Point Cloud Segmentation

Brian James Fitzgerald
School of Computer Science
University of Maryland
bjfitzgerald13@gmail.com

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

Stephen Xie
School of Computer Science
University of Maryland
zilix@terpmail.umd.edu

***Abstract*—Given RGB-D frames of a scene, it is possible to extract and create a three-dimensional model of the "object of interest" using Iterative Closest Point (ICP). Once modeling and segmentation are achieved, the object may then be recognized in a separate scene consisting of multiple objects, where a semantic map is then constructed to describe the scene.**

I. Introduction

Recognizing objects in a 3D dataset is useful for many things, such as robotics, augmented reality and more. 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).

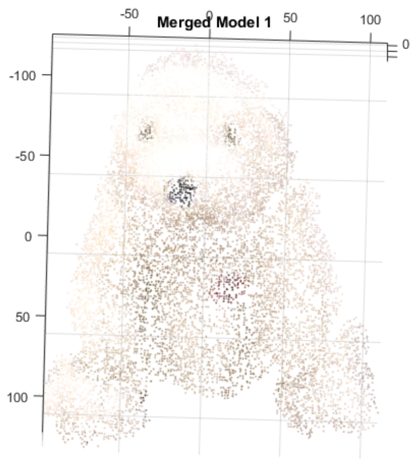

Fig. 1. Example of a single segmented object. The table and background has been removed

II. Single 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.

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.

The error of alignment for two sets of points p_i and q_i with normals n_i is defined as:

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

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

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

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

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

We define $c = p \times n$. t is the translation and α, β, γ is the rotation around the x, y, z axis.

C. *Estimating Surface Normals*

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 eigenvalue 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.

D. *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 and reduce it to only a few thousand points before running ICP. There are multiple ways to perform point cloud sub sampling, in this implementation we

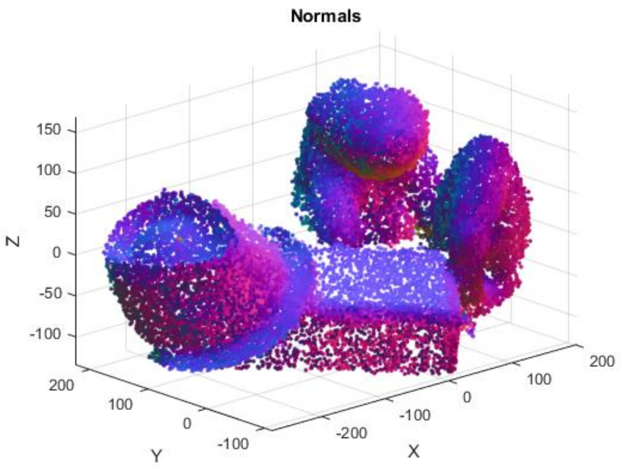

Fig. 2. A point cloud model draw using the normal values as color

III. Multi Object Segmentation

In the previous sections we consider segmenting under the assumption that there is a a single object sitting on a table. That acts as our training data that we use to collect object models. Now we consider the problem of segmenting a scene containing multiple objects where the number of objects in unknown. We still use the same method to remove the background and table but need to introduce a new method to separate the objects.

A. *Graph Flooding*

To label objects we treat the point cloud as a graph where voxels are nodes and are connected by an edge if they are within a small radius. Each edge is given a weight that represents the probability that the edge crosses an object boundary. The algorithm picks a point at random that has not been labeled yet and assigns a new label to it. For every edge to that node that has a weight lower than a threshold, the label is propagated along the edges to neighboring nodes. This is repeated in a breadth first pattern until we run out of unlabeled nodes or all reachable edges have a weight above the threshold.

B. *Edge Weights*

The quality of the object labeling is depended on the edge weights. In this implementation we tried two methods, cosine distance between normals and local convexity.

a) Cosine Distance: Using the cosine distance between normals for edge weights will in many cases correctly label boundaries but will also add many extra boundaries. It captures areas with similar normals but does not wrap around the full object. Figure 3 shows an example of labeling a point cloud using the cosine distance edge weights.

b) Local Convexity: The convexity metric is based on the observation that concave corners are frequently parts of an object boundary while convex corners unlikely to boundaries[1][2].

The weight w between two points p_i, p_j with normals n_i, n_j

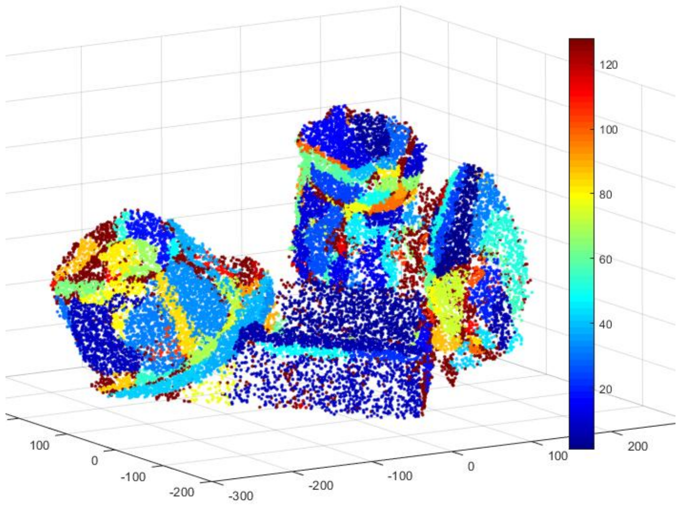

Fig. 3. Point cloud labeled using normal cosine distance for edge weights

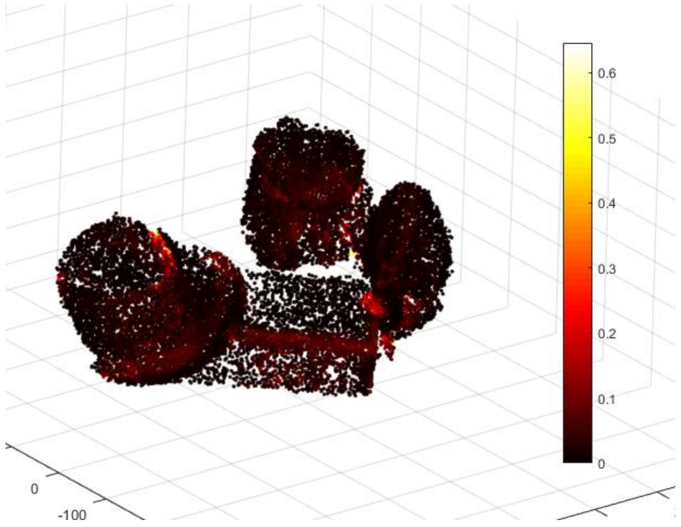

Fig. 4. Heat map showing the concavity metric on a model

$$w_{ij} \begin{cases} (1 - n_i \cdot n_j)^2 & \text{if } (p_j - p_i) \cdot n_j > 0 \\ 1 - n_i \cdot n_j & \text{otherwise} \end{cases}$$

Figure 4 shows the distribution of the concavity magnitude across a point cloud model. The magnitude is greatest on concave corners but due to noise in the normals and points on this model flat regions also light up. Some regions that we would like mark as a boundary do not light up.

References

[1] Simon Christoph Stein, Markus Schoeler, Jeremie Papon, Florentin Worgotter *Object Partitioning using Local Convexity*

[2] Andrej Karpathy, Stephen Miller, Li Fei-Fei *Object Discovery in 3D scenes via Shape Analysis*