Sexy Semantic Mapping.

Venkatesh Mullur
Robotics Engineering Department,
Worcester Polytechnic Institute,
Worcester, MA, USA.
vmullur@wpi.edu

I. Introduction

In Structure from motion, the goal was to find 3D depth using either a monocular camera from different poses or from a stereo camera; the goal of Visual Inertial Odometry was to combine camera shots and the readings of the IMU sensor. VIO was used to perform localization and mapping using a variant of the Extended Kalman Filter called MSCKF. We know that Lidar (Light Detection and Ranging) is a remote sensing technology that uses laser beams to measure the distance to objects and generate high-resolution 3D point clouds of the environment. Point clouds are sets of points in 3D space that represent the shape and surface characteristics of an object or environment. They are commonly used in computer vision, robotics, and 3D modeling to represent 3D data and can be generated using a variety of techniques, such as 3D scanning, photogrammetry, or lidar (Light Detection and Ranging). The main goal of this project is to use a camera to sfor semantic segmentation and project it on the point cloud to see a semantic segmentation of the point cloud. Semantic segmentation is the process of labeling each pixel in an image with a class label, such as ”car,” ”road,” or ”building.” By combining the data from a camera and Lidar, you can potentially use both the color information from the camera and the depth information from the Lidar to improve the accuracy of the semantic segmentation. Once you have a segmented point cloud, you can use it to perform various tasks such as object recognition, localization, and mapping. These techniques can be useful in a variety of applications, such as autonomous vehicles, robotics, and 3D modeling.

II. Background

Segmenting point clouds can indeed be challenging due to the lack of color information and the complexity of the data. One approach to improve the accuracy of point cloud segmentation is to use multiple sensors, such as both a Lidar and a camera, to capture complementary information about the environment. By combining the data from these different sensors, it can be easier to segment the point cloud and accurately identify different objects.

One way to combine the point cloud data with image data is to use a projection method. This involves projecting the point cloud onto the image plane and using image segmentation techniques to classify the points in the projected point cloud. This can be useful if the image has a higher resolution or if the image data is easier to process due to the presence of color information.

Another approach is to use a registration method to align the point cloud and image data. This involves finding the transformation that aligns the two data sets and applying it to the point cloud. This can be useful if the point cloud has a higher resolution or if the point cloud data is easier to process due to the presence of depth information.

Ultimately, the choice of approach will depend on the specific application and the characteristics of the data being used.

III. Building the Map

- The Kitti-360 dataset is a collection of Lidar and camera data collected from a 360-degree Lidar sensor mounted on a moving vehicle. It is commonly used for evaluating algorithms for tasks such as 3D reconstruction, semantic segmentation, and object detection.
- To build a map using this dataset, you will need to process the raw point cloud and camera data to extract relevant information about the environment. This can involve tasks such as point cloud segmentation, image segmentation, and registration to align the data from the different sensors.

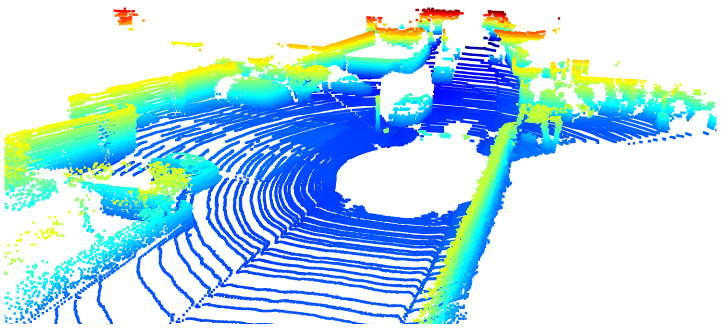

Fig. 1. Point Cloud at Time instant ”t”

- Once you have processed the data, you can use it to generate a 3D map of the environment. This can involve techniques such as SLAM (Simultaneous Localization and Mapping) to estimate the pose of the vehicle and build a map of the surrounding environment. You can also use the segmented point cloud data to identify and classify different objects in the environment, such as cars, pedestrians, and buildings.

- Overall, building a map using the Kitti-360 dataset can be a challenging task since it is a very big dataset and the scope of this project needs only ”Perspective Images for Train and Validation” and ”Raw Velodyne Scans”.
- Figure 1 shows the point cloud at a given time instant ”t”. Similarly, figure 2 shows the next point cloud. When extracting information from the point cloud, I could see that the point cloud contains information such as spatial location, intensity, and distance from the camera.
- Every point cloud was in the form of .bin and had to be converted into a point cloud using the Open3D library. Each point cloud contains around 150,000 points with the above-given information. We need to find point correspondences between the adjacent point clouds. For the same, this project uses the ICP or the Iterative Closest point algorithm.

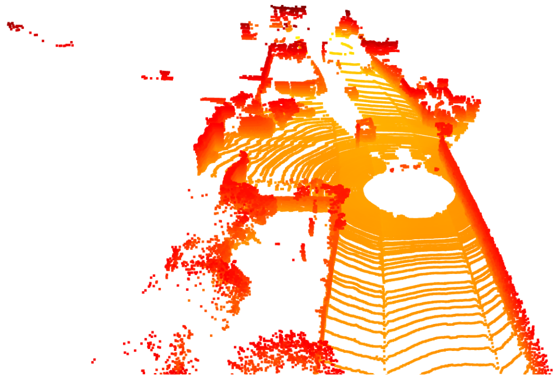

Fig. 2. Point Cloud at Time instant ”t+1”

- ICP, or Iterative Closest Point, is an algorithm for aligning two point clouds by minimizing the distance between corresponding points in the two clouds. It is widely used in computer vision and robotics for tasks such as object recognition, localization, and mapping.
- In the open3D library, ICP is implemented as a function that takes two input point clouds and returns the transformation that aligns the two clouds. The function iteratively adjusts the transformation based on the distance between corresponding points in the two clouds, until the alignment error is minimized. The function also provides various options for controlling the optimization process, such as the maximum number of iterations and the convergence threshold.
- There are several variations of the Iterative Closest Point (ICP) algorithm, each with slightly different optimization objectives and techniques. Some common types of ICP include:
    1) Point-to-Point ICP: This is the most basic form of ICP, in which the objective is to minimize the distance between corresponding points in the two point clouds.
    2) Point-to-Plane ICP: This variant of ICP minimizes the distance between points in one cloud and the nearest plane in the other cloud. This can be useful for aligning point clouds that represent surfaces, such as 3D scans of objects.
    3) Point-to-Surface ICP: This variant of ICP minimizes the distance between points in one cloud and the nearest point on a surface in the other cloud. This can be useful for aligning point clouds that represent more complex shapes, such as 3D scans of objects with curved surfaces.
    4) Robust ICP: This variant of ICP uses robust estimators, such as the RANSAC algorithm, to handle outliers and noisy data. This can be useful for aligning point clouds that contain a significant amount of noise or errors.

    Overall, the choice of ICP variant will depend on the specific application and the characteristics of the point cloud data.
- With the use of ICP, we basically get the transformations between the adjacent point clouds which can be registered or aligned to a particular point cloud. Figure **??** shows the output of ICP and shows what the map looks like.

$$E(\mathbf{T}) = \sum_{(\mathbf{p},\mathbf{q})\in\mathcal{K}} \|\mathbf{p} - \mathbf{T}\mathbf{q}\|^2 \tag{1}$$

- In the above equation, p and q are point clouds and T is the transformation from q to p. Iteratively, this error is to be minimized in such a way that they align with each other. This method is called $point to point ICP$

$$\mu_x = \frac{1}{N_x}\sum_{i=1}^{N_x} x_i \quad \text{and} \quad \mu_p = \frac{1}{N_p}\sum_{i=1}^{N_p} p_i$$

Fig. 3. Calculating the centroid of data

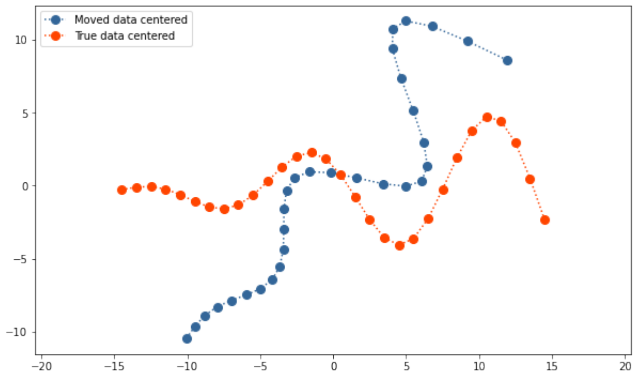

Fig. 4. How to make data centered

- Once the data is centered, from each point in the true data is taken and found the correspondences by using the equation 1 and the radius is predecided.

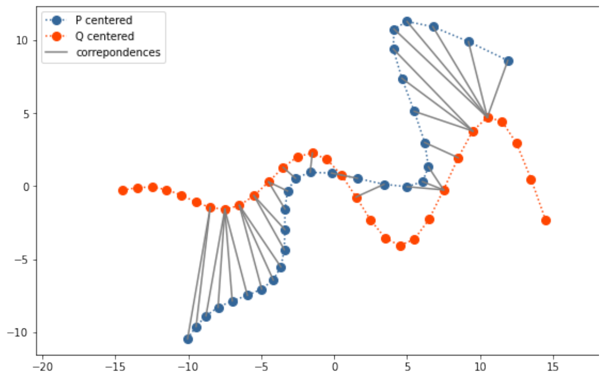

Fig. 5. How to make data centered

- The point correspondences are shown in figure 5 and Iteratively it is transformed in the frame of the true as shown in figure **??**

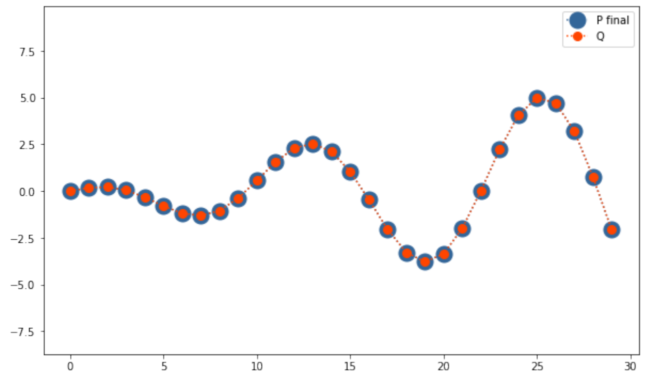

Fig. 6. How to make data centered

- Since the data is super large, this project takes use of only the first 100 samples of both Point clouds and the images. The output of ICP on the dataset is shown in the figure below.

IV. Semantic Segmentation

There are many semantic segmentation neural networks that you can use to predict semantic labels on each image frame. Some popular options include Fully Convolutional Networks (FCN), U-Net, and SegNet. These networks are trained on large datasets of annotated images and are able to predict the class label of each pixel in an image.

To transfer the semantic color labels from the RGB images to the Lidar point cloud, we can use the extrinsics between the sensors to project the labels onto the point cloud. Extrinsics refer to the transformation that aligns the coordinate systems of two sensors, such as a camera and Lidar. By using the extrinsics, you can transform the image coordinates to the Lidar coordinate system and apply the semantic labels to the corresponding points in the point cloud.
To do this, this project uses a registration method to align the

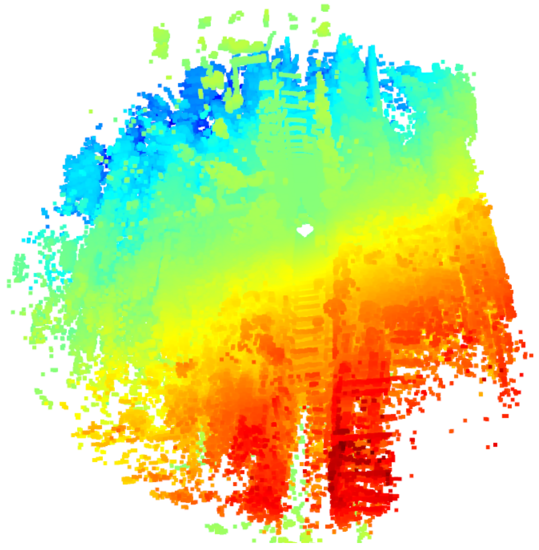

Fig. 7. Point Cloud After performing ICP

data and then apply the semantic labels to the point cloud. Once we have a segmented point cloud, we can use it to perform various tasks such as object recognition, localization, and mapping. These techniques can be useful in a variety of applications, such as autonomous vehicles, robotics, and 3D modeling.

1) To perform semantic segmentation, the project has taken help of DeepLabv3Plus-Pytorch which has pre-trained networks.
2) Specifically the project uses deeplabv3_resnet101.
3) Transferring the RGB colors from the images to the Lidar point cloud can be a useful way to validate the transformation operation and check that the alignment between the data is correct. If the transformation is accurate, you should see that the RGB colors are preserved in the point cloud and that the colors of different objects are distinct and do not blend into each other.
4) For example, if you have a red car in the image, you should see that the points in the point cloud corresponding to the car are colored red and do not blend into the colors of other objects in the scene. Similarly, if you have a green tree in the image, you should see that the points in the point cloud corresponding to the tree are colored green and do not blend into the colors of other objects in the scene.

Fig. 8. Image in the dataset

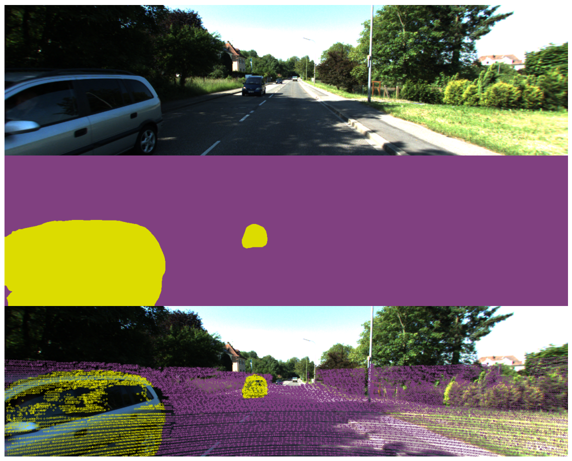
Fig. 9. Original Image, Segmented Image and the point cloud fused

5) By checking the RGB colors in the point cloud, you can ensure that the transformation operation is working correctly and that the alignment between the data is accurate. This can be useful for ensuring the quality of the segmented point cloud and for verifying the accuracy of the semantic labels applied to the point cloud.

Step 1: Transformation of raw depth values into meters

$$depth = 1.0/(raw_{depth} \times -0.0030711016 + 3.3309495161) \quad (5)$$

Step 2: Mapping depth pixels from depth image coordinates $[x_d, y_d]^T$ to depth camera coordinates $[X_d, Y_d, Z_d]^T$

$$X_d = (x_d - cx_d) \times depth(x_d, y_d)/fx_d \quad (6)$$

$$Y_d = (y_d - cy_d) \times depth(x_d, y_d)/fy_d \quad (7)$$

$$Z_d = depth(x_d, y_d) \quad (8)$$

Step 3: Transform point clouds from depth camera coordinates $\mathbf{X_d}$ to color camera coordinates $\mathbf{X_{RGB}}$

$$\mathbf{X_{RGB}} = R^{-1} \cdot \mathbf{X_d} - R^{-1} \cdot T \quad (9)$$

Step 4: Mapping point clouds from color camera coordinates $\mathbf{X_{RGB}}$ to color image coordinates $[x_{RGB}, y_{RGB}]^T$

$$x_{RGB} = (X_{RGB} \times fx_{RGB}/Z_{RGB}) + cx_{RGB} \quad (10)$$

$$y_{rgb} = (Y_{RGB} \times fy_{RGB}/Z_{RGB}) + cy_{RGB} \quad (11)$$

Note: After projecting to color image coordinates, x_{RGB} and y_{RGB} must be rounded to (1, 640) and (1, 480) respectively.

Fig. 10. Equations utilized to transfer the RGB semantic labels onto the LIDAR point cloud

One of the data images is given in figure 8 and the segmented figure is shown in figure 9.
Once the segmentation is done, then the project takes the help of GitHub to project the segmented images to the point cloud and then reprojects the color to the point cloud. After segmentation of the point cloud, ICP is used again to get a full segmented map which is given in figure 11
One way to improve the robustness of the segmentation is to use more image and point cloud frames. This can provide the algorithm with more data to work with, which can improve the accuracy of the segmentation. Additionally, using more frames can help to capture a wider range of viewpoints and lighting conditions, which can also improve the robustness of the segmentation.

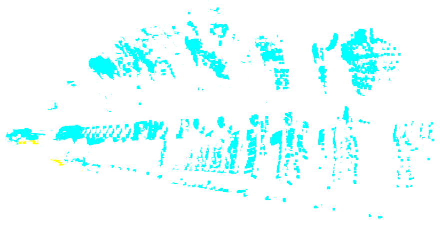
Fig. 11. Final Output

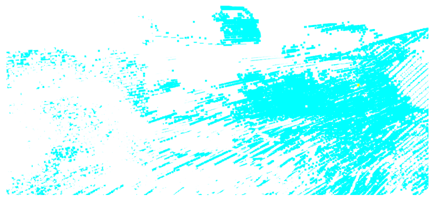
Fig. 12. Final Output

REFERENCES

[1] https://www.cvlibs.net/datasets/kitti-360/download.php

[2] https://github.com/VainF/DeepLabV3Plus-Pytorch/blob/master/

[3] http://www.open3d.org/docs/release/tutorial/pipelines/icp_registration.html

[4] https://www.youtube.com/c/CyrillStachniss?app=desktop

[5] https://github.com/naitri/PointPainting

[6] https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/tr98-71.pdf