LIDAR POINT CLOUD SEGMENTATION DEVELOPMENT DOCS

CREATED BY: ANISH SAHA

DATE OF PUBLISHING: Dec 9th, 2018

Version 1.0

PRODUCT OVERVIEW

The software processes raw 3D Lidar data point clouds to segment them into logical clusters. The output segments after execution are Roads, Buildings, and Miscellaneous clusters. This is a classic computer vision problem for autonomous cars and can be adapter for higher level object detection tasks.This document provides an understanding of the tasks performed by the software and how to use it. Note, it is still a work in progress with basic functionality implemented as of Version 1.0.

DIRECTORY STRUCTURE

The root folder contains 3 subdirectories and 1 README file. Make sure to read the README.txt before attempting to run the product. Each of the folders and their contents are listed below:

- ❏ **docs -** This contains the research paper the code implementation is based on, the development documentation and Labels.txt. Labels.txt contains the category attached to each label for the labelled dataset.
- ❏ **README.txt -** This contains important instruction to run the software. Make sure to read it before attempting an execution of the same.
- ❏ **dataset -** This has two subdirectories 'LabeledData' and 'UnlabeledData'
  - ❏ **LabeledData -** This contains LIDAR point cloud .csv files with each point labelled. The associate names for the labels are available in 'docs/Label.txt'
  - ❏ **UnlabeledData -** This contains LIDAR point cloud .csv files with the labels removed. This is also where the subdirectories are generated for each scene and their corresponding segments.
- ❏ **src -** This contains 3 subdirectories 'segmentation','rendering' and 'plots' and all executables reside in these location
  - ❏ **segmentation -** This contains the makefile, the .cpp and .hpp files and a header dependency folder for the segmentation module of the program.
  - ❏ **plots -** This contains the python file 'plotClusters.py' that can be used to visualize the point clouds as an interactive 3D scatter plot. It asks for a sub-directory in the 'UnlabelledData' directory as choice and all files in the chosen directory are plotted onto one single scatter plot.
  - ❏ **rendering -** This is a work in progress as of Version 1.0 and is scheduled to contain code for more efficient visualization of point clouds using OpenGL libraries.(Currently plots can be used to visualize the same)

SEGMENTATION WALKTHROUGH

ROAD/GROUND SEGMENTATION

This segments all points that belong to the road or ground surface.

A SceneModel object has to be first created by reading an unlabeled .csv file. This is accomplised by the **void** SceneManager::createScene(). This also populates the std::vector<Vertex> vertexList, which will store all the 3D vertices associated with the scene. Now this scene is available in the list of loaded scenes and is ready for processing.

To segment the scene, a call to **void** SceneManager::segmentScene() is made by the SceneManager. This initiates a list of processes. First the entire scene is divided into equal square blocks in the xy plane of side length #define SIDE. Now each block tries to fit a ground plane with the lowest height points. This is achieved using a RANSAC plane fitting technique. As of Version 1.0, number of iterations have been fixed to 15 for performance optimization. Future versions may allow a modifiable value. The method **void** SceneModel::fitRansacRoadPlane() performs this task. On completion, the Road and non-road vertices are put into a segment map segmentMap["ROAD"] and segmentMap["ROADComp"] respectively. They are also written in files in a subdirectory of 'UnlabelledData' with the corresponding std::string sceneId.

BUILDING FACADE SEGMENTATION

This segments all points that belong to building facades.

This is a two step process. First, a building score is computed using a height and a density feature by a call to **void** getBuildingScore(). This also divides the x-y plane into smaller equal square blocks. The height of each block is then compared against the global maximum height(Normalised) amongst all blocks. This ensures blocks with tall features are candidates for buildings. Notice that this also picks trees and street lights as possible candidates. The density score helps in removing candidates with small point cloud densities such as street lights or telephone poles. Now. the horizontal spread can be used to differentiate between the remaining building facades and trees. The ratio of the length of major axis and the ground area covered is a good measure to separate the two. The method **void** SceneModel::getBuildingShapeFeatures() computes this and puts the corresponding vertices into the segment map as segmentMap["Building"] and segmentMap["RoadBuildComp"]. They are also written in files in a subdirectory of 'UnlabelledData' with the corresponding std::string sceneId.

CLUSTERING THE REMAINING POINTS

The remainder of the points need to be clustered into individual objects. This is performed by the method **void** SceneModel::cluster(std::string dataDir). This first tries to overestimate the number of clusters using kMeans clustering. These clusters are written to files in a subdirectory of 'UnlabelledData' with the corresponding std::string sceneId. Once an over segmented object list is obtained, clusters that are close to each other and have the same direction of spread are merged to create supervoxels. The method **void**

SceneModel::makeSuperVoxels(std::map<**int**,std::vector<Vertex*> > cMap,std::vector<Vertex> centers,std::string dir) performs this and puts them in the map, std::map<**int**,std::vector<Vertex*> > superVoxelMap. These are written to files in a subdirectory of 'UnlabelledData' with the corresponding std::string sceneId.

CLASSIFICATION OF SUPERVOXELS *(NOT INCLUDED IN VERSION 1.0)*

This is a work in progress as of Version 1.0. Feature vectors are computed for the supervoxels using the method Feature SceneModel::getFeatures(std::vector<Vertex *> vertices). This populates the following feature vector: **struct** Feature{*//Geometric:***float** area;**float** edgeRatio;**float** maxEdge; std::vector<**float**> Covariance;*//Orientation and location:***float** height;std::vector<**float**> normal;*//3D features:***float** density;}. The 'LabelledData' subdirectory has labelled data that is used to train a classifier that classifies supervoxels into subcategories.