HW-2: Sexy Semantic Mapping

Shreyas Kanjalkar
Robotics Engineering Department
Worcester Polytechnic Institute
Worcester, USA
skanjalkar@wpi.edu

I. Introduction

Self driving cars often use LIDARs as a central depth sensing modality. But often during mapping or localization, higher level semantic information is desired. One such cases is when parked cars drastically reduce what the car sees in-terms of features, thereby making it hard for localization or mapping. You would end up mapping the cars instead of buildings. These are called pseudo-dynamic obstacles. Furthermore, moving cars present a big challenge, making it hard for LIDAR based Simultaneous Localization And Mapping (SLAM) systems to reject these objects from becoming mapping artifacts. To accelerate the situation further, current LIDAR sensors have very low vertical resolution, making it hard to detect semantics (labels of what the objects are). To this end, experts in the field of robotics and computer vision decided to combine LIDAR's with cameras to obtain a semantic painted point cloud. The goal is to build a map from raw LIDAR point cloud and then transfer the predicted semantic labels from the camera image onto the LIDAR point cloud.

II. Data

The data used in this data set is from the KITTI-360 dataset. We only used part of it, and only the raw LIDAR point clouds, RGB images and sensor intrinsic and extrinsics. Specifically, the dataset used for this project is 2013_05_28_drive_0000_sync which contains 11,518 images and their corresponding binary files of point cloud.

III. Approach

The pipeline for this homework invovles two key steps:

1) Building the Map using classical methods (ICP)
2) Semantic Point Painting the Map

IV. Building the Map

As we are allowed to utilize any third party ICP Point to Point or Point to Plane. I considered both of them, but noticed that point to point worked much better than point to plane. The math for both of them is shown in the next subsections

A. *Point to Point ICP*

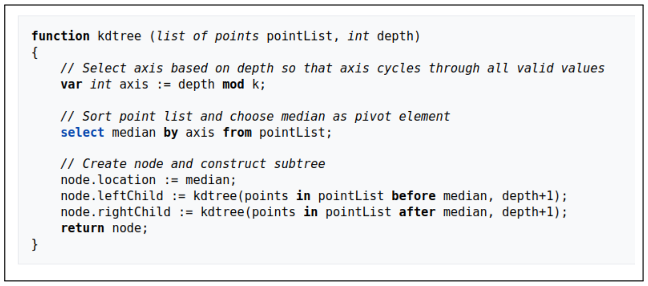
The Iterative Closest Point (ICP) algorithm is a method for aligning two point clouds, which are collections of 3D spatial data points. The goal of the algorithm is to find the transformation that best aligns one point cloud with another. The ICP algorithm operates iteratively, meaning it performs a series of steps repeatedly until the desired level of accuracy or convergence is achieved. Specifically, the algorithm initializes the transformation between the two point clouds and then, for each point in the source point cloud, it identifies the closest point in the target point cloud. The transformation that aligns the points in the source point cloud with the closest points in the target point cloud is then calculated and used to update the overall transformation between the two point clouds. This process is repeated until the transformation between the two point clouds converges to the desired level of accuracy.

The point to point ICP is explained as following in [1]:

1) Find correspondence set $\kappa = (\mathbf{p}, \mathbf{q})$ from target point cloud $\mathbf{P}$ and source point cloud $\mathbf{Q}$ transformed with current transformation matrix $\mathbf{T}$.
2) Update the transformation $\mathbf{T}$ by minimizing an objective function E($\mathbf{T}$) defined over the correspondence set κ. The objective function is given as:

$$E(\mathbf{T}) = \sum_{(p,q)\in\kappa} ||p - Tq||^2$$

3) The initial transformation matrix is a 4x4 is a matrix with random values. The ICP values over a certain number of iterations optimizes the Transformation matrix to find the best fit for the two point clouds. In a way, it can be seen as how we used to calculate Homography matrix, except there we used to choose 4 random points and applying RANSAC to find the best fit. In here we also find the best fit over a certain number of iterations.
4) Zhang et al[2] proposed a K-D tree approach for efficient computation of the point clouds and transformation. The pseudo code for the algorithm is shown below:



```
function kdtree (list of points pointList, int depth)
{
    // Select axis based on depth so that axis cycles through all valid values
    var int axis := depth mod k;

    // Sort point list and choose median as pivot element
    select median by axis from pointList;

    // Create node and construct subtree
    node.location := median;
    node.leftChild := kdtree(points in pointList before median, depth+1);
    node.rightChild := kdtree(points in pointList after median, depth+1);
    return node;
}
```

Fig. 1: KD-Tree algorithm

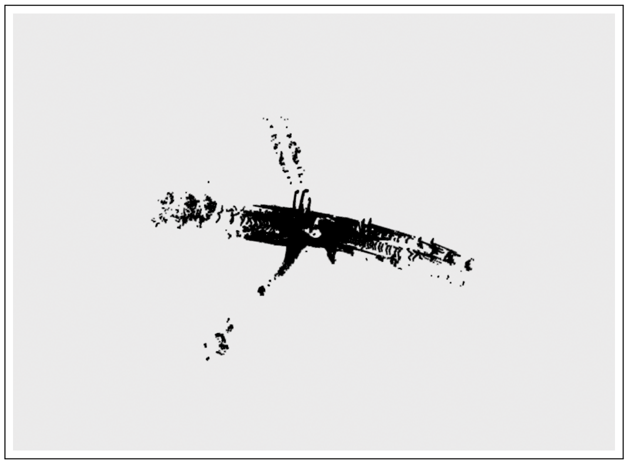


Fig. 2: Point cloud between frames using ICP

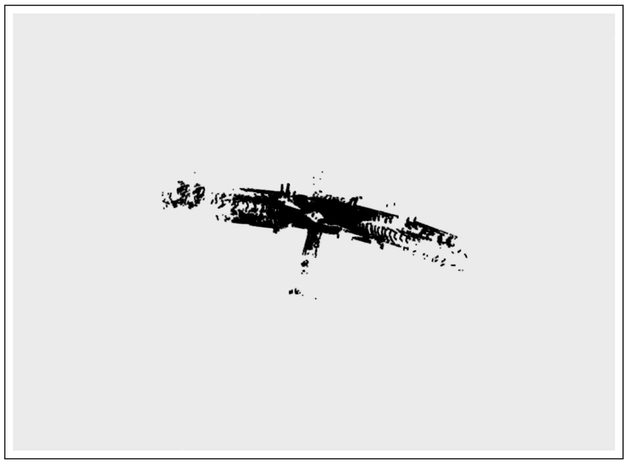


Fig. 3: Point cloud of the next two frames

B. Point to Plane ICP

The goal of ICP is to align two point clouds, the old one (the existing points and normals in 3D model) and new one (new points and normals, what we want to integrate to the exising 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:

$$E = \Sigma_i ||ppd(p_i, q_i, n_i||_2 \to 0$$

where $ppd(p, q, n)$ means that 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 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 point q in the old point cloud

Therefore $ppd(...)$ can be expressed as the dot product of (difference between p and q) and (n):

$$dot(T_{p2q}(p) - q, n) = dot\left((R.p + t) - q, n\right)$$

$$= \left[(R.p + t - q\right]^t .n$$

where T(p) is the rigid transform of point p:

$$T_{p2q} = (R.p + t)$$

where R - Rotation and t - translation.

T is the transform we search by ICP, its purpose is to bring each point p closer to the corresponding point q in terms of point to plane distance. In order to minimize the objective function, the Gauss-Newton method for local minimization.

In the implementation I used the open3d documentation, where they have given the point to plane implementation.

V. Issues

When I tried to build the map, my logic was as follows. Find the transform between Frame 0 and Frame 1 using ICP and convert the points from Frame 0 to Frame 1. Now these points that are from Frame 0 in respect to Frame 1 plus the points of Frame 1 will be combined and multiplied with the Transformation matrix between Frame 1 and Frame 2. All the points were not being saved on a global scale, instead they were always local and all the paths are always therefore with respect to the origin. I believe my logic is correct, but there were some implementation issues. Here is the logic in mathematical terms:

$$((P_0 * T_0 + P_1) * T_1 + P_2) * T_2....$$

This will give the final point cloud with all the points and which can be used to plot the map.

VI. Semantic Point Painting the Map

Now, utilizing the image we want to paint the image. Since we are allowed to use any third party semantic segmentation neural network to predict the semantic labels on every image, I used the one linked by AmrElsersy. Using the pretrained network, which is the BiSeNetv2 model. It is trained on KITTI dataset using TensortRT inference. I was not able to generate results on my original images that I got from the ICP, but I was able to run the code for the dataset that was provided by the person (which is the same dataset, just gives direct coloring instead of using the classical method). The results are shown in figure below:

A. BiSeNet

The network used in this is the BiSeNet. The authors propose a Spatial path to preverse the spatial size of the original input image and encode affluent spatial information. The Spatial Path contains three layers. Each layer includes a convolution, followed by batch normalization and ReLU. This path extracts the output feature maps that is $1/8$ of the original

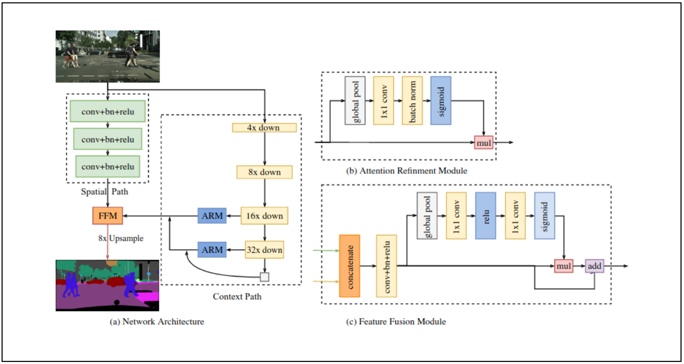


Fig. 4: BiSeNet architecture

image. Encoding seems to be the name of the game for deep learning paper since 2020s, to produce better results.

The authors also talk about Context path, to provide a sufficient receptive field. The context path utilizes lightweight model and global average pooling to provide large refceptive field. The lightweight model can downsample the feature map fast to obtain large receptive field, which encodes high level semantic context information. Then a global average pooling is added on the tail of the lightweight model, which can provide the maxiumum receptive field with global context information. The reason why there is a U shape in Figure 5, is because of the combination of unsampled output feature of global pooling nad features of the lightweight model.

B. Network Architecture

The BiSeNet architecture is shown in figure below. The most important takeaway from the architecture are two things, one is the *Feature fusion module* and *Loss function*. Let us talk about them.

- *Feature fusion module*: The features of the two paths(Spatial Path and Context Path) are different in level of feature representation. As they are in different representation, it is impossible simply sum them. Given the different level of the features, the authors first concatenate the output features of Spatial Path and Context Path. Then they utilize the batch normalization to balance the scales of the features. Now there is a set of combined features, where pooling of concatenated feature to a feature vector and compute a weight vector. This weight vector is used to re-weight the features.
- *Loss function*: The principal loss function is used to supervise the output of BiSeNet. The authors also add two specific auxilary loss functions to supervise the output of context path. All loss functions are Softmax loss. The equation is given as:

$$loss = \frac{1}{N}\sum_i L_i = \frac{1}{N}\sum_i -log\left(\frac{e^{p_i}}{\sum_j e^{p_j}}\right)$$

  where p is the output prediction of the network.

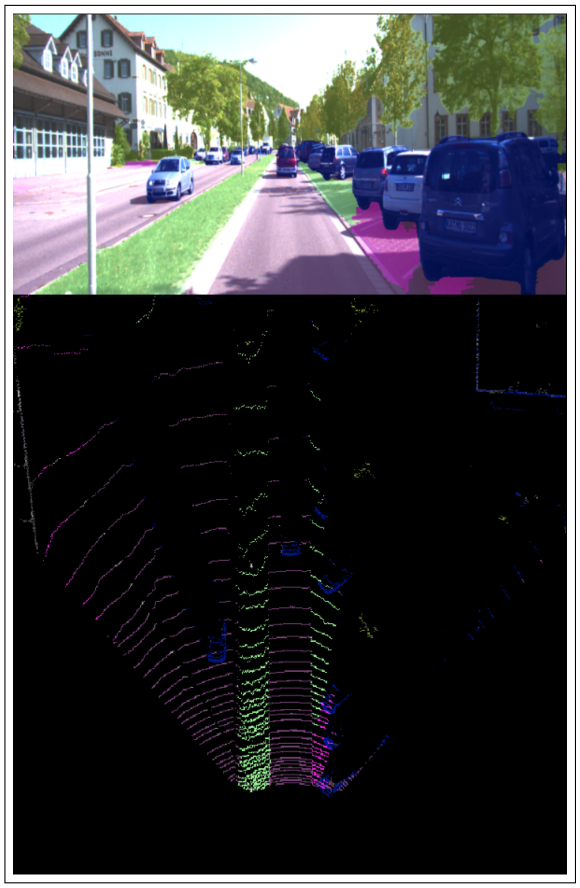


Fig. 5: Caption

- The authors also use a parameter α to balance the weight of the principal loss and auxiliary loss. The joint loss makes the optimizer more efficient.

$$L(X;W) = l_p(X;W) + \alpha \sum_{i=2}^{K} l_i(X_i;W)$$

  where l_p is the principal loss of the concatenated output, X_i is the output feature. l_i is the auxiliary loss. The joint loss is denoted by L

VII. References

1 Open3D
2 Zhang, Zhengyou (1994). "Iterative point matching for registration of free-form curves and surfaces". International Journal of Computer Vision. doi:10.1007/BF01427149
3 Wikipedia
4 P. J. Besl and N. D. McKay, "A method for registration of 3-D shapes," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 14, no. 2, pp. 239-256, Feb. 1992, doi: 10.1109/34.121791.
5 Changqian Yu et al. "BiSeNet: Bilateral Segmentation Network for Real-time Semantic Segmentation" https://arxiv.org/abs/1808.00897