Building a semantic map of a 3D scene

Brian James Fitzgerald
School of Computer Science
University of Maryland

Gudjon Einar Magnusson
School of Computer Science
University of Maryland
gmagnusson@fc-md.umd.edu

Stephen Xie
School of Computer Science
University of Maryland

***Abstract*—In this paper we present an implementation of object segmentation and recognition using and RGBD camera.**

I. Introduction

Recognizing objects in a 3D dataset is useful for many things.

For this implementation we assume that the objects of interest are placed on a flat table surface and are located roughly in the center of the frame. We believe this is not an unreasonable assumption since it holds in many practical use cases, for example, robot manipulators at a conveyor belt.

The pipeline goes roughly as follows: first a point cloud is obtained from RGB-D images. The background and table are removed to isolate the object of interest, this is repeated for multiple views of the object. Next an object model is constructed by merging the different views using iterative closest point (ICP).

II. 3D Object Segmentation

A. Background 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. The center is defined as the mean of all points in the scene. Since the object is close to the center of the frame and closer to the camera than the rest of the scene, the point cloud is usually densest close to the object. Therefor the center of mass is usually close to the object even if the object is not in he center oft the 3D space.

To separate the object from the table it stands on we rely on the fact that points on the table are co-planar. We use the RANSAC algorithm to find the plane in the scene that contains the highest number of points. The algorithm picks 3 points in the scene at random and defines a plane that containing those points. Next check the distance of every other point to that plane, if the distance is below a tolerance the point is considered part of the plane. This is repeated for a number of iterations and the plane containing the highest number of points is assumed to be the table.

Despite our best efforts to remove the table and background some noise is likely to remain. To remove point cloud noise we use a metric based on a points mean distance to its neighbors. For every point we find it's k closest neighbors and calculate the mean distance. If the mean distance is above a threshold $s\sigma$, the point is considered to be noise. σ is the standard deviation of the mean distance and s is a configuration parameter.

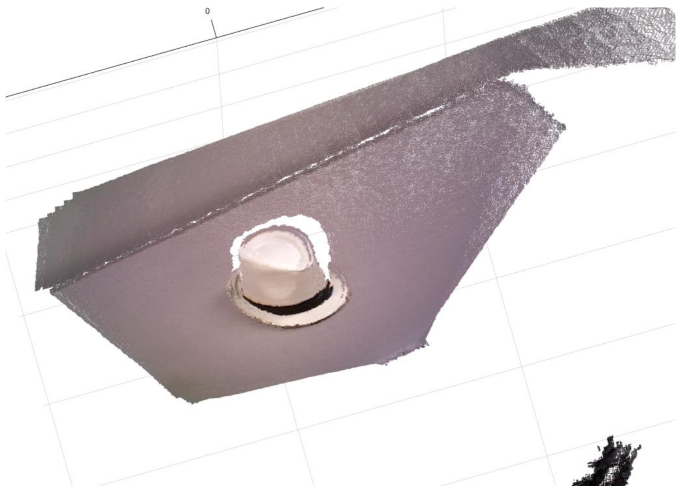

Fig. 1. Capturing the region of interest using a hand drawn polygon

B. Iterative Closest Point

ICP is an iterative algorithm that given two corresponding point sets will find a translation and rotation that minimizes the distance between corresponding points. For this implementation we use a variation of ICP that minimizes the distance of each point to a corresponding plane. This variation converges much faster.

$$E = \sum_{i=1}^{m} \left[(p_i - q_i) \cdot n_i + t \cdot n_i + r \cdot c_i \right]^2$$

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

$$x = \begin{bmatrix} \alpha \\ \beta \\ \gamma \\ t_x \\ t_y \\ t_z \end{bmatrix}$$

$$b = \sum_{i=1}^{m} \begin{bmatrix} c_{ix}(p_i - q_i) \cdot n_i \\ c_{iy}(p_i - q_i) \cdot n_i \\ c_{iz}(p_i - q_i) \cdot n_i \\ n_{ix}(p_i - q_i) \cdot n_i \\ n_{iy}(p_i - q_i) \cdot n_i \\ n_{iz}(p_i - q_i) \cdot n_i \end{bmatrix}$$

For point to plane ICP to work we need to estimate the normal of each point on the model. Normal estimation is done by fitting a plane through a small region of points and getting the normal of that plane. For each point we find the k closest neighbors, since these points are highly unlikely to be precisely co-planar we need to fit a plane in a least squares sense. To do this we form a matrix K out of the k points.

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

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

The direction of the normal is ubiquitous so in order to get consistent normal directions across the model surface we need some additional information. For a point cloud obtained by a depth camera we know that the normal should not be facing away from the camera. Assuming the camera is located at the origin we can treat the point coordinates as the view direction. For normal n at point p, n faces the camera if $n \cdot p < 0$, otherwise we flip the direction of n.

C. *Model Sub-Sampling*

Working with point clouds containing hundreds of thousands of points at full resolution can be extremely slow and offers little benefit. To speed up the processing we subsample the model before estimating normals and running ICP.

III. 3D Object Recognition

Given a scene containing one of more objects, we would like to recognize what those object are and where they are.

IV. Conclusion

The conclusion goes here.

Acknowledgment

The authors would like to thank...

References

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.