Semantic Segmentation RBE549 HW2

Shiva Kumar Tekumatla
Robotics Engineering Department
Worcester Polytechnic Institute
Worcester, MA, U.S.A.
stekumatla@wpi.edu

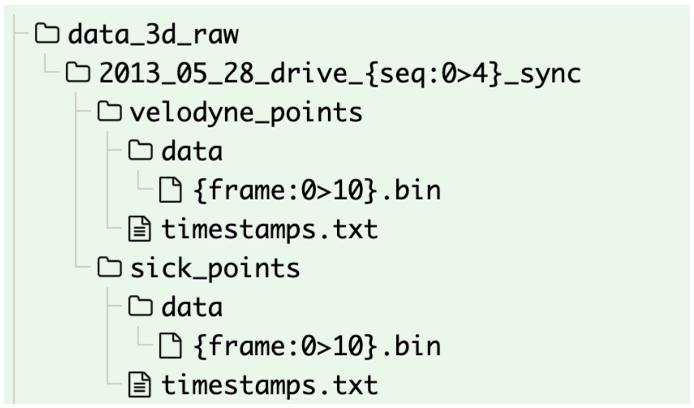
***Abstract*—This document consists of the project implementation of Semantic segmentation of LIDAR point cloud data. Since a lot of self-driving cars rely mostly rely on the LIDAR point clouds, it is essential to differentiate the pseudo-dynamic obstacles such as cars, to better simultaneous localization and mapping. In this work, I initially present the data that I am using for segmentation, followed by the Iterative closest point implementation for mapping the LIDAR , and then followed by the segmentation using neural networks.**

I. Introduction

LIDAR data is an essential part of self-driving architectures for depth sensing. But during the mapping and localization, apart from point clouds that are generated by LIDAR, a higher level of semantic information is required. One such example is differentiating the pseudo-dynamic objects such as cars. Furthermore, moving cars present a big challenge, making it difficult to depend on LIDAR-based SLAM. To make the situation worse, current LIDAR sensors have a very low vertical resolution, making it hard to detect semantics (labels of what the objects are). To fix this problem, we can rely on computer vision to paint the point clouds with semantics. For this exercise, I am using data from the KITTI360 dataset that is captured by the SICK LMS 200 laser scanner.

II. Data

The data that is used for this assignment, follows the structure given by figure 1.



```
data_3d_raw
  2013_05_28_drive_{seq:0>4}_sync
    velodyne_points
      data
        {frame:0>10}.bin
      timestamps.txt
    sick_points
      data
        {frame:0>10}.bin
      timestamps.txt
```

Fig. 1. Data Tree

This data contains the SICK scans in BINARY format, defined in the SICK coordinates. Note that the SICK laser scanner has a higher FPS, thus the frame indices of SICK scans do not align with those of images or Velodyne scans. This data is captured by the laser scanners installed at the poses as given by figure 2.

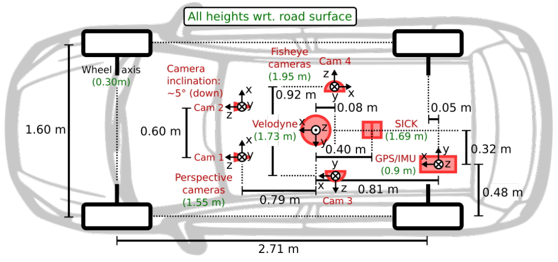


Fig. 2. Camera Poses

This data also contains the camera calibration data. The calibration folder contains the intrinsic and extrinsic parameters of the sensors.

III. Building The Map

To build the map, point cloud data from each binary data is stitched together using the Iterative Closest Point Algorithm (ICP). For this implementation, I used point-to-plane ICP. The theory for it is as given below.

A. *Point to Plane ICP*

The goal of ICP is to align two point clouds, the old one (the existing points and normals in the 3D model) and the new one (new points and normals, which we want to integrate into the existing model). ICP returns rotation+translation transform between these two point clouds. The Iterative Closest Point (ICP) minimizes the objective function which is the Point to Plane Distance (PPD) between the corresponding points in two-point clouds.

$$E = \Sigma_i ||ppd(p_i, q_i, n_i)||_2 \rightarrow 0 \quad (1)$$

Specifically, for each corresponding points P and Q, it is the distance from the point P to the plane determined by the point Q and the normal N located in the point Q. Two points

P and Q are considered correspondent if given the current camera pose they are projected in the same pixel.

p - i^{th} point in the new point cloud q - i^{th} point in the old point cloud n - normal in the point q in the old point cloud

TO minimize the objective function, we use the Gauss-Newton method. In the Gauss-Newton method we do sequential steps by changing R and t in the direction of the function E decrease, i.e. in the direction of its gradient. At each step, we approximate the function E linearly as its current value plus Jacobian matrix multiplied by delta x which is concatenated delta R and delta t vectors. We find delta R and delta t by solving the equation $E_approx(delta_x) = 0$. We apply delta R and delta t to current Rt transform and proceed to next iteration

To linearize E , we can approximate it in the infinitesimal neighborhood. Using R , and t , we can modify E to the following.

$$E = \Sigma||[(R.p + t) - q]^T.n||_2 \tag{2}$$

The output for this mapping is shown below. To speed up the process, I considered only part of the point cloud data. Figure **??** shows the mapping without transformation, and figure 8 shows with the transformation.

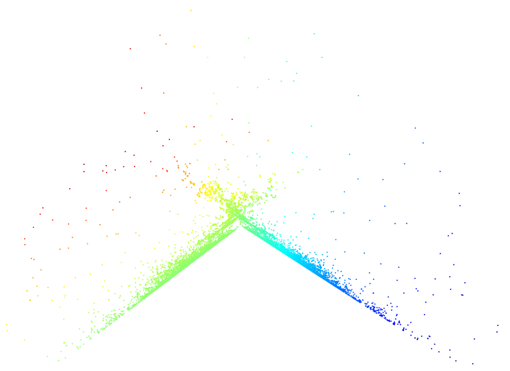


Fig. 3. Mapping of parts of the SICK LiDAR data

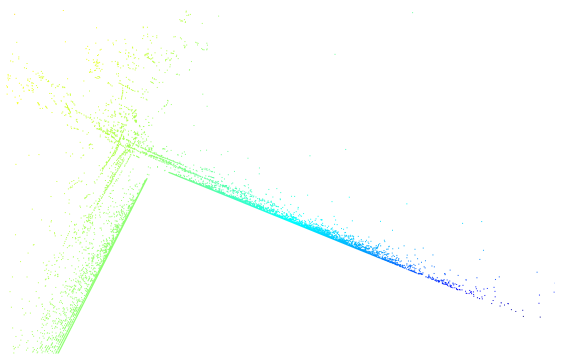


Fig. 4. Mapping of parts of the SICK LiDAR data

I could not succesfully make the ICP work for entire data , Hence I could not rely on this data for Segmentation. For the segmentatipn part, I used test images from the KITTI360 dataset.

IV. Semantic Point Painting The Map

For this exercise , I used psp$_resnet101_adearchitecture.Andfollowingaretheresults.$

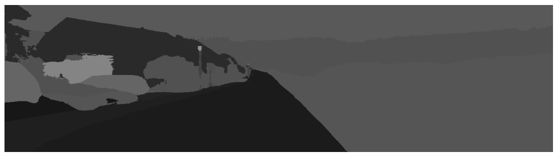


Fig. 5. Instance Image input

Fig. 6. Instance Image input

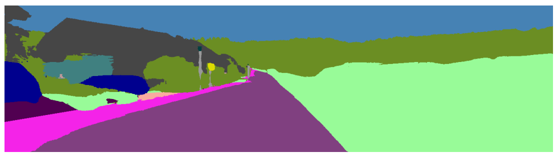


Fig. 7. Instance Image input

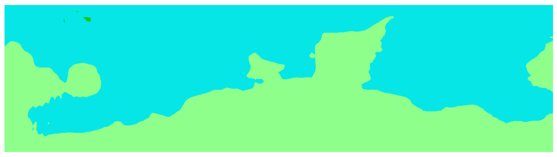


Fig. 8. Instance Image input

References

[1] https://rbe549.github.io/fall2022/proj/p4/
[2] http://www.open3d.org/docs/release/tutorial/geometry/pointcloud.html
[3] https://github.com/AmrElsersy/PointPainting
[4] https://github.com/ClayFlannigan/icp/blob/master/icp.py