# Identifying loop closure using OverlapNet

Internship report

Prathamesh More

## 1 Abstract

Simultaneous localization and mapping (SLAM) is an important part of most autonomous systems. It is generally considered a hard problem which is dealt according to scale and conditions of the area to be mapped. Loop closure is an important technique to improve the alignment of the overall map which gets distorted due to errors in sensor measurements. In this work, a deep neural network called OverlapNet is looked into for solving the loop closure problem. A pseudo-code for describing the framework integrating OverlapNet into Pose Graph-based optimization is explained, followed by the results obtained and possible future extensions to this work.

## 2 Introduction

Loop closure detection is an important problem for any SLAM system. Generally, the odometry measurements obtained from sensors drift away with time. Due to this, the map generated from odometry measurements becomes distorted and does not align properly overall. This problem can be reduced by closing loops in the map. The system must recognise when it has returned to a previously mapped region of the world. Essentially, at this point two regions in the map are found to be the same region in the world even though their position is incompatible given the uncertainty estimate in the map. The system must then be able to calculate the transformation needed to align these two regions to 'close the loop'. There are many algorithms in the literature which addresses this problem using vision and 3D-LiDAR sensors.

OverlapNet([1] and [2]) is deep neural network that used 3D-LiDAR sensor data to detect loop closure. It has been shown to outperform many state-of-the-art techniques like M2DP, OREOS, LocNet etc. Unlike pure scan-matching techniques, OverlapNet can even detect loop closure when the vehicle approaches a previously visited intersection from other side. Moreover, it also performs reasonably well on environments (US roads) which are different from those it has been trained on (German roads).

## 3 OverlapNet

OverlapNet predicts the overlap and yaw between pairs of LiDAR scans by exploiting multiple cues without using relative poses. Following gives a brief overview of the design of the OverlapNet.

### 3.1 Concept of overlap

The concept of overlap can be quantified by defining the overlap percentage as the percentage of pixels in the first image, which can successfully be projected back into the second image without occlusion. Note that this measure is not symmetric: If there is a large scale difference of the image pair, e.g., one image shows a wall and the other shows many buildings around that wall, the overlap percentage for the first to the second image can be large and from the second to the first image low. In OverlapNet, the idea of overlap is used for range images, exploiting the range information explicitly.

For loop closing, a threshold on the overlap percentage can be used to decide whether two LiDAR scans are at the same place and/or a loop closing can be done. For loop closing, this measure maybe even better than the commonly used distance between the recorded positions of a pair of scans, since the positions might

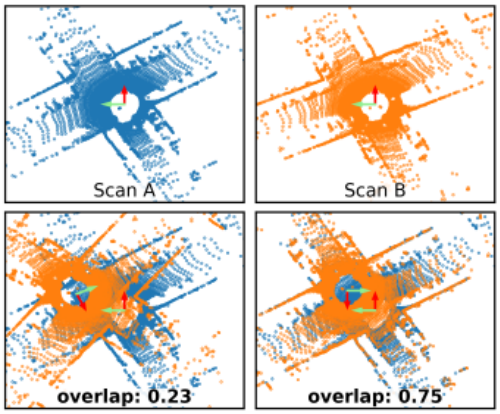


Figure 1: Overlap of two scans with different relative transformations

be affected by drift and therefore unreliable. The overlap predictions are independent of the relative poses and can be therefore used to find loop closures without knowing the correct relative pose between scans. Fig(1) shows an example of overlap between 2 scans.

### 3.2 Brief overview of the network architecture

Multiple cues are exploited from the lidar scan, namely, depth, normal, intensity and semantic class probability information. The depth information is stored in the range map $\mathcal{R}$, which consists of one channel. The point cloud $\mathcal{P}$ is projected to a so-called vertex map $\mathcal{V} : \mathbb{R}^2 \mapsto \mathbb{R}^3$ where each pixel is mapped to the nearest 3D point. Using neighborhood information of the vertex map, a normal map $\mathcal{N}$ is generated, which has three channels encoding the normal coordinates. The intensity information, also called remission, is directly obtained from the sensor and represented by a one-channel intensity map $\mathcal{I}$. The point-wise semantic class probabilities are computed using RangeNet++ [3] and represented as a semantic map $\mathcal{S}$. The semantic and intensity maps are optional for the working for OverlapNet.

OverlapNet is a siamese network architecture which consists of two legs having same architecture and sharing same weights, and two heads that use the same pair of feature volumes generated by the two legs. The pipeline overview is shown in Fig(2)

- *Legs:* There are two legs which have the same architecture and share the same weights. Each leg is a fully convolutional network (FCN) consisting of 11 convolutional layers. It generates feature volumes of size $1 \times 360 \times 128$. The number of columns of the feature volume defines the resolution of the yaw estimation, which is 1 degree in this case.

- *Delta Head:* The delta head estimates the overlap between two scans. It consists of a delta layer, three convolutional layers, and one fully connected layer. The delta layer, shown in Fig(3), computes all possible absolute differences of all pixels. It takes the output feature volumes $\mathbf{L}^l \in \mathbb{R}^{H \times W \times C}$ from the two legs $l$ as input. These are stacked in a tiled tensor $\mathbf{T}^l \in \mathbb{R}^{HW \times HW \times C}$ as follows:

$$\mathbf{T}^0(iW + j, k, c) = \mathbf{L}^0(i, j, c)$$

$$\mathbf{T}^1(k, iW + j, c) = \mathbf{L}^1(i, j, c)$$

with $k = \{0, ..., HW - 1\}, i = \{0, ..., H - 1\}$ and $j = \{0, ..., W - 1\}$
Note that $\mathbf{T}^1$ is transposed in respect to $\mathbf{T}^0$, as depicted in the middle of Fig(3). After that, all differences are calculated by element-wise absolute differences between $\mathbf{T}^1$ and $\mathbf{T}^0$. By using the delta layer, a representation of the latent difference information is obtained, which can be later exploited by the convolutional and fully-connected layers to estimate the overlap. Different overlaps induce different patterns in the output of the delta layer.

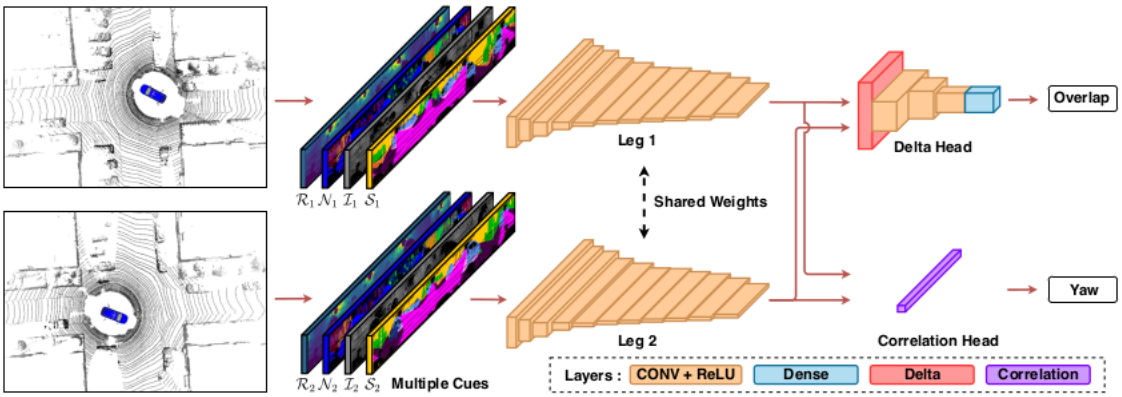


Figure 2: Architecture of the OverlapNet. The left hand side shows the preprocessing of the input lidar scan by exploiting multiple cues of the scan

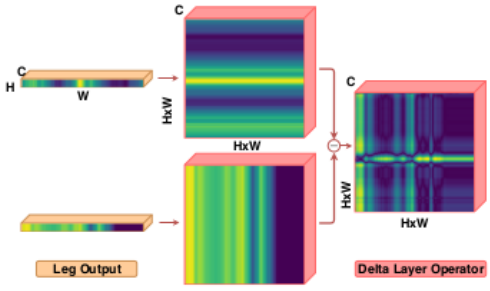


Figure 3: Delta layer. Computation of pairwise differences is efficiently performed by concatenating the feature volumes and transposition of one concatenated feature volume

- *Correlation Head:* The correlation head is designed to estimate the yaw angle between two scans using the feature volumes of the two legs. To perform the cross-correlation, one feature volume is first horizontally padded by copying the same values. This doubles the size of the feature volume.Then the other feature volume is used as a kernel that is shifted over the first feature volume generating a 1D output of size 360. The argmax of this feature serves as the estimate of the relative yaw angle of the two input scans with a 1 degree resolution.

### 3.3 Loss functions used

Consider $(I_1, I_2, Y_O, Y_Y)$ where $I_1, I_2$ are two inputs and $Y_O, Y_Y$ are the ground truth overlaps and ground truth yaw angles respectively. The legs part network with trainable weights is denoted as $f_L(.)$, the delta head as $f_D(.)$ and the correlation head as $f_C(.)$. The overlap estimation is treated as a regression problem and use a weighted absolute difference of ground truth $Y_O$ and network output $\hat{Y}_O = f_D(f_L(I_1), f_L(I_2))$ as the loss function.

$$L_O(I_1, I_2, Y_O) = \text{sigmoid}(s(|\hat{Y}_O - Y_O| + a) - b) \tag{1}$$

with $\text{sigmoid}(v) = (1 + exp(-v))^{-1}$ , a, b are offsets and s being a scaling factor.

The yaw angle estimation is treated as a binary classification problem and therefore, a cross-entropy loss is used given by

$$L_Y(I_1, I_2, Y_Y) = \sum_{i=\{1,\ldots,N\}} H(Y_Y^i, \hat{Y}_Y^i) \tag{2}$$

where $H(p, q) = plog(q) - (1-p)log(1-q)$ is the binary cross entropy, $N$ is the size of the output 1D vector and $\hat{Y}_Y = f_C(f_L(I_1), f_L(I_2))$ is the relative yaw angle estimate.

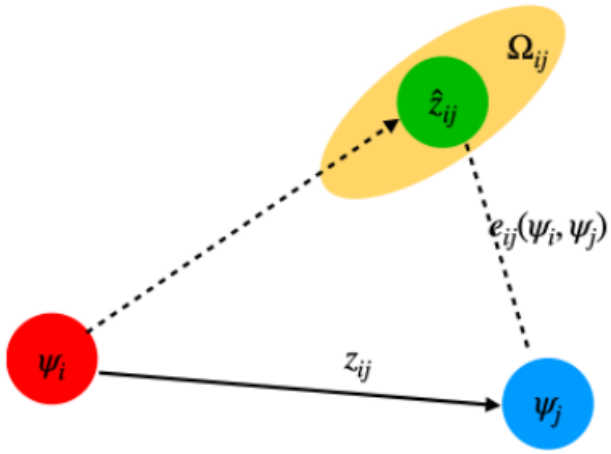


Figure 4: An edge connecting the vertex $\psi_i$ and $\psi_j$

### 3.4 Covariance propagation

Instead of using a fixed search radius for detecting loop closure, the covariance of the pose estimate and error propagation is used to automatically adjust the search radius. Let $Q_t$ be the sensor-centered coordinate frame at time t. Assume a noisy pose $\mathbf{T}_{Q_{t-1},Q_t} = \{\bar{\mathbf{T}}_{Q_{t-1},Q_t}, \Sigma_{Q_{t-1},Q_t}\}$ where $\mathbf{T}_{Q_{t-1},Q_t} \in \mathbb{R}^4$. To estimate the propagated uncertainty during the incrementally pose estimation, the mean and covariance is updated as follows:

$$\overline{\mathbf{T}}_{Q_{t-1}Q_{t+1}} = \overline{\mathbf{T}}_{Q_{t-1}Q_t}\overline{\mathbf{T}}_{Q_tQ_{t+1}}$$

$$\Sigma_{Q_{t-1}Q_{t+1}} \approx \Sigma_{Q_{t-1}Q_t} + \mathbf{J}^T_{Q_tQ_{t+1}}\Sigma_{Q_tQ_{t+1}}\mathbf{J}_{Q_tQ_{t+1}}$$

where $\mathbf{J}_{Q_tQ_{t+1}}$ is the Jacobian of $\mathbf{T}_{Q_tQ_{t+1}}$

## 4 Graph-based SLAM

A pose-graph is constructed in this SLAM technique [4]. The nodes in the graph represent the pose of a robot while the edges represent a spatial constraint. Graph-based SLAM is the problem of finding the best node configuration so that error introduced by the constraints is minimized. Consider Fig(4) which shows an edge connecting the vertex $\psi_i$ and the vertex $\psi_j$. This edge originates from the measurement $z_{ij}$. From the relative position of the two nodes, it is possible to compute the expected measurement $\hat{z}_{ij}$ that represents $\psi_j$ seen in the frame of $\psi_i$. The error $e_{ij}(\psi_i, \psi_j)$, depends on the displacement between the expected and the real measurement.

$$e_{ij}(\psi_i, \psi_j) = z_{ij} - \hat{z}_{ij}(\psi_i, \psi_j)$$

An edge is fully characterized by its error function $e_{ij}(\psi_i, \psi_j)$ and by the information matrix $\Omega_{ij}$ of the measurement that accounts for its uncertainty. The goal of the graph-based SLAM is to find the configuration of the nodes $\psi^* = \text{argmin}F(\psi)$ where $F(\psi)$ is given by

$$F(\psi) = \sum_{i,j} e^T_{ij}(\psi_i, \psi_j)\Omega_{ij}e_{ij}(\psi_i, \psi_j)$$

## 5 Integration of OverlapNet into Graph-based SLAM

The odometry measurements are used to construct edges between consecutive vertices (consecutive pose measurements). Due to unreliability and drift of odometry measurements, additional edges between non-consecutive vertices need to be constructed. OverlapNet is used to detect the vertices where loop closure is possible. And the corresponding overlap and yaw values are used to construct edges between the loop closure

vertices. g2o [5], an open-source framework for optimizing graph-based nonlinear error functions, is used to build and optimize the pose-graph. The pseudo code describing the implementation of the integration of OverlapNet into Graph-based SLAM is given below.

```
pose = list containing 4*4 matrices of pose transforms relative to initial pose
covariance = list of 6*6 covariance matrices
optimizer = PoseGraphOptimizer()
while i != total vertices - 1:
    optimizer.add_vertex(i)
    information_matrix = inverse(covariance[i])
    pose_diff = pose[i] relative to pose[i-1]
    optimizer.add_edge([vertex i-1, vertex i],pose_diff, information_matrix)

    search_ellipse = get_covariance_ellipse(i,covariance[i])
    if loop closure detected in search_ellipse:
        overlap, yaw = calculate_overlap_yaw(loop closure vertices)
        del_x, del_y = difference between x and y coordinates odometry
                       measurements between loop closure vertices
        pose_diff = [[ cos(yaw), sin(yaw), 0, del_x],
                     [-sin(yaw), cos(yaw), 0, del_y],
                     [0,       , 0       , 1, 0    ],
                     [0,       , 0       , 0, 1    ]]
        information_matrix = inverse(overlap*[6*6 identity matrix])
        optimizer.add_edge([loop closure vertices], pose_diff, information_matrix)
    i = i+1
optimizer.optimize()
```

# 6 Results and observations

The algorithm described above is tested on one of the KITTI odometry datasets. OverlapNet requires some preprocessing of the input data. After preprocessing, the data can be used infer overlap and yaw values to detect loop closures. The trajectory optimized after integrating OverlapNet into Graph-based SLAM is shown in Fig(5). The blue trajectory is the optimized trajectory whereas the red trajectory is plotted without any optimization. Since the odometry measurements are better in this example, the improvement in the trajectories is not much noticeable. However, OverlapNet is successfull in detecting loop closure for this dataset.

There is one limitation to the integration of OverlapNet in GraphSLAM. As you can see in Fig(6), there are some translational errors in aligning the loop closure vertices. This is because OverlapNet calculates only yaw and overlap probability which gives an estimation of rotational part of the pose transform between loop closure vertices. The translational part of the pose transform (i.e, the 4th column of $4 \times 4$ transformation matrix) is estimated from the odometry measurements which gives rise to the zig-zag shape in the initial part of the trajectory where loop closure is detected as shown in Fig(6).

# 7 Future work

- In this work, the proposed algorithm for integration of OverlapNet and Graph-based SLAM is only tested on a small-scale KITTI dataset in which the odometry readings are not drifted too much. One can test it on a larger dataset and check the detection of loop closures and alignment of the overall trajectory. One of the main challenges of using a larger dataset is obtaining a good covariance data which is accurate enough and does not explodes with time. This is crucial because the loop closures are checked only within the search space estimated from the covariance matrix.
- OverlapNet successfully detects loop closure and calculates overlap and yaw which can be used to estimate the rotational part of the $4 \times 4$ pose transform matrix. However, the translational part of the

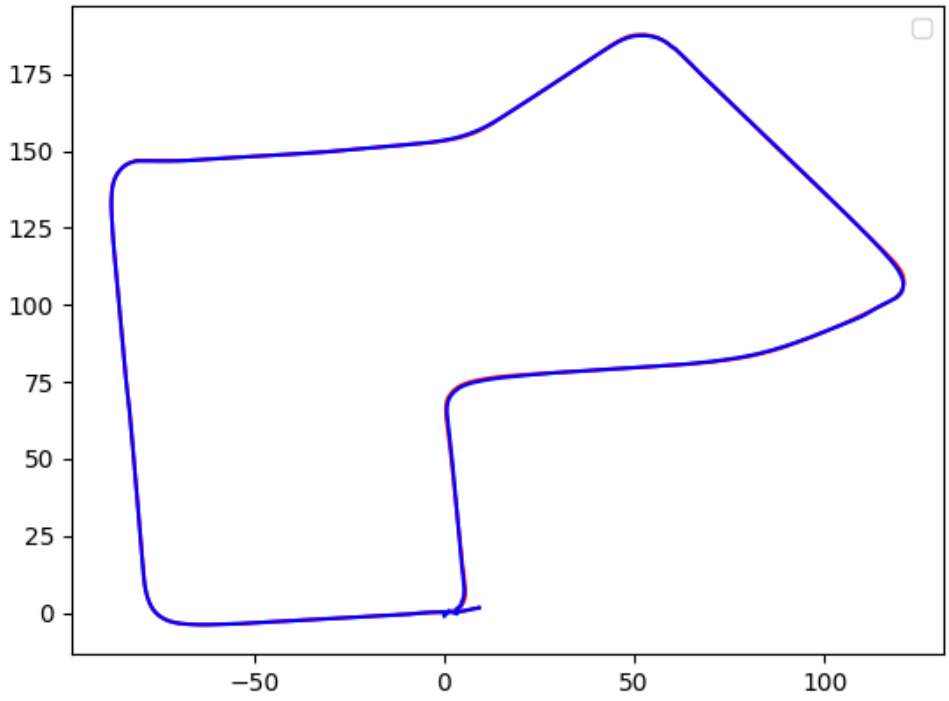


Figure 5: Trajectory obtained from odometry readings (red) and trajectory obtained after optimization (blue)

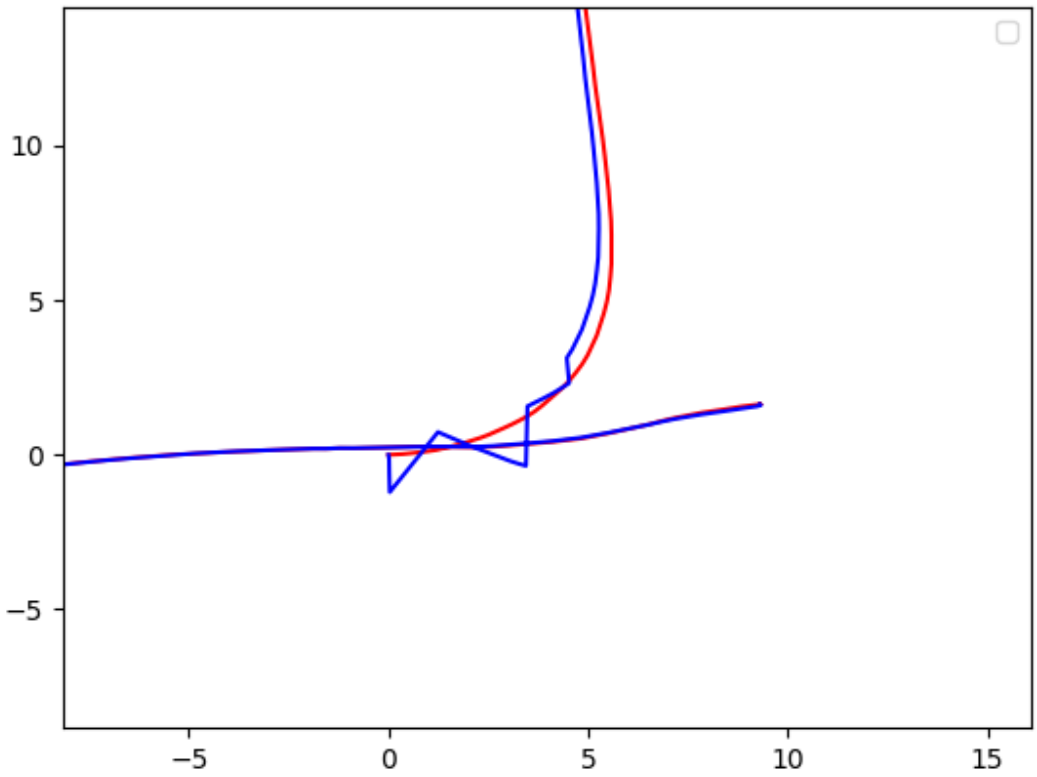


Figure 6: Zoomed-in view of the initial part of the trajectory

pose transform matrix can be estimated by ICP and other scan-matching techniques instead of directly using odometry measurements which often drift with time.

# References


[1] Xieyuanli Chen, Thomas Läbe, Andres Milioto, Timo Röhling, Olga Vysotska, Alexandre Haag, Jens Behley, and C. Stachniss. Overlapnet: Loop closing for lidar-based slam. *ArXiv*, abs/2105.11344, 2020.

[2] Github repository of overlapnet. `https://github.com/PRBonn/OverlapNet`, 2020. Accessed: 2021-06-01.

[3] Andres Milioto, Ignacio Vizzo, Jens Behley, and C. Stachniss. Rangenet ++: Fast and accurate lidar semantic segmentation. *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4213–4220, 2019.

[4] Cyrill Stachniss. Graph-based slam using pose graphs. `https://www.youtube.com/watch?app=desktop&v=uHbRKvD8TWg`, 2020. Accessed: 2021-06-01.

[5] Rainer Kümmerle, Giorgio Grisetti, Hauke Malte Strasdat, Kurt Konolige, and Wolfram Burgard. G2o: A general framework for graph optimization. *2011 IEEE International Conference on Robotics and Automation*, pages 3607–3613, 2011.