Final Report: Place Recognition using GNN and NetVlad

1 Motivation

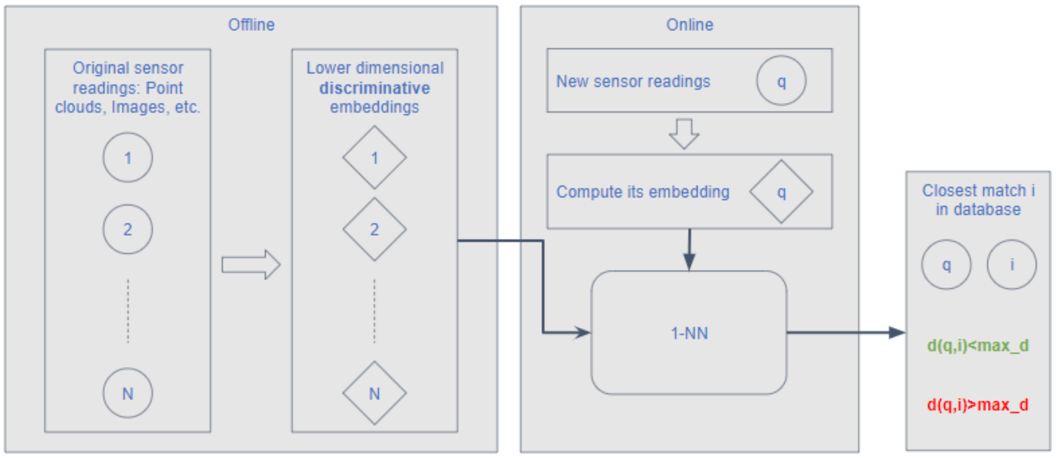

Offline
Original sensor readings: Point clouds, Images, etc.
1
2
N
Lower dimensional discriminative embeddings
1
2
N
Online
New sensor readings
q
Compute its embedding
q
1-NN
Closest match i in database
q
i
d(q,i)<max_d
d(q,i)>max_d

Figure 1: The working principle of place recognition

Autonomous driving requires various capabilities to work accurately and robustly. One of these capabilities is place recognition which is defined as the task of roughly localizing the place in which some sensor readings was taken in a map. These sensor readings could be for example images and/or point clouds. The working principle of place recognition is explained in the diagram in figure 1. In an offline step sensor readings are collected from multiple locations in the map, with different angles and perspectives, converted into a lower dimensional discriminative embedding space and stored in a database. In the online step during live runs, whenever a new sensor reading (called the query) is available, it is converted to its embedding using the same model. Using 1-NN method, its nearest neighbor is located from the database. Ideally the nearest neighbor in the database should correspond to a sensor reading taken from a location that is within a certain radius from the location of the query. If this is the case, then we say that the query was correctly matched.

A possibly suitable type of sensor reading is point clouds from LiDAR scans. LiDAR sensors have multiple advantages compared to other sensors like cameras. They are usually higher resolution and produce point clouds that include more granular detail. Since it utilizes laser beams, it is more resistant to changes in day time and lighting conditions. It is also more resistant to weather variations. Moreover they usually span a 360° field of view of the scene. As a result, they should enable a more accurate and robust place recognition.

However, point clouds have some special characteristics that need to be taken into account. Raw point clouds are stored as unordered sets, so the methods need to be permutation invariant. Point clouds of a location that get rotated/translated still show the same location, so the methods need to invariant to rigid transformations. Since the points belong to specific objects/landmarks in the environment, the methods should capture the interactions/relations between the points.

R. Qi et al. propesd PointNet [1] which is a special neural network developped to solve 3D classification and segmentation problems using point sets. Its main characteristics are invariance to permutations and to rigid transformations. It achieved good results on object detection, classification, and segmentation. PointNetVLAD [2] utilizes PointNet to extract local features from each point and then aggregates these local features using NetVLAD[3] into a global feature/embedding. This global feature/embedding is then used to recognize and find similar locations in the database. It uses different loss functions (lazy/non-lazy triplet/quadruplet loss) during training to bring embeddings of similar locations closer together and ones from different locations further apart. It achieved good results in the Oxford dataset (80.31% average recall for top 1%).

The main issue here is the PointNet's inability to capture the relations/interactions between points. The purpose of our project is to replace the PointNet part in PointNetVLAD by a Graph Neural Network (GNN) in order to extract these relations and interactions better with the aim to improve the performance.

2 Methods

2.1 Data Collection

For the CARLA dataset, data collection was done using a modified version of a script made by a group from last semester provided to us by Mariia. The first part of the script collects LiDAR scans from CARLA. Each tick, other than LiDAR frames, it also collects 1 RGB image and the GNSS information. Each scan is also already geo-fenced to maximum 50 m by setting the maximum range.

The second part generates sub maps from these scans. From the very first frame, it waits until it finds a frame located 10 m from it. This frame becomes the center of the first sub map. The next center is located 10 m from this one. This means each sub map combines scans in a 20 m region and each sub map has a 10 m overlap with the next sub map. Other than combining scans, this step also performs several pre-processing, such as removing points that belong to the class pedestrians, road lines, roads, sidewalks, vehicles, and ground, reducing the amount of points in each sub map to 4096 points, transforming so that the centroid becomes (0, 0, 0), normalizing the points using (point - mean)/std so that the points have values in the range of [-1, 1] and a mean of zero, and adding normals for each point

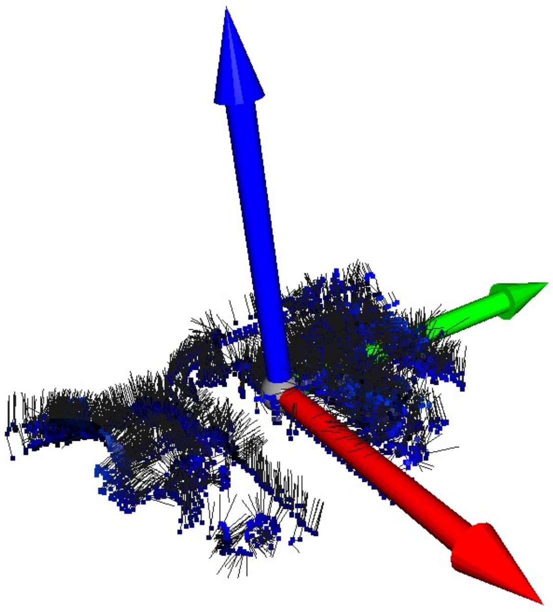
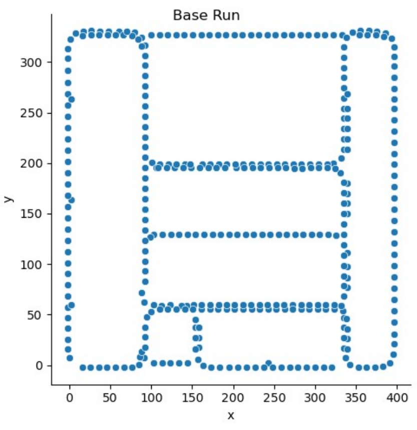
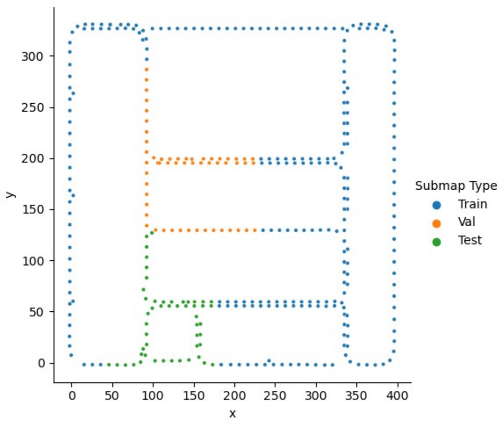


Figure 2: A sub map example

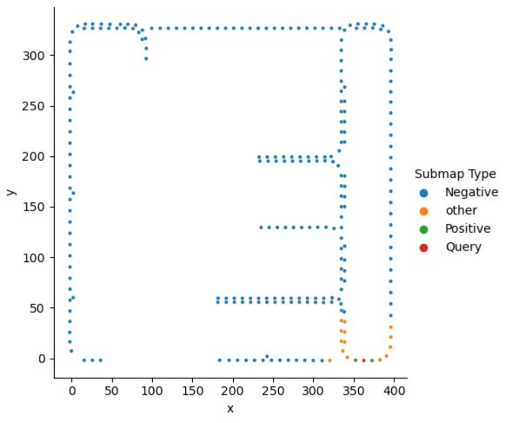
using Open3D. Unfortunately, the normalization is not perfect. The vast majority of points are located in the [-1, 1] range, however there is usually a singular outlier point. This can't be removed post voxel downsampling because it will cause the amount of points to drop to 4095, which doesn't work for our network.



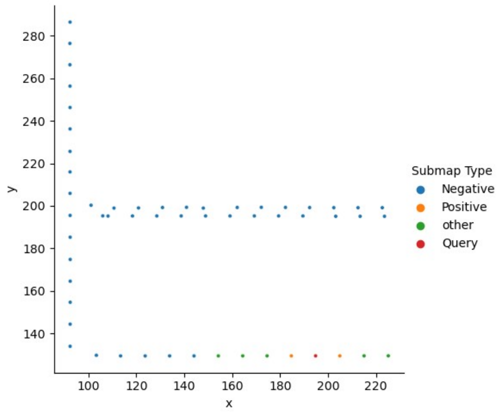
(a) Sub maps from the base/main run



(b) Train-Val-Test split

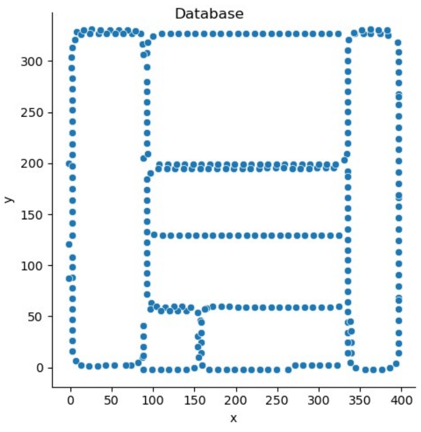


(c) Example train query

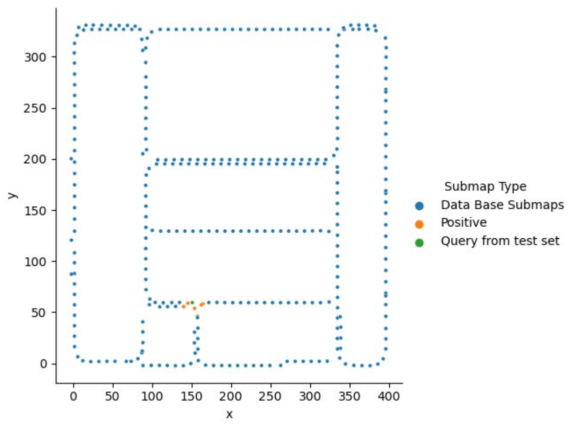


(d) Example test query

The next part generates the actual training and validation datasets and the queries they contain. This script uses the sub maps and all related information generated in the previous step. Based on the GNSS information of the sub maps and a threshold that is set manually, it divides the map into training, validation, and test regions. For each query, the script selects other sub maps in a 20 m radius from it as positives and points beyond 50 m from it as negatives. Points in the 20-50 m range are ignored to reduce ambiguity. Each query has this format: *{query: "path to query .ply", positives: ["list of paths to positive .ply"], negatives: ["list of paths to negative .ply"]}*. Using sub maps from 3 maps, the script generated 976 queries with 1-10 positives and 224-403 negatives each for the training dataset and 291 queries with 1-11 positives and 32-122 negatives each for the validation dataset.



(a)



(b)

Figure 4: Sub maps generated from the test run and an example test query

The final part generates the test dataset and a database for use during testing. It functions similarly to the previous part and also uses manually set thresholds. But, the frames and sub maps used here are taken from a separate run. This part also generates a database that contains all sub maps from the run. The queries in the test dataset contains a list of ground truths. This list contains the index of the sub map inside the database.

Another dataset that we used was the Oxford dataset [4]. This particular database is based on the work inside the original PointNetVLAD repository [5]. The point clouds are provided downloadable from a Google Drive linked in the readme of the repository. Unfortunately, these point clouds are stored in .bin files. So, a script was used to transform the point clouds into .ply files and add normals to the points. Other than this, no further work was needed for the point clouds themselves. They were already geo-fenced, unnecessary points like road and ground were already removed, each already contains exactly 4096 points, and the values are already normalized to [-1, 1] with a zero mean.

The queries are generated using a modified version of the *generate training tuples baseline* script from the PointNetVLAD repository [5]. The modification was done to how positives and negatives are referenced in queries. Oxford initially uses a database and the positives and negatives of each query are listed as their indices in the database. Our data module requires the path to the .ply files. Other than this, the amount of queries had to be reduced to 5481 training queries and 2621 validation queries from the initial 20000+ training queries. The amount of positives and negatives also had to be capped to 1024 and 4096 respectively. This had to be done to ensure that the remote server's 16GB RAM could handle it when using paths instead of indices.

The final training dataset contains 5481 queries with 1-50 positives and 4096 negatives each. The validation dataset contains 2621 queries with 3-43 positives and 2407-2581 negatives each. Even after being reduced this is still considerably more than what we could've collected if we went with collecting more data from the 3 CARLA maps but with dynamic objects (vehicles and pedestrians) instead.

2.2 Network Input

For each query, our data loader randomly selects 1 positive pair and 3 negative pairs out of the available options. It then loads the .ply files and turns it into a pytorch-geometric Data object. The point coordinates are placed in pos and the normals in normals. Optionally, the data module can also perform live data augmentation, such as random translation, rotation, and scaling. Each time a new point cloud is loaded, a random amount of translation, rotation, and scaling within the range [-x, x], where x is the value set in the trainer script, will be applied to the point cloud before its coordinates are stored in the pytorch-geometric Data object. The loader outputs a LightningDataset object containing the training and validation datasets, with the proper batch size set from the trainer script.

2.3 GNN Layers:

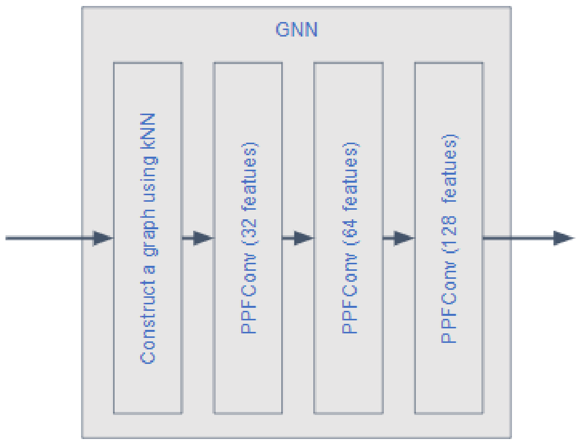


Figure 5: The architecture of the GNN part of the network

As mentioned in the motivation section, one of our contributions is the use of a graph neural network instead of PointNet to capture relations between points and their respective neighbors in addition to being invariant to permutations and rigid transformations. The architecture of the GNN that we used is presented in figure 5. The GNN takes as input the unordered set of points in a point cloud. It then builds a graph using the k-NN method (or optionally the ball query method). The k-NN ensures the permutation invariance because even though the points in a point set are unordered, the resulting graph would always be the same. The k-NN is also not affected by rigid transformations because the neighbors of a point are unchanged if the same rigid transformation is applied to the whole point set.
Once the graph is built, it is passed to 3 layers of PPFConv layers with increasing dimensions (32/64/128). The number of layers as well as their dimensions are chosen empirically. The PPFConv are chosen because their message aggregation method is invariant to rigid translations. In fact the message passing function follows equation 1, where the global neural network is just the ReLU activation function, the local neural network is a sequential network

composed of two linear layers with output dimensions equal to the number of dimension specified in 5 and a ReLU layer in between.

$$x_i^{k+1} = global\_nn(\max_{j \in N(i)} local\_nn(x_i^k, \|d_{j,i}\|, \angle(n_i, d_{j,i}), \angle(n_j, d_{j,i}), \angle(n_i, n_j))) \qquad (1)$$

As one can see from equation 1, the node features of node x_i in the k+1'th layer depends only on the node features of its neighbors x_j in the k'th layer, the distance between the node x_i and x_j, the angle between the normal vector n_i of node x_i and the vector pointing from x_i to x_j, the angle between the normal vector n_j of neighboring node x_j and the vector pointing from x_i to x_j, and the angle between the normal vector n_i and n_j. As one can notice, these features are robust to rigid transformations.

2.4 NetVLAD Layer

The NetVLAD module we used is taken from the PointNetVlad-Pytorch repository [6] created by Daniele Cattaneo (cattaneod). This repository is a faithful pytorch conversion of the tensorflow NetVLAD implementation of the original PointNetVLAD repository. Minimal changes were implemented to this module. This is mainly to adapt the parameters of this module to be loaded from the hyperparameter dict that we use in the trainer script and to add a single dropout layer. The dropout layer is placed after the first of this module's equivalents of CNN, batch normalization, and activation layers. For our experiments our NetVLAD layer expect 4096 initial points and has a cluster size of 32. We also enable batch normalization and gating for NetVLAD.

2.5 Loss Function

The loss function should make the network learn to predict discriminative embeddings for the point clouds, i.e. point clouds taken in close places sharing thus some common features should have similar embeddings, while point clouds taken from far away places should have a different embedding. We decided to use normal triplet loss for simplicity reasons.

$$L(q, p, n_1, n_2, n_3) = margin + \|q - p\|_2^2 - \sum_{j=1}^{3} (\|q - n_j\|_2^2) \qquad (2)$$

As we can see from equation 2 the loss is reduced if the embedding p of the positive point cloud is similar to the embedding q of the query and the embeddings of negative point clouds n are dissimilar from the embedding q of the query.

2.6 Training Process

During training, we tried multiple hyperparameter combinations. Some values are set to be the same everytime. For example, batch size of 4 and 1 positive and 3 negatives per query. We tried with and without data augmentations and dropout. When activated, we always use random translation of [-0.5 m, 0.5 m] in the x, y, and z axis and random rotation of [-90, 90] degrees around the z axis. Dropout probability is also set to 0.2. This value was discovered after testing different values. Dropout layers are placed after every PPFConv layer and in the location described in the NetVLAD subsection. We also trained for unlimited amount of epochs. Stopping was either done manually or using early stopping with a minimum delta of 0.00000001 and patience of 25.

3 Results

After training, we ended up with three representative results. One model was trained using our CARLA dataset only, a second one was trained using the Oxford dataset, and the last one was a fine-tuning of the second pre-trained model on our CARLA dataset. These models all suffered heavly from overfitting, most likely due to the small size of training data. Our CARLA dataset had only 976 queries, while our Oxford dataset had to be reduced from 21711 to 5481 queries due to hardware limitations This overfitting will be clearly visible in the graphs of the train/validation loss presented in the following subsections.

3.1 CARLA Dataset Only

This model was trained only on the CARLA dataset. During training, it had random translation, rotation, and dropout active. The learning rate used was 0.0001. After the training was stopped by early stopping, this model had 12.195 validation loss and 4.007 training loss as its best result. The loss graph is as follows:

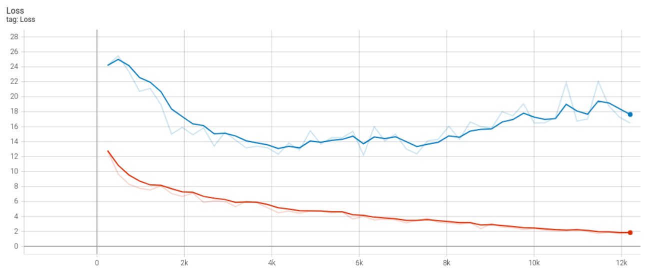


Figure 6: Loss Graph of the CARLA Only Training

3.2 Pre-training on Oxford Dataset

This is after pre-training on Oxford dataset. T model was trained with random translation and rotation but no dropout. The learning rate used was 0.00005. After the training was stopped by early stopping, this model had 9.458 validation loss and 3.658 training loss as its best result. The loss graph is as follows:

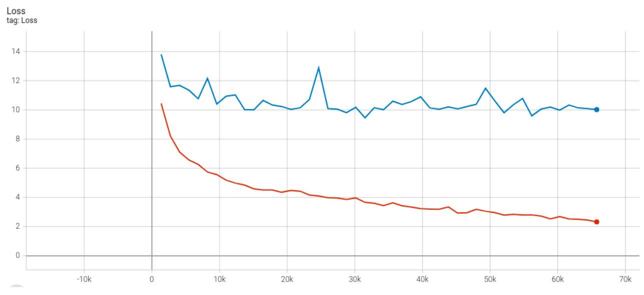


Figure 7: Loss Graph of the Oxford Pre-Training

3.3 Fine-tuning on CARLA Dataset

This is after the previous pre-trained model was finetuned on the CARLA dataset. The model was trained with random translation and rotation but no dropout. The learning rate used was 0.00005. After the training was stopped by early stopping, this model achieved 11.678 validation loss and 2.116 training loss. The loss graph is as follows:

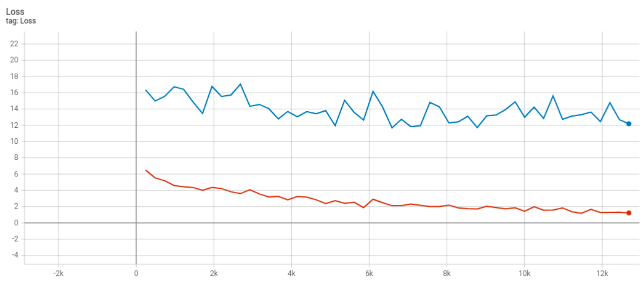


Figure 8: Loss Graph After Fine-tuning on the CARLA Dataset

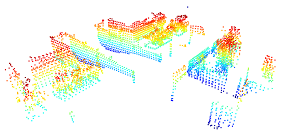
3.4 Testing on the CARLA Dataset Test Set and Database

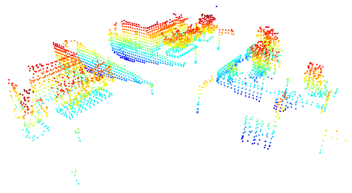
After the three models were trained, we then tested them on a test script made specifically for our CARLA test set. Other than measuring top 1-10 and 1-10% accuracy, this script also visualizes a randomly picked point clouds of one example query that was correctly matched and another one that was incorrectly matched. For each case, the script will produce an image of the query, 10 nearest neighbors (each named if they are a positive or negative based on query ground truth indices), and all of the ground truths. Finally, this script will also output a PCA-TSNE visualization of the query, the ground truths, and the whole dataset's embeddings, for both the match and no match cases.

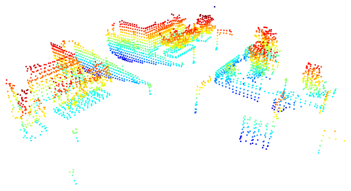
From this script, we recieved these results for the model's accuracy:

	CARLA Only	**Oxford Pre-Train**	**CARLA Fine-Tune**
Top 1% Accuracy	37.1%	44.35%	52.42%
Top 5% Accuracy	86.29%	81.45%	88.71%
Top 10% Accuracy	95.97%	93.55%	95.97%

Below are some examples of the point clouds and the PCA-TSNE graphs for both match and no match case when using the model fine-tuned on the CARLA dataset:


(a) The query


(b) An example of the ground truth


(c) Prediction (correct)

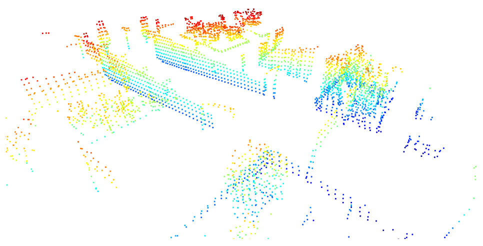
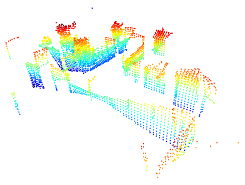
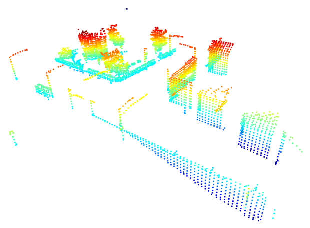
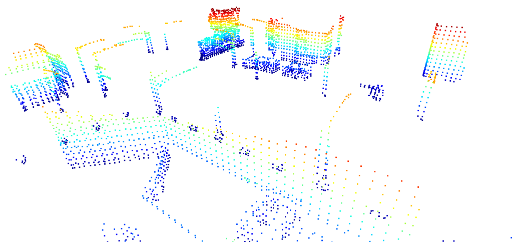

(d) An example of a wrong prediction

Figure 9: Point cloud example (correctly matched)


(a) The query


(b) An example of the ground truth


(c) Prediction (incorrect)

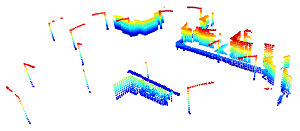

(d) An example of an incorrect prediction

Figure 10: Point cloud example (incorrectly matched)

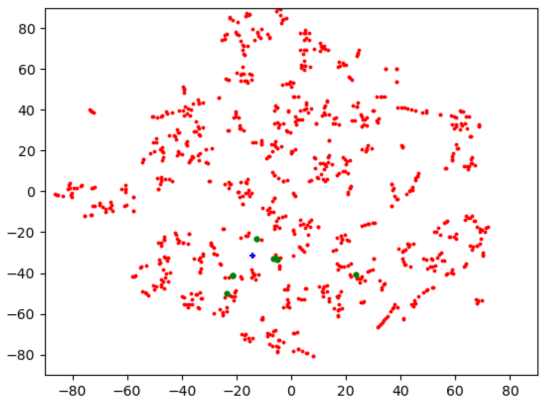


Figure 11: PCA-TSNE Graph for the Match Case

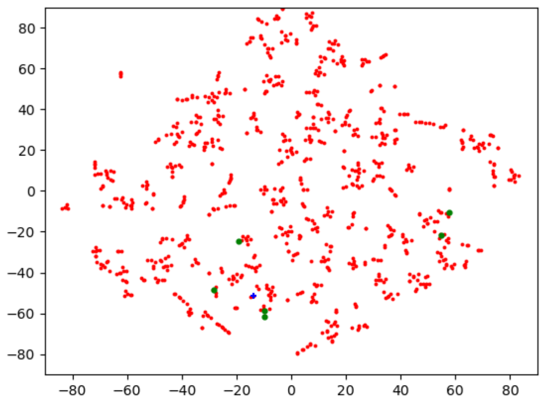


Figure 12: PCA-TSNE Graph for the No Match Case

As we can see from the table of accuracies, the qualitative point cloud visualizations, and the PCA-TSNE graphs, the results of the model are rather decent. The fine-tuned model managed to reach a 52.42% top-1% accuracy even when it was pre-trained with a significantly reduced Oxford dataset and the fine-tuned with an extremely small CARLA dataset.

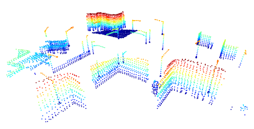
Based on the point cloud visualizations, we can also see that both in the match and no match cases, even the wrongly predicted point clouds generally still look similar to the ground truth. In the match case, the negative example is still a 4-way crossing with features similar to the ground truth in each corner. In the no match case, both negative examples show some sort of crossing, with traffic lights, and similar features to the ground truth as well.

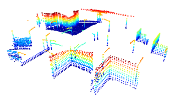
From the PCA-TSNE graphs, we can also see that the embeddings of the ground truths are located rather close to the query. In the no match case, this is a bit more spread out and we can see two ground truths that are located rather far apart from the query embedding.

4 Bug from the Data Generating Script from Previous Years

Originally, the same three models had significantly worse accuracy values when tested on the exact same testing script. For example, the CARLA fine-tuned model achieved only 1.1905% top-1% accuracy.

For the longest time, we assumed that this was simply because of either the model or our issue with lack of training data. However, in the last week we decided to visualize the point clouds. Once we do so and we sat together to polish the code, we noticed that the ground truths inside the dataset were often really wrong in the no match cases, while the "wrong" predictions are actually significantly more similar to the query.


(a) The Query


(b) The "Wrong" Prediction

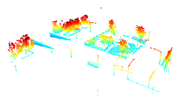

(c) The Ground Truth 3

Figure 13: Point Cloud Examples From the No Match Case

Based on this finding, we then checked the code used to generate our test set and database. We found that the parameter used to generate the KD-tree and query nearest neighbors from the tree in this script was wrong. The original script used longitude and latitude when it should have been using the centroid's cartesian coordinates. This was missed in earlier weeks because the script used to generate the train and val sets were already correct and used the centroid cartesian coordinates. In the rush to start training as soon as possible, this one difference between the two scripts was missed.

After this, we fixed the script and also wrote a function to remove the query itself from the list of ground truth indices. Once this issue was fixed and the new test dataset and database were used, our results immediately spiked to the final results.

	Top 1% Accuracy		
	CARLA Only	**Oxford Pre-Train**	**CARLA Fine-Tune**
Spherical coordinates	0.0000%	5.9524%	1.1905%
Cartesian coordinates	37.1%	44.35%	52.42%

5 Challenges

The first challenge that we faced during this project happened during data collection. The CARLA in the remote server crashes a lot during LiDAR scan collection. As a result, we had to reduce the amount of frames generated per run from the originally intended 5000 frames per run to 2000 frames per run. To compensate for this, the amount of runs had to be increased 5 runs for 25000 frames to 10 for only 20000 frames. Even with this, CARLA still often crashed during collection and then it had to be restarted from the beginning. As a result, data collection took extremely long, around 2-4 hours for 20000 frames on each map.

The second challenge that we faced was the fact that CARLA in the remote server only had 5 maps and out of the 5 only 3 was useable for our project. This is because the other two consisted mainly of highways instead of streets and buildings. This is not good because these highways look extremely similar at most points with very similar features. We decided that it would be too difficult for our project and we skipped the two maps. As a result, our CARLA dataset only has 976 training set queries. Originally we couldn’t solve it and decided to just work with it. In the final week we landed on the idea to pre-train using the Oxford dataset and fine-tune on our CARLA dataset. This helped with our lack of data slightly but not too much because of reasons we will mention below.

The third challenge was related to the hardware we had available for training. Oxford dataset originally had 20k+ queries. We had to make some modifications for the dataset to be usable by our data module without significant changes. This modification was done by changing how the dataset shows the query, positive, and negative examples. Oxford originally used database indices, while we needed the path to the .ply file. But, because of these changes, it became too large for our remote server. This means that the dataset couldn’t be generated and when it is reduced but by not enough, the remote can’t run the training and crashed because it ran out of memory. This is presumably because of the 16GB of RAM available in the remote servers, while Oxford was trained on a machine with 64GB RAM. As a workaround, we had to reduce it to a quarter of the original amount (around 5.5k queries). This is still significantly more than the final amount of queries if we added 3 more CARLA maps with dynamic objects (approx. 2 x 976 queries total) but still significantly less than the original Oxford dataset.

The last challenge that we faced was related to the previous 3 issues with data generation and our lack of training data. As the result of these issues, our models overfit heavily and fail to get low validation losses, while having rather low training losses. We tried several methods to reduce this. For example, we tried to use data augmentation (random translation and rotation) and dropout since they are techniques commonly used to combat overfitting and lack of data. However, they generally didn’t work well enough to solve the issue. We also tried to pre-train with the Oxford dataset and then fine-tuning on the CARLA dataset. But even this only helped a bit, reducing the validation loss from 12.194 with CARLA only to 11.677 after pre-training and fine-tuning. This challenge remains unsolved for our group.

References

[1] Qi, Charles R., 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.

[2] Uy, Mikaela Angelina, and Gim Hee Lee. "Pointnetvlad: Deep point cloud based retrieval for large-scale place recognition." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 4470-4479. 2018.

[3] Arandjelovic, Relja, Petr Gronat, Akihiko Torii, Tomas Pajdla, and Josef Sivic. "NetVLAD: CNN architecture for weakly supervised place recognition." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 5297-5307. 2016.

[4] Maddern, Will, Geoffrey Pascoe, Chris Linegar, and Paul Newman. "1 year, 1000 km: The Oxford RobotCar dataset." The International Journal of Robotics Research 36, no. 1 (2017): 3-15.

[5] https://github.com/mikacuy/pointnetvlad

[6] https://github.com/cattaneod/PointNetVlad-Pytorch