Supplementary Materials
DeepI2P: Image-to-Point Cloud Registration via Deep Classification

Jiaxin Li
Bytedance

Gim Hee Lee
National University of Singapore

A. "Grid Classification + PnP" Method

A.1. Grid Classification

We divide the $H \times W$ image into a tessellation of 32×32 regions and then assign a label to each region. For example, an 128×512 image effectively becomes $4 \times 16 = 64$ patches, and the respective regions are assigned to take a label $l^f \in \{0, 1, \cdots, 63\}$. In the per-point classification, a point taking a label l^f projects to the image region with the same label. Consequently, the grid classification is actually downsampling the image by a factor of 32, and reveals the pixel of the downsampled image to point correspondence. Formally, the grid classification assigns a label to each point, $L^f = \{l_1^f, l_2^f, \cdots, l_N^f\}$, where $l_n^f \in \{0, 1, \cdots, \frac{H \times W}{32 \times 32} - 1\}$.

Label Generation. Grid classification is performed only on points that are predicted as inside camera frustum, i.e. $\hat{l}_i = 1$. The goal of grid classification is to get the assignment of the point to one of the 32×32 patches. We define the labels from the grid classification as:

$$l_i^f = \left\lfloor \frac{p'_{x_i}}{32} \right\rfloor + \left\lfloor \frac{p'_{y_i}}{32} \right\rfloor \cdot \frac{W}{32}, \tag{1}$$

where $\lfloor . \rfloor$ is the floor operator. Note that the image width and height (W, H) are required to be a multiple of 32.

Training the Grid Classifier. As mentioned in Section 4.2, the training of the grid classifier is very similar to the frustum classifier with the exception that the labels are different. Nonetheless, the frustum and grid classifier can be trained together as shown in Fig. 1.

A.2. PnP

Given the grid classifier results, the pose estimation problem can be formulated as a Perspective-n-Point (PnP) problem. The grid classification effectively builds correspondences between each point to the subsampled image, e.g. subsampled by 32. The PnP problem is to solve the rotation R and translation $\mathbf{t}$ of the camera from a given set of 3D points $\{\mathbf{P}_1, \cdots, \mathbf{P}_M \mid \mathbf{P}_m \in \mathbb{R}^3\}$, the corresponding image pixels $\{\mathbf{p}_1, \cdots, \mathbf{p}_M \mid \mathbf{p}_m \in \mathbb{R}^2\}$, and the camera intrinsic matrix $K \in \mathbb{R}^{3 \times 3}$. The 3D points are those classified as within the image by the frustum classification, i.e. $P = \{\mathbf{P}_1, \cdots, \mathbf{P}_M\}$, where $\mathbf{P}_m$ is the point $\mathbf{P}_n \in P : \hat{l}_n^c = 1$. The corresponding pixels are acquired given by:

$$p_{y_i} = \left\lfloor \frac{l_i^f}{W'} \right\rfloor, p_{x_i} = l_i^f - W' p_{y_i}, \text{ where } W' = \frac{W}{32}, \tag{2}$$

and $L^f = \{l_1^f, \cdots, l_M^f\}$, $l_i^f \in [0, (HW/(32 \times 32)) - 1]$ is the prediction from the grid classification. We can effectively solve for the unknown pose $\hat{G} \in \mathrm{SE}(3)$ in the PnP problem after resizing the image into $1/32$ of the original size. Accordingly, the camera intrinsics K' after the resize is obtained by dividing f_x, f_y, c_x, c_y with 32. There are many off-the-shelf PnP solver like EPnP [3], UPnP [2], etc. We apply RANSAC on EPnP provided by OpenCV to robustly solve for $\hat{G}$.

Implementation details. The RANSAC PnP [1] from OpenCV does not require initialization. We set the threshold for inlier reprojection error to 0.6 and maximum iteration number to 500 in RANSAC. Note that we can optionally use the results from RANSAC PnP to initialize the inverse camera projection optimization.

A.3. Experiments

Inverse Camera Projection vs RANSAC PnP [1]. As shown in Table 1, the inverse camera projection solver with 3-DoF performs the best. This verifies the effectiveness of our solver design in Section 5. Nonetheless, the advantage of RANSAC PnP over the inverse camera projection solver is that it does not require initialization, and its performance with 6-DoF is also sufficiently good.

B. Classification Network Details

B.1. Point Cloud Encoder

The input point cloud is randomly downsampled to a size of 20,480. The Lidar intensity values are appended to the x-y-z coordinates for each point. Consequently, the size of the input data is $4 \times 20,480$. During the first sampling-grouping-PointNet operation, the FPS operation extracts $M_1 = 128$ nodes denoted as $\mathfrak{P}^{(1)}$. The grouping procedure is exactly the same as the point-to-node method described in SO-Net [4]. As shown in Fig. 1(a), our PointNet-like module, which produces the feature $P^{(1)}$, is a slight modification of the original PointNet [5]. At the second sampling-grouping-PointNet operation, the FPS extracts $M_2 = 64$ nodes denoted as $\mathfrak{P}^{(2)}$. The grouping step is a kNN-based operation as described in PointNet++ [6]. Each node in $\mathfrak{P}^{(2)}$ are connected to its 16 nearest neighbors in $\mathfrak{P}^{(1)}$. The feature $P^{(2)}$ for each node in $\mathfrak{P}^{(2)}$ is obtained by the PointNet-like module shown in Fig. 1(b). Finally, a global point cloud feature vector is obtained by feeding $\mathfrak{P}^{(2)}$ and $P^{(2)}$ into a PointNet module shown in Fig. 1(c)

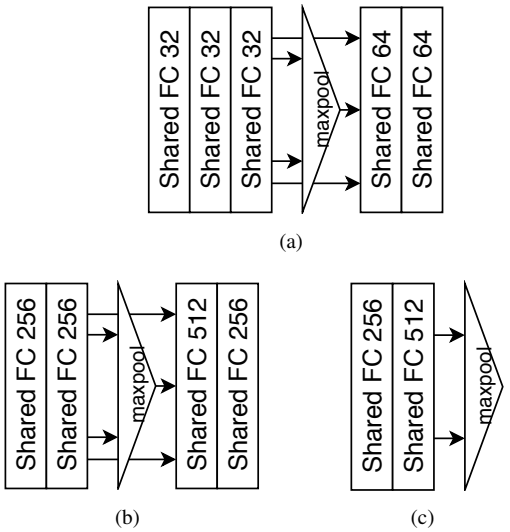

Figure 1. Network details in Point Cloud Encoder. (a) (b) (c) are the PointNet-like network structures used in the encoder.

B.2. Image-Point Cloud Attention Fusion

The first attention fusion module takes the image features $I^{(1)} \in \mathbb{R}^{256 \times H_1 \times W_1}, H_1 = H/16, W_1 = W/16$, global image feature $I^{(3)} \in \mathbb{R}^{512}$, and point cloud feature $P^{(1)} \in \mathbb{R}^{C_1 \times M_1}$ as input. A shared MLP takes $I^{(3)}, P^{(1)}$ as input and produces the weighting $S_{att}^{(1)} \in \mathbb{R}^{(H_1 \cdot W_1) \times M_1}$. The shared MLP consists of two fully connected layers. The weighted image feature $\tilde{I}^{(1)} \in \mathbb{R}^{256 \times M_1}$ is from the multiplication of $I^{(1)}$ with $S_{att}^{(1)}$. $\tilde{I}^{(1)}$ is then used in the Point Cloud Decoder. Similarly, $\tilde{I}^{(2)} \in \mathbb{R}^{512 \times M_1}$ is acquired using shared MLP of the same structure, which takes $I^{(3)}, P^{(2)}$ as input; and outputs the weighting $S_{att}^{(2)} \in \mathbb{R}^{(H_2 \cdot W_2) \times M_2}$.

B.3. Point Cloud Decoder

There are two concatenate-sharedMLP-interpolation processes in the decoder to get $\tilde{P}_{(itp)}^{(2)} \in \mathbb{R}^{C_2 \times M_1}$ and $\tilde{P}_{(itp)}^{(1)} \in \mathbb{R}^{C_1 \times N}$. In both interpolation operations, the k nearest neighbor search is configured as $k = 16$. The shared MLP that takes $[I^{(3)}, \tilde{I}^{(2)}, P^{(3)}, P^{(2)}]$ to produce $\tilde{P}^{(2)} \in \mathbb{R}^{C_2 \times M_2}$ is shown in Fig. 2(a). Similarly, the shared MLP that takes $[\tilde{P}_{(itp)}^{(2)}, \tilde{I}^{(1)}]$ to produce $\tilde{P}^{(1)} \in \mathbb{R}^{C_1 \times M_1}$ is shown in Fig. 2(b). Finally, the shared MLP shown in Fig. 2(c) takes $[P^{(1)}, \tilde{P}_{(itp)}^{(1)} \in \mathbb{R}^{C_1 \times N}]$ to produce the frustum and grid predictions scores.

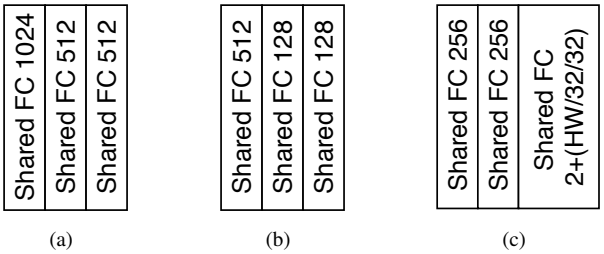

Figure 2. Network details in Point Cloud Decoder. (a) (b) (c) are the shared MLPs used in the encoder.

C. Experiment Details

C.1. Dataset Configurations

In the Oxford dataset, the point clouds are built from the accumulation of the 2D scans from a 2D Lidar. Each point cloud is set at the size of radius 50m, i.e. diameter 100m. Point clouds are built every 2m to get 130,078 point clouds for training and 19,156 for testing. There are a lot more training/testing images because they are randomly sampled within $\pm$10m. Note that we do not use night driving traversals for training and testing because the image quality at night is too low for cross-modality registration. The images are captured by the center camera of a Bumblebee tri-camera rig. The bottom 160 rows of the image is cropped out because those rows are occupied by the egocar. The 800×1280 image is resized to 400×640 and then random/center cropped into 384×640 during training/testing.

In KITTI Odometry dataset, point clouds are directly acquired from a 3D Lidar. Every point cloud in the dataset is used for either training or testing. We follow the common practice of utilizing the 0-8 sequences for training, and 9-10 for testing. In total there are 20,409 point clouds for training, and 2,792 for testing. The top 100 rows of the images are cropped out because they are mostly seeing the sky. The original 320×1224 images are resized into 160×612, and

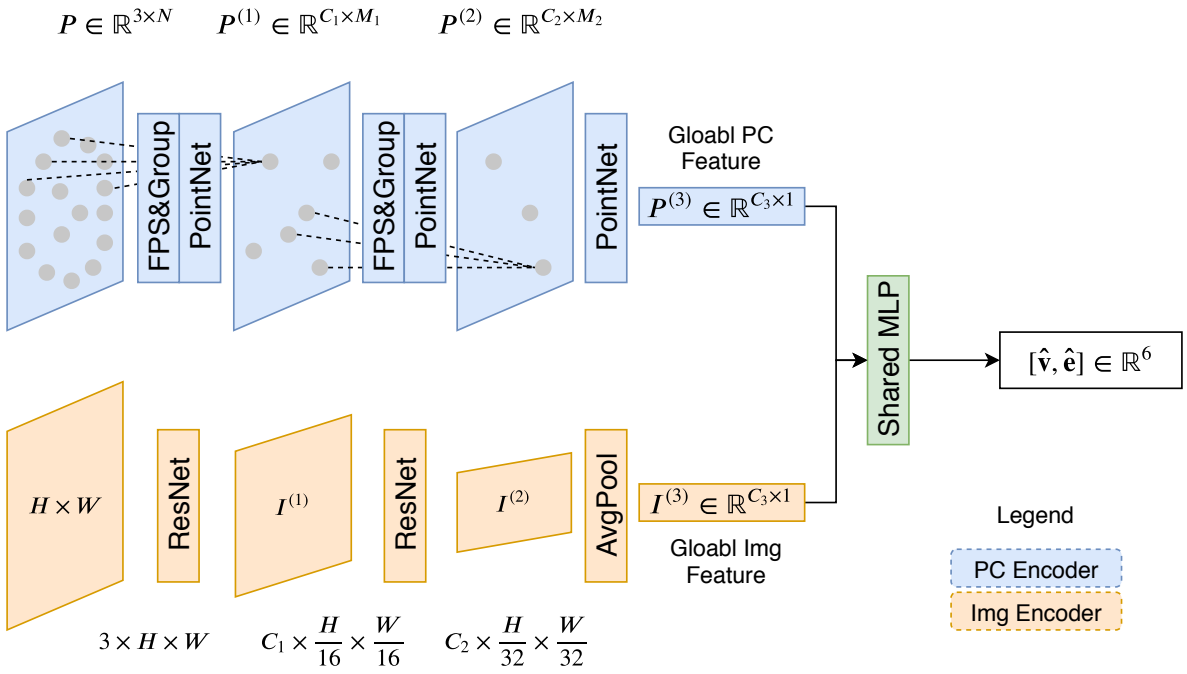

Figure 3. Our network architecture for the baseline method.

then random/center cropped into 160×512 during training/testing.

C.2. "Direct Regression" Method

The direct regression method is a deep network-based approach that directly regresses the pose between a pair of image and point cloud. The network architecture is shown in Fig. 3. The Point Cloud Encoder and Image Encoder are exactly the same as the classification network in our DeepI2P. The global point cloud feature $P^{(3)} \in \mathbb{R}^{512}$ and global image feature $I^{(3)} \in \mathbb{R}^{512}$ are fed into a MLP to produce the relative pose. The relative translation is represented by a vector $\hat{\mathbf{v}} \in \mathbb{R}^3$, while the relative rotation is represented by angle-axis $\hat{\mathbf{e}} \in \mathbb{R}^3$. Given the ground truth $\mathbf{v} \in \mathbb{R}^3$ and rotation $R \in \mathbb{R}^{3\times 3}$, the loss function is given by:

$$\mathcal{L} = \mathcal{L}_{tran} + \mathcal{L}_{rot} = \|\mathbf{v} - \hat{\mathbf{v}}\|_2 + \|f(\hat{\mathbf{e}}) - R\|_F, \quad (3)$$

where $f(\cdot)$ is the funtion that converts the angle-axis representation $\hat{\mathbf{e}} \in \mathbb{R}^3$ to a rotation matrix $\hat{R} \in \mathbb{R}^{3\times 3}$, and $\|\cdot\|_F$ is the matrix Frobenius norm. The training configurations are the same as our DeepI2P, i.e. the image and point cloud are within 10m and additional random 2D rotation is applied to the point cloud.

References

[1] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[2] Laurent Kneip, Hongdong Li, and Yongduek Seo. Upnp: An optimal o (n) solution to the absolute pose problem with universal applicability. In *European Conference on Computer Vision*, pages 127–142. Springer, 2014.

[3] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. Epnp: An accurate o (n) solution to the pnp problem. *International journal of computer vision*, 81(2):155, 2009.

[4] Jiaxin Li, Ben M Chen, and Gim Hee Lee. So-net: Self-organizing network for point cloud analysis. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 9397–9406, 2018.

[5] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

[6] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning

on point sets in a metric space. In *Advances in neural information processing systems*, pages 5099–5108, 2017.