DEEPLIO: DEEP LIDAR INERTIAL SENSOR FUSION FOR ODOMETRY ESTIMATION

Arash Javanmard-Gh.[1,*], Dorota Iwaszczuk[1], Stefan Roth[2]

[1] Remote Sensing and Image Analysis, Dept. of Civil and Environmental Engineering Sciences,
Technical University of Darmstadt, Germany, (arash.javanmard-ghareshiran, dorota.iwaszczuk)@tu-darmstadt.de
[2] Visual Inference Lab, Dept. of Computer Science, Technical University of Darmstadt, Germany,
stefan.roth@visinf.tu-darmstadt.de

Commission I, WG I/6

KEY WORDS: Deep Learning, LiDAR Intertial Odometry, Sensor Fusion, Pose Estimation.

ABSTRACT:

Having a good estimate of the position and orientation of a mobile agent is essential for many application domains such as robotics, autonomous driving, and virtual and augmented reality. In particular, when using LiDAR and IMU sensors as the inputs, most existing methods still use classical filter-based fusion methods to achieve this task. In this work, we propose DeepLIO, a modular, end-to-end learning-based fusion framework for odometry estimation using LiDAR and IMU sensors. For this task, our network learns an appropriate fusion function by considering different modalities of its input latent feature vectors. We also formulate a loss function, where we combine both global and local pose information over an input sequence to improve the accuracy of the network predictions. Furthermore, we design three sub-networks with different modules and architectures derived from DeepLIO to analyze the effect of each sensory input on the task of odometry estimation. Experiments on the benchmark dataset demonstrate that DeepLIO outperforms existing learning-based and model-based methods regarding orientation estimation and shows a marginal position accuracy difference.

1. INTRODUCTION

Odometry estimation – i.e. estimating the 3D position and orientation of an agent through time and space – using sensory information such as cameras, light detection and ranging sensors (LiDARs), and inertial measurements units (IMU) is a highly active field of research in computer vision with a wide range of application domains such as autonomous driving (Geiger et al., 2012), robotics (Cadena et al., 2016), building information modeling (BIM) (Roca et al., 2014) and virtual and augmented reality (Klein and Murray, 2007). In one of the first papers on visual odometry (VO) (Nistér et al., 2004) the authors fulfill the VO task by extracting image features (i.e. harris corners) and tracking them frame by frame over time. But the main issue of VO lies both in the scale drift due to scale ambiguity, especially when monocular cameras are used, and the unsoundness of the predictions in the lack of enough features in images. To tackle these problems many researchers start to fuse the VO predictions with other sensory information, like IMU (Nützi et al., 2011, Chu et al., 2012) by applying e.g. Extended Kalman Filter (EKF).

In the case of rangefinder sensors, like LiDARs mainly Iterative Closest Point (ICP) based algorithms (Segal et al., 2010, Besl and McKay, 1992) are used to match consecutive frames by minimizing the distance between the corresponding points in two point clouds. Also, EKF in a tightly (Ye et al., 2019) or loosely (Tang et al., 2015) fashion is here used to fuse the ICP results with the IMU measurements. But unfortunately, most ICP algorithms suffer from 1) Wrong correspondences 2) Inaccurate initialization 3) Lacking of correct covariance matrix 4) High computational power.

Recently some efforts have been made to apply learning-based methods - i.e. neural networks, on LiDAR measurements, to estimate the pose of a mobile agent over time (Wang et al., 2019, Velas et al., 2018). Analogously to these works we propose a supervised learning-based end-to-end trainable fusion method, which utilizes neural networks to extract relevant features from the multi-modal LiDAR and IMU measurements and fuses these latent features to regress the motion encoded in them. To the best of our knowledge, this work is the first comprehensive study of the supervised learning-based fusion method for odometry estimation, which uses both LiDAR and IMU measurements. Particularly our contributions can be summarized as follows:

- Introducing a novel learning-based fusion architecture for 6D pose estimation - i.e. 3D translation and 3D rotation, using LiDAR point cloud and IMU measurements.
- Introducing a local and global windowed loss function.

2. RELATED WORKS

Many research efforts have focused on multi-sensor odometry over last decades. In this section, we give an overview of model- and deep-learning based methods applied to this problem.

2.1 Model-based Lidar-(Inertial) Odometry Estimation

Most of the proposed algorithms for LiDAR-based pose estimation on range data are based on or are related to ICP (Besl and McKay, 1992). Since then, many variants have been suggested to improve the robustness and convergence of the algorithm. In (Segal et al., 2010) the authors formulates the ICP as a general probabilistic framework (GICP), which depending on parameterization permutes to a point-to-point, point-to-plane, or plane-to-plane algorithm. The proposed method in

* Corresponding author

(Serafin and Grisetti, 2015) uses the normal and curvature information of a local surface estimated around a point to find correspondences in an integral image-based fashion. Furthermore, the normal information combined with Cartesian coordinates of the corresponding points is used to estimate the transformation by minimizing the least-squares loss. A comparison to other methods showed that NICP registration offers better results, also it is more robust against poor initial guesses. To improve the ICP-based pose estimation in (Xue et al., 2019) the authors introduce a loosely coupled Extended Kalman-Filter-based IMU-ICP-fusion, where the IMU measurement are used at different processing stages. According to their experiments, the best odometry estimation result is achieved using IMU and a Lidar Odometry and Mapping (LOAM) method (Zhang and Singh, 2014), which is a feature-based point cloud registration approach based on a novel strategy for edge and surface features extraction.

2.2 Learning-based Lidar-(Inertial) Odometry Estimation

The sparsity and unstructured nature of 3D point clouds make the odometry estimation problem more difficult to solve. Hence reliable and accurate 6DoF learning-based pose estimation on point clouds is still an open problem. In (Nicolai et al., 2016) the authors introduce one of the first deep-learning-based odometry methods on point clouds by projecting the point clouds to an image before feeding in their CNN-network for pose estimation. But the presented results were not able to compete with model-based ICP matching methods. Another CNN based architecture proposed in (Velas et al., 2018), also uses a image representation of the input point cloud consisting of three channels (depth, height, intensity). Even though the translation estimation results, formulated as a regression task, were promising, the results on orientation estimation were rather poor, notwithstanding formulating it as a regression or classification task. In contrary to these approaches and inspired by (Yang et al., 2018), (Li et al., 2019) introduces LO-Net , a supervised end-to-end trainable 6DoF pose estimation method based on two consecutive LiDAR frame. Therein odometry is formulated as a 7D regression problem, with a 3D translation vector and a 4D quaternion. Also here the sparse and irregular point clouds are first projected in 2D matrices by cylindrical projection. Both supervised and unsupervised method for the odometry task between two consecutive LiDAR frames is introduced in (Cho et al., 2019). Here the network consists of two separate ResNet (He et al., 2016) like branches, which are trained by 2D matrix projections of the 3D Cartesian coordinate of points (vertex) and their normal vectors. Depending on the selected loss function the network can be trained in supervised or unsupervised mode.

3. DATA REPRESENTATION

3.1 Spherical Projection

A point cloud captured by a LiDAR is an unordered set of 3D Points $\boldsymbol{p}_i = (x, y, z)$. To be able to apply 3D CNN-technique on the point clouds we encode them into a 2D-image. To this end, we use spherical projection (Milioto et al., 2019) by defining the mapping function $\Pi : \mathbb{R}^3 \rightarrow \mathbb{R}^2$ as,

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} \frac{1}{2}[1 - \arctan(y, x)\pi^{-1}]\omega \\ [1 - (\arcsin(zr^{-1}) + f_{up})f^{-1}]h \end{pmatrix} \quad (1)$$

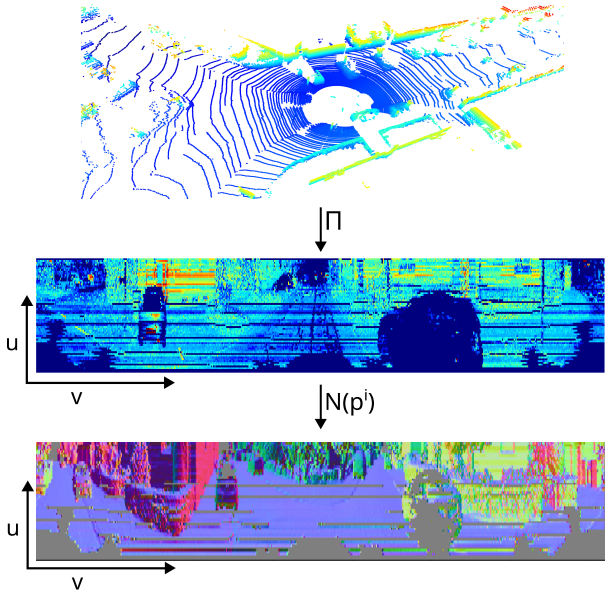

Figure 1. Entire point cloud encoding pipeline starting with, top) input point cloud, middle) discretized point cloud as an image using spherical projection with channels $C = (x, y, z)$, bottom) normal image with channels $C = (n_x, n_y, n_z)$.

(u, v) are the image coordinates, (h, w) are respectivly the desired height and width of the resulting image I, f is the field-of-view (FOV) of the sensor, consisting of an up and down FOV, $f = f_{up} + f_{down}$, and $r = ||\boldsymbol{p}_i||$ is the range of the i-th point in the point cloud.

In most scenarios the mobile agent is moving, while the sensor captures a frame, this result in de-skewed scans where there is no direct one-to-one mapping from a pixel coordinates $(u, v)_i$ to a point $\boldsymbol{p}_i$, i.g. multiple points fall in the same image pixel. For this reason, we order the point cloud in descending range order, and the closest points to the LiDAR sensor is assigned according to (Li et al., 2019, Milioto et al., 2019). The resulting image $\boldsymbol{I} \in \mathbb{R}^{M \times N \times C}$ can be augmented with multiple channels C. In this work we set $C = (x, y, z)$, see fig. 1.

3.2 Normal Estimation

As shown in (Serafin and Grisetti, 2015) normal vectors are strong features for point cloud registration. A Commonly used strategy to calculate the normal vector $\boldsymbol{n}_i$ at the point $\boldsymbol{p}_i$ is to sample some points in its neighborhood $\mathcal{N}_i = [p_{i,0}, p_{i,1}, p_{i,j}], j = [1 \dots n]$, build the covariance matrix and calculate the normal vector based on the Eigendecomposition. Corresponding to the formulation in (Li et al., 2019), given a point $\boldsymbol{p}_i = [x, y, z]_i$ and it's N neighbors $\boldsymbol{p}_{i,j}, j = 1, 2, \dots, N$. We can estimate the normal $\mathcal{N}(\boldsymbol{p}_i) := \boldsymbol{n}_i \in \mathbb{R}^3$ by,

$$\Delta \boldsymbol{p}_j^i = \boldsymbol{p}_j^i - \boldsymbol{p}^i \in \mathbb{R}^{3 \times 1} \quad (2)$$

$$\boldsymbol{D}_i = [\Delta \boldsymbol{p}_1^i, \Delta \boldsymbol{p}_2^i, \dots, \Delta \boldsymbol{p}_N^i] \in \mathbb{R}^{3 \times N} \quad (3)$$

$$\underset{\mathcal{N}(\boldsymbol{p}_i)}{arg\,\min} = ||\boldsymbol{D}_i^T \boldsymbol{n}_i|| \quad (4)$$

Instead of directly solving the objective defined in eq. (4), we use a simplified normal estimation method same as (Moosmann, 2013), where the normal at each point of an encoded point cloud is calculated by computing the weighted cross

products over $\boldsymbol{p}_i$'s four neighbors as defined by,

$$\mathcal{N}(\boldsymbol{p}_i) := \hat{\boldsymbol{n}}_i = \sum_{\boldsymbol{p}_k^i, \boldsymbol{p}_j^i \in \mathcal{S}} (\omega_k^i \Delta \boldsymbol{p}_k^i \times \omega_j^i \Delta \boldsymbol{p}_j^i) \tag{5}$$

$$\boldsymbol{n}_i = \frac{\hat{\boldsymbol{n}}_i}{||\hat{\boldsymbol{n}}_i||} \tag{6}$$

where $\mathcal{S}$ is a set of sorted four neighbors $\{\boldsymbol{p}_i^1, \boldsymbol{p}_i^2, \boldsymbol{p}_i^3, \boldsymbol{p}_i^4\}$ surrounding $\boldsymbol{p}_i$. To make sure that faraway neighboring points-e.g. around the borders of an object, do not distort the normal estimation, each distance vector $\Delta \boldsymbol{p}_j^i$ is weighted by ω_j^i. The weighting function $\phi(\boldsymbol{x}) : \mathbb{R}^3 \rightarrow \mathbb{R}$ is defined by,

$$\omega_j^i|_{\boldsymbol{x}=\Delta \boldsymbol{p}_j^i} := \phi(\boldsymbol{x}) = e^{-\alpha||\boldsymbol{x}||} \tag{7}$$

where the weighting is controlled by the parameter α.

Figure 1 shows the entire encoding pipeline, where a point cloud is first encoded as an image $I(u, v)$ with three channels $C = [x, y, z]$, and subsequently, the normal vectors are estimated using the x, y, z information at each pixel of this image.

4. DEEPLIO ARCHITECTURE

We introduce a cascade modular network architecture, that learns a robust fusion of LiDAR frames and IMU measurements for the task of relative pose estimation between a pair of consecutive LiDAR frames $\mathcal{F}_{t-1}$ and $\mathcal{F}_t$. The input to the network is a sequence of consecutive LiDAR frames and the corresponding IMU measurements in the same time interval. In essence, our network learns to extract appropriate features from both sensory information and subsequently learns a sophisticated fusion strategy suitable for mapping from both LiDAR and IMU feature space to the space of $se(3)$ defined by,

$$\begin{aligned} DeepLIO : \{(\mathbb{R}^{H \times W \times 2C}, \mathbb{R}^{S_{imu} \times 6})_{1:S}\} \\ \rightarrow \{(se(3))_{1:S}\} \end{aligned} \tag{8}$$

where (H, W, C) are the height, width, and the number of channels of the projected LiDAR frames and S_{imu} is the sequence length of IMU data measured during two consecutive LiDAR frames.

As you can see in fig. 2 the network consists of three main modules.

1. **Feature-Nets**: This module itself consists of two following sub-modules for feature extraction.
   - **LiDAR Feature Network**: This module is responsible for extracting and encoding the LiDAR frames which are first transformed by spherical projection.
   - **IMU Feature Network**: This module is responsible for extracting and encoding the IMU measurements, which consists of linear acceleration and angular velocity (dim=6).
2. **Fusion Network**: This module is responsible for fusing the features extracted from LiDAR and IMU feature networks.
3. **Odometry Network**: At least in this module the fused features are used to learn the hidden state, which shall explain the odometry information encoded in these features.

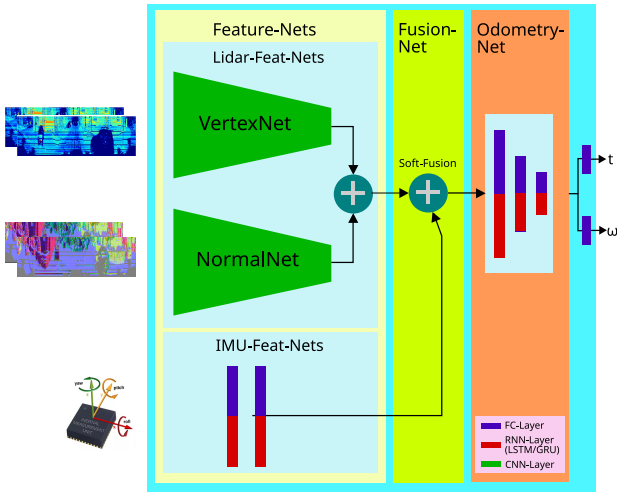

Figure 2. Architecture overview and data stream of the proposed DeepLIO network.

4.1 LiDAR Feature Network

Similar to (Cho et al., 2019) we also construct the *LiDAR-Feature-Nets* as a siamese network with each branch referred to as *VertexNet* (top branch) and *NormalNet* (bottom branch), but in contrast to (Cho et al., 2019) we use a more lightweight network based on fire and squeeze-and-excitation modules introduced in (Wang et al., 2018, Iandola et al., 2017, Hu et al., 2018).

4.2 IMU Feature Network

Usually, IMUs operate at a much higher frequency than LiDAR, which results in a sequence of IMU measurements (S_{imu}) during a full LiDAR spin. The IMU sub-network takes this sequence of raw IMU measurements $\boldsymbol{m}_{(t-1,t)} = [\boldsymbol{\alpha}, \boldsymbol{\omega}] \in \mathbb{R}^{S_{imu} \times 6}$ sampled between two consecutive frames as input and learns the dynamic encoded in this sequence for the task of pose estimation. For this reason, we experiment with three network configurations, one made of *fully connetected*-layers and the other two constructed by LSTM and GRU modules (Hochreiter and Schmidhuber, 1997, Cho et al., 2014).

4.3 Fusion Network

Concatenating different features together is a standard fusion strategy (Clark et al., 2017) with the drawback that different modalities and noise characteristics are not considered. For instance, concatenating feature vectors of multiple redundant sensory information without considering their reliability under certain environments may result in a sub-optimal and not robust solution. To remedy this issue we introduce an attention-based (Vaswani et al., 2017) soft-fusion method inspired and based on (Chen et al., 2019) for fusing LiDAR and IMU features.

The sequential features extracted from both *LiDAR-Feature-Net* and *IMU-Feature-Net* form the input to the fusion-layer. A key requirement on the fusion method is that it should be differentiable function so that it can be deployed in an end-to-end trainable model. Conditioning on both input feature vectors and similar to (Chen et al., 2019) a self-adaptive soft-fusion function is defined by the two weighting functions,

$$\boldsymbol{s}_L = \sigma(\boldsymbol{W}_L|[\boldsymbol{a}_L, \boldsymbol{a}_I]) \tag{9}$$

$$\boldsymbol{s}_I = \sigma(\boldsymbol{W}_I|[\boldsymbol{a}_L, \boldsymbol{a}_I]) \tag{10}$$

where $[.,.]$ denotes the concatenation operation, σ is the sigmoid-function and $\boldsymbol{W}_{[L,I]}$ are the corresponding weights. $\boldsymbol{s}_L$ and $\boldsymbol{s}_I$ form the weights corresponding to each element of combined LiDAR and IMU feature vector. Finally, we multiply these weights with their corresponding feature vector for an automatic recalibration and feature selection,

$$\boldsymbol{f}_f : \mathbb{R}^{K_L} \times \mathbb{R}^{K_I} \rightarrow \mathbb{R}^{K_L+K_I} \tag{11}$$

$$\boldsymbol{a}_f = \boldsymbol{f}_f(\boldsymbol{\theta}_f|(\boldsymbol{a}_L, \boldsymbol{a}_I)) = [\boldsymbol{s}_L \otimes \boldsymbol{a}_L, \boldsymbol{s}_I \otimes \boldsymbol{a}_I] \tag{12}$$

where $\boldsymbol{f}_f$ characterizes the fusion function with all parameters accommodated in $\boldsymbol{\theta}_f$.

4.4 Odometry Network

The input to this module is a sequence of fused LiDAR and IMU features. Hence the network should be constructed so that it can learn reasonable temporal correlations encoded in its input sequence. For this reason, we construct this sub-network using LSTM-modules (Hochreiter and Schmidhuber, 1997).

5. LOSS FUNCTIONS

So far, we have defined the inputs, outputs, and network architecture of DeepLIO, in this section we introduce the structure of the loss function used to train the network.

5.1 Global and Local Loss

Learning to predict a robust estimation of two different quantities, namely rotation and translation is a challenging task. Even if we assume that the network can predict the relative motion between two frames reasonably good with a small deviation, by the effect of propagation of uncertainty, accumulating these local predictions still result in an inaccurate global trajectory estimation. Therefore the loss function should be designed so that it enforces the network to learn reasonable features for a robust local and global pose estimation.

Given a sequence S of pairwise stacked and projected LiDAR frames $\boldsymbol{l}_i \in \mathbb{R}^{H \times W \times 2C}$ and their corresponding IMU measurments $\boldsymbol{m}_i \in \mathbb{R}^{S_{imu} \times 6}$ from a dataset, the input $\boldsymbol{X}$ and the ground truth poses $\boldsymbol{Y}$ are defined by,

$$\boldsymbol{X} = \{\boldsymbol{x}_i | \boldsymbol{x}_i = (\boldsymbol{l}_i, \boldsymbol{m}_i), i = [0, \cdots, S]\} \tag{13}$$

$$\boldsymbol{Y}_l = \{\boldsymbol{y}_i^l | \boldsymbol{y}_i^l = (\boldsymbol{t}_i, \boldsymbol{\omega}_i) \in se(3), i = [0, \cdots, S]\} \tag{14}$$

$$\boldsymbol{Y}_g = \{\boldsymbol{y}_i^g | \boldsymbol{y}_i^g = (\boldsymbol{p}_i, \boldsymbol{q}_i), \boldsymbol{p}_i \in \mathbb{R}^3, \boldsymbol{q}_i \in \mathbb{H}, i = [0, \cdots, S]\} \tag{15}$$

Where $\boldsymbol{Y}_l$ is the set of local relative motions between each consecutive frames and $\boldsymbol{Y}_g$ the set of global motions occurred between the first frame and other frames in the sequence.

Furthermore let us denote the network local motion prediction at the timestamp i as $(\hat{\boldsymbol{t}}_i, \hat{\boldsymbol{\omega}}_i) = \boldsymbol{f}^{\boldsymbol{\theta}}(\boldsymbol{x}_i) \in se(3)$, where $\boldsymbol{f}^{\boldsymbol{\theta}}$ is the neural network mapping function with $\boldsymbol{\theta}$ being the state of all parameters learned by the network during the training. The set of all local motion predictions of the network for a given input sequence S is defined by,

$$\hat{\boldsymbol{Y}}_l = \{\hat{\boldsymbol{y}}_i^l | \hat{\boldsymbol{y}}_i^l = (\hat{\boldsymbol{t}}_i, \hat{\boldsymbol{\omega}}_i) \in se(3), i = [0, \cdots, S]\} \tag{16}$$

Based on these definitions and w.l.o.g. we can convert these local motion estimations to global estimations regarding the first frame in the sequence. Building upon that, we define a set of global motion predictions by,

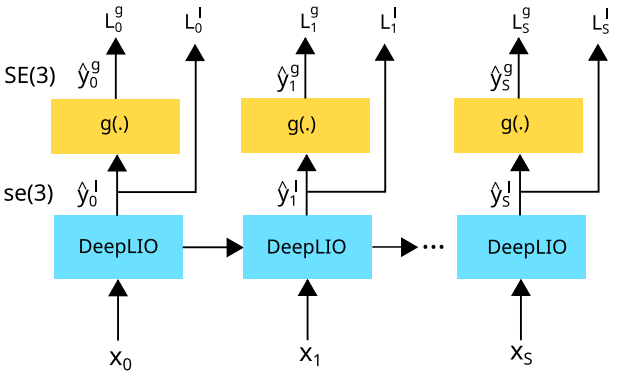

Figure 3. Schematic representation of the local and global loss calculated based on the local predictions of DeepLIO given a sequence of input data.

$$g : [\hat{\boldsymbol{y}}_i^l]_{i=0..n} \rightarrow \hat{\boldsymbol{y}}_n^g \tag{17}$$

$$\hat{\boldsymbol{Y}}_g = \{\hat{\boldsymbol{y}}_i^g | \hat{\boldsymbol{y}}_i^g = (\hat{\boldsymbol{p}}_i, \hat{\boldsymbol{q}}_i), \hat{\boldsymbol{p}}_i \in \mathbb{R}^3, \hat{\boldsymbol{q}}_i \in \mathbb{H}, i = [0, \cdots, S]\} \tag{18}$$

where $\hat{\boldsymbol{y}}_i^g = (\hat{\boldsymbol{p}}_i, \hat{\boldsymbol{q}}_i)$ are the corresponding global position and orientation estimations at the timestamp i, calculated using function g, which transforms a set of local motions to a global motion with respect to the first element of the input set. Endowed with this information, we can define the local $\mathcal{L}^l(\hat{\boldsymbol{y}}_i^l, \boldsymbol{y}_i^l)$ and global $\mathcal{L}^g(\hat{\boldsymbol{y}}_i^g, \boldsymbol{y}_i^g)$ loss as two functions that measure the discrepancy between the predicted motion and the ground truth. Figure 3 demonstrates the relation between the network predictions, the global and local loss functions, and a sequence of input data. An important aspect of designing the loss function is the representation of the output quantities the network has to predict. Even though the amount of a position displacement between two frames can be expressed in a Euclidean space without any effort, representing rotational movement is not as straight-forward, since a rotation can be presented among other forms as Euler-angles, rotation-matrices, and quaternions, but unfortunately, each of them has its advantages and disadvantages, for instance, only three parameters are needed to represent Euler-Angles, but they do not provide a unique parameterization and suffer from gimbal lock, in contrast, rotation-matrices are unique but they are overparameterized and the orthogonality constraint hampers the optimization. Regarding the number of parameters, quaternions are a compromise, but they need to be normalized into a unit length.

For these reasons and under the assumption that relative motion between two frames is supposed to be small, in this work, we represent the local frame-to-frame motion in the space of $se(3)$, since rotations are not constrained in this representation. At the other end due to the above reasons the global trajectory is represented as a quaternion $q \in \mathbb{H}$. Hence local and global loss functions are defined by,

$$\mathcal{L}_t(\hat{\boldsymbol{y}}_i^l) = \left\|\boldsymbol{t}_i - \hat{\boldsymbol{t}}_i\right\|_\gamma \tag{19}$$

$$\mathcal{L}_\omega(\hat{\boldsymbol{y}}_i^l) = \left\|\boldsymbol{\omega}_i - \hat{\boldsymbol{\omega}}_i\right\|_\gamma \tag{20}$$

$$\mathcal{L}_p(\hat{\boldsymbol{y}}_i^g) = \left\|\boldsymbol{p}_i - \hat{\boldsymbol{p}}_i\right\|_\gamma \tag{21}$$

$$\mathcal{L}_q(\hat{\boldsymbol{y}}_i^g) = \left\|\boldsymbol{q}_i - \frac{\hat{\boldsymbol{q}}_i}{||\hat{\boldsymbol{q}}_i||}\right\|_\gamma \tag{22}$$

This contribution has been peer-reviewed. The double-blind peer-review was conducted on the basis of the full paper.
https://doi.org/10.5194/isprs-annals-V-1-2021-47-2021 | © Author(s) 2021. CC BY 4.0 License.

Where $\|.\|_\gamma$ determines the distance norm. In this work we set $\gamma = 2$ for all loss functions.

5.2 Homoscedastic Weighted Sum Loss

Learning both translation and rotation simultaneously is due to the unit and scale differences between these quantities indeed challenging. Therefore for the loss function we need to find a sophisticated balancing strategy between both tasks. In (Kendall et al., 2015, Wang et al., 2019, Wang et al., 2017) the authors suggest a linearly weighted sum loss, with the weighting as a hyperparameter. Since an optimal balancing value is not known in advance, we need to find a sophisticated hyperparameter tuning strategy.

In contrast to the linearly weighted loss function and inspired by (Kendall and Cipolla, 2017) we define the loss function as a Homoscedastic Weighted Sum Loss. Homoscedastic uncertainty is one of the two subcategories of *Aleatoric* uncertainty (Kendall and Gal, 2017), which stays constant for different inputs but varies between different tasks (Kendall et al., 2017). One major advantage of this loss is that all hyperparameters used for balancing between each task are now learnable, hence they can be learned during the training.

Assuming Gaussian likelihood we can model the network outputs for a single local pose prediction as,

$$p(\boldsymbol{y}_i|\hat{\boldsymbol{y}}_i = f^{\boldsymbol{\theta}}(\boldsymbol{x}_i)) = \mathcal{N}(\boldsymbol{y}_i|\hat{\boldsymbol{y}}_i, \sigma^2) \tag{23}$$

Where $\boldsymbol{y}_i = (\boldsymbol{p}_i, \boldsymbol{q}_i)$ is the i-th ground truth pose, $\hat{\boldsymbol{y}}_i = (\hat{\boldsymbol{p}}_i, \hat{\boldsymbol{q}}_i)$ is the appropriate network pose estimation, and $\mathcal{N}$ is a Gaussian distribution with the mean $\hat{\boldsymbol{y}_i}$ and the variance σ. From now on we ignore the i-subscript to enable a better overview.

Given the network outputs, position $\boldsymbol{p}_i$ and orientation $\boldsymbol{q}_i$ are conditionally independent, hence reformulating the likelihood and substituting the results in a Gaussian distribution yields,

$$p(\boldsymbol{y}|\hat{\boldsymbol{y}}) = p(\boldsymbol{p}|\hat{\boldsymbol{p}})p(\boldsymbol{q}|\hat{\boldsymbol{q}}) \tag{24}$$

$$= \mathcal{N}(\boldsymbol{p}|\hat{\boldsymbol{p}}, \sigma_p^2)\mathcal{N}(\boldsymbol{q}|\hat{\boldsymbol{q}}, \sigma_q^2) \tag{25}$$

In Maximum-Likelihood-Methode (MLE) rather than maximizing the likelihood function itself, we maximize the *log* of it, since *log* is a monotonic function, the same parameters which maximize a specific function will also maximize its *log*.

The *log* of a Gaussian distribution is defined by,

$$\log \mathcal{N}(x|\mu, \sigma^2) = \log\left(\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{\|x-\mu\|^2}{2\sigma^2}}\right) \tag{26}$$

$$= -2\|x-\mu\|^2 \sigma^{-2} - \log\sigma - const. \tag{27}$$

We can now formulate the loss function $\mathcal{L}(\boldsymbol{\theta}, \sigma_p, \sigma_q)$ as a function of the network parameters $\boldsymbol{\theta}$ and $\sigma_{(p,q)}$ as,

$$\mathcal{L}(\boldsymbol{\theta}, \sigma_p, \sigma_q) = -\log p(\boldsymbol{y}|\hat{\boldsymbol{y}}) \tag{28}$$

$$= \frac{1}{2}\|\boldsymbol{p} - \hat{\boldsymbol{p}}\|^2 \sigma_p^{-2} + \log\sigma_p \tag{29}$$

$$+ \frac{1}{2}\left\|\boldsymbol{q} - \frac{\hat{\boldsymbol{q}}}{\|\hat{\boldsymbol{q}}\|}\right\|^2 \sigma_q^{-2} + \log\sigma_q \tag{30}$$

$$\propto \mathcal{L}_p\sigma_p^{-2} + \log\sigma_p + \mathcal{L}_q\sigma_q^{-2} + \log\sigma_q \tag{31}$$

Where the constant terms are neglected since they do not change the optimization result. Please note that this loss function now also depends on the weighting hyperparameters (σ_p, σ_q), hence we do not need to set them manually. Furthermore, these hyperparameters now represent the model's observation noise, i.e. they capture how much noise we have in the outputs (Kendall et al., 2017). Furthermore due to numerical stability and to avoid the division by zero in (Kendall and Cipolla, 2017) the authors suggest log variance $s := \log\sigma$.

By extending this method and combining both the local and global loss functions, the final *Homoscedastic Weighted Sum Loss* (HWS-Loss) is defined by,

$$\mathcal{L}(\boldsymbol{\theta}, \sigma_p, \sigma_q||\boldsymbol{X}) = \sum_{i=0}^{S}((\mathcal{L}_{p,i} + \mathcal{L}_{t,i})e^{-s_1} + s_1 \tag{32}$$

$$+ (\mathcal{L}_{\omega,i} + \mathcal{L}_{q,i})e^{-s_2} + s_2) \tag{33}$$

Where (s_1, s_2) are the observation uncertainties corresponding to position and orientation predictions.

6. EXPERIMENTS

6.1 KITTI Odometry Dataset

In this work we use the KITTI Odometry dataset (Geiger et al., 2013) to train and test DeepLIO [1]. The dataset consists of 22 sequences from which 11 sequences are provided with ground truth for training and testing. Each sequence is a trajectory captured by driving an augmented car in Karlsruhe, Germany. LiDAR frames are captured by a Velodyne HDL-64e with 64-beams and a sampling frequency of 10Hz. Also, an IMU with a sampling frequency of 100Hz and a GPS module is used to calculate the ground truth pose.

6.2 Implementation details

Using the aforementioned spherical projection we encode each input point cloud to an image matrix by setting $W = 720$ and $H = 64$ pixels, furthermore we set $alpha = 0.8$ for normal image calculation. During the training, we build a sequence of five paired consecutive frames and their corresponding IMU measurements. Furthermore, to assess the capability of the proposed DeepLIO network to improve the odometry results by fusing both partly redundant sensor information, we evaluate and compare the accuracy of three different sub-network architectures, which we denote as,

- **DeepIO**: This network uses only IMU measurements to predict the resulting odometry.
- **DeepLO**: This network uses only LiDAR images and normals to estimate the odometry.
- **DeepLIO**: this network uses both LiDAR as well as IMU measurements and fuses them to estimate the odometry.

In order to facilitate comparison with other learning-based odometry methods, we use sequences $00 - 08$ for training and $09-10$ for test. We also evaluate the results by utilizing the metric defined in the KITTI dataset, which is the average translation $t_{rel}(\%)$ and rotation $r_{rel}(^\circ/100m)$ RMSE on the distance of 100m-800m. We employed the Adam solver (Kingma and Ba, 2014) with $\beta_1 = 0.9$, $\beta_2 = 0.99$ and $w_{decay} = 10^{-5}$ for all aforementioned network configurations. We set the batch-size and the sequence length to $B = 8$ and $S = 5$. We implemented the entire framework using PyTorch (Paszke et al., 2019).

[1] Our code is available at `https://github.com/ArashJavan/DeepLIO.git`

Table 1. Odometry results of DeepIO networks on KITTI dataset.

DeepIO		Sequences 00	01	02	04	05	06	07	08	09	10	*mean*†	*mean**
t_{rel}	FCN	6.38	3.28	4.86	2.78	5.03	6.84	6.86	6.93	12.51	14.49	5.37	13.49
	LSTM	**2.15**	1.90	**2.42**	**3.25**	**2.20**	**2.35**	**2.96**	**3.33**	9.70	9.46	**2.56**	9.58
	GRU	2.37	**1.66**	3.02	5.84	2.57	3.93	3.60	4.13	**9.03**	**8.76**	3.39	**8.89**
r_{rel}	FCN1	2.44	**0.49**	1.50	**0.32**	1.87	2.19	3.30	2.17	1.86	3.62	1.87	2.74
	LSTM	0.69	0.50	0.60	0.60	0.65	0.42	0.68	0.82	0.64	**0.79**	0.62	**0.71**
	GRU	**0.47**	0.60	**0.46**	0.45	**0.32**	**0.34**	**0.48**	**0.45**	**0.52**	0.92	**0.44**	0.72

† Average translational $t(\%)$ and rotational $r(^\circ/100)$ RMSE on training set.
* Average translational $t(\%)$ and rotational $r(^\circ/100)$ RMSE on test set.

Table 2. Odometry result of DeepLO networks on KITTI dataset.

DeepLO		Sequences 00	01	02	04	05	06	07	08	09	10	*mean*†	*mean**
t_{rel}	PointSeg	**2.22**	**1.76**	**2.61**	**1.14**	**1.75**	**2.84**	**0.93**	**2.27**	**10.52**	**10.15**	**1.94**	**10.33**
	FlowNet	16.07	22.79	9.30	1.79	14.02	21.24	20.48	15.67	14.48	13.60	15.17	14.04
r_{rel}	PointSeg	**1.0**	**0.8**	**1.04**	**1.03**	**0.9**	**1.18**	**0.86**	**0.96**	**4.05**	**4.42**	**0.97**	**4.23**
	FlowNet	6.37	4.55	3.50	**1.72**	6.06	7.67	11.78	6.04	5.87	9.24	5.96	7.55

† Average translational $t(\%)$ and rotational $r(^\circ/100)$ RMSE on training set.
* Average translational $t(\%)$ and rotational $r(^\circ/100)$ RMSE on test set.

6.3 Evaluating DeepIO

By deploying the deep inertial odometry network (DeepIO) we aim to investigate the ability of different only IMU-based networks to extract the hidden information about the dynamic motion of the mobile agent from the IMU measurements to predict the current pose of the agent w.r.t. its previous pose. To this end, we designed three different network configurations. The first one is only built by fully-connected layers, where experimented with different hidden-layer settings. The next two networks are built by LSTM and GRU units. As the evaluation results in table 1 shows, both networks based on recurrent neural networks (LSTM, GRU) outperformed the fully-connected network, which confirms our assumption about the capabilities of these modules to learn the hidden dynamic in their inputs better.

6.4 Evaluating DeepLO

Consistent with the last section, we evaluated the deep lidar odometry network (DeepLO) as the next step. For this, we configured the DeepLIO-framework, in a way that only LiDAR-Feature Networks are activated and used. Furthermore, we deployed two architectures, one based on the FlowNet (Fischer et al., 2015) and the other based on PointSeg (Wang et al., 2018) for both siamese Vertex- and Normal-Net feature encoders. As the results in table 2 shows, the network based on PointSeg is able to learn and extract relevant features from the LiDAR images for odometry estimation, where the network based on FlowNet does not converge to a good solution.

6.5 Evaluating DeepLIO

In the last sections, we presented and analyzed different network combinations to investigate the effect of each sensory information and to measure their prediction strength. Now it is time to fuse the strengths of each of these networks into a one-unit network by utilizing the proposed fusion framework. Also in this section, we evaluate and compare DeepLIO with other ICP, Deep learning, and optimization based methods. We reduced the training time by the means of *transfer learning*, based on the already pre-trained sub-networks introduced in the last two sections.

Table 3 shows, that compared to both ICP based methods point-to-point and point-to-plane DeepLIO achieves better results across all sequences. On the other hand, regarding position estimation, LOAM and LO-Net achieve slightly better results throughout all sequences. But regarding orientation estimation, DeepLIO outperforms all other methods. Furthermore comparing the results in table 3 with table 1 and table 2 we can see, that the DeepLIO network was able to learn a sophisticated mapping to fuse both feature streams to improve its estimation capabilities compared to the other sub-networks. The estimated global trajectories in fig. 4 validate these results.

7. CONCLUSION AND OUTLOOK

In this paper, we proposed a novel modular deep-learning-based fusion framework *DeepLIO* for robust odometry estimation using LiDAR and IMU sensors. Furthermore, also a loss function based on homeostatic uncertainty, which also considers both global and local motion was introduced. In contrast to most other deep-learning fusion methods, where two state vectors are concatenated together, we proposed a self-adaptive fusion strategy based on (Chen et al., 2019), so that the fusion-layer can learn to weight each input based on its modality, which is learned during the training. We also showed that our method evaluated on the KITTI odometry dataset outperforms both learning-based and model-based methods regarding orientation estimation and shows minor differences regarding position estimation. A reason for this may be the considerable amount of information loss during the quantization process of the point clouds. Therefore in a further extension of this work, we will investigate the effect of using raw point clouds and scene flow estimation on the accuracy of odometry estimation.

Table 3. Comparison of odometry results of DeepLIO and other methods.

		Sequences											
		00	01	02	04	05	06	07	08	09	10	*mean*†	*mean**
t_{rel}	DeepLIO	1.6	5.9	1.96	3.7	1.24	1.97	1.92	2.34	4.4	4.0	2.57	4.2
	ICP-po2po	10.54	88.48	11.8	99	14.5	21.51	25.49	10.61	10.83	20.71	#	31.34
	ICP-po2pl	6.77	91.9	6.89	98.61	4.77	2.9	4.71	6.48	7.79	6.76	#	23.75
	LOAM	**1.1**	2.79	1.54	1.45	0.75	0.72	**0.69**	**1.18**	**1.2**	**1.51**	#	**1.29**
	LO-Net	1.47	**1.36**	**1.52**	**0.51**	1.04	**0.71**	1.7	2.12	1.37	1.8	**1.3**	1.58
r_{rel}	DeepLIO	**0.38**	**0.19**	**0.23**	**0.12**	**0.21**	**0.14**	**0.32**	**0.34**	**0.21**	**0.51**	**0.19**	**0.36**
	ICP-po2po	4.72	11.52	4.06	4.17	5.65	4.25	12.94	4.47	3.96	6.58	#	6.23
	ICP-po2pl	2.84	10.83	2.4	1.13	2.09	1.28	2.72	2.45	2.74	2.66	#	3.1
	LOAM	0.53	0.55	0.55	0.5	0.38	0.39	0.5	0.44	0.48	0.57	#	0.48
	LO-Net	0.72	0.47	0.71	0.65	0.69	0.5	0.89	0.77	0.58	0.93	0.67	0.75

† Average translational $t(\%)$ and rotational $r(^\circ/100)$ RMSE on training set.
* Average translational $t(\%)$ and rotational $r(^\circ/100)$ RMSE on test set.
# These methods are not trainable, so that we take their overall odometry results as a test reuslt.

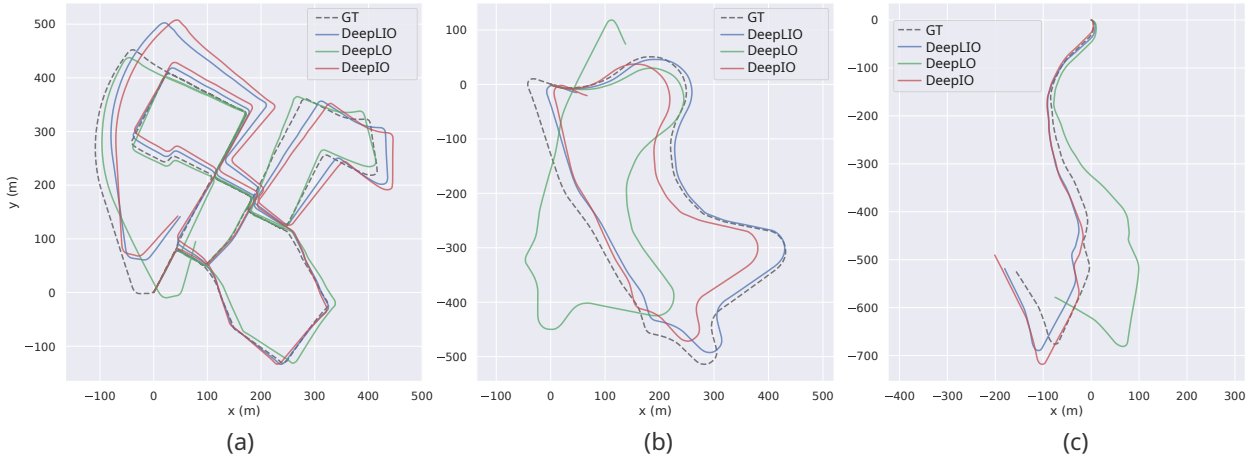

Figure 4. Trajectory comparison of DeepLIO, DeepLO and DeepIO, a) Seq. 00 of the training set, b,c) Seq. 09-10 of the test set plotted using (Grupp, 2017)

.

REFERENCES

Besl, P. J., McKay, N. D., 1992. A Method for Registration of 3-D Shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Cadena, C., Carlone, L., Carrillo, H., Latif, Y., Scaramuzza, D., Neira, J., Reid, I., Leonard, J. J., 2016. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on Robotics*.

Chen, C., Rosa, S., Miao, Y., Lu, C. X., Wu, W., Markham, A., Trigoni, N., 2019. Selective sensor fusion for neural visual-inertial odometry.

Cho, K., van Merrienboer, B., Bahdanau, D., Bengio, Y., 2014. On the properties of neural machine translation: Encoder-decoder approaches.

Cho, Y., Kim, G., Kim, A., 2019. Deeplo: Geometry-aware deep lidar odometry.

Chu, T., Guo, N., Backén, S., Akos, D., 2012. Monocular camera/IMU/GNSS integration for ground vehicle navigation in challenging GNSS environments. *Sensors*.

Clark, R., Wang, S., Wen, H., Markham, A., Trigoni, N., 2017. Vinet: Visual-inertial odometry as a sequence-to-sequence learning problem.

Fischer, P., Dosovitskiy, A., Ilg, E., Häusser, P., Hazırbaş, C., Golkov, V., van der Smagt, P., Cremers, D., Brox, T., 2015. Flownet: Learning optical flow with convolutional networks.

Geiger, A., Lenz, P., Stiller, C., Urtasun, R., 2013. Vision meets Robotics: The KITTI Dataset. *International Journal of Robotics Research (IJRR)*.

Geiger, A., Lenz, P., Urtasun, R., 2012. Are we ready for autonomous driving? the kitti vision benchmark suite. *2012 IEEE Conference on Computer Vision and Pattern Recognition*, 3354–3361.

Grupp, M., 2017. evo: Python package for the evaluation of odometry and slam.

He, K., Zhang, X., Ren, S., Sun, J., 2016. Deep residual learning for image recognition. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*.

Hochreiter, S., Schmidhuber, J., 1997. Long Short-term Memory. *Neural computation*, 9, 1735-80.

Hu, J., Shen, L., Sun, G., 2018. Squeeze-and-Excitation Networks. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*.

Iandola, F. N., Moskewicz, M. W., Ashraf, K., Han, S., Dally, W., Keutzer, K., 2017. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and ¡1MB model size. *ArXiv*, abs/1602.07360.

Kendall, A., Cipolla, R., 2017. Geometric loss functions for camera pose regression with deep learning.

Kendall, A., Gal, Y., 2017. What uncertainties do we need in bayesian deep learning for computer vision?

Kendall, A., Gal, Y., Cipolla, R., 2017. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics.

Kendall, A., Grimes, M., Cipolla, R., 2015. Posenet: A convolutional network for real-time 6-dof camera relocalization.

Kingma, D. P., Ba, J., 2014. Adam: A method for stochastic optimization.

Klein, G., Murray, D., 2007. Parallel tracking and mapping for small AR workspaces. *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality, ISMAR*.

Li, Q., Chen, S., Wang, C., Li, X., Wen, C., Cheng, M., Li, J., 2019. Lo-net: Deep real-time lidar odometry.

Milioto, A., Vizzo, I., Behley, J., Stachniss, C., 2019. Rangenet ++: Fast and accurate lidar semantic segmentation. *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 4213–4220.

Moosmann, F., 2013. Interlacing self-localization, moving object tracking and mapping for 3d range sensors.

Nicolai, A., Skeele, R., Eriksen, C., Hollinger, G. A., 2016. Deep learning for laser based odometry estimation.

Nistér, D., Naroditsky, O., Bergen, J., 2004. Visual odometry. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*.

Nützi, G., Weiss, S., Scaramuzza, D., Siegwart, R., 2011. Fusion of IMU and vision for absolute scale estimation in monocular SLAM. *Journal of Intelligent and Robotic Systems: Theory and Applications*.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S., 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems 32*.

Roca, D., Armesto, J., Lagüela, S., Díaz Vilariño, L., 2014. Lidar-equipped uav for building information modelling. XL-5.

Segal, A. V., Haehnel, D., Thrun, S., 2010. Generalized-ICP. *Robotics: Science and Systems*.

Serafin, J., Grisetti, G., 2015. Nicp: Dense normal based point cloud registration. *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 742–749.

Tang, J., Chen, Y., Niu, X., Wang, L., Chen, L., Liu, J., Shi, C., Hyyppä, J., 2015. LiDAR scan matching aided inertial navigation system in GNSS-denied environments. *Sensors (Switzerland)*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., Polosukhin, I., 2017. Attention is all you need. *Advances in Neural Information Processing Systems*.

Velas, M., Spanel, M., Hradis, M., Herout, A., 2018. CNN for IMU assisted odometry estimation using velodyne LiDAR. *18th IEEE International Conference on Autonomous Robot Systems and Competitions, ICARSC 2018*.

Wang, S., Clark, R., Wen, H., Trigoni, N., 2017. DeepVO: Towards end-to-end visual odometry with deep Recurrent Convolutional Neural Networks. *2017 IEEE International Conference on Robotics and Automation (ICRA)*. http://dx.doi.org/10.1109/ICRA.2017.7989236.

Wang, W., Saputra, M. R. U., Zhao, P., Gusmao, P., Yang, B., Chen, C., Markham, A., Trigoni, N., 2019. DeepPCO: End-to-End Point Cloud Odometry through Deep Parallel Neural Network. *IEEE International Conference on Intelligent Robots and Systems*.

Wang, Y., Shi, T., Yun, P., Tai, L., Liu, M., 2018. Pointseg: Real-time semantic segmentation based on 3d lidar point cloud.

Xue, H., Fu, H., Dai, B., 2019. IMU-aided high-frequency lidar odometry for autonomous driving. *Applied Sciences (Switzerland)*.

Yang, Z., Wang, P., Wang, Y., Xu, W., Nevatia, R., 2018. LEGO: Learning Edge with Geometry all at Once by Watching Videos. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*.

Ye, H., Chen, Y., Liu, M., 2019. Tightly coupled 3D Lidar inertial odometry and mapping. *Proceedings - IEEE International Conference on Robotics and Automation*.

Zhang, J., Singh, S., 2014. Loam: Lidar odometry and mapping in real-time.