Object Segmentation with PCL and ROS for Robotic Vision

George Paul Bora,Zoltan-Csaba Marton

Abstract—**Autonomous robots have long been a goal of scientists and engineers everywhere and as more advances are being made in the field it becomes clear that a major part of that autonomy will be the vision which the robots can employ to discover their environment. In the field of robotic vision one of the most important operations is segmentation, i.e the separation of parts of the images which represent objects from the overall image, in our case point clouds, these images can be then classified or be used in a number of other 3D vision applications the robot needs to apply to it as to function autonomously. This paper presents one possible implementation of the segmentation process, using the Robot Operating System (ROS) software platform for integration with the robot and the Point Cloud Library (PCL) for the image processing algorithm.**

I. Introduction

Recent advances in the acquisition of images of a more detailed nature then simple 2D, i.e. 3D at a minimum, of which we can name the products of Asus and very notably Microsoft's Kinect have made large amounts of 3D data much more readily available which means that the algorithms which process this data in more efficient and meaningful ways need to be created.

A. PCL

For this application we are using PCL [1], a open source library licensed under the Berkeley Software Distribution (BSD) free for both commercial and research use which provides state of the art implementations of algorithms from the field of computer vision. These algorithms are structured into multiple smaller libraries which offer the basic building blocks for problems like point cloud registration, segmentation, object recognition, surface reconstruction, normals estimation etc. Along with these there are a number of support application mainly for the visualization of point clouds, mapping and object recognition. The version of the PCL API used in this project is the one which comes with the current version of ROS i.e. PCL 1.1, which is not the most up to date version of PCL but has all that we require for the segmentation process.

B. ROS

The other software platform used in this application is The Robot Operating System,here after referred to as ROS [2], it is a software framework which provides operating system like facilities for robotics projects. It was initially developed by Standford Artificial Intelligence Laboratory and has since passed into the care of Willow Garage, a robotics research incubator and the same team which is also maintaining PCL, ROS is also licensed under the BSD thus free for commercial and research use.

ROS is organized into several higher level libraries which are named stacks, and can be installed independently, which are further split into smaller bundles called packages which can contain executables, libraries and tools. The application presented in this paper uses the current version ros-electric.

II. 3D Image Processing

The data gathered by 3D image sensors such as Kinect[1] , SICK LMS 400 [2] and other Time-Of-Flight(TOF) systems, triangulation or structured light sensors, even if they're not really used in robotics applications, first needs to be stored in a way it can be processed.

The most primitive representation unit for data in a 3D space is the point itself, which although in theory may be a N-dimensional tuple always contains at the x, y and z coordinates which are expressed in relation to a coordinate system which is most often positioned at the base of the sensor.

These points are virtually only ever used as part of a larger group which we will henceforth refer to as a point cloud. That data structure in the PCL for this is, as the name would imply, the point cloud which can be used to represent all the data your sensor can register, most importantly not being limited to the mere 3 floats which represent the points dimension along the X, Y and Z axes, those are just the bare minimum needed for it to be a point cloud and most point cloud structures expand on that.

A. Point Cloud Types

PointCloud <PointT> represents the base class in PCL for storing collections of 3D points. The class is templated, which means we don't actually use it as is in application rather we template it to a more specific point type. None the less this is where the basic elements which all point clouds share are defined, and they are:

- width - specifies the width of the point cloud dataset in the number of points, it can have two meanings if the point cloud is organized, i.e. you can arrange it as a matrix, then it is the number of points in a row or if

[1]Kinect is a motion sensing input device developed by Microsoft, intended to work with the Xbox 360 game console and Windows PC's. It is capable of acquiring 3D point clouds.

[2]The SICK LMS 400 is a laser positioning system capable of acquiring point clouds and providing additional reflectivity information

the point cloud is not organized it is the total number of points.

- height - specifies the height of the point cloud dataset in the number of points, it also has two meanings if the point cloud is organized it is the height in number of points or the number of rows if you prefer, for unorganized data sets it is always 1.
- points - the data array where all points of type PointT are stored.
- sensor origin - specifies the sensor acquisition pose i.e. the origin and translation, this is optional.
- sensor orientation - specifies the sensor acquisition i.e. the rotation.
- is dense - specifies if all the data in points is finite if (true), or whether it might contain Inf/NaN values if (false).

PointCloud <PointXYZ> is the point cloud template we use in this application at least in the PCL dependent parts,

It is a more basic template when compared to the others in PCL, but since we need to template the ROS messages into a PCL point cloud type and our segmentation process doesn't require anything more then the XYZ coordinates it will suffice.

PointCloud2 is actually quite different in implementation from the others, it fulfills the same basic purpose but it is actually a message type defined in the std_msgs package of ROS and thus has a different internal implementation.

PointCloud2 is used in the ROS dependent parts of the application, the important thing to us is that we can easily convert to and from PointCloud <PointXYZ> so that the result which we obtain as PCL types can be sent further along the processing pipeline i.e in the ROS environment.

B. Grids and RANSAC

In the application we will use PCL's implementations of several of the most important techniques in 3D image processing, ahead we will describe them in brief details:

- Voxel Grid - a voxel is the expansion to 3D dimensions of the 2D pixel,the voxel grid is super imposed over the point cloud, and all the points which are within 1 voxel are merged into a single point, positioned directly in the center of the voxel.thus the voxel grid is invaluable for down sizing point clouds which is essential for keeping processing times down.
- SAC Segmentation - PCL has a number of SAC classes implemented for different generic shapes, plane cylinder sphere, with which we can fit the points to a mathematical model and then extract those points which respect the conditions which we've imposed i.e we've segmented points which represent objects out of the overall point cloud. SAC is a shortening of the RANSAC[3] algorithm which itself is a acronym for "RANdome SAmple Consensus" a non deterministic iterative algorithms which given data with included outliers returns the parameters of a model which can explain the data and implicitly point out the outliers.

C. Inliers and Outliers

In statistics an outlier is a observation which is numerically distant from the other data gathered, in the field of computer vision we can interpret this as points which have their dimensions, usually the x, y and z coordinates distant from the other points, thus:

- outliers - points which have dimensions so far from the model we're trying to fit the points to, for example planes spheres etc that we simply accept they can't be fitted into a model with most of the other points.
- inliers - points which have dimensions which we can easily fit into a model with most of the other points from the cloud.

III. ROS Overview

As said before ROS is despite it's name not a operating system but a software platform which aims to make as much of the code robotics researchers write re-usable in other projects by employing a set of standards and design goals toward this ends.

In the paper we will make use of several ROS concepts and thus need to explain the nomenclature.

A. Nodes

Nodes are the processes which perform the computation within the ROS environment.

Due to the goal of supporting the language of the developers choice ROS allows nodes to be written 4 popular and very different languages for example C++, in which the nodes of this application were written in Python, Octave, and LISP.

Ports for other languages are in development and with the exception of Octave the ROS ports are written in the respective languages.

To combat the integration of algorithms with the middleware, thus limiting re-usability, developers are encouraged to write the nodes themselves as light as possible with the actual implementation of algorithms being in stand alone libraries.

We use the term node for these software modules as when they are best viewed when collaborating in the ROS environment as nodes within a directed graph.

The simplest graph of this sort is a pipeline nodes linked one after the other with messages, that is also the case in our implementation.

B. Messages

One of the ways nodes communicate with each other is through the use of messages.

Within ROS messages are strictly typed data structures, standard primitive types, integers, floats, strings etc are supported as well as arrays of primitive types and arrays of messages.

A important message type to note is PointCloud2 which is defined in ROS and to maintain re-usability should not be used in algorithm libraries.

C. Topics

Nodes interested in receiving or sending messages can subscribe or publish to a certain topic.

Messages published by a node on a topic is then broadcasted to all the nodes which have signed up as subscribers to that topic.

Relationships between nodes are peer to peer, in theory no one node is more important than the rest or has more privileges thus nodes have no knowledge of how many nodes are subscribed or publishing to any one topic.

D. Services

The broadcast model is not a universal solution to all communication processes in the ROS environment.

Sometimes we need a synchronous approach and not the asynchronous one topics rely upon, in this care we define a service which can be easily understood as analogous to web services.

They are defined by a string name and a pair of strictly typed messages, a request and a response, unlike topics only one node can advertise a service of a particular name.

IV. A Brief View of Object Segmentation

The approach presented in this paper is of course not the only method available for segmentation in the field of 3D vision.There are a multitude of methods one can use for segmentation ranging from the simple thresholding method, which splits a image into two parts based on a threshold to methods based on clustering[4] and even more more sophisticated methods a for example those based on partial differential equations as those proposed in this paper[5] for segmenting tomographies .

For example the method developed by Mårten Björkman and Danica Kragic [6] is a method which exploits multiple cues (colors, positions and disparities) and then integrates them into a temporal framework highlighting how the object hypothesis improve in quality over time. Often within the field you will see several methods used together to improve the result, even methods which are opposed by their definition as for example the top-down approach, starting from known characteristics of the objects, and the bottom-up, which has no such data and relies more on continuity of the points, the two methods being combined in this paper[7] by Eran Borenstein which deals with shape guided object segmentation.

V. Implementation

The application consists of two nodes, i.e. executables which are designed to run in the ROS environment, one is a nodelet which extract the original point cloud from the file in which it is stored and publishes it to a topic.

The overall flow of the points between the operations can be seen in this next picture 1

So we can with the second node subscribe to that topic and actually process the point cloud then publish it to another topic.

1.jpg

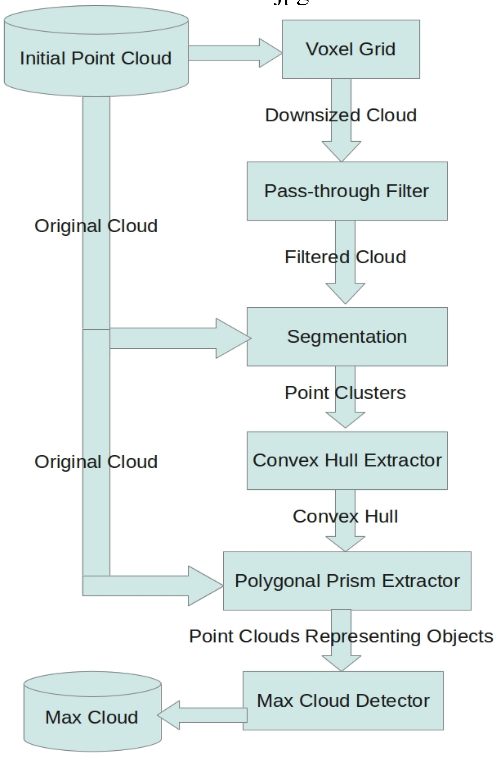

Fig. 1. The operation which take place within the node

A. Point Cloud Extraction

The point cloud in the case of this application isn't being published by the robots sensors but is a point cloud which has been saved into a file. In order to extract the point cloud

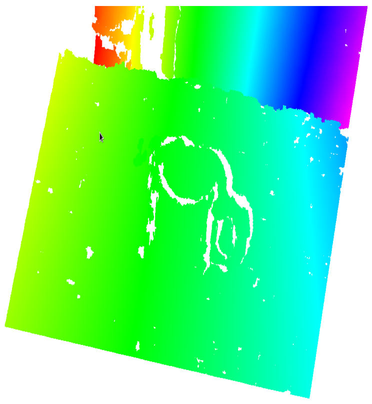

Fig. 2. Point Cloud after it's extraction from the .pcd file, the points rgb gradient are a indicator of their X coordinates.

from the file (e.g. the one shown in Figure 2), we use a nodelet i.e. a node which implements a single algorithm and for which we only give the parameters, nodelets are best used for simple tasks, if we need multiple algorithms or algorithms of our design within the ROS environment, we will chose to implement them as proper node.

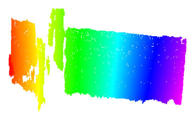
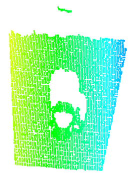
Fig. 3. The same point cloud after it's been downsized by running through the voxel grid.

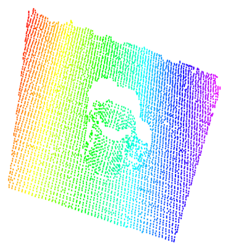
Fig. 4. The point cloud after it's been filtered on the Z axis.

As we have done with the following operations which are contained in one node.

B. Voxel Grid

The point cloud which has been extracted by the first nodelet is being published to a topic which our node will subscribe to and thus will obtain the data it needs to process.

And the first stage of the processing is downsampling the point cloud with a voxel grid like the one described in subsection II-B.

This is a important stage because there are further stages in the process in which having a cloud with a large number of points is more of a hindrance then a a advantage,

A important thing to note is that while we will use the downsized cloud for operation like normals calculations, we will be going back to the original cloud for the segmentation itself.

C. Pass Through Filter

The downsized cloud is then passed through another filter, this time a pass through filter with which we can select those points whose coordinates, on each axis fall within the limits we give it. In this application we've chosen to filter along the Z axis, with the lower bound being 0 and the upper bound being 1.1 meters.

The above image show the entirety of the remaining point cloud, all the points in the back we have judged to not be in our area of interested and thus we set up the filter so as to discard them.Of course filtering on the other axes is easily implementable but was unneeded in this application.

D. Segmentation

The first part of the segmentation process is to construct the normals, as we will be doing additional filtering based on them.A very useful feature of the PCL normals computation class is that we can use both the downsized and non downsized cloud within the same instance of the class.

Thus achieving both better computational times due to calculating the normals for a smaller amount of points and the ability to then segment for all the points from the original point cloud.

The actual segmentation class used is part of PCL's SAC family of classes which I described earlier in II-B, to be more precise it is the SACMODEL_NORMAL_PARALLEL_PLANE[8] which due to it's wonderfully descriptive names shows the most important aspects of our segmentation.

First we will only take into account planes which are parallel with a certain axes, in our case we've chosen the X axis, this is of a great help in practical application as we often need to deal with only one surface and it cuts down the number of points which have to be published thus increasing speed.

The other advantage of using the SAC-MODEL_NORMAL_PARALLEL_PLANE class is that we will be able to cast aside points depending on both their positions and normals, if we'd have gone only with position invariably we would have had points which we think are part of the right point cluster but in reality are noise or part of another object.

But by taking into account both the deviation of the point's normals and the point's positions we achieve a a much greater accuracy in separating the inliers from the outliers.

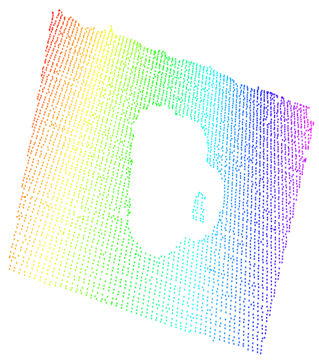
Fig. 5. The point cloud after segementation you will notice the points of the object we desired have been segmented while the points that form the plane have remained.

E. Convex Hull

The next stage needed for the extraction of the point cluster which represents the segmented object is constructing it's convex hull.

A convex hull for a set of points in Euclidean space is the minimum convex set contains all those points, it can be visualized as taking a rubber band and stretching it out so that all the points are in it's interior.

PCL once again has a class which can provide us with the convex hull, but as part of the application we've include for the user the option to shrink this convex hull as to extract fewer objects if he so choses.

The user supplies his choice on the scale of the convex hull through parameters of the ROS environment.

F. Polygonal Prism

Creating the polygonal prism is a important part of the segmentation process, what we had at the earlier stage the convex hull was limited in that it didn't have a height per say and thus separating points using only it wouldn't result in point clusters which adequately reflect real life objects.

Thus a key aspect of this stage is that we add another dimension height,this along with other parameters the user must change frequently have all been implemented with ROS parameters.

Fortunately the PCL class which creates the polygonal prism also allows us to publish the inliers i.e the points which actually represent the objects beign segmented,thus the processing is nearly done.

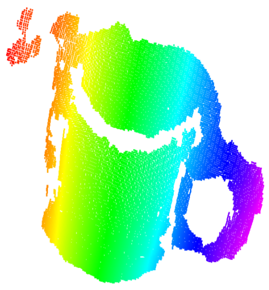
Fig. 6. The points located within the prism notice that there are at least two clusters out of which only one we want to use.

G. Extracting the biggest cloud

Within the polygonal prism there may be multiple point clusters, potentially all of them objects we might be interested in, at this stage of the application we need to make a choice whether to present all these point clusters,some of them or only one and on which criteria we judge them by.

For this application, the chose criteria is the total number of points of a cluster it's size and we chose to pass on only the biggest cluster.

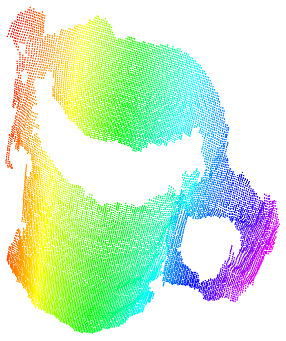
Fig. 7. The final point cloud representing our object segmentated from all the other points.

To do this we use PCL's EuclideanClusterExtraction[8] class then we simply iterate over the cluster and select the one with the biggest size which we then publish as the final result of this application.

VI. Conclusion

This application has produced point clouds which are relatively free of noise and false inliers thus which can be passed forward to other nodes within the robot's ROS environment which can then process them while being safe in the knowledge that these points do represent objects.

A great deal of attention was put into eliminating outliers via the normals estimation, as to not accidentally cluster other objects or people with the objects which for the robot it must manipulate, often times these manipulations being potentially destructive to other objects then the designated targets.

While this application can be used as a stand alone objects even segmented ones are of little use in of themselves to a robot, as it needs to know more about the objects in it's environment in order to properly interact with it. This pipeline are to our knowledge being right now used as the starting point of two other robotic vision applications, one in which the objects thus extracted are classified via Support Vector Machine (SVM) learning. And another in which the extracted objects have their histograms calculated and used for the calculation of a 4 Degrees of Freedom Pose (4DOF) which is then used in all grasping applications.

Of course the segmentation process presented in this paper is based solely on already existing capabilities of ROS and PCL, and thus a similar pipeline could be used and has been used in a much larger array of vision applications. All of which inform the robot about it's world starting from the basic building blocks of individual i.e. segmented objects.

Acknowledgment

G.P. Bora is studying via the Erasmus Programme at Intelligent Autonomous Systems, Technical University of Munich, Germany. bora.george@student.utcluj.ro

Z.C. Marton is a doctoral at Intelligent Autonomous Systems, Technical University of Munich, Germany. marton@cs.tum.edu

References

[1] R. B. Rusu, “Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments,” Ph.D. dissertation, Computer Science department, Technische Universitaet Muenchen, Germany, October 2009.

[2] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, “Ros: an open-source robot operating system,” in *ICRA Workshop on Open Source Software*, 2009.

[3] M. Zuliani, C. S. Kenney, and B. S. Manjunath, “The multiransac algorithm and its application to detect planar homographies,” in *IEEE International Conference on Image Processing*, Sep 2005. [Online]. Available: http://vision.ece.ucsb.edu/publications/05ICIPMarco.pdf

[4] K. G. Derpanis, “K-means clustering,” 2006.

[5] M. AlemŽn-Flores, L. Alvarez, P. AlemŽn-Flores, and R. Fuentes-Pavon, “Segmentation of computed tomography 3d images using partial differential equations,” *Signal-Image Technologies and Internet-Based System, International IEEE Conference on*, vol. 0, pp. 345–349, 2011.

[6] M. Mårten Björkman and D. Kragic, “Active 3d scene segmentation and detection of unknown objects,” in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, may 2010, pp. 3114 –3120.

[7] E. Borenstein, “Shape guided object segmentation,” in *In Proc. CVPR*, 2006, pp. 969–976.

[8] R. B. Rusu and S. Cousins, “3D is here: Point Cloud Library (PCL),” in *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.