End-to-End LIDAR Odometry with DNN

Benjamin Skikos, Chunshang Li, Hamid Tahir

***Abstract*—This project modifies a deep recurrent convolutional network architecture used for end-to-end visual odometry for use with a LIDAR sensor instead of camera. Performance is comparable to that obtained with end-to-end visual odometry, but behind state-of-the-art classical LIDAR odometry on the tested dataset.**

I. Introduction

Odometry is the problem of recovering ego-motion (ex. a car moving through its environment) given sensor data. It is a well-studied field and state-of-the-art solutions are able to achieve adequate results for most applications [1], [2], [3], [4]. It is similar to the SLAM (Simultaneous localization and mapping) problem because recovering ego-motion localizes within an environment, but different in that odometry does not (in general) recover a map, recognize previously visited areas, or optimize old states given new measurements.

There are two main classical approaches to camera odometry: feature-based (ex [5] [6]) and direct (ex [7]). Feature-based methods process each frame and extract unique feature points. The feature point extraction algorithms are commonly hand-crafted and hand-tuned. By projecting feature points image-to-image and penalizing distance between matching features, ego-motion can be recovered (up to a scale factor). Direct methods re-project pixel patches between images, and penalize the pixel-level intensity difference between the reprojected patch and the overlapping section of the image.

LIDAR (light detection and ranging) is a time of flight sensor that uses pairs of laser emitters and detectors to measure distance. The Velodyne LIDAR used in this project consists of 64 laser-detector pairs arranged in a vertical array, mounted on a spinning sensor body such that the lasers sweep the environment as the body rotates. The emitters are fired on a repeating, discrete, clock cycle and each detected laser return is processed into a 3D point in the environment. These points are then aggregated over one full revolution to form a point cloud (example in Figure 1). In addition to measuring distance, these sensors also return an intensity reading, which is a measure of how much of the laser light was reflected back towards the sensor.

For LIDAR odometry, there are mostly two categories of approaches: feature-based [9] & scan-registration [10]. Feature-based is largely the same as camera, whereas scan registration is more specialized. The most common scan-registration techniques are part of the ICP (Iterative Closest Point) family, but there are some many others, all having the same goal of merging two scans with overlapping sections such that a single cohesive point cloud is formed.

The greatest challenge in applying classical techniques to odometry is achieving robustness to both noise-driven and systemic errors [11]. For example:

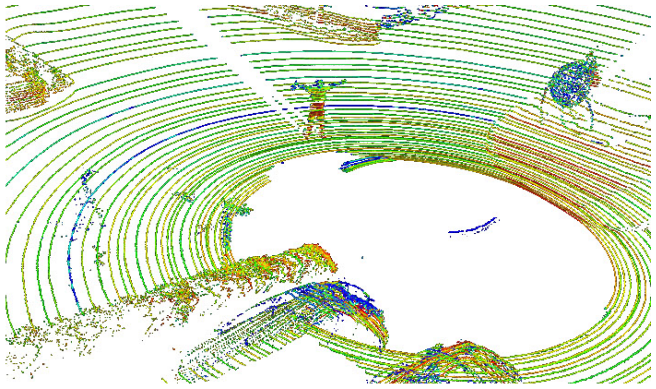
Fig. 1. Point cloud generated from a 64 beam Velodyne LIDAR [8]

- Occlusions between images/scans
- Incorrect feature correspondence
- Moving objects in the scene
- Changing lighting conditions between frames

In recent years, end-to-end visual odometry has started to be explored [12]. The appeal of using deep networks to perform odometry is that a deep network might learn to be robust to error caused by difficult-to-model noise or processes, in the same way that deep networks are able to classify difficult-to-model objects [13].

II. Related Work

In addition to [14] which this project is based on, there is also the work in [15]. In that work, several input frames are fed to a CNN-based architecture to perform LIDAR odometry. However, the choice of how many input frames to use is a hyperparameter that must be tuned, as opposed to a recurrent network that would learn how much past information to keep. Additionally, the authors were unable to train (or unsatisfied with the results from) a single network to output odometry for all of the six degrees of freedom that describe rigid body movement. Instead, a separate CNN was trained for each of the rotational degrees of freedom as well as a fourth CNN for the three translational degrees of freedom. If part of those networks are performing the same data processing, some efficiency is lost. Furthermore, the rotation prediction was reformulated as a classification task (classifying intervals of rotation) and then taking a weighted average across a softmax layer. This is limited in that the range of rotation must be decided ahead of time, loosing flexibility.

III. Network Architecture

The network architecture is quite similar to [14] with some simplifications in order to reduce the GPU memory requirements and training time. The input to the network is

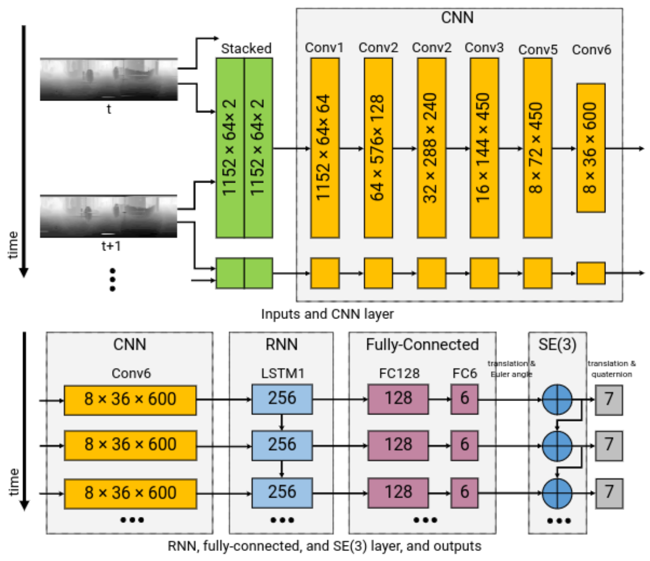

Fig. 2. LIDAR network architecture

a pair of sequential images stacked along the channel axis. The output is the 6DoF displacement between images.

A. *Network Input*

Comparing LIDAR to cameras, there are some important differences:

1) The points are collected continuously, so each aggregated point cloud has significant motion distortion due to ego-motion.
2) LIDAR rotation rate fluctuates a small amount, and some of the laser firings do not have a return, so the number of points varies between scans
3) The time of rotation is not an integer number of firing cycles, so the absolute azimuth angles sampled by LIDAR vary scan-to-scan
4) Since LIDAR is actively lighting the sampled surfaces with a laser, the intensity return is dependent on the angle of incidence and specularity of the surface.

As the network chosen for this project is designed to work with images, a scheme is required to convert LIDAR point clouds into images. The approach is based on [15]: The points in each point cloud are binned according their azimuth, and the mean from each bin is taken to populate the converted image. Empty bins are populated by linearly interpolating between populated bins. The final image has a width of 1152 azimuth bins and a height of 64 (each laser samples at a constant elevation). The width was chosen as a compromise between limiting the number of points in each bin (binning blurs the image) and to limit input image size to the network.

With this technique, each LIDAR point cloud is converted into a 2-channel image. The first channel is range, the second is intensity. The input to the network is a pair of sequential images stacked along the channel axis. An overview of the entire architecture is shown in Figure 2.

TABLE I
CNN CONFIGURATION

Layer	Kernel Size	Stride	Channels
Conv1	7 × 1	1 × 1	64
Conv2	5 × 1	2 × 1	128
Conv3	5 × 3	2 × 2	240
Conv4	3 × 3	2 × 2	450
Conv5	3 × 3	2 × 2	450
Conv6	3 × 3	2 × 1	600

B. *Convolutional Layer*

The role of this part of the network is to learn features from the pair of stacked images from which ego-motion can be inferred. This portion of the network in [14] is the front half of FlowNet [16] because FlowNet extracts optical flow, and it is possible to estimate ego-motion from optical flow [17]. For this project, the CNN layer was simplified by shortening it and reducing the channel size throughout. The kernel sizes and strides were also modified because LIDAR images are much wider than they are tall. This is because the sensor samples its environment much more densely in azimuth than elevation. For this reason, in the early CNN layers the kernels are also wide, in an attempt to summarize the denser information along azimuth before applying square kernels. A summary of CNN layers' configurations is shown in Table I. Dropout is applied in the last couple of layers to add regularization [18]. Relu activations are applied after each convolutional layer.

C. *Recurrent Layer*

The recurrent layer in this project is a single LSTM cell. An RNN is used to try to capture the dynamics of the ego-motion [14]. An LSTM is used because it avoids the problems of exploding/vanishing gradient [19]. The paper this project is inspired by used 2 LSTM cells with more channels, but a network that size is not practical to train on a single NVIDIA TITAN Xp GPU, so it was simplified for the purposes of the project.

D. *Fully Connected Layer*

There are two fully connected layers that convert the output of the RNN directly into the relative frame-to-frame displacement. The displacements are taken to be meters and the relative rotation is parameterized with moving-axis-zyx Euler angles in radians.

E. *SE3 Composition Layer*

This part of the network is a single layer that takes the relative displacements from the previous section and composes them with the previous pose. The previous pose is represented as a 3D position and a quaternion to represent orientations. This composition layer is present in order to penalize cumulative errors from the ground truth instead of just frame-to-frame. In addition, composing poses over multiple time steps minimizes the effect of noise in GPS ground truth. This layer does not contain any trainable parameters.

IV. Training

A. Cost Function

The cost function is comprised of two separate but related terms weighted by α, where alpha controls the relative contribution of each of term.

$$L = \alpha L_1 + (1 - \alpha) L_2$$

1) Frame-to-Frame Loss: This cost penalizes the frame-to-frame output before it passes through the SE3 composition layer. It is the same as the one in [14], except the term for uncertainty was removed. The cost function with uncertainty requires a regularization term to avoid a singularity when the network is outputting zero uncertainty. Since uncertainty estimation was not a goal of the project, that component was removed to simplify the problem. Therefore the cost function used was

$$L_1 = \frac{1}{t} \sum_{k=1}^{t} \|\hat{p}_k - p_k\|_2^2 + \kappa_\phi \, \|\hat{\phi}_k - \phi_k\|_2^2$$

Note that subtracting two orientations parameterized by Euler angles from each other is only a good approximation of distance between orientations when those orientations are close. In practice, the network output quickly converges to be close enough.

2) SE3 Composition Loss: This cost penalizes the difference between the networks output after the composition layer and the ground truth. The cost based on [14], except for the quaternion distance. Directly subtracting quaternions is problematic because every rotation can be represented by two quaternions. Instead, subtraction was replaced with cosine distance.

$$L_2 = \frac{1}{t} \sum_{k=1}^{t} \|\hat{p}_k - p_k\|_2^2 + \kappa_q (1 - \langle \hat{q}_k, q_k \rangle^2)$$

B. State Propagation

During training the state of the network is propagated to the temporally-subsequent batch when it is processed. In [14] the authors picked random start and end points within the training sequences to generate a set of sequences, then shuffled those sequences during training. A different strategy was used for the project because random subsequences share some images, resulting in some images being overrepresented within in each epoch.

For this project, one epoch was run with the batches being temporally-ordered. Subsequent epochs randomize batch order, using state initialization from the temporally-previous batch.

C. Data Augmentation

Any augmentation techniques that invalidate the ground truth trajectory can not be used (ex. rotation). In the project, the images were mirrored horizontally and in time for data augmentation. Mirroring horizontally is necessary because LIDAR continuously samples its environment, and the order of that sampling is encoded in column index in the LIDAR image.

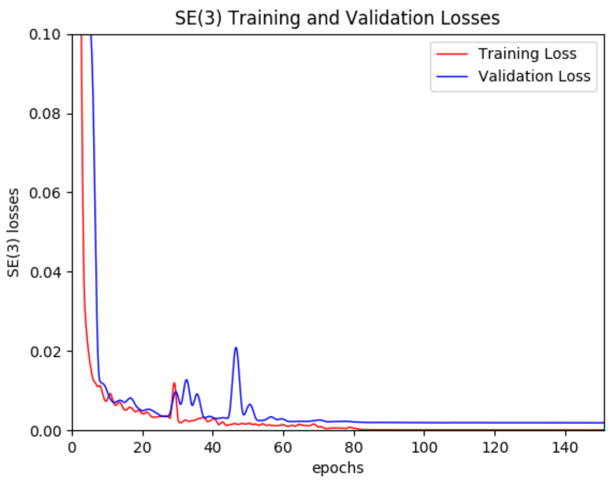

Fig. 3. SE(3) Loss plotted over epoch

D. Training

RNNs are notorious for being difficult to train, and this architecture follows suit in that respect. The authors of [14] used initial weights for the CNN portion from flownet to aid convergence. Since CNN for LIDAR is different, that was not possible. Instead, it was empirically found that immediately attempting to use backpropagation through time was unlikely to result in convergence. Instead, the network was first trained of pairs of frames. Once that converged (training loss plotted in Figure 3), the weights were used to initialize the network to fine-tune over longer time sequences.

V. Evaluation

A. Dataset

The project was evaluated using the popular KITTI odometry dataset [20]. The dataset was collected from car-mounted sensors (LIDAR and camera). It consists of several drives in Germany, with ground truth provided by a GPS-INS system. 11 sequences are provided with ground truth for development. Sequences 0, 1, 2, 8, & 9 were used for training, sequence 7 for validation, and sequences 3, 4, 5, 6, & 10 for testing. A sample trajectory result on one of the test sequences in shown in Figure 4.

VI. Discussion

As can be seen from Table II, the results are comparable with the results presented in [14]. Classical odometry techniques incorporating LIDAR tend to outperform camera-only [20] so the intuitive result is that end-to-end LIDAR would outperform end-to-end camera. Results for rotation follow this intuition but results for translation do not. Speculating, this may be because the LIDAR image has a 360 degree field of view, containing two vanishing points where the apparent optical flow when moving forward is zero. In contrast, a

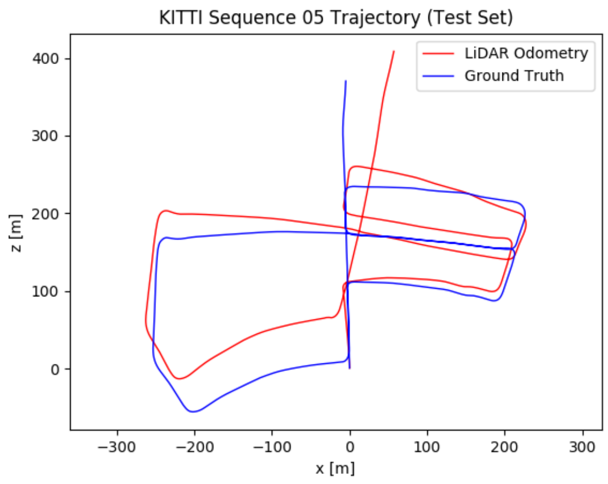

Fig. 4. Trajectory obtained on KITTI sequence 5 (subset of test set)

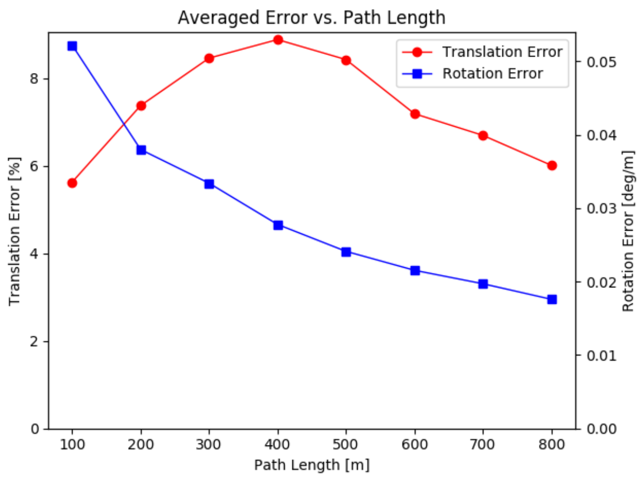

Fig. 5. Test error vs various path length. For a given path length, the error is averaged over path lengths form 100m to 800m from all test sequences

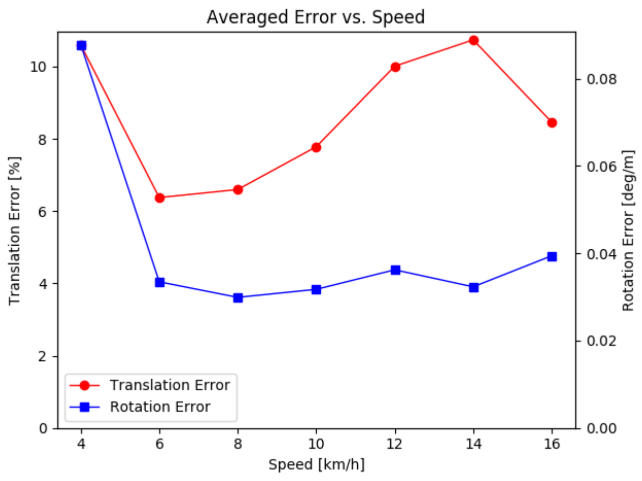

Fig. 6. Test error vs various trajectory speeds. The error is averaged over all subsequences of a given speed

TABLE II
RESULTS ON TESTING SEQUENCES

Seq.	Ours		ESP-VO	
	$t_{rel}(\%)$	$r_{rel}(\circ/100m)$	$t_{rel}(\%)$	$r_{rel}(\circ/100m)$
03	11.69	**3.95**	**6.72**	6.46
04	9.77	**4.96**	**6.33**	6.08
05	6.01	**2.62**	**3.35**	4.93
06	11.70	**3.85**	**7.24**	7.29
07	4.40	**2.25**	**3.52**	5.02
10	**8.82**	**3.65**	9.77	10.2
mean	8.73	**3.54**	**6.15**	6.66

- t_{rel}: average translational error on trajectories of length 100-800m
- r_{rel}: average rotational error on trajectories of length 100-800m

camera image contains just one vanishing point. Also, it is possible the various non-linearities throughout the network are not taking full advantage of the high-quality range information present in LIDAR images. Lastly, this project's network is smaller than that in [14], having reduced expressive capacity.

Figures 5 & 6 show how the error changes with path length and vehicle speed. For speed, it is not surprising that the error is relatively high when the speed is slow, because most turns happen at low speed, and turning is under-represented in the training data relative to mostly straight travel.

In this project, a simplified network (from [14]) was trained in order to check reproducibility. Later, it was noticed that the LIDAR version generalized more readily on test data, requiring less fiddling with hyper-parameters. This is an intuitive result given that LIDAR actively illuminates its scene, so does not suffer from changing lighting conditions or shadows.

It was discovered late in the project that the LIDAR data included in the KITTI odometry dataset has been corrected for ego-motion. Consequently, the horizontal mirroring was not required when using temporal mirroring for data augmentation. Even so, better performance was observed using this technique possibly because it ensures that there is an average forward velocity of zero in the training set, avoiding the network learning a bias.

The KITTI dataset is relatively small and does not contain adverse environmental conditions (such as snow or rain). Therefore, the ability of a deep network to learn to cope with adverse environmental conditions was not evaluated. A good dataset to explore this property is the Ford Campus dataset [21], which includes snowy scenes.

The planar convolutions in this network are not invariant to translation due to the 360 degree field of view. Spherical convolution has this property [22] and so might be useful in this application.

VII. CONCLUSION

This project demonstrates that end-to-end methods applied to visual odometry can also be applied to LIDAR odometry. The accuracy achieved in this project is comparable to that in [14], despite the network being smaller. However, the large gap in performance between classical visual odometry and classical LIDAR odometry was not reproduced, so further improvement is likely possible. There remain additional avenues to improve performance beyond tweaking the existing architecture.

References

[1] D. Scaramuzza and F. Fraundorfer, “Visual odometry [tutorial],” *IEEE Robotics Automation Magazine*, vol. 18, pp. 80–92, Dec 2011.

[2] F. Fraundorfer and D. Scaramuzza, “Visual odometry : Part ii: Matching, robustness, optimization, and applications,” *IEEE Robotics Automation Magazine*, vol. 19, pp. 78–90, June 2012.

[3] J. Zhang and S. Singh, “Loam: Lidar odometry and mapping in real-time,” in *Robotics: Science and Systems Conference*, July 2014.

[4] J. Zhang and S. Singh, “Visual-lidar odometry and mapping: low-drift, robust, and fast,” in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2174–2181, May 2015.

[5] A. I. Mourikis and S. I. Roumeliotis, “A multi-state constraint kalman filter for vision-aided inertial navigation,” vol. 22, pp. 3565 – 3572, 05 2007.

[6] C. Forster, M. Pizzoli, and D. Scaramuzza, “Svo: Fast semi-direct monocular visual odometry,” 05 2014.

[7] J. Engel, V. Koltun, and D. Cremers, “Direct sparse odometry,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Mar. 2018.

[8] *HDL-64E*. Velodyne LiDAR.

[9] T. Y. Tang, D. J. Yoon, F. Pomerleau, and T. D. Barfoot, “Learning a bias correction for lidar-only motion estimation,” *CoRR*, vol. abs/1801.04678, 2018.

[10] S. Rusinkiewicz and M. Levoy, “Efficient variants of the icp algorithm,” pp. 145–152, 02 2001.

[11] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. D. Reid, and J. J. Leonard, “Simultaneous localization and mapping: Present, future, and the robust-perception age,” *CoRR*, vol. abs/1606.05830, 2016.

[12] S. Wang, R. Clark, H. Wen, and N. Trigoni, “Deepvo: Towards end-to-end visual odometry with deep recurrent convolutional neural networks,” *CoRR*, vol. abs/1709.08429, 2017.

[13] K. He, X. Zhang, S. Ren, and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification,” *CoRR*, vol. abs/1502.01852, 2015.

[14] S. Wang, R. Clark, H. Wen, and N. Trigoni, “End-to-end, sequence-to-sequence probabilistic visual odometry through deep neural networks,” *The International Journal of Robotics Research*, vol. 0, no. 0, p. 0278364917734298, 0.

[15] M. Velas, M. Spanel, M. Hradis, and A. Herout, “CNN for IMU assisted odometry estimation using velodyne lidar,” *CoRR*, vol. abs/1712.06352, 2017.

[16] P. Fischer, A. Dosovitskiy, E. Ilg, P. Häusser, C. Hazirbas, V. Golkov, P. van der Smagt, D. Cremers, and T. Brox, “Flownet: Learning optical flow with convolutional networks,” *CoRR*, vol. abs/1504.06852, 2015.

[17] G. Adiv, “Determining three-dimensional motion and structure from optical flow generated by several moving objects,” vol. PAMI-7, pp. 384 – 401, 08 1985.

[18] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[19] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Comput.*, vol. 9, pp. 1735–1780, Nov. 1997.

[20] A. Geiger, P. Lenz, and R. Urtasun, “Are we ready for autonomous driving? the kitti vision benchmark suite,” in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[21] G. Pandey, J. R. Mcbride, and R. M. Eustice, “Ford campus vision and lidar data set,” *Int. J. Rob. Res.*, vol. 30, pp. 1543–1552, Nov. 2011.

[22] T. S. Cohen, M. Geiger, J. Köhler, and M. Welling, “Spherical cnns,” *CoRR*, vol. abs/1801.10130, 2018.