Capstone 44: 3D Object Segmentation

S. Bardewa, C. Ferris, S. Hendrickson, J. Schiffer, M. Whiteside

***Abstract*—Due to the recent influx of cheap stereo depth cameras, robotic vision has become a popular topic of research. The availability of state-of-the-art open source algorithms has also made this field more accessible to a wide community, from scientists to hobbyists.**

This report summarizes our research in the field of robotic vision, particularly object segmentation and recognition using a 3D depth camera. Here we summarize the most important aspects, including the effects of the various parameters on performance, as well as some of the theoretical background used in object segmentation and recognition.

I. Introduction

The problem our team was given was to enable a humanoid robot to locate and identify a common object, such as a medicine bottle, from a table or floor, and guide the robot's hand to the target object.

Section II is an overview of the segmentation pipeline we implemented. In section III, we describe the two popular approaches to object recognition, global and local. Section IV describes the calculation of bounding boxes for the segmented objects. Testing and verification of our approach is presented in section V. Section VI includes possible applications of our package. Finally, section VII and VIII present future work and conclusions.

II. The Segmentation Pipeline

The basic architecture of our package, as well as many others in this field, is a sequence of processing stages, aka. a pipeline. The segmentation pipeline starts with capturing an image from a 3D depth camera, and by the last stage of the pipeline we obtain the location and boundary information of the objects of interest in the scene such as the hand of the robot and the nearest pickable object.

A. Downsampling

The raw clouds coming in from the camera have a resolution which is far too high for segmentation to be feasible in realtime. The basic technique for solving this problem is called 'voxel filtering', which entails compressing several nearby points into a single point. In other words, all points in some specified cubical region of volume will be combined into a single point. The parameter which controls the size of this volume element is called the leaf size. Fig. 1 shows an example of applying the voxel filter with several different leaf sizes. As the leaf size increases, the point cloud density decreases proportionally.

B. Plane/Prism Extraction and the RANSAC Algorithm

RANSAC is a quick method of finding mathematical models. In the case of a plane, the RANSAC method will create a virtual plane that is then rotated and translated throughout the scene looking for the plane with the data points that fit the model (aka., Inliers). The two parameters used are the threshold distance and the number of iterations. The greater the threshold, the thicker the plane can be. The more iteration RANSAC is allowed to do, the greater the probability of finding the plane with the most inliers. In Fig. 2, one can see what happens as the number of iterations is changed. The blue points represent the original data, the red points represent the plane inliers, and the magenta points represent the noise (aka. Outliers) remaining after prism extraction. As can be seen, the image on the left shows how the plane of the table was not found due to RANSAC not being given enough iterations. The image on the right shows the plane being found and the objects above the plane being properly segmented from the original data.

C. Clustering

The last stage in the segmentation process is Euclidean clustering. This process takes the down sampled point cloud, without the plane and its convex hull, and breaks it into clusters, with each cluster hopefully corresponding to one of the objects on the table. This is accomplished by first creating a kd-tree data structure, which stores the remaining points in the cloud in a way that can be searched efficiently, and then the cloud points are iterated over, with a radius search being performed for each point. Neighboring points within the threshold radius are then added to the current cluster, and marked as processed. This continues until all points in the cloud have been marked as processed. When the algorithm terminates, all the points in the cloud have been put into different segments.

III. The Object Recognition Pipeline

The goal of object recognition is to pick out some object of interest from the scene, in order to get the information about its position and orientation. For our research, we are interested in identifying the position of the robot hand at a particular instant of time so that the robot hand can be guided to pick up the segmented object. This requires having a model of the robot hand in order to compare it against the objects in the scene.

There are two standard approaches in 3D object recognition to compare the model and the scene: the 1global pipeline and local pipeline. As shown in Fig. 3, these two techniques achieve the goal of object comparison/matching through the implementation of two distinct sequences (pipelines) of feature

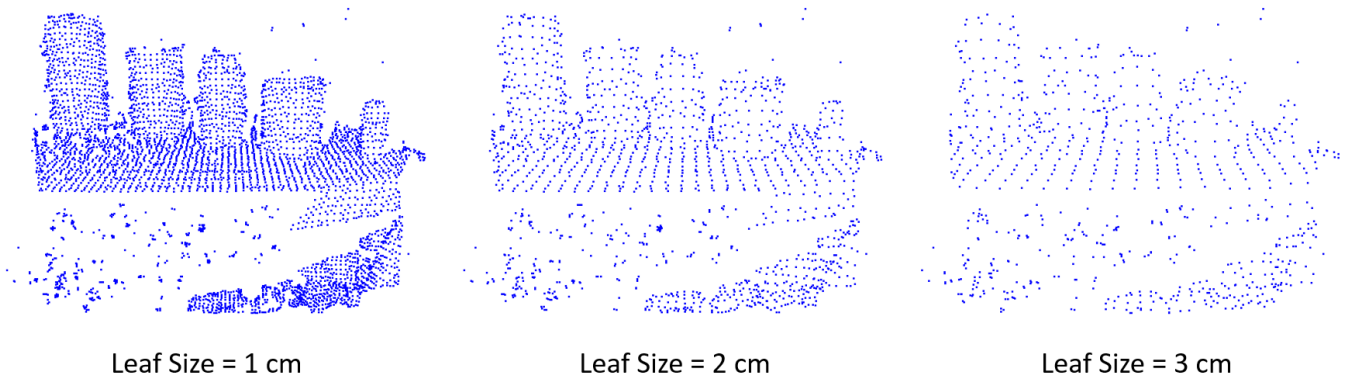

Fig. 1: Downsampling results for several different leaf sizes.

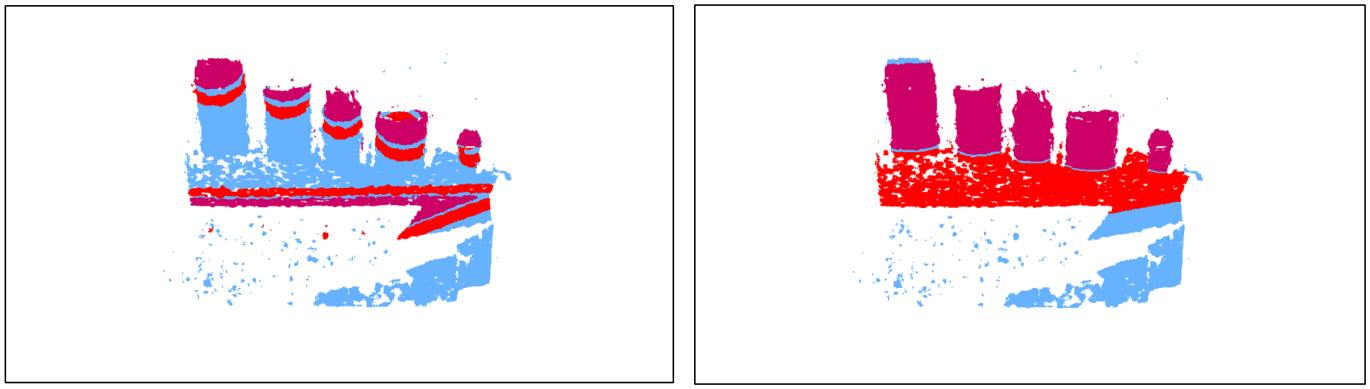

Fig. 2: The effects of varying the number of iterations of RANSAC. Notice that the plane on the left, which only used 200 iterations, was not correctly identified, while the one on the right, with 600 iterations was correctly identified.

recognition algorithms. Particularly, they use different descriptors to compare the objects. A descriptor encodes information on model and scene images to make it possible to compare them. Next, we give a description of the global and local approaches to object recognition .

A. *The Global Approach*

The global pipeline works by taking the object clusters from the earlier clustering step and computing a histogram for each of them. This histogram is then compared against a database of pre-calculated histograms for some object of interest, in our case, the robot's hand. These pre-calculated histograms are simply loaded at program boot-time. There are many histograms in the database for each object of interest because one needs to have data representing an object from many different camera positions.

There are several global descriptors that are available to store the data: Viewpoint Feature Histogram (VFH), Clustered Viewpoint Feature Histogram (CVFH), Oriented, Unique and Repeatable CVFH (OUR-CVFH), Ensemble of Shape Functions (ESF), and the Global Radius-based Surface Descriptor (GRSD). All of these variants were tested by our group, but we ended up choosing the OUR-CVFH descriptor, mainly because that was the approach favored by some of the original authors of PCL [1].

1) VFH: The VFH has a viewpoint dependent part, which stores data on the angles between the normal vectors of the points in the cloud and the vector from the camera viewpoint to the cluster centroid. Another portion of the histogram stores data on the distances of the points in the cluster to the clusters centroid. There are other components to the histogram as well, but the full details are somewhat beyond the scope of this handout, and the reader is referred to [2] for a detailed description.

2) CVFH: The Clustered VFH descriptor is similar to VFH except that it first divides the cluster into several regions using region-growing segmentation. An example would be a milk carton being divided into clusters corresponding to the sides of the carton. From here, VFHs are computed for each region. This offers an advantage over VFH, in that only one of the regions needs to be visible for the object to be identified. So, for example, if the desired object is occluded by some other object standing in front of it, the desired object can still be picked out based on the visible part.

3) OUR-CVFH: The Oriented, Unique, Repeatable CVFH descriptor extends CVFH by adding one additional processing stage after the region segmentation. It further filters the points

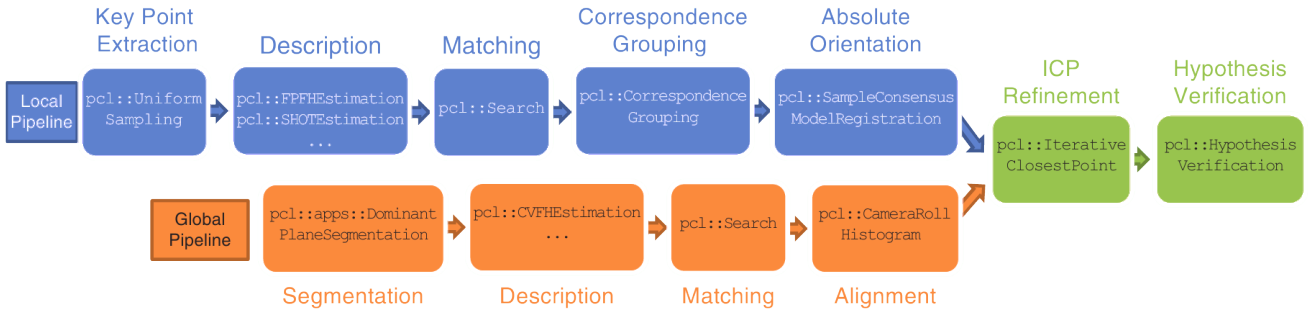

Fig. 3: The 2 approaches to object recognition: local pipeline (top) and global pipeline (bottom). Image taken from [3]

in each region according to their difference from the regions average normal, thereby refining knowledge of the regions shape. After this step, a Semi-Global Unique Reference Frame (SGURF), aka, a local coordinate system, is computed for each region. It is these reference frames which make up the content of the descriptor. The full detail of this process can be found in [1]. The result of these extra processing phases is an 82% detection rate which is a notable improvement over other global descriptors. [1]

B. The Local Approach

Local object recognition techniques extract key point information in the model and the scene images and associate them to a local 3D descriptor neighborhood [4]. Local keypoints provide information on the local geometric features such as image position and coverage area. A local descriptor compares the keypoints between the model and the scene. Unlike global descriptors that describe an entire object, local descriptors use multiple points in the image for comparison. Since the local feature recognition technique can operate on image patches, it gives promising recognition even for occluded and cluttered scenes in which the global feature recognition technique fails.

In order to quantify the performance of the local pipeline relative to the global pipeline, we evaluated the correspondence grouping and object alignment techniques which are based on local feature descriptors [5] [6]. In each of the pipelines, we first downsample the point cloud, and then take the keypoints associated with them. We then use the keypoints to create 3D local descriptors, namely Signature of Histograms of Orientations (SHOT) and Fast Point Feature Histogram (FPFH). After this, matching is performed to determine the presence of model instances in the scene. Although we were able to achieve high accuracy of object recognition using local pipelines, its computation time was higher than our execution time requirement, hence we abandoned this approach.

IV. Bounding Box

After the object segmentation and recognition has been performed, the robot knows which object to pick up, but it doesn't know the boundaries of the object. The boundary of the object is supplied by creating a rectangular bounding box around the object. This is the final stage in the task of segmenting a single frame of depth imagery. Next we present the steps taken in the bounding box calculation.

1. Compute mean value vector of the points in the cloud, call it $\overline{p}$

$$\overline{p} = \left(\frac{\sum p_x}{\text{points.size}}, \frac{\sum p_y}{\text{points.size}}, \frac{\sum p_z}{\text{points.size}} \right) \tag{1}$$

2. Compute the covariance matrix, M_{cov}

$$M_{\text{cov}} = \sum_{p \in \text{points}} \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} \begin{pmatrix} p_x & p_y & p_z \end{pmatrix} \tag{2}$$

$$= \sum_{p \in \text{points}} \begin{pmatrix} p_x p_x & p_x p_y & p_x p_z \\ p_x p_y & p_y p_y & p_y p_z \\ p_x p_z & p_y p_z & p_z p_z \end{pmatrix} \tag{3}$$

3. Solve for the eigenvectors and eigenvalues of the covariance matrix. The three eigenvectors — major, middle, and minor — then make up the columns, aka., the axes, of the rotation matrix

$$M_{\text{rot}} = \begin{pmatrix} \text{major}_x & \text{middle}_x & \text{minor}_x \\ \text{major}_y & \text{middle}_y & \text{minor}_y \\ \text{major}_z & \text{middle}_z & \text{minor}_z \end{pmatrix} \tag{4}$$

4. Compute the max and min points for the bounding box: for every point p in the cloud, compute the difference from the mean value vector, rotated by the rotation matrix, i.e.,

$$p' = M_{\text{rot}}^T (p - \overline{p}) \tag{5}$$

which is equivalent to,

$$p'_x = (p - \overline{p}) \cdot (\text{major\_axis}) \tag{6}$$

$$p'_y = (p - \overline{p}) \cdot (\text{middle\_axis}) \tag{7}$$

$$p'_z = (p - \overline{p}) \cdot (\text{minor\_axis}) \tag{8}$$

and then compare these to the current max/min point values. If they are greater or lesser respectively, then overwrite them with these new values.

5. Compute the midpoint between the max and min points. This is the shift vector.

$$\text{shift} = \frac{\text{max\_point} + \text{min\_point}}{2} \tag{9}$$

Adjust the max/min points by the shift, as well as setting the position vector to the center of the box

$$p_{\text{max}} = p_{\text{max}} - \text{shift} \tag{10}$$

$$p_{\text{min}} = p_{\text{min}} - \text{shift} \tag{11}$$

$$p_{\text{cm}} = \overline{p} + (M_{\text{rot}})(\text{shift}) \tag{12}$$

The bounding box is completely described by the rotation matrix, center of mass, and max and min points. This gives the robot everything it needs to pick the object of interest.

V. Testing

To test the performance of the segmentation pipeline, we first needed to find the effect of each function parameter on the execution time of the program. To do this, the program was run by only changing one parameter at a time over a range of values. As shown in Figure 4 (bottom-right), not all functions in the segmentation pipeline have an equal impact on the execution time. This led to some functions being optimized for performance over the resulting point cloud data.

The images in Figure 4 further show the results of each individual test. All parameters, with the exception of the one being tested, were held constant for each test. Also, the source point cloud data used was a raw unmodified image from the Realsense camera. For each graph, the dark line represents the execution time in comparison to the parameter being tested. The lighter (orange) line shows how many points are left in the resulting PCD file.

VI. Applications

There are several practical applications of object segmentation. Machine vision, medical imaging, human machine interaction, place recognition, object detection, video surveillance and manufacturing industry are some of the fields where our object segmentation and recognition package could be useful.

VII. Future Work

A. Accuracy Benchmarking

We also researched a method for measuring the accuracy of bounding boxes calculated by our segmentation pipeline. In this method an image is first analyzed without downsampling so that the resulting bounding box is regarded as an ideal model. Then, the segmentation parameters can be adjusted and the degradation in accuracy caused by these adjustments can be measured quantitatively.

The first step in the process is to find the best fit pairing between the corners of the cube. This is done by comparing, pointwise, all possible corner matchings, and calculating their distances

Once this best fit matching of the corners has been obtained, we can calculate the accuracy of a box with respect to the ideal box-model. This calculation was motivated with the following reasoning. A corner of the box is defined to be 100% accurate if it has the exact position of the corner in the box which is being compared, and the accuracy will fall off exponentially as the distance between the corners increases, reaching zero if the distance between them is infinity. So, it is reasonable to say that the accuracy will be halved for each unit of distance between the 2 corners. This distance unit should be chosen based on the context. For example, if the application is dealing with small objects, a distance unit of a few centimeters might be appropriate. Since there are 8 corners to compare between the 2 bounding boxes, the total accuracy will be the geometric mean of the product of the accuracies for the 8 corners.

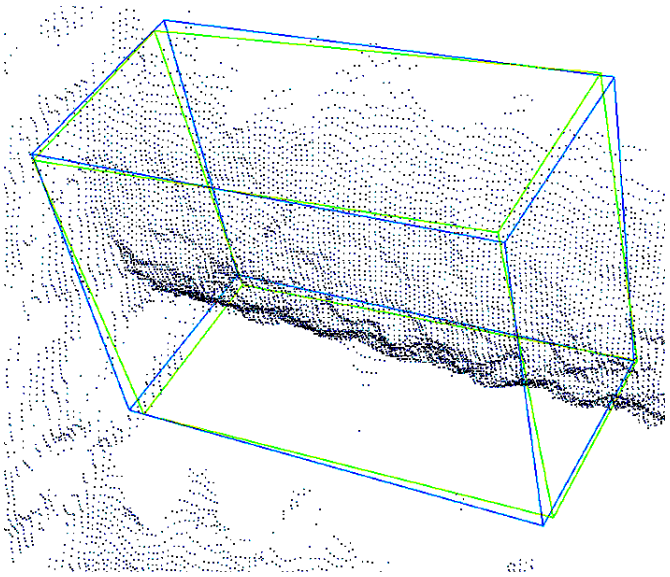

Fig. 5: Bounding box accuracy demo, showing two boxes which agree to 95% accuracy.

Symbolically, this is the quantitative definition of accuracy in the context of comparing boxes:

$$\text{accuracy} \equiv \sqrt[8]{\prod_{i=1}^{8} 2^{-sd_i}} \tag{13}$$

where d_i is the distance between the i'th corner in the first box, and the i'th corner in the second box, which is given by the standard Euclidean distance formula:

$$d_i = \sqrt{(x_{1_i} - x_{2_i})^2 + (y_{1_i} - y_{2_i})^2 + (z_{1_i} - z_{2_i})^2} \tag{14}$$

This algorithm has been implemented in our code base, but it took second priority to getting the segmentation pipeline working, and for that reason has not been extensively tested.

B. GPU Support

The next thing we could improve on would be to add GPU support, e.g., Nvidia's CUDA, which could greatly increase performance. GPU support comes as a part of PCL, but is disabled by default. Therefore, the only cost of adding this feature would be to determine the necessary compiler flags and changes to the build process.

C. Machine Learning

Another feature we would like to explore further is some kind of machine learning, aka, self-tuning capability. If the

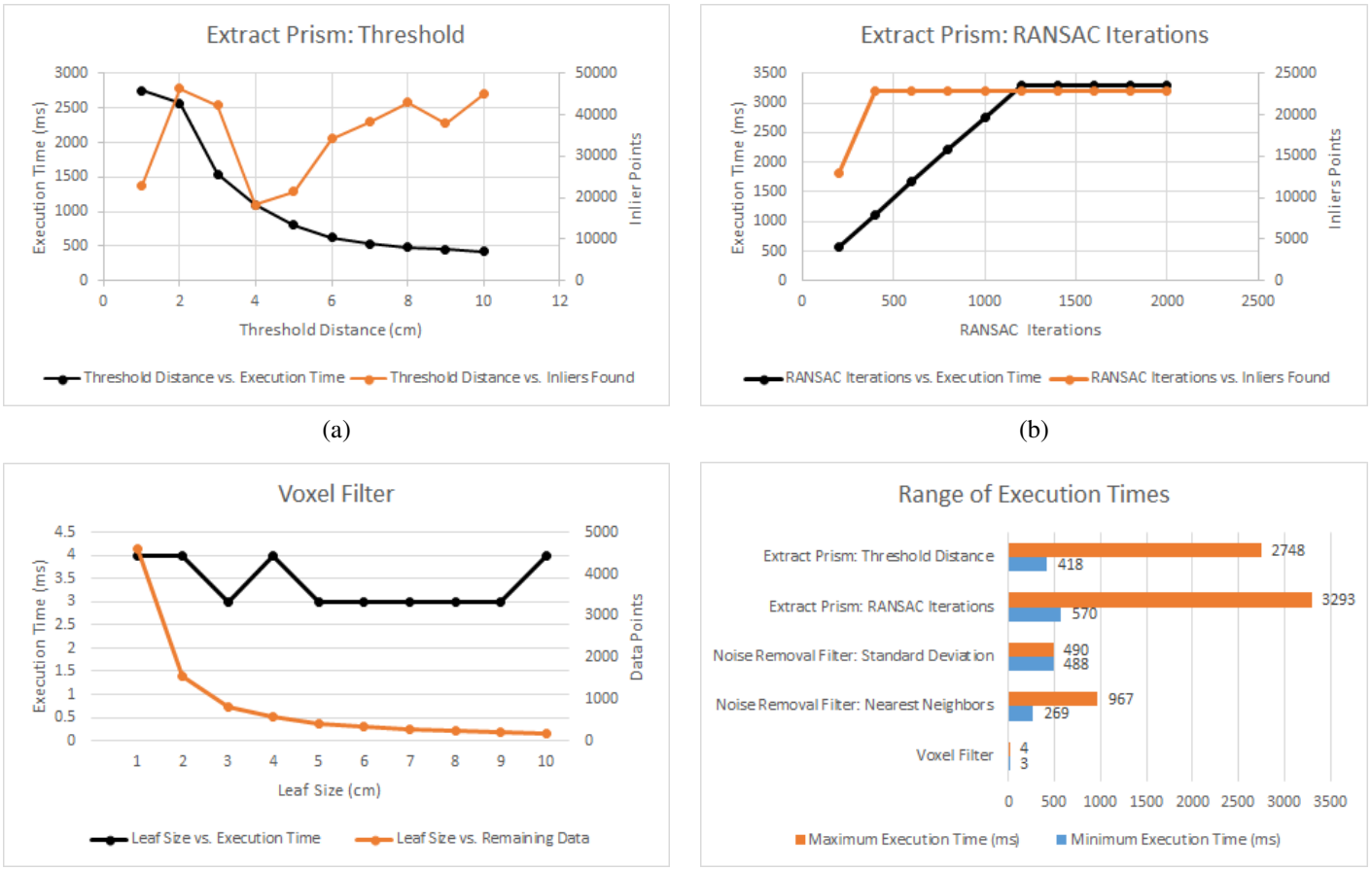

Fig. 4: Comparison of the effects of various parameters on segmentation pipeline performance

pipeline were able to adjust its parameters based on environmental conditions, it could be a lot more robust. For example, if the table surface was not very reflective, which tends to make plane extraction fraught with difficulty, the adaptive pipeline could increase the number of RANSAC iterations in order to correctly identify the plane. We have done some work implementing a genetic algorithm for this purpose; however, further work is necessary before it can be incorporated into the pipeline.

VIII. Conclusion

We were able to successfully implement an object segmentation pipeline and an object recognition pipeline, achieving a frame rate between 2-4 frames per second. We have provided our code as a ROS node which is freely available on github [8] under the BSD license, so that future users are free to build on our results.

References

[1] Aldoma, Aitor et al. "OUR-CVFH Oriented, Unique and Repeatable Clustered Viewpoint Feature Histogram for Object Recognition and 6DOF Pose Estimation." Computer Vision Laboratory, Dept. of Computer Science and Engineering, University of Bologna. Accessed June 2, 2016. http://vision.deis.unibo.it/fede/papers/dagm12.pdf

[2] "Cluster Recognition and 6DOF Pose Estimation using VFH descriptors." *pointclouds.org*. Perception Foundation. Accessed May 29, 2015. http://pointclouds.org/documentation/tutorials/vfh_recognition.php

[3] Aldoma, Aitor et al. "Three-Dimensional Object Recognition and 6 DoF Pose Estimation." *IEEE Robotics & Automation Magazine*, September 2012.

[4] Aldoma, Aitor, Federico Tombari, Luigi Di Stefano, and Markus Vincze, "A global hypotheses verification method for 3D object recognition.," In *Computer Vision-ECCV*, pp. 511-524. Springer Berlin Heidelberg, 2012.

[5] "Aligning Object Templates to a Point Cloud." - Point Cloud Library (PCL). Accessed May 30, 2016. http://pointclouds.org/documentation/tutorials/template_alignment.php.

[6] Aldoma, Aitor, and Federico Tombari. "3D Object Recognition Based on Correspondence Grouping." - Point Cloud Library (PCL). Accessed May 30, 2016. http://www.pointclouds.org/documentation /tutorials/correspondence_grouping.php.

[7] "PCL/OpenNI tutorial 5: 3D object recognition (pipeline)." Robotics Group of the University of Leon. Updated Nov 1, 2015. Accessed May 29, 2015. http://robotica.unileon.es/mediawiki/index.php /PCL/OpenNI_tutorial_5:_3D_object_recognition_(pipeline).

[8] "Object Vision 3D" source code: https://github.com/j1mb0/object_vision_3D