Deep Pose Estimation from 3D Point Clouds

Mustafa Shaikh
Electrical and Computer Engineering, University of California, San Diego

Richard Oliveira
Electrical and Computer Engineering, University of California, San Diego

Abstract

In this project, we perform relative pose estimation of a robot using LiDAR and RGBD camera data in order to determine the movement of the robot through space. Both data streams are fed to our Neural Network architecture which will extract low level information from point clouds for direct pose estimation. Our results will then be compared to traditional methods such as the Iterative Closest Point (ICP) algorithm for accuracy and speed. We train and test our model on trajectory sequences from the Eden dataset. We outperform ICP in terms of absolute pose error on a single test sequence, although we encounter challenges with the dataset which makes the task of odometry difficult.

1 Introduction

1.1 Problem Definition

The problem of pose estimation is critical for accurate localization of an autonomous system. Access to accurate pose information allows reliable, safe and effective control of such systems. A pose $T \in SE(3)$ contains information about the position $\boldsymbol{p} = (x, y, z)^T$ and orientation θ of a robot. This can be achieved by many methods, but one common method is to use onboard sensors such as LiDAR or cameras to obtain point clouds as the robot moves. Then, the optimal pose between the two point clouds is found and this is used to determine the position and orientation of the robot. We aim to develop a system that can take in images from onboard cameras or LiDAR and output pose estimates.

1.2 Motivation

A common approach used for estimating pose is the Iterative Closest Point (ICP) [1] algorithm, which relies on point cloud data that can be captured via LiDAR scans. It is a popular method since it is reliable, simple and can be used to perform pose estimation in real-time settings. One challenge for ICP is the data association problem, in which pairs of points in successive point clouds must be matched before the optimal pose between them can be determined. Traditional methods such as nearest neighbor search solve the data association problem, but these methods are computationally heavy, suffer from slow convergence, and are sensitive to initialization, so we aim to solve this problem by relying on rich representations from feature extraction networks. The main issue our approach aims to solve is the initialization issue - our method will run without any initialization required, and also aims to reduce the computational burden of ICP by estimating the relative pose using a single forward pass through our network at inference time.

1.3 Related Work

There have been some works that address similar problems to the one we aim to solve. Most similarly, the authors of PoseNet [4] implement a convolutional neural network which predicts camera pose relative to a fixed frame of reference. The point of this is to reconstruct 3D images from multiple images taken from different views, but with the camera pose missing. The key differences to our work is that we do point cloud registration, i.e. we find relative poses between successive point clouds that can be obtained from either RGBD images or LiDAR scans. PoseNet can only work with images. The main work in this field is the classical ICP paper [1], which will serve as our baseline. Though this does point cloud registration, it uses classical methods instead of a deep neural network. The only work that implements a neural network for point cloud registration that we are aware of is DeepVCP [3]. The main difference to this work that we present is that we do not use any classical methods inspired by ICP in our approach, whereas the authors use an optimization step to globally align the successive point clouds which is identical to the optimization step in ICP. Adding this step removes the novelty and the benefit of avoiding the computational load and sensitivity to initialization of ICP. Therefore, our work is motivated by the lack of existing works which successfully implement an end to end point cloud registration neural network.

2 Data

2.1 Dataset

We make use of the Eden dataset, which is a synthetically generated dataset of trajectories of a camera moving through a garden environment. The dataset contains RGB and depth images, as well as camera poses, for each of the images in the sequence. Each sequence has 100 images, and there are many sequences available. However, due to computational considerations, we are only able to use 8 of the sequences for training, 2 for validation, and 1 for testing. The images are available in PNG format.

2.2 Data Processing

We extract point clouds from RGB and Depth images in the following way: using the camera instrinsics and projection equations, we convert pixel coordinates to optical frame coordinates, and then to body frame coordinates. We then use extrinsics, i.e. the current estimated robot pose, to project to world frame coordinates. We also normalize the point clouds to 0-mean, unit variance to remove scaling effects. We then prepare two sets of successive point clouds and the relative pose as a single data point. The relative pose is obtained by multiplying the inverse of the pose matrix at time $t+1$ with the pose matrix at time t, i.e. $T_{t+1}^{-1}T_t$. Since the EDEN dataset images have a resolution of 480x640, this leads to over 3 million points in each point cloud, which causes memory issues with GPU. We downsample these points using SIFT as a keypoint detector, and end up with approximately 2000 keypoints. Since 2000 keypoints is a very small number of points, we experiment by randomly adding keypoints to the 2000 selected from SIFT. This way we run models with up to 50,000 keypoints until we run into memory issues with the GPU. Figure 1 shows the keypoints extracted from SIFT. These keypoints are interesting points with lots of texture, which confirms that this method is appropriate to use as a downsampling technique.

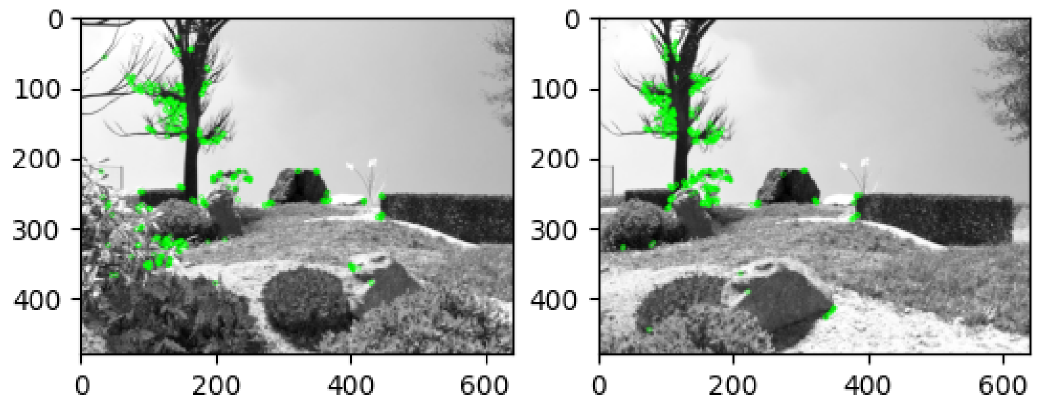


Figure 1: Keypoints extracted using SIFT, successive images

2.3 Challenges with Dataset

We encountered significant challenges with the Eden dataset. Firstly, it is a synthetic dataset which means the data was captured with robots not operating on a standard dynamics model. This led to many sharp turns which are difficult to capture with point cloud registration, and require odometry data which was not available. Secondly, since it is synthetic, we found that many of the generated images lacked texture, and the scenes of the environment were not consistent. For example, between two successive images, the relative position of objects in the images had changed much more than the shift in the camera. This led to issues implementing baseline as well as our neural network model.

3 Approach

We will design a neural network architecture that can take as input two successive point clouds obtained from the LiDAR sensor, as well as semantic information extracted from RGBD images, in order to directly predict the pose between successive scans. This will ultimately be achieved by a regression task at the output. We will make use of pretrained point cloud processing networks [6] to extract latent representations of our point clouds that encode local and global structure. Pretrained networks have learnt deep low level structure in the data which can assist downstream regression tasks such as ours. This extra information should assist the network with the data association portion of pose estimation. We will define a loss function that minimizes the Euclidean distance between the true and predicted position and normalized orientation [4]. Since speed is critical in robotics applications, we will experiment with downsampling the data in two main ways; first, we will reduce the dimensionality of the extracted feature vectors by applying Principal Component Analysis. This will preserve the structure of the data while reducing the dimensions, thereby improving processing speed. Second, we will use smart downsampling of the input point cloud using a key point detection algorithm. This will reduce the density of the point cloud, which will also improve processing speed. The idea is that these methods can reduce computational load while providing accurate pose estimation.

3.1 Deep Pose Estimator: Neural Network for Point Cloud Registration

Here we present details of our model architecture. The backbone of our architecture uses the well known PointNet [6] model which directly processes point cloud data. This network extracts both local and global features from the point clouds by passing it through sub-networks that align the point clouds, but also learn structure from the surrounding point clouds. We use an open source implementation of this network [9]. This information is passed through several layers of MLP and ends up in a single 128-dimensional feature vector. Figure 3 shows the layer that we utilize for feature extraction. We then apply PCA to this feature vector since the dimension is large and the number of datapoints relative to dimension is small, and so the learning process can suffer from the curse of dimensionality. We found that the first 5 dimensions retain most of the information of

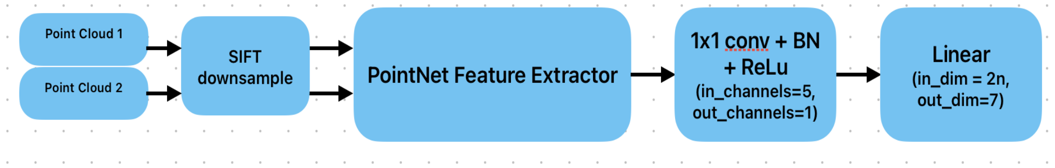

Point Cloud 1
Point Cloud 2
SIFT downsample
PointNet Feature Extractor
1x1 conv + BN + ReLu (in_channels=5, out_channels=1)
Linear (in_dim = 2n, out_dim=7)

Figure 2: Block diagram showing high level model architecture

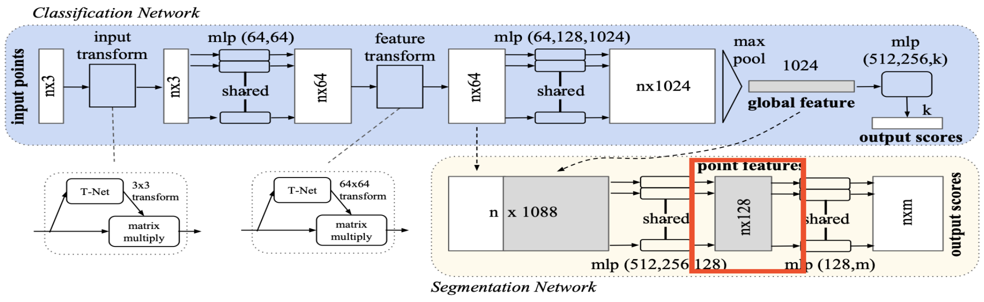


Figure 3: Diagram showing the layer of PointNet utilized for feature extraction

the vector, and so we apply PCA and project to the first 5 principal components. Figure 4 shows the mean square reconstruction loss tapers off around 5 principal components. This means that by using only the first 5 dimensions, we can accurately reconstruct the entire 128-dimensional vector. This is an important dimensionality reduction step which significantly reduces computational load and improves our results. We implement PCA as a frozen layer. After the PCA layer, we pass the 5-dimensional feature vector (one for each point cloud), through a 1x1 convolution layer, i.e. a fully connected layer. This learns further relationships between the points in the point cloud, and outputs a single, dense vector of dimension 2000. This vector encodes all the global and local information in the entire point cloud. This is then passed through a final linear layer which regresses to our custom pose loss. We use a batch size ranging from 2-20 depending on the number of keypoints, and train the model for 10 epochs with a learning rate starting at 0.001 and decreasing with a step learning rate scheduler. Figure 2 shows the high level architecture of our network.

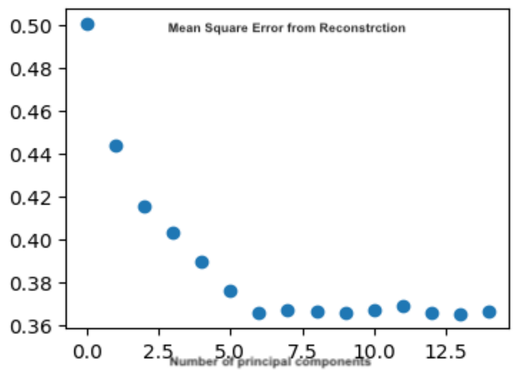


Figure 4: Reconstruction error from PCA by number of principal components

3.1.1 Loss

We use a custom pose regression loss which minimizes the mean squared error between the predicted and actual relative translation, and the predicted and actual relative rotation in quaternion form. As per [4], quaternions are easily converted to a valid rotation by normalizing, and the resulting vector becomes close enough to the ground truth that there is no explicit need to enforce the constraint that it belong on a manifold. In other words, we can treat it as if it operates in Euclidean space and simply use regression to find the optimal relative rotation. This leads to a vector of dimension 7, where the first 3 are the translation component, and the last 4 are the rotation component. The loss is given by:

$$\mathcal{L} = \alpha||\boldsymbol{x} - \hat{\boldsymbol{x}}||_2^2 + \beta||\boldsymbol{q} - \frac{\hat{\boldsymbol{q}}}{||\hat{\boldsymbol{q}}||_2^2}||_2^2 + \lambda||I - AA^T||_F^2 \;\; \alpha, \beta, \lambda > 0 \tag{1}$$

Where $\hat{\boldsymbol{x}}$ is the estimated translation of the model, $\hat{\boldsymbol{q}}$ is the estimated quaternion and A is a matrix which is part of PointNet which we are pushing to be orthogonal as this helps the learning process of PointNet [6]. Lastly α, β and λ are regularization constants. We kept $\lambda = 1$ and experimented with different values of α and β.

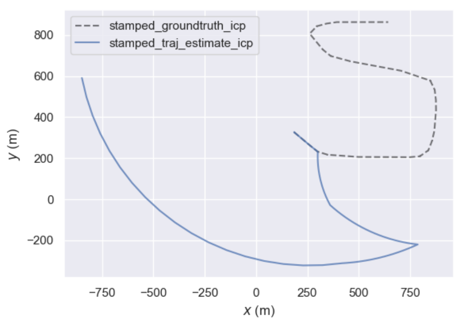
3.1.2 Implementation features

We implement some features that help the learning ability and the training process of the model. First we use 1x1 convolution instead of a simple linear layer, as this retains structure of the point cloud rather than flattening the vector for a linear layer. It learns relationships between the 5-dimensional representation of each point by treating each of the dimensions as a channel. Further, we implement gradient accumulation, in which we run several backpropagation steps before taking an optimization step, as this combats the small gradients encountered due to our small batch size. Finally, we scale the loss by a factor of 100x to make the loss values large enough so that the gradients are large enough for effective learning. We found that learning was very slow when using original loss values which were in the range of 10^{-2}.

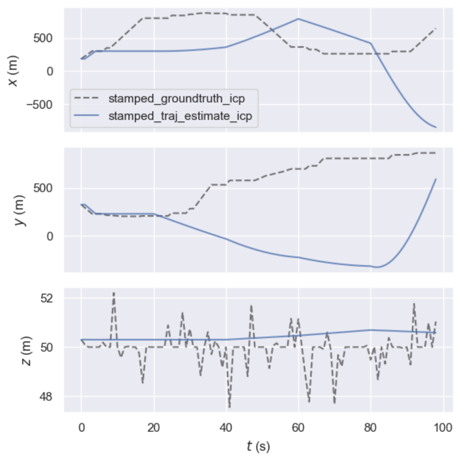
3.2 Baseline - ICP

In order to compare our result to classical state of the art techniques, we implement the well known ICP algorithm using the Open3D implementation to use as a baseline model. Given two successive point clouds from the processed EDEN dataset $\boldsymbol{m}_t \in \mathbb{R}^{M\times 3}$ (*source*) and $\boldsymbol{m}_{t+1}$(*target*) $\in \mathbb{R}^{M\times 3}$ and initializations R_0 and $\boldsymbol{p}_0$ as inputs to the ICP algorithm, we obtain a relative pose ${}_tT_{t+1} \in SE(3)$. Since the ICP algorithm is sensitive to initialization, more specifically the choice of R_0 can drastically impact the algorithms' performance, we decide to initialize ICP with the ground truth absolute pose for the first 5 time stamps in the robots' trajectory. Once we obtain ${}_tT_{t+1}$ from ICP, we compute the robots' pose at time $t + 1$ like so: $T_{t+1} =_t T_{t+1}T_t$.

Figure 5 [8] shows the estimated ICP trajectory overlaid with the ground truth trajectory for 100 time steps. In Figure 1(b) we show how the estimated translation vector $\boldsymbol{p}$ compares with the ground truth translation vector over 100 time steps. As we can see, the ICP results are not accurate compared to ground truth. We tried several ICP implementations, and settled on the Open3D implementation which works well on many datasets. As mentioned earlier, we faced many issues with the Eden dataset and believe this is the reason for poor ICP results, since we tried ICP on other well known odometry datasets such as KITTI and it performed well.

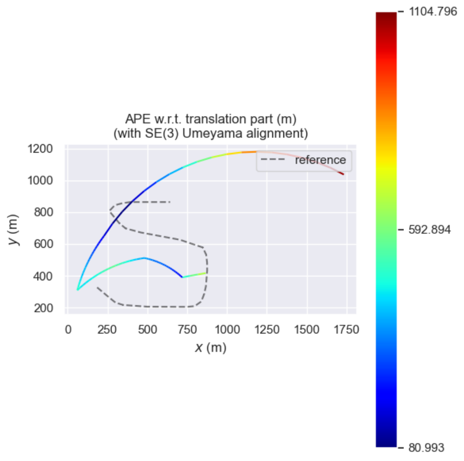


(a) ICP estimated trajectory over 100 time steps

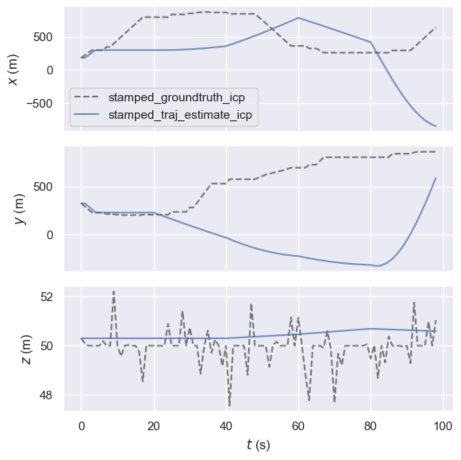


(b) ICP estimated translation over 100 time steps

Figure 5: ICP trajectory and translation estimates over time



(a) Predicted trajectory (with Umeyama alignment) with error colorbar overlaid on ground truth trajectory



(b) Absolute distance error (m) (with $SE(3)$ Umeyama alignment) over 100 time steps

Figure 6

4 Results and Analysis

Given two successive point clouds from the processed EDEN dataset $\boldsymbol{m}_t \in \mathbb{R}^{M\times 3}$ and $\boldsymbol{m}_{t+1} \in \mathbb{R}^{M\times 3}$ as inputs to our trained model, we obtain an estimated pose ${}_tT_{t+1} \in SE(3)$. Once we obtain the estimated pose from our model, we compute the robot pose at time $t+1$. This process is repeated for 100 time stamps.

Our network achieves fairly satisfactory results, outperforming the ICP baseline for a single test trajectory. Note that due to computational and time constraints, we were only able to analyze the predicted trajectory on a single test sequence. We had limited memory availability and our training set only had approximately 8 sequences. We achieve lower error than ICP as can be seen from figure 10, which shows that the absolute pose error distribution across the entire trajectory is lower for our model than for ICP. This means that we consistently achieve lower error than baseline for most parts of the trajectory.

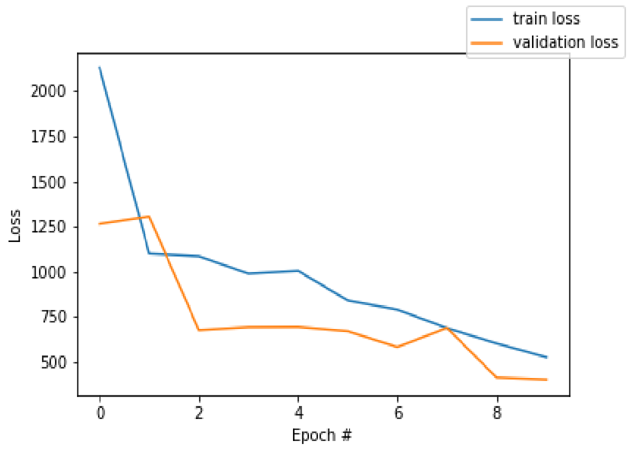


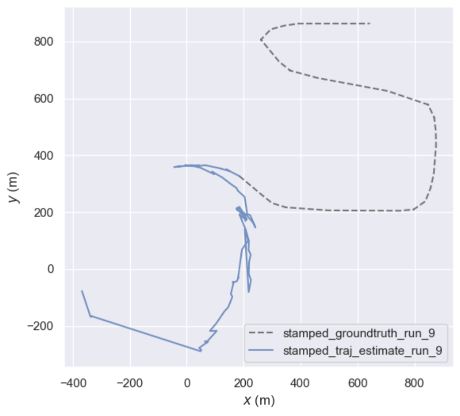
Figure 7: Loss curve for best performing model

It should be noted that from figure 8, we see that the trajectory from our model as well as ICP is not aligned correctly, and this is a challenge we persistently encountered when working with this dataset, despite thorough debugging of the dataset generation process. We are currently still looking into this issue.

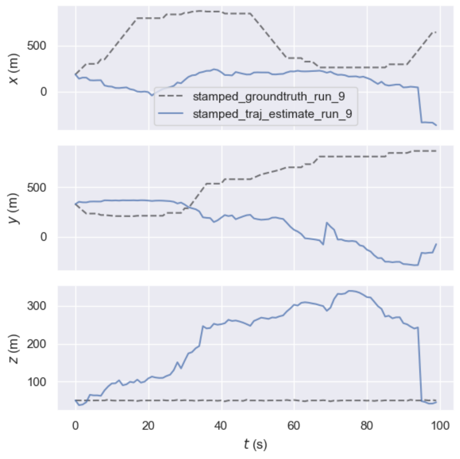
From figure 7, we see that the model is indeed learning, as the training and validation loss consistently decreases. For conciseness, we only show the loss curve and results for the best performing model. To achieve better results, we would train with the entire dataset rather than just 8 sequences, and this would bring large improvements in performance.

4.1 Best model

The configuration for the best performing model is as follows: we use SIFT downsampling to obtain 2000 keypoints, and augment this with an additional 60,000 points uniformly sampled from the original image. We scale the loss by 100x, and use a batch size of 2 (i.e. 2 sets of successive point clouds). We accumulate gradients for 10 steps before an optimization step, and use an initial learning rate of 0.001 and decrease this with a step learning rate scheduler which decreases the learning rate by a factor of 2 every 10 epochs. We only train for 10 epochs as we find the model does not learn much more after this point, and for computational load considerations as operate with limited GPU time. We use $\alpha, \beta = 1$ in the loss function, i.e. equal weight to position and orientation loss. Additionally, we freeze the pointnet backbone and do not update the parameters of this network during training. The model takes approximately 5-10 minutes per epoch to train using a single GPU.



(a) Model estimated trajectory over 100 time steps

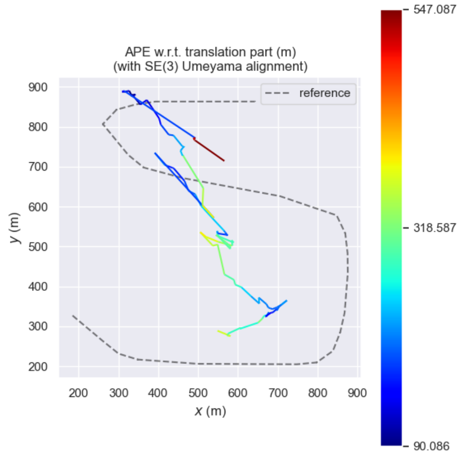


(b) Model estimated translation over 100 time steps

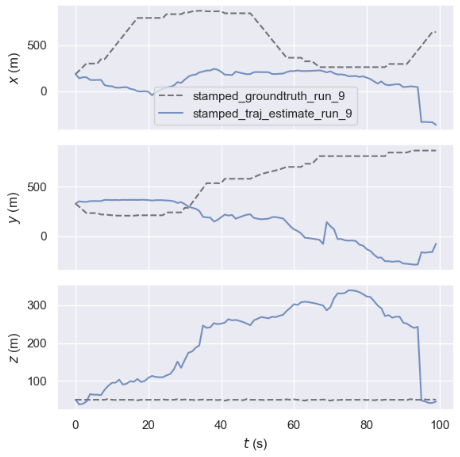
Figure 8: Model predicted trajectory and translation estimates over time

Below we see the absolute pose error error with respect to translation (metres) (with $SE(3)$ Umeyama alignment) in Figure 9. This plot was obtained by relying on the packages in [8]. The alignment relies on the *Kabsch-Umeyama* algorithm. It computes the centroid of two trajectories, shifts one to the other and computes an optimal rotation such that the two trajectories are aligned. In other words, the *Kabsch-Umeyama* algorithm computes the optimal alignment between two trajectories. The optimal alignment does not preserve the structure of the original trajectory, and distorts it during the alignment process. This makes the trajectory appear worse after alignment that without alignment.

Closer inspection of the model predicted trajectory shows that the model struggles with sharp turns, similarly to ICP baseline. There is lots of noise in the dataset, and the alignment is difficult to achieve when the point clouds are far apart. Nonetheless, it should be noted that our model predicts trajectories without any initialization whatsoever unlike the traditional methods such as ICP. Though the alignment is slightly off, the shape of the trajectory is quite close to the real one, although it is truncated as the model output several noisy predictions in the middle of the trajectory.

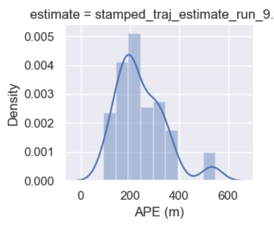


(a) Predicted trajectory (with Umeyama alignment) with error colorbar overlaid on ground truth trajectory

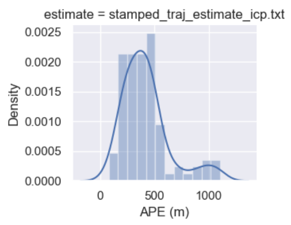


(b) Absolute distance error (m) (with $SE(3)$ Umeyama alignment) over 100 time steps

Figure 9



(a) Model pose error histogram over 100 time steps



(b) ICP pose error histogram over 100 time steps

Figure 10

5 Conclusion

We present an end to end point cloud registration method using deep neural networks. We remove the need for initialization and therefore sensitivity of traditional algorithms such as ICP, and quantitatively outperform ICP in terms of absolute pose error. Our method also runs inference fast and does not require costly nearest neighbour searches followed by optimization steps at each iteration. Instead, we compute the optimal pose with a single forward pass through our network. We outperform ICP in terms of absolute pose error across a single test trajectory. In future work, we will test across multiple test trajectories and compare the error. We will train on a much larger data sample if we have access to more compute capability. We will also expand the network to increase its learning capability, and try training our network on other datasets.

6 Implementation

We implemented the ICP baseline, as well as the point cloud creation process from RGB and depth images from the Eden dataset (i.e. the DataLoader). Additionally, we have also converted robot trajectories into the correct data format such that they would be able to be used in [8] for plotting and visualization of the results. The only existing code we have used is an open source implementation of PointNet in Pytorch [9]. We have used this as a backbone to our network, and added our own layers on top of this (preprocessing, PCA, 1x1 conv, linear layer).

References

[1] K.S. Arun, T. S. Huang, and S. D. Blostein, "Least square fitting of two 3-D point sets," IEEE Trans. Patt. Anal. Machine Intell, vol. PAMI-9, no. 5, pp. 698-700, 1987

[2] David Lowe, Object Recognition from Local Scale-Invariant Features

[3] Lu, Weixin and Wan, Guowei and Zhou, Yao and Fu, Xiangyu and Yuan, Pengfei and Song, Shiyu, DeepVCP: An End-to-End Deep Neural Network for Point Cloud Registration

[4] Alex Kendall and Matthew Grimes and Roberto Cipolla, PoseNet: A Convolutional Network for Real-Time 6-DOF Camera Relocalization

[5] Yu Xiang and Tanner Schmidt and Venkatraman Narayanan and Dieter Fox, A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes

[6] Charles R. Qi and Hao Su and Kaichun Mo and Leonidas J. Guibas, PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation

[7] Le, Hoang-An and Das, Partha and Mensink, Thomas and Karaoglu, Sezer and Gevers, Theo, EDEN: Multimodal Synthetic Dataset of Enclosed garDEN Scenes

[8] Grupp, Michael, Python package for the evaluation of odometry and SLAM

[9] Fen, Xia, PointNet implementation in Pytorch, https://github.com/fxia22/pointnet.pytorch?tab=readme-ov-file