RBE/CS549 Computer Vision
Homework 2 - LIDAR Semantics

Anagha R. Dangle
Email: ardangle@wpi.edu
Using 1 late day

***Abstract*—This homework is divided into 2 stages. The first stage is to build a high-definition map using the raw LIDAR data. Point-to-Point ICP variant is used to perform map building. It is an algorithm used to minimize the distance between two point clouds. The second stage involves utilizing the RGB images to perform semantic segmentation of every frame in the images. Once the semantic segmentation is performed, the task is to project the color information from the images onto the LIDAR ICP map built before.**

I. STAGE-I: MAP BUILDING USING ICP

We initially have the raw LIDAR data in the form of binary files. We first convert these binary files for each image to a PCD file which is a point cloud file format required for Open3D. This is done by using a separate file which converts all the .bin files to .pcd format. Now we have to add the first 2 pcd files, then add the 3rd pcd file to the result, and so on. To build a map, we need to add multiple of these point cloud files from the raw LIDAR data. Point-to-Point iterative closest point (ICP) algorithm was used to perform the same. From the Point-to-Point ICP we get transformations from one point cloud to another and that is then used to add all the point clouds by transforming each source to the target.

A. Point-to-Point ICP

The Point-to-point ICP works in the following way:

- Initialize the alignment between the two point clouds by selecting a few points from one cloud and finding the closest points in the other cloud.
- Iteratively improves the alignment by using an optimization algorithm to minimize the distance between the two point clouds.
- At each iteration, the algorithm calculates the transformation (rotation and translation) that aligns the points in one cloud with the points in the other cloud.
- The algorithm stops when the transformation between the two point clouds is below a certain threshold or when the maximum number of iterations has been reached.

The output of ICP is shown in the following 2 figures:

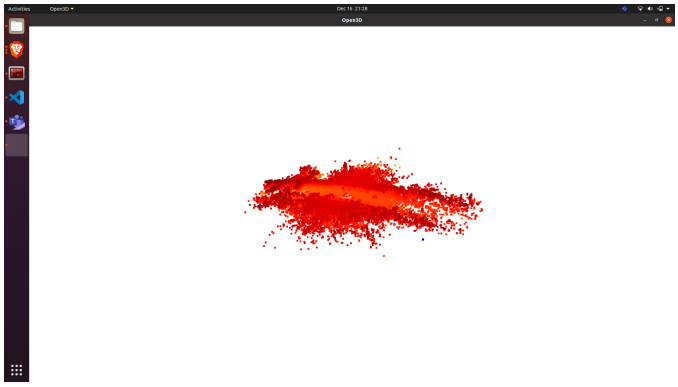

Fig. 1: Generated map

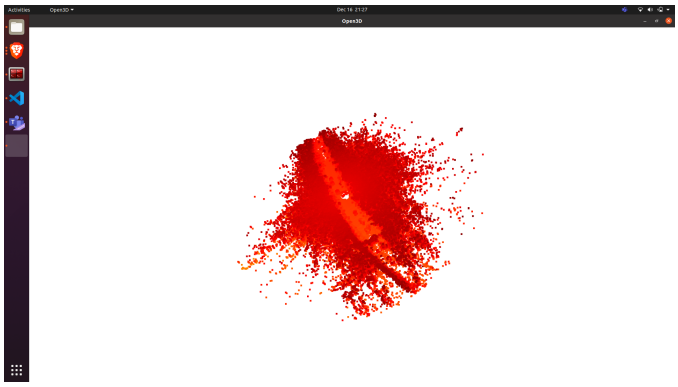

Fig. 2: Generated map

II. STAGE-II: SEMANTIC SEGMENTATION

Semantic segmentation bascically refers to assigning a label or class to each pixel of an image. This can be used to understand the contents of an image and to perform tasks such as object detection and tracking. For Semantic Segmentation of the RGB images, we use the BiSeNetv2 network architecture. It takes the RGB image as the input and gives the semantic segmented image as the output of the model.

A. BiSeNetv2

BiSeNetv2 is an encoder-decoder architecture for semantic image segmentation. It consists of three main components: an encoder, a decoder, and an auxiliary branch. The encoder is a convolutional neural network (CNN) that processes the input image and extracts features from it. The decoder is a CNN that upsamples the feature map produced by the encoder and generates a per-pixel prediction of the class labels. The auxiliary branch is a CNN that processes the input image at a lower resolution and produces a prediction for the class labels. It is designed to capture global context information and is used to complement the prediction produced by the decoder.

The Network architecture of BiSeNetv2 as given in the paper is as follows:

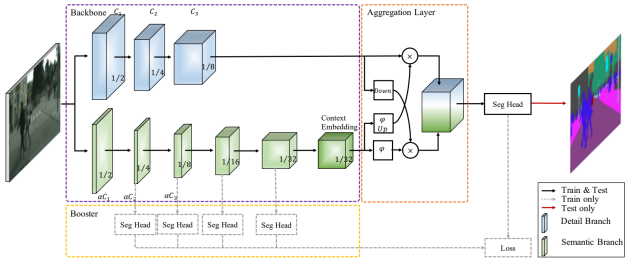

Fig. 3: BiSeNetv2 architecture

B. Coloring the PointCloud Map

Now that we have successfully segmented the RGB images using the model described above, in this part, we have to transfer the colors of the semantic segmented images onto the PointCloud-generated map. We utilize the camera extrinsic to perform the same. 3 different types of files from the KITTI 3D Objection Detection dataset as follows are used in the homework.
camera$_2$ image (.png),
calibration (.txt),
velodyne point cloud (.bin),

For each frame, there is one of these files with the same name but different extensions. The image files are regular png files. T

The point cloud file contains the location of a point and its reflectance in the lidar coordinate. The calibration file contains the values of 6 matrices — $P0–3, R0_{rect}, Tr_{velo-to-cam}, and Tr_{imu-to-velo}$.

The Px matrices project a point in the rectified referenced camera coordinate to the $camera_x$ image. $camera_0$ is the reference camera coordinate. $R0_{rect}$ is the rectifying rotation for reference coordinate (rectification makes images of multiple cameras lie on the same plan). $Tr_{velo-to-cam}$ maps a point-in-point cloud coordinate to reference coordinate.

Point Painting paints a point cloud with semantic labels, based on the provided semantic map. The function first clips the point cloud to a specified range, then resizes the semantic map to match the shape of the point cloud. It then projects all the points in the point cloud onto the image plane using the provided camera matrices. After filtering out any points that are outside the bounds of the image or have negative coordinates, the function assigns a class to each point in the point cloud based on the corresponding pixel in the semantic map. Finally, it returns the painted point cloud, which is a copy of the original point cloud with an additional channel indicating the class of each point.

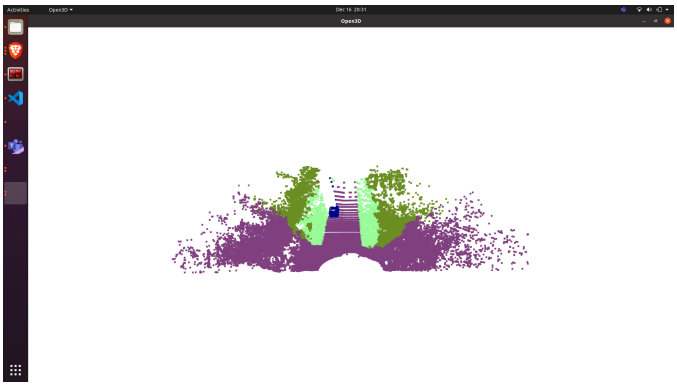

Fig. 4: Output of 1 image

Fig. 5: Output of 2 images

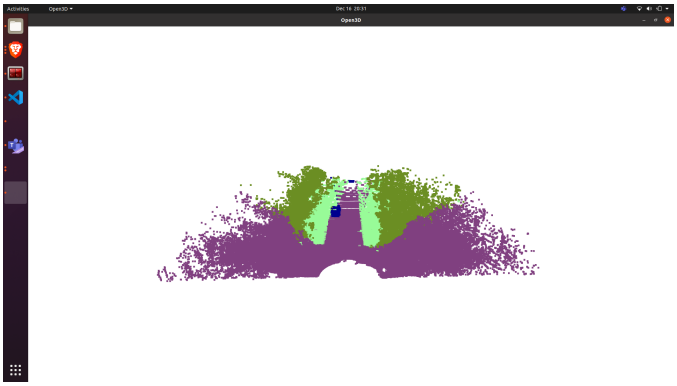

Fig. 6: Output of 3 images

REFERENCES

[1] https://github.com/ahosnyyy/PointPainting
[2] https://www.cvlibs.net/datasets/kitti-360/index.php

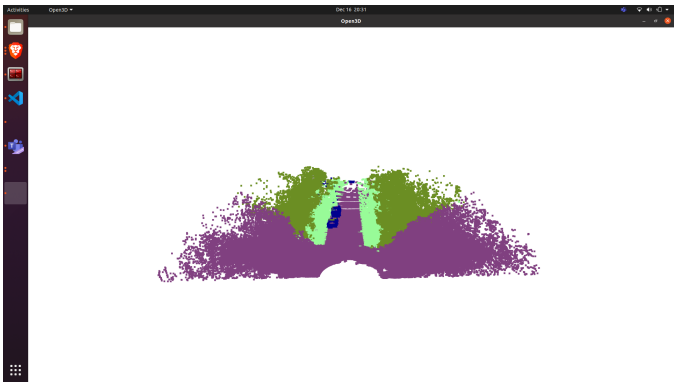

Fig. 7: Output of 4 images

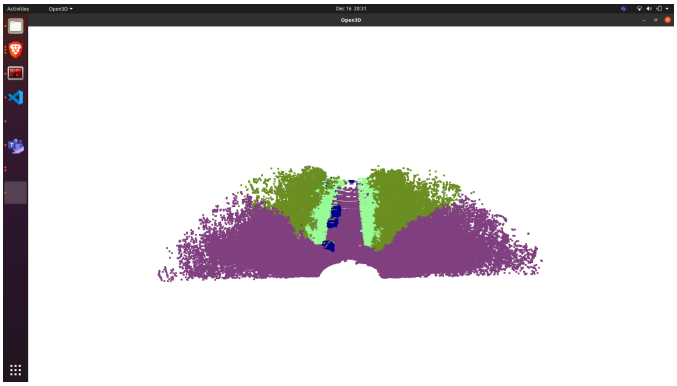

Fig. 8: Output of 5 images

Fig. 9: Output of 6 images

Fig. 10: Output of 7 images

Fig. 11: Output of 8 images

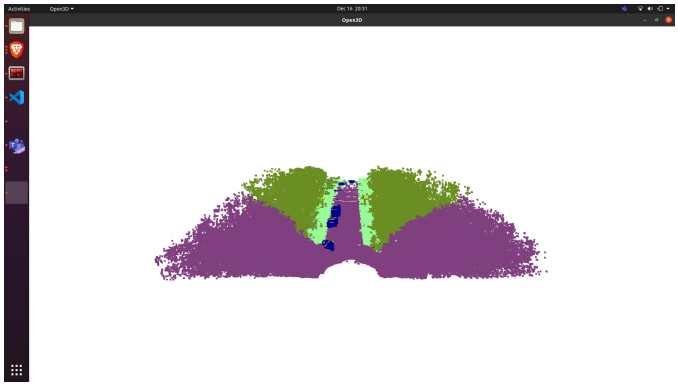

Fig. 12: Output of 9 images

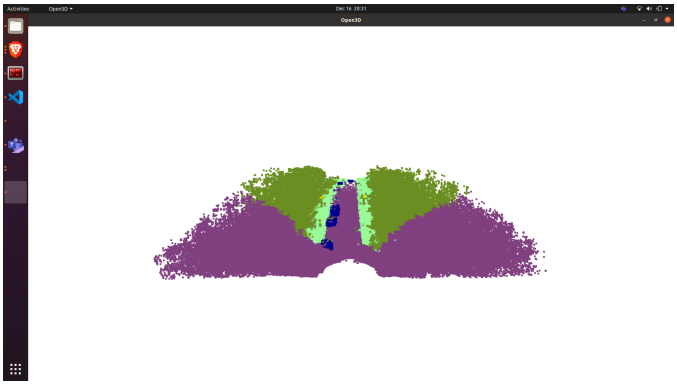

Fig. 13: Output of 10 images

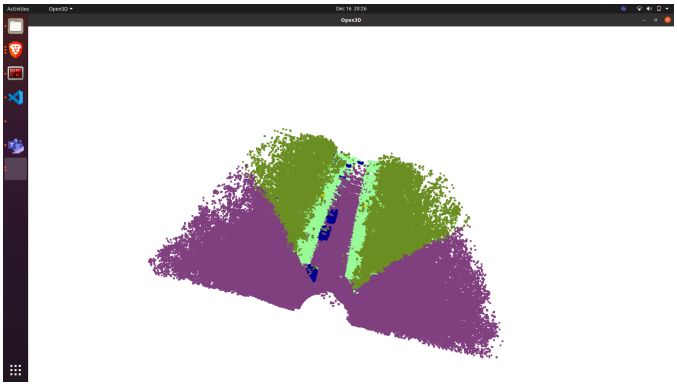

Fig. 14: Final Output with different view