Lidar Obstacle Detection

Meets Specifications

Dear student,
Congratulations on passing the lidar obstacle detection project of the course 👏🏽 . It was great to review your submission. I've added some tips that might help you further your learning process. So please go through the comments carefully. Continue on in the course with the same enthusiasm. **All the best to gain more knowledge throughout the course.**

Here is an external link that you can use to further your learning

- This Matlab tutorial page is extremely detailed documentation explaining with visual and graphical results the various methods that are used for tracking. **The page also shows the difference between the Constant Velocity (CV) model and the Constant Turn Rate - Velocity (CTRV) model.** You will learn more about this in your final project. Just have a heads up by going through this tutorial.

Happy learning. Have a great day.

Cheers.

Compiling and Testing

✓ **The project code must compile without errors using `cmake` and `make`.**

✅ The project code must compile without errors using cmake and make.

Feedback:
Well done👏🏽. **Your project is able to compile with the cmake and make commands without any error.**

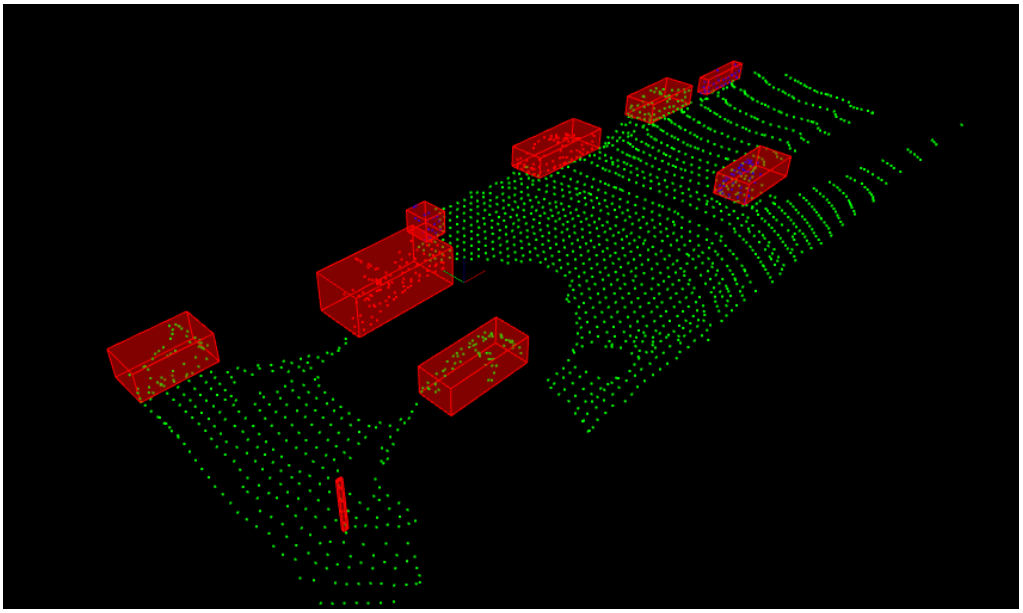
Pro Tips 💫**:**
Here is a tutorial blog from the CMake Organization which covers all the important functions that we use in `CMakeLists.txt` file. Feel free to take a look at the file and learn more about how each component of the CMakeLists.txt file works.

Obstacle Detection

Bounding boxes enclose vehicles, and the pole on the right side of the vehicle. There is one box per detected object.

✅ Bounding boxes enclose vehicles, and the pole on the right side of the vehicle. There is one box per detected object.

Feedback:
Great job. 👏 **Getting a bounding box on the small pole on the right side of the vehicle is one of the toughest things.** You were able to do it with ease. Nice work 😃.

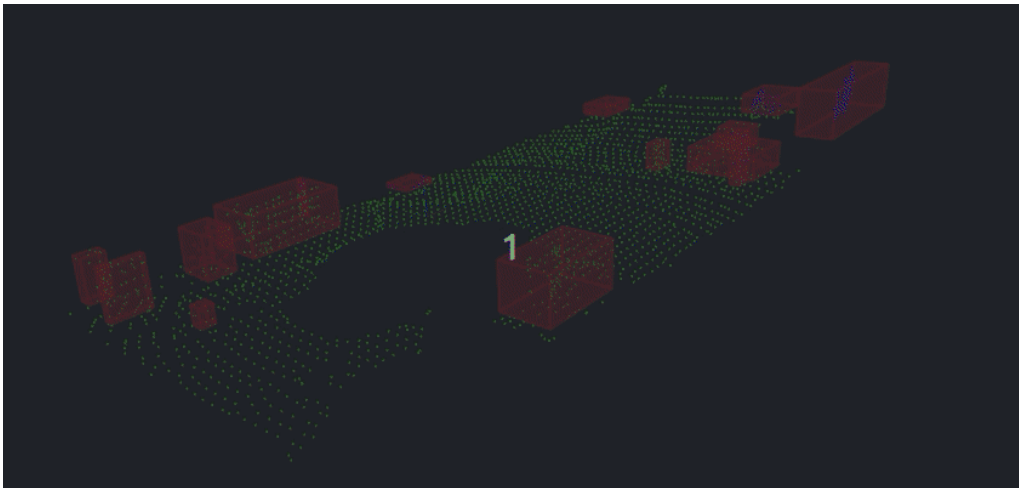


Most bounding boxes can be followed through the lidar stream, and major objects don't lose or gain bounding boxes in the middle of the lidar stream.

✅ Most bounding boxes can be followed through the lidar stream, and major objects don't lose or gain bounding boxes in the middle of the lidar stream.

Feedback:
I was able to see that the **bounding boxes were very consistent throughout all the lidar frames in the stream.** None of the bounding boxes were lost due to incorrect clustering.

I also tried running the code with data set 2 (data_2) and the results were consistent. Great work.👏 Keep it up.



The code used for segmentation uses the 3D RANSAC algorithm developed in the course lesson.

✅ The code used for segmentation uses the 3D RANSAC algorithm developed in the course lesson.

Feedback:
One of the **trickiest part of this project was extending the 2D RANSAC taught in the classroom to a 3D RANSAC algorithm** for use in this project. Nicely done👏.

Pro Tips💫:
There is an improvement that you can do in the filtering process that you do before the segmentation process.

Finding the indices of the point of roof and then removing it using ExtractIndices function is one skill that you must know. But there are other ways to do that. As shown in the image below, there is function called setNegative in the CropBox class.
This function if,

- set to `False` will give you the points outside the box value mentioned in the `setMin` and `setMax` functions.
- set to `True` will give you the points inside the box value mentioned in the s`etMin` and `setMax` functions.
  Below is an example of how you can do it more efficiently.

```
template<typename PointT>
typename pcl::PointCloud<PointT>::Ptr ProcessPointClouds<PointT>::FilterCloud(typename pcl::PointCloud<PointT>::Ptr cloud, float filterRes, Eig
xPoint)tor4f minPoint, Eigen::Vector4f maa
{
    typename pcl::PointCloud<PointT>::Ptr voxel_cloud (new pcl::PointCloud<PointT>);
    typename pcl::PointCloud<PointT>::Ptr cropped_cloud (new pcl::PointCloud<PointT>);

    // Create the filtering object
     typename pcl::VoxelGrid<PointT> vox_fileter_;
     vox_fileter_.setInputCloud (cloud);
     vox_fileter_.setLeafSize (filterRes, filterRes, filterRes);
     vox_fileter_.filter (*voxel_cloud);

     typename pcl::CropBox<PointT> cropbox_filter_;
     //first crop the interest region
     cropbox_filter_.setMin(minPoint);
     cropbox_filter_.setMax(maxPoint);
     cropbox_filter_.setInputCloud(voxel_cloud);
     cropbox_filter_.filter(*cropped_cloud);

     //filter the roof points
     cropbox_filter_.setMin(Eigen::Vector4f(-3.0, -3.0, -1.0, 1.0));
     cropbox_filter_.setMax(Eigen::Vector4f(3.0, 3.0, 1.0, 1.0));
     cropbox_filter_.setInputCloud(cropped_cloud);
     cropbox_filter_.setNegative(true);
     cropbox_filter_.filter(*cropped_cloud);

    return cropped_cloud;
}
```

Crop the region of interest

Remove the roof points directly.

The code used for clustering uses the Euclidean clustering algorithm along with the KD-Tree developed in the course lesson.

✅ The code used for clustering uses the Euclidean clustering algorithm along with the KD-Tree developed in the course lesson.

Feedback:
As before, **extending an algorithm from 2D implementation to 3D implementation is not that easy and straightforward**. But you've done it with ease.👏😃 Nice work.

Pro Tip💫:
There are some ways of improving the speed even with limited processing power. This method is called **Balanced Kd-Tree** which is also mentioned in the classroom lesson. It is a little tricky to do it but it is worth the time since the search is much more efficient in a balanced Kd-Tree. **This youtube video will help you understand how to implement a balanced tree.**

Code Efficiency

Your code does not need to sacrifice comprehension, stability, or robustness for speed. However, you should maintain good and efficient coding practices when writing your functions.

Here are some things to avoid. This is not a complete list, but there are a few examples of inefficiencies.

- Running the exact same calculation repeatedly when you can run it once, store the value and then reuse the value later.
- Loops that run too many times.
- Creating unnecessarily complex data structures when simpler structures work equivalently.
- Unnecessary control flow checks.

✅ Your code does not need to sacrifice comprehension, stability, or robustness for speed. However, you should maintain good and efficient coding practices when writing your functions.

Feedback:
Your code is neat and clean and doesn't use any unnecessarily complicated data structure to store data. 😃. Keep the code clean always.

DOWNLOAD PROJECT

RETURN TO PATH