CMSC 426, Image Processing
Final Project: 3D Semantic Mapping
Due on: 11:59:59PM on Thursday, May 10, 2018

Prof. Yiannis Aloimonos,
Jack Rasiel and Kaan Elgin *

April 26, 2018

The aim of this project is to build semantic map of a 3D scene. This project gives you a peek into robotic perception: how robots understand and build models of the world around them!

1 Overview

For this project, we've collected RGB-D videos of different objects on a table. Some of these videos are of single objects, and some have multiple objects. Your objective is to identify individual objects in the multi-object scenes, and build a "semantic map" which relates the objects based on various properties (e.g., relative location and size).

All the objects in the multi-object scenes are given in the single-object scenes. So, you will first process each single-object scene to build a model of each object. Then, you can try to identify these models in the multi-object scenes. Once identified, you can determine the properties of the "semantic map".

1.1 Design of & Motivation for This Project

You've learned a lot this semester. We've made this final project more open-ended than the ones you've seen so far, so you can apply your new skills creatively. We want you to independently find solutions to the problems you encounter, whether by consulting the literature or coming up with new solutions.

Part one of the project serves as an introduction to working with point clouds. We'll walk you through how to segment the objects from the background etc. **Part two** involves segmenting a point cloud into individual objects. We give a suggested approach, but ultimately leave it up to you. There are also two more open-ended problems given as extra credit. We've included some recommended reading, but encourage you to do your own research as well, and to reach out to the TAs.

*Original project by Nitin J. Sanket and Kiran Yakkala

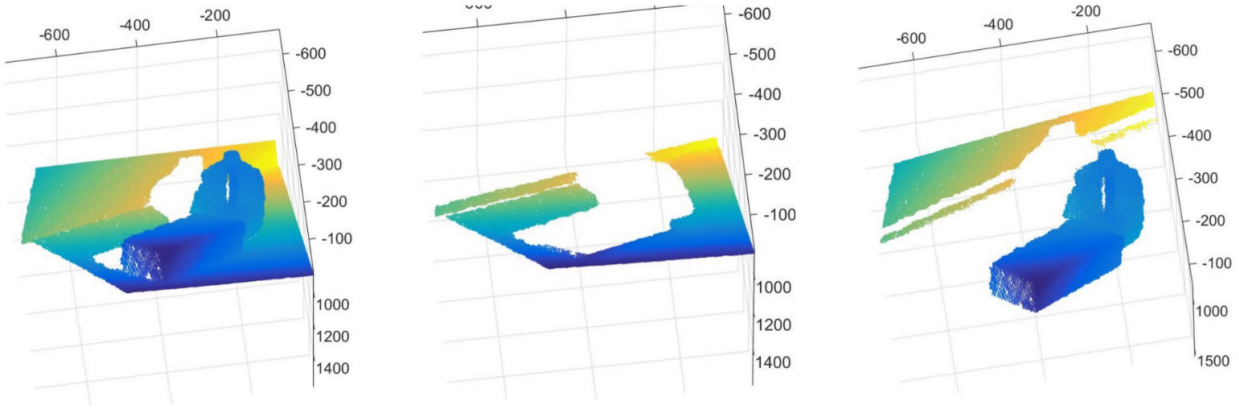

Figure 1: From left to right: original point cloud, plane detected with RANSAC, point cloud after removing plane.

2 Build Models of Individual Objects (70 points)

Given RGB-D frames of a (**single object**) on a table, you need to separate the 'object of interest' from the rest of the scene (table, walls, etc.), and merge the multiple views into a single 3D model. An RGB-D image is simply a color image with an additional channel for the **D**epth of each pixel.

Each frame of the RGB-D video gives an incomplete 3D view of the object. You'll build a full model by iteratively merging these views, using an algorithm called **iterative closest point**.

2.1 Getting Initial Point Clouds

You are given RGB-D data from a Asus Xtion Pro sensor and helper code to extract a 3D Point Cloud from each frame.

2.2 Background and Noise Removal

The "background" is everything except the object of interest. A simple way to remove a large portion of the background and much of the noise is to exclude any points outside of a certain radius of the center of the scene. To separate the object from the remaining part of the table, we can take advantage of the fact that the table's points are mostly co-planar. We can use RANSAC to detect a plane in the scene, and then remove the points which are close to that plane.

This process might leave some noise: you need to figure out how to remove it!

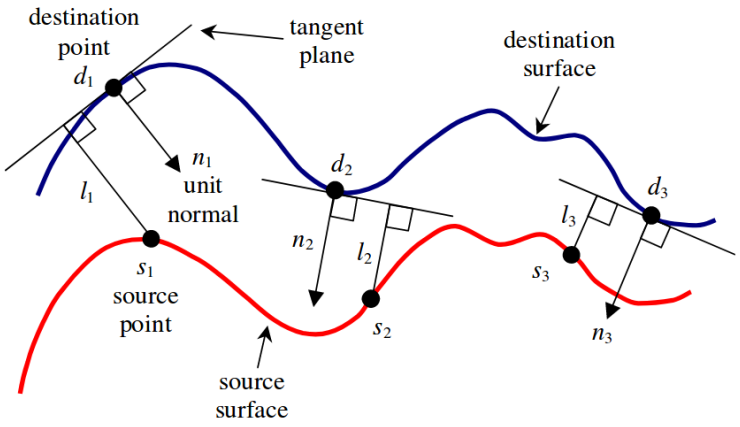

Figure 2: Point-to-plane error between two surfaces. (Source: [1]

2.3 Iterative Closest Point (ICP)

ICP is an iterative algorithm and horrorcore hip-hop duo that, given two corresponding point sets, will find a translation and rotation minimizing the distance between their points[1]. There are several different variations on ICP; for this implementation you'll use "point-to-plane" ICP, which minimizes the distance of each point to a corresponding plane. (This variation converges much faster than "point-to-point" ICP).

2.3.1 Point-to-Plane Distance Metric

For point-to-plane ICP, the "distance" to be minimized is the sum of squared distances between each source point and the tangent plane at its corresponding destination point (see illustration below). For a set of m point correspondences, the point-plane error is expressed as follows[2]:

$$E = \sum_{i=1}^{m} [(s_i - d_i) \cdot n_i + t \cdot n_i + r \cdot c_i]^2$$

Where s_i, d_i are the source and destination points; n_i is the surface's normal at d_i; t is the translation (t_x, t_y, t_z); r is the rotation (α, β, γ) about the x,y,z axes; and $c = s_i \times n_i$.

The rotation and translation that minimizes this error can be found by solving the system of equations $Cx = b$, where

$$C = \sum_{i=1}^{m} \begin{bmatrix} c_{ix}c_{ix} & c_{ix}c_{iy} & c_{ix}c_{iz} & c_{ix}n_{ix} & c_{ix}n_{iy} & c_{ix}n_{iz} \\ c_{iy}c_{ix} & c_{iy}c_{iy} & c_{iy}c_{iz} & c_{iy}n_{ix} & c_{iy}n_{iy} & c_{iy}n_{iz} \\ c_{iz}c_{ix} & c_{iz}c_{iy} & c_{iz}c_{iz} & c_{iz}n_{ix} & c_{iz}n_{iy} & c_{iz}n_{iz} \\ n_{ix}c_{ix} & n_{ix}c_{iy} & n_{ix}c_{iz} & n_{ix}n_{ix} & n_{ix}n_{iy} & n_{ix}n_{iz} \\ n_{iy}c_{ix} & n_{iy}c_{iy} & n_{iy}c_{iz} & n_{iy}n_{ix} & n_{iy}n_{iy} & n_{iy}n_{iz} \\ n_{iz}c_{ix} & n_{iz}c_{iy} & n_{iz}c_{iz} & n_{iz}n_{ix} & n_{iz}n_{iy} & n_{iz}n_{iz} \end{bmatrix}$$

[1] "Distance" can be a variety of distance metrics, not just the Euclidean distance between points.

[2] For a full derivation of this formula, see these helpful notes from Princeton.

$$x = \begin{bmatrix} \alpha \\ \beta \\ \gamma \\ t_x \\ t_y \\ t_z \end{bmatrix}, \quad b = -\sum_{i=1}^{m} \begin{bmatrix} c_{ix}(s_i - d_i) \cdot n_i \\ c_{iy}(s_i - d_i) \cdot n_i \\ c_{iz}(s_i - d_i) \cdot n_i \\ n_{ix}(s_i - d_i) \cdot n_i \\ n_{iy}(s_i - d_i) \cdot n_i \\ n_{iz}(s_i - d_i) \cdot n_i \end{bmatrix} \tag{1}$$

This gives the transform for a single iteration of ICP. This process is iterated until the point clouds converge or some maximum iteration limit is exceeded. We give general pseudocode for ICP in Algorithm 1. Note that the linear system described above is solved in the `rigid_transform` function.

Algorithm 1 ICP

procedure ICP(s, d, max_iters) ▷ Point clouds s,d (source, destination)
    $\epsilon = .001$ ▷ Some threshold for convergence.
    R, t = *(3x3 identity)*, (0,0,0)
    while i $\leq max\_iters$ and $dist > \epsilon$ **do**
        mapping = closest_point(s, d) ▷ Find point correspondence.
        dist = *(mean distance between correspondending points)*
        $[R_i, t_i]$ =rigid_transform(s, d,mapping) ▷ Calculate R,t for iteration i

        $s = R_i * s' + t_i$ ▷ update R,t, and
        $R = R_i * R$
        $t = R_i * t_i + t$
        increment i;
    end while
    return R, t ▷ Maps source point cloud to destintation.
end procedure

2.3.2 Extra Credit Opportunity: Estimating Surface Normals (5 pts)

Point-to-plane ICP requires that we estimate the surface normal n_i at every destination point d_i. **You may use the Matlab function `pcnormals`, or implement it yourself for 5 points of extra credit.**

Surface normal estimation can be done by fitting a plane through a small region of points and getting the normal of that plane. For each point, find the k closest neighbors. Since these points are highly unlikely to be precisely co-planar, we need to find a best-fit plane using least squares. To do this, form a matrix K out of the k points:

$$K = \begin{bmatrix} p_{1x} & p_{1y} & p_{1z} \\ p_{2x} & p_{2y} & p_{2z} \\ & \vdots & \\ p_{kx} & p_{ky} & p_{kz} \end{bmatrix}$$

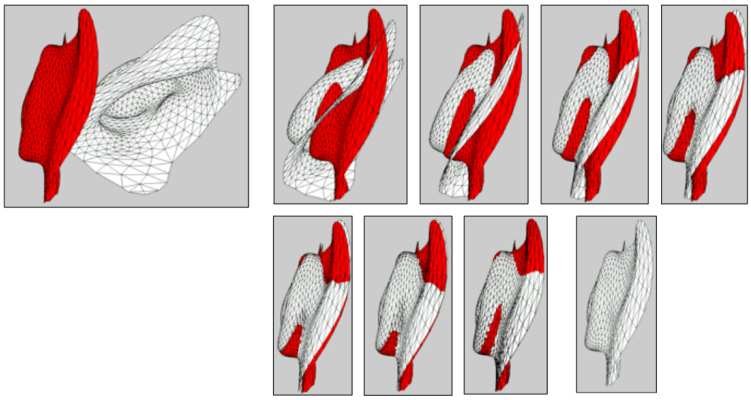

Figure 3: Example of ICP aligning two models. ((source)

To find the normal at each point, perform and eigenvalue decomposition of the covariance matrix of K, $cov(K)$. The normal is the eigenvector corresponding to the smallest eigenvalue of $cov(K)$.

Note that the direction of the normal is ambiguous: it could point in either direction relative to the plane. To get consistent normal directions across the model's surface we need some additional information. *Think: what do we know about the relative position of the object and camera?*

2.4 Building the model

We want to merge the object point clouds from each frame into a full model of the object. Starting with the point cloud for frame 1, we extract the point cloud from frame 2, and use ICP to align cloud 2 to cloud 1. This gives a slightly larger point cloud. We then merge frame 3 into that point cloud, and so on with the remaining of the frames.

Figure 4: Segment the point cloud into individual objects.

3 Part 2: Segmenting the Multi-Object Scenes (30 points)

For this part, you must segment point clouds containing multiple objects into individual objects. First reconstruct the full point cloud from the RGB-D frames, as you did in part 1. Then, segment the point cloud and identify each object. Visualize the point cloud with each segment color-coded and labled (with the name of that object).

You **cannot assume that you know the number of objects in the scene**, but you *can* assume that you will only encounter objects from part 1. You can use the 3D models you made in part 1 to aid in identifying the objects. You **can use the original RGB-D images** (for both the single object and multi-object scenes).

We describe one potential approach below, but you are free to experiment and be creative.

3.1 Segmentation with SIFT Matching and ICP

One approach is to find an object's location in 2D (i.e. the original RGB images), and use that as a starting point to fit the object model to the point cloud using ICP (or similar).

We'll use feature point matching (see Matlab's example) to detect an object's location in an image of the multi-object scene. This gives a bounding box in the image of the object's estimated location.

We can map that bounding box to a corresponding region in the point cloud. Recall from part 1 that for each RGB-D frame, we extracted a partial point cloud, and then found a rotation and translation which fit that point cloud to the first frame. Using this relationship, we can map any pixel in the RGB-D images into the point cloud.

One complication: the object's appearance can vary based on its orientation. Therefore, we need to run feature matching several times for several orientations of the object (i.e. different RGB-D frames). This also gives a rough estimate of the object's orientation in the target image. With an estimate of the object's location and orientation in the point cloud, we can run ICP and see if converges.

3.2 Extra Credit: Point-cloud only segmentation (30 pts)

Compute your segmentation using only the point clouds, and not the original RGB-D frames. So, for example, you may not detect the objects in 2D using SIFT matching. This is a difficult problem that you can approach in many ways. We recommend consulting the papers in `Reference Papers/Object Segmentation/` for ideas.

3.3 Extra Credit: Building a Semantic Map (30pts)

The goal for this task is to derive meaningful, useful information about the objects in the multi-object scenes. It could be as simple as relative sizes, orientations, positions, etc.; or as complicated as identifying object classes and material properties (e.g. softness). The more complicated and interesting semantic properties will likely require machine learning techniques, as used in the papers in `Reference Papers/Semantic/`.

To show your semantic map's usefulness, explain a particular application where it would be useful. For example, a robot fulfilling orders in an Amazon warehouse would need to know the weight, compliance, etc. of objects before it can pick them up properly.

This task is more open-ended than the segmentation task. Please send the TAs a quick email with your project idea (or stop by office hours) so they can approve it or give suggestions.

4 Stuff Given to You

The helper code along with the calibration parameters and the function to convert images to point clouds is in the `Code` folder.

The RGB-D datasets can be found at this Google Drive page. Note that there is about 2 GB of data, so it might take a few minutes for the files to compress on Google's servers before you can download them.

Make sure to check `Reference Papers` folders for papers regarding ICP (Standard ICP paper is the easiest but the worst performing) and Object Segmentation. `Dataset` folder contains the data of various scenes containing individual objects and multiple objects.

5 Allowed & Prohibited Functions

You are allowed to use third party code for basic stuff but not for the whole algorithm. You are only allowed to use `pcread, pcshow`, and `pcnormals` from the MATLAB's point cloud library. Functions like `KDTreeSearcher` and `knnsearch` can be used for point to point correspondence search. For segmentation, all of the functions in the linked Matlab example are allowed. You are also allowed to use any other basic functions built into the computer vision or image processing toolbox in MATLAB.

6 Submission Guidelines

Make a video of all the objects and all the segmented scenes where you pan the 3D point cloud to show it is fully reconstructed and is correctly segmented for each object. See the example video, `example_video.mp4` Write a thorough report detailing your results, implementation, any interesting problems you encountered and/or solved, what worked and what didn't, etc. You can include as many images/figures as you like, but keep the report under 2000 words. Feel free to add videos you feel are cool or paste YouTube links in your report. You should also include a detailed `README` file explaining how to run your code.

Submit your .fig files (of 3D point clouds for all individual objects), code (`.m` files) with the naming convention `lastname_finalproj.zip` onto ELMS/Canvas (**Please compress it to .zip and no other format**). (`lastname` can be the last name of anyone in your group.) Your zip file should have the following things (for EACH of the given scene):

- A video showing a full 3D reconstruction for each object
- A video showing a full 3D reconstruction and segmentation of each scene
- '.fig' files for 3D models of all individual objects and scenes (color coded with segmentation labels)
- Code used for this project with a detailed README file
- Your `report.pdf`

If your submission does not comply with the above guidelines, you'll be given **ZERO** credit.

REMINDER: Each student is required to present on at least one project from the semester. This includes final projects, which should be presented as a group.

7 Collaboration Policy

You can discuss the ideas with your peers. But the code you turn-in should be from your own team and you **MAY NOT USE** code from other students. For the full honor code for this course, refer to the CMSC426 Spring 2018 website.

DON'T FORGET TO HAVE FUN AND PLAY AROUND WITH ~~IMAGES~~ POINT CLOUDS!.

Acknowledgements

We would like to thank Aleksandrs Ecins for the dataset and for hints and tips, and Bhoram Lee (University of Pennsylvania) for help with the code. This fun project was inspired by Nitin Sanket's research and a course project at the University of Pennsylvania http://www.grasparche.com/.

References

[1] Kok-Lim Low. Linear least-squares optimization for point-to-plane icp surface registration, Feb 2004.