7. Supplementary Material

7.1. AutoEncoder Architecture

As explained in Sect 3.2, an autoencoder structure is used to extract salient features from the DEM and to reconstruct the DEM which is later used to perform pose estimation (explained in Sect. 3.5). The *AutoEncoder* architecture is shown in Table 4.

Operator	Stride	Filter	Padding	Output Shape
Conv2D	(1,1)	(3,3)	(1,1)	500×500
MaxPool2D	(1,1)	(2,2)	(0,0)	250×250
Conv2D	(1,1)	(3,3)	(1,1)	250×250
MaxPool2D	(1,1)	(2,2)	(0,0)	125×125
Conv2DTrans	(2,2)	(2,2)	-	250×250
Conv2DTrans	(2,2)	(2,2)	-	500×500

Table 4: First four rows show the encoder, and next two rows the decoder. *Conv2DTranspose* refers to the convolution 2D transpose operation within PyTorch that is used for convolution with upsampling.

7.2. Differene Layer CNN Architecture

ResNet based CNN layers have been used before the *Difference Layer*, to extract features of the anchor's (positive or negative) yaw aligned embedding. As explained in Sect. 3.4, the output F_{diff} of the *Difference Layer* is sent to a CNN to get a scalar value indicating the distance between the two DEMs. The architecture of the CNN is shown in Table 5, the input to this part of the model is a feature volume of size $10 \times 1024 \times 1024$.

Operator	Stride	Filter	Padding	Output Shape
Conv2D	(2,2)	(5,5)	(0,0)	$64 \times 510 \times 510$
Conv2D	(2,2)	(5,5)	(0,0)	$32 \times 253 \times 253$
Conv2D	(2,2)	(1,1)	(0,0)	$4 \times 127 \times 127$
Linear	-	-	-	1×64516
Linear	-	-	-	1×100
Linear	-	-	-	1×10

Table 5: Architecture of the CNN used after the *Difference Layer*.

In *GPR-15* our method is better than *LCDNet* by 4% and the *KITTI-00* sequence we are the second best to *LCDNet* by less than 1%. In Fig. 8, we show qualitative results for *Loop Closure* on *GPR-10* [32] and *KITTI-08* [9].

7.3. Integration with Lio-SAM

Here we detail the experimental procedure to integrate our proposed LDC pipeline into LIO-SAM . Similar to *OverlapNet, ScanContext* we utilize the geometry of the factor graph for the LDC task, for every new state x_{i+1} added to the factor graph a local area of $15m$ is searched

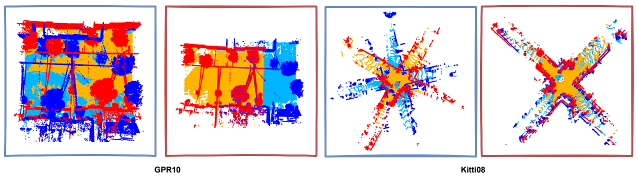

Figure 8: Qualitative Results for *Loop Closure* on *GPR-10* (left) and *KITTI-08* (right). The query point cloud is shown in the orange, the retrieved point cloud is shown in blue. Observe that our method is able to register point clouds even with large displacements of 90° between the query and retrieved point clouds. **Note**: The color given to the point clouds are only for visual appeal and does not have a specific meaning.

and the closest subset of possible matches is recovered. Our pipeline robustly estimates a viewpoint invariant loop from these initial set of matches. We measure the distance (as explained in Section 3.5) between the query all the point clouds in the subset, if the sample with the closest distance is lesser than a fixed threshold it is considered as a loop and a new link would be added into the graph for optimization.

We test the integration of LIO-SAM on *KITTI-08* sequence, please refer to video on our project page [*] for demonstration. *KITTI-08* is a challenging sequence consisting of loops with a large change in view point such as those with 90° and 180° (opposite side). Our method is capable of handling such large changes in viewpoint, a qualitative sample of the DYT response is shown in Fig. 9.

7.4. Runtime Analysis

The time required to pass through the *Encoder* and *Decoder CNN* is 1.7×10^{-3} sec (encoder: 1×10^{-3}s, decoder: 7×10^{-4} s). On the *Kitti-08* sequence, *FinderNet* performs complete LDC with an average time of 223.24 ms (min time: 218.44 ms, max time: 227.15 ms). Our method works in real-time and is competitive with the baselines: *LcdNet* [3] takes 204.21 ms to complete LDC, and *OverlapNet* [4] completes LDC in 630 ms.

7.5. Ablation Study

7.5.1 Canonicalizer and DYT Performance Analysis

We independently evaluate the performance of the canonicalizer and DYT in this section. The experimental procedure is as follows: **(1)** 100 pairs of point clouds from the *GPR* dataset is sampled with poses less than $4m$ is sampled. **(2)** A random yet known synthetic 3-*DOF* rotation of up-to 60° (in roll, pitch and yaw) magnitude is applied both the point clouds. **(3)** The canonicalization procedure is performed and the estimated relative rotation is recorded. **(4)** The estimated values are compared with the known rotations and the mean of the absolute error values are obtained.. The coarse RP canonicalization had an error of $3.249/4.1582$

[*] https://gsc2001.github.io/FinderNet/

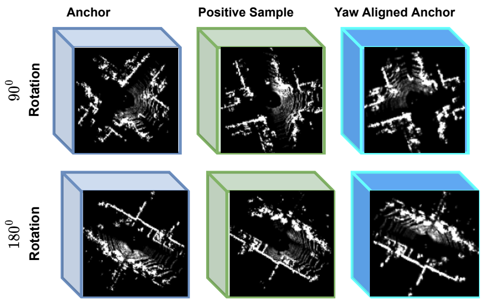

Figure 9: DYT response visualization of the latent embedding. It can be observed that the DYT predicts accurate yaw angles even and is able to transform the latent embedding even for large rotation such as 90° or 180°. Observe that for the 180° change in viewpoint the latent space appears to be flipped post DYT.

(R/P) degrees while the fine alignment works with an error of $1.2598/1.352$ (R/P) degrees respectively. The DYT was able to estimate the yaw with a error of $3.12°$.

7.5.2 Memory Analysis

It can be observed that our method bypasses the computationally expensive decompression procedure for LDC. To measure the true gain in bandwidth we benchmark our method against other point cloud compression techniques. In Table 6 we report the bandwidth required to transfer *100* point clouds (100 as it is practical number of point clouds that needs to transferred between agents) which is of 500000 kB in size. The process of DEM creation itself, has lead to a compression by a factor close to 166 (the DEM requires 3000 (kB) of space) , and the latent embedding further compresses this by five times, leading to a representation that requires 830 times less bandwidth in comparison to the original point cloud (the embedding is of size 600 (kB)). Our method further leads to closely 1.6 times better compression than [25] . *Range Images* are a popular form of point cloud representation [4], a DEM is a sparse representation in comparison to a Range Image and leads to an improvement of 1.50 times. Methods such as [3] are not meant for distributed setting, we however compare against their latent space and observe a massive improvement by a factor of 7000.

Table 6: Memory Benchmark Comparison (kB) Acronyms: **R** Range Image, **D** DEM, **E** Latent Embedding

Input Size	[25]	[3]	**R**	**D**	**E**
500000	1000	4220000	4500	3000	600

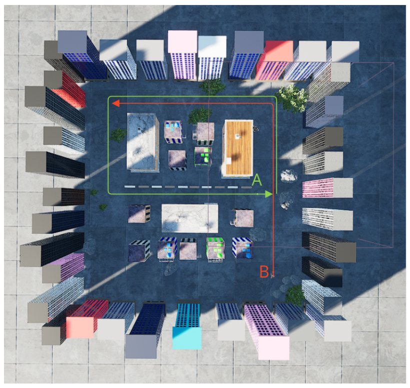

Figure 10: Drone A and B moving in sequence 4. Drone A holds database of point clouds, while Drone B possesses query point clouds

7.5.3 Evaluation in Multi Robot Sequences

FinderNet finds extensive application in bandwidth constrained settings such as collaborative SLAM. To further assess the robustness of *FinderNet* on point clouds with SE(3) motion and demonstrate the significance of performing LDC in the compressed space, we have developed a multi-robot dataset named *LUF-Multi*. It consists of two drones moving in the *Sequence 4* environment shown in Fig. 10 and we use the model trained on *Sequence:1,2,3* to perform LDC. Through this experiment, our objective is to conduct inter-robot loop detection LDC. The goal is to identify situations where two or more robots revisit the same or similar locations along their respective paths.

In Figure 10, we illustrate the setup where *Drone A* holds a database of point clouds, while *Drone B* possesses the query point clouds. Our objective is to determine whether the point cloud observed by *Drone B* has been previously encountered by *Drone A*. Every query point cloud would be matched with all the point clouds in the database (similar to ***protocol 2*** in [3]). The average precision for the *loop detection* task was found to be 0.62, and for loop closure, the error was estimated as $(1.53/26.84)$ (*ATE/ARE*). It is important to note that instead of transmitting the entire point cloud between agents, we only transmit the *latent space*, which is the output of the auto-encoder (ϕ), as explained in Section 3.2. Additionally, the *loop closure* process operates on the decoded DEM, as elaborated in Section 3.5.

7.6. The 6-DOF Experiment

A portion of literature solely work on loop detection for SE(2) motion [4, 13], and recent methods such as [3] that work on 6-DOF. However, there is no explicit experiment designed to validate the performance of these algorithms on 6-DOF. Therefore, we design a novel experiment to validate the performance of the loop detection algorithm for a large 6-DOF change in viewpoint. The experimental setup is as follows, a query point cloud A is to be matched against

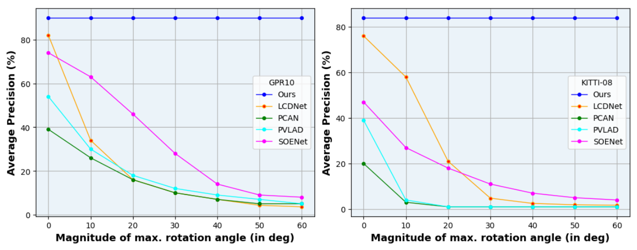

Figure 11: The plots demonstrate the results for the *6DOF Experiment*. It can be clearly seen that while other methods' *AP* values monotonically decrease, our method is robust to large changes in viewpoint and has a constant value.

a database set of point clouds $\{B\}$. We apply a synthetic *roll, pitch and yaw* rotations to both A and $\{B\}$, and use the augmented query and database point clouds to perform loop detection. *Average Precision* is used as the evaluation metric. The test is conducted on six levels of increasing difficulty. In the first stage a random *Roll and Pitch* rotation between $(-10, 10)$ is applied, in the second we apply a random rotation between $(-20, 20)$ and so-on until the sixth where a random rotation between $(-60, 60)$ is applied. The rotated set of point clouds are fed into each of the algorithms and the resulting *Average Precision* is recorded at every level. [3, 4, 23, 28] are chosen for benchmarking as they are expected to work with 6-DOF motion and *KITTI and GPR* are used for testing. The results obtained are shown in Fig. 11. We observe that with the increase in the value of synthetic rotation all the **baselines** see a **monotonic drop** in performance. However, our method **maintains constant** value of *AP* even for larger values of rotation. The use of the **canonicalization** procedure is the primary reason for the superior performance of our method. This experiment demonstrates the robustness of our pipeline for large change in viewpoints. It further highlights that pure data-augmentation during training need not result in a good performance in all scenarios. Note that although randomly sampled rotations are used, we apply the same value of rotation to all the algorithms. Even with a large rotation, such as 60 degrees, our method performs better than *LCD-Net* on an average by upto 90%.