Deep Iterative Closest Point Based Simultaneous Localization and Mapping

Santhosh Vasa
Khoury College of Computer Science
Northeastern University
vasa.s@northeastern.edu

Skanda Akkihebbal Prasanna
Dept. Electrical and Computer Eng.
Northeastern University
akkihebbalprasanna.s@northeastern.edu

Ravina Lad
Khoury College of Computer Science
Northeastern University
lad.ra@northeastern.edu

***Abstract*— Simultaneous Localization and Mapping (SLAM) is a crucial process in robotics and autonomous vehicles, enabling them to navigate in unknown environments. Traditional methods like Iterative Closest Point (ICP) and Deep Point Cloud Registration are commonly used for aligning raw Lidar data with known maps, but they may not be robust in challenging environments with dynamic objects or sparse and noisy data. Recently, deep learning-based approaches like Deep Closest Point (DCP) have shown promising results in solving SLAM problems. By using DCP, the system learns feature representations of point clouds that are more robust to noise and can handle dynamic environments. The SLAM system discussed in this context uses both traditional ICP and DCP algorithms, with ICP being used for data association and transformation, and DCP overcoming the limitations of ICP, such as drift over time. The system also uses loop closure detection and pose graph optimization techniques for accurate and robust localization and mapping. The integration of deep learning-based approaches in SLAM leads to more accurate and robust localization and mapping, especially in challenging environments with dynamic objects or sparse and noisy data, and reduces drift over time.**

I. INTRODUCTION

SLAM, or simultaneous localization and mapping, is a problem in robotics and computer vision that involves constructing a map of an unknown environment while simultaneously localizing the robot within that map. The goal of SLAM is to enable a robot to navigate in an unknown environment without external sensors or prior maps. Smoothing and filtering are two important concepts in SLAM. Smoothing refers to the process of refining the estimated trajectory of the robot after all the measurements have been collected. Filtering refers to the process of estimating the robot's pose while measurements are being collected.

In filtering, two popular approaches are the extended Kalman filter (EKF) and particle filter. The EKF is a recursive algorithm that estimates the state of a system with a nonlinear dynamic model and nonlinear measurements. It approximates the nonlinear function using the first-order Taylor expansion. On the other hand, the particle filter represents the belief with a set of weighted samples, where each sample represents a possible state of the system.

In smoothing, one approach is pose graph optimization, which involves optimizing the trajectory of the robot by minimizing the error between the estimated trajectory and the measurements collected by the robot. This optimization problem can be formulated as a graph, where nodes represent robot poses and edges represent the constraints between poses. The constraints are obtained from the measurements, and the optimization is performed using techniques such as nonlinear least squares. Pose graph optimization is used to extract trajectories estimated using a complete set of measurements, which can improve the accuracy of the estimated trajectory.

Traditionally, methods like Iterative Closest Point (ICP) and Deep Point Cloud Registration are commonly used for aligning raw Lidar data with known maps. However, these methods may not be robust in challenging environments with dynamic objects or sparse and noisy data.

Recently, deep learning-based approaches like Deep Closest Point (DCP) have shown promising results in solving SLAM problems. By using DCP, the system learns feature representations of point clouds that are more robust to noise and can handle dynamic environments. DCP uses a neural network to learn feature representations of point clouds and aligns them using an optimization algorithm. The neural network takes two point clouds as input and outputs a transformation matrix that aligns the two point clouds.

In the SLAM system discussed in this context, both traditional ICP and DCP algorithms are used, with ICP being used for data association and transformation, and DCP overcoming the limitations of ICP, such as drift over time. The system also uses loop closure detection and pose graph optimization techniques for accurate and robust localization and mapping.

The system first uses ICP to perform data association and transformation between the raw Lidar data and the known map. However, ICP is prone to drift over time and may not work well in challenging environments. Therefore, DCP is used to refine the alignment between the Lidar data and the map. DCP can learn feature representations of point clouds that are more robust to noise and can handle dynamic environments, leading to more accurate alignment and reduced drift over time.

In addition to data association and alignment, the system also uses loop closure detection to detect when the robot returns to a previously visited location. This allows the system to correct any accumulated errors in the estimated trajectory and map. Finally, the system uses pose graph optimization to refine the estimated trajectory and map by minimizing the error between the estimated trajectory and the measurements collected by the robot. This optimization problem can be

formulated as a graph, where nodes represent robot poses and edges represent the constraints between poses.

Fig. 1. CARLA Environment

Fig. 2. Data Collection From CARLA

II. DATASET

A. Dataset Utilized from KITTI

The KITTI dataset is a widely used dataset in computer vision and robotics research, particularly for the task of odometry and localization. It includes 21 sequences (numbered from 00 to 20) of raw sensor data captured from a vehicle equipped with a Velodyne Lidar, a camera, and other sensors. The dataset provides 3D point cloud data along with camera images, GPS/IMU measurements, and ground truth odometry information. The data is captured in various urban and highway environments, including challenging scenarios such as sharp turns and occlusions. The total size of the dataset is 80 GB, and it is commonly used for benchmarking and evaluating odometry and localization algorithms.

B. Dataset Collected from CARLA

To enhance the KITTI dataset, we installed the Carla simulator 0.9.14 and configured a simulation environment. An interface was developed using PyGame, which allows for loading and controlling a vehicle in the simulated environment. The team collected their own 3D point cloud data from the Lidar sensor in the Carla environment. Additionally, they extracted the initial ground truth (IGT) from the Carla environment, which includes the position of the vehicle in the environment in terms of x, y, z coordinates, and the orientation of the vehicle in terms of roll, pitch, and yaw (Euler angles). This IGT can be used as a reference for evaluating the accuracy of the odometry and localization algorithms developed using the KITTI dataset.

III. Related Work

A. Categories in SLAM

Previous works with the SLAM methods are majorly divided in to two categories, filtering based methods and smoothing based methods. Filtering-based SLAM methods, such as Extended Kalman Filter (EKF) SLAM and its variants, are typically implemented in an online manner. This means that they estimate the robot's pose and the map incrementally as new sensor measurements arrive. The poses have a prediction phase which is based on readings from relative sensors and an update step where the measurements are corrected in real time based on readings from absolute sensors. The estimated map is updated based on the most recent sensor measurements, and the estimate of the robot's pose is propagated forward in time.

On the other hand, smoothing-based SLAM methods, such as Graph-SLAM and its variants, are typically implemented in an offline manner. This means that they estimate the robot's pose and the map based on a batch of sensor measurements, rather than incrementally as new measurements arrive. These methods first collect a large number of sensor measurements, and then estimate the robot's trajectory and the map by minimizing a cost function that includes all the measurements. The estimated map and trajectory can then be refined by iterative optimizing the cost function, potentially improving the accuracy of the estimates. The iterative optimizing can take the form of post graph optimization and have loop closure identification in the form of landmarks. The system then makes use of the landmarks to optimize the map. Our approach falls under this category and uses post graph optimization and loop closures.

B. Odometry

Odometry prediction is an important component of SLAM systems that rely on motion models to estimate the pose of a robot between consecutive sensor measurements. The odometry estimation can be achieved by one or more of the relative sensors such as Wheel Odometry encoders, Inertial Measurement Unit, Visual Odometry based on images, Point Cloud Registration odometry and more.

ICP (Iterative Closest Point) is a widely used technique in SLAM for aligning two point clouds to estimate the pose change between consecutive time steps. 1. Point-to-Point and Point-to-Plane ICP: Two main variants of ICP exist: point-to-point ICP and point-to-plane ICP. In point-to-point ICP, the algorithm seeks to minimize the Euclidean distance between corresponding point pairs. Point-to-plane ICP, on the other hand, minimizes the orthogonal distance between points and the tangent plane of their corresponding points. The point-to-plane variant often provides better results when the point

clouds have uneven sampling densities or contain planar surfaces. 2. ICP with Feature Matching Feature-based ICP methods incorporate additional information, such as local geometric features or descriptors, to improve the correspondence search and convergence rate. These methods can be more robust and accurate, particularly in the presence of repetitive structures or occlusions.

C. Deep learning based Odometry

Deep learning has shown promise in improving point cloud registration by addressing some of the limitations of ICP. There are multiple approaches that have been surfaced in the deep learning research to achieve point cloud registration. The methods include point correspondences (similar to ICP) and correspondences free methods which do not inherently find any correspondences. The correspondences free methods use neural network architectures like pointnet, pointnet++ and graph based methods such as DGCNN. This feature representation can capture complex geometric structures and facilitate robust matching between the point clouds. The method we followed is similar to this approach. These approaches extract feature descriptors that are invariant to the order of input points and are sensitive to motion and are robust to outliers and noise.

D. Loop Detections

1. SegMatch [26] is a loop closure detection method that segments 3D LiDAR point clouds into distinct geometric structures and matches them across different scans. The method uses a combination of supervised and unsupervised learning techniques to segment and describe the point clouds, making it robust to different types of environments. 2. 3D NDT [27]: The Normal Distributions Transform (NDT) is a point cloud registration algorithm that represents the environment as a continuous probability density function. The 3D NDT method extends the NDT concept to work with 3D point clouds and can be used for loop closure detection by matching these probabilistic representations of the environment. 3. PointNetVLAD [28] is a deep learning-based approach for place recognition and loop closure detection in 3D point clouds. The method uses the PointNet architecture for point cloud feature extraction and combines it with the NetVLAD layer for global descriptor generation. PointNetVLAD is capable of learning discriminative features from large-scale 3D point cloud datasets, making it robust to various environments and conditions.

E. Pose Graph Optimization

1. g2o [29](General Graph Optimization) is a flexible and efficient framework for optimizing various types of graphs, including pose graphs. It has been used extensively for optimizing pose graphs built from 3D LiDAR point cloud data. The framework is designed to handle different types of constraints and is highly extensible, making it suitable for various applications. 2. iSAM2 [30] (incremental Smoothing and Mapping) is an incremental optimization algorithm for pose graph optimization in both 2D and 3D environments. It is designed to handle large-scale, real-time mapping tasks using LiDAR point clouds. iSAM2 uses the Bayes tree data structure to exploit the sparse nature of the SLAM problem, resulting in a computationally efficient algorithm. 3. Ceres Solver [31] is a general-purpose optimization library that has been used for pose graph optimization in 3D LiDAR point cloud data. It supports various types of constraints and loss functions, making it suitable for a wide range of applications. Ceres Solver has been used in the Google Cartographer project for optimizing the pose graph built from 3D LiDAR data.

IV. METHODOLOGY

A. Classical Registartion - ICP

ICP can be used to estimate the motion of a robot based on the point cloud data obtained from a sensor. This is done by computing the difference between the current point cloud and the previous point cloud, and then aligning them using ICP. The resulting transformation is then used to update the robot's pose. This approach is commonly referred to as "ICP odometry". ICP variants depend on calculating the closest points between two datasets, which takes a lot of time as the number of points increases. ICP requires a good initial alignment to converge to the correct solution, as it is prone to getting stuck in local minima. ICP is not compatible with deep learning methods because it relies on specific point matches rather than continuous values, preventing the use of learned features for alignment. ICP can fail even if the initial alignment between the source and target point clouds is good due to the presence of outliers, noise, and non-rigid deformations. These issues can lead to incorrect correspondences between the source and target points, resulting in an incorrect registration. Therefore, there is a need for an algorithm that can handle more complex data by incorporating robustness to outliers, noise, and deformations in the registration process. This will help reduce the drift that's accumulated over the period of time. By leveraging the power of deep learning, PCRnet addresses many of the shortcomings of traditional ICP algorithms.

B. Point Cloud Registration Network

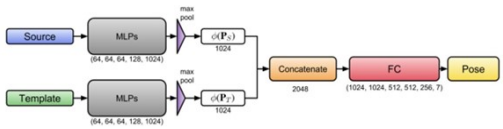

Fig. 3. PCRNet Model Architecture

This section introduces the PCRNet architecture. A block diagram of the architecture is shown in figure. The point cloud data obtained from a sensor is referred to as the source and the point cloud corresponding to the known model of the object to be registered is referred to as the template. The model consists of five multi-layered perceptrons (MLPs) similar to

the PointNet architecture having size 64,64, 64, 128, 1024. The MLPs are arranged similar to a Siamese architecture [17]. Both source PS and template PT are given as input to the MLPs which are arranged in Siamese architecture and symmetric max-pooling function is used to find the global feature vectors (PS) and (PT). Weights are shared between MLPs used for source and template. The global features are concatenated and given as an input to a number of fully connected layers. In this work, we choose five fully connected layers, as they seemed to be sufficient enough for robust performance. We tried using lesser number of FC layers, but the performance of the network was poor. The FC layers shown by the red block in figure above has five hidden layers, 1024, 1024, 512, 512, 256, and an output layer of size 7 whose parameters will represent the estimated transformation T. The first three of the output values we use to represent the translation vector t R3 and last four represents the rotation quaternion q R4 , q T q = 1.

Loss Function: The aim of the loss function used to train registration networks should be minimization of distance between the corresponding points in source and template point cloud. This distance can be computed using Earth Mover Distance(EMD) function, where PT is the template point cloud

$$\text{EMD}(\mathbf{P}_S^{\text{est}}, \mathbf{P}_T) = \min_{\psi:\mathbf{P}_S^{\text{est}}\to\mathbf{P}_T} \frac{1}{|\mathbf{P}_S^{\text{est}}|} \sum_{x\in\mathbf{P}_S^{\text{est}}} \|x - \psi(x)\|_2,$$

Fig. 4. PCRNet Loss function

and PS(est) is the source point cloud PS.

C. Loop Detections

Loop closure detection is crucial for correcting drift and errors accumulated during the car's motion. The Scan Context algorithm is a loop closure detection method used in SLAM (Simultaneous Localization and Mapping). It is designed to be fast and efficient, making it suitable for real-time applications. The primary goal of loop closure detection is to identify when the robot revisits a previously explored area, enabling the correction of accumulated errors in the robot's pose estimates and the map.

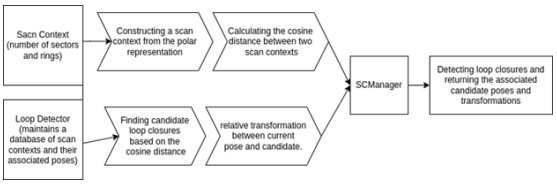

Fig. 5. Loop Detection Algorithm

Here's an major components of the Scan Context algorithm: 1. Preprocessing: Given a 3D point cloud (typically generated from a LiDAR scan), the algorithm preprocesses the data to retain only the most important points. This can be achieved through downsampling, filtering, or other similar techniques. 2. Polar-coordinate-based representation: The preprocessed point cloud is converted into a spherical coordinate representation (in 3D). The origin is set to the robot's position during the scan, and the points are represented by their distance and angle (or elevation angle) with respect to the origin. 3. Scan context construction: The polar or spherical space is divided into a grid of sectors (angular divisions) and rings (distance divisions). For each cell in the grid, the algorithm counts the number of points that fall within its boundaries. This results in a 2D histogram-like representation called the scan context. 4. Scan context comparison: To compare two scan contexts, the algorithm calculates their cosine distance. The cosine distance is a measure of similarity between two matrices (or vectors) that takes into account their orientations rather than their magnitudes. If the cosine distance between two scan contexts is below a certain threshold, they are considered similar, indicating a potential loop closure. 5. Candidate loop closure: When a new scan context is added, the algorithm compares it to all previously stored scan contexts. If a candidate loop closure is found, the algorithm returns the associated pose and the relative transformation between the current pose and the candidate pose.

Scan Context's parameters: Ring: 20, Sector: 60 The number of ringkey candidates: 30 Correct Loop threshold: 0.17 for 09, 0.15 for 14, and 0.11 for all others

Observations:

1. If the loop threshold is too low (0.07 in the below figure), no loops are detected and thus the odometry errors cannot be reduced.

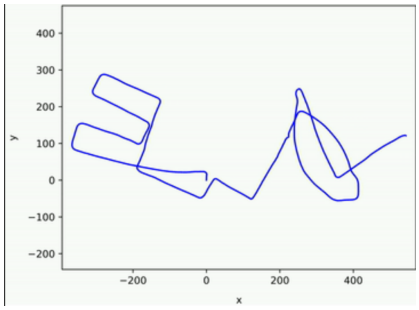

Fig. 6. Loop Detection with Low Threshold

2. If the loop threshold is high (0.20 in the below figure), false loops are detected and thus the graph optimization failed. 3. The Scan Context algorithm sometimes has trouble detecting loop closures when there is a lane change. Although Scan Context is designed to be invariant to rotation, it assumes that the environment is mostly flat and has limited vertical structure. When there are significant lane-level changes, this assumption no longer holds, causing the method to struggle in detecting loops.

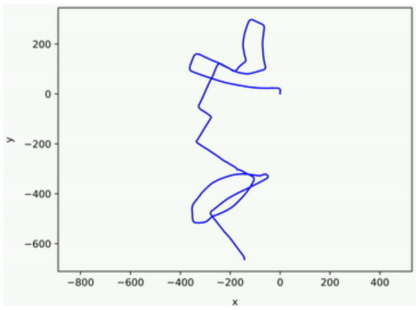

Fig. 7. Loop Detection with High Threshold

D. Pose Graph Optimization

The Pose Graph Optimization is implemented as follows:

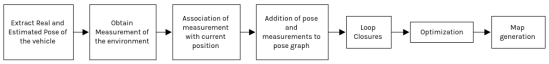

Fig. 8. Flow of Pose Graph Optimizer

1.Initialize the PoseGraphManager class. 2.Define the prior covariance using gtsam.noiseModel.Diagonal.Sigmas with small values. 3.Define the constant covariance for odometry and loop factors using np.array and gtsam.noiseModel.Diagonal.Sigmas respectively. 4.Create an empty nonlinear factor graph using gtsam.NonlinearFactorGraph(). 5.Create an empty values object using gtsam.Values(). 6.Define the Levenberg-Marquardt optimization parameters using gtsam.LevenbergMarquardtParams(). 7.Create a Levenberg-Marquardt optimizer object using gtsam.LevenbergMarquardtOptimizer() with the empty factor graph and values object. 8.Set the current node index and previous node index to zero. 9.Set the current 4x4 homogeneous transformation matrix to the identity matrix using np.eye(4). 10.Insert a prior factor into the factor graph using gtsam.PriorFactorPose3 with the current node index, current pose, and prior covariance. 11.Insert the current pose into the values object using gtsam.symbol() and gtsam.Pose3(). 12.Use addOdometryFactor() to add a between factor between the previous node and the current node using gtsam.BetweenFactorPose3, the previous node index, the current node index, the odometry transform, and the odometry covariance. 13.Use addLoopFactor() to add a between factor between a loop node and the current node using gtsam.BetweenFactorPose3, the loop node index, the current node index, the loop transform, and the loop covariance. 14.Use optimizePoseGraph() to optimize the factor graph using Levenberg-Marquardt optimization. 15.Retrieve the optimized pose using getGraphNodePose() and update the current homogeneous transformation matrix with the optimized pose. 16.Repeat steps 12-15 for all subsequent nodes in the graph. 17.Use the optimized pose graph to generate a map of the environment.

V. Analysis and Results

The accuracy metrics for point cloud registrations typically involve calculating the difference between the ground-truth and predicted transformation matrices, which include the rotation and translation components.

Rotation error: To measure the quality of the rotations, we can use metrics such as mean squared error and Frobenius norm. Mean squared error measures the average difference between the ground-truth and predicted rotation matrices element-wise, while the Frobenius norm calculates the Euclidean distance between the two matrices.

Translation error: To measure the accuracy of the translations, we can use metrics such as mean squared error and Euclidean norm. Mean squared error measures the average difference between the ground-truth and predicted translation vectors element-wise, while the Euclidean norm calculates the Euclidean distance between the two vectors.

A. ModelNet40

The ModelNet40 dataset is a collection of 3D CAD models from 40 object categories, including furniture, vehicles, and household items. The models are uniformly aligned and normalized to the same scale, and come with class labels and viewpoints. The dataset is commonly used in computer vision research for tasks such as shape classification, segmentation, and retrieval, as well as for point cloud registration. We obtained the following results when tested on modelnet40:

Method	Test Loss	Rotation Error	Translation Error
PCRNet	0.0687	14.65	0.0065
ICP	0.0473	12.48	0.1787

TABLE I
Comparison of Test Loss and Errors for PCRNet and ICP

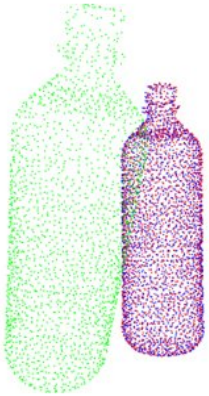

Fig. 9. PCRNet on input1

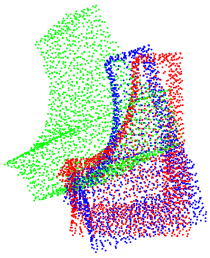

Fig. 10. PCRNet on input2

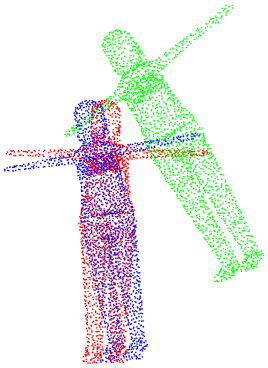

Fig. 11. PCRNet on input3

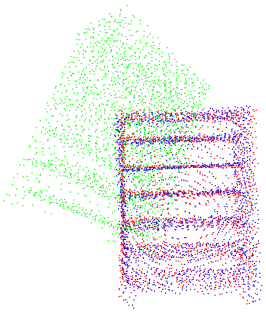

Fig. 12. PCRNet on input4

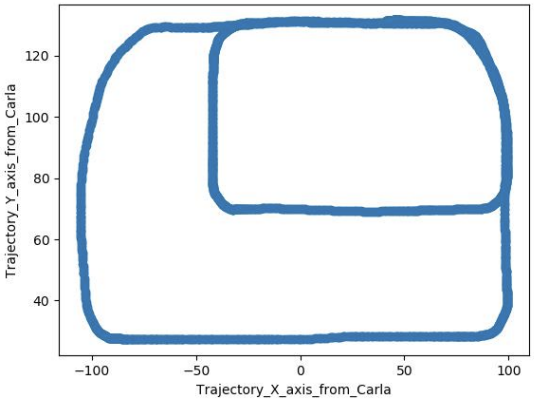

Fig. 13. Input to ICP

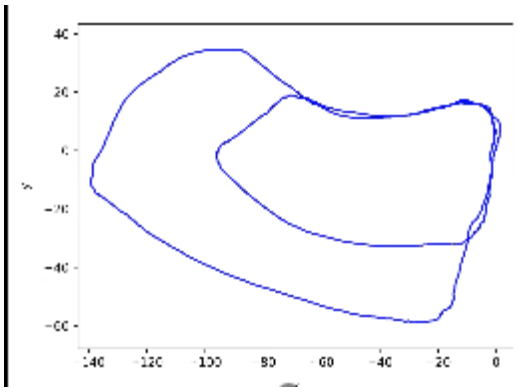

Fig. 14. ICP output with pose graph optimization

B. Synthetic Dataset from CARLA

The use of ICP (Iterative Closest Point) with pose graph optimization has demonstrated good performance on simulated data in the CARLA environment, as depicted in the accompanying figures. However, despite the incorporation of the pose graph optimization correction step, the output still requires further enhancement. In other words, while the approach is effective, it can still benefit from additional refinement to achieve even better results.

C. Synthetic Dataset from CARLA

While PCRNet performs effectively on the ModelNet40 dataset, it doesn't exhibit good generalization when it's presented with synthetic street data generated from CARLA simulator that's different from ModelNet40. Specifically, when PCRNet is trained on ModelNet40 and then tested on synthetic street data, it produces arbitrary or unpredictable trajectories. This suggests that PCRNet may have difficulty adapting to the new visual characteristics and environmental features that are present in the synthetic street data, which can lead to suboptimal performance in these scenarios.

To address the issue of poor generalization of PCRNet when presented with synthetic street data, it is necessary to perform either a fine-tuning phase or a training phase from scratch. This means that the PCRNet model needs to be further trained using the synthetic street data in order to adapt its parameters and learn the relevant features and patterns that are specific to this new data. Fine-tuning involves training the model on the new data while keeping some of the previously learned parameters fixed, while training from scratch involves re-initializing all the parameters and training the model entirely on the new data. Both of these approaches can help the model better generalize to the synthetic street data and improve its performance on this task.

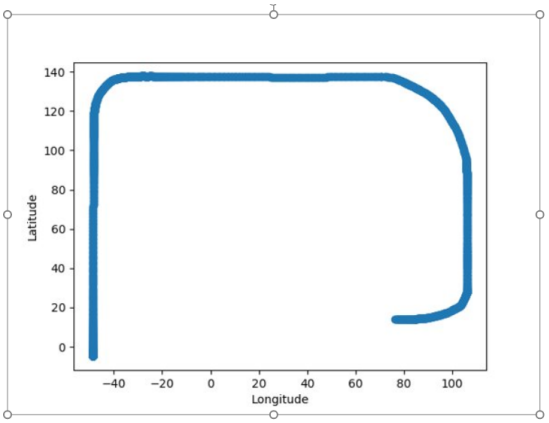

Fig. 15. Data collection trajectory

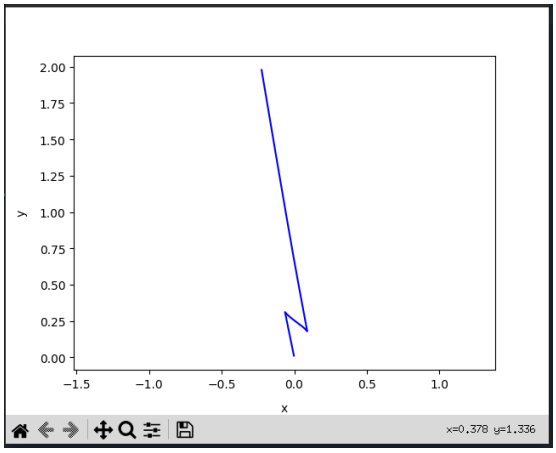

Fig. 16. Result on initial data

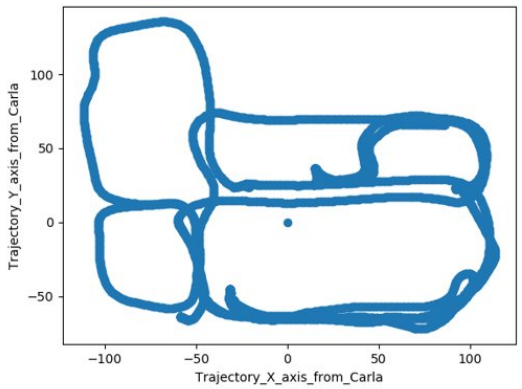

Fig. 17. Updated data collection trajectory

textbfInitial data collection: As a starting point, we used pygame to control the car and generate basic trajectories such as straight lines. The figure below illustrates some of these simple trajectories. To train the PCR model, we first collected data from lidar point clouds and absolute locations of the vehicles. Specifically, we used the absolute locations at each point cloud frame that we collected to compute the transformations between consecutive frames. This allowed us to pair the consecutive point clouds and create a dataset of point cloud pairs and their corresponding transformations.

We then used this dataset to train the PCR model, which learns to predict the transformation between two given point clouds. Upon evaluating the initial results of the PCRNet model, we observed that the odometry predictions for some input sequences only showed straight lines, without any noticeable changes or turns. This indicates that the model may have learned to predict a constant or fixed transformation matrix for these sequences, which results in the car moving in a straight line. While this behavior is expected for input sequences where the car is indeed moving straight, it is not desirable for sequences where the car is supposed to turn or follow a curved trajectory. To address the issue of the PCRNet model only predicting straight line trajectories, we improved our data collection procedure by including more complex transformation data. Specifically, we used the Carla simulator to generate trajectories that were no longer simple straight lines, but instead involved constant changes in rotation and translation. This resulted in point cloud sequences with more pronounced changes in rotation and translation between consecutive frames. To further enhance the diversity of our training data, we moved from using consecutive point cloud pairs to using a time band approach. For each source image, we randomly selected a target point cloud from within the previous 40 point cloud frames in the order they were collected. This helped to ensure that the training data was not biased towards any particular time interval or trajectory.

To balance the low and high rotation and translation transformations in the data, we also developed code that adjusted the number of examples in the training data for each transformation range. This helped to ensure that the PCRNet model learned to predict accurate transformations across the entire range of rotation and translation values, rather than just focusing on the more extreme values. Fig 13 shows image with midway through the updated trajectory. These improvements to our data collection and preprocessing procedures allowed us to train a more robust PCRNet model that was able to predict better.

The baseline approachin is working perfectly on Kitti Dataset The above is the result from Carla Dataset

VI. Acknowledgements

We would like to express our heartfelt gratitude to Professor Plat for their exceptional guidance and mentorship throughout our academic journey. Their knowledge, expertise, and dedication to their field have been an endless source of inspiration for us. We are grateful for the opportunities and challenges they have presented to us, which have helped us to develop and grow as students and individuals. We would also like to

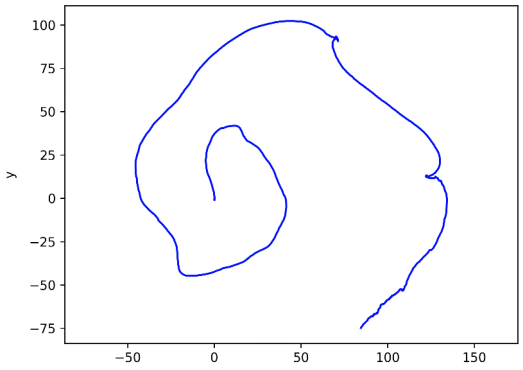

Fig. 18. PCRNet mid way through updated data

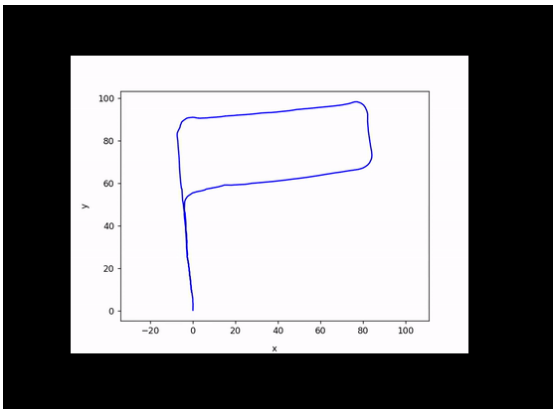

Fig. 19. Visualized Data From Kitti

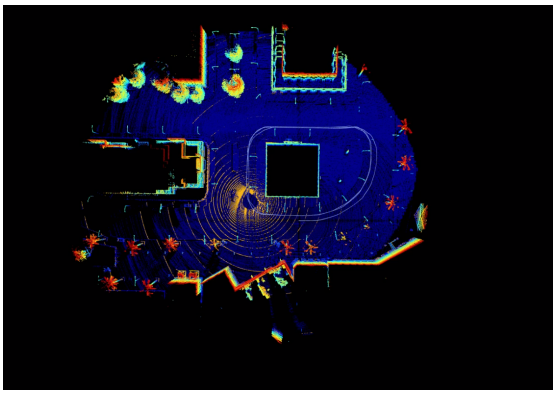

Fig. 20. Visualized Data From CARLA

extend our thanks to the teaching assistants and Northeastern University for providing us with the support and resources necessary to succeed.

VII. CONCLUSIONS

Iterative Closest Point (ICP) is a widely used method for registering 3D point clouds obtained from various sensors such as LiDAR, stereo cameras, or RGB-D cameras. However, traditional ICP suffers from some limitations, such as being sensitive to initialization, prone to getting stuck in local minima, and being sensitive to outliers. Unlike ICP, Deep learning algorithms automatically learn complex representations and extract relevant features from data, without explicit models or assumptions. This makes them ideal for handling noisy, ambiguous, or complex data that is difficult to model using traditional methods. Deep learning models also allow flexible input shapes which lets us input sequences of point clouds as input, something that the traditional methods do not. This can help us in real world point cloud registration situations with noise, outliers, occlusions, non-rigid deformations, or non-uniform sampling densities.

Given the limitations of traditional methods in handling complex scenarios, it is important to continue research and development on deep learning-based point cloud registration models. These models have shown great promise in handling noisy, ambiguous, and complex data, and have the potential to significantly improve the accuracy and robustness of point cloud registration in a wide range of applications.

REFERENCES

[1] Zhiyuan Zhang, Yuchao Dai, and Jiadai Sun. "Deep learning based point cloud registration: an overview." Virtual Reality & Intelligent Hardware 2.3 (2020): 222-246.

[2] Xiaoshui Huang, Guofeng Mei, Jian Zhang, and Rana Abbas. "A comprehensive survey on point cloud registration." arXiv preprint arXiv:2103.02690 (2021).

[3] Guillaume Bresson, Zayed Alsayed, Li Yu, and Sébastien Glaser. "Simultaneous Localization and Mapping: A Survey of Current Trends in Autonomous Driving." IEEE Transactions on Intelligent Vehicles 2.3 (2017): 194-220. doi: 10.1109/TIV.2017.2749181.

[4] S. Rusinkiewicz and M. Levoy, "Efficient variants of the ICP algorithm," in Proceedings Third International Conference on 3-D Digital Imaging and Modeling, 2001, pp. 145-152, doi: 10.1109/IM.2001.924423.

[5] Dmitry Chetverikov, Dmitry Svirko, Dmitry Stepanov, and Pavel Krsek. "The trimmed iterative closest point algorithm." In 2002 International Conference on Pattern Recognition, vol. 3, pp. 545-548. IEEE, 2002.

[6] Yue Wang and Justin M Solomon. "Deep closest point: Learning representations for point cloud registration." In Proceedings of the IEEE/CVF international conference on computer vision, pp. 3523-3532, 2019.

[7] Weixin Lu, Guowei Wan, Yao Zhou, Xiangyu Fu, Pengfei Yuan, and Shiyu Song. "Deepvcp: An end-to-end deep neural network for point cloud registration." In Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 12-21, 2019.

[8] Charles R Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J Guibas. "Frustum pointnets for 3d object detection from rgb-d data." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 918-927, 2018.

[9] 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, pp. 652-660, 2017.

[10] 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, vol. 30, 2017.

[11] Vinit Sarode, Xueqian Li, Hunter Goforth, Yasuhiro Aoki, Rangaprasad Arun Srivatsan, Simon Lucey, and Howie Choset. "Pcrnet: Point cloud registration network using pointnet encoding." arXiv preprint arXiv:1908.07906 (2019).

[12] Anh Viet Phan, Minh Le Nguyen, Yen Lam Hoang Nguyen, and Lam Thu Bui. "Dgcnn: A convolutional neural network over large-scale labeled graphs." Neural Networks 108 (2018): 533-543.

[13] Yue Wang and Justin M Solomon. "Deep closest point: Learning representations for point cloud registration." In Proceedings of the IEEE/CVF international conference on computer vision, pp. 3523-3532, 2019.

[14] Juyong Zhang, Yuxin Yao, and Bailin Deng. "Fast and robust iterative closest point." IEEE Transactions on Pattern Analysis and Machine Intelligence 44, no. 7 (2021): 3450-3466.

[15] Giseop Kim and Ayoung Kim. "Scan context: Egocentric spatial descriptor for place recognition within 3d point cloud map." In 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 4802-4809, 2018.

[16] Han Wang, Chen Wang, and Lihua Xie. "Intensity scan context: Coding intensity and geometry relations for loop closure detection." In 2020 IEEE International Conference on Robotics and Automation (ICRA), pp. 2095-2101, 2020.

[17] Saba Arshad and Gon-Woo Kim. "Role of deep learning in loop closure detection for visual and lidar slam: A survey." Sensors 21, no. 4 (2021): 1243.

[18] Luca Carlone, Roberto Tron, Kostas Daniilidis, and Frank Dellaert. "Initialization techniques for 3D SLAM: A survey on rotation estimation and its use in pose graph optimization." In 2015 IEEE International Conference on Robotics and Automation (ICRA), pp. 4597-4604, 2015.

[19] Giulia Sammartano and Antonia Spanò. "Point clouds by SLAM-based mobile mapping systems: accuracy and geometric content validation in multisensor survey and stand-alone acquisition." Applied Geomatics 10 (2018): 317-339.

[20] Pileun Kim, Jingdao Chen, and Yong K Cho. "SLAM-driven robotic mapping and registration of 3D point clouds." Automation in Construction 89 (2018): 38-48.

[21] Bashar Alsadik and Samer Karam. "The simultaneous localization and mapping (SLAM)-An overview." Surveying and Geospatial Engineering Journal 2 (2021): 34-45.

[22] Jean-Emmanuel Deschaud. "KITTI-CARLA: a KITTI-like dataset generated by CARLA Simulator." arXiv preprint arXiv:2109.00892 (2021).

[23] Andreas Geiger, Philip Lenz, and Raquel Urtasun. "Are we ready for autonomous driving? The KITTI vision benchmark suite." 2012 IEEE Conference on Computer Vision and Pattern Recognition. IEEE, 2012, pp. 3354-3361.

[24] Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, Oscar Beijbom. "nuScenes: A multimodal dataset for autonomous driving." Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2020, pp. 11621-11631.

[25] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Scharw"achter, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. "The cityscapes dataset." CVPR Workshop on the Future of Datasets in Vision 2 (2015).

[26] SegMatch (Dubé et al., 2017): "SegMatch: Segment based place recognition in 3D point clouds."

[27] 3D NDT (Biber and Straßer, 2003): "The normal distributions transform: A new approach to laser scan matching."

[28] PointNetVLAD (Angelina Uy and Lee, 2018): "PointNetVLAD: Deep Point Cloud Based Retrieval for Large-Scale Place Recognition."

[29] g2o (Kümmerle et al., 2011): "g2o: A General Framework for Graph Optimization."

[30] iSAM2 (Kaess et al., 2012): "iSAM2: Incremental Smoothing and Mapping Using the Bayes Tree."

[31] Ceres Solver (Agarwal et al., 2018): "Ceres Solver: A Large Scale Nonlinear Optimization Library." (Note: This is not a paper but the official documentation for the Ceres Solver library.)