Point Cloud Compression for 3D LiDAR Sensor using Recurrent Neural Network with Residual Blocks

Conference Paper · May 2019

DOI: 10.1109/ICRA.2019.8794264

4 authors, including:

Chenxi Tu
Nagoya University
6 PUBLICATIONS **175** CITATIONS
SEE PROFILE

Alexander Carballo
Nagoya University
51 PUBLICATIONS **1,263** CITATIONS
SEE PROFILE

Kazuya Takeda
Nagoya University
475 PUBLICATIONS **7,671** CITATIONS
SEE PROFILE

Point Cloud Compression for 3D LiDAR Sensor using Recurrent Neural Network with Residual Blocks

Chenxi Tu[1] Eijiro Takeuchi[1] Alexander Carballo[2] Kazuya Takeda[2]

***Abstract*— The use of 3D LiDAR, which has proven its capabilities in autonomous driving systems, is now expanding into many other fields. The sharing and transmission of point cloud data from 3D LiDAR sensors has broad application prospects in robotics. However, due to the sparseness and disorderly nature of this data, it is difficult to compress it directly into a very low volume. A potential solution is utilizing raw LiDAR data. We can rearrange the raw data from each frame losslessly in a 2D matrix, making the data compact and orderly. Due to the special structure of 3D LiDAR data, the texture of the 2D matrix is irregular, in contrast to 2D matrices of camera images. In order to compress this raw, 2D formatted LiDAR data efficiently, in this paper we propose a method which uses a recurrent neural network and residual blocks to progressively compress one frame's information from 3D LiDAR. Compared to our previous image compression based method and generic octree point cloud compression method, the proposed approach needs much less volume while giving the same decompression accuracy. Potential application scenarios for point cloud compression are also considered in this paper. We describe how decompressed point cloud data can be used with SLAM (simultaneous localization and mapping) as well as for localization using a given map, illustrating potential uses of the proposed method in real robotics applications.**

I. INTRODUCTION

With the development of autonomous driving systems, 3D LiDAR sensors have demonstrated their usefulness for the digital perception of real world environments. Over the years, 3D LiDAR systems have become cheaper, and according to manufacturers such as Osram and Draper, mass production unit prices could eventually be under $50 [1], [2]. As a result, we believe that 3D LiDAR systems will soon be widely used in the field of robotics.

Sharing and storing point cloud from LiDAR is meaningful for many real robotics applications, for example, remote control. Many robotic applications employ remote control systems based on camera sensors, such as [3]. A 3D LiDAR scanner generating point cloud of local environment could be a good alternative or supplement to video based control, like the case in [4]. Another possible application is multi robot synergy. Researchers have used laser scanners for mapping tasks involving multiple robots [5], in which the data collected by the robot-mounted sensors must be transmitted using limited bandwidth [6]. In addition, Vehicle-to-vehicle (V2V) networks for autonomous driving [7] are another promising application scenario which also need to share data from LiDAR sensors.

Point cloud data is sparse and disorderly, which is difficult to process and data compression is no exception. Current generic point cloud compression methods usually convert 3D point cloud data into a 2D format (height maps, panorama, etc.) or use a tree structure to represent the point cloud.

Because it is hard to compress point cloud to very low volume directly by these generic methods, other approaches targeting the raw packet data from a 3D LiDAR have been developed. Raw packet data information can be arranged into a 2D matrix naturally and losslessly to convert into point cloud using calibrations. The challenge is that, on account of the structure of the LiDAR, 2D matrices of raw LiDAR packet data are irregular, i.e., the data is not directly spatially correlated and cannot be understood intuitively. Some of these raw LiDAR packet data compression methods sacrifice accuracy and reduce the number of bits used by each "pixel" [8], or use existing image compression methods to compress the 2D matrix [9]. Considering the irregularity of image-like raw packet data, these compression methods are not suitable.

Deep learning has demonstrated its ability to interpret the textures or features of data in recognition tasks, and has also been used in data compression tasks. Toderici [10] proposed a deep learning based image compression method which slightly outperforms conventional image compression methods.

In this paper, we focus on static point cloud compression, which is a component of streaming point cloud compression, and propose a compression method which uses a recurrent neural network with residual block structures to compress one frame of point cloud data progressively.

The main contributions of this paper are as follows:

- A recurrent neural network to compress a point cloud from a 3D LiDAR, which is, as far as we know, the first use of deep learning for point cloud compression.
- A new decoder with a residual block structure, which improves point cloud decompression accuracy.
- Considering application scenarios, a evaluation of how decompressed point cloud data works in SLAM and localization tasks.

We also compare the performance of our proposed method with other raw packet data compression method quantitatively, as well as a widely used octree compression method. Results show that proposed method outperforms other approaches.

[1] C. Tu and E. Takeuchi are with Department of Intelligent System, Graduate School of Informatics, Nagoya University, Nagoya 464-8603 Japan
tu.chenxi@g.sp.m.is.nagoya-u.ac.jp
takeuchi@coi.nagoya-u.ac.jp

[2] A. Carballo and K. Takeda are with Institutes of Innovation for Future Society, Nagoya University, Nagoya 464-8603 Japan.
alexander@g.sp.m.is.nagoya-u.ac.jp
kazuya.takeda@nagoya-u.jp

II. RELATED WORK

The key goal of data compression is reducing redundancy, and in the case of point clouds we are concerned about reducing spatial redundancy. Most point cloud compression methods follow one of the following strategies; either converting a 3D point cloud into a 2D format and then using image compression method, or using a tree structure to store the point cloud.

Height map based methods, which were inspired by 3D mesh compression [11], have been studied extensively in the field of computer graphics. Pauly and Gross [12] were the first to propose using height maps to compress point cloud data. Other approaches, like those of Hubo et al. [13] and Ochotta and Saupe [14], [15] were then developed in response. Golla and Klein [16] used pre-processing and JPEG, leading to real-time compression. Rather than converting one point cloud into many images like height map based methods, some methods use only one 2D matrix to represent all of a point cloud's information. Houshiar and Nüchter [17] proposed mapping point clouds to panorama images using equirectangular projection, then conventional image compression methods are used to reduce redundancy. Kohira and Masuda [18] mapped point cloud data onto 2D pixels using GPS time and the parameters of the laser scanner. Directly converting a 3D point cloud into a 2D image will inevitably result in information loss, but some methods use the raw packet data from the LiDAR, which can roughly be seen as a kind of polar coordinate presentation of a point cloud. This packet data can then be rearranged into a 2D matrix naturally and losslessly. Yin and Berger [8] used such a representation, sacrificing the accuracy of the distance and yaw angle information in order to compress the volume. A method we have proposed in previous studies [9], [19] converts raw packet data into a 2D matrix, and then uses an existing image compression method to process the data.

Taking advantage of a tree structure is another popular strategy for compressing point clouds directly. Widely used tree structures include kd-trees [20] and octrees [21], [22]. Since octree compression is supported by the Point Cloud Library (PCL), octree compression may be the most popular generic point cloud compression method.

In our previous work we have already shown that using raw packet data from LiDAR is a highly cost-effective approach for compression. An issue when using this approach, however, is that the 2D matrix, which comes from raw LiDAR packet data without any calibration, does not have any direct spatial correlation. Having spatial correlation is a important hypothesis for using image compression methods. Therefore we need to find a more suitable approach for compressing these "irregular 2D matrices" constructed from raw packet data.

Deep learning has already achieved state-of-the-art results in many fields, and data compression may be a task which deep leaning is good at. Many researchers have investigated image compression using deep learning, which can be thought of as the compression of a 2D matrix. Auto-encoders have also been used to reduce the dimensionality of images [23] or to convert images to a smaller volume format so they can be used in other applications [24], [25]. In a subsequent study, auto-encoders were used directly in a compression task [26]. Recently, Toderici [10] proposed a recurrent neural network based approach which can compress arbitrary resolution images, slightly outperforming conventional image compression methods.

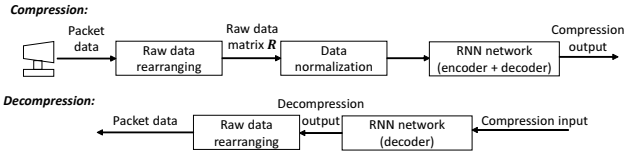

Fig. 1: Flowchart of proposed compression method.

Considering the irregularity of raw LiDAR packet data and the strong capability of deep learning to understand the features/textures of 2D data, in this study we use a recurrent neural network based approach to compress 3D LiDAR point clouds.

III. METHOD

3D LiDAR sensors perceive the surrounding environment by emitting pulsed laser beams and then measuring the reflected pulses. The raw data from the LiDAR scanner, known as packet data, are converted into point clouds, which are basically 3D arrays of laser reflection points, by calibration. In this paper we utilize packet data to format point cloud information, and a recurrent neural network is used to compress the data.

Our overall approach can be divided into three parts, as shown in Fig. 1. First, we convert one frame of raw packet data, into a 2D matrix $\boldsymbol{R}$ plus a few extra bits. Second, we conduct pre-processing to normalize the data before sending it along the network. Third, a recurrent neural network composed of an encoder, a binarizer and a decoder is used for data compression. In contrast to a related study [10], we propose a new type of decoder network which uses residual blocks to improve decompression performance.

A. Raw data rearranging

Raw packet data from LiDAR represent each reflection point using a distance value, a rotation (yaw) angle and a laser ID. 3D LiDAR systems from different manufacturers differ slightly, so here we use a Velodyne system as an example. Laser ID here contains pitch angle information, because it has a fixed one-to-one mapping relationship between pitch angle.

Following the same process proposed in [9], 3D information from LiDAR can be stored into a 2D matrix $\boldsymbol{R}$ like an image. For each pixel $R_{i,j}$ in $\boldsymbol{R}$, laser ID (pitch angle) information can be naturally represented by row numbers i, while information about rotation angles, which corresponds to columns j, can be coded into a few bits of data because the differences between adjacent rotation angles are generally the same. So by using run length coding, all of

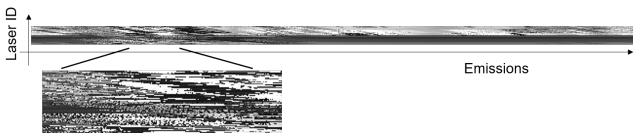

Fig. 2: Raw packet data transformed into an image-like format. A pixel's grayscale value from black to white represents a distance value from 0 to 130 meters.

the rotation angle information could be captured by only a few bits. Using this method, 3D LiDAR point cloud $\boldsymbol{P}$ can be converted losslessly into 2D matrix $\boldsymbol{R}$ with a few additional bits losslessly. As shown in Fig. 2, without calibration, $\boldsymbol{R}$ is irregular and do not have spatial correlation like depth image. However, by doing this, data is no more sparse and disorder, which helps following processing. In addition, other information, such as intensity or color, can also be easily included when using this format.

B. Data normalization

Before sending data into network, normalizing the data is a useful and important step to improve the network performance. There are many different methods for data normalization. For images, normalizing color image $\boldsymbol{I}$ into $\boldsymbol{I}/255 - 0.5$ allows the data range from 0 to 1 and be zero-centered, which is a good choice under the hypothesis that all colors appear averagely. However packet data in 2D format is very different because the long detection range and high accuracy of LiDAR scanners means each pixel's value will have a much larger range of possible values than a color image. For example, most LiDAR from Velodyne using 2 Bytes to store one point's distance information, which means that can range from 0 to 65535. At the same time, the distribution of distance value is unique and highly depends on the scenario. Generally speaking, today's 3D LiDAR scanners can detect objects over 100 meters away, and even up to 300 meters away, although in real world applications most of reflection points obtained are 50 meters away or closer. To normalize data well, before training we randomly chose packet data from 100 frames, calculated the mean μ and created a histogram of the distance values. By setting threshold θ so that 95% of the distance values fall under this value, we could normalize each packet data matrix $\boldsymbol{R}$ by calculating $(\boldsymbol{R} - \mu)/\theta$.

C. Network structure

Toderici et al. [10] proposed a recurrent neural network consisting of an encoder E, a binarizer B and a decoder D which could be used to compress 2D images efficiently. Inspired by this work, our basic network uses a similar structure, although the input and output in our application contain only one channel each, not the three channels used for compressing images. However, when using such a network it is difficult to obtain highly accurate decompression results, even if we reduce the compression ratio. To obtain more accurate decompression results, we propose using a

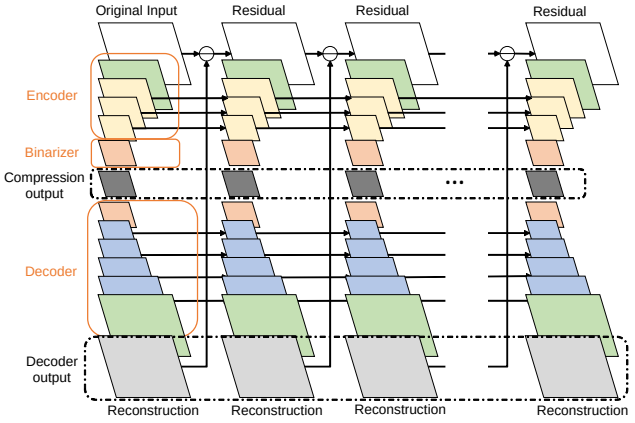

Fig. 3: Architecture of proposed network

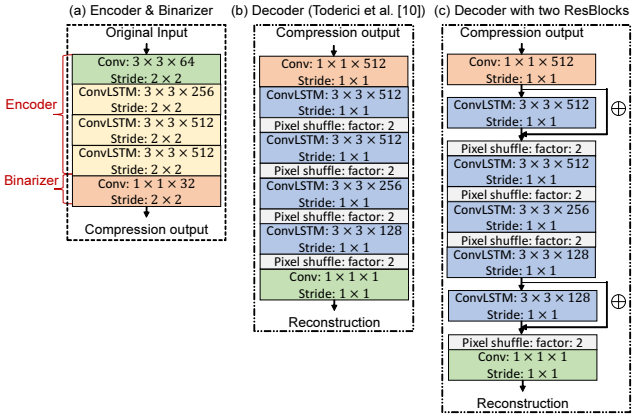

Fig. 4: Network detail.

decoder with residual blocks, which allows us to obtain better decompression accuracy while maintaining the same compression ratio.

1) Overview:

As shown in Fig. 3, compression network is composed of an encoding network E, a binarizer B and a decoding network D, where D and E contain recurrent network components. Original 2D matrix $\boldsymbol{R}$ is first sent into encoder, and then binarizer transforms encoder's output into a binary file which can be stored as compression output and transmitted to decoder. Decoder tries to reconstruct the original input using received binary file. This process represents one iteration. The calculated residual between original input data and reconstruction, become the input of next iteration. Keeping repeating it, after one more iteration, we obtain more bits of compression output while decompressed data can be more accurate.

We can compactly represent a single iteration of our networks as follows:

$$b_t = B(E_t(r_{t-1})), \quad \hat{x}_t = D_t(b_t) + \hat{x}_{t-1}, \tag{1}$$

$$r_0 = x, \quad \hat{x}_0 = 0 \tag{2}$$

Subscript t here represents iteration t, for example, D_t and E_t represent the decoder and encoder with their states at

iteration t respectively. x represents original input, $\hat{x}_t$ means progressive reconstruction of original input, r_t is the residual between x and the reconstruction $\hat{x}_t$, $b_t \in \{-1, 1\}^m$ is m bits binarized stream from B. After k iterations, compression output totally need $m \times k$ bits.

During training, calculating average residuals generated at each iteration as loss for the network:

$$\frac{1}{t}\sum_{t}|r_t| \tag{3}$$

Input x is a 32×32×1 array. During training, we randomly sampled a patch this size from each training data, and during compression we make sure the height and width of the input were divisible by 32 through padding. After encoder and binarizer, input is reduced to a 2×2×32 binary representation per iteration, which leads to add $1/8$ bit per point (bpp) for compression output after each iteration. By using more iterations, bits per point, in other words volume needed by compression output, will increase linearly. At the same time, decompression can be more accurate.

During compression, data passes through the encoder, binarizer and decoder, but during decompression only the decoder is needed.

2) Encoder:

Fig. 4(a) shows the detail of encoder network which consists of one convolutional layer and three convolutional LSTM layers. Different with normal LSTM, here we use use convolution operator in place of matrix multiplication.

Let x_t, h_t, and c_t denote the input, hidden states and cell at iteration t. Given the current input x_t with previous hidden state h_{t-1} and cell state c_{t-1}, new cell state c_t and hidden state h_t can be computed as:

$$\begin{aligned}
i_t &= \sigma(W_{xi} * x_t + W_{hi} * h_{t-1} + b_i) \\
f_t &= \sigma(W_{xf} * x_t + W_{hf} * h_{t-1} + b_f) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \tanh(W_{xc} * x_t + W_{hc} * h_{t-1} + b_c) \\
o_t &= \sigma(W_{xo} * x_t + W_{ho} * h_{t-1} + b_o) \\
h_t &= o_t \circ \tanh(c_t)
\end{aligned} \tag{4}$$

where '$*$' denotes the convolution operator and '$\circ$' denotes the Hadamard product. W is the weight of convolution and b is the bias. The activation function σ is the sigmoid function $\sigma(x) = 1/(1 + \exp(-x))$.

3) Binarizer:

To output binary compressed data, a binarizer follows encoder. Binarizer consists of one convolution layer, a $tanh$ function and a $sign$ function. Convolution layer fix output size into a 2×2×32 array, then a $tanh$ function with $sign$ function binarize the value into bits. After it, each 32×32×1 input array is reduced to a 2×2×32 binarized representation per iteration, which results in each iteration representing $1/8$ bit per point (bpp).

4) Basic decoder:

In decoder, we need to reconstruct original 32×32×1 input from 2×2×32 binary compressed data with previous state. Basic decoder's structure follows Toderici et al. [10], like Fig. 4(b) shows. At the beginning of decoder, a convolution layer expands the input channel. Next, a convolutional LSTM layer and a pixel shuffle layer [27] are used to double the size of the input sensor, which is similar to a super resolution task. After four pixel shuffle layers, we can reconstruct the same size output as original input.

5) Decoder with residual block:

When using the basic decoder network, it is difficult to achieve highly accurate decompression, even if we increase the number of iterations. Many issues may cause this, but we think one of the most important is that the decoder is not "flexible" enough to handle different textures or scenarios. If this is the case, adding more iterations will not really improve the quality of the decompression results.

To address this problem, we propose utilizing a residual block structure within the decoder, which we think will allow processing of a wider range of scenarios, leading to more accurate decompression.

A residual block is the fundamental building block of residual networks, and was first proposed by He [28] in 2015 for object recognition. On account of its performance, residual block structures were then widely applied to other tasks. One example is super resolution [29], [30], which is used to enhance the resolution of images, which is similar to the task of our decoder.

Researchers have explained the advantages of using residual blocks from various viewpoints [31], [32], [33]. According to Veit et al. [33], residual block is a kind of ensemble algorithms. We consider that using residual block should make decoder more flexible.

As shown in Fig. 4(c), a residual block structure can be inserted before each pixel shuffling operation. For other tasks, like super-resolution or classification, some researchers [30] prefer to stack many residual blocks in order to build a deep network. Since data compression is often a time-sensitive task and our network is a recurrent neural network, adding one more residual block means adding it during each iteration. As a result, rather then stacking many residual blocks, we only use a few residual blocks to improve decompression performance efficiently. Using one residual block is the most economical approach which involves adding one skip route on the first convolutional LSTM layer and uses almost same calculation cost with basic decoder.

IV. EVALUATION

Whether a lossy compression algorithm performs well depends not only on its compression rate and information loss, but also on the application in which the decompression output will be used.

In section IV-A, we quantitatively compare the proposed method, using various numbers of residual blocks, with two existing point cloud compression methods.

In section IV-B, we discuss the effects of information loss when using the proposed method in robotics applications.

To train network, we use 33134 frames as training data, consisting of 1 Hz sampling of driving data from 11 areas of

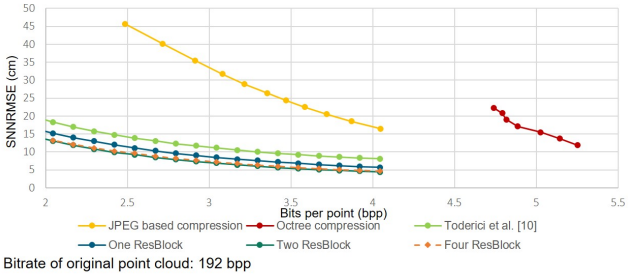

Fig. 5: Comparison of various point cloud compression methods

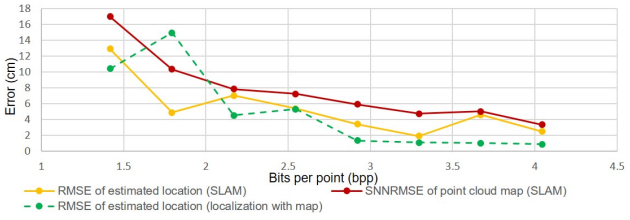

Fig. 6: Evaluating decompression result by applications.

Japan. All the point cloud data came from Velodyne HDL-32E sensor and includes various situations like urban road, country road, forest park, etc. Test data comes from a 32 minutes driving data in Akagi, which is not one of the areas included in the training data. For direct evaluation in subsection IV-A, we extract 32 frames by 1/60 Hz sampling as test data. For evaluation using applications in subsection IV-B, we take first 1 minute (600 frames) continuous data as test data. All data are from TierIV open dataset `https://rosbag.tier4.jp/`.

A. Comparative evaluation

To evaluate compression performance quantitatively, appropriate criteria is needed. This paper uses bits per point (bpp) to measure the volume after compression and Symmetric Nearest Neighbor Root Mean Squared Error (SNNRMSE) to measure loss after decompression, methods generally used to evaluate the quality of point cloud compression [16].

Given two point cloud frames $\boldsymbol{P}, \boldsymbol{Q}$, for each point $p \in \boldsymbol{P}$ find the closest (Euclidean distance) point $q \in \boldsymbol{Q}$, $q = \mathrm{NN}(p, \boldsymbol{Q})$. Then calculate the error as:

$$\mathrm{MSE}_{\mathrm{NN}}(\boldsymbol{P}, \boldsymbol{Q}) = \sum_{p \epsilon \boldsymbol{P}} (p - q)^2 / |\boldsymbol{P}| \tag{5}$$

where $|\boldsymbol{P}|$ represents the number of points in $\boldsymbol{P}$:

$$\mathrm{RMSE}_{\mathrm{NN}}(\boldsymbol{P}, \boldsymbol{Q}) = \sqrt{\mathrm{MSE}_{\mathrm{NN}}(\boldsymbol{P}, \boldsymbol{Q})} \tag{6}$$

By considering both $\mathrm{RMSE}_{\mathrm{NN}}(\boldsymbol{P}, \boldsymbol{Q})$ and $\mathrm{RMSE}_{\mathrm{NN}}(\boldsymbol{Q}, \boldsymbol{P})$, $\mathrm{SNNRMSE}(\boldsymbol{P}, \boldsymbol{Q})$ can be calculated:

$$\begin{aligned} &\mathrm{SNNRMSE}(\boldsymbol{P}, \boldsymbol{Q}) \\ &= \sqrt{0.5\mathrm{MSE}_{\mathrm{NN}}(\boldsymbol{P}, \boldsymbol{Q}) + 0.5\mathrm{MSE}_{\mathrm{NN}}(\boldsymbol{Q}, \boldsymbol{P})} \end{aligned} \tag{7}$$

Fig. 5 shows bpp vs. SNNRMSE for proposed method using various number of residual blocks, JPEG image compression based approach [9] and the generally used octree compression. We tune bpp vs. SNNRMSE by setting iteration of RNN from 16 to 32. Compressing one frame averagely costs from 0.8s to 1.2s using a GeForce GTX 1080 GPU. From Fig. 5, we can see proposed RNN based method outperforms other two methods.

Fig. 7 shows visualizations of decompressed point cloud from various compression approaches. With close SNNRMSE, JPEG based method performs worst visually. Compared with original point cloud, decompression from octree compression can be regarded as a "low resolution" version because many points are discarded or overlapped. Decompression from our RNN based method keeps same "resolution" while brings some errors.

When comparing the proposed method using various numbers of residual blocks with our basic network method, we can see that decompression error is reduced noticeably even when just one residual block is inserted, while only slightly increasing the calculation cost. Increasing the number of residual blocks improves decompression accuracy, but the degree of improvement steadily decreases.

It is interesting to observe that using four residual blocks results in almost the same accuracy as using two residual blocks. Generally speaking, using more residual blocks should increase performance, however in our study the training data was not exactly the same each time because we used $32 \times 32 \times 1$ patches randomly sampled from each 2D packet data matrix, and we also do not fix random seed for every training. Both of these issues would have slightly affected the performance of the compression network. Considering the fluctuation caused by these two issues, using four residual blocks will not reliably outperform using two residual blocks.

As mentioned earlier, compression is a time-sensitive task and adding more residual blocks will increase calculation cost linearly. Therefore, using one or two residual blocks is the best choice in terms of cost and performance. By using RNN based method with one residual block, we can compress point cloud from HDL-32 LiDAR sensor to 1 Mbps with 15 cm SNNRMSE or 2Mbps with 6 cm SNNRMSE, which can be easily handled by current wireless networks.

B. Evaluation by application

As mentioned before, the evaluation of any lossy compression method should take into consideration possible application scenarios. In this section, we use two applications: SLAM and localization with a provided map to evaluate the availability of our method.

We performed these two applications on the original, uncompressed point cloud and on the decompressed point cloud after using various compression rates, respectively. Both SLAM and localization with a provided map are based on the normal distribution transform (NDT) [34] and we used the implementations provided by the Autoware open source autonomous driving platform [35].

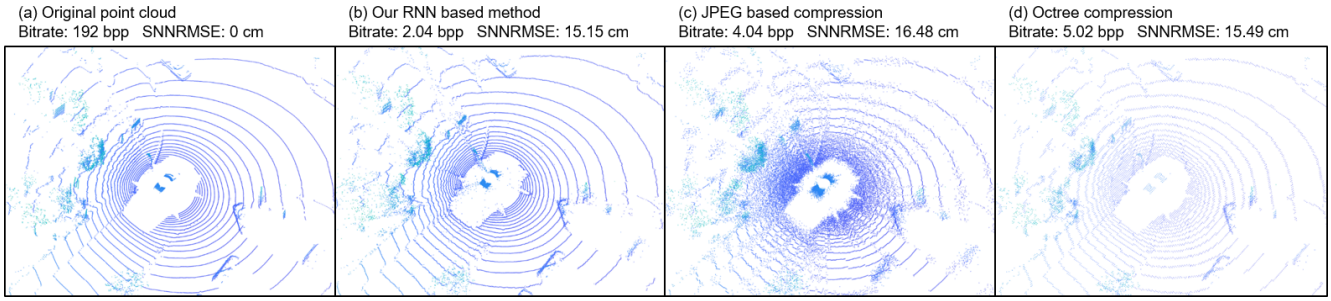

Fig. 7: Example of original point cloud and decompressed point cloud from various approaches (colored by height).

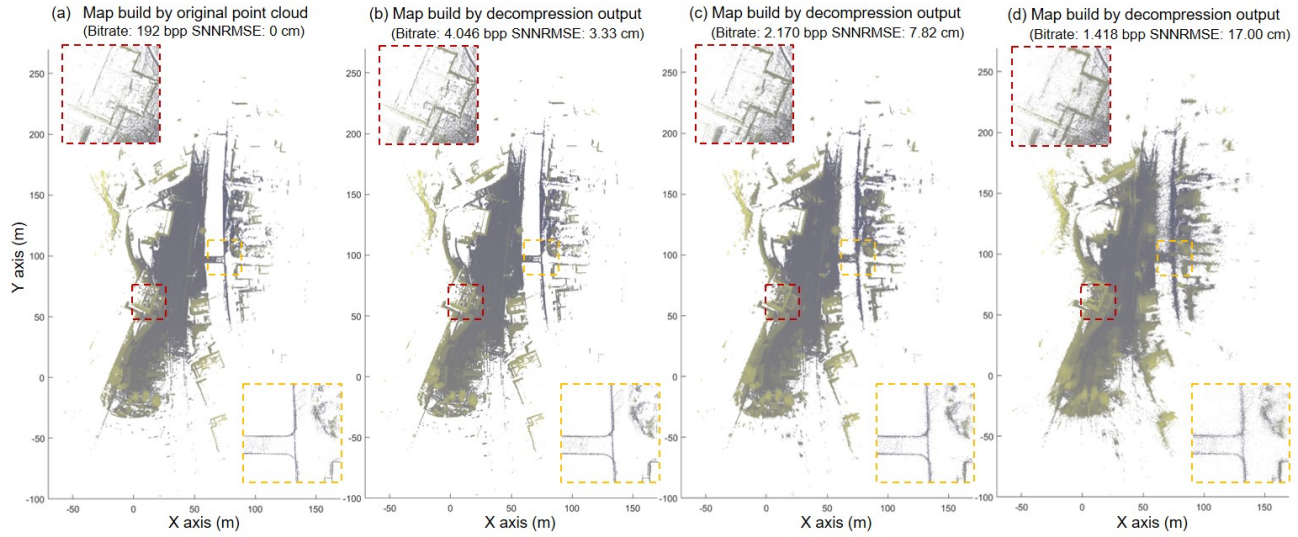

Fig. 8: Visualizing point cloud maps built by SLAM using original point cloud and decompressed point cloud at various compression rate (colored by height).

For both two applications, we set the results using the original point cloud as our ground truth. The SNNRMSE of the point cloud map and RMSE of localization (for all frames) were then calculated to evaluate error.

As show in Fig. 6, on the whole, by increasing the compression rate, error of two application also increases. However, no matter in SLAM or using a provided map, we can still perform localization tasks with very little error, even when using sharply compressed data. And the SNNRMSE of the map built using SLAM is much smaller than the SNNRMSE of the decompressed point cloud at the same bpp.

Comparing the map built using the original point cloud which needed 192 bpp, shown in Fig. 8(a), and the map in Fig. 8(b), which was built from a decompressed point cloud at 4.046 bpp, we can see that they are almost the same. Even as the compression rate increases, the level of quality remains high, as shown in Fig. 8(c), which was constructed from data compressed at a 1/88 compression rate. However, as shown in Fig. 8(d), the errors eventually become obvious.

The results of these two experiments demonstrate the potential shown by the proposed method for its use in real applications.

V. CONCLUSION

In this paper we proposed the use of a recurrent neural network to compress point cloud data from 3D LiDAR. By taking advantage of the feature extraction and context analysis capability of RNN with convolutional layers, the proposed method can tune the compression rate with decompression error and outperform previous image compression based approach and octree compression approach. By adding a residual block structure to the decoder, we were able to further improve decompression performance at almost the same calculation cost, without adding volume. Considering application scenarios such as SLAM and localization using a provided map, this paper shows the potential uses of the proposed method in real robotics applications.

Our next goal is to extent our method to the compression of streaming point cloud data, which has wider application prospects. Using generative networks such as PredNet [36], for example, is a possible strategy for dealing with temporal redundancy.

REFERENCES

[1] H. Sean. (2017, Jul.) R&D Firm Draper is Building a $50 Chip-Based Lidar. [Online]. Available: https://www.spar3d.com/news/lidar/rd-firm-draper-building-50-chip-based-lidar/

[2] H. Sean. (2016, Nov.) Osram Teases $50 Automotive LiDAR. [Online]. Available: https://www.spar3d.com/news/lidar/osram-teases-50-automotive-lidar/

[3] J.-W. Kim, B.-D. Choi, S.-H. Park, K.-K. Kim, and S.-J. Ko, "Remote control system using real-time mpeg-4 streaming technology for mobile robot," in *Digest of Technical Papers. International Conference on Consumer Electronics*, 2002, pp. 200–201.

[4] K. Nagatani, S. Kiribayashi, Y. Okada, K. Otake, K. Yoshida, S. Tadokoro, T. Nishimura, T. Yoshida, E. Koyanagi, M. Fukushima *et al.*, "Emergency response to the nuclear accident at the fukushima daiichi nuclear power plants using mobile rescue robots," *Journal of Field Robotics*, vol. 30, no. 1, pp. 44–63, 2013.

[5] M. N. Rooker and A. Birk, "Multi-robot exploration under the constraints of wireless networking," *Control Engineering Practice*, vol. 15, no. 4, pp. 435–445, 2007.

[6] S. Thrun, W. Burgard, and D. Fox, "A real-time algorithm for mobile robot mapping with applications to multi-robot and 3d mapping," in *IEEE International Conference on Robotics and Automation (ICRA)*, vol. 1, 2000, pp. 321–328.

[7] J. Harding, G. Powell, R. Yoon, J. Fikentscher, C. Doyle, D. Sade, M. Lukuc, J. Simons, and J. Wang, "Vehicle-to-vehicle communications: Readiness of v2v technology for application," National Highway Traffic Safety Administration (NHTSA), Washington, DC,, Tech. Rep. DOT HS 812 014, 2014.

[8] H. Yin and C. Berger, "Mastering data complexity for autonomous driving with adaptive point clouds for urban environments," in *IEEE Intelligent Vehicles Symposium (IV)*, 2017, pp. 1364–1371.

[9] C. Tu, E. Takeuchi, C. Miyajima, and K. Takeda, "Compressing continuous point cloud data using image compression methods," in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, 2016, pp. 1712–1719.

[10] G. Toderici, D. Vincent, N. Johnston, S. J. Hwang, D. Minnen, J. Shor, and M. Covell, "Full resolution image compression with recurrent neural networks," in *IEEE Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 5435–5443.

[11] J. Peng, C.-S. Kim, and C.-C. J. Kuo, "Technologies for 3D mesh compression: A survey," *Journal of Visual Communication and Image Representation*, vol. 16, no. 6, pp. 688–733, 2005.

[12] M. Pauly and M. Gross, "Spectral processing of point-sampled geometry," in *ACM Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, 2001, pp. 379–386.

[13] E. Hubo, T. Mertens, T. Haber, and P. Bekaert, "Self-similarity based compression of point set surfaces with application to ray tracing," *Computers & Graphics*, vol. 32, no. 2, pp. 221–234, 2008.

[14] T. Ochotta and D. Saupe, "Compression of point-based 3D models by shape-adaptive wavelet coding of multi-height fields," in *Proceedings of the First Eurographics conferenceon Point-Based Graphics*, 2004, pp. 103–112.

[15] T. Ochotta and D. Saupe, "Image-based surface compression," in *Computer graphics forum*. Wiley Online Library, 2008, pp. 1647–1663.

[16] T. Golla and R. Klein, "Real-time point cloud compression," in *IEEE Intelligent Robots and Systems (IROS)*, 2015, pp. 5087–5092.

[17] H. Houshiar and A. Nüchter, "3D point cloud compression using conventional image compression for efficient data transmission," in *IEEE International Conference on Information, Communication and Automation Technologies (ICAT)*, 2015, pp. 1–8.

[18] K. Kohira and H. Masuda, "Point-cloud compression for vehicle-based mobile mapping systems using portable network graphics." *ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences*, vol. 4, 2017.

[19] C. Tu, E. Takeuchi, C. Miyajima, and K. Takeda, "Continuous point cloud data compression using SLAM based prediction," in *IEEE Intelligent Vehicles Symposium (IV)*, 2017, pp. 1744–1751.

[20] E. Hubo, T. Mertens, T. Haber, and P. Bekaert, "The quantized kd-tree: Efficient ray tracing of compressed point clouds," in *Interactive Ray Tracing 2006, IEEE Symposium on*. IEEE, 2006, pp. 105–113.

[21] R. Schnabel and R. Klein, "Octree-based point-cloud compression." *Symposium on Point Based Graphics(SPBG)*, vol. 6, pp. 111–120, 2006.

[22] J. Elseberg, D. Borrmann, and A. Nüchter, "One billion points in the cloud–an octree for efficient processing of 3D laser scans," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 76, pp. 76–88, 2013.

[23] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[24] A. Krizhevsky and G. E. Hinton, "Using very deep autoencoders for content-based image retrieval." in *European Symposium on Artificial Neural (ESANN)*, 2011.

[25] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *ACM Proceedings of the 25th international conference on Machine learning*, 2008, pp. 1096–1103.

[26] K. Gregor, F. Besse, D. J. Rezende, I. Danihelka, and D. Wierstra, "Towards conceptual compression," in *Advances In Neural Information Processing Systems*, 2016, pp. 3549–3557.

[27] W. Shi, J. Caballero, F. Huszár, J. Totz, A. P. Aitken, R. Bishop, D. Rueckert, and Z. Wang, "Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 1874–1883.

[28] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[29] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang *et al.*, "Photo-realistic single image super-resolution using a generative adversarial network," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 105–114.

[30] B. Lim, S. Son, H. Kim, S. Nah, and K. M. Lee, "Enhanced deep residual networks for single image super-resolution," in *IEEE conference on computer vision and pattern recognition (CVPR) workshops*, vol. 1, no. 2, 2017, p. 4.

[31] D. Balduzzi, M. Frean, L. Leary, J. Lewis, K. W.-D. Ma, and B. McWilliams, "The shattered gradients problem: If resnets are the answer, then what is the question?" pp. 342–350, 2017.

[32] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *European Conference on Computer Vision*, 2016, pp. 630–645.

[33] A. Veit, M. J. Wilber, and S. Belongie, "Residual networks behave like ensembles of relatively shallow networks," in *Advances in Neural Information Processing Systems*, 2016, pp. 550–558.

[34] E. Takeuchi and T. Tsubouchi, "A 3-D scan matching using improved 3-D normal distributions transform for mobile robotic mapping," in *IEEE Intelligent Robots and Systems (IROS)*, 2006, pp. 3068–3073.

[35] S. Kato, E. Takeuchi, Y. Ishiguro, Y. Ninomiya, K. Takeda, and T. Hamada, "An open approach to autonomous vehicles," *IEEE Micro*, vol. 35, no. 6, pp. 60–68, 2015.

[36] W. Lotter, G. Kreiman, and D. Cox, "Deep predictive coding networks for video prediction and unsupervised learning," *arXiv preprint arXiv:1605.08104*, 2016.