Journal of Intelligent & Robotic Systems manuscript No.
(will be inserted by the editor)

Convolutional Neural Networks for the Odometry Estimation

Real-time positional information for the mapping with 3D LiDAR

Martin Velas · Michal Spanel · Michal Hradis · Adam Herout

Received: date / Accepted: date

Abstract This article presents a novel method for odometry estimation from 3D data of Velodyne LiDAR scanner using convolutional neural networks. For training and forward evaluation of the proposed networks, the original data is encoded into 2D matrices. In experiments with the KITTI dataset, our networks show significantly higher accuracy in estimation of the translational motion parameters compared to the state of the art LOAM method. In addition, they achieve higher speed and real-time performance. Using data provided by the IMU sensor, it is possible to estimate odometry and align the point cloud with a high precision. The proposed method can replace the odometry estimation from the wheel encoders or supplement the missing GPS data when the GNSS signal is not available (for example, during the interior mapping). In addition, we propose alternate CNNs for the estimation of the rotational motion that achieve results comparable to the state of the art. Our solution delivers real-time performance and accuracy to provide online preview of the mapping and to verify the completeness of the map during the mission.

Keywords Odometry · Velodyne · LiDAR · CNN · KITTI

Mathematics Subject Classification (2000) 65D19 · 68T40 · 68T45

1 Introduction

In recent years, many solutions for indoor and outdoor *3D mapping* have been developed using LiDAR sensors, demonstrating the relevance of the *odometry estimation* and *point cloud registration* problem, as well as the demand for the complex 3D mapping solutions.

This work has been supported by the Artemis Joint Undertaking grant agreement ALMARVI (no. 621439), the TACR Competence Centres project V3C (no. TE01020415), the IT4IXS IT4Innovations Excellence project (LQ1602), and the Ministry of Industry and Trade of the Czech Republic project 4RECON CZ.01.1.02/0.0/0.0/15_013/0004722.

All authors come from:
Brno University of Technology, Bozetechova 1/2, Brno, Czech Republic
Tel.: +420-54114-1144
Fax: +420-54114-1270
E-mail: {ivelas|spanel|ihradis|herout} @ fit.vutbr.cz

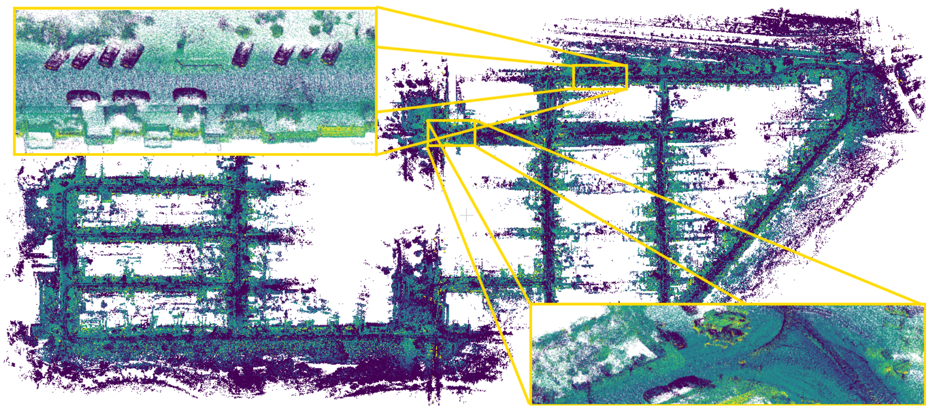


Fig. 1: Example of the LiDAR point clouds registered by CNN-estimated translation with rotations provided by an IMU sensor. The map was built using the sequence 08 of KITTI odometry dataset [8].

Leica[1] company introduced a Pegasus backpack that includes a pair of Velodyne scanners along with several wide-angle cameras and IMU/GNSS positional sensors to support the mapping process. Geoslam[2] in its hand-held mapping products ZEB and ZEB-REVO uses a simple LiDAR rangefinder along with an inertial unit. Other companies like LiDARUSA [3] and RIEGL[4] are building their LiDAR systems primarily for the ground and airborne outdoor mapping. Besides the commercial solutions listed above, multiple scientific non-commercial backpack mapping systems exist [10,11,13] providing betters insights into the practical issues and solutions to these problems. All such systems require measurements from IMU and GNSS positioning sensors for the alignment of the point clouds. This requirement limits the deployment of these solutions when the GNSS signal is not available.

In the solutions listed above, the building of precise 3D maps from recorded data is performed off-line as a post-processing step. During the process of data acquisition, the operator is not able to verify whether the entire mapped environment (building, park, forest, ...) is correctly captured without any part missing. This is a significant disadvantage because the repetition of the measurement process can be expensive and time-consuming. The LiDAR data alignment requires the estimation of motion parameters (both orientation and position) directly during the mission. Although the orientation can be estimated online in a relatively robust way using an inertial unit, *exact positional information* requires reliable GNSS signal measurements including the online corrections of position such as Differential GPS (DGPS), Real Time Kinematics (RTK), etc. [7,12]. Since these requirements are not met in many scenarios (interior scenery, forests, tunnels, mining sites, etc.), less precise methods are used as a common practice. For example, estimation of the odometry by the wheel encoders of the mobile platform is used, but this can not be done with hand-held or backpack-mounted devices.

In this work, we propose an alternative solution using the *convolutional neural networks (CNN)* – a frame to frame *odometry estimation* based on the Velodyne LiDAR data. Similar

[1] `http://leica-geosystems.com`
[2] `https://geoslam.com`
[3] `https://www.lidarusa.com`
[4] `http://www.riegl.com`

deployments of the convolutional neural networks have already proven to be successful when dealing with this type of the sparse 3D data. Particularly in the tasks such as ground segmentation [22] or, for example, in challenging vehicle detection [14].

The main contribution of our work is the fast and accurate estimation of positioning parameters (translation) in real-time, that outperforms state of the art. We also propose alternative networks for full estimation of all 6DoF (6 Degrees of Freedom) motion parameters including rotation. The results of these networks are also comparable to the state of the art. Our deployment of convolutional neural networks for the odometry estimation, together with the existing methods of the vehicle detection [14] and ground segmentation [22] also illustrates the general usability of CNNs for this type of sparse LiDAR data.

This article is an extended version of a conference paper [21] published in ICARSC 2018. Besides the additional description of the related work in Section 2, details and further insights into our methodology (Section 3), this paper delivers new experiments and results providing deeper understanding of the pros and cons of our approach in Section 4. Both these and also the original results are further elaborated in the additional discussion section.

2 Related Work

The methods for visual odometry estimation, which have been published so far, can be divided into two basic groups. The first one consists of *direct methods* that calculate the motion parameters in one step. These parameters can be estimated from a pair of camera images, from depth RGB-D frames, or from 3D data, for example in the form of a cloud of points. In comparison with the second group – *iterative methods* – direct methods have the potential for better time performance. Unfortunately, to the best of our knowledge, no direct method of point cloud registration or odometry estimation from the LiDAR scans has been published or developed yet.

When the indoor environment is mapped using the backpack solution, neither the GNSS positional system is available nor the wheel odometry estimation is feasible. This problem can be overcome by the additional tracking hardware, such as Nikon iSpace [13] providing the precise position in real time. This system requires placing laser transmitters all over the mapped environment and including a compatible receiver into the backpack platform. Using the triangulation, the position within the environment covered by the transmitters is estimated with an error below $\pm 0.25\,\mathrm{mm}$. These data can be used when the exact position is required, but also as the source of ground truth when a new SLAM solution is developed. The most significant drawback is the time-demanding process of spreading the transmitters all over the scene and the calibration of the system prior to the mapping process.

Since the introduction of the well-known Iterative Closest Points (ICP) algorithm [1,5], many modifications of this approach were developed. In all derivatives, two basic steps are iteratively repeated until the termination conditions are met: matching the elements between two point clouds (originally the points were used directly) and the estimation of target frame transformation, minimizing the error represented by the distance of matching elements. This approach assumes that there actually exist matching elements in the target cloud for a significant amount of elements in the source point cloud. However, such assumption does not often hold for sparse LiDAR data and it causes significant inaccuracies.

Grant [9] used planes detected in Velodyne LiDAR data as the basic elements. The planes are identified by an analysis of depth gradients within readings from a single laser beam and then by accumulating them in a modified Hough space. The detected planes are matched and the optimal transformation is found by using a previously published method [17].

Their evaluation shows the significant error ($\approx 1\,\mathrm{m}$ after a $15\,\mathrm{m}$ run) when mapping indoor office environment. Douillard et al. [6] used the ground removal and clustering of the remaining points into segments. The transformation estimated from matching the segments is only approximate and it is probably compromised by using quite coarse ($20\,\mathrm{cm}$) voxel grid.

Generalized ICP (GICP) [19] replaces the standard point-to-point matching by the plane-to-plane strategy. Small local surfaces are estimated and their covariance matrices are used for their matching. When using Velodyne LiDAR data, the authors achieved $\pm 20\,\mathrm{cm}$ accuracy when registering pairwise scans. In our evaluation [20] using the KITTI dataset [8], the method yields average error of $11.5\,\mathrm{cm}$ in the frame-to-frame registration task. The robustness of GICP drops in the case of a large distance between the scans ($> 6\,\mathrm{m}$). This was improved by employing visual SIFT features extracted from omnidirectional Ladybug camera [15] and the codebook quantization of extracted features for building a sparse histogram and for maximization of the mutual information [16].

Bose and Zlot [3] are able to build consistent 3D maps of various environments, including challenging natural scenes, deploying visual loop closure over the odometry provided by inaccurate wheel encoders and the orientation by IMU. Their robust place recognition is based on Gestalt keypoint detection and description [2]. Deployment of our CNN in such a system would overcome the requirement of the wheel platform and the same approach would be useful for human-carried sensory gears (Pegasus, ZEB, etc.) as mentioned in the introduction.

The extension of the simple 2D rangefinder into the 3D LiDAR SLAM solution used in ZEB and ZEB-REVO by the Geoslam company was originally proposed in 2012 as a Zebedee scanner [4]. It is achieved by placing the rangefinder atop of the flexible spring amplifying external motion introduced by the person carrying the whole system. The initial rough estimation of both the odometry and the orientation of the spring-mounted LiDAR head is done by the IMU sensor. Then, the surface patches (surfels) are estimated within the cells of the voxel grid. The surfels are matched and the corrections are computed by minimizing the distance between the matches, by using the difference between the estimation and the original IMU readings, and by constraints enforcing the continuity with the previous trajectory.

Our previous work [20] proposed sampling the 3D LiDAR point clouds by *Collar Line Segments (CLS)* to overcome data sparsity. First, the original Velodyne point cloud is split into polar bins. The line segments are randomly generated within each bin, matched by nearest neighbor search and the resulting transformation fits the matched lines into common planes. The CLS approach was also evaluated using the KITTI dataset and achieves $7\,\mathrm{cm}$ error of the pairwise scan registration. Splitting into polar bins is also used in this work for encoding the 3D data to 2D representation (see Section 3.1).

The top ranks in KITTI Visual odometry benchmark [8] are for last years occupied by LiDAR Odometry and Mapping (LOAM) [24] and Visual LOAM (V-LOAM) [25] methods. Planar and edge points are detected and used to estimate the optimal transformation in two stages: fast scan-to-scan and precise scan-to-map. The map consists of keypoints found in the previous LiDAR point clouds. Scan-to-scan registration enables real-time performance and only each n-th frame is actually registered within the map.

The implementation of LOAM was publicly released under BSD license but later withdrawn after being commercialized. The original code is accessible through the documentation[5] and we used it for evaluation and comparison with our proposed solution. In our experiments, we were able to achieve superior accuracy in the estimation of the translation

[5] `http://docs.ros.org/indigo/api/loam_velodyne/html/files.html`

parameters and comparable results in the estimation of full 6DoF (degrees of freedom) motion parameters including rotation. In V-LOAM [25], the original method was improved by fusion with RGB data from omnidirectional camera and the authors also prepared a method which fuses LiDAR and RGB-D data [23].

The encoding of 3D LiDAR data into the 2D representation, which can be processed by convolutional neural network (CNN), were previously proposed and used in the task of ground segmentation [22] and in vehicle detection [14]. We use a similar CNN approach for quite a different task of visual odometry estimation. Besides the precision and the real-time performance, our method also contributes as the illustration of general usability of CNNs for sparse LiDAR data. The key difference is the amount and the ordering of input data processed by the neural network (described in the next chapter and in Figure 5). While the previous methods [14,22] process only a single frame, in order to estimate the transformation parameters precisely we process multiple frames simultaneously.

3 Proposed Methodology

Our *goal* is the estimation of transformation $\boldsymbol{T_n} = [t_n^x, t_n^y, t_n^z, r_n^x, r_n^y, r_n^z]$ representing the 6DoF motion of a platform carrying LiDAR sensor, given the current LiDAR frame $\boldsymbol{P_n}$ and N previous frames $\boldsymbol{P_{n-1}}, \boldsymbol{P_{n-2}}, \ldots, \boldsymbol{P_{n-N}}$ in the form of point clouds. This can be written as a mapping Θ from the point cloud domain $\mathbb{P}$ to the domain of motion parameters (1) and (2).

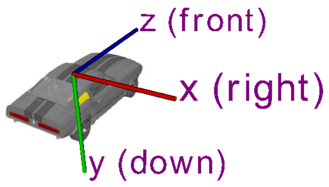
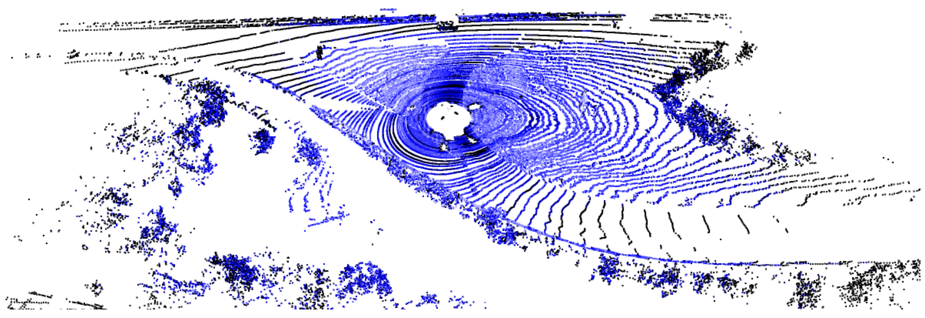


Fig. 2: The coordinate system used in this work.

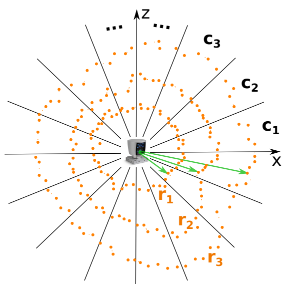
Each element of the point cloud $\boldsymbol{p} \in \boldsymbol{P}$ is the vector $\boldsymbol{p} = [p^x, p^y, p^z, p^r, p^i]$, where $[p^x, p^y, p^z]$ are its coordinates in the 3D space originating at the sensor position as shown in the Figure 2. The coordinates in horizontal XZ plane determines the depth, Y coordinate is equivalent to the height above ground. p^r is the index of the laser beam that captured this point, which is commonly referred as the “ring” index since the Velodyne data resembles the rings of points shown in Figure 3 (top, left). The laser intensity reading (which belongs to this particular point) is denoted as p^i.

$$\boldsymbol{T_n} = \Theta(\boldsymbol{P_n}, \boldsymbol{P_{n-1}}, \boldsymbol{P_{n-2}}, \ldots, \boldsymbol{P_{n-N}}) \tag{1}$$

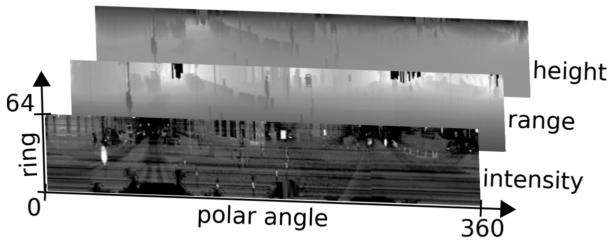
$$\Theta : \mathbb{P}^{N+1} \to \mathbb{R}^6 \tag{2}$$



(a)



(b)



(c)

Fig. 3: Transformation of the sparse Velodyne point (a) cloud into the multi-channel dense matrix (c). Each row represents measurements of a single laser beam (single ring $r_1, r_2, r_3, \ldots$) done during one rotation of the sensor. Each column contains measurements of all 64 laser beams captured within the specific rotational angle interval (polar cone $c_1, c_2, c_3, \ldots$).

3.1 Data Encoding

We represent the mapping Θ by a convolutional neural network. Since we use sparse 3D point clouds and convolutional neural networks are commonly designed for dense 1D and 2D data, we adopt a previously proposed [14, 22] *encoding* $\mathcal{E}$ (3) of 3D LiDAR data to a dense matrix $\boldsymbol{M} \in \mathbb{M}$. These encoded data are used for actual training of the neural network by implementing the mapping $\tilde{\Theta}$ (4, 5).

$$\boldsymbol{M} = \mathcal{E}(\boldsymbol{P}); \quad \mathcal{E} : \mathbb{P} \to \mathbb{M} \tag{3}$$

$$\boldsymbol{T_n} = \tilde{\Theta}(\mathcal{E}(\boldsymbol{P_n}), \mathcal{E}(\boldsymbol{P_{n-1}}), \ldots, \mathcal{E}(\boldsymbol{P_{n-N}})) \tag{4}$$

$$\tilde{\Theta} : \mathbb{M}^{N+1} \to \mathbb{R}^6 \tag{5}$$

Each element $\boldsymbol{m_{r,c}}$ of the matrix $\boldsymbol{M}$ encodes points of *the polar bin* $\boldsymbol{b_{r,c}} \subset \boldsymbol{P}$ (6) as a vector of 3 values: depth, the vertical height relative to the sensor position, and the intensity of laser return (7). Since multiple points fall into the same bin (angular resolution of the 2D representation is smaller than angular resolution of the LiDAR), the representative values are computed by averaging. On the other hand, if a polar bin is empty, the missing element of the resulting matrix is interpolated from its neighborhood using linear interpolation.

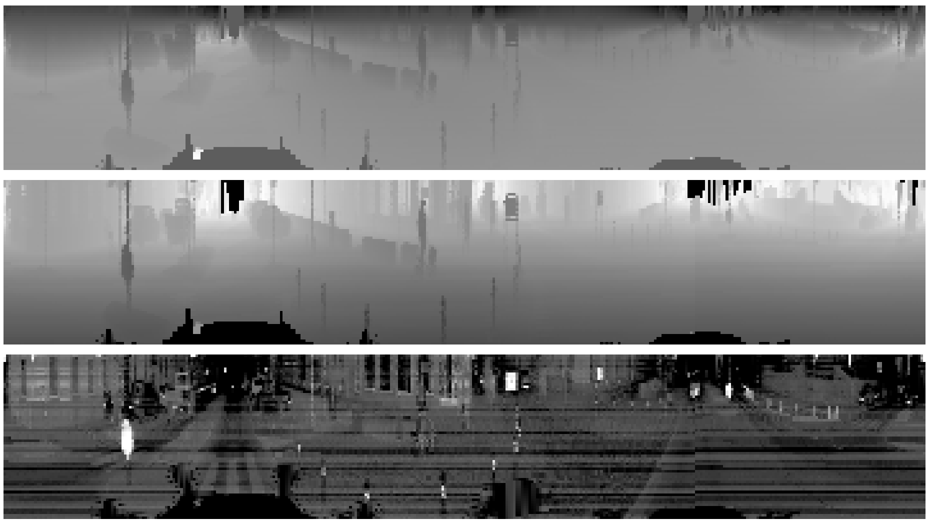


Fig. 4: Detail of the channels in the 2D representation: height (top), range (middle) and intensity (bottom)

$$\boldsymbol{m_{r,c}} = \varepsilon(\boldsymbol{b_{r,c}}); \quad \varepsilon : \mathbb{P} \to \mathbb{R}^3 \tag{6}$$

$$\varepsilon(\boldsymbol{b_{r,c}}) = \frac{\sum\limits_{\boldsymbol{p} \in \boldsymbol{b_{r,c}}} \left[p^y, \|p^x, p^z\|_2, p^i\right]}{|\boldsymbol{b_{r,c}}|} \tag{7}$$

Indices r, c denote both the row (r) and the column (c) of the encoded matrix and the ring index (r) the polar cone (c) in the original point cloud (see Figure 3). Dividing the point cloud into the polar bins follows the same strategy as described in our previous work [20]. Each polar bin is identified by the polar cone $\varphi(.)$ and the ring index p^r.

$$\boldsymbol{b_{r,c}} = \{\boldsymbol{p} \in \boldsymbol{P} \,|\, p^r = r \wedge \varphi(\boldsymbol{p}) = c\} \tag{8}$$

$$\varphi(\boldsymbol{p}) = \left\lfloor \frac{\text{atan}\left(\frac{p^z}{p^x}\right) + 180°}{\frac{360°}{R}} \right\rfloor \tag{9}$$

where R is the horizontal angular resolution of the polar cones. In our experiments we used the resolution $R = 1°$ (and $0.2°$ in the classification formulation described below).

The detailed view on the example of encoded data can be found in the Figure 4. Although the range and the height channels are quite smooth, the information regarding even the small elements in the scene (road delineators, bus stop, etc.) is still preserved. On the other hand, the intensity readings are quite noisy but this channel preserves the visual information (e.g. the zebra crossing on the left part of the image).

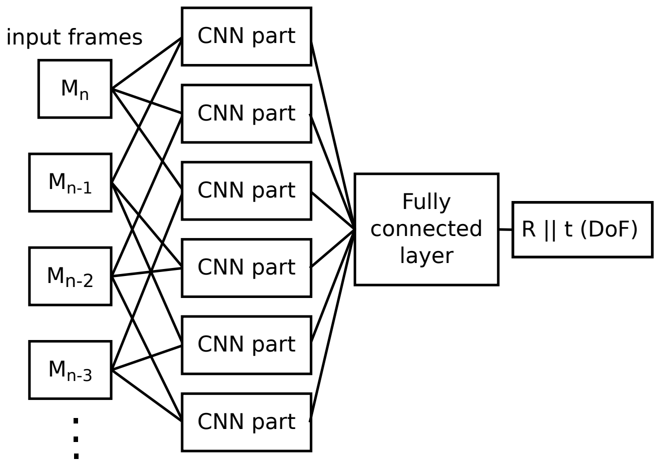


Fig. 5: Topology of the network implementing $\tilde{\Theta}_{\boldsymbol{R}}$ and $\tilde{\Theta}_{\boldsymbol{t}}$. All combinations of current $\boldsymbol{M_n}$ and previous $\boldsymbol{M_{n-1}}, \boldsymbol{M_{n-2}}, \ldots$ frames (3 previous frames in this example) are pairwise processed by the same CNN part (see structure in Figure 6) with shared weights. The final estimation of rotation or translation parameters is done in the fully connected layer joining the outputs of the CNN parts. For training, the euclidean loss was used.

3.2 From Regression to Classification

In our preliminary experiments, we trained the network $\tilde{\Theta}$ estimating full 6DoF motion parameters. Unfortunately, such network provided very inaccurate results. The output parameters consist of two different motion modalities – rotation and translation $\boldsymbol{T_n} = [\boldsymbol{R_n}|\boldsymbol{t_n}]$ – and it is difficult to determine (or weight) the importance of angular and positional differences in backward propagation. So we decided to split the mapping into the estimation of rotation parameters $\tilde{\Theta}_{\boldsymbol{R}}$ (10) and translation $\tilde{\Theta}_{\boldsymbol{t}}$ (11).

$$\boldsymbol{R_n} = \tilde{\Theta}_{\boldsymbol{R}}(\boldsymbol{M_n}, \boldsymbol{M_{n-1}}, \ldots, \boldsymbol{M_{n-N}}) \tag{10}$$

$$\boldsymbol{t_n} = \tilde{\Theta}_{\boldsymbol{t}}(\boldsymbol{M_n}, \boldsymbol{M_{n-1}}, \ldots, \boldsymbol{M_{n-N}}) \tag{11}$$

$$\tilde{\Theta}_{\boldsymbol{R}} : \mathbb{M}^{N+1} \to \mathbb{R}^3; \quad \tilde{\Theta}_{\boldsymbol{t}} : \mathbb{M}^{N+1} \to \mathbb{R}^3 \tag{12}$$

The implementation of $\tilde{\Theta}_{\boldsymbol{R}}$ and $\tilde{\Theta}_{\boldsymbol{t}}$ by convolutional neural network is shown in Figure 5. We use *multiple input frames* in order to improve stability and robustness of the method. Such a multi-frame approach was also successfully used in our previous work [20] and comes from the assumption that motion parameters are similar within a small time window ($0.1 - 0.7$ s in our experiments below).

The idea behind the proposed topology is the expectation that shared CNN components for pairwise frame processing will estimate the motion map across the input frame space (analogous to the optical flow in image processing). The final estimation of rotation or translation parameters is performed in the fully connected layer joining the outputs of purely convolutional components.

Our experiments presented in the Table 1 prove the benefits of such multi-view approach in comparison with the approach of processing only the current and the single previous

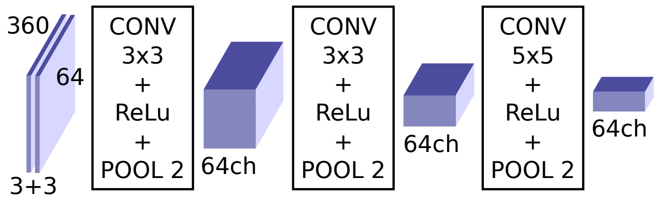


Fig. 6: Topology of the shared CNN block denoted as *"CNN part"* in Figure 5. Each block processes a pair of encoded LiDAR frames. The topology is quite shallow with small convolutional kernels, ReLu nonlinearities and max polling after each convolutional layer. The output blob size is $45 \times 8 \times 64$ ($W \times H \times Ch$).

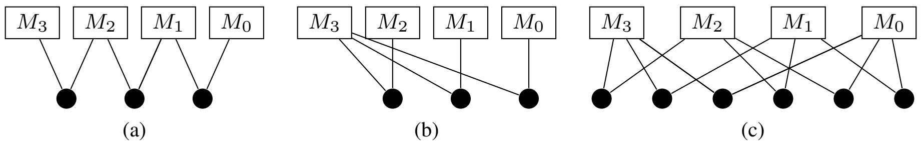


Fig. 7: Different strategies for the multi-view approach. In this example, current M_3 and three previous frames M_2, M_1, M_0 are used as an input into the network. Each pair joined by the node (black dot) is processed separately by the part of the convolutional network. In the *neighborly* strategy (a), only the following frames are processed together. On the other hand, all previous frames can be compared only *with the current frame* (b) or *all permutations* (c) can be processed.

frame. We have also experimented with the different strategies for arranging the frames on the input of our networks. We have tried all arrangements presented in Figure 7 and the best odometry estimation was provided when *all permutations* (Figure 7c) were processed by the convolutional part (as also shown in the previous Figure 5).

Comparing with the other two strategies, when only neighboring frames (Figure 7a) were processed together or only the current frame was processed with the previous frames (Figure 7b), the strategy with all the permutations (Figure 7c) is able to get a wider context of the current situation. The idea behind is that the transformation of the current frame with respect to the previous one would be internally estimated multiple times to make the final decision (done by the fully connected layer) more precise. Following the example from the Figure 7, the goal is the estimation of the transformation from frame M_2 to M_3 which we will denote as a $T_{2\to3}$. Ideally, each CNN part would estimate the guess $\hat{T}$ of transformation between its input frames (e.g. $\hat{T}_{0\to2}$ between frames M_0 and M_2). Different combinations (13) of these guesses provide multiple pieces of evidence regarding the desired output transformation.

$$T_{2\to3} \approx \hat{T}_{2\to3} \approx \hat{T}^{-1}_{1\to2}\cdot\hat{T}_{1\to3} \approx \hat{T}^{-1}_{0\to2}\cdot\hat{T}_{0\to3} \approx \hat{T}^{-1}_{0\to1}\cdot\hat{T}^{-1}_{1\to2}\cdot\hat{T}_{0\to3} \approx \dots \quad (13)$$

Splitting the task of odometry estimation into two separated networks and sharing the same topology and input data significantly improved the results – especially the precision of the estimated translation parameters. However, the precision of the predicted rotation was still insufficient. The rotation is represented by Euler angles, but the experiments with quaternions and axis-angles were also performed with no improvement. The original formulations of our goal (1) can be considered as solving the *regression task*. However, the space

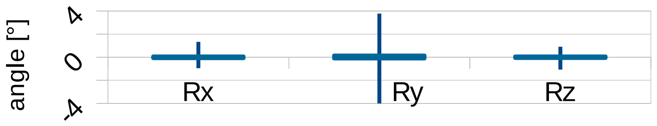


Fig. 8: Minimum and maximum rotations around x, y, z axis as observed in the training data sequences of the KITTI dataset.

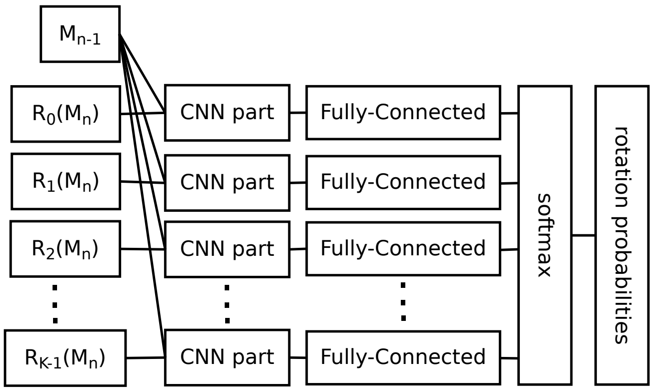


Fig. 9: Modification of the original topology (Figure 5) for precise estimation of rotation parameters. Rotation parameter space (each axis separately) is densely sampled into K rotations $R_0, R_1, \ldots, R_{K-1}$ and applied to the current frame $\boldsymbol{M_n}$. Both the CNN part and the fully connected layer are trained as comparators Γ with previous frame $\boldsymbol{M_{n-1}}$ estimating the probability of the given rotation R_i. All the CNN blocks (structure can be found in Figure 10) and fully connected layers share the weights of the activations.

of possible rotations between consequent frames is quite small for reasonable data (distribution of rotations for KITTI dataset can be found in Figure 8). Such small space can be densely sampled and we can reformulate this problem to the *classification task* (14, 15).

$$R = \underset{i \in \{0,\ldots,K-1\}}{\arg\max}\ \Gamma(R_i(\boldsymbol{M_n}), \boldsymbol{M_{n-1}}) \tag{14}$$

$$\Gamma : \mathbb{M}^2 \to \mathbb{R} \tag{15}$$

where $R_i(\boldsymbol{M_n})$ represents rotation R_i of the current LiDAR frame $\boldsymbol{M_n}$ and $\Gamma(.)$ estimates the probability of R_i to be the correct rotation between the frames $\boldsymbol{M_n}$ and $\boldsymbol{M_{n-1}}$.

A similar approach was previously used in the task of human age estimation [18]. Instead of training the CNN to estimate the age directly, a person image is classified to be $0, 1, \ldots, 100$ years old.

The implementation of Γ comparator by a convolutional neural network can be found in Figure 9. In the following sections, this network will be referred to as the *classification CNN* while the original one will be referred to as the *regression CNN*. We have also experimented with the classification-like formulation of the problem using the original CNN topology (Figure 5) without sampling and applying the rotations, but this did not bring the improvement.

For the classification network we experienced better results when wider input (horizontal resolution $R = 0.2°$) is provided to the network. This affected also the properties

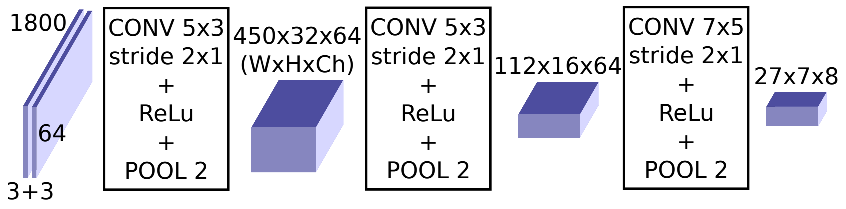


Fig. 10: Modification of the CNN block of the classification network. Wider input (angular resolution $R = 0.2°$) and wider convolution kernels with horizontal stride 2 are used.

of the convolutional component (the CNN part), where wider convolution kernels are used with horizontal stride (see Figure 10) to reduce the amount of data processed by the fully connected layer.

Although the space of the observed rotations is quite small (approximately $\pm 1°$ around x and z axis, and $\pm 4°$ for y axis, see Figure 8), sampling densely (by a fraction of degree) this subspace of 3D rotations would result in thousands of possible rotations. Because such amount of rotations would be infeasible to process, we decided to estimate the rotation around each axis separately, so we trained 3 CNNs for rotations around x, y and z axis separately. These networks share the same topology (Figure 9).

In the formulation of our classification problem (14), the final decision of the best rotation R^* is done by max polling. Since Γ estimates the probability of a rotation angle $p(R_i)$ (16), assuming the normal distribution we can also compute the maximum likelihood solution by the weighted average (17).

$$p(R_i) = \Gamma(R_i(\boldsymbol{M_n}), \boldsymbol{M_{n-1}}) \tag{16}$$

$$R^* = \frac{\sum_{i \in \boldsymbol{S_W}} p(R_i).R_i}{\sum_{i \in \boldsymbol{S_W}} p(R_i)} \tag{17}$$

$$\boldsymbol{S_W} = \underset{\boldsymbol{S}=\{i_0,\ldots,i_0+W\}}{\arg\max} \sum_{i \in \boldsymbol{S}} p(R_i) \tag{18}$$

Moreover, this estimation can done for a window of fixed size W which is limited only for the highest rotation probabilities (18). When the size of the window is set to 1, the result is the same as for the max polling (picking the rotation of the highest probability).

3.3 Data Processing

We used the encoded Velodyne LiDAR data to train and test the proposed networks. As previously mentioned, the original raw point cloud elements consist of the coordinates x, y and z, the identification of laser beam that captured the given point and the received laser intensity value. Encoding to the 2D representation transforms coordinates of x and z (in the horizontal plane) into the depth information (represented by the range channel) and the horizontal angle that is analogous to the column index in the encoded matrix. The measured intensities and the y coordinates are mapped directly into the matrix channels, and the laser beam index is represented by the line index in the 2D matrix. This means that our encoding does not cause any loss of information (except for the aggregation of multiple points into the same polar bin).

Furthermore, we use the same data normalization and rescaling as we used in our previous work [22]:

$$\overline{h} = \frac{y^i}{H} \tag{19}$$

$$\overline{d} = \log(d) \tag{20}$$

This applies only to the vertical height h and depth d, since the intensity values are already normalized to interval $(0; 1)$. We set the height normalization constant to $H = 3$, since in the usual scenarios, the Velodyne (model HDL-64E) captures vertical slice approximately $3\,\mathrm{m}$ high.

In our preliminary experiments, we trained the convolutional neural networks without this normalization (19) and rescaling (20) and we also experimented with using the 3D point coordinates as the channels of the CNN input matrices. All these approaches resulted only in worse odometry precision.

4 Experimental Results

We implemented the proposed networks using *Caffe*[6] deep learning framework. For training and testing, data from the KITTI odometry benchmark[7] were used together with provided development kit for the evaluation and error estimation. The LiDAR data were collected by Velodyne HDL-64E sensor mounted on top of a vehicle together with an IMU sensor and a GPS localization unit with RTK correction signal providing precise position and orientation measurements [8]. Velodyne spins with the frequency of $10\,\mathrm{Hz}$ providing 10 LiDAR scans per second. The dataset consist of 11 data sequences where the ground truth is provided. We split these data into a training (sequences 00-07) and a testing set (sequences 08-10). For the rest of the dataset (sequences 11-21) there is no ground truth and they serve only for the KITTI benchmarking purposes. Therefore these sequences are not useful for us.

The error of the estimated odometry is evaluated by the development kit provided with the KITTI benchmark. The data sequences are split into subsequences of $100, 200, \dots, 800$ frames ($10, 20, \dots, 80$ seconds duration). The error e_s of each subsequence is computed as:

$$e_s = \frac{\|\boldsymbol{E_s}, \boldsymbol{C_s}\|_2}{l_s} \tag{21}$$

where $\boldsymbol{E_s}$ is the expected position (from the ground truth data) and $\boldsymbol{C_s}$ is the estimated position of the LiDAR where the last frame of subsequence was taken with respect to the initial position (within the given subsequence). The difference is divided by the length l_s of the followed trajectory. The final error value is the average of errors e_s across all the subsequences of all the lengths.

First, we trained and evaluated the regression networks (their topology is described in Figure 5) for direct estimation of rotation or translation parameters. The results can be found in Table 1. To determine the error of the network predicting translation or rotation motion parameters, the missing rotation or translation parameters respectively were taken from the ground truth data since the evaluation requires all the 6DoF parameters.

The evaluation shows that the proposed CNNs predict the translation (*CNN-t* in Table 1) with a high precision – the best results were achieved for the network taking the current and

[6] `caffe.berkeleyvision.org`

[7] `www.cvlibs.net/datasets/kitti/eval_odometry.php`

	Error			**Forward time** [s/frame]	
N	**CNN-t**	**CNN-R**	**CNN-Rt**	**GPU**	**CPU**
1	0.0184	0.3794	0.3827	0.004	0.065
2	0.0129	0.2752	0.2764	0.013	0.194
3	0.0111	0.2615	0.2617	0.026	0.393
5	**0.0102**	0.2646	0.2656	0.067	0.987
7	0.0130	0.2534	0.2546	0.125	1.873

Table 1: Evaluation of the regression networks for different size of the input data – N is the number of previous frames. The convolutional neural networks were used to determine the translation parameters only (column CNN-t), the rotation only (CNN-R) and both the rotation and translation (CNN-Rt) parameters for KITTI sequences 00–08. The error of the estimated odometry together with the processing time per a single frame (using CPU only or GPU acceleration) is presented.

Window size W	**Odom. error**	**Window size**	**Odom. error**
1 (max polling)	0.03573	9	0.03704
3	**0.03433**	11	0.03712
5	0.03504	13	0.03719
7	0.03629	all	0.03719

Table 2: The impact of the window size on the error of odometry estimation, when the rotation parameters are estimated by the classification strategy. Window size $W = 1$ is equivalent to the max pooling, maximal likelihood solution is found also when "*all*" probabilities are taken into the account without the window restriction.

$N = 5$ previous frames as the input. The results also show that all these networks outperform LOAM (error 0.0186, see evaluation in Table 3 for more details) in the estimation of translation parameters. On the contrary, this method is unable to estimate rotations (*CNN-R* and *CNN-Rt*) with a sufficient precision. All networks except for the largest one ($N < 7$) are capable of real-time performance with GPU support (GeForce GTX 770 used) and the smallest one also without any acceleration (running on i5-6500 CPU). It should be noted that the Velodyne standard frame-rate is 10 fps.

We also wanted to explore whether the CNNs are capable of predicting the full 6DoF motion parameters, including the rotation angles with a sufficient precision. Hence the classification network schema shown in Figure 9 was implemented and trained also using the Caffe framework. The network predicts the probabilities for densely sampled rotation angles. We used sampling resolution of $0.2°$, which is equivalent to the horizontal angular resolution of Velodyne data in the KITTI dataset. Given the statistics from the training data shown in Figure 8, we sampled the interval $\pm1.3°$ of rotations around x and z axis into 13 classes, and the interval $\pm5.6°$ into 56 classes, including approximately 30 % tolerance.

Since the network predicts the probabilities of given rotations, the final estimation of the rotation angle is obtained by max polling (14) or by the window approach of maximum likelihood estimation (17, 18). Table 2 shows that optimal results are achieved when the window size $W = 3$ is used.

	Translation only			**Rotation and translation**			
Seq. #	**LOAM -full**	**LOAM -online**	**CNN -regression**	**LOAM -full**	**LOAM -online**	**CNN -regression**	**CNN -classif.**
00	0.0152	0.0193	0.0084	0.0225	0.0516	0.2877	0.0302
01	0.0368	0.0255	0.0079	0.0396	0.0385	0.1492	0.0444
02	0.0383	0.0293	0.0076	0.0461	0.0550	0.2290	0.0342
03	0.0120	0.0117	0.0166	0.0191	0.0294	0.0648	0.0494
04	0.0076	0.0085	0.0089	0.0148	0.0150	0.0757	0.0177
05	0.0092	0.0096	0.0056	0.0184	0.0246	0.1357	0.0235
06	0.0088	0.0130	0.0036	0.0160	0.0335	0.0812	0.0188
07	0.0137	0.0155	0.0077	0.0192	0.0380	0.1308	0.0177
Train avg.	0.0214	0.0197	0.0077	0.0287	0.0433	0.1970	0.0303
08	0.0107	0.0145	0.0096	0.0239	0.0349	0.2716	0.0289
09	0.0368	0.0380	0.0098	0.0322	0.0430	0.2373	0.0494
10	0.0213	0.0196	0.0128	0.0295	0.0399	0.2823	0.0327
Test avg.	0.0186	0.0208	**0.0102**	**0.0268**	0.0376	0.2655	0.0343

Table 3: Comparison of the odometry estimation precision by the proposed method and LOAM for sequences of the KITTI dataset [8] (sequences $00 - 07$ were used for training the CNN, $08 - 10$ for testing only). LOAM was tested in the on-line mode (LOAM-online) when the time spent for a single frame processing is limited to Velodyne fps (0.1s/frame) and in the full mode (LOAM-full) where each frame is fully registered within the map. Both the regression (CNN-regression) and the classification (CNN-classification) strategies of our method are included. When only translation parameters are estimated, our method outperforms LOAM. On the contrary, LOAM outperforms our CNN odometry when full 6DoF motion parameters are estimated.

We compared our CNN approach for odometry estimation with the LOAM method [24]. We used the originally published ROS implementation (see link in Section 2) with a slight modification to enable KITTI Velodyne HDL-64E data processing. In the original package, the input data format of Velodyne VLP-16 is “hardcoded”. The results of this implementation are labeled as *LOAM-online* in Table 3, since the data are processed online in real time (10 fps). This real-time performance is achieved by skipping the full mapping procedure (registration of the current frame against the internal map) for particular input frames.

Comparing with this original online mode of the LOAM method, our CNN approach achieves better results in estimation of both translation and rotation motion parameters. However, it is important to mention that our classification network for the orientation estimation requires 0.27 s per frame when using GPU acceleration.

The portion of skipped frames in the LOAM method depends on the input frame rate, size of the input data, available computational power and it affects the precision of the estimated odometry. In our experiments with the KITTI dataset (on the same machine as was used for the CNN experiments), $31.7\,\%$ of input frames are processed by the full mapping procedure.

In order to determine the full potential of the LOAM method and for fair comparison, we made further modifications of the original implementation, so that the mapping procedure runs for each input frame. The results of this method are labeled as *LOAM-full* in Table 3 and, in estimation of all 6DoF motion parameters, it outperforms our proposed CNNs. However, the prediction of translation parameters by our regression networks is still significantly more precise and faster and the average processing time of a single frame by the LOAM-full method is $0.7\,\mathrm{s}$.

The visualization of the estimated transformations can be found in the Figure 11. When only the translation (t) parameters are estimated, both methods achieve very good precision and the estimation is visually almost identical with the ground truth. When also the rotation parameters are estimated, better performance of the LOAM can be observed. The most significant error of our method can be observed for the sequence no. 08. The failing cases will be elaborated in more detail at the end of this section.

We also provide the results for the training sequences in the Figure 12 and also in the Table 3. The CNNs perform better on these data as expected. It is also important to mention that this difference is not as dramatic and therefore it is possible to say that no significant overfitting happened during the training process. There is also an interesting observation that the LOAM method is quite failing on the sequence no. 02. This sequence is distinct because of the presence of natural objects (many trees, grass, bushes). This can be probably considered a very inconvenient environment for a method similar to LOAM, since LOAM uses the features typical for flat surfaces and the corners of the “Manhattan world”.

In the next experiment, we wanted to discover which factor or element causes worse behavior of our approach when compared to LOAM, in terms of the rotation angles estimation. In the Figures 13 and 14, we present dependency of the rotation error on the driving speed and the magnitude of the rotation itself. These evaluations are made for the total rotation and also for separate Euler angles. This experiment shows that this dependency is similar for both methods. For higher driving speeds or larger rotations, the error of rotation raises.

It is possible to observe that the error of the cumulated rotation is lower for our CNN approach than for LOAM. However, the total odometry error, presented in the experiment above, is higher for our approach. This is caused by the larger error in heading orientation (yaw Euler angle). Since the car platform is moving predominantly straight forward, the precision of heading affects the total error more significantly. Another interesting conclusion from the Figure 14 is that the LOAM method has problems with estimating roll for the higher rotation magnitudes.

Since there are RGB images also available, we can visualize situations where the error of the estimated odometry is the highest. Such situations are presented in the Figure 15 for LOAM and in the Figure 16 for our CNNs. LOAM suffers from the biggest error when the vehicle goes along the long smooth curves or when it passes a sharp, tilted turn. An error occurred also after a sharp turn when the second vehicle appeared in the opposite direction. Our CNN fails in situations when there is a significant change of direction – 180° turn or a sudden change in the direction of the vehicle to avoid manhole covers. The convolutional neural networks also fail when the vehicle stops at a crossing and the second car or truck passes in front of the vehicle.

Both the results of our method and the outputs of the LOAM approach yield a very large difference from the KITTI Odometry ground truth for the beginning of the sequence no. 08. This situation is visualized and described in the Figure 17 and it is obvious that this error is caused by the imprecision of the ground truth annotations rather than imprecisions of estimated odometry. It is possible that the navigation subsystem (GNSS, inertial, odometer sensors) is not properly initialized yet and therefore an incorrectly large slope in the positional data appears.

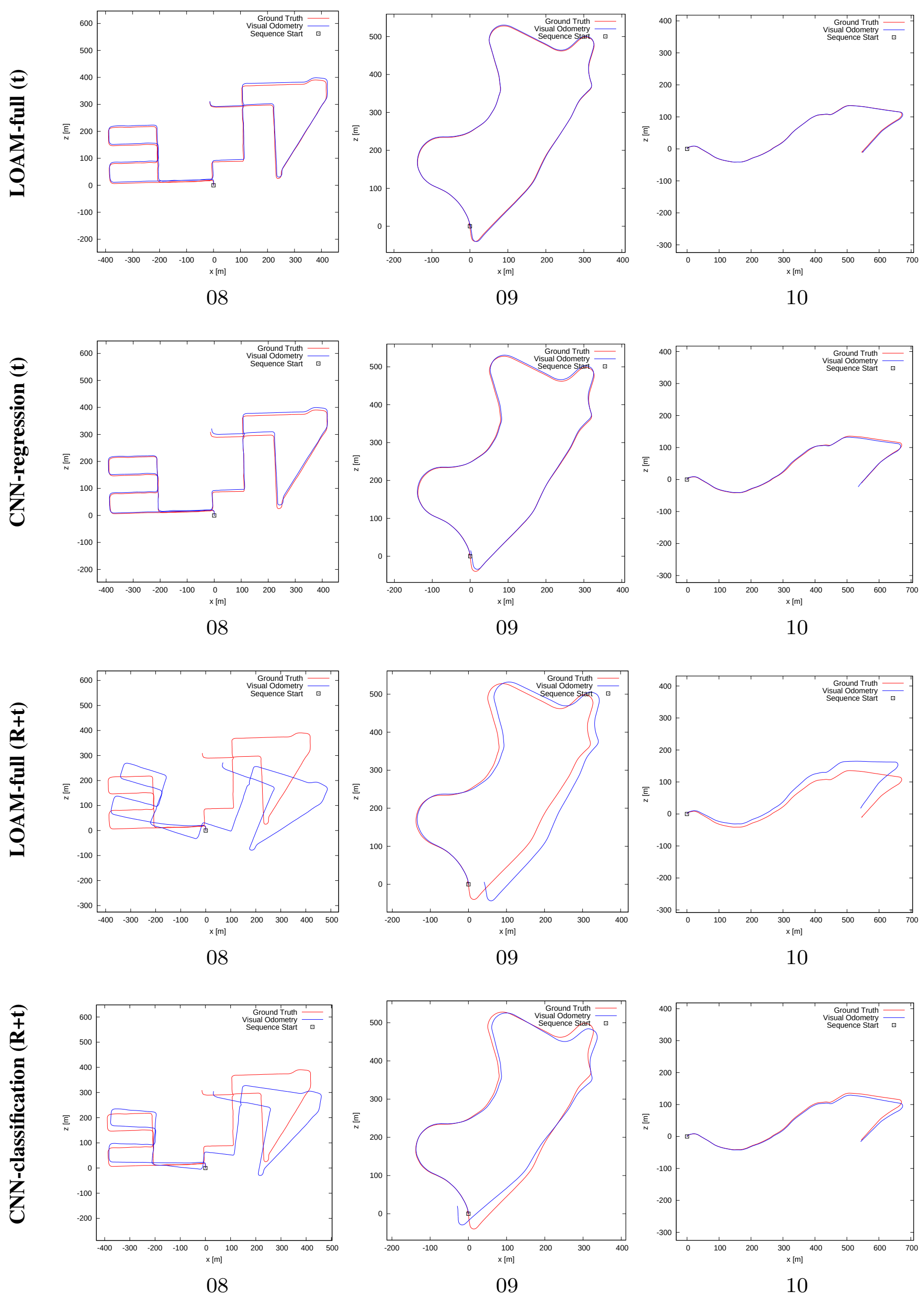
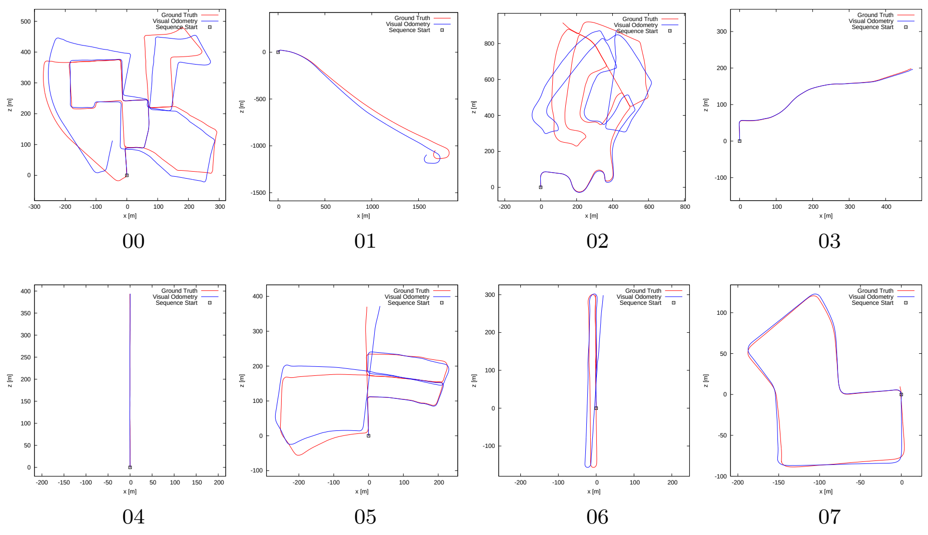


Fig. 11: Visual comparison of the results for LOAM (1st and 3rd row) and our CNNs (2nd and 4th row) on the KITTI sequences used for the testing only (sequences $08 - 10$). In the first two rows, only translation (t) parameters are estimated. The differences between the ground truth (red) and the estimated odometry (blue) are barely visible here. Bottom rows show the cases when all 6DoF motion parameters ($R + t$) are estimated.

LOAM-full:



CNN-classification (R+t):

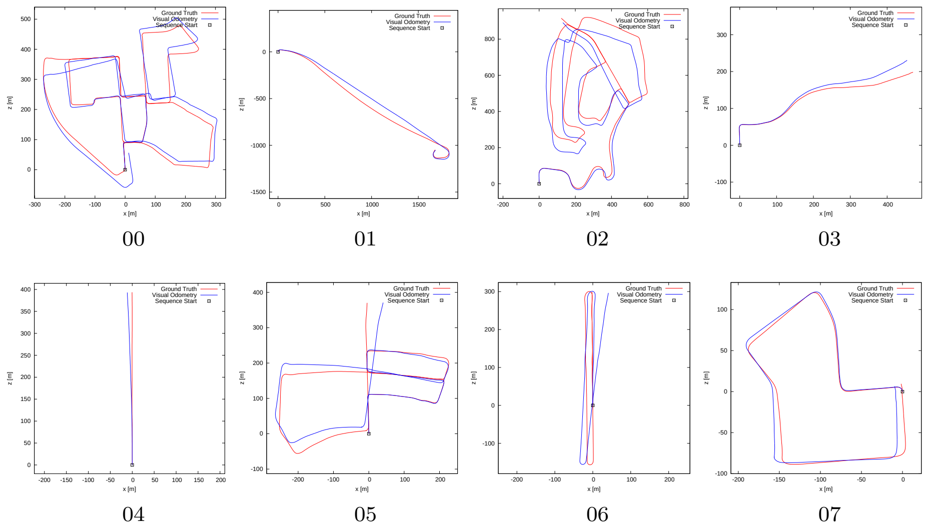


Fig. 12: Visualization of the LOAM and our CNNs results (blue) for the KITTI train sequences (00 - 07). The reference ground truth is colored by the red color.

We also submitted the results of our networks (i.e. the regression CNN estimating translational parameters only and the classification CNN estimating rotations) to the KITTI benchmark together with the outputs we achieved using the LOAM method in the online and the full mapping mode. The results are similar as in our experiments – the best performing LOAM-full achieves 3.49% and our CNNs 4.59% error. LOAM-online performed worse than in our experiments with error 9.21%. Interestingly, the error of our refactored

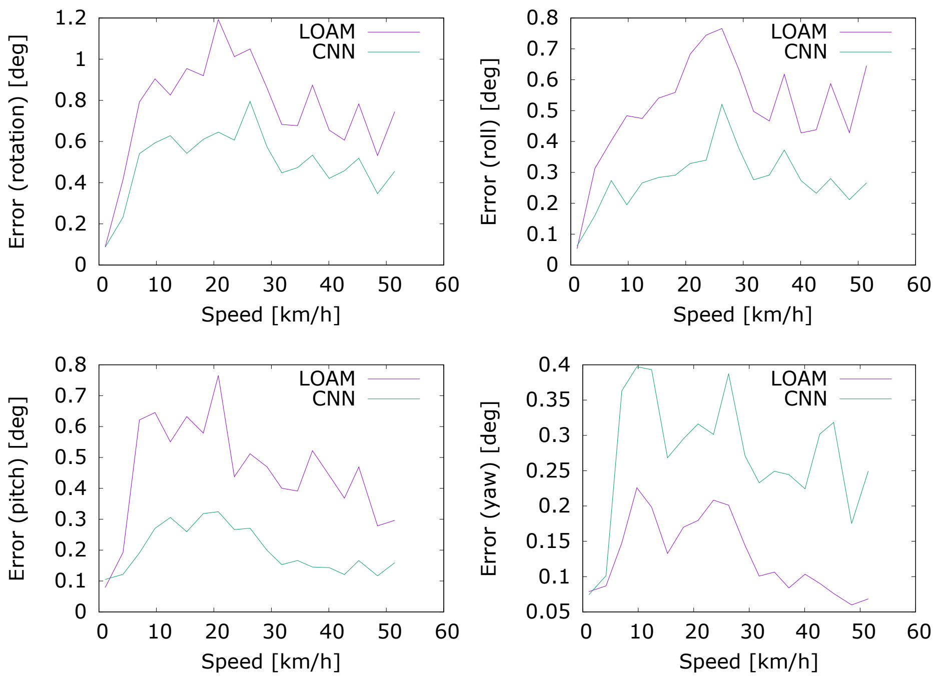


Fig. 13: The dependency of the rotation angle error on the driving speed for both our CNN method and the LOAM-full method. The error is evaluated for the total rotation and also for all the Euler angles separately.

original implementation of LOAM is more significant than errors reported for the original submission of the LOAM authors. This is probably caused by a fine-tuning of the method for the KITTI dataset which has never been published and authors refused to share both the specification/implementation used and the outputs of their method with us.

4.1 Discussion

The evaluation showed that our approach is able to estimate the translation motion parameters at a high rate with very good precision comparing to the state of the art method. This is the most significant outcome of our work since with the aiding of the IMU sensor, online odometry estimation and the point cloud registration is possible.

When we tried to demonstrate the ability of the proposed convolutional neural networks to estimate both the translation and the rotation, our evaluation showed that the results we achieved are inferior to the abilities of the LOAM approach. After deeper evaluation, we were able to identify the issue – our method performs worse in the estimation of heading direction. This is definitely the space for future work. One of the possible solutions would be probably an improvement of the training dataset, where this heading changes can be artificially introduced and a better model can be learned. The inspections of the failing cases showed that the situations when the driving direction is changing are challenging for both of the methods.

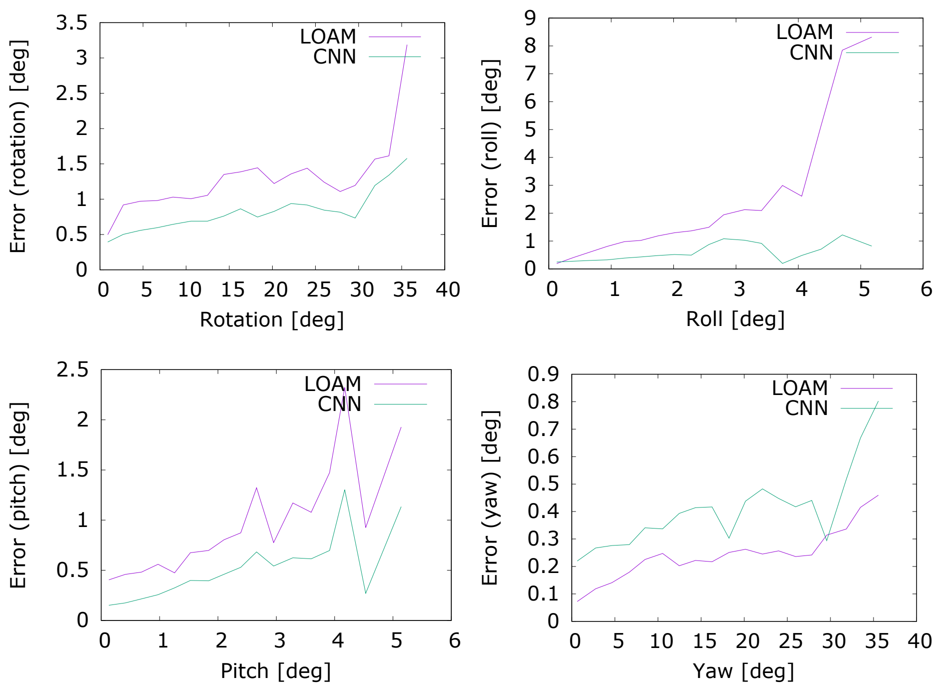


Fig. 14: The dependency of the rotation angle error on the rotation magnitude for both our CNN method and the LOAM-full method. The error is evaluated for the total rotation and also for all the Euler angles separately.

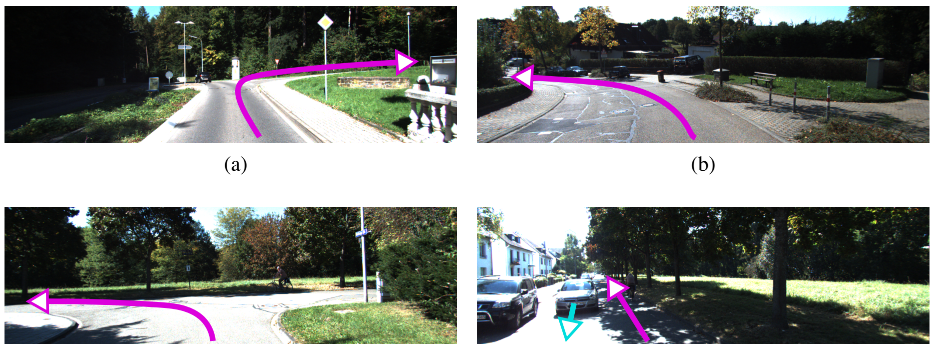


Fig. 15: Examples of failing cases with the largest error for LOAM: sharp sloped curve (a), long smooth curve (b) and the situation, when the car appears in the opposite direction (d) right after the vehicle turns left (c).

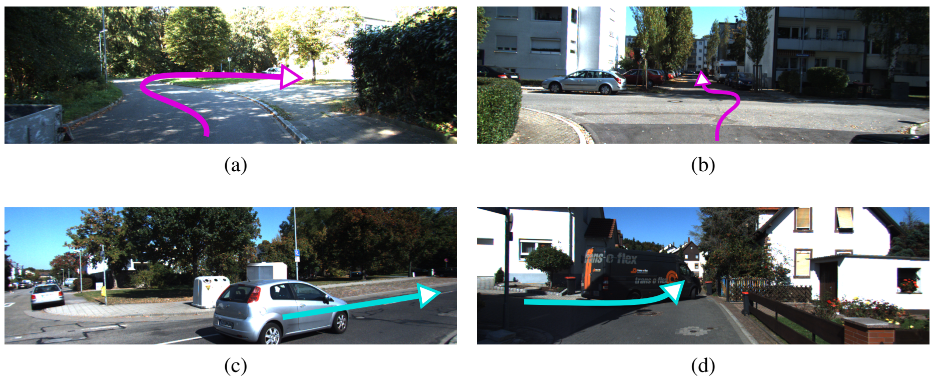


Fig. 16: Examples of failing cases with the largest error for our CNN method for the odometry estimation: sharp curve with almost 180° orientation change (a), long smooth curve (b) and the situation, when the car appears in the opposite direction (c) right after the vehicle turns left (d).

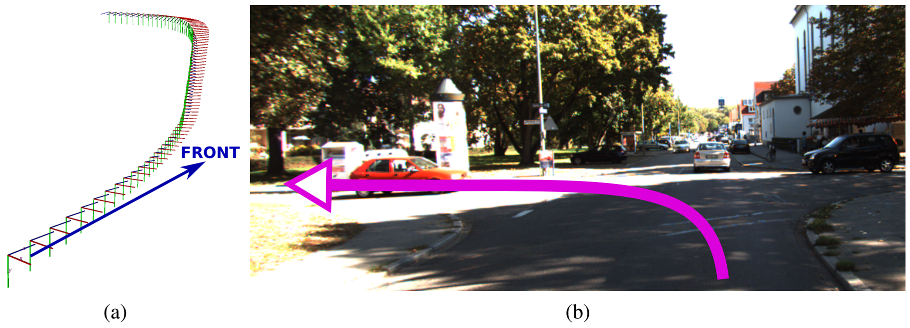


Fig. 17: The error in the ground truth of the KITTI Odometry dataset. At the beginning of the sequence 08, there is evidence in ground truth data about going uphill during the first turn left (a). According to the image data available (b), there is no such slope on this crossing.

The last but probably not least contribution of our work is the evidence that the convolutional neural networks can be used as a general concept for processing 3D Velodyne LiDAR data. This concept has been previously used in the ground segmentation task [22] and also in the vehicle detection problem [14]. However, it is necessary to admit that our (and also previous) approaches are limited to this type of 3D data. On the other hand, the LOAM method was originally designed for a spinning 3D rangefinder and therefore it can be considered more general in terms of input data.

5 Conclusion

This article presents a new method for the odometry estimation using convolutional neural networks. The most important outcome of our work is the proposal of networks for a very fast and accurate estimation of real-time translation motion parameters. For translation, the networks overcome the state of the art performance both in terms of precision and the speed in evaluations using the standard KITTI Odometry dataset.

The proposed solution can replace less accurate methods, such as odometry estimated from wheel encoders on the mobile platform. It is also able to solve the problems of the GPS-based systems in cases where the GNSS signal is insufficient or corrections are not available (e.g. in the indoor environments, in the forests, mines, etc.). If the rotational movement parameters (orientation) are provided by the IMU sensor, full 6DoF position is determined online. For the 3D mapping processes, therefore, the preview of the results and the collected data can be directly displayed to the system operator in the real time during the mission.

We also introduced an alternate topology of convolutional neural networks and a strategy for the prediction of the orientation (Euler angles roll, pitch and yaw). This allows for full estimation of visual odometry in real time. Our method uses the existing encoding of the 3D LiDAR data to be processed by CNN [14, 22]. It contributes as a proof of the universal usability of the convolutional neural networks for this data set.

As a future work, we have the ambition to deploy our methods of the odometry estimation in the real-word 3D LiDAR mapping solutions for both indoor and outdoor environments.

References

1. Besl, P.J., McKay, N.D.: A method for registration of 3-D shapes. IEEE Transactions on Pattern Analysis and Machine Intelligence **14**(2), 239–256 (1992). DOI 10.1109/34.121791
2. Bosse, M., Zlot, R.: Keypoint design and evaluation for place recognition in 2D lidar maps. Robotics and Autonomous Systems **57**(12), 1211 – 1224 (2009). Inside Data Association
3. Bosse, M., Zlot, R.: Place recognition using keypoint voting in large 3D lidar datasets. In: 2013 IEEE International Conference on Robotics and Automation, pp. 2677–2684 (2013). DOI 10.1109/ICRA.2013.6630945
4. Bosse, M., Zlot, R., Flick, P.: Zebedee: Design of a spring-mounted 3-d range sensor with application to mobile mapping. IEEE Transactions on Robotics **28**(5), 1104–1119 (2012). DOI 10.1109/TRO.2012.2200990
5. Chen, Y., Medioni, G.: Object modelling by registration of multiple range images. Image Vision Comput. **10**, 145–155 (1992). DOI 10.1016/0262-8856(92)90066-C
6. Douillard, B., Quadros, A., et al.: Scan segments matching for pairwise 3D alignment. In: Robotics and Automation (ICRA), 2012 IEEE Int. Conference on, pp. 3033–3040 (2012). DOI 10.1109/ICRA.2012.6224788
7. Eissfeller, B., Ameres, G., Kropp, V., Sanroma, D.: Performance of gps, glonass and galileo. In: Photogrammetric Week, vol. 7, pp. 185–199 (2007)
8. Geiger, A., Lenz, P., Stiller, C., Urtasun, R.: Vision meets robotics: The KITTI dataset. Int. Journal of Robotics Research (IJRR) (2013)
9. Grant, W., Voorhies, R., Itti, L.: Finding planes in lidar point clouds for real-time registration. In: Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ Int. Conference on, pp. 4347–4354 (2013). DOI 10.1109/IROS.2013.6696980
10. Kukko, A.: Mobile laser scanning system development, performance and applications (2013). URL `http://urn.fi/URN:ISBN:978-951-711-307-6`
11. Kukko, A., Kaartinen, H., Hyypp, J., Chen, Y.: Multiplatform mobile laser scanning: Usability and performance. Sensors **12**(9), 11712–11733 (2012). DOI 10.3390/s120911712. URL `http://www.mdpi.com/1424-8220/12/9/11712`
12. Langley, R.B.: Rtk gps. GPS World **9**(9), 70–76 (1998)
13. Lauterbach, H.A., Borrmann, D., He, R., Eck, D., Schilling, K., Nchter, A.: Evaluation of a backpack-mounted 3d mobile scanning system. Remote Sensing **7**(10), 13753–13781 (2015). DOI 10.3390/rs71013753. URL `http://www.mdpi.com/2072-4292/7/10/13753`
14. Li, B., Zhang, T., Xia, T.: Vehicle detection from 3D lidar using fully convolutional network. CoRR **abs/1608.07916** (2016). URL `http://arxiv.org/abs/1608.07916`

15. Pandey, G., McBride, J., Savarese, S., Eustice, R.: Visually bootstrapped generalized icp. In: Robotics and Automation (ICRA), 2011 IEEE Int. Conference on, pp. 2660–2667 (2011). DOI 10.1109/ICRA.2011.5980322
16. Pandey, G., McBride, J.R., Savarese, S., Eustice, R.M.: Toward mutual information based automatic registration of 3D point clouds. In: 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 2698–2704 (2012). DOI 10.1109/IROS.2012.6386053
17. Pathak, K., Birk, A., et al.: Fast registration based on noisy planes with unknown correspondences for 3-D mapping. Robotics, IEEE Transactions on **26**(3), 424–441 (2010). DOI 10.1109/TRO.2010.2042989
18. Rothe, R., Timofte, R., Gool, L.V.: Deep expectation of real and apparent age from a single image without facial landmarks. International Journal of Computer Vision (IJCV) (2016)
19. Segal, A., Haehnel, D., Thrun, S.: Generalized-icp. In: Proceedings of Robotics: Science and Systems. Seattle, USA (2009)
20. Velas, M., Spanel, M., Herout, A.: Collar line segments for fast odometry estimation from velodyne point clouds. In: IEEE Int. Conference on Robotics and Automation, pp. 4486–4495 (2016). DOI 10.1109/ICRA.2016.7487648
21. Velas, M., Spanel, M., Hradis, M., Herout, A.: Cnn for imu assisted odometry estimation using velodyne lidar. In: 2018 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC), pp. 71–77 (2018). DOI 10.1109/ICARSC.2018.8374163
22. Velas, M., Spanel, M., Hradis, M., Herout, A.: Cnn for very fast ground segmentation in velodyne lidar data. In: 2018 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC), pp. 97–103 (2018). DOI 10.1109/ICARSC.2018.8374167
23. Zhang, J., Kaess, M., Singh, S.: Real-time depth enhanced monocular odometry. In: 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 4973–4980 (2014). DOI 10.1109/IROS.2014.6943269
24. Zhang, J., Singh, S.: Loam: Lidar odometry and mapping in real-time. In: Robotics: Science and Systems Conference (RSS 2014) (2014)
25. Zhang, J., Singh, S.: Visual-lidar odometry and mapping: Low-rift, robust, and fast. In: IEEE ICRA. Seattle, WA (2015)