe the progress of ICP, the trajectory estimation errors, and any noticeable artifacts in the final reconstruction, taking notes for written problem 3.

   (Note: the default settings try matching 3 point clouds together, each spaced 5 frames apart in the video sequence.)

3. Now, implement an outlier rejection technique where you first subsample of the source point cloud with 1000 points, then retain only the 50% of *closest matches* for doing fitting. Repeat

your observations as you did in step 2.

4. Finally, perform some other enhancement to the basic ICP algorithm. Two obvious options would be to use matching on color and normal information, or fitting using the point-to-plane distance. Repeat your observations as you did in step 2 for this enhancement.

Written Problem 1: Perspective Camera Modeling

A. The third cell of the programming assignment contains utility code that converts depth / color images to point clouds. Are the values of the depth image proportional to the Euclidean distances between points and the camera focal point, or are they something else? How might you tell from the code?

B. Can you reverse-engineer the camera's horizontal and vertical field of view, in angles, from the data given in this cell, and the knowledge that images are 640x480 pixels? If so, calculate these quantities. If not, explain why not.

Written Problem 2: Scan Merging

A. In the `merge_scan` method, what is the computational complexity of combining k scans? Describe the generally foreseeable problem using the naïve `merge_scan` function for reconstructing maps from large video sequences.

B. In the `merge_scan_grid` method, what is the computational complexity of combining k scans? What dictates the storage complexity? List some of the empirical storage requirements for k scans of maximum depth zmax, where k and zmax are parameters that you sweep over some reasonable range.

C. What tradeoffs are involved in choosing the resolution of the volumetric grid? Why might you want a fine resolution? Why might you want a large resolution?

Written Problem 3: ICP analysis

A. The ICP algorithm prints out progress of the root-mean-squared error (RMSE) before and after the fitting step as the iteration progresses. After fitting, the error is always less than the before-fitting error. But the error doesn't seem to monotonically decrease between iterations. Explain why this is the case.

B. In Programming Problem C.2, describe the progress of your basic ICP algorithm. How well does it perform empirically, in terms of computation time, camera position estimation error, and the quality of the reconstruction?

C. After completing Programming Problems C.3 and 4, repeat part B and describe the performance of each variant. Justify how the changes that you've made affect the performance of each variant. (Do so to the best of your ability; sometimes, these impacts will be rather mysterious!)

D. Examine the way that the reconstructed model is produced in the last cell: a "world model" is maintained over the entire iteration, and each scan is aligned to the world model. Consider the alternative approach of aligning one scan s_i to the prior scan s_{i-1}, and then accumulating the relative transforms T_i^{i-1} to determine the world transform of scan *i*. (In other words, use the recursive equation $T_i = T_{i-1} \cdot T_i^{i-1}$)

   Do you think the scan-to-scan alignment approach will do a better or worse job than aligning to a world model? Explain why.