Koç University

3D Object Recognition

Final Project Report for Computer Vision Course

Doğancan Kebüde
11.01.2017

Contents

1. Introduction

There are various devices that can provide sensory information to a robot regarding their surroundings, addressing the simultaneous localization and mapping (SLAM) problem. However, none of them are as efficient as time-of-flight based sensors such as Microsoft's Kinect. GPS works only outdoors and can cause errors of ~8 m [1], LIDAR and acoustic ranging devices (e.g. SONAR) work only in 2D while stereo cameras are expensive and it is comparably hard to reconstruct 3D information from them. Time-of-flight based sensors, on the other hand, can provide both color and depth (3D) information in either RGB-D or Point Cloud format quite fast with high resolution. The information gathered through them is very similar to the human eye and can help improve solutions to SLAM problem.

It can be said that a robot can "see" with a time-of-flight based sensor. Their "understanding" of what they see, however, is another problem. For that reason, this work is trying to achieve 3D object recognition through point cloud data. It is important that a robot can recognize an object to complete certain tasks, even something as simple as identifying a mug and grasping that among other objects.

3D object recognition is a very new area compared to its 2D counterpart. However, there are tools to work with point clouds (data gathered through 3D sensors) such as Willow Garage's open source PCL (Point Cloud Library). PCL provides us with tools to address several problems regarding point clouds, such as I/O, feature estimation, segmentation, classification, search, visualization and even recognition. Using their provided tutorials and library functions, addressing 3D vision related problems become comparably easier.

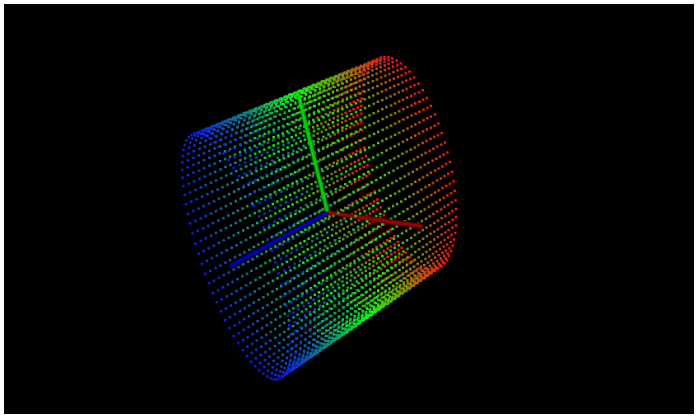

Figure 1. A 3D Point Cloud.

In this work, PCL's "Cluster recognition using VFH descriptors" algorithm [2] is vastly utilized. VFH stands for Viewpoint Feature Histogram and it is a very powerful descriptor. Using VFH signatures of point cloud objects, a K-d tree nearest neighborhood search among a data set will be conducted and possible *candidates* of what the query object can be, will be shown to the user. The methodology will be further explained in the chapter 5.

The remainder of this report is constructed as follows: Chapter 2 will introduce related background information regarding this project, chapter 3 will explain the problem specification, chapter 4 will investigate the problem, chapter 5 will describe the proposed solution to the problem, chapter 6 will talk about how the solution was implemented and how it was tested, chapter 7 will evaluate project performance and chapter 8 will conclude the report and talk about future work.

2. Background

This chapter provides background information regarding 3D object recognition scheme implemented in this work. The information that will be investigated in the following sub-chapters will be related to VFH descriptor, Chi-Square Distance, K-d Tree Structure, the data set used in the project, FLANN and PCL libraries.

2.1. VFH Descriptor

VFH is a powerful feature descriptor that works on big (more than 100 points in the cloud) object clusters [3]. There are 308-bins in each descriptor which provide information regarding three major components:

1. Three normal deviation angles between the centroid and points in the cloud (binned to 45 bins for each deviation angle -> 135-bins)
2. The distance between the centroid and points in the cloud (binned to 45 bins)
3. The angles between viewpoint and the point normals (128-bins)

First one of these components is actually named FPFH (fast point feature histogram) [4]; adding the second component, the descriptor becomes extended FPFH and with the addition of the third component, VFH is developed.

The main differences and advantages of VFH against FPFH are that FPFH is calculated for each point in the point cloud while VFH is calculated for the whole point cloud cluster and thanks to viewpoint component VFH becomes scale invariant (since the viewpoint component is providing us with the scale/depth ratio).

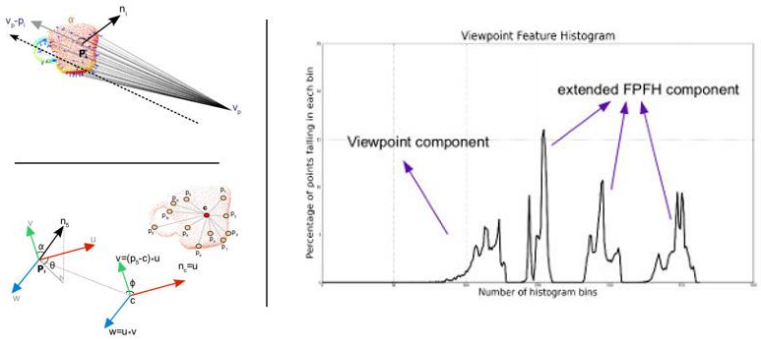

Figure 2. (Upper left) How viewpoint bins are calculated, (bottom left) how extended FPFH bins are calculated, (right) resulting VFH.

2.2. Chi-Square Distance

Chi-square distance is a weighted Euclidean distance measure that is used to calculate similarities between histograms. The chi square distance is calculated as follows in this work:

$$\sum_{j=1}^{n} \frac{(p_i - p_j)^2}{p_i + p_j}$$

Where p_i is the VFH of interest, and p_j is the other histograms. Therefore, a distance metric of a VFH to all other VFHs is constructed using this equation.

2.3. K-d Tree Clustering

K dimensional tree is a data structure for organizing data in a k dimensional space. A K-d tree is basically a binary decision tree that helps with data clustering and search. Through a K-d tree, nearest neighborhood search becomes easier. In this work, a K-d tree is utilized so that we can find the most similar objects to the query object of user's choice. The query object is given to the K-d tree and most similar candidates are found through K-d tree search.

2.4. Data Set

Unfortunately, a data set for object recognition purposes could not be gathered. Instead, the PCL data set [5] specifically gathered for object recognition through VFH was used. The point cloud object clusters in the data set consists of everyday kitchenware objects such as wine glasses, coffee mugs, measuring cups etc. The dataset also includes different sizes of cardboard boxes (figure 3). The dataset containstwo different point cloud formats, (1) the original point cloud format that VFH estimation can be conducted on and (2) VFH format which can be used in the algorithm directly. The designed algorithm can work with both formats. There are 195 object clusters in the data set, containing different objects and their poses in different orientations.

Figure 3. Objects (left) and their point cloud counterpart (right).

2.5. Libraries

For the purpose of object recognition two main libraries' capabilities were utilized:

2.5.1. FLANN

Fast Library for Approximate Nearest Neighbors (FLANN) [6] is a library that provides functions to utilize very efficient and fast nearest neighborhood approximations. The K-d tree clustering in this work is achieved through FLANN's capabilities.

2.5.2. PCL

Point Cloud Library (PCL) [7] is an open source library with an enormous community worldwide. PCL provides library functions which help with point cloud processing. In this work, their I/O [8], feature estimation [9] and visualization [10] classes are vastly utilized.

Also, the functions they provide in their tutorials were re-used for achieving efficient object recognition. The same tutorials helped with addressing VFH estimation [11] and object recognition problems [2].

3. Problem Specification

The original intention of this project was to achieve object recognition through RGB-D data instead of point clouds. In the project proposal, it was specified that object recognition would have been done through depth map analysis and segmentation [12].

However, since it was comparably harder to find depth map data sets than point clouds, the project had to be migrated to a point cloud data set as explained in 2.4. The usage of this data set caused a change of the methodology as well. Instead of Triantafyllidis et al.'s algorithm [13], Rusu et al.'s algorithm [3] was chosen.

Furthermore, during the background readings, it was understood that point cloud information can be better processed using the point cloud library through C++ since MATLAB's point cloud processing capabilities is not as developed as it is in PCL. Therefore, the project language was changed from MATLAB to C++ for a better point cloud processing.

4. Problem Analysis

In the scope of this work, the goal is to come up with a mechanism that can help us find candidates about what the query object can be. Thus, the recognition problem can be formulated as a nearest neighborhood estimation problem as told in [2]. To achieve nearest neighborhood estimation, the sub-problems are as follows:

1. **Training:**
    a. The feature estimation for each object in the training set should be achieved.
    b. The estimated features should be clustered in a K-d tree.
2. **Testing:**
    a. Estimate the feature of a given query object.
    b. Use the estimated feature to search for similar candidates in the K-d tree.

5. Solution Design

After the feature descriptor and the data set is chosen as shown in chapters 2.1. and 2.4, the solution becomes straight-forward: There are three basic steps to implement for the training phase and two basic steps to implement at the testing phase. These steps are as below:

1. To estimate VFH features from the training set (which is the whole data set).
2. To calculate chi-squared distances between VFHs.
3. To feed the distances and features for indexing in the K-d tree.
4. To estimate VFH feature from the query object.
5. To search for similar candidates in the K-d tree.

To implement this solution, PCL and FLANN capabilities were utilized and some functions from PCL tutorials were re-used after modification for relevant usage as explained in chapter 6.

6. Implementation and Testing

The implementation and testing were done back-to-back for each basic unit of the solution (i.e. first the feature estimator was implemented and then it got tested, then the K-d tree was implemented and it was tested, later the query object was given and search was tested etc.). Therefore, the implementation will be explained step by step as the testing and verification was done back to back with each step's implementation.

6.1. vfh_model Data Type

A vfh_model type definition is made at the beginning of the code. This type is actually an std::pair that holds a string as its first member and a float vector as its second member. The string will later hold VFH data file name and the float vector will hold VFH data in it.

6.2. Feature Estimator

VFH estimator was taken as it is in the VFH estimation tutorial [11] and re-used with the addition of normal estimation of point cloud. The estimation steps of the estimator are as follows:

- Read point cloud information from *.pcd file.
- Compute normals for the point cloud.
- Pass 3D location and normal information to the VFH Estimation function of PCL.
- Estimate VFH for the point cloud.
- Write VFH signature back as a point cloud.

The verification of feature estimator was achieved through the data set samples. Data set had both point cloud format files and VFH signatures for the same data sample. The files in the point cloud format was passed through the feature estimator and the sample's VFH signature was compared with the feature estimator result by calculating the difference between the two. The result was always zero, verifying that the feature estimator is working properly.

6.3. VFH Checker

This unit checks if there is a VFH signature in the *.pcd that is being tried to be loaded. This function achieves this through reading the file header of the *.pcd file. If there is a "vfh" field in the file header, it returns true and if there is no "vfh" part in the file header, it returns false.

VFH Checker was verified through *.pcd files that does and does not contain VFH signatures and checking the results printed in the console.

6.4. Data Loader

Loading of the data was achieved through Boost library's filesystem class. Through Boost library's functionality, the program searches through the folders in the given data set directory. If it finds *.pcd files, it first runs VFH Checker to see if the file contains a VFH signature, if so, it passes that data to the Histogram Loader. If there is no VFH signature in the file that is being tried to be loaded, it passes that file to the Feature Estimator to estimate its VFH and it creates a VFH signature file to later be loaded by the Histogram Loader. After histogram information is gathered from Histogram Loader than that information is pushed inside a vfh_model vector.

The verification of data loader was achieved through printing the found *.pcd file names in the console. The printed result in the data was compared with the folders and verified that Boost library can reach every file in the folder successfully. Furthermore, *.pcd files without VFH signature was passed to the Data Loader and it was seen that the loader estimates cloud's VFH signature properly and then finds the VFH signature file to print on the screen.

The Data Loader is another function re-use from PCL tutorials [2]. The part that passes the data through the VFH Checker and passing the data to the Feature Estimator if it does not contain a VFH signature was implemented by this work.

6.5. Histogram Loader

This function loads the histogram data into the float vector inside vfh_model.second and saves *.pcd file name that contains the VFH signature as a string inside vfh_model.first. That vfh_model is later pushed into a vfh_model vector by the Data Loader.

The verification of Histogram Loader was done by printing vfh_model information on the screen and comparing them with the file names and VFH histogram values.

Histogram Loader is a function re-use from the PCL tutorials [2].

6.6. Nearest K Search

This function searches for specified number of similar candidates to a query vfh_model inside a FLANN K-d tree. Then it passes the K-d tree indices for each candidate and distances between the query vfh_model and candidate models as output.

The verification was done by checking the end result of the whole program.

Nearest K Search is a function re-use from the PCL tutorials [2].

6.7. Visualizer

This function visualizes the k found candidates in a window with their distances to the query vfh_model on the left bottom corner.

The verification was done by checking the end result of the whole program.

Visualizer is a code re-use from the PCL tutorials [2].

6.8. Main Program

The main program flow is as follows:

- Get data set location, query object file, candidate count and distance threshold from the user.
- Check if the query object has VFH signature, if not, estimate its VFH and pass it to a vfh_model (VFH Checker+ VFH Estimator)
- Use data loader to load VFH signatures into vfh_model vector (Data Loader + VFH Checker + VFH Estimator + Histogram Loader)
- Use FLANN library to estimate Chi-square distances and create a K-d tree of the vfh_model vector.
- Search for similar candidates in the K-d tree (Nearest K Search)
- Call visualizer to show candidates on the user's screen. (Visualizer)

The verification of results is achieved through running the training samples through the algorithm as query objects and looking for only one candidate. The algorithm resulted in finding the same object from the K-d tree. Then, the training samples were run through the algorithm as query objects again and nine candidates were searched for this time. The algorithm resulted in finding the same object with different poses. Furthermore, some unrelated objects were passed to the algorithm and the resulting candidates were the ones looking very similar to the query objects.

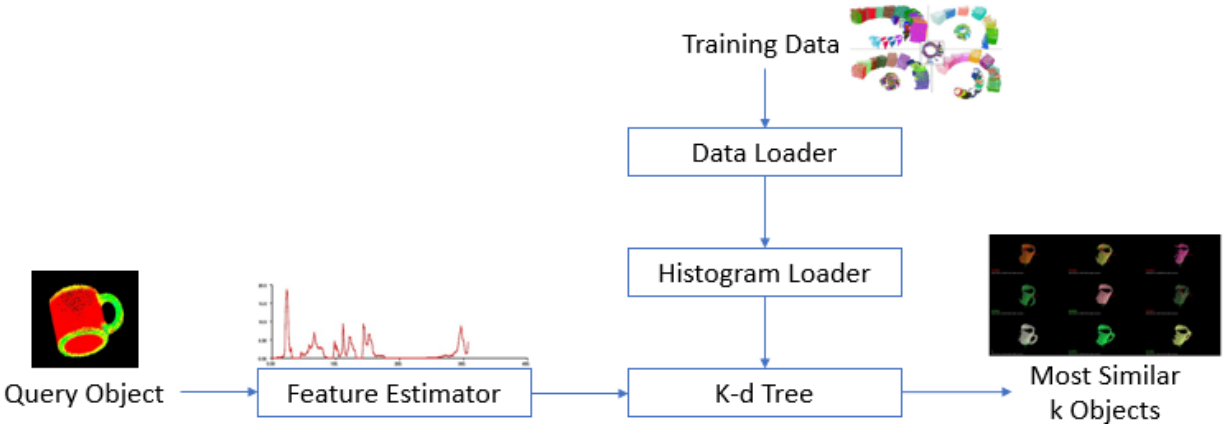

Figure 4. The general program flow.

7. Performance Evaluation

The project works as it was intended and explained in chapters 3, 4 and 5. Furthermore, as shown in chapter 6, every basic unit of the algorithm was verified with different methods and the end result was checked by using different verification methods.

The implemented object recognition scheme seems to be working with a data set that consists of objects in different poses with each object cluster having more than 100 points in their cloud. A different data set that suits these criteria could not have been found, and thus the algorithm could not have been tested on another data set. For the algorithm to work, the used data set should agree with these criteria.

8. Conclusion and Future Work

The algorithm in this project proved to work very well if a data set with different objects in different poses is provided. The end product of this project could be used as a robot decision making scheme with some future work: For example, if a crowded scene of objects was segmented into different objects and each object in the scene was run through this algorithm, a robot could decide which object to reach and grab among others. To achieve this, a scene segmentation algorithm such as Euclidean Cluster Extraction [14] could be used to get different objects from a scene.

References

[1] *GPS Accuracy.* (n.d.). Retrieved 01 11, 2017, from GPS.gov: http://www.gps.gov/systems/gps/performance/accuracy/

[2] PCL. (n.d.). *Cluster Recognition and 6DOF Pose Estimation using VFH descriptors.* Retrieved 01 11, 2017, from pointclouds.org: http://pointclouds.org/documentation/tutorials/vfh_recognition.php#vfh-recognition

[3] Rusu, R. B., Bradski, G., Thibaux, R., & Hsu, J. (2010). Fast 3D Recognition and Pose Using the Viewpoint Feature Histogram. *International Conference on Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ .* Taipei, Taiwan: IEEE.

[4] PCL. (n.d.). *Fast Point Feature Histograms (FPFH) descriptors.* Retrieved 01 11, 2017, from pointclouds.org: http://pointclouds.org/documentation/tutorials/fpfh_estimation.php#fpfh-estimation

[5] PCL. (n.d.). *VFH Recognition Tutorial Data.* Retrieved 01 11, 2017, from Point Cloud Library: https://github.com/PointCloudLibrary/data/tree/master/tutorials/vfh_recognition

[6] Muja, M. (n.d.). *Fast Library for Approximate Nearest Neighbors.* Retrieved 01 11, 2017, from UBC: http://www.cs.ubc.ca/research/flann/

[7] PCL. (n.d.). *What is PCL?* Retrieved 01 11, 2017, from pointclouds.org: http://pointclouds.org/about/

[8] PCL. (n.d.). *pcl::io Namespace Reference.* Retrieved 01 11, 2017, from Point Cloud Library (PCL): http://docs.pointclouds.org/trunk/namespacepcl_1_1io.html

[9] PCL. (n.d.). *pcl::feature Class Template Reference.* Retrieved 01 11, 2017, from Point Cloud Library: http://docs.pointclouds.org/trunk/classpcl_1_1_feature.html

[10] PCL. (n.d.). *pcl::visualization Namespace Reference.* Retrieved 01 11, 2017, from Point Cloud Library (PCL): http://docs.pointclouds.org/trunk/namespacepcl_1_1visualization.html

[11] PCL. (n.d.). *Estimating VFH signatures for a set of points.* Retrieved 01 11, 2017, from pointclouds.org: http://pointclouds.org/documentation/tutorials/vfh_estimation.php#vfh

[12] Kebude, D. (2016). *Object Recognition with RGB-D Sensors: A Project Proposal for Computer Vision Course.* Istanbul.

[13] Triantafyllidis, G., Dimitriou, M., Kounalakis, T., & Vidakis, N. (2013). Detection and Classification of Multiple Objects using an RGB-D Sensor and Linear Spatial Pyramid Matching. *Electronic Letters on Computer Vision and Image Analysis (ELCVIA), 12*(No: 2, Special Issue on Recent Applications on Computer Vision & Pattern Recognition), 78-87.

[14] PCL. (n.d.). *Euclidean Cluster Extraction.* Retrieved 01 11, 2017, from pointclouds.org: http://www.pointclouds.org/documentation/tutorials/cluster_extraction.php