Deep Learning for Laser Based Odometry Estimation

Austin Nicolai, Ryan Skeele, Christopher Eriksen, and Geoffrey A. Hollinger
Robotics Program
School of Mechanical, Industrial, & Manufacturing Engineering
Oregon State University, Corvallis, Oregon 97331
Email: {nicolaia, skeeler, eriksenc, geoff.hollinger}@oregonstate.edu

***Abstract*—In this paper we take advantage of recent advances in deep learning techniques focused on image classification to estimate transforms between consecutive point clouds. A standard technique for feature learning is to use convolutional neural networks. Leveraging this technique can help with one of the biggest challenges in robotic motion planning, real time odometry. Sensors have advanced in recent years to provide vast amounts of precise environmental data, but localization methods can have a difficult time efficiently parsing these large quantities. In order to address this hurdle we utilize convolution neural networks for reducing the state space of the laser scan. We implement our network in the Theano framework with the Keras wrapper. Input data is collected from a VLP-16 in both small office and large open environments. We present the results of our experiments on varying network configurations. Our approach shows promising results, achieving (per direction) accuracy within 10 cm and an average network prediction time of 4.58 ms.**

I. Introduction

One of the most difficult problems in robotics is accurate position estimation. We have sensors capable of providing us with high fidelity data, however, current processing methods are unable to take advantage of this in real-time. The Velodyne VLP-16 Puck (VLP-16) used in this work can output up to 300,000 laser points per second. An example map created with the VLP-16 can be seen in Figure 1. We propose to leverage deep learning to capitalize on GPU processing speed, reduce the state space of the sensor, and predict robot odometry without loop closure directly from the laser returns of the VLP-16.

In this project, we leverage the benefits afforded by deep learning and apply it to the robotic localization domain. Specifically we apply deep learning to develop a novel technique for laser-based odometry, the use of a laser scanner to estimate the change in a robot's position over time. The work presented in this paper provides a method that leverages the benefits of deep learning to estimate odometry directly from high fidelity sensors. More specifically, we consider state-of-the-art 2D and 3D feature extraction approaches, such as those used in image classification, to reduce the state space of the VLP-16 and deal with the vast amounts of data provided by these sensors. After the state space has been reduced, we propose using standard fully connected layers to learn the odometry estimation from these extracted features.

To implement our approach, we generate a custom data set for an indoor laboratory and office based environment with the VLP-16 sensor using wheel based odometry. We perform a variety of experiments with this data set on different networks in order to fine tune and improve our results. We carry out these experiments in a Python environment using the Theano framework and Keras wrapper.

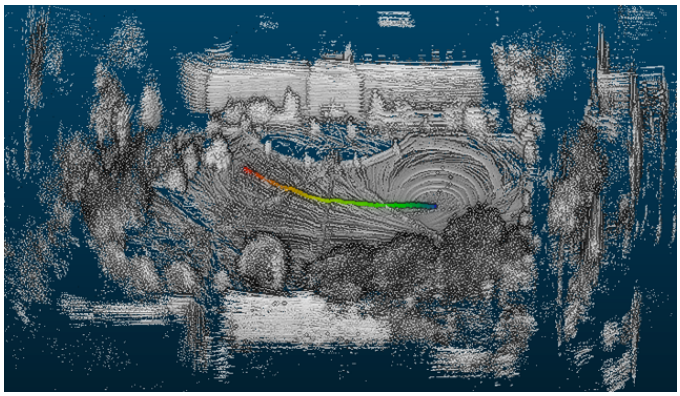

Fig. 1: Map of Oregon State University's library quad using the VLP-16. Post processing provided by Real Earth, Inc. *http://www.realearth.us*

The remainder of this paper is organized as follows. We first discuss related work in image classification domains and current odometry methods in Section II. Next, we present a formulation of our proposed method and discuss the different network structures considered and experimented on in Section IV. Finally, we discuss the results of our experiments in Section V and discuss avenues for future work in Section VII.

II. Related Work

In recent years, deep learning research has met with a remarkable level of success in a variety of applications, most notably object recognition and classification [8] [9] [17]. This is in large part due to the fact that deep learning has allowed training data sets to grow from an order of thousands (e.g. LabelMe [14]), to tens of thousands (e.g. CIFAR-10/100 [7], NORB [10]), and even millions of training examples (e.g. ImageNet [3]). Many such networks utilize convolutional neural networks (CNNs) as they are much easier to train than standard feedforward neural networks due to fewer connections and parameters [9], making learning in a deep architecture actually feasible.

Wheel based odometry utilizes encoders on a robot's wheels, factoring wheel rotation into a forward kinematic model to estimate the motion of the robotic base. This process is known as *dead reckoning*, and is known to result in a diverging estimate of a robot's position over time due to the fact that odometry errors are compounded over time. For this reason, localization methods usually combine odometry measures with some form of absolute position measure (e.g. using laser scanner returns to estimate a robot's position in a pre-constructed map) to keep errors from diverging. In addition to wheel encoders, classical odometry measures often use an IMU, a combination of a three accelerometers and gyroscopres. Both of these measures are often known to give noisy estimates however, the most prominent source of which is wheel slippage.

One method to circumvent these kinds of errors is a process known as *scan-matching* (e.g. [11] [18]), which often uses a technique known as Iterative Closest Point (ICP) [2] to iteratively minimize the difference between two clouds of points, in this case two consecutively laser scans. The estimated transformation is used to estimate the motion of the robotic base. Despite popular use and developments in efficiency, ICP is still expensive to compute and furthermore sensitive to large differences in the initial point distributions, as the correct point correspondence is difficult to find. We note recent success by Zhang and Singh [19] [20] in achieving low-drift, real-time odometry using scan-matching by simultaneously running two algorithms - one that runs at a fast rate but performs only course matching and another that runs an order of magnitude slower but performs fine grained matching and point registration. The fidelity achieved in this approach able to maintain accurate position estimates over time without the use of external, absolute position measures.

Another technique that has met with success is visual odometry (e.g. [1], [13]), a process that uses image features to estimate the ego-motion of a camera. While sensitive to features used in image processing, recent performance improvements on image-based applications (particularly with the use of deep learning) have made this a particularly appealing approach. Of particular interest is recent work done by Konda and Memisevic [6] that uses a convolutional neural network to estimate visual odometry from stereo images, using a softmax at the final layer to represent discretized direction and velocity changes.

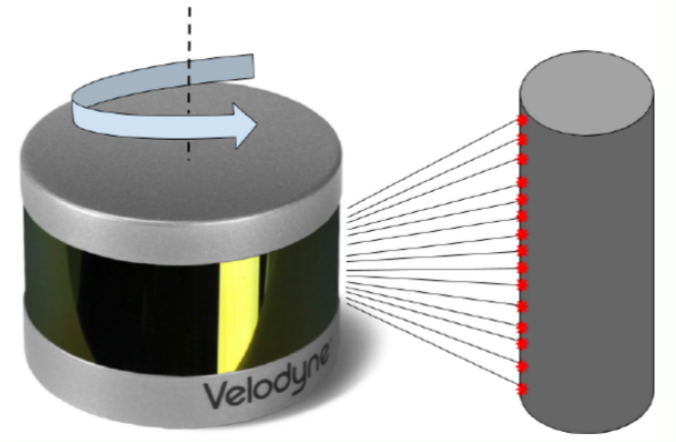

Fig. 2: VLP-16 spinning 16 lasers a full 360°

Similar to Konda and Memisevic's work, we reduce high-dimensional point cloud data to a depth image that we can pass into a CNN to perform motion estimation. Unlike their approach, we estimate motion from lidar rather than visual information and estimate motion changes using a regression rather than discretized/softmax approach. The benefit of our approach is that it circumvents the high computational constraints typically associated with scan matching, provides predictions in continuous space, and requires no hand-tuning of features for course scan matching.

III. Data Set

A. Velodyne Puck

The VLP-16 has a range of 100m with an accuracy of ±3cm. The VLP-16 can also provide intensity values, but for now we ignore this as we are only interested in spatial features. There are 16 lasers angled from ±15°. These spin a full 360° at up to 20Hz. The VLP-16 communicates data packets over User Datagram Protocol (UDP), and can be recorded for later playback as a pcap file. An illustration of the VLP-16 can be seen in Figure 2.

B. Data Set Generation

We generated our dataset by first mounting the VLP-16 on top of a mobile robotic base and recording both depth scans from the laser as well as odometry measurements from the wheeled robot as it was driven manually. Data collection was performed at 5 Hz and generated roughly 3,300 depth scans. For input into the network, each scan was first converted

(a) The office area used for collecting training data.

(b) The modified Turtlebot robot used for data collection.

(c) The open lab area used for collecting training data.

Fig. 3: Images of our modified Turtlebot gathering training data in our various indoor environments.

into a panoramic depth image, before being paired with the subsequent scan. Each image pairing was annotated with the recorded, ground truth odometry change. We augmented our dataset via two methods: First, we additionally paired each scan with the second and third subsequent scans to create additional examples. Second, for each set of scan pairings, we additionally created an example for the reverse ordering. This gave us a final data set of roughly 16,000 image pairs. Given our data collection rate and robot movement speed, the second and third subsequent scans were not too dissimilar for odometry purposes. Data was collected in both a confined, indoor office environment as well as a large, open high-bay lab environment. This can be seen in Figure 3.

IV. Problem Formulation

A. Voxel 3D Convolution

A point cloud represents a continuous 3D space. In order to extract features from this, a standard method is to begin by discretizing the space. In 3D, this discretization could come in the form of a voxel grid. With a 3D voxel grid, this would allow us to perform 3D convolution to leverage the spatial relationship of the point cloud points in feature extraction. This has been done in previous work for extracting features from 3D point clouds for terrain classification [12]. Others have used voxel grids to extract generic features from point clouds [5]

However, as always, when discretizing large continuous spaces, dimensionality becomes an issue. In our case, the precision and range of the VLP-16 poses a dimensionality concern. Given the VLP-16's 100m range and $\pm 15^{\circ}$ field-of-view, we would need a voxel grid length/width of 200m. Assuming the VLP-16 is sitting on a robot 1m off the ground, on flat terrain, the voxel grid height would need to be 27m. Given the VLP-16's 3cm accuracy, this yields a total of 38,811,960,000 voxels. Assuming only 1m discretization (far too inaccurate), over 1,000,000 voxels would still be required. As such, it is intractable for us to use a discretized, 3D convolution approach to this problem.

B. Image Based 2D Convolution

A simple approach to using point clouds for odometry is to revert back to standard image classification techniques. To do this we can modify our point cloud into a 2D projection of the environment; that is, a panoramic depth image. We do this by binning the raw VLP-16 scans into pixel representations. Using this depth image and a standard 2D convolutional neural network, we can extract spatial features of the environment.

By inputting consecutive VLP-16 scans into the network, it allows us to find feature locations in both projections. By using additional fully-connected layers in our network, the network can learn the patterns in these feature movements to provide odometry estimates.

C. Experiments

As previously discussed, since 3D, voxel based, convolution is not feasible for our problem, we based our network structure on that of standard 2D convolution based image networks. That is, in general, our network begins with a series of 2D convolutional and max pooling layers for feature extraction. It then contains several different sized fully-connected layers, before finally outputting the estimated odometry values. An example network structure can be seen in Figure 4.

More specifically, we experimented with a variety of network parameters and structures to find the one that worked best for our problem. For all network structures, the "Mean Squared Error" loss function was used. The following network changes were experimented with:

- Pre-training the convolution filters
- Varying the colvolution filter sizes
- Varying the number and ordering of convolution and pooling layers
- Varying the fully connected layer sizes
- Varying the number of fully connected layers
- Varying the type/amount of regularization on layers
- Varying the training method

For all experiments, the Theano framework with Keras wrappers were used in a Python environment. Training was done on a dedicated, high end PC. The dedicated PC is a quad-core, i7 machine with 16GB RAM and an Nvidia GeForce GTX TITAN Z GPU. The TITAN Z has 12GB of total RAM with 5760 CUDA cores, allowing for fast parallelization in deep learning training.

V. Results and Discussion

A. Pre-training the Convolution Filters

The first set of network variations tested was the impact of pre-training the convolution filters. Pre-training refers to using autoencoder based, unsupervised training on the input depth images to learn good features. The results of the unsupervised training are then used to initialize the convolution layers in the full network before the supervised training begins.

In tests both with and without pre-training, we observed that pre-training the weights results in better performance. Without pre-training, the learning both takes longer and does not perform as well. This could be due to several reasons: First, when learning from scratch, the network learns noisier features that do not represent the data as well. Second, learning from scratch results in different features being learned for the two separate inputs. Depending on what is learned, the same features may not be tracked in each. By utilizing pre-training, the convolution filters for both inputs can be initialized to with the same features.

B. Convolution/Pooling Layer Variations

Next, experiments were performed to investigate the effect of different convolution/pooling structures, with motivation taken from the work of Simonyan and Zisserman [15]. In these experiments, different combinations of convolutions sized 3x3 and 5x5 and filter bank sizes of 32, 64, and 128 were tested. Additionally, the number of convolution layers between each pooling layer was varied. The full set of combinations

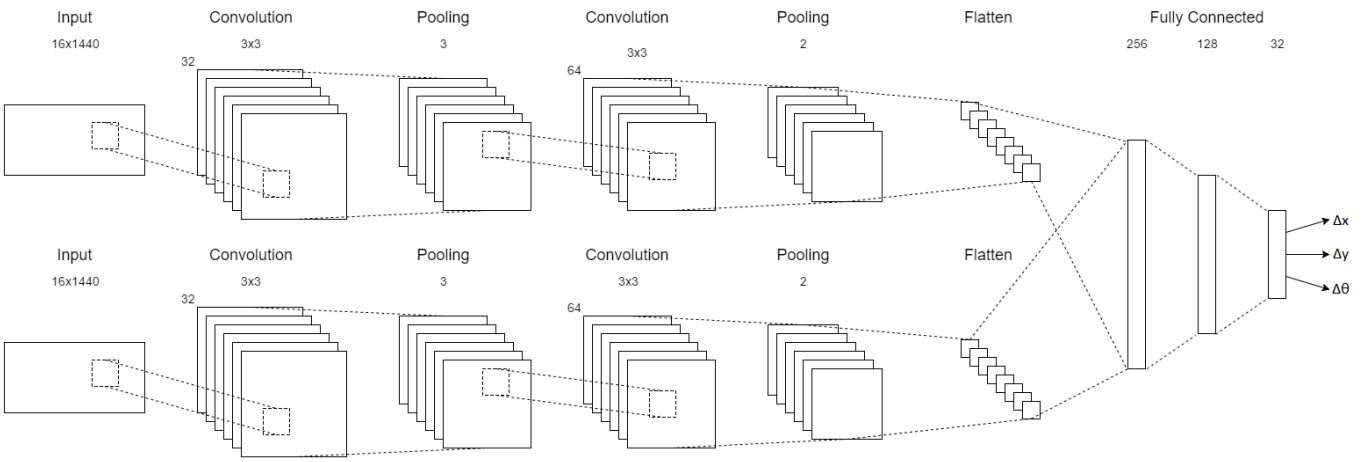

Fig. 4: An example network structure used in this work. In this network, the subsequent depth image inputs go through two sets of convolution/pooling layers before being fed into a series of three fully connected layers to estimate the odometry. In this network, the convolution was of size 3x3 with filter banks sized 32 and 64, respectively. The fully connected layers were of sizes 256, 128, and 32.

tested were as follows (notation: "conv{filter size}-{filter bank size}"):

- conv3-32, pool, conv3-32, pool
- conv5-32, pool, conv3-32, pool
- conv3-32, pool, conv3-64, pool
- conv3-32, conv3-32, pool, conv3-32, conv3-32, pool
- conv5-32, conv5-32, pool, conv3-32, conv3-32, pool
- conv3-32, conv3-32, pool, conv3-64, conv3-64, pool
- conv5-32, conv5-32, pool, conv3-64, conv3-64, pool
- conv3-32, conv3-32, pool, conv3-64, conv3-64, pool, conv3-64, conv3-64, pool
- conv3-32, conv3-32, pool, conv3-64, conv3-64, pool, conv3-128, conv3-128, pool
- conv3-32, conv3-32, conv3-32, pool, conv3-64, conv3-64, pool
- conv3-32, conv3-32, conv3-32, pool, conv3-64, conv3-64, conv3-64, pool

For these architectures, the results matched those reported by Simonyan and Zisserman. Using multiple, small convolution filter sizes achieved better results than that of a single, larger filter size. Additionally, increasing the filter bank size after pooling layers also achieved better results.

In our experiments, both of these factors appeared to have a saturation point after which no additional benefit was gained. For us, adding more than three convolution layers between pooling, and increasing the filter bank size above 64, gave no additional benefit. The highest accuracy architecture tested was *conv3-32, conv3-32, pool, conv3-64, conv3-64, pool*.

C. Fully Connected Layer Variations

For these tests, the number of fully connected layers, and their sizes, were varied in the overall network. The number of layers was varied between two and four, with the number of hidden nodes ranging from 16 to 1024.

In our experiments, we found that steadily decreasing the size of the fully connected layers achieved better results. In general, we achieved better results, and faster learning, with more, smaller sized layers. That is, an architecture of (notation: "fc{number of hidden nodes}") *fc128, fc64* was more desirable than *fc256*. The highest accuracy architecture tested was *fc128, fc64, fc16*.

D. Regularization and Training Variations

The last set of experiments we performed was in changing the amount of regularization used, and the training method employed. Dropout layers of different amounts were experimented with. Additionally, maxnorm constraints were tested on the convolution layers, and L1/L2 regularization was tested on the fully connected layers. The various training optimizations used were: RMSProp, Adagrad, Adadelta, and Adam.

Similar to Srivastava et al. [16], we found that adding dropout layers to our architectures increased performance. In our experiments, increasing the dropout beyond 25% began to lower performance. Additionally, adding a maxnorm constraint to the convolution layers helped to reduce overfitting in the unsupervised pre-training stage. Both L1 and L2 regularization aided in reducing overfitting for the overall network, with L2 providing the best performance. The combination that provided the best performance was using dropout layers of 10%, a maxnorm constraint of 4, and L2 regularization penalty of 0.01.

For the different training optimizers used, no significant difference in performance was noticed. All of the optimizers converged to similar performance. The only difference of note was that the Adam optimizer tended to converge more quickly than the rest.

E. Full Network

Overall, the most accurate network architecture tested was: *conv3-32, conv3-32, pool, conv3-64, conv3-64, pool* for the

feature extraction stage and *fc128, fc64, fc16* for the fully connected stage. The network used dropout layers of 10%, a maxnorm constraint of 4 on the convolution layers, and an L2 regularization penalty of 0.01 on the fully connected layers. Additionally, it took advantage of unsupervised pre-training to seed the convolution filters in the full network.

For the test data set, this network architecture achieved an average odometry estimate error of 0.0763 m, or 7.63 cm. This represented a roughly 30% error in the estimate. Across the test set, the largest error measured was 0.611 m, or 61.1 cm. On average, the network took 4.58 ms to provide an estimate.

VI. Conclusion

While the preliminary results presented here are not able to compete with state-of-the-art scan matching techniques, we can conclude that the architecture presented is able to provide reasonable odometry estimates from large scale lidar data. While the current results have room for improvement, the per scan pair estimate time of the network is promising. We believe that this work is a step towards leveraging deep learning for efficient use of high fidelity sensor data.

VII. Future Work

Moving forward, the authors have several avenues they would like to pursue as future work. These avenues address the challenges of increasing the accuracy of these results, and realizing this work on actual robots in the field.

The first avenue involves exploring additional network structures/parameters to increase prediction accuracy. One possibility is to discretize the predictions and formulate the network as a classifier. To address the loss of precision, classification ranges could be intelligently chosen based on the spread of values represented in the data set. In this case, a separate network for each degree of freedom could be trained, similar to Konda and Memisevic [6]. Additionally, exploring the addition of LSTMs into the network architecture may allow for a more intelligent prediction. LSTMs may allow for the network to learn information about the robot's dynamics, resulting in a more accurate odometry prediction when given a sequence of lidar scans.

Next, the authors are interested in creating a method that allows us to do 3D, spatial feature extraction directly on point cloud data. This directly addresses the issues we encountered with creating voxel grids of the entire space spanned by the VLP-16 while allowing for the true 3D spatial relationships to be leveraged rather than projected into 2D. In addressing this issue, the authors propose implementing 3D convolutional and pooling layers using nearest-neighbors and clustering techniques.

Lastly, we wish to create a more robust, and realistic, data set. The authors propose attaching the VLP-16 to DJI Matrice quadcopters [4]. This would allow for the collection of data in a full 6 degrees of freedom. Additionally, data can be collected in a wider variety of environments and features can be viewed from a larger variety of angles.

VIII. Acknowledgments

This research has been funded in part by the following: NASA grant NNX14AI10G, DWFritz Automation, and the Oregon Metals Initiative.

References

[1] Jason Campbell, Rahul Sukthankar, and Illah Nourbakhsh. Techniques for evaluating optical flow for visual odometry in extreme terrain. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2004.

[2] Yang Chen and Gerard Medioni. Object modeling by registration of multiple range images. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, April 1991.

[3] Jia Deng, Wei Dong, Richard Socher, Li jia Li, Kai Li, and Li Fei-fei. Imagenet: A large-scale hierarchical image database. In *In CVPR*, 2009.

[4] DJI. Matrice 100, 2016. URL http://www.dji.com/product/matrice100.

[5] Jens Garstka and Gabriele Peters. Fast and robust keypoint detection in unstructured 3-d point clouds. In *Informatics in Control, Automation and Robotics (ICINCO), 2015 12th International Conference on*, volume 2, pages 131–140. IEEE, 2015.

[6] Kishore Konda and Roland Memisevic. Learning visual odometry with a convolutional network. In *International Conference on Computer Vision Theory and Applications)*, 2015.

[7] Alex Krizhevsky. Learning Multiple Layers of Features from Tiny Images. Master's thesis, University of Toronto, Department of Computer Science, 2009.

[8] Alex Krizhevsky. Convolutional deep belief networks on cifar-10. 2010. Unpublished manuscript.

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[10] Y. LeCun, Fu Jie Huang, and L. Bottou. Learning methods for generic object recognition with invariance to pose and lighting. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 2, pages II–97–104 Vol.2, June 2004. doi: 10.1109/CVPR.2004.1315150.

[11] Feng Lu. *Shape Registration using Optimization for Mobile Robot Navigation*. PhD thesis, University of Toronto, 1995.

[12] Daniel Maturana and Sebastian Scherer. 3d convolutional neural networks for landing zone detection from lidar. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 3471–3478. IEEE, 2015.

[13] David Nister, Oleg Naroditsky, and James Bergen. Visual

odometry. In *Computer Vision and Pattern Recognition (CVPR)*, 2004.

[14] Bryan C. Russell, Antonio Torralba, Kevin P. Murphy, and William T. Freeman. Labelme: A database and web-based tool for image annotation. *Int. J. Comput. Vision*, 77(1-3):157–173, May 2008. ISSN 0920-5691. doi: 10.1007/s11263-007-0090-8.

[15] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[16] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[17] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. 2014.

[18] G. Weiss and E. Puttkamer. A map based on laserscans without geometric interpretation. *Intelligent Autonomous Systems*, pages 403–407, 1995.

[19] Ji Zhang and Sanjiv Singh. Loam: Lidar odometry and mapping in real-time. In *Robotics: Science and Systems Conference (RSS)*, 2014.

[20] Ji Zhang and Sanjiv Singh. Low-drift and real-time lidar odometry and mapping. *Autonomous Robots*, 10.1007, 2016.